

Analysis

May 23, 2024

Contents

1	Linear Algebra	27
1.1	L2 Norm	27
1.2	Inner Product Spaces and Gradient Derivative	30
1.2.1	Real inner product spaces	30
1.2.2	Class instances	35
1.2.3	Gradient derivative	38
1.3	Cartesian Products as Vector Spaces	39
1.3.1	Product is a Module	40
1.3.2	Product is a Real Vector Space	40
1.3.3	Product is a Metric Space	41
1.3.4	Product is a Complete Metric Space	52
1.3.5	Product is a Normed Vector Space	52
1.3.6	Product is Finite Dimensional	56
1.4	Finite-Dimensional Inner Product Spaces	58
1.4.1	Interlude: Some properties of real sets	58
1.4.2	Type class of Euclidean spaces	58
1.4.3	Subclass relationships	62
1.4.4	Class instances	63
1.4.5	Locale instances	65
1.5	Elementary Linear Algebra on Euclidean Spaces	67
1.5.1	More interesting properties of the norm	68
1.5.2	Substandard Basis	69
1.5.3	Orthogonality	70
1.5.4	Orthogonality of a transformation	71
1.5.5	Bilinear functions	73
1.5.6	Adjoint	74
1.5.7	Euclidean Spaces as Typeclass	75
1.5.8	Linearity and Bilinearity continued	75
1.5.9	We continue	79
1.5.10	Infinity norm	83
1.5.11	Collinearity	87
1.5.12	Properties of special hyperplanes	90
1.5.13	Orthogonal bases and Gram-Schmidt process	92

1.5.14	Decomposing a vector into parts in orthogonal subspaces	94
1.5.15	Linear functions are (uniformly) continuous on any set	100
1.5.16	Topological properties of linear functions	100
1.6	Affine Sets	101
1.6.1	Affine set and affine hull	103
1.6.2	Affine Dependence	113
1.6.3	Some Properties of Affine Dependent Sets	117
1.6.4	Affine Dimension of a Set	121
1.7	Convex Sets and Functions	130
1.7.1	Convex Sets	130
1.7.2	Explicit expressions for convexity in terms of arbitrary sums	133
1.7.3	Convex Functions on a Set	136
1.7.4	Arithmetic operations on sets preserve convexity	141
1.7.5	Convexity of real functions	150
1.7.6	Some inequalities: Applications of convexity	154
1.7.7	Misc related lemmas	155
1.7.8	Cones	156
1.7.9	Connectedness of convex sets	158
1.7.10	Convex hull	159
1.7.11	Relations among closure notions and corresponding hulls	169
1.7.12	Caratheodory's theorem	169
1.7.13	Some Properties of subset of standard basis	172
1.7.14	Moving and scaling convex hulls	172
1.7.15	Convexity of cone hulls	173
1.8	Conic sets and conic hull	174
1.9	Convex cones and corresponding hulls	178
1.9.1	Radon's theorem	181
1.9.2	Helly's theorem	183
1.9.3	Epigraphs of convex functions	185
1.9.4	A bound within a convex hull	186
1.10	Definition of Finite Cartesian Product Type	188
1.10.1	Finite Cartesian products, with indexing and lambdas	188
1.10.2	Cardinality of vectors	189
1.10.3	Group operations and class instances	191
1.10.4	Basic componentwise operations on vectors	192
1.10.5	Real vector space	194
1.10.6	Topological space	194
1.10.7	Metric space	196
1.10.8	Normed vector space	199
1.10.9	Inner product space	200
1.10.10	Euclidean space	201
1.10.11	A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space	203

1.10.12	Some frequently useful arithmetic lemmas over vectors	205
1.10.13	Matrix operations	207
1.10.14	Inverse matrices (not necessarily square)	212
1.11	Linear Algebra on Finite Cartesian Products	214
1.11.1	Type (a, n) <i>vec</i> and fields as vector spaces	214
1.11.2	Some interesting theorems and interpretations	222
1.11.3	Rank of a matrix	223
1.11.4	Lemmas for working on $real^{1/2/3/4}$	226
1.11.5	The collapse of the general concepts to dimension one	227
1.11.6	Routine results connecting the types $(real, 1)$ <i>vec</i> and <i>real</i>	228
1.11.7	Explicit vector construction from lists	228
1.11.8	lambda skolemization on cartesian products	229
1.11.9	Explicit formulas for low dimensions	231
1.11.10	Orthogonality of a matrix	231
1.11.11	Finding an Orthogonal Matrix	233
1.11.12	Scaling and isometry	235
1.11.13	Induction on matrix row operations	236
1.12	Traces and Determinants of Square Matrices	241
1.12.1	Trace	242
1.12.2	Relation to invertibility	257
1.12.3	Cramer's rule	260
1.12.4	Rotation, reflection, rotoinversion	263
1.13	Operators involving abstract topology	265
1.13.1	General notion of a topology as a value	266
1.13.2	The discrete topology	269
1.13.3	Subspace topology	270
1.13.4	The canonical topology from the underlying type class	274
1.13.5	Basic "localization" results are handy for connectedness.	275
1.13.6	Derived set (set of limit points)	278
1.13.7	Closure with respect to a topological space	281
1.13.8	Frontier with respect to topological space	289
1.13.9	Locally finite collections	294
1.13.10	Continuous maps	297
1.13.11	Open and closed maps (not a priori assumed continuous)	304
1.13.12	Quotient maps	313
1.13.13	Separated Sets	321
1.13.14	Homeomorphisms	324
1.13.15	Relation of homeomorphism between topological spaces	332
1.13.16	Connected topological spaces	333
1.13.17	Compact sets	343
1.13.18	Embedding maps	352
1.13.19	Retraction and section maps	354
1.13.20	Continuity	356

1.13.21	Half-global and completely global cases	357
1.13.22	The topology generated by some (open) subsets	361
1.13.23	Topology bases and sub-bases	363
1.13.24	Continuity via bases/subbases, hence upper and lower semicontinuity	366
1.13.25	Pullback topology	370
1.13.26	Proper maps (not a priori assumed continuous)	372
1.13.27	Perfect maps (proper, continuous and surjective)	377
1.14	F -Sigma and G -Delta sets in a Topological Space	379
1.15	Disjoint sum of arbitrarily many spaces	384
2	Topology	389
2.1	Elementary Topology	389
2.1.1	Topological Basis	390
2.1.2	Countable Basis	392
2.1.3	Polish spaces	400
2.1.4	Limit Points	401
2.1.5	Interior of a Set	406
2.1.6	Closure of a Set	410
2.1.7	Frontier (also known as boundary)	412
2.1.8	Filters and the “eventually true” quantifier	414
2.1.9	Limits	414
2.1.10	Compactness	417
2.1.11	Cartesian products	429
2.1.12	Continuity	432
2.1.13	Homeomorphisms	434
2.1.14	On Linorder Topologies	438
2.1.15	nhdsin and atin	440
2.1.16	Limits in a topological space	442
2.1.17	Pointwise continuity in topological spaces	445
2.1.18	Combining theorems for continuous functions into the reals	445
2.2	Non-Denumerability of the Continuum	448
2.2.1	Abstract	448
2.3	Abstract Topology 2	452
2.3.1	Closure	455
2.3.2	Frontier	456
2.3.3	Compactness	456
2.3.4	Continuity	457
2.3.5	Equality of continuous functions on closure and related results	457
2.3.6	A function constant on a set	458
2.3.7	Continuity relative to a union.	459

2.3.8	Inverse function property for open/closed maps	460
2.3.9	Seperability	462
2.3.10	Closed Maps	463
2.3.11	Open Maps	463
2.3.12	Quotient maps	464
2.3.13	Pasting lemmas for functions, for of casewise definitions	466
2.3.14	Retractions	470
2.3.15	Retractions on a topological space	475
2.3.16	Paths and path-connectedness	478
2.3.17	Connected components	481
2.3.18	Combining theorems for continuous functions into the reals	487
2.3.19	A few cardinality results	488
2.4	Connected Components	493
2.4.1	Connectedness	493
2.4.2	Connected components, considered as a connectedness relation or a set	494
2.4.3	The set of connected components of a set	498
2.4.4	Proving a function is constant on a connected set by proving that a level set is open	502
2.4.5	Preservation of Connectedness	502
2.4.6	Lemmas about components	504
2.4.7	Constancy of a function from a connected set into a finite, disconnected or discrete set	507
2.5	Function Topology	508
2.5.1	The product topology	509
2.5.2	The Alexander subbase theorem	526
2.5.3	Open Pi-sets in the product topology	534
2.5.4	Relationship with connected spaces, paths, etc.	537
2.5.5	Projections from a function topology to a component	542
2.5.6	Limits	543
2.6	The binary product topology	544
2.7	Product Topology	545
2.7.1	Definition	545
2.7.2	Continuity	549
2.7.3	Homeomorphic maps	556
2.8	T1 and Hausdorff spaces	561
2.9	T1 spaces with equivalences to many naturally "nice" properties.	561
2.9.1	Hausdorff Spaces	566
2.10	Lindelöf spaces	578
3	Functional Analysis	585
3.1	A decision procedure for metric spaces	585

4 Elementary Metric Spaces	589
4.1 Open and closed balls	589
4.2 Limit Points	595
4.3 Perfect Metric Spaces	596
4.4 Finite and discrete	597
4.5 Interior	598
4.6 Frontier	598
4.7 Limits	598
4.8 Continuity	601
4.9 Closure and Limit Characterization	603
4.10 Boundedness	605
4.11 Compactness	607
4.12 Banach fixed point theorem	612
4.13 Edelstein fixed point theorem	615
4.14 The diameter of a set	616
4.15 Metric spaces with the Heine-Borel property	620
4.16 Completeness	623
4.17 Cauchy continuity	628
4.18 Finite intersection property	630
4.19 Properties of Balls and Spheres	631
4.20 Distance from a Set	631
4.21 Infimum Distance	632
4.22 Separation between Points and Sets	637
4.23 Uniform Continuity	638
4.24 Continuity on a Compact Domain Implies Uniform Continuity	640
4.25 Theorems relating continuity and uniform continuity to closures	642
4.26 With Abstract Topology (TODO: move and remove dependency?)	647
4.27 Closed Nest	648
4.28 Making a continuous function avoid some value in a neighbourhood	650
4.29 Consequences for Real Numbers	651
4.30 The infimum of the distance between two sets	654
4.31 Diameter Lemma	658
4.32 Elementary Normed Vector Spaces	659
4.32.1 Orthogonal Transformation of Balls	660
4.32.2 Various Lemmas Combining Imports	660
4.32.3 Support	661
4.32.4 Intervals	662
4.32.5 Limit Points	663
4.32.6 Balls and Spheres in Normed Spaces	665
4.32.7 Various Lemmas on Normed Algebras	668
4.32.8 Filters	669
4.32.9 Trivial Limits	669

4.32.10	Limits	670
4.32.11	Limit Point of Filter	672
4.32.12	Boundedness	673
4.32.13	Relations among convergence and absolute convergence for power series	676
4.32.14	Normed spaces with the Heine-Borel property	677
4.32.15	Intersecting chains of compact sets and the Baire property	677
4.32.16	Continuity	681
4.32.17	Arithmetic Preserves Topological Properties	683
4.32.18	Homeomorphisms	691
4.32.19	Discrete	693
4.32.20	Completeness of "Isometry" (up to constant bounds) . .	693
4.32.21	Connected Normed Spaces	695
4.33	Linear Decision Procedure for Normed Spaces	697
5	Vector Analysis	701
5.1	Elementary Topology in Euclidean Space	701
5.1.1	Continuity of the representation WRT an orthogonal basis	701
5.1.2	Balls in Euclidean Space	703
5.1.3	Boxes	707
5.1.4	General Intervals	721
5.1.5	Bounded Projections	724
5.1.6	Structural rules for pointwise continuity	725
5.1.7	Structural rules for setwise continuity	725
5.1.8	Openness of halfspaces.	725
5.1.9	Closure and Interior of halfspaces and hyperplanes . . .	726
5.1.10	Some more convenient intermediate-value theorem for- mulations	728
5.1.11	Limit Component Bounds	729
5.1.12	Class Instances	734
5.1.13	Compact Boxes	736
5.1.14	Componentwise limits and continuity	737
5.1.15	Continuous Extension	739
5.1.16	Separability	741
5.1.17	Diameter	743
5.1.18	Relating linear images to open/closed/interior/closure/con- nected	744
5.1.19	"Isometry" (up to constant bounds) of Injective Linear Map	746
5.1.20	Some properties of a canonical subspace	750
5.1.21	Set Distance	751
5.2	Line Segment	754

5.2.1	Topological Properties of Convex Sets, Metric Spaces and Functions	754
5.2.2	Midpoint	757
5.2.3	Open and closed segments	758
5.2.4	Betweenness	773
5.3	Convex Sets and Functions on (Normed) Euclidean Spaces . . .	777
5.3.1	Topological Properties of Convex Sets and Functions . . .	777
5.3.2	Relative interior of a set	778
5.3.3	Openness and compactness are preserved by convex hull operation	792
5.3.4	Extremal points of a simplex are some vertices	797
5.3.5	Closest point of a convex set is unique, with a continuous projection	801
5.3.6	More convexity generalities	809
5.3.7	Convex set as intersection of halfspaces	810
5.3.8	Convexity of general and special intervals	810
5.3.9	On <i>real</i> , <i>is_interval</i> , <i>convex</i> and <i>connected</i> are all equivalent	812
5.3.10	Another intermediate value theorem formulation	814
5.3.11	A bound within an interval	814
5.3.12	Representation of any interval as a finite convex hull . . .	816
5.3.13	Bounded convex function on open set is continuous . . .	818
5.3.14	Upper bound on a ball implies upper and lower bounds . . .	820
6	Unsorted	823
6.0.1	Shrinking towards the interior of a convex set	824
6.0.2	Some obvious but surprisingly hard simplex lemmas	829
6.0.3	Relative interior of convex set	837
6.0.4	The relative frontier of a set	846
6.0.5	Convexity on direct sums	876
6.0.6	Explicit formulas for interior and relative interior of convex hull	883
6.0.7	Similar results for closure and (relative or absolute) frontier	890
6.0.8	Coplanarity, and collinearity in terms of affine hull . . .	894
6.0.9	Basic lemmas about hyperplanes and halfspaces	904
6.0.10	Use set distance for an easy proof of separation properties . . .	905
6.0.11	Connectedness of the intersection of a chain	907
6.0.12	Proper maps, including projections out of compact sets . . .	910
6.0.13	Closure of conic hulls	911
6.0.14	Trivial fact: convexity equals connectedness for collinear sets	914
6.0.15	Some stepping theorems	924

6.0.16	General case without assuming closure and getting non-strict separation	927
6.0.17	Some results on decomposing convex hulls: intersections, simplicial subdivision	931
6.0.18	Lower-dimensional affine subsets are nowhere dense	943
6.0.19	Parallel slices, etc	945
6.0.20	Paracompactness	949
6.0.21	Closed-graph characterization of continuity	952
6.0.22	The union of two collinear segments is another segment	954
6.0.23	Covering an open set by a countable chain of compact sets	958
6.0.24	Orthogonal complement	960
6.0.25	A non-injective linear function maps into a hyperplane.	962
6.1	Path-Connectedness	964
6.1.1	Paths and Arcs	964
6.1.2	Invariance theorems	965
6.1.3	Basic lemmas about paths	967
6.1.4	Path Images	971
6.1.5	Simple paths with the endpoints removed	973
6.1.6	The operations on paths	974
6.1.7	Some reversed and "if and only if" versions of joining theorems	976
6.1.8	The joining of paths is associative	979
6.1.9	Subpath	981
6.1.10	There is a subpath to the frontier	984
6.1.11	Shift Path to Start at Some Given Point	986
6.1.12	Straight-Line Paths	989
6.1.13	Segments via convex hulls	992
6.1.14	Bounding a point away from a path	994
6.1.15	Path component	995
6.1.16	Path connectedness of a space	996
6.1.17	Lemmas about path-connectedness	1002
6.1.18	Path components	1004
6.1.19	Path components	1005
6.1.20	Paths and path-connectedness	1015
6.1.21	Path components	1016
6.1.22	Sphere is path-connected	1018
6.1.23	Every annulus is a connected set	1025
6.1.24	Relations between components and path components	1026
6.1.25	Existence of unbounded components	1028
6.1.26	The <i>inside</i> and <i>outside</i> of a Set	1030
6.1.27	Condition for an open map's image to contain a ball	1047
6.1.28	Rectangular paths	1052
6.2	Neighbourhood bases and Locally path-connected spaces	1054

6.2.1	Neighbourhood Bases	1054
6.2.2	Locally path-connected spaces	1056
6.2.3	Locally connected spaces	1066
6.2.4	Dimension of a topological space	1075
6.3	Some Uncountable Sets	1080
6.4	Homotopy of Maps	1082
6.4.1	Trivial properties	1083
6.4.2	Homotopy with P is an equivalence relation	1085
6.4.3	Continuity lemmas	1087
6.4.4	Homotopy of paths, maintaining the same endpoints	1093
6.4.5	Group properties for homotopy of paths	1096
6.4.6	Homotopy of loops without requiring preservation of endpoints	1098
6.4.7	Relations between the two variants of homotopy	1099
6.4.8	Homotopy of "nearby" function, paths and loops	1104
6.4.9	Homotopy and subpaths	1105
6.4.10	Simply connected sets	1108
6.4.11	Contractible sets	1111
6.4.12	Starlike sets	1114
6.4.13	Local versions of topological properties in general	1117
6.4.14	An induction principle for connected sets	1121
6.4.15	Basic properties of local compactness	1123
6.4.16	Sura-Bura's results about compact components of sets	1131
6.4.17	Special cases of local connectedness and path connectedness	1135
6.4.18	Relations between components and path components	1142
6.4.19	Components, continuity, openin, closedin	1146
6.4.20	Existence of isometry between subspaces of same dimension	1148
6.4.21	Retracts, in a general sense, preserve (co)homotopic triviality)	1153
6.4.22	Homotopy equivalence	1156
6.4.23	Homotopy equivalence of topological spaces.	1156
6.4.24	Contractible spaces	1159
6.4.25	Misc other results	1171
6.4.26	Some simple positive connection theorems	1172
6.4.27	Self-homeomorphisms shuffling points about	1178
6.4.28	Nullhomotopic mappings	1197
6.5	Euclidean space and n-spheres, as subtopologies of n-dimensional space	1201
6.5.1	Euclidean spaces as abstract topologies	1201
6.5.2	n-dimensional spheres	1205
6.6	Various Forms of Topological Spaces	1211
6.6.1	Connected topological spaces	1211

6.6.2	The notion of "separated between" (complement of "connected between")	1212
6.6.3	Connected components	1217
6.6.4	Monotone maps (in the general topological sense)	1221
6.6.5	Other countability properties	1225
6.6.6	Neighbourhood bases EXTRAS	1231
6.6.7	T_0 spaces and the Kolmogorov quotient	1233
6.6.8	Kolmogorov quotients	1235
6.6.9	Closed diagonals and graphs	1239
6.6.10	KC spaces, those where all compact sets are closed.	1241
6.6.11	Technical results about proper maps, perfect maps, etc	1248
6.6.12	Regular spaces	1255
6.6.13	Locally compact spaces	1269
6.6.14	Special characterizations of classes of functions into and out of \mathbb{R}	1284
6.6.15	Normal spaces	1288
6.6.16	Hereditary topological properties	1295
6.6.17	Limits in a topological space	1296
6.6.18	Quasi-components	1298
6.6.19	Additional quasicomponent and continuum properties like Boundary Bumping	1313
6.6.20	Compactly generated spaces (k-spaces)	1320
6.7	Abstract Metric Spaces	1332
6.7.1	Metric topology	1334
6.7.2	Bounded sets	1337
6.7.3	Subspace of a metric space	1341
6.7.4	Abstract type of metric spaces	1342
6.7.5	The discrete metric	1345
6.7.6	Metrizable spaces	1345
6.7.7	Limits at a point in a topological space	1351
6.7.8	Normal spaces and metric spaces	1351
6.7.9	Topological limit in metric spaces	1352
6.7.10	Cauchy sequences and complete metric spaces	1355
6.7.11	Totally bounded subsets of metric spaces	1370
6.7.12	Compactness in metric spaces	1376
6.7.13	Continuous functions on metric spaces	1387
6.7.14	Completely metrizable spaces	1391
6.7.15	Product metric	1395
6.7.16	More sequential characterizations in a metric space	1405
6.7.17	Three strong notions of continuity for metric spaces	1413
6.7.18	Isometries	1431
6.7.19	"Capped" equivalent bounded metrics and general product metrics	1432
6.8	Infinite sums	1444

6.8.1	Definition and syntax	1444
6.8.2	General properties	1445
6.8.3	Absolute convergence	1484
6.8.4	Extended reals and nats	1488
6.8.5	Real numbers	1491
6.8.6	Complex numbers	1491
6.9	Ordered Euclidean Space	1519
6.10	Arcwise-Connected Sets	1526
6.10.1	The Brouwer reduction theorem	1527
6.10.2	Arcwise Connections	1529
6.10.3	Density of points with dyadic rational coordinates	1529
6.10.4	Accessibility of frontier points	1577
6.11	The Urysohn lemma, its consequences and other advanced material about metric spaces	1581
6.11.1	Urysohn lemma and Tietze's theorem	1581
6.11.2	Random metric space stuff	1596
6.11.3	Hereditarily normal spaces	1597
6.11.4	Completely regular spaces	1600
6.11.5	More generally, the k-ification functor	1610
6.11.6	One-point compactifications and the Alexandroff extension construction	1614
6.11.7	Extending continuous maps "pointwise" in a regular space	1632
6.11.8	Extending Cauchy continuous functions to the closure	1635
6.11.9	Metric space of bounded functions	1647
6.11.10	Metric space of continuous bounded functions	1653
6.11.11	Existence of completion for any metric space M as a subspace of $M \Rightarrow \mathbb{R}$	1657
6.11.12	Contractions	1660
6.11.13	The Baire Category Theorem	1663
6.11.14	Sierpinski-Hausdorff type results about countable closed unions	1669
6.11.15	The Tychonoff embedding	1672
6.11.16	Urysohn and Tietze analogs for completely regular spaces	1674
6.11.17	Size bounds on connected or path-connected spaces	1679
6.11.18	Lavrentiev extension etc	1684
6.11.19	Embedding in products and hence more about completely metrizable spaces	1692
6.11.20	Theorems from Kuratowski	1699
6.11.21	A perfect set in common cases must have at least the cardinality of the continuum	1708
6.11.22	Isolate and discrete	1717
6.11.23	A set of points sparse in another set	1724
6.11.24	Co-sparseness filter	1726
6.12	Operator Norm	1729

6.13	Limits on the Extended Real Number Line	1734
6.13.1	Extended-Real.thy	1740
6.13.2	Extended-Nonnegative-Real.thy	1754
6.13.3	monoset	1754
6.13.4	Relate extended reals and the indicator function	1772
6.14	Radius of Convergence and Summation Tests	1773
6.14.1	Convergence tests for infinite sums	1773
6.14.2	Radius of convergence	1783
6.15	Uniform Limit and Uniform Convergence	1793
6.15.1	Definition	1793
6.15.2	Exchange limits	1794
6.15.3	Uniform limit theorem	1795
6.15.4	Comparison Test	1801
6.15.5	Weierstrass M-Test	1803
6.15.6	Structural introduction rules	1806
6.15.7	Power series and uniform convergence	1813
6.16	Bounded Linear Function	1814
6.16.1	Intro rules for <i>bounded_linear</i>	1814
6.16.2	declaration of derivative/continuous/tendsto introduc- tion rules for bounded linear functions	1815
6.16.3	Type of bounded linear functions	1816
6.16.4	Type class instantiations	1816
6.16.5	On Euclidean Space	1821
6.16.6	concrete bounded linear functions	1826
6.16.7	The strong operator topology on continuous linear op- erators	1830
6.17	Derivative	1832
6.17.1	Derivatives	1832
6.17.2	Derivative with composed bilinear function	1832
6.17.3	Differentiability	1834
6.17.4	Frechet derivative and Jacobian matrix	1836
6.17.5	Differentiability implies continuity	1837
6.17.6	The chain rule	1838
6.17.7	Composition rules stated just for differentiability	1839
6.17.8	Uniqueness of derivative	1839
6.17.9	Derivatives of local minima and maxima are zero	1842
6.17.10	One-dimensional mean value theorem	1844
6.17.11	More general bound theorems	1845
6.17.12	Differentiability of inverse function (most basic form)	1854
6.17.13	Uniformly convergent sequence of derivatives	1859
6.17.14	Differentiation of a series	1866
6.17.15	Derivative as a vector	1868
6.17.16	Field differentiability	1875
6.17.17	Field derivative	1880

6.17.18	Relation between convexity and derivative	1884
6.17.19	Partial derivatives	1885
6.17.20	Differentiable case distinction	1889
6.17.21	The Inverse Function Theorem	1891
6.17.22	Piecewise differentiable functions	1898
6.17.23	The concept of continuously differentiable	1901
6.18	Finite Cartesian Products of Euclidean Spaces	1909
6.18.1	Closures and interiors of halfspaces	1910
6.18.2	Bounds on components etc. relative to operator norm .	1911
6.18.3	Convex Euclidean Space	1918
6.18.4	Arbitrarily good rational approximations	1918
6.18.5	Derivative	1919
6.18.6	Routine results connecting the types (<i>real</i> , <i>1</i>) <i>vec</i> and <i>real</i>	1919
6.19	Complex Analysis Basics	1921
6.19.1	General lemmas	1921
6.19.2	Holomorphic functions	1924
6.19.3	Analyticity on a set	1929
6.19.4	Analyticity at a point	1934
6.19.5	Combining theorems for derivative with “analytic at” hypotheses	1935
6.19.6	Complex differentiation of sequences and series	1935
6.19.7	Taylor on Complex Numbers	1937
6.20	Complex Transcendental Functions	1941
6.20.1	Möbius transformations	1941
6.20.2	The Exponential Function	1942
6.20.3	Euler and de Moivre formulas	1943
6.20.4	Relationships between real and complex trigonometric and hyperbolic functions	1945
6.20.5	More on the Polar Representation of Complex Numbers	1946
6.20.6	Taylor series for complex exponential, sine and cosine .	1955
6.20.7	The argument of a complex number (HOL Light version)	1959
6.20.8	Analytic properties of tangent function	1964
6.20.9	The principal branch of the Complex logarithm	1965
6.20.10	Relation to Real Logarithm	1966
6.20.11	Derivative of Ln away from the branch cut	1967
6.20.12	Quadrant-type results for Ln	1973
6.20.13	More Properties of Ln	1974
6.20.14	Uniform convergence and products	1978
6.20.15	The Argument of a Complex Number	1983
6.20.16	The Unwinding Number and the Ln product Formula .	1987
6.20.17	Characterisation of $Im(Ln z)$ (Wenda Li)	1989
6.20.18	Relation between Ln and Arg2pi, and hence continuity of Arg2pi	1990

6.20.19	Complex Powers	1992
6.20.20	Some Limits involving Logarithms	1999
6.20.21	Relation between Square Root and \exp/\ln , hence its derivative	2003
6.20.22	Complex arctangent	2006
6.20.23	Real arctangent	2011
6.20.24	Bounds on pi using real arctangent	2014
6.20.25	Inverse Sine	2015
6.20.26	Inverse Cosine	2018
6.20.27	Upper and Lower Bounds for Inverse Sine and Cosine	2022
6.20.28	Interrelations between Arcsin and Arccos	2023
6.20.29	Relationship with Arcsin on the Real Numbers	2023
6.20.30	Relationship with Arccos on the Real Numbers	2024
6.20.31	Continuity results for arcsin and arccos	2024
6.20.32	Roots of unity	2025
7	Measure and Integration Theory	2029
7.1	Sigma Algebra	2029
7.1.1	Families of sets	2029
7.1.2	Measure type	2059
7.1.3	The smallest σ -algebra regarding a function	2072
7.2	Measurability Prover	2078
7.2.1	Measurability for (co)inductive predicates	2087
7.3	Measure Spaces	2093
7.3.1	Relate extended reals and the indicator function	2093
7.3.2	Extend binary sets	2094
7.3.3	Properties of a premeasure μ	2094
7.3.4	Properties of <i>emeasure</i>	2102
7.3.5	μ -null sets	2112
7.3.6	The almost everywhere filter (i.e. quantifier)	2114
7.3.7	σ -finite Measures	2120
7.3.8	Measure space induced by distribution of (\rightarrow_M) -functions	2123
7.3.9	Real measure values	2126
7.3.10	Set of measurable sets with finite measure	2130
7.3.11	Measurable sets formed by unions and intersections	2131
7.3.12	Measure spaces with <i>emeasure</i> M (<i>space</i> M) $< \infty$	2138
7.3.13	Counting space	2143
7.3.14	Measure restricted to space	2146
7.3.15	Null measure	2149
7.3.16	Scaling a measure	2150
7.3.17	Complete lattice structure on measures	2151
7.4	Borel Space	2176
7.4.1	Generic Borel spaces	2179

7.4.2	Borel spaces on order topologies	2187
7.4.3	Borel spaces on topological monoids	2194
7.4.4	Borel spaces on Euclidean spaces	2194
7.4.5	Borel measurable operators	2201
7.4.6	Borel space on the extended reals	2206
7.4.7	Borel space on the extended non-negative reals	2210
7.4.8	LIMSEQ is borel measurable	2211
7.5	Lebesgue Integration for Nonnegative Functions	2220
7.5.1	Approximating functions	2221
7.5.2	Simple function	2223
7.5.3	Simple integral	2233
7.5.4	Integral on nonnegative functions	2238
7.5.5	Integral under concrete measures	2259
7.6	Binary Product Measure	2279
7.6.1	Binary products	2280
7.6.2	Binary products of σ -finite emeasure spaces	2287
7.6.3	Fubinis theorem	2292
7.6.4	Products on counting spaces, densities and distributions	2294
7.6.5	Product of Borel spaces	2305
7.7	Finite Product Measure	2306
7.7.1	More about Function restricted by <i>extensional</i>	2307
7.7.2	Finite product spaces	2309
7.7.3	Measurability	2334
7.8	Caratheodory Extension Theorem	2337
7.8.1	Characterizations of Measures	2338
7.8.2	Caratheodory's theorem	2347
7.8.3	Volumes	2348
7.9	Bochner Integration for Vector-Valued Functions	2357
7.9.1	Restricted measure spaces	2405
7.9.2	Measure spaces with an associated density	2406
7.9.3	Distributions	2408
7.9.4	Lebesgue integration on <i>count_space</i>	2410
7.9.5	Point measure	2412
7.9.6	Lebesgue integration on <i>null_measure</i>	2412
7.9.7	Legacy lemmas for the real-valued Lebesgue integral	2412
7.9.8	Product measure	2419
7.10	Complete Measures	2430
7.11	Regularity of Measures	2456
7.12	Lebesgue Measure	2466
7.12.1	Measures defined by monotonous functions	2466
7.12.2	Lebesgue-Borel measure	2473
7.12.3	Borel measurability	2476
7.12.4	Measurability of continuous functions	2479
7.12.5	Affine transformation on the Lebesgue-Borel	2485

7.12.6	Lebesgue measurable sets	2493
7.12.7	Translation preserves Lebesgue measure	2494
7.12.8	A nice lemma for negligibility proofs	2496
7.12.9	F_sigma and G_delta sets.	2501
7.13	Tagged Divisions for Henstock-Kurzweil Integration	2504
7.13.1	Some useful lemmas about intervals	2504
7.13.2	Bounds on intervals where they exist	2505
7.13.3	The notion of a gauge — simply an open set containing the point	2507
7.13.4	Attempt a systematic general set of "offset" results for components	2508
7.13.5	Divisions	2508
7.13.6	Tagged (partial) divisions	2521
7.13.7	Functions closed on boxes: morphisms from boxes to monoids	2527
7.13.8	Special case of additivity we need for the FTC	2537
7.13.9	Fine-ness of a partition w.r.t. a gauge	2538
7.13.10	Some basic combining lemmas	2538
7.13.11	The set we're concerned with must be closed	2539
7.13.12	General bisection principle for intervals; might be useful elsewhere	2539
7.13.13	Cousin's lemma	2544
7.13.14	A technical lemma about "refinement" of division	2545
7.13.15	Division filter	2553
7.14	Henstock-Kurzweil Gauge Integration in Many Dimensions	2554
7.14.1	Content (length, area, volume...) of an interval	2554
7.14.2	Gauge integral	2560
7.14.3	Basic theorems about integrals	2561
7.14.4	Cauchy-type criterion for integrability	2575
7.14.5	Additivity of integral on abutting intervals	2577
7.14.6	A sort of converse, integrability on subintervals	2583
7.14.7	Bounds on the norm of Riemann sums and the integral itself	2587
7.14.8	Similar theorems about relationship among components	2589
7.14.9	Uniform limit of integrable functions is integrable	2593
7.14.10	Negligible sets	2596
7.14.11	Some other trivialities about negligible sets	2604
7.14.12	Finite case of the spike theorem is quite commonly needed	2606
7.14.13	In particular, the boundary of an interval is negligible	2607
7.14.14	Integrability of continuous functions	2608
7.14.15	Specialization of additivity to one dimension	2610
7.14.16	A useful lemma allowing us to factor out the content size	2610
7.14.17	Fundamental theorem of calculus	2611
7.14.18	Taylor series expansion	2616

7.14.19	Only need trivial subintervals if the interval itself is trivial	2618
7.14.20	Integrability on subintervals	2620
7.14.21	Combining adjacent intervals in 1 dimension	2621
7.14.22	Reduce integrability to "local" integrability	2622
7.14.23	Second FTC or existence of antiderivative	2623
7.14.24	Combined fundamental theorem of calculus	2625
7.14.25	General "twiddling" for interval-to-interval function image	2626
7.14.26	Special case of a basic affine transformation	2628
7.14.27	Special case of stretching coordinate axes separately	2632
7.14.28	even more special cases	2633
7.14.29	Stronger form of FCT; quite a tedious proof	2635
7.14.30	Stronger form with finite number of exceptional points	2643
7.14.31	This doesn't directly involve integration, but that gives an easy proof	2649
7.14.32	Generalize a bit to any convex set	2650
7.14.33	Integrating characteristic function of an interval	2653
7.14.34	Integrals on set differences	2659
7.14.35	More lemmas that are useful later	2662
7.14.36	Continuity of the integral (for a 1-dimensional interval)	2664
7.14.37	A straddling criterion for integrability	2668
7.14.38	Adding integrals over several sets	2672
7.14.39	Also tagged divisions	2675
7.14.40	Henstock's lemma	2676
7.14.41	Monotone convergence (bounded interval first)	2681
7.14.42	differentiation under the integral sign	2695
7.14.43	Exchange uniform limit and integral	2700
7.14.44	Integration by parts	2701
7.14.45	Integration by substitution	2703
7.14.46	Compute a double integral using iterated integrals and switching the order of integration	2705
7.14.47	Definite integrals for exponential and power function	2712
8	Kronecker's Theorem with Applications	2721
8.1	Dirichlet's Approximation Theorem	2721
8.2	Kronecker's Approximation Theorem: the One-dimensional Case	2727
8.3	Extension of Kronecker's Theorem to Simultaneous Approxima- tion	2731
8.3.1	Towards Lemma 1	2731
8.3.2	Towards Lemma 2	2734
8.3.3	Towards lemma 3	2737
8.3.4	And finally Kroncker's theorem itself	2743
8.4	Bernstein-Weierstrass and Stone-Weierstrass	2751
8.4.1	Bernstein polynomials	2751

8.4.2	Explicit Bernstein version of the 1D Weierstrass approximation theorem	2752
8.4.3	General Stone-Weierstrass theorem	2755
8.4.4	Polynomial functions	2767
8.4.5	Stone-Weierstrass theorem for polynomial functions	2772
8.4.6	Polynomial functions as paths	2776
8.5	Radon-Nikodým Derivative	2782
8.5.1	Absolutely continuous	2786
8.5.2	Existence of the Radon-Nikodym derivative	2787
8.5.3	Uniqueness of densities	2796
8.5.4	Radon-Nikodym derivative	2802
9	Integrals over a Set	2809
9.1	Notation	2809
9.2	Basic properties	2810
9.3	Complex integrals	2821
9.4	NN Set Integrals	2822
9.5	Scheffé's lemma	2827
9.6	Convergence of integrals over an interval	2831
9.7	Integrable Simple Functions	2835
9.7.1	Totally Ordered Banach Spaces	2845
9.7.2	Auxiliary Lemmas for Set Integrals	2847
9.7.3	Integrability and Measurability of the Diameter	2848
9.7.4	Averaging Theorem	2850
9.8	Homeomorphism Theorems	2855
9.8.1	Homeomorphism of all convex compact sets with nonempty interior	2856
9.8.2	Homeomorphisms between punctured spheres and affine sets	2866
9.8.3	Locally compact sets in an open set	2872
9.8.4	Covering spaces and lifting results for them	2877
9.8.5	Lifting of general functions to covering space	2893
9.8.6	Homeomorphisms of arc images	2904
9.8.7	Equivalence Lebesgue integral on <i>lborel</i> and HK-integral	2912
9.8.8	Absolute integrability (this is the same as Lebesgue integrability)	2924
9.8.9	Applications to Negligibility	2933
9.8.10	Negligibility of image under non-injective linear map	2940
9.8.11	Negligibility of a Lipschitz image of a negligible set	2942
9.8.12	Measurability of countable unions and intersections of various kinds.	2950
9.8.13	Negligibility is a local property	2953
9.8.14	Integral bounds	2953

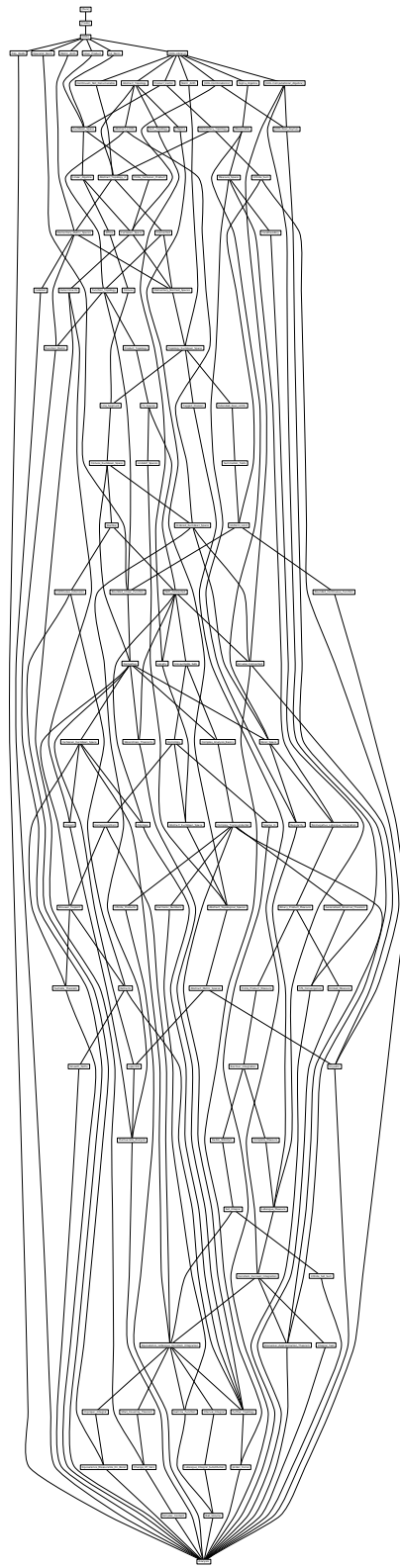
9.8.15	Outer and inner approximation of measurable sets by well-behaved sets.	2966
9.8.16	Transformation of measure by linear maps	2969
9.8.17	Lemmas about absolute integrability	2976
9.8.18	Componentwise	2978
9.8.19	Dominated convergence	2987
9.8.20	Fundamental Theorem of Calculus for the Lebesgue integral	2990
9.8.21	Integration by parts	2993
9.8.22	A non-negative continuous function whose integral is zero must be zero	2995
9.8.23	Various common equivalent forms of function measurability	2998
9.8.24	Lebesgue sets and continuous images	3003
9.8.25	Affine lemmas	3005
9.8.26	More results on integrability	3008
9.8.27	Relation between Borel measurability and integrability.	3010
9.9	Harmonic Numbers	3018
9.9.1	The Harmonic numbers	3019
9.9.2	The Euler-Mascheroni constant	3021
9.9.3	Bounds on the Euler-Mascheroni constant	3025
9.10	The Gamma Function	3031
9.10.1	The Euler form and the logarithmic Gamma function	3036
9.10.2	The Polygamma functions	3045
9.10.3	Basic properties	3055
9.10.4	Differentiability	3059
9.10.5	The complex Gamma function	3060
9.10.6	The real Gamma function	3068
9.10.7	The uniqueness of the real Gamma function	3075
9.10.8	The Beta function	3078
9.10.9	Legendre duplication theorem	3079
9.10.10	Limits and residues	3093
9.10.11	Alternative definitions	3094
9.10.12	The Weierstraß product formula for the sine	3111
9.10.13	The Solution to the Basel problem	3112
9.10.14	Approximating a (possibly infinite) interval	3115
9.10.15	Basic properties of integration over an interval	3118
9.10.16	Basic properties of integration over an interval wrt lebesgue measure	3121
9.10.17	General limit approximation arguments	3125
9.10.18	A slightly stronger Fundamental Theorem of Calculus	3127
9.10.19	The substitution theorem	3131
9.11	Integration by Substitution for the Lebesgue Integral	3139
9.12	The Volume of an n -Dimensional Ball	3148

9.13	Integral Test for Summability	3155
9.14	Continuity of the indefinite integral; improper integral theorem	3158
9.14.1	Equiintegrability	3158
9.14.2	Subinterval restrictions for equiintegrable families . . .	3167
9.14.3	Continuity of the indefinite integral	3193
9.14.4	Second mean value theorem and corollaries	3201
9.15	Continuous Extensions of Functions	3210
9.15.1	Partitions of unity subordinate to locally finite open coverings	3210
9.15.2	Urysohn's Lemma for Euclidean Spaces	3212
9.15.3	Dugundji's Extension Theorem and Tietze Variants . .	3216
9.16	Equivalence Between Classical Borel Measurability and HOL Light's	3221
9.16.1	Austin's Lemma	3221
9.16.2	A differentiability-like property of the indefinite integral.	3224
9.16.3	HOL Light measurability	3232
9.16.4	Composing continuous and measurable functions; a few variants	3236
9.16.5	Monotonic functions are Lebesgue integrable	3256
9.16.6	Measurability on generalisations of the binary product	3257
9.17	Embedding Measure Spaces with a Function	3261
9.18	Brouwer's Fixed Point Theorem	3270
9.18.1	Retractions	3270
9.18.2	Kuhn Simplices	3274
9.18.3	Brouwer's fixed point theorem	3302
9.18.4	Applications	3309
9.19	Fashoda Meet Theorem	3323
9.19.1	Bijections between intervals	3323
9.19.2	Fashoda meet theorem	3324
9.19.3	Some slightly ad hoc lemmas I use below	3331
9.19.4	Useful Fashoda corollary pointed out to me by Tom Hales	3333
9.20	Vector Cross Products in 3 Dimensions	3337
9.20.1	Basic lemmas	3337
9.20.2	Preservation by rotation, or other orthogonal transformation up to sign	3340
9.20.3	Continuity	3341
9.21	Bounded Continuous Functions	3342
9.21.1	Definition	3342
9.21.2	Complete Space	3345
9.21.3	Supremum norm for a normed vector space	3346
9.21.4	(bounded) continuous extension	3348
9.22	Infinite Products	3348
9.22.1	Preliminaries	3348
9.22.2	Definitions and basic properties	3349

9.22.3	Absolutely convergent products	3353
9.22.4	Ignoring initial segments	3358
9.22.5	More elementary properties	3360
9.22.6	Infinite products on ordered topological monoids	3370
9.22.7	Infinite products on topological spaces	3374
9.22.8	Infinite summability on real normed fields	3376
9.22.9	Exponentials and logarithms	3382
9.22.10	Embeddings from the reals into some complete real normed field	3389
9.23	Sums over Infinite Sets	3390
9.24	Faces, Extreme Points, Polytopes, Polyhedra etc	3420
9.24.1	Faces of a (usually convex) set	3420
9.24.2	Exposed faces	3435
9.24.3	Extreme points of a set: its singleton faces	3439
9.24.4	Facets	3442
9.24.5	Edges: faces of affine dimension 1	3443
9.24.6	Existence of extreme points	3443
9.24.7	Krein-Milman, the weaker form	3444
9.24.8	Applying it to convex hulls of explicitly indicated finite sets	3446
9.24.9	Polytopes	3454
9.24.10	Polyhedra	3456
9.24.11	Canonical polyhedron representation making facial structure explicit	3458
9.24.12	More general corollaries from the explicit representation	3469
9.24.13	Relation between polytopes and polyhedra	3476
9.24.14	Relative and absolute frontier of a polytope	3478
9.24.15	Special case of a triangle	3479
9.24.16	Subdividing a cell complex	3480
9.24.17	Simplexes	3484
9.24.18	Simplicial complexes and triangulations	3487
9.24.19	Refining a cell complex to a simplicial complex	3487
9.24.20	Some results on cell division with full-dimensional cells only	3502
9.25	Finitely generated cone is polyhedral, and hence closed	3504
9.26	Absolute Retracts, Absolute Neighbourhood Retracts and Euclidean Neighbourhood Retracts	3507
9.26.1	Analogous properties of ENRs	3515
9.26.2	More advanced properties of ANRs and ENRs	3535
9.26.3	Original ANR material, now for ENRs	3540
9.26.4	Finally, spheres are ANRs and ENRs	3542
9.26.5	Spheres are connected, etc	3543
9.26.6	Borsuk homotopy extension theorem	3544
9.26.7	More extension theorems	3551

9.26.8	The complement of a set and path-connectedness . . .	3560
9.27	Extending Continuous Maps, Invariance of Domain, etc	3562
9.27.1	A map from a sphere to a higher dimensional sphere is nullhomotopic	3562
9.27.2	Some technical lemmas about extending maps from cell complexes	3570
9.27.3	Special cases and corollaries involving spheres	3583
9.27.4	Extending maps to spheres	3586
9.27.5	Invariance of domain and corollaries	3603
9.27.6	Formulation of loop homotopy in terms of maps out of type complex	3621
9.27.7	Homeomorphism of simple closed curves to circles . . .	3625
9.27.8	Dimension-based conditions for various homeomorphisms	3627
9.27.9	more invariance of domain	3628
9.27.10	The power, squaring and exponential functions as covering maps	3635
9.27.11	Hence the Borsukian results about mappings into circles	3642
9.27.12	Upper and lower hemicontinuous functions	3646
9.27.13	Complex logs exist on various "well-behaved" sets . . .	3650
9.27.14	Another simple case where sphere maps are nullhomotopic	3651
9.27.15	Holomorphic logarithms and square roots	3653
9.27.16	The "Borsukian" property of sets	3657
9.27.17	Unicoherence (closed)	3669
9.27.18	Several common variants of unicoherence	3675
9.27.19	Some separation results	3676
9.28	The Jordan Curve Theorem and Applications	3683
9.28.1	Janiszewski's theorem	3683
9.28.2	The Jordan Curve theorem	3687
9.29	Polynomial Functions: Extremal Behaviour and Root Counts .	3698
9.29.1	Basics about polynomial functions: extremal behaviour and root counts	3698
9.30	Generalised Binomial Theorem	3703
9.31	Vitali Covering Theorem and an Application to Negligibility .	3709
9.31.1	Vitali covering theorem	3714
9.32	Change of Variables Theorems	3724
9.32.1	Measurable Shear and Stretch	3724
9.32.2	Borel measurable Jacobian determinant	3749
9.32.3	Simplest case of Sard's theorem (we don't need continuity of derivative)	3767
9.32.4	A one-way version of change-of-variables not assuming injectivity.	3775
9.32.5	Change-of-variables theorem	3784
9.32.6	Change of variables for integrals: special case of linear function	3800

9.32.7	Change of variable for measure	3801
9.33	Lipschitz Continuity	3802
9.33.1	Local Lipschitz continuity	3810
9.33.2	Local Lipschitz continuity (uniform for a family of functions)	3813
9.34	Volume of a Simplex	3821
9.35	Convergence of Formal Power Series	3827
9.35.1	Balls with extended real radius	3827
9.35.2	Basic properties of convergent power series	3828
9.35.3	Lower bounds on radius of convergence	3830
9.35.4	Evaluating power series	3836
9.35.5	Power series expansions of analytic functions	3839
9.36	Smooth paths	3849
9.36.1	Homeomorphisms of arc images	3849
9.36.2	Piecewise differentiability of paths	3850
9.36.3	Valid paths, and their start and finish	3854
9.37	Metrics on product spaces	3861



Chapter 1

Linear Algebra

```
theory L2_Norm
imports Complex_Main
begin
```

1.1 L2 Norm

```
definition L2_set :: ('a ⇒ real) ⇒ 'a set ⇒ real where
L2_set f A = sqrt (∑ i∈A. (f i)2)
```

```
lemma L2_set_cong:
[[A = B; ∧x. x ∈ B ⇒ f x = g x]] ⇒ L2_set f A = L2_set g B
unfolding L2_set_def by simp
```

```
lemma L2_set_cong_simp:
[[A = B; ∧x. x ∈ B =simp=> f x = g x]] ⇒ L2_set f A = L2_set g B
unfolding L2_set_def simp_implies_def by simp
```

```
lemma L2_set_infinite [simp]: ¬ finite A ⇒ L2_set f A = 0
unfolding L2_set_def by simp
```

```
lemma L2_set_empty [simp]: L2_set f {} = 0
unfolding L2_set_def by simp
```

```
lemma L2_set_insert [simp]:
[[finite F; a ∉ F]] ⇒
L2_set f (insert a F) = sqrt ((f a)2 + (L2_set f F)2)
unfolding L2_set_def by (simp add: sum_nonneg)
```

```
lemma L2_set_nonneg [simp]: 0 ≤ L2_set f A
unfolding L2_set_def by (simp add: sum_nonneg)
```

```
lemma L2_set_0': ∀ a∈A. f a = 0 ⇒ L2_set f A = 0
unfolding L2_set_def by simp
```

lemma *L2_set_constant*: $L2_set (\lambda x. y) A = \text{sqrt} (\text{of_nat} (\text{card } A)) * |y|$
unfolding *L2_set_def* **by** (*simp add: real_sqrt_mult*)

lemma *L2_set_mono*:

assumes $\bigwedge i. i \in K \implies f i \leq g i$
assumes $\bigwedge i. i \in K \implies 0 \leq f i$
shows $L2_set f K \leq L2_set g K$
unfolding *L2_set_def*
by (*simp add: sum_nonneg sum_mono power_mono assms*)

lemma *L2_set_strict_mono*:

assumes *finite K and* $K \neq \{\}$
assumes $\bigwedge i. i \in K \implies f i < g i$
assumes $\bigwedge i. i \in K \implies 0 \leq f i$
shows $L2_set f K < L2_set g K$
unfolding *L2_set_def*
by (*simp add: sum_strict_mono power_strict_mono assms*)

lemma *L2_set_right_distrib*:

$0 \leq r \implies r * L2_set f A = L2_set (\lambda x. r * f x) A$
unfolding *L2_set_def*
apply (*simp add: power_mult_distrib*)
apply (*simp add: sum_distrib_left [symmetric]*)
apply (*simp add: real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_left_distrib*:

$0 \leq r \implies L2_set f A * r = L2_set (\lambda x. f x * r) A$
unfolding *L2_set_def*
apply (*simp add: power_mult_distrib*)
apply (*simp add: sum_distrib_right [symmetric]*)
apply (*simp add: real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_eq_0_iff*: $\text{finite } A \implies L2_set f A = 0 \iff (\forall x \in A. f x = 0)$

unfolding *L2_set_def*
by (*simp add: sum_nonneg sum_nonneg_eq_0_iff*)

proposition *L2_set_triangle_ineq*:

$L2_set (\lambda i. f i + g i) A \leq L2_set f A + L2_set g A$

proof (*cases finite A*)

case *False*
thus *?thesis* **by** *simp*

next

case *True*
thus *?thesis*
proof (*induct set: finite*)
case *empty*
show *?case* **by** *simp*


```

next
  case (insert x F)
  hence sqrt ((f x + g x)2 + (L2_set (λi. f i + g i) F)2) ≤
    sqrt ((f x + g x)2 + (L2_set f F + L2_set g F)2)
  by (intro real_sqrt_le_mono add_left_mono power_mono insert
      L2_set_nonneg add_increasing zero_le_power2)
  also have
    ... ≤ sqrt ((f x)2 + (L2_set f F)2) + sqrt ((g x)2 + (L2_set g F)2)
  by (rule real_sqrt_sum_squares_triangle_ineq)
  finally show ?case
    using insert by simp
qed
qed

```

```

lemma L2_set_le_sum [rule_format]:
  (∀ i ∈ A. 0 ≤ f i) ⟶ L2_set f A ≤ sum f A
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply clarsimp
  apply (erule order_trans [OF sqrt_sum_squares_le_sum])
  apply simp
  apply simp
  apply simp
  done

```

```

lemma L2_set_le_sum_abs: L2_set f A ≤ (∑ i ∈ A. |f i|)
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply simp
  apply (rule order_trans [OF sqrt_sum_squares_le_sum_abs])
  apply simp
  apply simp
  done

```

```

lemma L2_set_mult_ineq: (∑ i ∈ A. |f i| * |g i|) ≤ L2_set f A * L2_set g A
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply (rule power2_le_imp_le, simp)
  apply (rule order_trans)
  apply (rule power_mono)
  apply (erule add_left_mono)
  apply (simp add: sum_nonneg)
  apply (simp add: power2_sum)
  apply (simp add: power_mult_distrib)
  apply (simp add: distrib_left distrib_right)
  apply (rule ord_le_eq_trans)

```

```

apply (rule L2_set_mult_ineq_lemma)
apply simp_all
done

```

```

lemma member_le_L2_set:  $\llbracket \text{finite } A; i \in A \rrbracket \implies f\ i \leq L2\_set\ f\ A$ 
  unfolding L2_set_def
  by (auto intro!: member_le_sum_real_le_rsqr)

```

```

end

```

1.2 Inner Product Spaces and Gradient Derivative

```

theory Inner_Product
imports Complex_Main
begin

```

1.2.1 Real inner product spaces

Temporarily relax type constraints for *open*, *uniformity*, *dist*, and *norm*.

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const_name  $\langle \text{open} \rangle$ , SOME typ  $\langle 'a::\text{open\_set} \Rightarrow \text{bool} \rangle$ )

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const_name  $\langle \text{dist} \rangle$ , SOME typ  $\langle 'a::\text{dist} \Rightarrow 'a \Rightarrow \text{real} \rangle$ )

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const_name  $\langle \text{uniformity} \rangle$ , SOME typ  $\langle ('a::\text{uniformity} \times 'a)\ \text{filter} \rangle$ )

```

```

setup  $\langle \text{Sign.add\_const\_constraint}$ 
  (const_name  $\langle \text{norm} \rangle$ , SOME typ  $\langle 'a::\text{norm} \Rightarrow \text{real} \rangle$ )

```

```

class real_inner = real_vector + sgn_div_norm + dist_norm + uniformity_dist
+ open_uniformity +

```

```

  fixes inner :: 'a  $\Rightarrow$  'a  $\Rightarrow$  real

```

```

  assumes inner_commute: inner x y = inner y x

```

```

  and inner_add_left: inner (x + y) z = inner x z + inner y z

```

```

  and inner_scaleR_left [simp]: inner (scaleR r x) y = r * (inner x y)

```

```

  and inner_ge_zero [simp]:  $0 \leq \text{inner } x\ x$ 

```

```

  and inner_eq_zero_iff [simp]:  $\text{inner } x\ x = 0 \iff x = 0$ 

```

```

  and norm_eq_sqrt_inner:  $\text{norm } x = \text{sqrt } (\text{inner } x\ x)$ 

```

```

begin

```

```

lemma inner_zero_left [simp]: inner 0 x = 0
  using inner_add_left [of 0 0 x] by simp

```

```

lemma inner_minus_left [simp]: inner (- x) y = - inner x y
  using inner_add_left [of x - x y] by simp

```

lemma *inner_diff_left*: $\text{inner } (x - y) z = \text{inner } x z - \text{inner } y z$
using *inner_add_left* [of $x - y z$] **by** *simp*

lemma *inner_sum_left*: $\text{inner } (\sum_{x \in A}. f x) y = (\sum_{x \in A}. \text{inner } (f x) y)$
by (*cases finite A, induct set: finite, simp_all add: inner_add_left*)

lemma *all_zero_iff* [*simp*]: $(\forall u. \text{inner } x u = 0) \longleftrightarrow (x = 0)$
by *auto* (use *inner_eq_zero_iff* **in** *blast*)

Transfer distributivity rules to right argument.

lemma *inner_add_right*: $\text{inner } x (y + z) = \text{inner } x y + \text{inner } x z$
using *inner_add_left* [of $y z x$] **by** (*simp only: inner_commute*)

lemma *inner_scaleR_right* [*simp*]: $\text{inner } x (\text{scaleR } r y) = r * (\text{inner } x y)$
using *inner_scaleR_left* [of $r y x$] **by** (*simp only: inner_commute*)

lemma *inner_zero_right* [*simp*]: $\text{inner } x 0 = 0$
using *inner_zero_left* [of x] **by** (*simp only: inner_commute*)

lemma *inner_minus_right* [*simp*]: $\text{inner } x (-y) = - \text{inner } x y$
using *inner_minus_left* [of $y x$] **by** (*simp only: inner_commute*)

lemma *inner_diff_right*: $\text{inner } x (y - z) = \text{inner } x y - \text{inner } x z$
using *inner_diff_left* [of $y z x$] **by** (*simp only: inner_commute*)

lemma *inner_sum_right*: $\text{inner } x (\sum_{y \in A}. f y) = (\sum_{y \in A}. \text{inner } x (f y))$
using *inner_sum_left* [of $f A x$] **by** (*simp only: inner_commute*)

lemmas *inner_add* [*algebra_simps*] = *inner_add_left inner_add_right*
lemmas *inner_diff* [*algebra_simps*] = *inner_diff_left inner_diff_right*
lemmas *inner_scaleR* = *inner_scaleR_left inner_scaleR_right*

Legacy theorem names

lemmas *inner_left_distrib* = *inner_add_left*
lemmas *inner_right_distrib* = *inner_add_right*
lemmas *inner_distrib* = *inner_left_distrib inner_right_distrib*

lemma *inner_gt_zero_iff* [*simp*]: $0 < \text{inner } x x \longleftrightarrow x \neq 0$
by (*simp add: order_less_le*)

lemma *power2_norm_eq_inner*: $(\text{norm } x)^2 = \text{inner } x x$
by (*simp add: norm_eq_sqrt_inner*)

Identities involving real multiplication and division.

lemma *inner_mult_left*: $\text{inner } (\text{of_real } m * a) b = m * (\text{inner } a b)$
by (*metis real_inner_class.inner_scaleR_left scaleR_conv_of_real*)

lemma *inner_mult_right*: $\text{inner } a (\text{of_real } m * b) = m * (\text{inner } a b)$
by (*metis real_inner_class.inner_scaleR_right scaleR_conv_of_real*)

lemma *inner_mult_left'*: $\text{inner } (a * \text{of_real } m) b = m * (\text{inner } a b)$
by (*simp add: of_real_def*)

lemma *inner_mult_right'*: $\text{inner } a (b * \text{of_real } m) = (\text{inner } a b) * m$
by (*simp add: of_real_def real_inner_class.inner_scaleR_right*)

lemma *Cauchy_Schwarz_ineq*:

$(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$

proof (*cases*)

assume $y = 0$

thus *?thesis* **by** *simp*

next

assume $y: y \neq 0$

let $?r = \text{inner } x y / \text{inner } y y$

have $0 \leq \text{inner } (x - \text{scaleR } ?r y) (x - \text{scaleR } ?r y)$

by (*rule inner_ge_zero*)

also have $\dots = \text{inner } x x - \text{inner } y x * ?r$

by (*simp add: inner_diff*)

also have $\dots = \text{inner } x x - (\text{inner } x y)^2 / \text{inner } y y$

by (*simp add: power2_eq_square inner_commute*)

finally have $0 \leq \text{inner } x x - (\text{inner } x y)^2 / \text{inner } y y$.

hence $(\text{inner } x y)^2 / \text{inner } y y \leq \text{inner } x x$

by (*simp add: le_diff_eq*)

thus $(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$

by (*simp add: pos_divide_le_eq y*)

qed

lemma *Cauchy_Schwarz_ineq2*:

$|\text{inner } x y| \leq \text{norm } x * \text{norm } y$

proof (*rule power2_le_imp_le*)

have $(\text{inner } x y)^2 \leq \text{inner } x x * \text{inner } y y$

using *Cauchy_Schwarz_ineq* .

thus $|\text{inner } x y|^2 \leq (\text{norm } x * \text{norm } y)^2$

by (*simp add: power_mult_distrib power2_norm_eq_inner*)

show $0 \leq \text{norm } x * \text{norm } y$

unfolding *norm_eq_sqrt_inner*

by (*intro mult_nonneg_nonneg real_sqrt_ge_zero inner_ge_zero*)

qed

lemma *norm_cauchy_schwarz*: $\text{inner } x y \leq \text{norm } x * \text{norm } y$

using *Cauchy_Schwarz_ineq2* [*of x y*] **by** *auto*

subclass *real_normed_vector*

proof

fix $a :: \text{real}$ **and** $x y :: 'a$

show $\text{norm } x = 0 \iff x = 0$

unfolding *norm_eq_sqrt_inner* **by** *simp*

show $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$

```

proof (rule power2_le_imp_le)
  have inner x y ≤ norm x * norm y
    by (rule norm_cauchy_schwarz)
  thus (norm (x + y))2 ≤ (norm x + norm y)2
    unfolding power2_sum power2_norm_eq_inner
    by (simp add: inner_add inner_commute)
  show 0 ≤ norm x + norm y
    unfolding norm_eq_sqrt_inner by simp
qed

have sqrt (a2 * inner x x) = |a| * sqrt (inner x x)
  by (simp add: real_sqrt_mult)
then show norm (a *R x) = |a| * norm x
  unfolding norm_eq_sqrt_inner
  by (simp add: power2_eq_square mult.assoc)
qed

end

lemma square_bound_lemma:
  fixes x :: real
  shows x < (1 + x) * (1 + x)
proof -
  have (x + 1/2)2 + 3/4 > 0
    using zero_le_power2[of x+1/2] by arith
  then show ?thesis
    by (simp add: field_simps power2_eq_square)
qed

lemma square_continuous:
  fixes e :: real
  shows e > 0 ⇒ ∃ d. 0 < d ∧ (∀ y. |y - x| < d ⇒ |y * y - x * x| < e)
  using isCont_power[OF continuous_ident, of x, unfolded isCont_def LIM_eq,
  rule_format, of e 2]
  by (force simp add: power2_eq_square)

lemma norm_le: norm x ≤ norm y ↔ inner x x ≤ inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_lt: norm x < norm y ↔ inner x x < inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_eq: norm x = norm y ↔ inner x x = inner y y
  apply (subst order_eq_iff)
  apply (auto simp: norm_le)
  done

lemma norm_eq_1: norm x = 1 ↔ inner x x = 1
  by (simp add: norm_eq_sqrt_inner)

```

```

lemma inner_divide_left:
  fixes  $a :: 'a :: \{real\_inner, real\_div\_algebra\}$ 
  shows  $inner (a / of\_real\ m) b = (inner\ a\ b) / m$ 
  by (metis (no_types) divide_inverse inner_commute inner_scaleR_right mult.left_neutral
mult.right_neutral mult_scaleR_right of_real_inverse scaleR_conv_of_real times_divide_eq_left)

```

```

lemma inner_divide_right:
  fixes  $a :: 'a :: \{real\_inner, real\_div\_algebra\}$ 
  shows  $inner\ a\ (b / of\_real\ m) = (inner\ a\ b) / m$ 
  by (metis inner_commute inner_divide_left)

```

Re-enable constraints for *open*, *uniformity*, *dist*, and *norm*.

```

setup  $\langle Sign.add\_const\_constraint$ 
  (const_name  $\langle open \rangle$ , SOME typ  $\langle 'a::topological\_space\ set \Rightarrow bool \rangle$ )

```

```

setup  $\langle Sign.add\_const\_constraint$ 
  (const_name  $\langle uniformity \rangle$ , SOME typ  $\langle ('a::uniform\_space \times 'a)\ filter \rangle$ )

```

```

setup  $\langle Sign.add\_const\_constraint$ 
  (const_name  $\langle dist \rangle$ , SOME typ  $\langle 'a::metric\_space \Rightarrow 'a \Rightarrow real \rangle$ )

```

```

setup  $\langle Sign.add\_const\_constraint$ 
  (const_name  $\langle norm \rangle$ , SOME typ  $\langle 'a::real\_normed\_vector \Rightarrow real \rangle$ )

```

```

lemma bounded_bilinear_inner:
  bounded_bilinear (inner:: $'a::real\_inner \Rightarrow 'a \Rightarrow real$ )
proof
  fix  $x\ y\ z :: 'a$  and  $r :: real$ 
  show  $inner\ (x + y)\ z = inner\ x\ z + inner\ y\ z$ 
    by (rule inner_add_left)
  show  $inner\ x\ (y + z) = inner\ x\ y + inner\ x\ z$ 
    by (rule inner_add_right)
  show  $inner\ (scaleR\ r\ x)\ y = scaleR\ r\ (inner\ x\ y)$ 
    unfolding real_scaleR_def by (rule inner_scaleR_left)
  show  $inner\ x\ (scaleR\ r\ y) = scaleR\ r\ (inner\ x\ y)$ 
    unfolding real_scaleR_def by (rule inner_scaleR_right)
  show  $\exists K. \forall x\ y::'a. norm\ (inner\ x\ y) \leq norm\ x * norm\ y * K$ 
proof
  show  $\forall x\ y::'a. norm\ (inner\ x\ y) \leq norm\ x * norm\ y * 1$ 
    by (simp add: Cauchy_Schwarz_ineq2)
qed
qed

```

```

lemmas tendsto_inner [tendsto_intros] =
  bounded_bilinear.tendsto [OF bounded_bilinear_inner]

```

```

lemmas isCont_inner [simp] =
  bounded_bilinear.isCont [OF bounded_bilinear_inner]

```

lemmas *has_derivative_inner* [*derivative_intros*] =
bounded_bilinear.FDERIV [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_left* =
bounded_bilinear.bounded_linear_left [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_right* =
bounded_bilinear.bounded_linear_right [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_left_comp* = *bounded_linear_inner_left*[*THEN*
bounded_linear_compose]

lemmas *bounded_linear_inner_right_comp* = *bounded_linear_inner_right*[*THEN*
bounded_linear_compose]

lemmas *has_derivative_inner_right* [*derivative_intros*] =
bounded_linear.has_derivative [*OF bounded_linear_inner_right*]

lemmas *has_derivative_inner_left* [*derivative_intros*] =
bounded_linear.has_derivative [*OF bounded_linear_inner_left*]

lemma *differentiable_inner* [*simp*]:
*f differentiable (at x within s) \implies g differentiable at x within s \implies (λx . inner (f
x) (g x)) differentiable at x within s*
unfolding *differentiable_def* **by** (*blast intro: has_derivative_inner*)

1.2.2 Class instances

instantiation *real* :: *real_inner*
begin

definition *inner_real_def* [*simp*]: *inner* = (*)

instance

proof

fix *x y z r* :: *real*

show *inner x y = inner y x*

unfolding *inner_real_def* **by** (*rule mult.commute*)

show *inner (x + y) z = inner x z + inner y z*

unfolding *inner_real_def* **by** (*rule distrib_right*)

show *inner (scaleR r x) y = r * inner x y*

unfolding *inner_real_def real_scaleR_def* **by** (*rule mult.assoc*)

show $0 \leq \text{inner } x \ x$

unfolding *inner_real_def* **by** *simp*

show *inner x x = 0 \longleftrightarrow x = 0*

unfolding *inner_real_def* **by** *simp*

show *norm x = sqrt (inner x x)*

unfolding *inner_real_def* **by** *simp*

qed

end

lemma

shows *real_inner_1_left*[simp]: $\text{inner } 1 \ x = x$
 and *real_inner_1_right*[simp]: $\text{inner } x \ 1 = x$
 by *simp_all*

instantiation *complex* :: *real_inner*

begin

definition *inner_complex_def*:

$\text{inner } x \ y = \text{Re } x * \text{Re } y + \text{Im } x * \text{Im } y$

instance

proof

fix $x \ y \ z :: \text{complex}$ and $r :: \text{real}$
 show $\text{inner } x \ y = \text{inner } y \ x$
 unfolding *inner_complex_def* by (simp add: *mult commute*)
 show $\text{inner } (x + y) \ z = \text{inner } x \ z + \text{inner } y \ z$
 unfolding *inner_complex_def* by (simp add: *distrib_right*)
 show $\text{inner } (\text{scaleR } r \ x) \ y = r * \text{inner } x \ y$
 unfolding *inner_complex_def* by (simp add: *distrib_left*)
 show $0 \leq \text{inner } x \ x$
 unfolding *inner_complex_def* by simp
 show $\text{inner } x \ x = 0 \longleftrightarrow x = 0$
 unfolding *inner_complex_def*
 by (simp add: *add_nonneg_eq_0_iff complex_eq_iff*)
 show $\text{norm } x = \text{sqrt } (\text{inner } x \ x)$
 unfolding *inner_complex_def norm_complex_def*
 by (simp add: *power2_eq_square*)

qed

end

lemma *complex_inner_1* [simp]: $\text{inner } 1 \ x = \text{Re } x$

unfolding *inner_complex_def* by simp

lemma *complex_inner_1_right* [simp]: $\text{inner } x \ 1 = \text{Re } x$

unfolding *inner_complex_def* by simp

lemma *complex_inner_i_left* [simp]: $\text{inner } i \ x = \text{Im } x$

unfolding *inner_complex_def* by simp

lemma *complex_inner_i_right* [simp]: $\text{inner } x \ i = \text{Im } x$

unfolding *inner_complex_def* by simp

lemma *dot_square_norm*: $\text{inner } x \ x = (\text{norm } x)^2$

by (simp only: power2_norm_eq_inner)

lemma norm_eq_square: $\text{norm } x = a \iff 0 \leq a \wedge \text{inner } x \ x = a^2$
 by (auto simp add: norm_eq_sqrt_inner)

lemma norm_le_square: $\text{norm } x \leq a \iff 0 \leq a \wedge \text{inner } x \ x \leq a^2$
apply (simp add: dot_square_norm abs_le_square_iff[symmetric])
using norm_ge_zero[of x]
apply arith
done

lemma norm_ge_square: $\text{norm } x \geq a \iff a \leq 0 \vee \text{inner } x \ x \geq a^2$
apply (simp add: dot_square_norm abs_le_square_iff[symmetric])
using norm_ge_zero[of x]
apply arith
done

lemma norm_lt_square: $\text{norm } x < a \iff 0 < a \wedge \text{inner } x \ x < a^2$
 by (metis not_le norm_ge_square)

lemma norm_gt_square: $\text{norm } x > a \iff a < 0 \vee \text{inner } x \ x > a^2$
 by (metis norm_le_square not_less)

Dot product in terms of the norm rather than conversely.

lemmas inner_simps = inner_add_left inner_add_right inner_diff_right inner_diff_left
 inner_scaleR_left inner_scaleR_right

lemma dot_norm: $\text{inner } x \ y = ((\text{norm } (x + y))^2 - (\text{norm } x)^2 - (\text{norm } y)^2) / 2$
 by (simp only: power2_norm_eq_inner inner_simps inner_commute) auto

lemma dot_norm_neg: $\text{inner } x \ y = (((\text{norm } x)^2 + (\text{norm } y)^2) - (\text{norm } (x - y))^2) / 2$
 by (simp only: power2_norm_eq_inner inner_simps inner_commute)
 (auto simp add: algebra_simps)

lemma of_real_inner_1 [simp]:
 $\text{inner } (\text{of_real } x) \ (1 :: 'a :: \{\text{real_inner}, \text{real_normed_algebra_1}\}) = x$
 by (simp add: of_real_def dot_square_norm)

lemma summable_of_real_iff:
 $\text{summable } (\lambda x. \text{of_real } (f \ x)) :: 'a :: \{\text{real_normed_algebra_1}, \text{real_inner}\} \iff$
 $\text{summable } f$

proof

assume *: summable $(\lambda x. \text{of_real } (f \ x)) :: 'a$

interpret bounded_linear $\lambda x :: 'a. \text{inner } x \ 1$

by (rule bounded_linear_inner_left)

from summable [OF *] **show** summable f **by** simp

qed (auto intro: summable_of_real)

1.2.3 Gradient derivative

definition

gderiv ::
 ['a::real_inner \Rightarrow real, 'a, 'a] \Rightarrow bool
 ((GDERIV ($_$)/ ($_$)/ \Rightarrow ($_$)) [1000, 1000, 60] 60)

where

$GDERIV f x \Rightarrow D \iff FDERIV f x \Rightarrow (\lambda h. inner h D)$

lemma *gderiv_deriv* [simp]: $GDERIV f x \Rightarrow D \iff DERIV f x \Rightarrow D$

by (*simp only: gderiv_def has_field_derivative_def inner_real_def mult_commute_abs*)

lemma *GDERIV_DERIV_compose*:

$\llbracket GDERIV f x \Rightarrow df; DERIV g (f x) \Rightarrow dg \rrbracket$
 $\implies GDERIV (\lambda x. g (f x)) x \Rightarrow scaleR dg df$

unfolding *gderiv_def has_field_derivative_def*

apply (*drule (1) has_derivative_compose*)

apply (*simp add: ac_simps*)

done

lemma *has_derivative_subst*: $\llbracket FDERIV f x \Rightarrow df; df = d \rrbracket \implies FDERIV f x \Rightarrow d$

by *simp*

lemma *GDERIV_subst*: $\llbracket GDERIV f x \Rightarrow df; df = d \rrbracket \implies GDERIV f x \Rightarrow d$

by *simp*

lemma *GDERIV_const*: $GDERIV (\lambda x. k) x \Rightarrow 0$

unfolding *gderiv_def inner_zero_right* **by** (*rule has_derivative_const*)

lemma *GDERIV_add*:

$\llbracket GDERIV f x \Rightarrow df; GDERIV g x \Rightarrow dg \rrbracket$
 $\implies GDERIV (\lambda x. f x + g x) x \Rightarrow df + dg$

unfolding *gderiv_def inner_add_right* **by** (*rule has_derivative_add*)

lemma *GDERIV_minus*:

$GDERIV f x \Rightarrow df \implies GDERIV (\lambda x. - f x) x \Rightarrow - df$

unfolding *gderiv_def inner_minus_right* **by** (*rule has_derivative_minus*)

lemma *GDERIV_diff*:

$\llbracket GDERIV f x \Rightarrow df; GDERIV g x \Rightarrow dg \rrbracket$
 $\implies GDERIV (\lambda x. f x - g x) x \Rightarrow df - dg$

unfolding *gderiv_def inner_diff_right* **by** (*rule has_derivative_diff*)

lemma *GDERIV_scaleR*:

$\llbracket DERIV f x \Rightarrow df; GDERIV g x \Rightarrow dg \rrbracket$
 $\implies GDERIV (\lambda x. scaleR (f x) (g x)) x$
 $\Rightarrow scaleR (f x) dg + scaleR df (g x)$

unfolding *gderiv_def has_field_derivative_def inner_add_right inner_scaleR_right*
apply (*rule has_derivative_subst*)

```

apply (erule (1) has_derivative_scaleR)
apply (simp add: ac_simps)
done

```

```

lemma GDERIV_mult:
  [[GDERIV f x :> df; GDERIV g x :> dg]]
  ==> GDERIV ( $\lambda x. f x * g x$ ) x :> scaleR (f x) dg + scaleR (g x) df
unfolding gderiv_def
apply (rule has_derivative_subst)
apply (erule (1) has_derivative_mult)
apply (simp add: inner_add ac_simps)
done

```

```

lemma GDERIV_inverse:
  [[GDERIV f x :> df; f x  $\neq$  0]]
  ==> GDERIV ( $\lambda x. \text{inverse} (f x)$ ) x :> - (inverse (f x))2 *R df
by (metis DERIV_inverse GDERIV_DERIV_compose numerals(2))

```

```

lemma GDERIV_norm:
assumes x  $\neq$  0 shows GDERIV ( $\lambda x. \text{norm } x$ ) x :> sgn x
unfolding gderiv_def norm_eq_sqrt_inner
by (rule derivative_eq_intros | force simp add: inner_commute sgn_div_norm
norm_eq_sqrt_inner assms)+

```

```

lemmas has_derivative_norm = GDERIV_norm [unfolded gderiv_def]

```

```

bundle inner_syntax begin
notation inner (infix · 70)
end

```

```

bundle no_inner_syntax begin
no_notation inner (infix · 70)
end

```

```

end

```

1.3 Cartesian Products as Vector Spaces

```

theory Product_Vector
imports
  Complex_Main
  HOL-Library.Product_Plus
begin

```

```

lemma Times_eq_image_sum:
fixes S :: 'a :: comm_monoid_add set and T :: 'b :: comm_monoid_add set
shows S × T = {u + v | u v. u ∈ ( $\lambda x. (x, 0)$ ) ‘ S ∧ v ∈ Pair 0 ‘ T}
by force

```

1.3.1 Product is a Module

locale *module_prod* = *module_pair* **begin**

definition *scale* :: 'a \Rightarrow 'b \times 'c \Rightarrow 'b \times 'c
where *scale* a v = (s1 a (fst v), s2 a (snd v))

lemma *scale_prod*: *scale* x (a, b) = (s1 x a, s2 x b)
by (*auto simp: scale_def*)

sublocale *p*: *module scale*

proof qed (*simp_all add: scale_def*
m1.scale_left_distrib m1.scale_right_distrib m2.scale_left_distrib m2.scale_right_distrib)

lemma *subspace_Times*: *m1.subspace* A \Longrightarrow *m2.subspace* B \Longrightarrow *p.subspace* (A \times B)

unfolding *m1.subspace_def m2.subspace_def p.subspace_def*
by (*auto simp: zero_prod_def scale_def*)

lemma *module_hom_fst*: *module_hom scale s1 fst*
by *unfold_locales (auto simp: scale_def)*

lemma *module_hom_snd*: *module_hom scale s2 snd*
by *unfold_locales (auto simp: scale_def)*

end

locale *vector_space_prod* = *vector_space_pair* **begin**

sublocale *module_prod s1 s2*
rewrites *module_hom* = *Vector_Spaces.linear*
by *unfold_locales (fact module_hom_eq_linear)*

sublocale *p*: *vector_space scale* **by** *unfold_locales (auto simp: algebra_simps)*

lemmas *linear_fst* = *module_hom_fst*
and *linear_snd* = *module_hom_snd*

end

1.3.2 Product is a Real Vector Space

instantiation *prod* :: (*real_vector*, *real_vector*) *real_vector*
begin

definition *scaleR_prod_def*:
scaleR r A = (*scaleR* r (fst A), *scaleR* r (snd A))

lemma *fst_scaleR* [*simp*]: *fst* (*scaleR* r A) = *scaleR* r (fst A)
unfolding *scaleR_prod_def* **by** *simp*

lemma *snd_scaleR [simp]*: $\text{snd} (\text{scaleR } r \ A) = \text{scaleR } r (\text{snd } A)$
unfolding *scaleR_prod_def* **by** *simp*

proposition *scaleR_Pair [simp]*: $\text{scaleR } r (a, b) = (\text{scaleR } r \ a, \text{scaleR } r \ b)$
unfolding *scaleR_prod_def* **by** *simp*

instance

proof

fix $a \ b :: \text{real}$ **and** $x \ y :: 'a \times 'b$
show $\text{scaleR } a (x + y) = \text{scaleR } a \ x + \text{scaleR } a \ y$
by (*simp add: prod_eq_iff scaleR_right_distrib*)
show $\text{scaleR } (a + b) \ x = \text{scaleR } a \ x + \text{scaleR } b \ x$
by (*simp add: prod_eq_iff scaleR_left_distrib*)
show $\text{scaleR } a (\text{scaleR } b \ x) = \text{scaleR } (a * b) \ x$
by (*simp add: prod_eq_iff*)
show $\text{scaleR } 1 \ x = x$
by (*simp add: prod_eq_iff*)

qed

end

lemma *module_prod_scale_eq_scaleR*: $\text{module_prod.scale } (*_R) (*_R) = \text{scaleR}$
apply (*rule ext*) **apply** (*rule ext*)
apply (*subst module_prod_scale_def*)
subgoal by *unfold_locales*
by (*simp add: scaleR_prod_def*)

interpretation *real_vector?*: *vector_space_prod scaleR:: $_ \Rightarrow _ \Rightarrow 'a :: \text{real_vector}$ scaleR:: $_ \Rightarrow _ \Rightarrow 'b :: \text{real_vector}$*
rewrites $\text{scale} = ((*_R)::_ \Rightarrow _ \Rightarrow ('a \times 'b))$
and *module.dependent* $(*_R) = \text{dependent}$
and *module.representation* $(*_R) = \text{representation}$
and *module.subspace* $(*_R) = \text{subspace}$
and *module.span* $(*_R) = \text{span}$
and *vector_space.extend_basis* $(*_R) = \text{extend_basis}$
and *vector_space.dim* $(*_R) = \text{dim}$
and *Vector_Spaces.linear* $(*_R) (*_R) = \text{linear}$
subgoal by *unfold_locales*
subgoal by (*fact module_prod_scale_eq_scaleR*)
unfolding *dependent_raw_def representation_raw_def subspace_raw_def span_raw_def*
extend_basis_raw_def dim_raw_def linear_def
by (*rule refl*)**+**

1.3.3 Product is a Metric Space

instantiation *prod* :: $(\text{metric_space}, \text{metric_space}) \ \text{dist}$
begin

definition *dist_prod_def*[*code del*]:

$$\text{dist } x \ y = \text{sqrt } ((\text{dist } (\text{fst } x) (\text{fst } y))^2 + (\text{dist } (\text{snd } x) (\text{snd } y))^2)$$

instance ..
end

instantiation prod :: (uniformity, uniformity) uniformity begin

definition [code del]: $\langle (\text{uniformity} :: ('a \times 'b) \times ('a \times 'b)) \text{ filter} \rangle =$
 $\text{filtermap } (\lambda((x1,x2),(y1,y2)). ((x1,y1),(x2,y2))) (\text{uniformity} \times_F \text{uniformity}) \rangle$

instance..
end

Uniform spaces

instantiation prod :: (uniform_space, uniform_space) uniform_space
begin

instance

proof standard

fix $U :: \langle ('a \times 'b) \text{ set} \rangle$

show $\langle \text{open } U \longleftrightarrow (\forall x \in U. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U) \rangle$

proof (intro iffI ballI)

fix x **assume** $\langle \text{open } U \rangle$ **and** $\langle x \in U \rangle$

then obtain $A \ B$ **where** $\langle \text{open } A \rangle \langle \text{open } B \rangle \langle x \in A \times B \rangle \langle A \times B \subseteq U \rangle$

by (metis open_prod_elim)

define UA **where** $\langle UA = (\lambda(x::'a,y). x' = \text{fst } x \longrightarrow y \in A) \rangle$

from $\langle \text{open } A \rangle \langle x \in A \times B \rangle$

have $\langle \text{eventually } UA \text{ uniformity} \rangle$

unfolding open_uniformity UA_def **by** auto

define UB **where** $\langle UB = (\lambda(x::'b,y). x' = \text{snd } x \longrightarrow y \in B) \rangle$

from $\langle \text{open } A \rangle \langle \text{open } B \rangle \langle x \in A \times B \rangle$

have $\langle \text{eventually } UA \text{ uniformity} \rangle \langle \text{eventually } UB \text{ uniformity} \rangle$

unfolding open_uniformity UA_def UB_def **by** auto

then have $\langle \forall_F ((x'1, y1), (x'2, y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. (x'1, x'2) = x \longrightarrow (y1, y2) \in U \rangle$

apply (auto intro!: exI[of_ UA] exI[of_ UB] simp add: eventually_prod_filter)

using $\langle A \times B \subseteq U \rangle$ **by** (auto simp: UA_def UB_def)

then show $\langle \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U \rangle$

by (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold)

next

assume $asm: \langle \forall x \in U. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U \rangle$

show $\langle \text{open } U \rangle$

proof (unfold open_prod_def, intro ballI)

fix x **assume** $\langle x \in U \rangle$

with asm **have** $\langle \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U \rangle$

by auto

then have $\langle \forall_F ((x'1, y1), (x'2, y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. (x'1, x'2) = x \longrightarrow (y1, y2) \in U \rangle$

```

    by (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold)
    then obtain UA UB where ‹eventually UA uniformity› and ‹eventually UB
uniformity›
      and UA_UB_U: ‹UA (a1, a2)  $\implies$  UB (b1, b2)  $\implies$  (a1, b1) = x
 $\implies$  (a2, b2)  $\in$  U› for a1 a2 b1 b2
      apply atomize_elim by (simp add: case_prod_beta eventually_prod_filter)
      have ‹eventually ( $\lambda a.$  UA (fst x, a)) (nhds (fst x))›
      using ‹eventually UA uniformity› eventually_mono eventually_nhds_uniformity
by fastforce
    then obtain A where ‹open A› and A_UA: ‹A  $\subseteq$  {a. UA (fst x, a)}› and
‹fst x  $\in$  A›
      by (metis (mono_tags, lifting) eventually_nhds mem_Collect_eq subsetI)
      have ‹eventually ( $\lambda b.$  UB (snd x, b)) (nhds (snd x))›
      using ‹eventually UB uniformity› eventually_mono eventually_nhds_uniformity
by fastforce
    then obtain B where ‹open B› and B_UB: ‹B  $\subseteq$  {b. UB (snd x, b)}› and
‹snd x  $\in$  B›
      by (metis (mono_tags, lifting) eventually_nhds mem_Collect_eq subsetI)
      have ‹x  $\in$  A  $\times$  B›
      by (simp add: ‹fst x  $\in$  A› ‹snd x  $\in$  B› mem_Times_iff)
      have ‹A  $\times$  B  $\subseteq$  U›
      using A_UA B_UB UA_UB_U by fastforce
      show ‹ $\exists A B.$  open A  $\wedge$  open B  $\wedge$  x  $\in$  A  $\times$  B  $\wedge$  A  $\times$  B  $\subseteq$  U›
      using ‹A  $\times$  B  $\subseteq$  U› ‹open A› ‹open B› ‹x  $\in$  A  $\times$  B› by auto
    qed
  qed
next
  show ‹eventually E uniformity  $\implies$  E (x, x)› for E and x :: ‹a  $\times$  b›
    apply (simp add: uniformity_prod_def eventually_filtermap case_prod_unfold
eventually_prod_filter)
    by (metis surj_pair uniformity_refl)
next
  show ‹eventually E uniformity  $\implies$   $\forall_F$  (x::'a $\times$ 'b, y) in uniformity. E (y, x)› for
E
    apply (simp only: uniformity_prod_def eventually_filtermap case_prod_unfold
eventually_prod_filter)
    apply (erule exE, erule exE, rename_tac Pf Pg)
    apply (rule_tac x= $\lambda(x,y).$  Pf (y,x) in exI)
    apply (rule_tac x= $\lambda(x,y).$  Pg (y,x) in exI)
    by (auto simp add: uniformity_sym)
next
  show ‹ $\exists D.$  eventually D uniformity  $\wedge$  ( $\forall x y z.$  D (x::'a $\times$ 'b, y)  $\longrightarrow$  D (y, z)  $\longrightarrow$ 
E (x, z))›
    if ‹eventually E uniformity› for E
  proof –
    from that
    obtain EA EB where ‹eventually EA uniformity› and ‹eventually EB uniformity›
      and EA_EB_E: ‹EA (a1, a2)  $\implies$  EB (b1, b2)  $\implies$  E ((a1, b1), (a2,
```

```

b2))› for a1 a2 b1 b2
  by (auto simp add: uniformity_prod_def eventually_filtermap case_prod_unfold
    eventually_prod_filter)
  obtain DA where ‹eventually DA uniformity› and DA_EA: ‹DA (x,y) ⇒
DA (y,z) ⇒ EA (x,z)› for x y z
    using ‹eventually EA uniformity› uniformity_transE by blast
  obtain DB where ‹eventually DB uniformity› and DB_EB: ‹DB (x,y) ⇒
DB (y,z) ⇒ EB (x,z)› for x y z
    using ‹eventually EB uniformity› uniformity_transE by blast
  define D where ‹D = (λ((a1,b1),(a2,b2)). DA (a1,a2) ∧ DB (b1,b2))›
  have ‹eventually D uniformity›
    using ‹eventually DA uniformity› ‹eventually DB uniformity›
  by (auto simp add: uniformity_prod_def eventually_filtermap case_prod_unfold
    eventually_prod_filter D_def)
  moreover have ‹D ((a1, b1), (a2, b2)) ⇒ D ((a2, b2), (a3, b3)) ⇒ E
((a1, b1), (a3, b3))› for a1 b1 a2 b2 a3 b3
    using DA_EA DB_EB D_def EA_EB_E by blast
  ultimately show ?thesis
    by auto
qed
qed
end

```

```

lemma (in uniform_space) nhds_eq_comap_uniformity: nhds x = filtercomap
(λy. (x, y)) uniformity
proof -
  have *: eventually P (filtercomap (λy. (x, y)) F) ↔
    eventually (λz. fst z = x → P (snd z)) F for P :: 'a ⇒ bool and F
  unfolding eventually_filtercomap
  by (smt (verit) eventually_elim2 fst_conv prod.collapse snd_conv)
  thus ?thesis
    unfolding filter_eq_iff
  by (subst *) (auto simp: eventually_nhds_uniformity case_prod_unfold)
qed

```

```

lemma uniformity_of_uniform_continuous_invariant:
  fixes f :: 'a :: uniform_space ⇒ 'a ⇒ 'a
  assumes filterlim (λ((a,b),(c,d)). (f a c, f b d)) uniformity (uniformity ×F uni-
formity)
  assumes eventually P uniformity
  obtains Q where eventually Q uniformity ∧ a b c. Q (a, b) ⇒ P (f a c, f b c)
  using eventually_compose_filterlim[OF assms(2,1)] uniformity_refl
  by (fastforce simp: case_prod_unfold eventually_filtercomap eventually_prod_same)

```

```

class uniform_topological_monoid_add = topological_monoid_add + uniform_space
+
  assumes uniformly_continuous_add':
    filterlim (λ((a,b), (c,d)). (a + c, b + d)) uniformity (uniformity ×F uniformity)

```


lemma *uniformly_continuous_add*:

uniformly_continuous_on UNIV ($\lambda(x :: 'a :: \text{uniform_topological_monoid_add}, y). x + y$)
using *uniformly_continuous_add'* [**where** $?'a = 'a$]
by (*simp add: uniformly_continuous_on_uniformity case_prod_unfold uniformity_prod_def filterlim_filtermap*)

lemma *filterlim_fst*: *filterlim fst F* ($F \times_F G$)

by (*simp add: filterlim_def filtermap_fst_prod_filter*)

lemma *filterlim_snd*: *filterlim snd G* ($F \times_F G$)

by (*simp add: filterlim_def filtermap_snd_prod_filter*)

class *uniform_topological_group_add* = *topological_group_add* + *uniform_topological_monoid_add*
 +

assumes *uniformly_continuous_uminus'*: *filterlim* ($\lambda(a, b). (-a, -b)$) *uniformity*
uniformity

begin

lemma *uniformly_continuous_minus'*:

filterlim ($\lambda((a,b), (c,d)). (a - c, b - d)$) *uniformity* (*uniformity* \times_F *uniformity*)

proof -

have *filterlim* ($(\lambda((a,b), (c,d)). (a + c, b + d)) \circ (\lambda((a,b), (c,d)). ((a, b), (-c, -d)))$)

uniformity (*uniformity* \times_F *uniformity*)

unfolding *o_def* **using** *uniformly_continuous_uminus'*

by (*intro filterlim_compose* [*OF* *uniformly_continuous_add'*])

(*auto simp: case_prod_unfold intro!: filterlim_Pair*

filterlim_fst filterlim_compose [*OF* *filterlim_snd*])

thus *?thesis*

by (*simp add: o_def case_prod_unfold*)

qed

end

lemma *uniformly_continuous_uminus*:

uniformly_continuous_on UNIV ($\lambda(x :: 'a :: \text{uniform_topological_group_add}). -x$)

using *uniformly_continuous_uminus'* [**where** $?'a = 'a$]

by (*simp add: uniformly_continuous_on_uniformity*)

lemma *uniformly_continuous_minus*:

uniformly_continuous_on UNIV ($\lambda(x :: 'a :: \text{uniform_topological_group_add}, y). x - y$)

using *uniformly_continuous_minus'* [**where** $?'a = 'a$]

by (*simp add: uniformly_continuous_on_uniformity case_prod_unfold uniformity_prod_def filterlim_filtermap*)

```

lemma real_normed_vector_is_uniform_topological_group_add [Pure.intro]:
  OFCLASS('a :: real_normed_vector, uniform_topological_group_add_class)
proof
  show filterlim (λ((a::'a,b), (c,d)). (a + c, b + d)) uniformity (uniformity ×F
uniformity)
  unfolding filterlim_def le_filter_def eventually_filtermap case_prod_unfold
proof safe
  fix P :: 'a × 'a ⇒ bool
  assume eventually P uniformity
  then obtain ε where ε: ε > 0 ∧ x y. dist x y < ε ⇒ P (x, y)
  by (auto simp: eventually_uniformity_metric)
  define Q where Q = (λ(x::'a,y). dist x y < ε / 2)
  have Q: eventually Q uniformity
  unfolding eventually_uniformity_metric Q_def using ⟨ε > 0⟩
  by (meson case_prodI divide_pos_pos zero_less_numeral)
  have P (a + c, b + d) if Q (a, b) Q (c, d) for a b c d
proof -
  have dist (a + c) (b + d) ≤ dist a b + dist c d
  by (simp add: dist_norm norm_diff_triangle_ineq)
  also have ... < ε
  using that by (auto simp: Q_def)
  finally show ?thesis
  by (intro ε)
qed
  thus ∀F x in uniformity ×F uniformity. P (fst (fst x) + fst (snd x), snd (fst
x) + snd (snd x))
  unfolding eventually_prod_filter by (intro exI[of _ Q] conjI Q) auto
qed
next
  show filterlim (λ((a::'a), b). (-a, -b)) uniformity uniformity
  unfolding filterlim_def le_filter_def eventually_filtermap
proof safe
  fix P :: 'a × 'a ⇒ bool
  assume eventually P uniformity
  then obtain ε where ε: ε > 0 ∧ x y. dist x y < ε ⇒ P (x, y)
  by (auto simp: eventually_uniformity_metric)
  show ∀F x in uniformity. P (case x of (a, b) ⇒ (- a, - b))
  unfolding eventually_uniformity_metric
  by (intro exI[of _ ε]) (auto intro!: ε simp: dist_norm norm_minus_commute)
qed
qed
instance real :: uniform_topological_group_add ..
instance complex :: uniform_topological_group_add ..

lemma cauchy_seq_finset_iff_vanishing:
  uniformity = filtercomap (λ(x,y). y - x :: 'a :: uniform_topological_group_add)

```

```

(nhds 0)
proof -
  have filtercomap ( $\lambda x. (0, \text{case } x \text{ of } (x, y) \Rightarrow y - (x :: 'a))$ ) uniformity  $\leq$  uniformity
  apply (simp add: le_filter_def eventually_filtercomap)
  using uniformity_of_uniform_continuous_invariant[OF uniformly_continuous_add]
  by (metis diff_self eq_diff_eq)
  moreover
  have uniformity  $\leq$  filtercomap ( $\lambda x. (0, \text{case } x \text{ of } (x, y) \Rightarrow y - (x :: 'a))$ ) uniformity
  apply (simp add: le_filter_def eventually_filtercomap)
  using uniformity_of_uniform_continuous_invariant[OF uniformly_continuous_minus]
  by (metis (mono_tags) diff_self eventually_mono surjective_pairing)
  ultimately show ?thesis
  by (simp add: nhds_eq_comap_uniformity filtercomap_filtercomap)
qed

```

Metric spaces

```

instantiation prod :: (metric_space, metric_space) uniformity_dist begin
instance
proof
  show  $\langle \text{uniformity} = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x :: 'a \times 'b, y). \text{dist } x \ y < e\}) \rangle$ 
  proof (subst filter_eq_iff, intro allI iffI)
    fix P ::  $\langle ('a \times 'b) \times ('a \times 'b) \Rightarrow \text{bool} \rangle$ 

    have 1:  $\langle \exists e \in \{0 < ..\}. \{ (x, y). \text{dist } x \ y < e \} \subseteq \{ (x, y). \text{dist } x \ y < a \} \wedge \{ (x, y). \text{dist } x \ y < e \} \subseteq \{ (x, y). \text{dist } x \ y < b \} \text{ if } \langle a > 0 \rangle \langle b > 0 \rangle \text{ for } a \ b \rangle$ 
    apply (rule bexI[of _  $\langle \min a \ b \rangle$ ])
    using that by auto

    have 2:  $\langle \text{mono } (\lambda P. \text{eventually } (\lambda x. P (Q \ x)) \ F) \rangle$  for F ::  $\langle 'z \ \text{filter} \rangle$  and Q ::  $\langle 'z \Rightarrow 'y \rangle$ 
    unfolding mono_def using eventually_mono le_funD by fastforce

    have  $\langle \forall_F ((x1 :: 'a, y1), (x2 :: 'b, y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. \text{dist } x1 \ y1 < e/2 \wedge \text{dist } x2 \ y2 < e/2 \rangle \text{ if } \langle e > 0 \rangle \text{ for } e$ 
    by (auto intro!: eventually_prodI exI[of _  $\langle e/2 \rangle$ ] simp: case_prod_unfold eventually_uniformity_metric that)

    then have 3:  $\langle \forall_F ((x1 :: 'a, y1), (x2 :: 'b, y2)) \text{ in } \text{uniformity} \times_F \text{uniformity}. \text{dist } (x1, x2) (y1, y2) < e \rangle \text{ if } \langle e > 0 \rangle \text{ for } e$ 
    apply (rule eventually_rev_mp)
    by (auto intro!: that eventuallyI simp: case_prod_unfold dist_prod_def sqrt_sum_squares_half_less)

    show  $\langle \text{eventually } P (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x \ y < e\}) \Rightarrow \text{eventually } P \ \text{uniformity} \rangle$ 
    apply (subst (asm) eventually_INF_base)
    using 1 3 apply (auto simp: uniformity_prod_def case_prod_unfold eventually_filtermap 2 eventually_principal)
    by (smt (verit, best) eventually_mono)

```

```

next
  fix P :: ⟨('a × 'b) × ('a × 'b) ⇒ bool⟩
  assume ⟨eventually P uniformity⟩
  then obtain P1 P2 where ⟨eventually P1 uniformity⟩ ⟨eventually P2 uniformity⟩
  and P1P2P: ⟨P1 (x1, y1) ⇒ P2 (x2, y2) ⇒ P ((x1, x2), (y1, y2))⟩ for
  x1 y1 x2 y2
  by (auto simp: eventually_filtermap case_prod_beta eventually_prod_filter
  uniformity_prod_def)
  from ⟨eventually P1 uniformity⟩ obtain e1 where ⟨e1 > 0⟩ and e1P1: ⟨dist x
  y < e1 ⇒ P1 (x,y)⟩ for x y
  using eventually_uniformity_metric by blast
  from ⟨eventually P2 uniformity⟩ obtain e2 where ⟨e2 > 0⟩ and e2P2: ⟨dist x
  y < e2 ⇒ P2 (x,y)⟩ for x y
  using eventually_uniformity_metric by blast
  define e where ⟨e = min e1 e2⟩
  have ⟨e > 0⟩
  using ⟨0 < e1⟩ ⟨0 < e2⟩ e_def by auto
  have ⟨dist (x1,x2) (y1,y2) < e ⇒ dist x1 y1 < e1⟩ for x1 y1 :: 'a and x2
  y2 :: 'b
  unfolding dist_prod_def e_def apply auto
  by (smt (verit, best) real_sqrt_sum_squares_ge1)
  moreover have ⟨dist (x1,x2) (y1,y2) < e ⇒ dist x2 y2 < e2⟩ for x1 y1 ::
  'a and x2 y2 :: 'b
  unfolding dist_prod_def e_def apply auto
  by (smt (verit, best) real_sqrt_sum_squares_ge1)
  ultimately have *: ⟨dist (x1,x2) (y1,y2) < e ⇒ P ((x1, x2), (y1, y2))⟩ for
  x1 y1 x2 y2
  using e1P1 e2P2 P1P2P by auto

  show ⟨eventually P (INF e∈{0<..}. principal {(x, y). dist x y < e})⟩
  apply (rule eventually_INF1[where i=e])
  using ⟨e > 0⟩ * by (auto simp: eventually_principal)
qed
qed
end

declare uniformity_Abort[where 'a='a :: metric_space × 'b :: metric_space,
code]

instantiation prod :: (metric_space, metric_space) metric_space
begin

proposition dist_Pair_Pair: dist (a, b) (c, d) = sqrt ((dist a c)2 + (dist b d)2)
  unfolding dist_prod_def by simp

lemma dist_fst_le: dist (fst x) (fst y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge1)

```

```

lemma dist_snd_le: dist (snd x) (snd y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge2)

instance
proof
  fix x y :: 'a × 'b
  show dist x y = 0 ⟷ x = y
    unfolding dist_prod_def prod_eq_iff by simp
next
  fix x y z :: 'a × 'b
  show dist x y ≤ dist x z + dist y z
    unfolding dist_prod_def
    by (intro order_trans [OF real_sqrt_sum_squares_triangle_ineq]
      real_sqrt_le_mono add_mono power_mono dist_triangle2 zero_le_dist)
next
  fix S :: ('a × 'b) set
  have *: open S ⟷ (∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S)
  proof
    assume open S show ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S
    proof
      fix x assume x ∈ S
      obtain A B where open A open B x ∈ A × B A × B ⊆ S
        using ⟨open S⟩ and ⟨x ∈ S⟩ by (rule open_prod_elim)
      obtain r where r: 0 < r ∀ y. dist y (fst x) < r ⟶ y ∈ A
        using ⟨open A⟩ and ⟨x ∈ A × B⟩ unfolding open_dist by auto
      obtain s where s: 0 < s ∀ y. dist y (snd x) < s ⟶ y ∈ B
        using ⟨open B⟩ and ⟨x ∈ A × B⟩ unfolding open_dist by auto
      let ?e = min r s
      have 0 < ?e ∧ (∀ y. dist y x < ?e ⟶ y ∈ S)
      proof (intro allI impI conjI)
        show 0 < min r s by (simp add: r(1) s(1))
      next
        fix y assume dist y x < min r s
        hence dist y x < r and dist y x < s
          by simp_all
        hence dist (fst y) (fst x) < r and dist (snd y) (snd x) < s
          by (auto intro: le_less_trans dist_fst_le dist_snd_le)
        hence fst y ∈ A and snd y ∈ B
          by (simp_all add: r(2) s(2))
        hence y ∈ A × B by (induct y, simp)
        with ⟨A × B ⊆ S⟩ show y ∈ S ..
      qed
      thus ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S ..
    qed
  next
    assume *: ∀ x ∈ S. ∃ e > 0. ∀ y. dist y x < e ⟶ y ∈ S show open S
  proof (rule open_prod_intro)
    fix x assume x ∈ S
    then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
  
```

```

    using * by fast
    define r where r = e / sqrt 2
    define s where s = e / sqrt 2
    from ⟨0 < e⟩ have 0 < r and 0 < s
      unfolding r_def s_def by simp_all
    from ⟨0 < e⟩ have e = sqrt (r2 + s2)
      unfolding r_def s_def by (simp add: power_divide)
    define A where A = {y. dist (fst x) y < r}
    define B where B = {y. dist (snd x) y < s}
    have open A and open B
      unfolding A_def B_def by (simp_all add: open_ball)
    moreover have x ∈ A × B
      unfolding A_def B_def mem_Times_iff
      using ⟨0 < r⟩ and ⟨0 < s⟩ by simp
    moreover have A × B ⊆ S
    proof (clarify)
      fix a b assume a ∈ A and b ∈ B
      hence dist a (fst x) < r and dist b (snd x) < s
        unfolding A_def B_def by (simp_all add: dist_commute)
      hence dist (a, b) x < e
        unfolding dist_prod_def ⟨e = sqrt (r2 + s2)⟩
        by (simp add: add_strict_mono power_strict_mono)
      thus (a, b) ∈ S
        by (simp add: S)
    qed
    ultimately show ∃ A B. open A ∧ open B ∧ x ∈ A × B ∧ A × B ⊆ S by
fast
    qed
  qed
end

declare [[code abort: dist::('a::metric_space*'b::metric_space)⇒('a*'b) ⇒ real]]

lemma Cauchy_fst: Cauchy X ⇒ Cauchy (λn. fst (X n :: 'a::metric_space ×
'b::metric_space))
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_fst_le])

lemma Cauchy_snd: Cauchy X ⇒ Cauchy (λn. snd (X n :: 'a::metric_space ×
'b::metric_space))
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_snd_le])

lemma Cauchy_Pair:
  assumes Cauchy X and Cauchy Y
  shows Cauchy (λn. (X n :: 'a::metric_space, Y n :: 'a::metric_space))
proof (rule metric_CauchyI)
  fix r :: real assume 0 < r
  hence 0 < r / sqrt 2 (is 0 < ?s) by simp

```

```

obtain  $M$  where  $M: \forall m \geq M. \forall n \geq M. \text{dist } (X\ m) (X\ n) < ?s$ 
using metric_CauchyD [OF  $\langle \text{Cauchy } X \rangle \langle 0 < ?s \rangle$ ] ..
obtain  $N$  where  $N: \forall m \geq N. \forall n \geq N. \text{dist } (Y\ m) (Y\ n) < ?s$ 
using metric_CauchyD [OF  $\langle \text{Cauchy } Y \rangle \langle 0 < ?s \rangle$ ] ..
have  $\forall m \geq \max M\ N. \forall n \geq \max M\ N. \text{dist } (X\ m, Y\ m) (X\ n, Y\ n) < r$ 
using  $M\ N$  by (simp add: real_sqrt_sum_squares_less dist_Pair_Pair)
then show  $\exists n0. \forall m \geq n0. \forall n \geq n0. \text{dist } (X\ m, Y\ m) (X\ n, Y\ n) < r$  ..
qed

```

Analogue to *uniformly_continuous_on_def* for two-argument functions.

```

lemma uniformly_continuous_on_prod_metric:
fixes  $f :: \langle 'a::\text{metric\_space} \times 'b::\text{metric\_space} \rangle \Rightarrow 'c::\text{metric\_space}$ 
shows  $\langle \text{uniformly\_continuous\_on } (S \times T) f \iff (\forall e > 0. \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x\ y < d \longrightarrow \text{dist } x'\ y' < d \longrightarrow \text{dist } (f\ (x, x')) (f\ (y, y')) < e) \rangle$ 
proof (unfold uniformly_continuous_on_def, intro iffI impI allI)
fix  $e :: \text{real}$ 
assume  $\langle e > 0 \rangle$  and  $\langle \forall e > 0. \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x\ y < d \longrightarrow \text{dist } x'\ y' < d \longrightarrow \text{dist } (f\ (x, x')) (f\ (y, y')) < e \rangle$ 
then obtain  $d$  where  $\langle d > 0 \rangle$ 
and  $d: \langle \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x\ y < d \longrightarrow \text{dist } x'\ y' < d \longrightarrow \text{dist } (f\ (x, x')) (f\ (y, y')) < e \rangle$ 
by auto
show  $\langle \exists d > 0. \forall x \in S \times T. \forall y \in S \times T. \text{dist } y\ x < d \longrightarrow \text{dist } (f\ y) (f\ x) < e \rangle$ 
apply (rule exI[of _ d])
using  $\langle d > 0 \rangle$  d [rule_format] apply auto
by (smt (verit, del_insts) distfst_le distsnd_le fst_conv snd_conv)
next
fix  $e :: \text{real}$ 
assume  $\langle e > 0 \rangle$  and  $\langle \forall e > 0. \exists d > 0. \forall x \in S \times T. \forall x' \in S \times T. \text{dist } x'\ x < d \longrightarrow \text{dist } (f\ x') (f\ x) < e \rangle$ 
then obtain  $d$  where  $\langle d > 0 \rangle$  and  $d: \langle \forall x \in S \times T. \forall x' \in S \times T. \text{dist } x'\ x < d \longrightarrow \text{dist } (f\ x') (f\ x) < e \rangle$ 
by auto
show  $\langle \exists d > 0. \forall x \in S. \forall y \in S. \forall x' \in T. \forall y' \in T. \text{dist } x\ y < d \longrightarrow \text{dist } x'\ y' < d \longrightarrow \text{dist } (f\ (x, x')) (f\ (y, y')) < e \rangle$ 
proof (intro exI conjI impI ballI)
from  $\langle d > 0 \rangle$  show  $\langle d / 2 > 0 \rangle$  by auto
fix  $x\ y\ x'\ y'$ 
assume [simp]:  $\langle x \in S \rangle \langle y \in S \rangle \langle x' \in T \rangle \langle y' \in T \rangle$ 
assume  $\langle \text{dist } x\ y < d / 2 \rangle$  and  $\langle \text{dist } x'\ y' < d / 2 \rangle$ 
then have  $\langle \text{dist } (x, x') (y, y') < d \rangle$ 
by (simp add: dist_Pair_Pair sqrt_sum_squares_half_less)
with  $d$  show  $\langle \text{dist } (f\ (x, x')) (f\ (y, y')) < e \rangle$ 
by auto
qed
qed

```

Analogue to *isUCont_def* for two-argument functions.

```

lemma isUCont_prod_metric:
  fixes  $f :: \langle 'a::\text{metric\_space} \times 'b::\text{metric\_space} \Rightarrow 'c::\text{metric\_space} \rangle$ 
  shows  $\langle \text{isUCont } f \longleftrightarrow (\forall e>0. \exists d>0. \forall x. \forall y. \forall x'. \forall y'. \text{dist } x \ y < d \longrightarrow \text{dist } x' \ y' < d \longrightarrow \text{dist } (f \ (x, x')) \ (f \ (y, y')) < e) \rangle$ 
  using uniformly_continuous_on_prod_metric[of UNIV UNIV]
  by auto

```

This logically belong with the real vector spaces but we only have the necessary lemmas now.

```

lemma isUCont_plus[simp]:
  shows  $\langle \text{isUCont } (\lambda(x::'a::\text{real\_normed\_vector}, y). x+y) \rangle$ 
proof (rule isUCont_prod_metric[THEN iffD2], intro allI impI, simp)
  fix  $e :: \text{real}$  assume  $\langle 0 < e \rangle$ 
  show  $\langle \exists d>0. \forall x \ y :: 'a. \text{dist } x \ y < d \longrightarrow (\forall x' \ y'. \text{dist } x' \ y' < d \longrightarrow \text{dist } (x + x') \ (y + y') < e) \rangle$ 
  apply (rule exI[of  $\langle e/2 \rangle$ ])
  using  $\langle 0 < e \rangle$  apply auto
  by (smt (verit, ccfv_SIG) dist_add_cancel dist_add_cancel2 dist_commute dist_triangle_lt)
qed

```

1.3.4 Product is a Complete Metric Space

```

instance prod :: (complete_space, complete_space) complete_space
proof
  fix  $X :: \text{nat} \Rightarrow 'a \times 'b$  assume Cauchy X
  have  $1: (\lambda n. \text{fst } (X \ n)) \longrightarrow \text{lim } (\lambda n. \text{fst } (X \ n))$ 
  using Cauchy_fst [OF  $\langle \text{Cauchy } X \rangle$ ]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  have  $2: (\lambda n. \text{snd } (X \ n)) \longrightarrow \text{lim } (\lambda n. \text{snd } (X \ n))$ 
  using Cauchy_snd [OF  $\langle \text{Cauchy } X \rangle$ ]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  have  $X \longrightarrow (\text{lim } (\lambda n. \text{fst } (X \ n)), \text{lim } (\lambda n. \text{snd } (X \ n)))$ 
  using tendsto_Pair [OF 1 2] by simp
  then show convergent X
  by (rule convergentI)
qed

```

1.3.5 Product is a Normed Vector Space

```

instantiation prod :: (real_normed_vector, real_normed_vector) real_normed_vector
begin

```

```

definition norm_prod_def[code del]:
   $\text{norm } x = \text{sqrt } ((\text{norm } (\text{fst } x))^2 + (\text{norm } (\text{snd } x))^2)$ 

```

```

definition sgn_prod_def:
   $\text{sgn } (x::'a \times 'b) = \text{scaleR } (\text{inverse } (\text{norm } x)) \ x$ 

```


proposition *norm_Pair*: $\text{norm } (a, b) = \text{sqrt } ((\text{norm } a)^2 + (\text{norm } b)^2)$
unfolding *norm_prod_def* **by** *simp*

instance

proof

```

fix r :: real and x y :: 'a × 'b
show  $\text{norm } x = 0 \longleftrightarrow x = 0$ 
  unfolding norm_prod_def
  by (simp add: prod_eq_iff)
show  $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$ 
  unfolding norm_prod_def
  apply (rule order_trans [OF _ real_sqrt_sum_squares_triangle_ineq])
  apply (simp add: add_mono power_mono norm_triangle_ineq)
  done
show  $\text{norm } (\text{scaleR } r \ x) = |r| * \text{norm } x$ 
  unfolding norm_prod_def
  apply (simp add: power_mult_distrib)
  apply (simp add: distrib_left [symmetric])
  apply (simp add: real_sqrt_mult)
  done
show  $\text{sgn } x = \text{scaleR } (\text{inverse } (\text{norm } x)) \ x$ 
  by (rule sgn_prod_def)
show  $\text{dist } x \ y = \text{norm } (x - y)$ 
  unfolding dist_prod_def norm_prod_def
  by (simp add: dist_norm)

```

qed

end

declare [[*code abort: norm::('a::real_normed_vector*'b::real_normed_vector) => real*]]

instance *prod* :: (*banach, banach*) *banach* ..

Pair operations are linear

lemma *bounded_linear_fst*: *bounded_linear fst*
using *fst_add fst_scaleR*
by (*rule bounded_linear_intro [where K=1], simp add: norm_prod_def*)

lemma *bounded_linear_snd*: *bounded_linear snd*
using *snd_add snd_scaleR*
by (*rule bounded_linear_intro [where K=1], simp add: norm_prod_def*)

lemmas *bounded_linear_fst_comp* = *bounded_linear_fst [THEN bounded_linear_compose]*

lemmas *bounded_linear_snd_comp* = *bounded_linear_snd [THEN bounded_linear_compose]*

lemma *bounded_linear_Pair*:

```

    assumes f: bounded_linear f
    assumes g: bounded_linear g
    shows bounded_linear ( $\lambda x. (f x, g x)$ )
  proof
    interpret f: bounded_linear f by fact
    interpret g: bounded_linear g by fact
    fix x y and r :: real
    show (f (x + y), g (x + y)) = (f x, g x) + (f y, g y)
      by (simp add: f.add g.add)
    show (f (r *R x), g (r *R x)) = r *R (f x, g x)
      by (simp add: f.scale g.scale)
    obtain Kf where 0 < Kf and norm_f:  $\bigwedge x. \text{norm } (f x) \leq \text{norm } x * Kf$ 
      using f.pos_bounded by fast
    obtain Kg where 0 < Kg and norm_g:  $\bigwedge x. \text{norm } (g x) \leq \text{norm } x * Kg$ 
      using g.pos_bounded by fast
    have  $\forall x. \text{norm } (f x, g x) \leq \text{norm } x * (Kf + Kg)$ 
      apply (rule allI)
      apply (simp add: norm_Pair)
      apply (rule order_trans [OF sqrt_add_le_add_sqrt], simp, simp)
      apply (simp add: distrib_left)
      apply (rule add_mono [OF norm_f norm_g])
    done
    then show  $\exists K. \forall x. \text{norm } (f x, g x) \leq \text{norm } x * K ..$ 
  qed

```

Frechet derivatives involving pairs

```

  proposition has_derivative_Pair [derivative_intros]:
    assumes f: (f has_derivative f') (at x within s)
      and g: (g has_derivative g') (at x within s)
    shows (( $\lambda x. (f x, g x)$ ) has_derivative ( $\lambda h. (f' h, g' h)$ )) (at x within s)
  proof (rule has_derivativeI_sandwich[of 1])
    show bounded_linear ( $\lambda h. (f' h, g' h)$ )
      using f g by (intro bounded_linear_Pair has_derivative_bounded_linear)
    let ?Rf =  $\lambda y. f y - f x - f' (y - x)$ 
    let ?Rg =  $\lambda y. g y - g x - g' (y - x)$ 
    let ?R =  $\lambda y. ((f y, g y) - (f x, g x) - (f' (y - x), g' (y - x)))$ 

    show (( $\lambda y. \text{norm } (?Rf y) / \text{norm } (y - x) + \text{norm } (?Rg y) / \text{norm } (y - x)$ )
       $\longrightarrow 0$ ) (at x within s)
      using f g by (intro tendsto_add_zero) (auto simp: has_derivative_iff_norm)

    fix y :: 'a assume y  $\neq$  x
    show  $\text{norm } (?R y) / \text{norm } (y - x) \leq \text{norm } (?Rf y) / \text{norm } (y - x) + \text{norm } (?Rg y) / \text{norm } (y - x)$ 
      unfolding add_divide_distrib [symmetric]
      by (simp add: norm_Pair divide_right_mono order_trans [OF sqrt_add_le_add_sqrt])
  qed simp

```

lemma *differentiable_Pair* [simp, derivative_intros]:
 f differentiable at x within $s \implies g$ differentiable at x within $s \implies$
 $(\lambda x. (f x, g x))$ differentiable at x within s
unfolding *differentiable_def* **by** (blast intro: *has_derivative_Pair*)

lemmas *has_derivative_fst* [derivative_intros] = *bounded_linear.has_derivative*
[*OF bounded_linear_fst*]

lemmas *has_derivative_snd* [derivative_intros] = *bounded_linear.has_derivative*
[*OF bounded_linear_snd*]

lemma *has_derivative_split* [derivative_intros]:
 $((\lambda p. f (fst p) (snd p)) \text{ has_derivative } f') F \implies ((\lambda (a, b). f a b) \text{ has_derivative } f') F$
unfolding *split_beta'* .

Vector derivatives involving pairs

lemma *has_vector_derivative_Pair*[derivative_intros]:
assumes (f *has_vector_derivative* f') (at x within s)
(g *has_vector_derivative* g') (at x within s)
shows $((\lambda x. (f x, g x)) \text{ has_vector_derivative } (f', g'))$ (at x within s)
using *assms*
by (auto simp: *has_vector_derivative_def* intro!: *derivative_eq_intros*)

lemma
fixes $x :: 'a::\text{real_normed_vector}$
shows *norm_Pair1* [simp]: $\text{norm } (0, x) = \text{norm } x$
and *norm_Pair2* [simp]: $\text{norm } (x, 0) = \text{norm } x$
by (auto simp: *norm_Pair*)

lemma *norm_commute*: $\text{norm } (x, y) = \text{norm } (y, x)$
by (simp add: *norm_Pair*)

lemma *norm_fst_le*: $\text{norm } x \leq \text{norm } (x, y)$
by (metis *dist_fst_le fst_conv fst_zero norm_conv_dist*)

lemma *norm_snd_le*: $\text{norm } y \leq \text{norm } (x, y)$
by (metis *dist_snd_le snd_conv snd_zero norm_conv_dist*)

lemma *norm_Pair_le*:
shows $\text{norm } (x, y) \leq \text{norm } x + \text{norm } y$
unfolding *norm_Pair*
by (metis *norm_ge_zero sqrt_sum_squares_le_sum*)

lemma (in *vector_space_prod*) *span_Times_sing1*: $p.\text{span } (\{0\} \times B) = \{0\} \times$
 $vs2.\text{span } B$
apply (rule *p.span_unique*)
subgoal by (auto intro!: *vs1.span_base vs2.span_base*)
subgoal using *vs1.subspace_single_0 vs2.subspace_span* **by** (rule *subspace_Times*)

```

subgoal for T
proof safe
  fix b
  assume subset_T: {0} × B ⊆ T and subspace: p.subspace T and b_span: b
  ∈ vs2.span B
  then obtain t r where b: b = (∑ a∈t. r a * b a) and t: finite t t ⊆ B
  by (auto simp: vs2.span_explicit)
  have (0, b) = (∑ b∈t. scale (r b) (0, b))
  unfolding b scale_prod sum_prod
  by simp
  also have ... ∈ T
  using ⟨t ⊆ B⟩ subset_T
  by (auto intro!: p.subspace_sum p.subspace_scale subspace)
  finally show (0, b) ∈ T .
qed
done

```

```

lemma (in vector_space_prod) span_Times_sing2: p.span (A × {0}) = vs1.span
A × {0}
  apply (rule p.span_unique)
  subgoal by (auto intro!: vs1.span_base vs2.span_base)
  subgoal using vs1.subspace_span vs2.subspace_single_0 by (rule subspace_Times)
  subgoal for T
  proof safe
    fix a
    assume subset_T: A × {0} ⊆ T and subspace: p.subspace T and a_span: a
    ∈ vs1.span A
    then obtain t r where a: a = (∑ a∈t. r a * a a) and t: finite t t ⊆ A
    by (auto simp: vs1.span_explicit)
    have (a, 0) = (∑ a∈t. scale (r a) (a, 0))
    unfolding a scale_prod sum_prod
    by simp
    also have ... ∈ T
    using ⟨t ⊆ A⟩ subset_T
    by (auto intro!: p.subspace_sum p.subspace_scale subspace)
    finally show (a, 0) ∈ T .
  qed
done

```

1.3.6 Product is Finite Dimensional

```

lemma (in finite_dimensional_vector_space) zero_not_in_Basis[simp]: 0 ∉ Ba-
sis
  using dependent_zero local.independent_Basis by blast

```

```

locale finite_dimensional_vector_space_prod = vector_space_prod + finite_dimensional_vector_space
begin

```

```

definition Basis_pair = B1 × {0} ∪ {0} × B2

```

```

sublocale p: finite_dimensional_vector_space scale Basis_pair
proof unfold_locales
  show finite_Basis_pair
    by (auto intro!: finite_cartesian_product vs1.finite_Basis vs2.finite_Basis
simp: Basis_pair_def)
  show p.independent_Basis_pair
    unfolding p.dependent_def Basis_pair_def
  proof safe
    fix a
    assume a: a ∈ B1
    assume  $(a, 0) \in p.\text{span } (B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\})$ 
    also have  $B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\} = (B1 - \{a\}) \times \{0\} \cup \{0\} \times$ 
B2
      by auto
    finally show False
      using a vs1.dependent_def vs1.independent_Basis
      by (auto simp: p.span_Un span_Times_sing1 span_Times_sing2)
  next
    fix b
    assume b: b ∈ B2
    assume  $(0, b) \in p.\text{span } (B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\})$ 
    also have  $(B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\}) = B1 \times \{0\} \cup \{0\} \times (B2 -$ 
 $\{b\})$ 
      by auto
    finally show False
      using b vs2.dependent_def vs2.independent_Basis
      by (auto simp: p.span_Un span_Times_sing1 span_Times_sing2)
  qed
show p.span Basis_pair = UNIV
  by (auto simp: p.span_Un span_Times_sing2 span_Times_sing1 vs1.span_Basis
vs2.span_Basis
Basis_pair_def)
qed

proposition dim_Times:
assumes vs1.subspace S vs2.subspace T
shows  $p.\text{dim}(S \times T) = vs1.\text{dim } S + vs2.\text{dim } T$ 
proof –
  interpret p1: Vector_Spaces.linear s1 scale  $(\lambda x. (x, 0))$ 
    by unfold_locales (auto simp: scale_def)
  interpret pair1: finite_dimensional_vector_space_pair  $(*a) B1$  scale Basis_pair
    by unfold_locales
  interpret p2: Vector_Spaces.linear s2 scale  $(\lambda x. (0, x))$ 
    by unfold_locales (auto simp: scale_def)
  interpret pair2: finite_dimensional_vector_space_pair  $(*b) B2$  scale Basis_pair
    by unfold_locales
  have ss: p.subspace  $((\lambda x. (x, 0)) \text{ ‘ } S)$  p.subspace  $(\text{Pair } 0 \text{ ‘ } T)$ 
    by (rule p1.subspace_image p2.subspace_image assms)+

```

```

have  $p.\dim(S \times T) = p.\dim(\{u + v \mid u \in (\lambda x. (x, 0)) \text{ ' } S \wedge v \in \text{Pair } 0 \text{ ' } T\})$ 
by (simp add: Times_eq_image_sum)
moreover have  $p.\dim((\lambda x. (x, 0::'c)) \text{ ' } S) = vs1.\dim S + p.\dim(\text{Pair } 0::'b \text{ ' } T)$ 
 $= vs2.\dim T$ 
by (simp_all add: inj_on_def p1.linear_axioms pair1.dim_image_eq p2.linear_axioms pair2.dim_image_eq)
moreover have  $p.\dim((\lambda x. (x, 0)) \text{ ' } S \cap \text{Pair } 0 \text{ ' } T) = 0$ 
by (subst p.dim_eq_0) auto
ultimately show ?thesis
using p.dim_sums_Int [OF ss] by linarith
qed

```

```

lemma dimension_pair:  $p.\text{dimension} = vs1.\text{dimension} + vs2.\text{dimension}$ 
using dim_Times [OF vs1.subspace_UNIV vs2.subspace_UNIV]
by (auto simp: p.dimension_def vs1.dimension_def vs2.dimension_def)

```

end

end

1.4 Finite-Dimensional Inner Product Spaces

```

theory Euclidean_Space
imports
  L2_Norm
  Inner_Product
  Product_Vector
begin

```

1.4.1 Interlude: Some properties of real sets

```

lemma seq_mono_lemma:
assumes  $\forall (n::\text{nat}) \geq m. (d \ n :: \text{real}) < e \ n$ 
and  $\forall n \geq m. e \ n \leq e \ m$ 
shows  $\forall n \geq m. d \ n < e \ m$ 
using assms by force

```

1.4.2 Type class of Euclidean spaces

```

class euclidean_space = real_inner +
fixes Basis :: 'a set
assumes nonempty_Basis [simp]:  $\text{Basis} \neq \{\}$ 
assumes finite_Basis [simp]: finite Basis
assumes inner_Basis:
   $\llbracket u \in \text{Basis}; v \in \text{Basis} \rrbracket \implies \text{inner } u \ v = (\text{if } u = v \text{ then } 1 \text{ else } 0)$ 
assumes euclidean_all_zero_iff:
   $(\forall u \in \text{Basis}. \text{inner } x \ u = 0) \longleftrightarrow (x = 0)$ 

```

```

syntax _type_dimension :: type  $\Rightarrow$  nat ((1DIM/(1'(_'))))
translations DIM('a)  $\rightarrow$  CONST card (CONST Basis :: 'a set)
typed_print_translation  $\langle$ 
  [(const_syntax  $\langle$  card  $\rangle$ ,
    fn ctxt  $\Rightarrow$  fn  $\_ \Rightarrow$  fn [Const (const_syntax  $\langle$  Basis  $\rangle$ , Type (type_name  $\langle$  set  $\rangle$ ,
  [T])]]  $\Rightarrow$ 
    Syntax.const syntax_const  $\langle$  _type_dimension  $\rangle$  $ Syntax_Phases.term_of_typ
  ctxt T)]
 $\rangle$ 

```

lemma (in *euclidean_space*) *norm_Basis[simp]*: $u \in \text{Basis} \implies \text{norm } u = 1$
unfolding *norm_eq_sqrt_inner* **by** (*simp add: inner_Basis*)

lemma (in *euclidean_space*) *inner_same_Basis[simp]*: $u \in \text{Basis} \implies \text{inner } u \ u = 1$
by (*simp add: inner_Basis*)

lemma (in *euclidean_space*) *inner_not_same_Basis*: $u \in \text{Basis} \implies v \in \text{Basis} \implies u \neq v \implies \text{inner } u \ v = 0$
by (*simp add: inner_Basis*)

lemma (in *euclidean_space*) *sgn_Basis*: $u \in \text{Basis} \implies \text{sgn } u = u$
unfolding *sgn_div_norm* **by** (*simp add: scaleR_one*)

lemma *inner_sum_Basis[simp]*: $i \in \text{Basis} \implies \text{inner } (\sum \text{Basis}) \ i = 1$
by (*simp add: inner_sum_left sum.If_cases inner_Basis*)

lemma (in *euclidean_space*) *Basis_zero [simp]*: $0 \notin \text{Basis}$

proof

assume $0 \in \text{Basis}$ **thus** *False*

using *inner_Basis [of 0 0]* **by** *simp*

qed

lemma (in *euclidean_space*) *nonzero_Basis*: $u \in \text{Basis} \implies u \neq 0$
by *clarsimp*

lemma (in *euclidean_space*) *SOME_Basis*: $(\text{SOME } i. i \in \text{Basis}) \in \text{Basis}$
by (*metis ex_in_conv nonempty_Basis someI_ex*)

lemma *norm_some_Basis [simp]*: $\text{norm } (\text{SOME } i. i \in \text{Basis}) = 1$
by (*simp add: SOME_Basis*)

lemma (in *euclidean_space*) *inner_sum_left_Basis[simp]*:
 $b \in \text{Basis} \implies \text{inner } (\sum_{i \in \text{Basis}} f \ i *_{\mathbb{R}} \ i) \ b = f \ b$
by (*simp add: inner_sum_left inner_Basis if_distrib comm_monoid_add_class.sum.If_cases*)

lemma (in *euclidean_space*) *euclidean_eqI*:

assumes $b: \bigwedge b. b \in \text{Basis} \implies \text{inner } x \ b = \text{inner } y \ b$ **shows** $x = y$

proof –

from b **have** $\forall b \in \text{Basis}. \text{inner } (x - y) b = 0$
by (*simp add: inner_diff_left*)
then show $x = y$
by (*simp add: euclidean_all_zero_iff*)
qed

lemma (*in euclidean_space*) *euclidean_eq_iff*:
 $x = y \longleftrightarrow (\forall b \in \text{Basis}. \text{inner } x b = \text{inner } y b)$
by (*auto intro: euclidean_eqI*)

lemma (*in euclidean_space*) *euclidean_representation_sum*:
 $(\sum i \in \text{Basis}. f i *_R i) = b \longleftrightarrow (\forall i \in \text{Basis}. f i = \text{inner } b i)$
by (*subst euclidean_eq_iff*) *simp*

lemma (*in euclidean_space*) *euclidean_representation_sum'*:
 $b = (\sum i \in \text{Basis}. f i *_R i) \longleftrightarrow (\forall i \in \text{Basis}. f i = \text{inner } b i)$
by (*auto simp add: euclidean_representation_sum[symmetric]*)

lemma (*in euclidean_space*) *euclidean_representation*: $(\sum b \in \text{Basis}. \text{inner } x b *_R b) = x$
unfolding *euclidean_representation_sum* **by** *simp*

lemma (*in euclidean_space*) *euclidean_inner*: $\text{inner } x y = (\sum b \in \text{Basis}. (\text{inner } x b) * (\text{inner } y b))$
by (*subst (1 2) euclidean_representation [symmetric]*)
(simp add: inner_sum_right inner_Basis ac_simps)

lemma (*in euclidean_space*) *choice_Basis_iff*:
fixes $P :: 'a \Rightarrow \text{real} \Rightarrow \text{bool}$
shows $(\forall i \in \text{Basis}. \exists x. P i x) \longleftrightarrow (\exists x. \forall i \in \text{Basis}. P i (\text{inner } x i))$
unfolding *bchoice_iff*
proof *safe*
fix f **assume** $\forall i \in \text{Basis}. P i (f i)$
then show $\exists x. \forall i \in \text{Basis}. P i (\text{inner } x i)$
by (*auto intro!: exI[of _ \sum i \in \text{Basis}. f i *_R i]*)
qed *auto*

lemma (*in euclidean_space*) *bchoice_Basis_iff*:
fixes $P :: 'a \Rightarrow \text{real} \Rightarrow \text{bool}$
shows $(\forall i \in \text{Basis}. \exists x \in A. P i x) \longleftrightarrow (\exists x. \forall i \in \text{Basis}. \text{inner } x i \in A \wedge P i (\text{inner } x i))$
by (*simp add: choice_Basis_iff Bex_def*)

lemma (*in euclidean_space*) *euclidean_representation_sum_fun*:
 $(\lambda x. \sum b \in \text{Basis}. \text{inner } (f x) b *_R b) = f$
by (*rule ext*) (*simp add: euclidean_representation_sum*)

lemma *euclidean_isCont*:
assumes $\bigwedge b. b \in \text{Basis} \Longrightarrow \text{isCont } (\lambda x. (\text{inner } (f x) b) *_R b) x$


```

  shows isCont f x
  apply (subst euclidean_representation_sum_fun [symmetric])
  apply (rule isCont_sum)
  apply (blast intro: assms)
  done

```

```

lemma DIM_positive [simp]: 0 < DIM('a::euclidean_space)
  by (simp add: card_gt_0_iff)

```

```

lemma DIM_ge_Suc0 [simp]: Suc 0 ≤ card Basis
  by (meson DIM_positive Suc_leI)

```

```

lemma sum_inner_Basis_scaleR [simp]:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_vector
  assumes b ∈ Basis shows (∑ i∈Basis. (inner i b) *R f i) = f b
  by (simp add: comm_monoid_add_class.sum_remove [OF finite_Basis assms]
      assms inner_not_same_Basis comm_monoid_add_class.sum_neutral)

```

```

lemma sum_inner_Basis_eq [simp]:
  assumes b ∈ Basis shows (∑ i∈Basis. (inner i b) * f i) = f b
  by (simp add: comm_monoid_add_class.sum_remove [OF finite_Basis assms]
      assms inner_not_same_Basis comm_monoid_add_class.sum_neutral)

```

```

lemma sum_if_inner [simp]:
  assumes i ∈ Basis j ∈ Basis
  shows inner (∑ k∈Basis. if k = i then f i *R i else g k *R k) j = (if j=i then
f j else g j)
  proof (cases i=j)
  case True
  with assms show ?thesis
  by (auto simp: inner_sum_left if_distrib [of λx. inner x j] inner_Basis cong:
if_cong)
  next
  case False
  have (∑ k∈Basis. inner (if k = i then f i *R i else g k *R k) j) =
    (∑ k∈Basis. if k = j then g k else 0)
  apply (rule sum.cong)
  using False assms by (auto simp: inner_Basis)
  also have ... = g j
  using assms by auto
  finally show ?thesis
  using False by (auto simp: inner_sum_left)
  qed

```

```

lemma norm_le_componentwise:
  (∧ b. b ∈ Basis ⇒ abs(inner x b) ≤ abs(inner y b)) ⇒ norm x ≤ norm y
  by (auto simp: norm_le euclidean_inner [of x x] euclidean_inner [of y y] abs_le_square_iff
power2_eq_square intro!: sum_mono)

```

lemma *Basis_le_norm*: $b \in \text{Basis} \implies |\text{inner } x \ b| \leq \text{norm } x$
by (*rule order_trans [OF Cauchy_Schwarz_ineq2]*) *simp*

lemma *norm_bound_Basis_le*: $b \in \text{Basis} \implies \text{norm } x \leq e \implies |\text{inner } x \ b| \leq e$
by (*metis Basis_le_norm order_trans*)

lemma *norm_bound_Basis_lt*: $b \in \text{Basis} \implies \text{norm } x < e \implies |\text{inner } x \ b| < e$
by (*metis Basis_le_norm le_less_trans*)

lemma *norm_le_l1*: $\text{norm } x \leq (\sum_{b \in \text{Basis}} |\text{inner } x \ b|)$
apply (*subst euclidean_representation[of x, symmetric]*)
apply (*rule order_trans[OF norm_sum]*)
apply (*auto intro!: sum_mono*)
done

lemma *sum_norm_allsubsets_bound*:
fixes $f :: 'a \Rightarrow 'n::\text{euclidean_space}$
assumes fP : *finite P*
and fPs : $\bigwedge Q. Q \subseteq P \implies \text{norm } (\text{sum } f \ Q) \leq e$
shows $(\sum_{x \in P}. \text{norm } (f \ x)) \leq 2 * \text{real } \text{DIM}('n) * e$
proof –
have $(\sum_{x \in P}. \text{norm } (f \ x)) \leq (\sum_{x \in P}. \sum_{b \in \text{Basis}} |\text{inner } (f \ x) \ b|)$
by (*rule sum_mono*) (*rule norm_le_l1*)
also have $(\sum_{x \in P}. \sum_{b \in \text{Basis}} |\text{inner } (f \ x) \ b|) = (\sum_{b \in \text{Basis}}. \sum_{x \in P}. |\text{inner } (f \ x) \ b|)$
by (*rule sum.swap*)
also have $\dots \leq \text{of_nat } (\text{card } (\text{Basis} :: 'n \ \text{set})) * (2 * e)$
proof (*rule sum_bounded_above*)
fix $i :: 'n$
assume i : $i \in \text{Basis}$
have $\text{norm } (\sum_{x \in P}. |\text{inner } (f \ x) \ i|) \leq$
 $\text{norm } (\text{inner } (\sum_{x \in P \cap \{x. \text{inner } (f \ x) \ i < 0\}}. f \ x) \ i) + \text{norm } (\text{inner } (\sum_{x \in P \cap \{x. \text{inner } (f \ x) \ i < 0\}}. f \ x) \ i)$
by (*simp add: abs_real_def sum.If_cases[OF fP] sum_negf norm_triangle_ineq4 inner_sum_left del: real_norm_def*)
also have $\dots \leq e + e$
unfolding *real_norm_def*
by (*intro add_mono norm_bound_Basis_le i fPs*) *auto*
finally show $(\sum_{x \in P}. |\text{inner } (f \ x) \ i|) \leq 2 * e$ **by** *simp*
qed
also have $\dots = 2 * \text{real } \text{DIM}('n) * e$ **by** *simp*
finally show *?thesis* .
qed

1.4.3 Subclass relationships

instance *euclidean_space* \subseteq *perfect_space*

```

proof
  fix  $x :: 'a$  show  $\neg \text{open } \{x\}$ 
  proof
    assume  $\text{open } \{x\}$ 
    then obtain  $e$  where  $0 < e$  and  $e: \forall y. \text{dist } y \ x < e \longrightarrow y = x$ 
    unfolding  $\text{open\_dist}$  by  $\text{fast}$ 
    define  $y$  where  $y = x + \text{scaleR } (e/2) (\text{SOME } b. b \in \text{Basis})$ 
    have  $[\text{simp}]: (\text{SOME } b. b \in \text{Basis}) \in \text{Basis}$ 
    by  $(\text{rule } \text{someI\_ex}) (\text{auto } \text{simp: } \text{ex\_in\_conv})$ 
    from  $\langle 0 < e \rangle$  have  $y \neq x$ 
    unfolding  $y\_def$  by  $(\text{auto } \text{intro!: } \text{nonzero\_Basis})$ 
    from  $\langle 0 < e \rangle$  have  $\text{dist } y \ x < e$ 
    unfolding  $y\_def$  by  $(\text{simp } \text{add: } \text{dist\_norm})$ 
    from  $\langle y \neq x \rangle$  and  $\langle \text{dist } y \ x < e \rangle$  show  $\text{False}$ 
    using  $e$  by  $\text{simp}$ 
  qed
qed

```

1.4.4 Class instances

Type real

instantiation $\text{real} :: \text{euclidean_space}$
begin

definition

$[\text{simp}]: \text{Basis} = \{1 :: \text{real}\}$

instance

by $\text{standard } \text{auto}$

end

lemma $\text{DIM_real}[\text{simp}]: \text{DIM}(\text{real}) = 1$

by simp

Type complex

instantiation $\text{complex} :: \text{euclidean_space}$
begin

definition $\text{Basis_complex_def}: \text{Basis} = \{1, i\}$

instance

by $\text{standard } (\text{auto } \text{simp } \text{add: } \text{Basis_complex_def } \text{intro: } \text{complex_eqI } \text{split: } \text{if_split_asm})$

end

lemma $\text{DIM_complex}[\text{simp}]: \text{DIM}(\text{complex}) = 2$

unfolding Basis_complex_def **by** simp

lemma *complex_Basis_1 [iff]:* $(1::\text{complex}) \in \text{Basis}$
by (*simp add: Basis_complex_def*)

lemma *complex_Basis_i [iff]:* $i \in \text{Basis}$
by (*simp add: Basis_complex_def*)

Type $'a \times 'b$

instantiation *prod :: (real_inner, real_inner) real_inner*
begin

definition *inner_prod_def:*
 $\text{inner } x \ y = \text{inner } (\text{fst } x) (\text{fst } y) + \text{inner } (\text{snd } x) (\text{snd } y)$

lemma *inner_Pair [simp]:* $\text{inner } (a, b) (c, d) = \text{inner } a \ c + \text{inner } b \ d$
unfolding *inner_prod_def* **by** *simp*

instance

proof

fix $r :: \text{real}$

fix $x \ y \ z :: 'a::\text{real_inner} \times 'b::\text{real_inner}$

show $\text{inner } x \ y = \text{inner } y \ x$

unfolding *inner_prod_def*

by (*simp add: inner_commute*)

show $\text{inner } (x + y) \ z = \text{inner } x \ z + \text{inner } y \ z$

unfolding *inner_prod_def*

by (*simp add: inner_add_left*)

show $\text{inner } (\text{scaleR } r \ x) \ y = r * \text{inner } x \ y$

unfolding *inner_prod_def*

by (*simp add: distrib_left*)

show $0 \leq \text{inner } x \ x$

unfolding *inner_prod_def*

by (*intro add_nonneg_nonneg inner_ge_zero*)

show $\text{inner } x \ x = 0 \longleftrightarrow x = 0$

unfolding *inner_prod_def prod_eq_iff*

by (*simp add: add_nonneg_eq_0_iff*)

show $\text{norm } x = \text{sqrt } (\text{inner } x \ x)$

unfolding *norm_prod_def inner_prod_def*

by (*simp add: power2_norm_eq_inner*)

qed

end

lemma *inner_Pair_0:* $\text{inner } x \ (0, b) = \text{inner } (\text{snd } x) \ b$ $\text{inner } x \ (a, 0) = \text{inner } (\text{fst } x) \ a$
by (*cases x, simp*)**+**

instantiation *prod :: (euclidean_space, euclidean_space) euclidean_space*

begin

definition

$Basis = (\lambda u. (u, 0)) \text{ ' } Basis \cup (\lambda v. (0, v)) \text{ ' } Basis$

lemma *sum_Basis_prod_eq*:

fixes $f :: ('a * 'b) \Rightarrow ('a * 'b)$

shows $sum\ f\ Basis = sum\ (\lambda i. f\ (i, 0))\ Basis + sum\ (\lambda i. f\ (0, i))\ Basis$

proof –

have *inj_on* $(\lambda u. (u :: 'a, 0 :: 'b))\ Basis$ *inj_on* $(\lambda u. (0 :: 'a, u :: 'b))\ Basis$

by (*auto intro!*: *inj_onI Pair_inject*)

thus *?thesis*

unfolding *Basis_prod_def*

by (*subst sum.union_disjoint*) (*auto simp: Basis_prod_def sum.reindex*)

qed

instance proof

show $(Basis :: ('a \times 'b)\ set) \neq \{\}$

unfolding *Basis_prod_def* **by** *simp*

next

show *finite* $(Basis :: ('a \times 'b)\ set)$

unfolding *Basis_prod_def* **by** *simp*

next

fix $u\ v :: 'a \times 'b$

assume $u \in Basis$ **and** $v \in Basis$

thus *inner* $u\ v = (if\ u = v\ then\ 1\ else\ 0)$

unfolding *Basis_prod_def inner_prod_def*

by (*auto simp add: inner_Basis split: if_split_asm*)

next

fix $x :: 'a \times 'b$

show $(\forall u \in Basis. inner\ x\ u = 0) \longleftrightarrow x = 0$

unfolding *Basis_prod_def ball_Un ball_simps*

by (*simp add: inner_prod_def prod_eq_iff euclidean_all_zero_iff*)

qed

lemma *DIM_prod[simp]*: $DIM('a \times 'b) = DIM('a) + DIM('b)$

unfolding *Basis_prod_def*

by (*subst card_Un_disjoint*) (*auto intro!: card_image arg_cong2[where f=(+)] inj_onI*)

end

1.4.5 Locale instances

lemma *finite_dimensional_vector_space_euclidean*:

finite_dimensional_vector_space $(*_{\mathbb{R}})\ Basis$

proof *unfold locales*

show *finite* $(Basis :: 'a\ set)$ **by** (*metis finite_Basis*)

show *real_vector.independent* $(Basis :: 'a\ set)$

```

    unfolding dependent_def dependent_raw_def[symmetric]
    apply (subst span_finite)
    apply simp
    apply clarify
    apply (drule_tac f=inner a in arg_cong)
    apply (simp add: inner_Basis inner_sum_right eq_commute)
    done
  show module.span (*R) Basis = UNIV
    unfolding span_finite [OF finite_Basis] span_raw_def[symmetric]
    by (auto intro!: euclidean_representation[symmetric])
qed

interpretation eucl?: finite_dimensional_vector_space scaleR :: real => 'a =>
'a::euclidean_space Basis
  rewrites module.dependent (*R) = dependent
    and module.representation (*R) = representation
    and module.subspace (*R) = subspace
    and module.span (*R) = span
    and vector_space.extend_basis (*R) = extend_basis
    and vector_space.dim (*R) = dim
    and Vector_Spaces.linear (*R) (*R) = linear
    and Vector_Spaces.linear (*) (*R) = linear
    and finite_dimensional_vector_space.dimension Basis = DIM('a)
    and dimension = DIM('a)
  by (auto simp add: dependent_raw_def representation_raw_def
    subspace_raw_def span_raw_def extend_basis_raw_def dim_raw_def lin-
ear_def
    real_scaleR_def[abs_def]
    finite_dimensional_vector_space.dimension_def
    intro!: finite_dimensional_vector_space.dimension_def
    finite_dimensional_vector_space_euclidean)

interpretation eucl?: finite_dimensional_vector_space_pair_1
scaleR::real=>'a::euclidean_space=>'a Basis
scaleR::real=>'b::real_vector => 'b
by unfold_locales

interpretation eucl?: finite_dimensional_vector_space_prod scaleR scaleR Basis
Basis
  rewrites Basis_pair = Basis
    and module_prod.scale (*R) (*R) = (scaleR::_=>_=>('a × 'b))
proof –
  show finite_dimensional_vector_space_prod (*R) (*R) Basis Basis
    by unfold_locales
  interpret finite_dimensional_vector_space_prod (*R) (*R) Basis::'a set Ba-
sis::'b set
    by fact
  show Basis_pair = Basis
    unfolding Basis_pair_def Basis_prod_def by auto

```

```

  show module_prod.scale (*R) (*R) = scaleR
    by (fact module_prod_scale_eq_scaleR)
qed
end

```

1.5 Elementary Linear Algebra on Euclidean Spaces

```

theory Linear_Algebra
imports
  Euclidean_Space
  HOL-Library.Infinite_Set
begin

```

```

lemma linear_simps:
  assumes bounded_linear f
  shows
    f (a + b) = f a + f b
    f (a - b) = f a - f b
    f 0 = 0
    f (- a) = - f a
    f (s *R v) = s *R (f v)

```

```

proof -
  interpret f: bounded_linear f by fact
  show f (a + b) = f a + f b by (rule f.add)
  show f (a - b) = f a - f b by (rule f.diff)
  show f 0 = 0 by (rule f.zero)
  show f (- a) = - f a by (rule f.neg)
  show f (s *R v) = s *R (f v) by (rule f.scale)
qed

```

```

lemma finite_Atleast_Atmost_nat[simp]: finite {f x | x. x ∈ (UNIV::'a::finite set)}
  using finite_finite_image_set by blast

```

```

lemma substdbasis_expansion_unique:
  includes inner_syntax
  assumes d: d ⊆ Basis
  shows (∑ i ∈ d. f i *R i) = (x::'a::euclidean_space) ⟷
    (∀ i ∈ Basis. (i ∈ d ⟶ f i = x · i) ∧ (i ∉ d ⟶ x · i = 0))
proof -
  have *: ∀ x a b P. x * (if P then a else b) = (if P then x * a else x * b)
    by auto
  have **: finite d
    by (auto intro: finite_subset[OF assms])
  have ***: ∀ i. i ∈ Basis ⟹ (∑ i ∈ d. f i *R i) · i = (∑ x ∈ d. if x = i then f x
    else 0)
    using d
    by (auto intro!: sum.cong simp: inner_Basis inner_sum_left)
  show ?thesis

```

```

    unfolding euclidean_eq_iff[where 'a='a] by (auto simp: sum.delta[OF **]
    ***)
qed

```

```

lemma independent_substbasis:  $d \subseteq \text{Basis} \implies \text{independent } d$ 
  by (rule independent_mono[OF independent_Basis])

```

```

lemma subset_translation_eq [simp]:
  fixes  $a :: 'a::\text{real\_vector}$  shows  $(+) a \text{ ` } s \subseteq (+) a \text{ ` } t \iff s \subseteq t$ 
  by auto

```

```

lemma translate_inj_on:
  fixes  $A :: 'a::\text{ab\_group\_add}$  set
  shows inj_on  $(\lambda x. a + x) A$ 
  unfolding inj_on_def by auto

```

```

lemma translation_assoc:
  fixes  $a b :: 'a::\text{ab\_group\_add}$ 
  shows  $(\lambda x. b + x) \text{ ` } ((\lambda x. a + x) \text{ ` } S) = (\lambda x. (a + b) + x) \text{ ` } S$ 
  by auto

```

```

lemma translation_invert:
  fixes  $a :: 'a::\text{ab\_group\_add}$ 
  assumes  $(\lambda x. a + x) \text{ ` } A = (\lambda x. a + x) \text{ ` } B$ 
  shows  $A = B$ 
  using assms translation_assoc by fastforce

```

```

lemma translation_galois:
  fixes  $a :: 'a::\text{ab\_group\_add}$ 
  shows  $T = ((\lambda x. a + x) \text{ ` } S) \iff S = ((\lambda x. (- a) + x) \text{ ` } T)$ 
  by (metis add.right_inverse_group_cancel.rule0 translation_invert translation_assoc)

```

```

lemma translation_inverse_subset:
  assumes  $(\lambda x. - a + x) \text{ ` } V \subseteq (S :: 'n::\text{ab\_group\_add}$  set)
  shows  $V \subseteq ((\lambda x. a + x) \text{ ` } S)$ 
  by (metis assms subset_image_iff translation_galois)

```

1.5.1 More interesting properties of the norm

unbundle *inner_syntax*

Equality of vectors in terms of (\cdot) products.

```

lemma linear_componentwise:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_inner}$ 
  assumes  $lf: \text{linear } f$ 
  shows  $(f x) \cdot j = (\sum_{i \in \text{Basis}} (x \cdot i) * (f i \cdot j))$  (is ?lhs = ?rhs)
proof -
  interpret linear f by fact
  have ?rhs =  $(\sum_{i \in \text{Basis}} (x \cdot i) *_R (f i)) \cdot j$ 

```



```

  by (simp add: inner_sum_left)
  then show ?thesis
  by (simp add: euclidean_representation sum[symmetric] scale[symmetric])
qed

```

```

lemma vector_eq:  $x = y \iff x \cdot x = x \cdot y \wedge y \cdot y = x \cdot x$ 
  by (metis (no_types, opaque_lifting) inner_commute inner_diff_right inner_eq_zero_iff
  right_minus_eq)

```

```

lemma norm_triangle_half_r:
   $norm (y - x1) < e/2 \implies norm (y - x2) < e/2 \implies norm (x1 - x2) < e$ 
  using dist_triangle_half_r unfolding dist_norm[symmetric] by auto

```

```

lemma norm_triangle_half_l:
  assumes  $norm (x - y) < e/2$  and  $norm (x' - y) < e/2$ 
  shows  $norm (x - x') < e$ 
  by (metis assms dist_norm dist_triangle_half_l)

```

```

lemma abs_triangle_half_r:
  fixes  $y :: 'a::linordered\_field$ 
  shows  $abs (y - x1) < e/2 \implies abs (y - x2) < e/2 \implies abs (x1 - x2) < e$ 
  by linarith

```

```

lemma abs_triangle_half_l:
  fixes  $y :: 'a::linordered\_field$ 
  assumes  $abs (x - y) < e/2$  and  $abs (x' - y) < e/2$ 
  shows  $abs (x - x') < e$ 
  using assms by linarith

```

```

lemma sum_clauses:
  shows  $sum f \{\} = 0$ 
  and  $finite S \implies sum f (insert x S) = (if x \in S then sum f S else f x + sum f S)$ 
  by (auto simp add: insert_absorb)

```

```

lemma vector_eq_ldot:  $(\forall x. x \cdot y = x \cdot z) \iff y = z$  and vector_eq_rdot:  $(\forall z. x \cdot z = y \cdot z) \iff x = y$ 
  by (metis inner_commute vector_eq)+

```

1.5.2 Substandard Basis

```

lemma ex_card:
  assumes  $n \leq card A$ 
  shows  $\exists S \subseteq A. card S = n$ 
  by (meson assms obtain_subset_with_card_n)

```

```

lemma subspace_substandard:  $subspace \{x::'a::euclidean\_space. (\forall i \in Basis. P i \implies x \cdot i = 0)\}$ 
  by (auto simp: subspace_def inner_add_left)

```

lemma *dim_substandard*:
assumes $d: d \subseteq \text{Basis}$
shows $\dim \{x::'a::\text{euclidean_space}. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} = \text{card } d$ (**is**
 $\dim ?A = _$)
proof (*rule dim_unique*)
from d **show** $d \subseteq ?A$
by (*auto simp: inner_Basis*)
from d **show** *independent* d
by (*rule independent_mono [OF independent_Basis]*)
have $x \in \text{span } d$ **if** $\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0$ **for** x
proof –
have *finite* d
by (*rule finite_subset [OF d finite_Basis]*)
then **have** $(\sum i \in d. (x \cdot i) *_{\mathbb{R}} i) \in \text{span } d$
by (*simp add: span_sum span_clauses*)
also **have** $(\sum i \in d. (x \cdot i) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (x \cdot i) *_{\mathbb{R}} i)$
by (*rule sum.mono_neutral_cong_left [OF finite_Basis d]*) (*auto simp: that*)
finally **show** $x \in \text{span } d$
by (*simp only: euclidean_representation*)
qed
then **show** $?A \subseteq \text{span } d$ **by** *auto*
qed *simp*

1.5.3 Orthogonality

definition (*in real_inner*) *orthogonal* $x y \longleftrightarrow x \cdot y = 0$

context *real_inner*
begin

lemma *orthogonal_self*: *orthogonal* $x x \longleftrightarrow x = 0$
by (*simp add: orthogonal_def*)

lemma *orthogonal_clauses*:
orthogonal $a 0$
orthogonal $a x \implies \text{orthogonal } a (c *_{\mathbb{R}} x)$
orthogonal $a x \implies \text{orthogonal } a (-x)$
orthogonal $a x \implies \text{orthogonal } a y \implies \text{orthogonal } a (x + y)$
orthogonal $a x \implies \text{orthogonal } a y \implies \text{orthogonal } a (x - y)$
orthogonal $0 a$
orthogonal $x a \implies \text{orthogonal } (c *_{\mathbb{R}} x) a$
orthogonal $x a \implies \text{orthogonal } (-x) a$
orthogonal $x a \implies \text{orthogonal } y a \implies \text{orthogonal } (x + y) a$
orthogonal $x a \implies \text{orthogonal } y a \implies \text{orthogonal } (x - y) a$
unfolding *orthogonal_def inner_add inner_diff* **by** *auto*

end

lemma *orthogonal_commute*: $\text{orthogonal } x \ y \longleftrightarrow \text{orthogonal } y \ x$
by (*simp add: orthogonal_def inner_commute*)

lemma *orthogonal_scaleR* [*simp*]: $c \neq 0 \implies \text{orthogonal } (c *_{\mathbb{R}} x) = \text{orthogonal } x$
by (*rule ext*) (*simp add: orthogonal_def*)

lemma *pairwise_ortho_scaleR*:
 $\text{pairwise } (\lambda i \ j. \text{orthogonal } (f \ i) \ (g \ j)) \ B$
 $\implies \text{pairwise } (\lambda i \ j. \text{orthogonal } (a \ i *_{\mathbb{R}} f \ i) \ (a \ j *_{\mathbb{R}} g \ j)) \ B$
by (*auto simp: pairwise_def orthogonal_clauses*)

lemma *orthogonal_rvsum*:
 $\llbracket \text{finite } s; \bigwedge y. y \in s \implies \text{orthogonal } x \ (f \ y) \rrbracket \implies \text{orthogonal } x \ (\text{sum } f \ s)$
by (*induction s rule: finite_induct*) (*auto simp: orthogonal_clauses*)

lemma *orthogonal_lvsum*:
 $\llbracket \text{finite } s; \bigwedge x. x \in s \implies \text{orthogonal } (f \ x) \ y \rrbracket \implies \text{orthogonal } (\text{sum } f \ s) \ y$
by (*induction s rule: finite_induct*) (*auto simp: orthogonal_clauses*)

lemma *norm_add_Pythagorean*:
assumes *orthogonal a b*
shows $(\text{norm } (a + b))^2 = (\text{norm } a)^2 + (\text{norm } b)^2$
proof –
from *assms* **have** $(a - (0 - b)) \cdot (a - (0 - b)) = a \cdot a - (0 - b \cdot b)$
by (*simp add: algebra_simps orthogonal_def inner_commute*)
then show *?thesis*
by (*simp add: power2_norm_eq_inner*)
qed

lemma *norm_sum_Pythagorean*:
assumes *finite I pairwise* $(\lambda i \ j. \text{orthogonal } (f \ i) \ (f \ j)) \ I$
shows $(\text{norm } (\text{sum } f \ I))^2 = (\sum i \in I. (\text{norm } (f \ i))^2)$
using *assms*
proof (*induction I rule: finite_induct*)
case empty **then show** *?case* **by** *simp*
next
case (*insert x I*)
then have $\text{orthogonal } (f \ x) \ (\text{sum } f \ I)$
by (*metis pairwise_insert orthogonal_rvsum*)
with *insert* **show** *?case*
by (*simp add: pairwise_insert norm_add_Pythagorean*)
qed

1.5.4 Orthogonality of a transformation

definition *orthogonal_transformation* $f \longleftrightarrow \text{linear } f \wedge (\forall v \ w. f \ v \cdot f \ w = v \cdot w)$

lemma *orthogonal_transformation*:
 $\text{orthogonal_transformation } f \longleftrightarrow \text{linear } f \wedge (\forall v. \text{norm } (f \ v) = \text{norm } v)$

by (*smt* (*verit*, *ccfv_threshold*) *dot_norm linear_add norm_eq_sqrt_inner orthogonal_transformation_def*)

lemma *orthogonal_transformation_id* [*simp*]: *orthogonal_transformation* ($\lambda x. x$)
by (*simp add: linear_iff orthogonal_transformation_def*)

lemma *orthogonal_orthogonal_transformation*:
orthogonal_transformation $f \implies \text{orthogonal } (f\ x) (f\ y) \iff \text{orthogonal } x\ y$
by (*simp add: orthogonal_def orthogonal_transformation_def*)

lemma *orthogonal_transformation_compose*:
 $\llbracket \text{orthogonal_transformation } f; \text{orthogonal_transformation } g \rrbracket \implies \text{orthogonal_transformation}(f \circ g)$
by (*auto simp: orthogonal_transformation_def linear_compose*)

lemma *orthogonal_transformation_neg*:
orthogonal_transformation($\lambda x. -(f\ x)$) $\iff \text{orthogonal_transformation } f$
by (*auto simp: orthogonal_transformation_def dest: linear_compose_neg*)

lemma *orthogonal_transformation_scaleR*: *orthogonal_transformation* $f \implies f\ (c *_{\mathbb{R}} v) = c *_{\mathbb{R}} f\ v$
by (*simp add: linear_iff orthogonal_transformation_def*)

lemma *orthogonal_transformation_linear*:
orthogonal_transformation $f \implies \text{linear } f$
by (*simp add: orthogonal_transformation_def*)

lemma *orthogonal_transformation_inj*:
orthogonal_transformation $f \implies \text{inj } f$
unfolding *orthogonal_transformation_def inj_on_def*
by (*metis vector_eq*)

lemma *orthogonal_transformation_surj*:
orthogonal_transformation $f \implies \text{surj } f$
for $f :: 'a::\text{euclidean_space} \Rightarrow 'a::\text{euclidean_space}$
by (*simp add: linear_injective_imp_surjective orthogonal_transformation_inj orthogonal_transformation_linear*)

lemma *orthogonal_transformation_bij*:
orthogonal_transformation $f \implies \text{bij } f$
for $f :: 'a::\text{euclidean_space} \Rightarrow 'a::\text{euclidean_space}$
by (*simp add: bij_def orthogonal_transformation_inj orthogonal_transformation_surj*)

lemma *orthogonal_transformation_inv*:
orthogonal_transformation $f \implies \text{orthogonal_transformation } (\text{inv } f)$
for $f :: 'a::\text{euclidean_space} \Rightarrow 'a::\text{euclidean_space}$
by (*metis (no_types, opaque_lifting) bijection.inv_right bijection_def inj_linear_imp_inv_linear orthogonal_transformation orthogonal_transformation_bij orthogonal_transformation_inj*)

lemma *orthogonal_transformation_norm*:
orthogonal_transformation $f \implies \text{norm } (f x) = \text{norm } x$
by (*metis orthogonal_transformation*)

1.5.5 Bilinear functions

definition

bilinear :: ('a::real_vector \Rightarrow 'b::real_vector \Rightarrow 'c::real_vector) \Rightarrow bool **where**
bilinear $f \iff (\forall x. \text{linear } (\lambda y. f x y)) \wedge (\forall y. \text{linear } (\lambda x. f x y))$

lemma *bilinear_ladd*: *bilinear* $h \implies h (x + y) z = h x z + h y z$
by (*simp add: bilinear_def linear_iff*)

lemma *bilinear_radd*: *bilinear* $h \implies h x (y + z) = h x y + h x z$
by (*simp add: bilinear_def linear_iff*)

lemma *bilinear_times*:

fixes $c::'a::\text{real_algebra}$ **shows** *bilinear* $(\lambda x y. c * x * y)$
by (*auto simp: bilinear_def distrib_left distrib_right intro!: linearI*)

lemma *bilinear_lmul*: *bilinear* $h \implies h (c *_R x) y = c *_R h x y$
by (*simp add: bilinear_def linear_iff*)

lemma *bilinear_rmul*: *bilinear* $h \implies h x (c *_R y) = c *_R h x y$
by (*simp add: bilinear_def linear_iff*)

lemma *bilinear_lneg*: *bilinear* $h \implies h (- x) y = - h x y$
by (*drule bilinear_lmul [of _ - 1] simp*)

lemma *bilinear_rneg*: *bilinear* $h \implies h x (- y) = - h x y$
by (*drule bilinear_rmul [of _ _ - 1] simp*)

lemma (*in ab_group_add*) *eq_add_iff*: $x = x + y \iff y = 0$
using *add_left_imp_eq*[*of x y 0*] **by** *auto*

lemma *bilinear_lzero*:

assumes *bilinear* h
shows $h 0 x = 0$
using *bilinear_ladd* [*OF assms, of 0 0 x*] **by** (*simp add: eq_add_iff field_simps*)

lemma *bilinear_rzero*:

assumes *bilinear* h
shows $h x 0 = 0$
using *bilinear_radd* [*OF assms, of x 0 0*] **by** (*simp add: eq_add_iff field_simps*)

lemma *bilinear_lsub*: *bilinear* $h \implies h (x - y) z = h x z - h y z$
using *bilinear_ladd* [*of h x - y*] **by** (*simp add: bilinear_lneg*)

lemma *bilinear_rsub*: *bilinear* $h \implies h z (x - y) = h z x - h z y$

using *bilinear_radd* [of $h _ x - y$] by (simp add: *bilinear_rneg*)

lemma *bilinear_sum*:

assumes *bilinear* h

shows $h (sum\ f\ S) (sum\ g\ T) = sum\ (\lambda(i,j). h (f\ i) (g\ j)) (S \times T)$

proof –

interpret l : linear $\lambda x. h\ x\ y$ for y using *assms* by (simp add: *bilinear_def*)

interpret r : linear $\lambda y. h\ x\ y$ for x using *assms* by (simp add: *bilinear_def*)

have $h (sum\ f\ S) (sum\ g\ T) = sum\ (\lambda x. h (f\ x) (sum\ g\ T))\ S$

by (simp add: *l.sum*)

also have $\dots = sum\ (\lambda x. sum\ (\lambda y. h (f\ x) (g\ y))\ T)\ S$

by (rule *sum.cong*) (simp_all add: *r.sum*)

finally show *?thesis*

unfolding *sum.cartesian_product* .

qed

1.5.6 Adjoints

definition *adjoint* :: $((a::real_inner) \Rightarrow (b::real_inner)) \Rightarrow 'b \Rightarrow 'a$ where
adjoint $f = (SOME\ f'. \forall x\ y. f\ x \cdot y = x \cdot f'\ y)$

lemma *adjoint_unique*:

assumes $\forall x\ y. inner (f\ x) y = inner\ x (g\ y)$

shows *adjoint* $f = g$

unfolding *adjoint_def*

proof (rule *some_equality*)

show $\forall x\ y. inner (f\ x) y = inner\ x (g\ y)$

by (rule *assms*)

next

fix h

assume $\forall x\ y. inner (f\ x) y = inner\ x (h\ y)$

then show $h = g$

by (*metis* *assms* *ext* *vector_eq_ldot*)

qed

TODO: The following lemmas about adjoints should hold for any Hilbert space (i.e. complete inner product space). (see https://en.wikipedia.org/wiki/Hermitian_adjoint)

lemma *adjoint_works*:

fixes $f :: 'n::euclidean_space \Rightarrow 'm::euclidean_space$

assumes lf : linear f

shows $x \cdot adjoint\ f\ y = f\ x \cdot y$

proof –

interpret linear f by *fact*

have $\forall y. \exists w. \forall x. f\ x \cdot y = x \cdot w$

proof (intro *allI* *exI*)

fix $y :: 'm$ and x

let $?w = (\sum i \in Basis. (f\ i \cdot y) *_{\mathbb{R}} i) :: 'n$

have $f\ x \cdot y = f\ (\sum i \in Basis. (x \cdot i) *_{\mathbb{R}} i) \cdot y$

```

    by (simp add: euclidean_representation)
  also have ... = ( $\sum_{i \in \text{Basis}} (x \cdot i) *_{\mathbb{R}} f i$ ) \cdot y
    by (simp add: sum_scale)
  finally show  $f x \cdot y = x \cdot ?w$ 
    by (simp add: inner_sum_left inner_sum_right mult.commute)
qed
then show ?thesis
  unfolding adjoint_def choice_iff
  by (intro someI2_ex[where  $Q = \lambda f'. x \cdot f' y = f x \cdot y$ ]) auto
qed

```

lemma *adjoint_clauses*:

```

  fixes  $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$ 
  assumes  $lf: \text{linear } f$ 
  shows  $x \cdot \text{adjoint } f y = f x \cdot y$ 
    and  $\text{adjoint } f y \cdot x = y \cdot f x$ 
  by (simp_all add: adjoint_works[OF lf] inner_commute)

```

lemma *adjoint_linear*:

```

  fixes  $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$ 
  assumes  $lf: \text{linear } f$ 
  shows  $\text{linear } (\text{adjoint } f)$ 
  by (simp add: lf linear_iff euclidean_eq_iff[where 'a='n] euclidean_eq_iff[where
'a='m]
    adjoint_clauses[OF lf] inner_distrib)

```

lemma *adjoint_adjoint*:

```

  fixes  $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$ 
  assumes  $lf: \text{linear } f$ 
  shows  $\text{adjoint } (\text{adjoint } f) = f$ 
  by (rule adjoint_unique, simp add: adjoint_clauses [OF lf])

```

1.5.7 Euclidean Spaces as Typeclass

lemma *independent_Basis*: *independent Basis*

```

  by (rule independent_Basis)

```

lemma *span_Basis* [simp]: *span Basis = UNIV*

```

  by (rule span_Basis)

```

lemma *in_span_Basis*: $x \in \text{span Basis}$

```

  unfolding span_Basis ..

```

1.5.8 Linearity and Bilinearity continued

lemma *linear_bounded*:

```

  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::real\_normed\_vector$ 
  assumes  $lf: \text{linear } f$ 
  shows  $\exists B. \forall x. \text{norm } (f x) \leq B * \text{norm } x$ 

```

proof

```

interpret linear f by fact
let ?B =  $\sum b \in \text{Basis}. \text{norm } (f b)$ 
show  $\forall x. \text{norm } (f x) \leq ?B * \text{norm } x$ 
proof
  fix x :: 'a
  let ?g =  $\lambda b. (x \cdot b) *_R f b$ 
  have  $\text{norm } (f x) = \text{norm } (f (\sum b \in \text{Basis}. (x \cdot b) *_R b))$ 
    unfolding euclidean_representation ..
  also have  $\dots = \text{norm } (\text{sum } ?g \text{ Basis})$ 
    by (simp add: sum_scale)
  finally have th0:  $\text{norm } (f x) = \text{norm } (\text{sum } ?g \text{ Basis})$  .
  have th:  $\text{norm } (?g i) \leq \text{norm } (f i) * \text{norm } x$  if  $i \in \text{Basis}$  for i
  proof -
    from Basis_le_norm[OF that, of x]
    show  $\text{norm } (?g i) \leq \text{norm } (f i) * \text{norm } x$ 
    unfolding norm_scaleR by (metis mult.commute mult_left_mono norm_ge_zero)
  qed
  from sum_norm_le[of _ ?g, OF th]
  show  $\text{norm } (f x) \leq ?B * \text{norm } x$ 
    by (simp add: sum_distrib_right th0)
  qed
qed

```

lemma *linear_conv_bounded_linear*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
shows linear f  $\iff$  bounded_linear f
by (metis mult.commute bounded_linear_axioms.intro bounded_linear_def linear_bounded)

```

lemmas *linear_linear = linear_conv_bounded_linear[symmetric]*

lemma *inj_linear_imp_inv_bounded_linear*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'a
shows  $\llbracket \text{bounded\_linear } f; \text{inj } f \rrbracket \implies \text{bounded\_linear } (\text{inv } f)$ 
by (simp add: inj_linear_imp_inv_linear linear_linear)

```

lemma *linear_bounded_pos*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
assumes lf: linear f
obtains B where  $B > 0 \wedge \forall x. \text{norm } (f x) \leq B * \text{norm } x$ 
by (metis bounded_linear.pos_bounded lf linear_linear mult.commute)

```

lemma *linear_invertible_bounded_below_pos*:

```

fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space
assumes linear f linear g and gf:  $g \circ f = \text{id}$ 
obtains B where  $B > 0 \wedge \forall x. B * \text{norm } x \leq \text{norm}(f x)$ 
proof -
  obtain B where  $B > 0$  and B:  $\forall x. \text{norm } (g x) \leq B * \text{norm } x$ 
    using linear_bounded_pos [OF  $\langle \text{linear } g \rangle$ ] by blast

```



```

show thesis
proof
  show  $0 < 1/B$ 
    by (simp add: <B > 0>)
  show  $1/B * \text{norm } x \leq \text{norm } (f x)$  for  $x$ 
    by (smt (verit, ccfv_SIG) B <0 < B> gf comp_apply divide_inverse id_apply
inverse_eq_divide
      less_divide_eq mult.commute)
qed
qed

```

```

lemma linear_inj_bounded_below_pos:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes linear f inj f
  obtains  $B$  where  $B > 0 \wedge x. B * \text{norm } x \leq \text{norm}(f x)$ 
  using linear_injective_left_inverse [OF assms]
  linear_invertible_bounded_below_pos assms by blast

```

```

lemma bounded_linearI':
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes  $\wedge x y. f (x + y) = f x + f y$ 
  and  $\wedge c x. f (c *_{\mathbb{R}} x) = c *_{\mathbb{R}} f x$ 
  shows bounded_linear f
  using assms linearI linear_conv_bounded_linear by blast

```

```

lemma bilinear_bounded:
  fixes  $h :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space} \Rightarrow 'k::\text{real\_normed\_vector}$ 
  assumes bh: bilinear h
  shows  $\exists B. \forall x y. \text{norm } (h x y) \leq B * \text{norm } x * \text{norm } y$ 
proof (clarify intro!: exI[of _ \sum i \in Basis. \sum j \in Basis. norm (h i j)])
  fix  $x :: 'm$ 
  fix  $y :: 'n$ 
  have  $\text{norm } (h x y) = \text{norm } (h (\text{sum } (\lambda i. (x \cdot i) *_{\mathbb{R}} i) \text{Basis}) (\text{sum } (\lambda i. (y \cdot i) *_{\mathbb{R}} i) \text{Basis}))$ 
  by (simp add: euclidean_representation)
  also have  $\dots = \text{norm } (\text{sum } (\lambda (i,j). h ((x \cdot i) *_{\mathbb{R}} i) ((y \cdot j) *_{\mathbb{R}} j)) (\text{Basis} \times \text{Basis}))$ 
  unfolding bilinear_sum [OF bh] ..
  finally have th: norm (h x y) = \dots ..
  have  $\wedge i j. \llbracket i \in \text{Basis}; j \in \text{Basis} \rrbracket$ 
     $\implies |x \cdot i| * (|y \cdot j| * \text{norm } (h i j)) \leq \text{norm } x * (\text{norm } y * \text{norm } (h i j))$ 
  by (auto simp add: zero_le_mult_iff Basis_le_norm mult_mono)
  then show  $\text{norm } (h x y) \leq (\sum i \in \text{Basis}. \sum j \in \text{Basis}. \text{norm } (h i j)) * \text{norm } x * \text{norm } y$ 
  unfolding sum_distrib_right th sum.cartesian_product
  by (clarsimp simp add: bilinear_rmul [OF bh] bilinear_lmul [OF bh]
    field_simps simp del: scaleR_scaleR intro!: sum_norm_le)
qed

```

```

lemma bilinear_conv_bounded_bilinear:
  fixes  $h :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space \Rightarrow 'c::real\_normed\_vector$ 
  shows  $bilinear\ h \longleftrightarrow bounded\_bilinear\ h$ 
proof
  assume bilinear h
  show bounded_bilinear h
  proof
    fix  $x\ y\ z$ 
    show  $h\ (x + y)\ z = h\ x\ z + h\ y\ z$ 
      using  $\langle bilinear\ h \rangle$  unfolding bilinear_def linear_iff by simp
    next
    fix  $x\ y\ z$ 
    show  $h\ x\ (y + z) = h\ x\ y + h\ x\ z$ 
      using  $\langle bilinear\ h \rangle$  unfolding bilinear_def linear_iff by simp
    next
    show  $h\ (scaleR\ r\ x)\ y = scaleR\ r\ (h\ x\ y)$   $h\ x\ (scaleR\ r\ y) = scaleR\ r\ (h\ x\ y)$ 
  for  $r\ x\ y$ 
    using  $\langle bilinear\ h \rangle$  unfolding bilinear_def linear_iff
    by simp_all
  next
  have  $\exists B. \forall x\ y. norm\ (h\ x\ y) \leq B * norm\ x * norm\ y$ 
    using  $\langle bilinear\ h \rangle$  by (rule bilinear_bounded)
  then show  $\exists K. \forall x\ y. norm\ (h\ x\ y) \leq norm\ x * norm\ y * K$ 
    by (simp add: ac_simps)
  qed
next
assume bounded_bilinear h
then interpret  $h: bounded\_bilinear\ h$  .
show bilinear h
  unfolding bilinear_def linear_conv_bounded_linear
  using  $h.bounded\_linear\_left\ h.bounded\_linear\_right$  by simp
qed

lemma bilinear_bounded_pos:
  fixes  $h :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space \Rightarrow 'c::real\_normed\_vector$ 
  assumes  $bh: bilinear\ h$ 
  shows  $\exists B > 0. \forall x\ y. norm\ (h\ x\ y) \leq B * norm\ x * norm\ y$ 
  by (metis mult.assoc bh bilinear_conv_bounded_bilinear bounded_bilinear.pos_bounded
mult.commute)

lemma bounded_linear_imp_has_derivative:
   $bounded\_linear\ f \implies (f\ has\_derivative\ f)\ net$ 
  by (auto simp add: has_derivative_def linear_diff linear_linear linear_def
dest: bounded_linear.linear)

lemma linear_imp_has_derivative:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::real\_normed\_vector$ 
  shows  $linear\ f \implies (f\ has\_derivative\ f)\ net$ 
  by (simp add: bounded_linear_imp_has_derivative linear_conv_bounded_linear)

```

lemma *bounded_linear_imp_differentiable*: *bounded_linear f \implies f differentiable net*

using *bounded_linear_imp_has_derivative differentiable_def* **by** *blast*

lemma *linear_imp_differentiable*:

fixes *f :: 'a::euclidean_space \Rightarrow 'b::real_normed_vector*

shows *linear f \implies f differentiable net*

by (*metis linear_imp_has_derivative differentiable_def*)

lemma *of_real_differentiable* [*simp, derivative_intros*]: *of_real differentiable F*

by (*simp add: bounded_linear_imp_differentiable bounded_linear_of_real*)

1.5.9 We continue

lemma *independent_bound*:

fixes *S :: 'a::euclidean_space set*

shows *independent S \implies finite S \wedge card S \leq DIM('a)*

by (*metis dim_subset_UNIV finiteI_independent dim_span_eq_card_independent*)

lemmas *independent_imp_finite = finiteI_independent*

corollary *independent_card_le*:

fixes *S :: 'a::euclidean_space set*

assumes *independent S*

shows *card S \leq DIM('a)*

using *assms independent_bound* **by** *auto*

lemma *dependent_biggerset*:

fixes *S :: 'a::euclidean_space set*

shows (*finite S \implies card S $>$ DIM('a)*) \implies *dependent S*

by (*metis independent_bound not_less*)

Picking an orthogonal replacement for a spanning set.

lemma *vector_sub_project_orthogonal*:

fixes *b x :: 'a::euclidean_space*

shows $b \cdot (x - ((b \cdot x) / (b \cdot b)) *_{\mathbb{R}} b) = 0$

unfolding *inner_simps* **by** *auto*

lemma *pairwise_orthogonal_insert*:

assumes *pairwise orthogonal S*

and $\bigwedge y. y \in S \implies$ *orthogonal x y*

shows *pairwise orthogonal (insert x S)*

using *assms* **by** (*auto simp: pairwise_def orthogonal_commute*)

lemma *basis_orthogonal*:

fixes *B :: 'a::real_inner set*

assumes *fB: finite B*

shows $\exists C. \text{finite } C \wedge \text{card } C \leq \text{card } B \wedge \text{span } C = \text{span } B \wedge \text{pairwise orthogonal}$

```

C
(is  $\exists C. ?P B C$ )
using fB
proof (induct rule: finite_induct)
  case empty
  then show ?case
    using pairwise_empty by blast
next
case (insert a B)
note fB =  $\langle \text{finite } B \rangle$  and aB =  $\langle a \notin B \rangle$ 
from  $\langle \exists C. \text{finite } C \wedge \text{card } C \leq \text{card } B \wedge \text{span } C = \text{span } B \wedge \text{pairwise orthogonal } C \rangle$ 
obtain C where C:  $\text{finite } C \wedge \text{card } C \leq \text{card } B$ 
  span C = span B pairwise orthogonal C by blast
let ?a =  $a - \text{sum } (\lambda x. (x \cdot a / (x \cdot x)) *_{\mathbb{R}} x) C$ 
let ?C = insert ?a C
from C(1) have fC:  $\text{finite } ?C$ 
  by simp
have cC:  $\text{card } ?C \leq \text{card } (\text{insert } a B)$ 
  using C aB card_insert_if local.insert(1) by fastforce
{
  fix x k
  have th0:  $\bigwedge (a::'a) b c. a - (b - c) = c + (a - b)$ 
    by (simp add: field_simps)
  have  $x - k *_{\mathbb{R}} (a - (\sum_{x \in C. (x \cdot a / (x \cdot x)) *_{\mathbb{R}} x)) \in \text{span } C \iff x - k *_{\mathbb{R}} a \in \text{span } C$ 
    unfolding scaleR_right_diff_distrib th0
    by (intro span_add_eq span_scale span_sum span_base)
}
then have SC:  $\text{span } ?C = \text{span } (\text{insert } a B)$ 
  unfolding set_eq_iff span_breakdown_eq C(3)[symmetric] by auto
{
  fix y
  assume yC:  $y \in C$ 
  then have Cy:  $C = \text{insert } y (C - \{y\})$ 
    by blast
  have fth:  $\text{finite } (C - \{y\})$ 
    using C by simp
  have  $y \neq 0 \implies \forall x \in C - \{y\}. x \cdot a * (x \cdot y) / (x \cdot x) = 0$ 
    using  $\langle \text{pairwise orthogonal } C \rangle$ 
    by (metis Cy DiffE div_0 insertCI mult_zero_right orthogonal_def pairwise_insert)
  then have orthogonal ?a y
    unfolding orthogonal_def
    unfolding inner_diff inner_sum_left right_minus_eq
    unfolding sum.remove [OF  $\langle \text{finite } C \rangle \langle y \in C \rangle$ ]
    by (auto simp add: sum.neutral inner_commute[of y a])
}
with  $\langle \text{pairwise orthogonal } C \rangle$  have CPO:  $\text{pairwise orthogonal } ?C$ 

```

```

  by (rule pairwise_orthogonal_insert)
  from fC cC SC CPO have ?P (insert a B) ?C
  by blast
  then show ?case by blast
qed

```

```

lemma orthogonal_basis_exists:
  fixes V :: ('a::euclidean_space) set
  shows  $\exists B. \text{independent } B \wedge B \subseteq \text{span } V \wedge V \subseteq \text{span } B \wedge (\text{card } B = \text{dim } V)$ 
 $\wedge \text{pairwise\_orthogonal } B$ 
proof -
  from basis_exists[of V] obtain B where
    B:  $B \subseteq V \text{ independent } B \wedge V \subseteq \text{span } B \wedge \text{card } B = \text{dim } V$ 
  by force
  from B have fB: finite B card B = dim V
  using independent_bound by auto
  from basis_orthogonal[OF fB(1)] obtain C where
    C: finite C card C  $\leq$  card B span C = span B pairwise_orthogonal C
  by blast
  from C B have CSV:  $C \subseteq \text{span } V$ 
  by (metis span_superset span_mono subset_trans)
  from span_mono[OF B(3)] C have SVC:  $\text{span } V \subseteq \text{span } C$ 
  by (simp add: span_span)
  from C fB have card C  $\leq$  dim V
  by simp
  moreover have dim V  $\leq$  card C
  using span_card_ge_dim[OF CSV SVC C(1)]
  by simp
  ultimately have card C = dim V
  using C(1) by simp
  with C B CSV show ?thesis
  by (metis SVC card_eq_dim dim_span)
qed

```

Low-dimensional subset is in a hyperplane (weak orthogonal complement).

```

lemma span_not_UNIV_orthogonal:
  fixes S :: ('a::euclidean_space) set
  assumes sU: span S  $\neq$  UNIV
  shows  $\exists a. 'a. a \neq 0 \wedge (\forall x \in \text{span } S. a \cdot x = 0)$ 
proof -
  from sU obtain a where a:  $a \notin \text{span } S$ 
  by blast
  from orthogonal_basis_exists obtain B where
    B: independent B  $B \subseteq \text{span } S \wedge S \subseteq \text{span } B \wedge \text{card } B = \text{dim } S \wedge \text{pairwise\_orthogonal } B$ 
  by blast
  from B have fB: finite B card B = dim S
  using independent_bound by auto
  have sSB: span S = span B

```

```

    by (simp add: B span_eq)
  let ?a = a - sum (λb. (a · b / (b · b)) *R b) B
  have sum (λb. (a · b / (b · b)) *R b) B ∈ span S
    by (simp add: sSB span_base span_mul span_sum)
  with a have a0: ?a ≠ 0
    by auto
  have ?a · x = 0 if x ∈ span B for x
  proof (rule span_induct [OF that])
    show subspace {x. ?a · x = 0}
      by (auto simp add: subspace_def inner_add)
  next
  {
    fix x
    assume x: x ∈ B
    from x have B': B = insert x (B - {x})
      by blast
    have fth: finite (B - {x})
      using fB by simp
    have (∑ b ∈ B - {x}. a · b * (b · x) / (b · b)) = 0 if x ≠ 0
      by (smt (verit) B' B(5) DiffD2 divide_eq_0_iff inner_real_def inner_zero_right
insertCI orthogonal_def pairwise_insert sum.neutral)
    then have ?a · x = 0
      apply (subst B')
      using fB fth
      unfolding sum_clauses(2)[OF fth]
      by (auto simp add: inner_add_left inner_diff_left inner_sum_left)
  }
  then show ?a · x = 0 if x ∈ B for x
    using that by blast
  qed
  with a0 sSB show ?thesis
    by blast
qed

```

lemma *span_not_univ_subset_hyperplane*:

```

  fixes S :: 'a::euclidean_space set
  assumes SU: span S ≠ UNIV
  shows ∃ a. a ≠ 0 ∧ span S ⊆ {x. a · x = 0}
  using span_not_UNIV_orthogonal[OF SU] by auto

```

lemma *lowdim_subset_hyperplane*:

```

  fixes S :: 'a::euclidean_space set
  assumes d: dim S < DIM('a)
  shows ∃ a: 'a. a ≠ 0 ∧ span S ⊆ {x. a · x = 0}
  using d dim_eq_full nless_le span_not_univ_subset_hyperplane by blast

```

lemma *linear_eq_stdbasis*:

```

  fixes f :: 'a::euclidean_space ⇒ _
  assumes lf: linear f

```

```

  and lg: linear g
  and fg:  $\bigwedge b. b \in \text{Basis} \implies f b = g b$ 
shows  $f = g$ 
using linear_eq_on_span[OF lf lg, of Basis] fg by auto

```

Similar results for bilinear functions.

```

lemma bilinear_eq:
  assumes bf: bilinear f
  and bg: bilinear g
  and SB:  $S \subseteq \text{span } B$ 
  and TC:  $T \subseteq \text{span } C$ 
  and xS yT:  $x \in S \implies y \in T$ 
  and fg:  $\bigwedge x y. [x \in B; y \in C] \implies f x y = g x y$ 
shows  $f x y = g x y$ 
proof -
  let ?P =  $\{x. \forall y \in \text{span } C. f x y = g x y\}$ 
  from bf bg have sp: subspace ?P
  unfolding bilinear_def linear_iff subspace_def bf bg
  by (auto simp add: span_zero bilinear_lzero[OF bf] bilinear_lzero[OF bg]
      span_add Ball_def
      intro: bilinear_ladd[OF bf])
  have sfg:  $\bigwedge x. x \in B \implies \text{subspace } \{a. f x a = g x a\}$ 
  by (auto simp: subspace_def bf bg bilinear_rzero bilinear_radd bilinear_rmul)
  have  $\forall y \in \text{span } C. f x y = g x y$  if  $x \in \text{span } B$  for x
  using span_induct [OF that sp] fg sfg span_induct by blast
  then show ?thesis
  using SB TC assms by auto
qed

```

```

lemma bilinear_eq_stdbasis:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  _
  assumes bf: bilinear f
  and bg: bilinear g
  and fg:  $\bigwedge i j. i \in \text{Basis} \implies j \in \text{Basis} \implies f i j = g i j$ 
shows  $f = g$ 
using bilinear_eq[OF bf bg equalityD2[OF span_Basis] equalityD2[OF span_Basis]]
fg by blast

```

1.5.10 Infinity norm

definition $\text{infnorm } (x :: 'a::\text{euclidean_space}) = \text{Sup } \{|x \cdot b| \mid b \in \text{Basis}\}$

```

lemma infnorm_set_image:
  fixes x :: 'a::euclidean_space
  shows  $\{|x \cdot i| \mid i \in \text{Basis}\} = (\lambda i. |x \cdot i|) ` \text{Basis}$ 
  by blast

```

```

lemma infnorm_Max:
  fixes x :: 'a::euclidean_space

```

shows $\text{infnorm } x = \text{Max } ((\lambda i. |x \cdot i|) \text{ ` } \text{Basis})$
by (*simp add: infnorm_def infnorm_set_image cSup_eq_Max*)

lemma *infnorm_set_lemma*:
fixes $x :: 'a::\text{euclidean_space}$
shows $\text{finite } \{|x \cdot i| \mid i. i \in \text{Basis}\}$
and $\{|x \cdot i| \mid i. i \in \text{Basis}\} \neq \{\}$
unfolding *infnorm_set_image* **by** *auto*

lemma *infnorm_pos_le*:
fixes $x :: 'a::\text{euclidean_space}$
shows $0 \leq \text{infnorm } x$
by (*simp add: infnorm_Max Max_ge_iff ex_in_conv*)

lemma *infnorm_triangle*:
fixes $x :: 'a::\text{euclidean_space}$
shows $\text{infnorm } (x + y) \leq \text{infnorm } x + \text{infnorm } y$
proof –
have $*$: $\bigwedge a b c d :: \text{real}. |a| \leq c \implies |b| \leq d \implies |a + b| \leq c + d$
by *simp*
show *?thesis*
by (*auto simp: infnorm_Max inner_add_left intro!: **)
qed

lemma *infnorm_eq_0*:
fixes $x :: 'a::\text{euclidean_space}$
shows $\text{infnorm } x = 0 \iff x = 0$
proof –
have $\text{infnorm } x \leq 0 \iff x = 0$
unfolding *infnorm_Max* **by** (*simp add: euclidean_all_zero_iff*)
then show *?thesis*
using *infnorm_pos_le[of x]* **by** *simp*
qed

lemma *infnorm_0*: $\text{infnorm } 0 = 0$
by (*simp add: infnorm_eq_0*)

lemma *infnorm_neg*: $\text{infnorm } (-x) = \text{infnorm } x$
unfolding *infnorm_def* **by** *simp*

lemma *infnorm_sub*: $\text{infnorm } (x - y) = \text{infnorm } (y - x)$
by (*metis infnorm_neg minus_diff_eq*)

lemma *absdiff_infnorm*: $|\text{infnorm } x - \text{infnorm } y| \leq \text{infnorm } (x - y)$
by (*smt (verit, del_insts) diff_add_cancel infnorm_sub infnorm_triangle*)

lemma *real_abs_infnorm*: $|\text{infnorm } x| = \text{infnorm } x$
using *infnorm_pos_le[of x]* **by** *arith*

lemma *Basis_le_infnorm*:

fixes $x :: 'a::euclidean_space$
shows $b \in \text{Basis} \implies |x \cdot b| \leq \text{infnorm } x$
by (*simp add: infnorm_Max*)

lemma *infnorm_mul*: $\text{infnorm } (a *_R x) = |a| * \text{infnorm } x$

unfolding *infnorm_Max*

proof (*safe intro!: Max_eqI*)

let $?B = (\lambda i. |x \cdot i|) \text{ `Basis}$

{ fix $b :: 'a$

assume $b \in \text{Basis}$

then show $|a *_R x \cdot b| \leq |a| * \text{Max } ?B$

by (*simp add: abs_mult mult_left_mono*)

next

from *Max_in[of ?B]* **obtain** b **where** $b \in \text{Basis}$ $\text{Max } ?B = |x \cdot b|$

by (*auto simp del: Max_in*)

then show $|a| * \text{Max } ((\lambda i. |x \cdot i|) \text{ `Basis}) \in (\lambda i. |a *_R x \cdot i|) \text{ `Basis}$

by (*intro image_eqI[where x=b]*) (*auto simp: abs_mult*)

}

qed *simp*

lemma *infnorm_mul_lemma*: $\text{infnorm } (a *_R x) \leq |a| * \text{infnorm } x$

unfolding *infnorm_mul ..*

lemma *infnorm_pos_lt*: $\text{infnorm } x > 0 \iff x \neq 0$

using *infnorm_pos_le[of x] infnorm_eq_0[of x]* **by** *arith*

Prove that it differs only up to a bound from Euclidean norm.

lemma *infnorm_le_norm*: $\text{infnorm } x \leq \text{norm } x$

by (*simp add: Basis_le_norm infnorm_Max*)

lemma *norm_le_infnorm*:

fixes $x :: 'a::euclidean_space$

shows $\text{norm } x \leq \text{sqrt } \text{DIM}('a) * \text{infnorm } x$

unfolding *norm_eq_sqrt_inner id_def*

proof (*rule real_le_sqrt*)

show $\text{sqrt } \text{DIM}('a) * \text{infnorm } x \geq 0$

by (*simp add: zero_le_mult_iff infnorm_pos_le*)

have $x \cdot x \leq (\sum_{b \in \text{Basis}} x \cdot b * (x \cdot b))$

by (*metis euclidean_inner_order_refl*)

also have $\dots \leq \text{DIM}('a) * |\text{infnorm } x|^2$

by (*rule sum_bounded_above*) (*metis Basis_le_infnorm abs_le_square_iff power2_eq_square real_abs_infnorm*)

also have $\dots \leq (\text{sqrt } \text{DIM}('a) * \text{infnorm } x)^2$

by (*simp add: power_mult_distrib*)

finally show $x \cdot x \leq (\text{sqrt } \text{DIM}('a) * \text{infnorm } x)^2$.

qed

lemma *tendsto_infnorm [tendsto_intros]*:

```

assumes (f  $\longrightarrow$  a) F
shows (( $\lambda x$ . infnorm (f x))  $\longrightarrow$  infnorm a) F
proof (rule tendsto_compose [OF LIM_I assms])
  fix r :: real
  assume r > 0
  then show  $\exists s > 0. \forall x. x \neq a \wedge \text{norm } (x - a) < s \longrightarrow \text{norm } (\text{infnorm } x - \text{infnorm } a) < r$ 
    by (metis real_norm_def le_less_trans absdiff_infnorm infnorm_le_norm)
qed

```

Equality in Cauchy-Schwarz and triangle inequalities.

```

lemma norm_cauchy_schwarz_eq:  $x \cdot y = \text{norm } x * \text{norm } y \iff \text{norm } x *_R y = \text{norm } y *_R x$ 
  (is ?lhs  $\iff$  ?rhs)
proof (cases x=0)
  case True
  then show ?thesis
    by auto
  next
  case False
  from inner_eq_zero_iff[of norm y *_R x - norm x *_R y]
  have ?rhs  $\iff$ 
    (norm y * (norm y * norm x * norm x - norm x * (x · y)) -
     norm x * (norm y * (y · x) - norm x * norm y * norm y) = 0)
  using False unfolding inner_simps
  by (auto simp add: power2_norm_eq_inner[symmetric] power2_eq_square
    inner_commute field_simps)
  also have ...  $\iff$  (2 * norm x * norm y * (norm x * norm y - x · y) = 0)
  using False by (simp add: field_simps inner_commute)
  also have ...  $\iff$  ?lhs
  using False by auto
  finally show ?thesis by metis
qed

```

```

lemma norm_cauchy_schwarz_abs_eq:
   $|x \cdot y| = \text{norm } x * \text{norm } y \iff$ 
   $\text{norm } x *_R y = \text{norm } y *_R x \vee \text{norm } x *_R y = - \text{norm } y *_R x$ 
  using norm_cauchy_schwarz_eq [symmetric, of x y]
  using norm_cauchy_schwarz_eq [symmetric, of -x y] Cauchy_Schwarz_ineq2
  [of x y]
  by auto

```

```

lemma norm_triangle_eq:
  fixes x y :: 'a::real_inner
  shows  $\text{norm } (x + y) = \text{norm } x + \text{norm } y \iff \text{norm } x *_R y = \text{norm } y *_R x$ 
proof (cases x = 0  $\vee$  y = 0)
  case True
  then show ?thesis
    by force

```

```

next
  case False
  then have n:  $\text{norm } x > 0 \text{ norm } y > 0$ 
    by auto
  have  $\text{norm } (x + y) = \text{norm } x + \text{norm } y \iff (\text{norm } (x + y))^2 = (\text{norm } x + \text{norm } y)^2$ 
    by simp
  also have  $\dots \iff \text{norm } x *_{\mathbb{R}} y = \text{norm } y *_{\mathbb{R}} x$ 
    by (smt (verit, best) dot_norm_inner_real_def inner_simps norm_cauchy_schwarz_eq power2_eq_square)
  finally show ?thesis .
qed

```

```

lemma dist_triangle_eq:
  fixes x y z :: 'a::real_inner'
  shows  $\text{dist } x z = \text{dist } x y + \text{dist } y z \iff$ 
     $\text{norm } (x - y) *_{\mathbb{R}} (y - z) = \text{norm } (y - z) *_{\mathbb{R}} (x - y)$ 
  by (metis (no_types, lifting) add_diff_eq diff_add_cancel dist_norm norm_triangle_eq)

```

1.5.11 Collinearity

```

definition collinear :: 'a::real_vector set  $\Rightarrow$  bool'
  where  $\text{collinear } S \iff (\exists u. \forall x \in S. \forall y \in S. \exists c. x - y = c *_{\mathbb{R}} u)$ 

```

```

lemma collinear_alt:
   $\text{collinear } S \iff (\exists u v. \forall x \in S. \exists c. x = u + c *_{\mathbb{R}} v)$  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    unfolding collinear_def by (metis add commute diff_add_cancel)
next
  assume ?rhs
  then obtain u v where  $\forall x. x \in S \implies \exists c. x = u + c *_{\mathbb{R}} v$ 
    by auto
  have  $\exists c. x - y = c *_{\mathbb{R}} v$  if  $x \in S$   $y \in S$  for x y
    by (metis  $*[OF \langle x \in S \rangle] * [OF \langle y \in S \rangle]$  scaleR_left diff_add_diff_cancel_left)
  then show ?lhs
    using collinear_def by blast
qed

```

```

lemma collinear:
  fixes S :: 'a::{perfect_space, real_vector} set'
  shows  $\text{collinear } S \iff (\exists u. u \neq 0 \wedge (\forall x \in S. \forall y \in S. \exists c. x - y = c *_{\mathbb{R}} u))$ 
proof -
  have  $\exists v. v \neq 0 \wedge (\forall x \in S. \forall y \in S. \exists c. x - y = c *_{\mathbb{R}} v)$ 
    if  $\forall x \in S. \forall y \in S. \exists c. x - y = c *_{\mathbb{R}} u$  u = 0 for u
  proof -
    have  $\forall x \in S. \forall y \in S. x = y$ 
      using that by auto

```

```

moreover
obtain  $v::'a$  where  $v \neq 0$ 
  using UNIV_not_singleton [of 0] by auto
ultimately have  $\forall x \in S. \forall y \in S. \exists c. x - y = c *_R v$ 
  by auto
then show ?thesis
  using  $\langle v \neq 0 \rangle$  by blast
qed
then show ?thesis
  by (metis collinear_def)
qed

```

```

lemma collinear_subset:  $\llbracket \text{collinear } T; S \subseteq T \rrbracket \implies \text{collinear } S$ 
  by (meson collinear_def subsetCE)

```

```

lemma collinear_empty [iff]:  $\text{collinear } \{\}$ 
  by (simp add: collinear_def)

```

```

lemma collinear_sing [iff]:  $\text{collinear } \{x\}$ 
  by (simp add: collinear_def)

```

```

lemma collinear_2 [iff]:  $\text{collinear } \{x, y\}$ 
  by (simp add: collinear_def) (metis minus_diff_eq scaleR_left.minus scaleR_one)

```

```

lemma collinear_lemma:  $\text{collinear } \{0, x, y\} \longleftrightarrow x = 0 \vee y = 0 \vee (\exists c. y = c *_R x)$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)

```

```

proof (cases  $x = 0 \vee y = 0$ )
  case True

```

```

    then show ?thesis
      by (auto simp: insert_commute)

```

```

next

```

```

  case False

```

```

    show ?thesis

```

```

    proof

```

```

      assume  $h: ?lhs$ 

```

```

      then obtain  $u$  where  $u: \forall x \in \{0, x, y\}. \forall y \in \{0, x, y\}. \exists c. x - y = c *_R u$ 

```

```

        unfolding collinear_def by blast

```

```

        from  $u[\text{rule\_format, of } x \ 0]$   $u[\text{rule\_format, of } y \ 0]$ 

```

```

        obtain  $cx$  and  $cy$  where

```

```

           $cx: x = cx *_R u$  and  $cy: y = cy *_R u$ 

```

```

          by auto

```

```

          from  $cx \ cy \ False$  have  $cx0: cx \neq 0$  and  $cy0: cy \neq 0$  by auto

```

```

          let  $?d = cy / cx$ 

```

```

          from  $cx \ cy \ cx0$  have  $y = ?d *_R x$ 

```

```

          by simp

```

```

          then show ?rhs using False by blast

```

```

next

```

```

  assume  $h: ?rhs$ 

```

```

then obtain  $c$  where  $c: y = c *_{\mathbb{R}} x$ 
using False by blast
show  $?lhs$ 
apply (simp add: collinear_def c)
by (metis (mono_tags, lifting) scaleR_left.minus scaleR_left_diff_distrib
scaleR_one)
qed
qed

```

lemma *collinear_iff_Reals*: $collinear \{0::\text{complex}, w, z\} \longleftrightarrow z/w \in \mathbb{R}$

proof

show $z/w \in \mathbb{R} \implies collinear \{0, w, z\}$

by (*metis Reals_cases collinear_lemma nonzero_divide_eq_eq scaleR_conv_of_real*)

qed (*auto simp: collinear_lemma scaleR_conv_of_real*)

lemma *collinear_scaleR_iff*: $collinear \{0, \alpha *_{\mathbb{R}} w, \beta *_{\mathbb{R}} z\} \longleftrightarrow collinear \{0, w, z\}$

$\vee \alpha=0 \vee \beta=0$

(*is ?lhs = ?rhs*)

proof (*cases $\alpha=0 \vee \beta=0$*)

case *False*

then have $(\exists c. \beta *_{\mathbb{R}} z = (c * \alpha) *_{\mathbb{R}} w) = (\exists c. z = c *_{\mathbb{R}} w)$

by (*metis mult.commute scaleR_scaleR vector_fraction_eq_iff*)

then show $?thesis$

by (*auto simp add: collinear_lemma*)

qed (*auto simp: collinear_lemma*)

lemma *norm_cauchy_schwarz_equal*: $|x \cdot y| = norm\ x * norm\ y \longleftrightarrow collinear \{0, x, y\}$

proof (*cases $x=0$*)

case *True*

then show $?thesis$

by (*auto simp: insert_commute*)

next

case *False*

then have $nnz: norm\ x \neq 0$

by *auto*

show $?thesis$

proof

assume $|x \cdot y| = norm\ x * norm\ y$

then show $collinear \{0, x, y\}$

unfolding *norm_cauchy_schwarz_abs_eq collinear_lemma*

by (*meson eq_vector_fraction_iff nnz*)

next

assume $collinear \{0, x, y\}$

with *False* **show** $|x \cdot y| = norm\ x * norm\ y$

unfolding *norm_cauchy_schwarz_abs_eq collinear_lemma* **by** (*auto simp:*
abs_if)

qed

qed

```

lemma norm_triangle_eq_imp_collinear:
  fixes  $x y :: 'a::real\_inner$ 
  assumes  $norm (x + y) = norm x + norm y$ 
  shows  $collinear\{0,x,y\}$ 
  using assms norm_cauchy_schwarz_abs_eq norm_cauchy_schwarz_equal norm_triangle_eq

  by blast

```

1.5.12 Properties of special hyperplanes

```

lemma subspace_hyperplane: subspace  $\{x. a \cdot x = 0\}$ 
  by (simp add: subspace_def inner_right_distrib)

```

```

lemma subspace_hyperplane2: subspace  $\{x. x \cdot a = 0\}$ 
  by (simp add: inner_commute inner_right_distrib subspace_def)

```

```

lemma special_hyperplane_span:
  fixes  $S :: 'n::euclidean\_space$  set
  assumes  $k \in Basis$ 
  shows  $\{x. k \cdot x = 0\} = span (Basis - \{k\})$ 
proof -
  have  $*$ :  $x \in span (Basis - \{k\})$  if  $k \cdot x = 0$  for  $x$ 
  proof -
    have  $x = (\sum_{b \in Basis. (x \cdot b) *_{\mathbb{R}} b})$ 
      by (simp add: euclidean_representation)
    also have  $\dots = (\sum_{b \in Basis - \{k\}. (x \cdot b) *_{\mathbb{R}} b)$ 
      by (auto simp: sum.remove [of _ k] inner_commute assms that)
    finally have  $x = (\sum_{b \in Basis - \{k\}. (x \cdot b) *_{\mathbb{R}} b)$ .
    then show ?thesis
      by (simp add: span_finite)
  qed
  show ?thesis
    apply (rule span_subspace [symmetric])
    using assms
    apply (auto simp: inner_not_same_Basis intro: * subspace_hyperplane)
  done
qed

```

```

lemma dim_special_hyperplane:
  fixes  $k :: 'n::euclidean\_space$ 
  shows  $k \in Basis \implies dim \{x. k \cdot x = 0\} = DIM('n) - 1$ 
  by (metis Diff_subset card_Diff_singleton indep_card_eq_dim_span independent_substdbasis special_hyperplane_span)

```

```

proposition dim_hyperplane:
  fixes  $a :: 'a::euclidean\_space$ 
  assumes  $a \neq 0$ 
  shows  $dim \{x. a \cdot x = 0\} = DIM('a) - 1$ 

```

proof –

have $\text{span}0: \text{span } \{x. a \cdot x = 0\} = \{x. a \cdot x = 0\}$
by (*rule span_unique*) (*auto simp: subspace_hyperplane*)
then obtain B **where** *independent* B
and $B_{\text{sub}}: B \subseteq \{x. a \cdot x = 0\}$
and $\text{subsp}B: \{x. a \cdot x = 0\} \subseteq \text{span } B$
and $\text{card}0: (\text{card } B = \text{dim } \{x. a \cdot x = 0\})$
and *ortho*: *pairwise orthogonal* B
using *orthogonal_basis_exists* **by** *metis*
with *assms* **have** $a \notin \text{span } B$
by (*metis (mono_tags, lifting) span_eq inner_eq zero_iff mem_Collect_eq span0*)
then have *ind*: *independent* (*insert* a B)
by (*simp add: <independent B> independent_insert*)
have *finite* B
using *<independent B> independent_bound* **by** *blast*
have $\text{UNIV} \subseteq \text{span } (\text{insert } a B)$
proof **fix** $y::'a$
obtain $r z$ **where** $y = r *_R a + z a \cdot z = 0$
by (*metis add commute diff_add_cancel vector_sub_project_orthogonal*)
then show $y \in \text{span } (\text{insert } a B)$
by (*metis (mono_tags, lifting) Bsub add_diff_cancel_left'*
mem_Collect_eq span0 span_breakdown_eq span_eq subspB)
qed
then have $\text{DIM}('a) = \text{dim}(\text{insert } a B)$
by (*metis independent_Basis span_Basis dim_eq_card top.extremum_uniqueI*)
then show *?thesis*
by (*metis One_nat_def <a $\notin \text{span } B$> <finite B> card0 card_insert_disjoint diff_Suc_Suc diff_zero dim_eq_card_independent ind span_base*)
qed

lemma *lowdim_eq_hyperplane*:

fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes $\text{dim } S = \text{DIM}('a) - 1$
obtains a **where** $a \neq 0$ **and** $\text{span } S = \{x. a \cdot x = 0\}$
proof –
obtain b **where** $b \neq 0$ $\text{span } S \subseteq \{x. b \cdot x = 0\}$
by (*metis DIM_positive assms diff_less zero_less_one lowdim_subset_hyperplane*)
then show *?thesis*
by (*metis assms dim_hyperplane dim_span dim_subset subspace_dim_equal subspace_hyperplane subspace_span that*)
qed

lemma *dim_eq_hyperplane*:

fixes $S :: 'n::\text{euclidean_space}$ *set*
shows $\text{dim } S = \text{DIM}('n) - 1 \iff (\exists a. a \neq 0 \wedge \text{span } S = \{x. a \cdot x = 0\})$
by (*metis One_nat_def dim_hyperplane dim_span lowdim_eq_hyperplane*)

1.5.13 Orthogonal bases and Gram-Schmidt process

lemma *pairwise_orthogonal_independent*:

assumes *pairwise orthogonal S* and $0 \notin S$

shows *independent S*

proof –

have $0: \bigwedge x y. \llbracket x \neq y; x \in S; y \in S \rrbracket \implies x \cdot y = 0$

using *assms* by (*simp add: pairwise_def orthogonal_def*)

have *False* if $a \in S$ and $a: a \in \text{span } (S - \{a\})$ for a

proof –

obtain $T U$ where $T \subseteq S - \{a\}$ $a = (\sum_{v \in T}. U v *_R v)$

using a by (*force simp: span_explicit*)

then have $a \cdot a = a \cdot (\sum_{v \in T}. U v *_R v)$

by *simp*

also have $\dots = 0$

apply (*simp add: inner_sum_right*)

by (*smt (verit) 0 DiffE <T ⊆ S - {a}> in_mono insertCI mult_not_zero sum.neutral that(1)*)

finally show *?thesis*

using $\langle 0 \notin S \rangle \langle a \in S \rangle$ by *auto*

qed

then show *?thesis*

by (*force simp: dependent_def*)

qed

lemma *pairwise_orthogonal_imp_finite*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes *pairwise orthogonal S*

shows *finite S*

by (*metis Set.set_insert assms finite_insert independent_bound pairwise_insert*

pairwise_orthogonal_independent)

lemma *subspace_orthogonal_to_vector*: *subspace {y. orthogonal x y}*

by (*simp add: subspace_def orthogonal_clauses*)

lemma *subspace_orthogonal_to_vectors*: *subspace {y. $\forall x \in S. \text{orthogonal } x y$ }*

by (*simp add: subspace_def orthogonal_clauses*)

lemma *orthogonal_to_span*:

assumes $a: a \in \text{span } S$ and $x: \bigwedge y. y \in S \implies \text{orthogonal } x y$

shows *orthogonal x a*

by (*metis a orthogonal_clauses(1,2,4)*

span_induct_alt x)

proposition *Gram_Schmidt_step*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes S : *pairwise orthogonal S* and $x: x \in \text{span } S$

shows *orthogonal x (a - ($\sum_{b \in S}. (b \cdot a / (b \cdot b)) *_R b$))*

proof –


```

have finite S
  by (simp add: S pairwise_orthogonal_imp_finite)
have orthogonal (a - (∑ b∈S. (b · a / (b · b)) *R b)) x
  if x ∈ S for x
proof -
  have a · x = (∑ y∈S. if y = x then y · a else 0)
    by (simp add: ⟨finite S⟩ inner_commute that)
  also have ... = (∑ b∈S. b · a * (b · x) / (b · b))
    apply (rule sum.cong [OF refl], simp)
    by (meson S orthogonal_def pairwise_def that)
  finally show ?thesis
    by (simp add: orthogonal_def algebra_simps inner_sum_left)
qed
then show ?thesis
  using orthogonal_to_span orthogonal_commute x by blast
qed

lemma orthogonal_extension_aux:
  fixes S :: 'a::euclidean_space set
  assumes finite T finite S pairwise_orthogonal S
  shows ∃ U. pairwise_orthogonal (S ∪ U) ∧ span (S ∪ U) = span (S ∪ T)
using assms
proof (induction arbitrary: S)
  case empty then show ?case
    by simp (metis sup_bot_right)
next
  case (insert a T)
  have 0: ∧x y. [x ≠ y; x ∈ S; y ∈ S] ⇒ x · y = 0
    using insert by (simp add: pairwise_def orthogonal_def)
  define a' where a' = a - (∑ b∈S. (b · a / (b · b)) *R b)
  obtain U where orthU: pairwise_orthogonal (S ∪ insert a' U)
    and spanU: span (insert a' S ∪ U) = span (insert a' S ∪ T)
  by (rule exE [OF insert.IH [of insert a' S]])
    (auto simp: Gram_Schmidt_step a'_def insert.premis orthogonal_commute
      pairwise_orthogonal_insert span_clauses)
  have orthS: ∧x. x ∈ S ⇒ a' · x = 0
    using Gram_Schmidt_step a'_def insert.premis orthogonal_commute ortho-
    nal_def span_base by blast
  have span (S ∪ insert a' U) = span (insert a' (S ∪ T))
    using spanU by simp
  also have ... = span (insert a (S ∪ T))
    by (simp add: a'_def span_neg span_sum span_base span_mul eq_span_insert_eq)
  also have ... = span (S ∪ insert a T)
    by simp
  finally show ?case
    using orthU by blast
qed

```

proposition *orthogonal_extension*:
fixes $S :: 'a::\text{euclidean_space set}$
assumes S : pairwise orthogonal S
obtains U **where** pairwise orthogonal $(S \cup U)$ $\text{span } (S \cup U) = \text{span } (S \cup T)$
proof –
obtain B **where** finite B $\text{span } B = \text{span } T$
using *basis_subspace_exists* [of $\text{span } T$] *subspace_span* **by** *metis*
with *orthogonal_extension_aux* [of B S]
obtain U **where** pairwise orthogonal $(S \cup U)$ $\text{span } (S \cup U) = \text{span } (S \cup B)$
using *assms pairwise_orthogonal_imp_finite* **by** *auto*
with $\langle \text{span } B = \text{span } T \rangle$ **show** *?thesis*
by (*rule_tac* $U=U$ **in** *that*) (*auto simp: span_Un*)
qed

corollary *orthogonal_extension_strong*:
fixes $S :: 'a::\text{euclidean_space set}$
assumes S : pairwise orthogonal S
obtains U **where** $U \cap (\text{insert } 0 S) = \{\}$ pairwise orthogonal $(S \cup U)$
 $\text{span } (S \cup U) = \text{span } (S \cup T)$
proof –
obtain U **where** U : pairwise orthogonal $(S \cup U)$ $\text{span } (S \cup U) = \text{span } (S \cup T)$
using *orthogonal_extension* *assms* **by** *blast*
moreover **have** pairwise orthogonal $(S \cup (U - \text{insert } 0 S))$
by (*smt* (*verit*, *best*) *Un_Diff_Int Un_iff U pairwise_def*)
ultimately **show** *?thesis*
by (*metis Diff_disjoint Un_Diff_cancel Un_insert_left inf_commute span_insert_0 that*)
qed

1.5.14 Decomposing a vector into parts in orthogonal subspaces

existence of orthonormal basis for a subspace.

lemma *orthogonal_spanningset_subspace*:
fixes $S :: 'a :: \text{euclidean_space set}$
assumes *subspace* S
obtains B **where** $B \subseteq S$ pairwise orthogonal B $\text{span } B = S$
by (*metis assms basis_orthogonal basis_subspace_exists span_eq*)

lemma *orthogonal_basis_subspace*:
fixes $S :: 'a :: \text{euclidean_space set}$
assumes *subspace* S
obtains B **where** $0 \notin B$ $B \subseteq S$ pairwise orthogonal B independent B
 $\text{card } B = \text{dim } S$ $\text{span } B = S$
by (*metis assms dependent_zero orthogonal_basis_exists span_eq span_eq_iff*)

proposition *orthonormal_basis_subspace*:

```

fixes  $S :: 'a :: euclidean\_space$  set
assumes subspace  $S$ 
obtains  $B$  where  $B \subseteq S$  pairwise orthogonal  $B$ 
  and  $\bigwedge x. x \in B \implies \text{norm } x = 1$ 
  and independent  $B$   $\text{card } B = \text{dim } S$   $\text{span } B = S$ 
proof –
  obtain  $B$  where  $0 \notin B$   $B \subseteq S$ 
    and orth: pairwise orthogonal  $B$ 
    and independent  $B$   $\text{card } B = \text{dim } S$   $\text{span } B = S$ 
    by (blast intro: orthogonal_basis_subspace [OF assms])
  have 1:  $(\lambda x. x /_R \text{norm } x) \text{ ` } B \subseteq S$ 
    using  $\langle \text{span } B = S \rangle$  span_superset span_mul by fastforce
  have 2: pairwise orthogonal  $((\lambda x. x /_R \text{norm } x) \text{ ` } B)$ 
    using orth by (force simp: pairwise_def orthogonal_clauses)
  have 3:  $\bigwedge x. x \in (\lambda x. x /_R \text{norm } x) \text{ ` } B \implies \text{norm } x = 1$ 
    by (metis (no_types, lifting) 0notinB image_iff norm_sgn sgn_div_norm)
  have 4: independent  $((\lambda x. x /_R \text{norm } x) \text{ ` } B)$ 
    by (metis 2 3 norm_zero pairwise_orthogonal_independent zero_neq_one)
  have inj_on  $(\lambda x. x /_R \text{norm } x) B$ 
proof
  fix  $x y$ 
  assume  $x \in B$   $y \in B$   $x /_R \text{norm } x = y /_R \text{norm } y$ 
  moreover have  $\bigwedge i. i \in B \implies \text{norm } (i /_R \text{norm } i) = 1$ 
    using 3 by blast
  ultimately show  $x = y$ 
    by (metis norm_eq_1 orth_orthogonal_clauses(7) orthogonal_commute orthogonal_def pairwise_def zero_neq_one)
  qed
  then have 5:  $\text{card } ((\lambda x. x /_R \text{norm } x) \text{ ` } B) = \text{dim } S$ 
    by (metis 1 card B = dim S card_image)
  have 6:  $\text{span } ((\lambda x. x /_R \text{norm } x) \text{ ` } B) = S$ 
    by (metis 1 4 5 assms card_eq_dim independent_imp_finite span_subspace)
  show ?thesis
    by (rule that [OF 1 2 3 4 5 6])
qed

```

proposition *orthogonal_to_subspace_exists_gen*:

```

fixes  $S :: 'a :: euclidean\_space$  set
assumes span  $S \subset \text{span } T$ 
obtains  $x$  where  $x \neq 0$   $x \in \text{span } T$   $\bigwedge y. y \in \text{span } S \implies \text{orthogonal } x y$ 
proof –
  obtain  $B$  where  $B \subseteq \text{span } S$  and orthB: pairwise orthogonal  $B$ 
    and  $\bigwedge x. x \in B \implies \text{norm } x = 1$ 
    and independent  $B$   $\text{card } B = \text{dim } S$   $\text{span } B = \text{span } S$ 
    by (metis dim_span orthonormal_basis_subspace subspace_span)
  with assms obtain  $u$  where spanBT:  $\text{span } B \subseteq \text{span } T$  and  $u \notin \text{span } B$   $u \in \text{span } T$ 
    by auto

```

```

obtain  $C$  where  $orthBC$ : pairwise orthogonal  $(B \cup C)$  and  $spanBC$ :  $span (B \cup C) = span (B \cup \{u\})$ 
  by (blast intro: orthogonal_extension [OF orthB])
show thesis
proof (cases  $C \subseteq insert\ 0\ B$ )
  case True
    then have  $C \subseteq span\ B$ 
      using  $span\_eq$ 
      by (metis span_insert_0 subset_trans)
    moreover have  $u \in span (B \cup C)$ 
      using  $\langle span (B \cup C) = span (B \cup \{u\}) \rangle span\_superset$  by force
    ultimately show ?thesis
      using True  $\langle u \notin span\ B \rangle$ 
      by (metis Un_insert_left span_insert_0 sup.orderE)
  next
    case False
      then obtain  $x$  where  $x \in C\ x \neq 0\ x \notin B$ 
        by blast
      then have  $x \in span\ T$ 
        by (smt (verit, ccfv_SIG) Set.set_insert  $\langle u \in span\ T \rangle$  empty_subsetI insert_subset
          le_sup_iff spanBC spanBT span_mono span_span span_superset subset_trans)
      moreover have orthogonal  $x\ y$  if  $y \in span\ B$  for  $y$ 
        using that
      proof (rule span_induct)
        show subspace  $\{a.\ orthogonal\ x\ a\}$ 
          by (simp add: subspace_orthogonal_to_vector)
        show  $\bigwedge b. b \in B \implies orthogonal\ x\ b$ 
          by (metis Un_iff  $\langle x \in C \rangle \langle x \notin B \rangle orthBC\ pairwise\_def$ )
      qed
    ultimately show ?thesis
      using  $\langle x \neq 0 \rangle$  that  $\langle span\ B = span\ S \rangle$  by auto
  qed
qed

corollary orthogonal_to_subspace_exists:
  fixes  $S :: 'a :: euclidean\_space\ set$ 
  assumes  $dim\ S < DIM('a)$ 
  obtains  $x$  where  $x \neq 0 \wedge y. y \in span\ S \implies orthogonal\ x\ y$ 
proof –
  have  $span\ S \subset UNIV$ 
    by (metis assms dim_eq_full order_less_imp_not_less top.not_eq_extremum)
  with orthogonal_to_subspace_exists_gen [of S UNIV] that show ?thesis
    by (auto)
qed

corollary orthogonal_to_vector_exists:
  fixes  $x :: 'a :: euclidean\_space$ 

```

```

  assumes  $2 \leq DIM('a)$ 
  obtains  $y$  where  $y \neq 0$  orthogonal  $x$   $y$ 
proof -
  have  $dim \{x\} < DIM('a)$ 
  using assms by auto
  then show thesis
  by (rule orthogonal_to_subspace_exists) (simp add: orthogonal_commute span_base
  that)
qed

```

proposition *orthogonal_subspace_decomp_exists*:

```

  fixes  $S :: 'a :: euclidean\_space$  set
  obtains  $y$   $z$ 
  where  $y \in span\ S$ 
  and  $\bigwedge w. w \in span\ S \implies orthogonal\ z\ w$ 
  and  $x = y + z$ 
proof -
  obtain  $T$  where  $0 \notin T$   $T \subseteq span\ S$  pairwise orthogonal  $T$  independent  $T$ 
  card  $T = dim (span\ S)$   $span\ T = span\ S$ 
  using orthogonal_basis_subspace_subspace_span by blast
  let  $?a = \sum b \in T. (b \cdot x / (b \cdot b)) *R\ b$ 
  have orth: orthogonal  $(x - ?a)$   $w$  if  $w \in span\ S$  for  $w$ 
  by (simp add: Gram_Schmidt_step <pairwise orthogonal  $T$ > <span  $T = span$ 
   $S$ >
  orthogonal_commute that)
  with that[of  $?a$   $x - ?a$ ] < $T \subseteq span\ S$ > show thesis
  by (simp add: span_mul span_sum subsetD)
qed

```

lemma *orthogonal_subspace_decomp_unique*:

```

  fixes  $S :: 'a :: euclidean\_space$  set
  assumes  $x + y = x' + y'$ 
  and ST:  $x \in span\ S$   $x' \in span\ S$   $y \in span\ T$   $y' \in span\ T$ 
  and orth:  $\bigwedge a\ b. [a \in S; b \in T] \implies orthogonal\ a\ b$ 
  shows  $x = x' \wedge y = y'$ 
proof -
  have  $x + y - y' = x'$ 
  by (simp add: assms)
  moreover have  $\bigwedge a\ b. [a \in span\ S; b \in span\ T] \implies orthogonal\ a\ b$ 
  by (meson orth orthogonal_commute orthogonal_to_span)
  ultimately have  $0 = x' - x$ 
  using assms
  by (metis add.commute add_diff_cancel_right' diff_right_commute orthogonal_self span_diff)
  with assms show thesis by auto
qed

```

lemma *vector_in_orthogonal_spanningset*:

```

  fixes  $a :: 'a :: euclidean\_space$ 

```

obtains S **where** $a \in S$ pairwise orthogonal S span $S = UNIV$
by (*metis UnI1 Un_UNIV_right insertI1 orthogonal_extension pairwise_singleton span_UNIV*)

lemma *vector_in_orthogonal_basis*:

fixes $a :: 'a::euclidean_space$

assumes $a \neq 0$

obtains S **where** $a \in S$ $0 \notin S$ pairwise orthogonal S independent S finite S
span $S = UNIV$ card $S = DIM('a)$

proof –

obtain S **where** S : $a \in S$ pairwise orthogonal S span $S = UNIV$

using *vector_in_orthogonal_spanningset* .

show *thesis*

proof

show pairwise orthogonal $(S - \{0\})$

using *pairwise_mono S(2)* **by** *blast*

show independent $(S - \{0\})$

by (*simp add: <pairwise orthogonal (S - {0})> pairwise_orthogonal_independent*)

show finite $(S - \{0\})$

using *<independent (S - {0})> independent_imp_finite* **by** *blast*

show card $(S - \{0\}) = DIM('a)$

using *span_delete_0 [of S] S*

by (*simp add: <independent (S - {0})> indep_card_eq_dim_span*)

qed (*use S <a ≠ 0> in auto*)

qed

lemma *vector_in_orthonormal_basis*:

fixes $a :: 'a::euclidean_space$

assumes $norm\ a = 1$

obtains S **where** $a \in S$ pairwise orthogonal $S \wedge x. x \in S \implies norm\ x = 1$
independent S card $S = DIM('a)$ span $S = UNIV$

proof –

have $a \neq 0$

using *assms* **by** *auto*

then obtain S **where** $a \in S$ $0 \notin S$ finite S

and S : pairwise orthogonal S independent S span $S = UNIV$ card $S = DIM('a)$

by (*metis vector_in_orthogonal_basis*)

let $?S = (\lambda x. x /_R norm\ x) \text{ ` } S$

show *thesis*

proof

show $a \in ?S$

using *<a ∈ S> assms image_iff* **by** *fastforce*

next

show pairwise orthogonal $?S$

using *<pairwise orthogonal S>* **by** (*auto simp: pairwise_def orthogonal_def*)

show $\bigwedge x. x \in (\lambda x. x /_R norm\ x) \text{ ` } S \implies norm\ x = 1$

using *<0 ∉ S>* **by** (*auto simp: field_split_simps*)

then show *ind: independent ?S*

```

    by (metis ‹pairwise orthogonal (( $\lambda x. x /_R \text{norm } x$ ) ‘  $S$ )› norm_zero pairwise_orthogonal_independent zero_neq_one)
  have inj_on ( $\lambda x. x /_R \text{norm } x$ )  $S$ 
    unfolding inj_on_def
  by (metis (full_types)  $S(1) \langle 0 \notin S \rangle$  inverse_nonzero_iff_nonzero norm_eq_zero orthogonal_scaleR orthogonal_self pairwise_def)
  then show  $\text{card } ?S = \text{DIM}('a)$ 
    by (simp add: card_image  $S$ )
  then show  $\text{span } ?S = \text{UNIV}$ 
    by (metis ind dim_eq_card dim_eq_full)
qed
qed

```

proposition *dim_orthogonal_sum:*

```

  fixes  $A :: 'a::\text{euclidean\_space set}$ 
  assumes  $\bigwedge x y. [x \in A; y \in B] \implies x \cdot y = 0$ 
  shows  $\text{dim}(A \cup B) = \text{dim } A + \text{dim } B$ 
proof -
  have 1:  $\bigwedge x y. [x \in \text{span } A; y \in B] \implies x \cdot y = 0$ 
    by (erule span_induct [OF _ subspace_hyperplane2]; simp add: assms)
  have  $\bigwedge x y. [x \in \text{span } A; y \in \text{span } B] \implies x \cdot y = 0$ 
    using 1 by (simp add: span_induct [OF _ subspace_hyperplane])
  then have 0:  $\bigwedge x y. [x \in \text{span } A; y \in \text{span } B] \implies x \cdot y = 0$ 
    by simp
  have  $\text{dim}(A \cup B) = \text{dim}(\text{span}(A \cup B))$ 
    by (simp)
  also have  $\text{span}(A \cup B) = ((\lambda(a, b). a + b) \text{ ‘ } (\text{span } A \times \text{span } B))$ 
    by (auto simp add: span_Un image_def)
  also have  $\text{dim} \dots = \text{dim} \{x + y \mid x y. x \in \text{span } A \wedge y \in \text{span } B\}$ 
    by (auto intro!: arg_cong [where f=dim])
  also have  $\dots = \text{dim} \{x + y \mid x y. x \in \text{span } A \wedge y \in \text{span } B\} + \text{dim}(\text{span } A \cap \text{span } B)$ 
    by (auto dest: 0)
  also have  $\dots = \text{dim } A + \text{dim } B$ 
    using dim_sums_Int by fastforce
  finally show ?thesis .
qed

```

lemma *dim_subspace_orthogonal_to_vectors:*

```

  fixes  $A :: 'a::\text{euclidean\_space set}$ 
  assumes subspace  $A$  subspace  $B$   $A \subseteq B$ 
  shows  $\text{dim} \{y \in B. \forall x \in A. \text{orthogonal } x y\} + \text{dim } A = \text{dim } B$ 
proof -
  have  $\text{dim}(\text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x y\} \cup A)) = \text{dim}(\text{span } B)$ 
  proof (rule arg_cong [where f=dim, OF subset_antisym])
    show  $\text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x y\} \cup A) \subseteq \text{span } B$ 
      by (simp add: ‹ $A \subseteq B$ › Collect_restrict span_mono)
  next
    have *:  $x \in \text{span}(\{y \in B. \forall x \in A. \text{orthogonal } x y\} \cup A)$ 

```

```

    if  $x \in B$  for  $x$ 
  proof -
    obtain  $y z$  where  $x = y + z$   $y \in \text{span } A$  and  $\text{orth}: \bigwedge w. w \in \text{span } A \implies$ 
    orthogonal  $z w$ 
      using orthogonal_subspace_decomp_exists [of  $A x$ ] that by auto
    moreover
    have  $y \in \text{span } B$ 
      using  $\langle y \in \text{span } A \rangle$  assms(3) span_mono by blast
    ultimately have  $z \in B \wedge (\forall x. x \in A \longrightarrow \text{orthogonal } x z)$ 
      using assms by (metis orthogonal_commute span_add_eq span_eq_iff that)
    then have  $z: z \in \text{span } \{y \in B. \forall x \in A. \text{orthogonal } x y\}$ 
      by (simp add: span_base)
    then show ?thesis
      by (smt (verit, best)  $\langle x = y + z \rangle \langle y \in \text{span } A \rangle$  le_sup_iff span_add_eq
      span_subspace_induct
      span_superset_subset_iff subspace_span)
    qed
    show  $\text{span } B \subseteq \text{span } (\{y \in B. \forall x \in A. \text{orthogonal } x y\} \cup A)$ 
      by (rule span_minimal) (auto intro: * span_minimal)
    qed
    then show ?thesis
      by (metis (no_types, lifting) dim_orthogonal_sum dim_span mem_Collect_eq
      orthogonal_commute orthogonal_def)
  qed

```

1.5.15 Linear functions are (uniformly) continuous on any set

1.5.16 Topological properties of linear functions

lemma *linear_lim_0*:

assumes *bounded_linear* f
shows $(f \longrightarrow 0)$ (at 0)

proof -

interpret $f: \text{bounded_linear } f$ by *fact*
 have $(f \longrightarrow f 0)$ (at 0)
 using *tendsto_ident_at* by (*rule* $f.tendsto$)
 then show ?thesis **unfolding** $f.zero$.

qed

lemma *linear_continuous_at*:

bounded_linear $f \implies \text{continuous (at } a) f$
 by (*simp* add: *bounded_linear.isUCont* *isUCont_isCont*)

lemma *linear_continuous_within*:

bounded_linear $f \implies \text{continuous (at } x \text{ within } s) f$
 using *continuous_at_imp_continuous_at_within* *linear_continuous_at* by *blast*

lemma *linear_continuous_on*:

bounded_linear $f \implies \text{continuous_on } s f$


```

using continuous_at_imp_continuous_on[of s f] using linear_continuous_at[of
f] by auto

```

lemma *Lim_linear*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space and h :: 'b  $\Rightarrow$  'c::real_normed_vector
assumes (f  $\longrightarrow$  l) F linear h
shows (( $\lambda$ x. h(f x))  $\longrightarrow$  h l) F

```

proof –

```

obtain B where B: B > 0  $\wedge$  x. norm (h x)  $\leq$  B * norm x

```

```

using linear_bounded_pos [OF <linear h>] by blast

```

```

show ?thesis

```

```

unfolding tendsto_iff

```

```

by (simp add: assms bounded_linear.tendsto_linear_linear tendstoD)

```

qed

lemma *linear_continuous_compose*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space and g :: 'b  $\Rightarrow$  'c::real_normed_vector
assumes continuous F f linear g
shows continuous F ( $\lambda$ x. g(f x))
using assms unfolding continuous_def by (rule Lim_linear)

```

lemma *linear_continuous_on_compose*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space and g :: 'b  $\Rightarrow$  'c::real_normed_vector
assumes continuous_on S f linear g
shows continuous_on S ( $\lambda$ x. g(f x))
using assms by (simp add: continuous_on_eq_continuous_within linear_continuous_compose)

```

Also bilinear functions, in composition form

lemma *bilinear_continuous_compose*:

```

fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::real_normed_vector
assumes continuous F f continuous F g bilinear h
shows continuous F ( $\lambda$ x. h (f x) (g x))

```

```

using assms bilinear_conv_bounded_bilinear bounded_bilinear.continuous by
blast

```

lemma *bilinear_continuous_on_compose*:

```

fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::real_normed_vector
and f :: 'd::t2_space  $\Rightarrow$  'a

```

```

assumes continuous_on S f continuous_on S g bilinear h

```

```

shows continuous_on S ( $\lambda$ x. h (f x) (g x))

```

```

using assms by (simp add: continuous_on_eq_continuous_within bilinear_continuous_compose)

```

end

1.6 Affine Sets

theory *Affine*

imports *Linear_Algebra*

begin

lemma *if_smult*: (if P then x else $(y::\text{real})$) $*_R v = (\text{if } P \text{ then } x *_R v \text{ else } y *_R v)$
by *simp*

lemma *sum_delta_notmem*:
assumes $x \notin s$
shows $\text{sum } (\lambda y. \text{if } (y = x) \text{ then } P x \text{ else } Q y) s = \text{sum } Q s$
and $\text{sum } (\lambda y. \text{if } (x = y) \text{ then } P x \text{ else } Q y) s = \text{sum } Q s$
and $\text{sum } (\lambda y. \text{if } (y = x) \text{ then } P y \text{ else } Q y) s = \text{sum } Q s$
and $\text{sum } (\lambda y. \text{if } (x = y) \text{ then } P y \text{ else } Q y) s = \text{sum } Q s$
by (*smt (verit, best) assms sum.cong*)⁺

lemma *span_substd_basis*:
assumes $d: d \subseteq \text{Basis}$
shows $\text{span } d = \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$
(is $_ = ?B$ **)**
proof –
have $d \subseteq ?B$
using d **by** (*auto simp: inner_Basis*)
moreover have $s: \text{subspace } ?B$
using *subspace_substandard*[of $\lambda i. i \notin d$].
ultimately have $\text{span } d \subseteq ?B$
using *span_mono*[of $d ?B$] *span_eq_iff*[of $?B$] **by** *blast*
moreover have $*$: $\text{card } d \leq \text{dim } (\text{span } d)$
by (*simp add: d dim_eq_card_independent_independent_substdbasis*)
moreover from $*$ **have** $\text{dim } ?B \leq \text{dim } (\text{span } d)$
using *dim_substandard[OF assms]* **by** *auto*
ultimately show $?thesis$
by (*simp add: s subspace_dim_equal*)
qed

lemma *basis_to_substdbasis_subspace_isomorphism*:
fixes $B :: 'a::\text{euclidean_space}$ set
assumes *independent* B
shows $\exists f d::'a \text{ set}. \text{card } d = \text{card } B \wedge \text{linear } f \wedge f' B = d \wedge$
 $f' \text{span } B = \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\} \wedge \text{inj_on } f (\text{span } B) \wedge d \subseteq$
 Basis
proof –
have $B: \text{card } B = \text{dim } B$
using *dim_unique*[of $B B \text{card } B$] *assms span_superset*[of B] **by** *auto*
have $\text{dim } B \leq \text{card } (\text{Basis} :: 'a \text{ set})$
using *dim_subset_UNIV*[of B] **by** *simp*
from *obtain_subset_with_card_n*[OF *this*]
obtain $d :: 'a \text{ set}$ **where** $d: d \subseteq \text{Basis}$ **and** $t: \text{card } d = \text{dim } B$
by *auto*
let $?t = \{x::'a::\text{euclidean_space}. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$
have $\exists f. \text{linear } f \wedge f' B = d \wedge f' \text{span } B = ?t \wedge \text{inj_on } f (\text{span } B)$
proof (*intro basis_to_basis_subspace_isomorphism subspace_span subspace_substandard span_superset*)

```

  show  $d \subseteq \{x. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$ 
    using  $d \text{ inner\_not\_same\_Basis}$  by blast
  qed (auto simp: span_substd_basis independent_substdbasis dim_substandard d
  t B assms)
  with  $t \langle \text{card } B = \text{dim } B \rangle d$  show ?thesis by auto
  qed

```

1.6.1 Affine set and affine hull

```

definition affine :: 'a::real_vector set  $\Rightarrow$  bool
  where  $\text{affine } S \longleftrightarrow (\forall x \in S. \forall y \in S. \forall u v. u + v = 1 \longrightarrow u *_R x + v *_R y \in S)$ 

```

```

lemma affine_alt:  $\text{affine } S \longleftrightarrow (\forall x \in S. \forall y \in S. \forall u::\text{real}. (1 - u) *_R x + u *_R y \in S)$ 
  unfolding affine_def by (metis eq_diff_eq')

```

```

lemma affine_empty [iff]:  $\text{affine } \{\}$ 
  unfolding affine_def by auto

```

```

lemma affine_sing [iff]:  $\text{affine } \{x\}$ 
  unfolding affine_alt by (auto simp: scaleR_left_distrib [symmetric])

```

```

lemma affine_UNIV [iff]:  $\text{affine UNIV}$ 
  unfolding affine_def by auto

```

```

lemma affine_Inter [intro]:  $(\bigwedge S. S \in \mathcal{F} \Longrightarrow \text{affine } S) \Longrightarrow \text{affine } (\bigcap \mathcal{F})$ 
  unfolding affine_def by auto

```

```

lemma affine_Int [intro]:  $\text{affine } S \Longrightarrow \text{affine } T \Longrightarrow \text{affine } (S \cap T)$ 
  unfolding affine_def by auto

```

```

lemma affine_scaling:  $\text{affine } S \Longrightarrow \text{affine } ((*_R) c ' S)$ 
  apply (clarsimp simp: affine_def)
  apply (rule_tac  $x=u *_R x + v *_R y$  in image_eqI)
  apply (auto simp: algebra_simps)
  done

```

```

lemma affine_affine_hull [simp]:  $\text{affine}(\text{affine hull } S)$ 
  unfolding hull_def
  using affine_Inter [of  $\{T. \text{affine } T \wedge S \subseteq T\}$ ] by auto

```

```

lemma affine_hull_eq [simp]:  $(\text{affine hull } s = s) \longleftrightarrow \text{affine } s$ 
  by (metis affine_affine_hull hull_same)

```

```

lemma affine_hyperplane:  $\text{affine } \{x. a \cdot x = b\}$ 
  by (simp add: affine_def algebra_simps) (metis distrib_right mult.left_neutral)

```

Some explicit formulations

Formalized by Lars Schewe.

lemma *affine*:

fixes $V::'a::real_vector\ set$

shows *affine* $V \longleftrightarrow$

$(\forall S\ u.\ finite\ S \wedge S \neq \{\} \wedge S \subseteq V \wedge sum\ u\ S = 1 \longrightarrow (\sum_{x \in S} u\ x *_{R}\ x) \in V)$

proof –

have $u *_{R}\ x + v *_{R}\ y \in V$ **if** $x \in V\ y \in V\ u + v = (1::real)$

and $*$: $\bigwedge S\ u.\ \llbracket finite\ S; S \neq \{\}; S \subseteq V; sum\ u\ S = 1 \rrbracket \implies (\sum_{x \in S} u\ x *_{R}\ x) \in V$ **for** $x\ y\ u\ v$

proof (*cases* $x = y$)

case *True*

then show *?thesis*

using *that* **by** (*metis scaleR_add_left scaleR_one*)

next

case *False*

then show *?thesis*

using *that* $*$ [*of* $\{x,y\}$ $\lambda w.$ *if* $w = x$ *then* u *else* v] **by** *auto*

qed

moreover have $(\sum_{x \in S} u\ x *_{R}\ x) \in V$

if $*$: $\bigwedge x\ y\ u\ v.\ \llbracket x \in V; y \in V; u + v = 1 \rrbracket \implies u *_{R}\ x + v *_{R}\ y \in V$

and *finite* $S\ S \neq \{\} S \subseteq V\ sum\ u\ S = 1$ **for** $S\ u$

proof –

define n **where** $n = card\ S$

consider $card\ S = 0 \mid card\ S = 1 \mid card\ S = 2 \mid card\ S > 2$ **by** *linarith*

then show $(\sum_{x \in S} u\ x *_{R}\ x) \in V$

proof *cases*

assume $card\ S = 1$

then obtain a **where** $S = \{a\}$

by (*auto simp: card_Suc_eq*)

then show *?thesis*

using *that* **by** *simp*

next

assume $card\ S = 2$

then obtain $a\ b$ **where** $S = \{a, b\}$

by (*metis Suc_1 card_1_singletonE card_Suc_eq*)

then show *?thesis*

using $*$ [*of* $a\ b$] *that*

by (*auto simp: sum_clauses(2)*)

next

assume $card\ S > 2$

then show *?thesis* **using** *that* n_def

proof (*induct* n *arbitrary: u S*)

case 0

then show *?case* **by** *auto*

next

case (*Suc* $n\ u\ S$)

```

have sum u S = card S if  $\neg (\exists x \in S. u x \neq 1)$ 
  using that unfolding card_eq_sum by auto
with Suc.prem1 obtain x where  $x \in S$  and  $x: u x \neq 1$  by force
have c:  $\text{card } (S - \{x\}) = \text{card } S - 1$ 
  by (simp add: Suc.prem1)  $\langle x \in S \rangle$ 
have sum u (S - {x}) = 1 - u x
  by (simp add: Suc.prem1 sum_diff1)  $\langle x \in S \rangle$ 
with x have eq1:  $\text{inverse } (1 - u x) * \text{sum } u (S - \{x\}) = 1$ 
  by auto
have inV:  $(\sum y \in S - \{x\}. \text{inverse } (1 - u x) * u y) *_R y \in V$ 
proof (cases  $\text{card } (S - \{x\}) > 2$ )
  case True
  then have S:  $S - \{x\} \neq \{\}$   $\text{card } (S - \{x\}) = n$ 
    using Suc.prem1 c by force+
  show ?thesis
  proof (rule Suc.hyps)
    show  $(\sum a \in S - \{x\}. \text{inverse } (1 - u x) * u a) = 1$ 
      by (auto simp: eq1 sum_distrib_left[symmetric])
    qed (use S Suc.prem1 True in auto)
  next
  case False
  then have card (S - {x}) = Suc (Suc 0)
    using Suc.prem1 c by auto
  then obtain a b where  $ab: (S - \{x\}) = \{a, b\}$   $a \neq b$ 
    unfolding card_Suc_eq by auto
  then show ?thesis
    using eq1  $\langle S \subseteq V \rangle$ 
    by (auto simp: sum_distrib_left distrib_left intro!: Suc.prem1)[of a b]
  qed
have  $u x + (1 - u x) = 1 \implies$ 
 $u x *_R x + (1 - u x) *_R ((\sum y \in S - \{x\}. u y *_R y) /_R (1 - u x)) \in V$ 
  by (rule Suc.prem1) (use  $\langle x \in S \rangle$  Suc.prem1 inV in  $\langle \text{auto simp:}$ 
scaleR_right.sum  $\rangle$ )
moreover have  $(\sum a \in S. u a *_R a) = u x *_R x + (\sum a \in S - \{x\}. u a *_R$ 
a)
  by (meson Suc.prem1) sum.remove  $\langle x \in S \rangle$ 
ultimately show  $(\sum x \in S. u x *_R x) \in V$ 
  by (simp add: x)
qed
qed (use  $\langle S \neq \{\} \rangle$   $\langle \text{finite } S \rangle$  in auto)
qed
ultimately show ?thesis
  unfolding affine_def by meson
qed

```

lemma *affine_hull_explicit*:

affine hull $p = \{y. \exists S u. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p \wedge \text{sum } u S = 1 \wedge \text{sum } (\lambda v. u v *_R v) S = y\}$

```

(is _ = ?rhs)
proof (rule hull_unique)
  have  $\bigwedge x. \text{sum } (\lambda z. 1) \{x\} = 1$ 
    by auto
  show  $p \subseteq ?rhs$ 
  proof (intro subsetI CollectI exI conjI)
    show  $\bigwedge x. \text{sum } (\lambda z. 1) \{x\} = 1$ 
      by auto
  qed auto
  show  $?rhs \subseteq T$  if  $p \subseteq T$  affine  $T$  for  $T$ 
    using that unfolding affine by blast
  show affine ?rhs
    unfolding affine_def
  proof clarify
    fix  $u v :: \text{real}$  and  $sx ux sy uy$ 
    assume  $uv: u + v = 1$ 
      and  $x: \text{finite } sx \ sx \neq \{\}$   $sx \subseteq p \ \text{sum } ux \ sx = (1::\text{real})$ 
      and  $y: \text{finite } sy \ sy \neq \{\}$   $sy \subseteq p \ \text{sum } uy \ sy = (1::\text{real})$ 
    have **:  $(sx \cup sy) \cap sx = sx \ (sx \cup sy) \cap sy = sy$ 
      by auto
    show  $\exists S w. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p \wedge$ 
       $\text{sum } w \ S = 1 \wedge (\sum v \in S. w \ v \ *_R \ v) = u \ *_R \ (\sum v \in sx. ux \ v \ *_R \ v) + v \ *_R$ 
       $(\sum v \in sy. uy \ v \ *_R \ v)$ 
    proof (intro exI conjI)
      show finite  $(sx \cup sy)$ 
        using  $x \ y$  by auto
      show  $\text{sum } (\lambda i. (\text{if } i \in sx \ \text{then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \ \text{then } v \ * \ uy \ i \ \text{else } 0))$ 
       $(sx \cup sy) = 1$ 
        using  $x \ y \ uv$ 
      by (simp add: sum_Un sum.distrib sum.inter_restrict[symmetric] sum_distrib_left
      [symmetric] **)
      have  $(\sum i \in sx \cup sy. ((\text{if } i \in sx \ \text{then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \ \text{then } v \ * \ uy \ i \ \text{else } 0))$ 
       $*_R \ i)$ 
        =  $(\sum i \in sx. (u \ * \ ux \ i) \ *_R \ i) + (\sum i \in sy. (v \ * \ uy \ i) \ *_R \ i)$ 
        using  $x \ y$ 
        unfolding scaleR_left_distrib scaleR_zero_left if_smult
        by (simp add: sum_Un sum.distrib sum.inter_restrict[symmetric] **)
      also have  $\dots = u \ *_R \ (\sum v \in sx. ux \ v \ *_R \ v) + v \ *_R \ (\sum v \in sy. uy \ v \ *_R \ v)$ 
        unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric] by blast
      finally show  $(\sum i \in sx \cup sy. ((\text{if } i \in sx \ \text{then } u \ * \ ux \ i \ \text{else } 0) + (\text{if } i \in sy \ \text{then } v \ * \ uy \ i \ \text{else } 0))$ 
       $*_R \ i)$ 
        =  $u \ *_R \ (\sum v \in sx. ux \ v \ *_R \ v) + v \ *_R \ (\sum v \in sy. uy \ v \ *_R \ v)$  .
    qed (use  $x \ y$  in auto)
  qed
qed

```

lemma affine_hull_finite:

assumes finite S

shows affine hull $S = \{y. \exists u. \text{sum } u \ S = 1 \wedge \text{sum } (\lambda v. u \ v \ *_R \ v) \ S = y\}$

```

proof -
  have *:  $\exists h. \text{sum } h \ S = 1 \wedge (\sum v \in S. h \ v \ *_R \ v) = x$ 
    if  $F \subseteq S$  finite  $F \ F \neq \{\}$  and sum:  $\text{sum } u \ F = 1$  and x:  $(\sum v \in F. u \ v \ *_R \ v)$ 
    =  $x$  for  $x \ F \ u$ 
  proof -
    have  $S \cap F = F$ 
      using that by auto
    show ?thesis
  proof (intro exI conjI)
    show  $(\sum x \in S. \text{if } x \in F \ \text{then } u \ x \ \text{else } 0) = 1$ 
      by (metis (mono_tags, lifting) ‹S ∩ F = F› assms sum.inter_restrict sum)
    show  $(\sum v \in S. (\text{if } v \in F \ \text{then } u \ v \ \text{else } 0) \ *_R \ v) = x$ 
      by (simp add: if_smult cong: if_cong) (metis (no_types) ‹S ∩ F = F›)
    assms sum.inter_restrict x
  qed
qed
show ?thesis
  unfolding affine_hull_explicit using assms
  by (fastforce dest: *)
qed

```

Stepping theorems and hence small special cases

```

lemma affine_hull_empty[simp]: affine hull  $\{\}$  =  $\{\}$ 
  by simp

```

```

lemma affine_hull_finite_step:

```

```

  fixes  $y :: 'a::\text{real\_vector}$ 

```

```

  shows finite  $S \implies$ 

```

```

     $(\exists u. \text{sum } u \ (\text{insert } a \ S) = w \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ (\text{insert } a \ S) = y) \longleftrightarrow$ 

```

```

     $(\exists v \ u. \text{sum } u \ S = w - v \wedge \text{sum } (\lambda x. u \ x \ *_R \ x) \ S = y - v \ *_R \ a)$  (is  $\_ \implies$ 

```

```

    ?lhs = ?rhs)

```

```

proof -

```

```

  assume fin: finite  $S$ 

```

```

  show ?lhs = ?rhs

```

```

  proof

```

```

    assume ?lhs

```

```

    then obtain  $u$  where  $u: \text{sum } u \ (\text{insert } a \ S) = w \wedge (\sum x \in \text{insert } a \ S. u \ x \ *_R$ 

```

```

 $x) = y$ 

```

```

    by auto

```

```

  show ?rhs

```

```

  proof (cases  $a \in S$ )

```

```

    case True

```

```

    then show ?thesis

```

```

    using  $u$  by (simp add: insert_absorb) (metis diff_zero real_vector.scale_zero_left)

```

```

  next

```

```

    case False

```

```

    show ?thesis

```

```

    by (rule exI [where x=u a]) (use u fin False in auto)

```

```

    qed
  next
    assume ?rhs
    then obtain v u where vu: sum u S = w - v ( $\sum x \in S. u x *_R x$ ) = y - v
  *_R a
    by auto
    have *:  $\bigwedge x M. (if x = a then v else M) *_R x = (if x = a then v *_R x else M$ 
  *_R x)
    by auto
    show ?lhs
    proof (cases a  $\in S$ )
      case True
        show ?thesis
          by (rule exI [where x= $\lambda x. (if x=a then v else 0) + u x$ ])
            (simp add: True scaleR_left_distrib sum.distrib sum_clauses fin vu *
  cong: if_cong)
        next
          case False
            then show ?thesis
              apply (rule_tac x= $\lambda x. if x=a then v else u x$  in exI)
                apply (simp add: vu sum_clauses(2)[OF fin] *)
                  by (simp add: sum_delta_notmem(3) vu)
    qed
  qed
  qed

```

lemma affine_hull_2:

```

  fixes a b :: 'a::real_vector
  shows affine_hull {a,b} = {u *_R a + v *_R b | u v. (u + v = 1)}
  (is ?lhs = ?rhs)
  proof -
    have *:
       $\bigwedge x y z. z = x - y \longleftrightarrow y + z = (x::real)$ 
       $\bigwedge x y z. z = x - y \longleftrightarrow y + z = (x::'a)$  by auto
    have ?lhs = {y.  $\exists u. \text{sum } u \{a, b\} = 1 \wedge (\sum v \in \{a, b\}. u v *_R v) = y$ }
      using affine_hull_finite[of {a,b}] by auto
    also have ... = {y.  $\exists v u. u b = 1 - v \wedge u b *_R b = y - v *_R a$ }
      by (simp add: affine_hull_finite_step[of {b} a])
    also have ... = ?rhs unfolding * by auto
    finally show ?thesis by auto
  qed

```

lemma affine_hull_3:

```

  fixes a b c :: 'a::real_vector
  shows affine_hull {a,b,c} = { u *_R a + v *_R b + w *_R c | u v w. u + v + w =
  1 }
  proof -
    have *:
       $\bigwedge x y z. z = x - y \longleftrightarrow y + z = (x::real)$ 

```



```

 $\wedge x y z. z = x - y \longleftrightarrow y + z = (x::'a)$  by auto
show ?thesis
  apply (simp add: affine_hull_finite affine_hull_finite_step)
  unfolding *
  apply safe
  apply (metis add.assoc)
  apply (rule_tac x=u in exI, force)
  done
qed

```

```

lemma mem_affine:
  assumes affine S x ∈ S y ∈ S u + v = 1
  shows u *R x + v *R y ∈ S
  using assms affine_def[of S] by auto

```

```

lemma mem_affine_3:
  assumes affine S x ∈ S y ∈ S z ∈ S u + v + w = 1
  shows u *R x + v *R y + w *R z ∈ S
proof –
  have u *R x + v *R y + w *R z ∈ affine_hull {x, y, z}
    using affine_hull_3[of x y z] assms by auto
  moreover
  have affine_hull {x, y, z} ⊆ affine_hull S
    using hull_mono[of {x, y, z} S] assms by auto
  moreover
  have affine_hull S = S
    using assms affine_hull_eq[of S] by auto
  ultimately show ?thesis by auto
qed

```

```

lemma mem_affine_3_minus:
  assumes affine S x ∈ S y ∈ S z ∈ S
  shows x + v *R (y-z) ∈ S
  using mem_affine_3[of S x y z 1 v -v] assms
  by (simp add: algebra_simps)

```

```

corollary mem_affine_3_minus2:
   $\llbracket$  affine S; x ∈ S; y ∈ S; z ∈ S  $\rrbracket \implies x - v *_{\mathbb{R}} (y-z) \in S$ 
  by (metis add_uminus_conv_diff mem_affine_3_minus real_vector.scale_minus_left)

```

Some relations between affine hull and subspaces

```

lemma affine_hull_insert_subset_span:
  affine_hull (insert a S) ⊆ {a + v | v . v ∈ span {x - a | x . x ∈ S}}
proof –
  have  $\exists v T u. x = a + v \wedge (\text{finite } T \wedge T \subseteq \{x - a \mid x. x \in S\} \wedge (\sum_{v \in T} u v$ 
  *R v) = v)
  if finite F F ≠ {} F ⊆ insert a S sum u F = 1  $(\sum_{v \in F} u v *_{\mathbb{R}} v) = x$ 
  for x F u

```

```

proof -
  have *:  $(\lambda x. x - a) \text{ ' } (F - \{a\}) \subseteq \{x - a \mid x. x \in S\}$ 
    using that by auto
  show ?thesis
  proof (intro exI conjI)
    show finite  $(\lambda x. x - a) \text{ ' } (F - \{a\})$ 
      by (simp add: that(1))
    show  $(\sum v \in (\lambda x. x - a) \text{ ' } (F - \{a\}). u(v+a) *_R v) = x - a$ 
      by (simp add: sum.reindex[unfolded inj_on_def] algebra_simps
        sum_subtractf scaleR_left.sum[symmetric] sum_diff1 that)
  qed (use <F ⊆ insert a S> in auto)
qed
then show ?thesis
  unfolding affine_hull_explicit span_explicit by fast
qed

lemma affine_hull_insert_span:
  assumes  $a \notin S$ 
  shows affine hull (insert a S) = {a + v | v . v ∈ span {x - a | x. x ∈ S}}
proof -
  have *:  $\exists G u. \text{finite } G \wedge G \neq \{\} \wedge G \subseteq \text{insert } a \text{ } S \wedge \text{sum } u \text{ } G = 1 \wedge (\sum v \in G. u \text{ } v *_R v) = y$ 
    if  $v \in \text{span } \{x - a \mid x. x \in S\}$   $y = a + v$  for  $y \text{ } v$ 
  proof -
    from that
    obtain  $T \text{ } u$  where  $u: \text{finite } T \text{ } T \subseteq \{x - a \mid x. x \in S\} \text{ } a + (\sum v \in T. u \text{ } v *_R v) = y$ 
    unfolding span_explicit by auto
    define  $F$  where  $F = (\lambda x. x + a) \text{ ' } T$ 
    have  $F: \text{finite } F \text{ } F \subseteq S \text{ } (\sum v \in F. u \text{ } (v - a) *_R (v - a)) = y - a$ 
    unfolding  $F\_def$  using  $u$  by (auto simp: sum.reindex[unfolded inj_on_def])
    have *:  $F \cap \{a\} = \{\} \text{ } F \cap - \{a\} = F$ 
    using  $F$  assms by auto
    show  $\exists G u. \text{finite } G \wedge G \neq \{\} \wedge G \subseteq \text{insert } a \text{ } S \wedge \text{sum } u \text{ } G = 1 \wedge (\sum v \in G. u \text{ } v *_R v) = y$ 
    apply (rule_tac x = insert a F in exI)
    apply (rule_tac x = λx. if x=a then 1 - sum (λx. u (x - a)) F else u (x - a) in exI)
    using assms F
    apply (auto simp: sum_clauses sum.If_cases if_smult sum_subtractf scaleR_left.sum algebra_simps *)
    done
  qed
  show ?thesis
  by (intro subset_antisym affine_hull_insert_subset_span) (auto simp: affine_hull_explicit dest!: *)
qed

lemma affine_hull_span:

```

```

assumes  $a \in S$ 
shows  $\text{affine\_hull } S = \{a + v \mid v. v \in \text{span } \{x - a \mid x. x \in S - \{a\}\}\}$ 
using  $\text{affine\_hull\_insert\_span}$  [of  $a$   $S - \{a\}$ ,  $\text{unfolded insert\_Diff}$  [OF  $\text{assms}$ ]]
by auto

```

Parallel affine sets

```

definition  $\text{affine\_parallel} :: 'a::\text{real\_vector\_set} \Rightarrow 'a::\text{real\_vector\_set} \Rightarrow \text{bool}$ 
where  $\text{affine\_parallel } S \ T \longleftrightarrow (\exists a. T = (\lambda x. a + x) \ ` S)$ 

```

```

lemma  $\text{affine\_parallel\_expl\_aux}$ :
fixes  $S \ T :: 'a::\text{real\_vector\_set}$ 
assumes  $\bigwedge x. x \in S \longleftrightarrow a + x \in T$ 
shows  $T = (\lambda x. a + x) \ ` S$ 

```

proof –

```

have  $x \in ((\lambda x. a + x) \ ` S)$  if  $x \in T$  for  $x$ 
using that

```

```

by (simp add: image_iff) (metis add.commute diff_add_cancel assms)

```

```

moreover have  $T \geq (\lambda x. a + x) \ ` S$ 

```

```

using assms by auto

```

```

ultimately show ?thesis by auto

```

qed

```

lemma  $\text{affine\_parallel\_expl}$ :  $\text{affine\_parallel } S \ T \longleftrightarrow (\exists a. \forall x. x \in S \longleftrightarrow a + x \in T)$ 

```

```

by (auto simp add: affine_parallel_def)
(use affine_parallel_expl_aux [of S _ T] in blast)

```

```

lemma  $\text{affine\_parallel\_reflex}$ :  $\text{affine\_parallel } S \ S$ 
unfolding  $\text{affine\_parallel\_def}$ 
using  $\text{image\_add\_0}$  by blast

```

```

lemma  $\text{affine\_parallel\_commute}$ :
assumes  $\text{affine\_parallel } A \ B$ 
shows  $\text{affine\_parallel } B \ A$ 
by (metis affine_parallel_def assms translation_galois)

```

```

lemma  $\text{affine\_parallel\_assoc}$ :
assumes  $\text{affine\_parallel } A \ B$ 
and  $\text{affine\_parallel } B \ C$ 
shows  $\text{affine\_parallel } A \ C$ 
by (metis affine_parallel_def assms translation_assoc)

```

```

lemma  $\text{affine\_translation\_aux}$ :
fixes  $a :: 'a::\text{real\_vector}$ 
assumes  $\text{affine } ((\lambda x. a + x) \ ` S)$ 
shows  $\text{affine } S$ 

```

proof –

```

{

```

```

fix x y u v
assume xy: x ∈ S y ∈ S (u :: real) + v = 1
then have (a + x) ∈ ((λx. a + x) ‘ S) (a + y) ∈ ((λx. a + x) ‘ S)
  by auto
then have h1: u *R (a + x) + v *R (a + y) ∈ (λx. a + x) ‘ S
  using xy assms unfolding affine_def by auto
have u *R (a + x) + v *R (a + y) = (u + v) *R a + (u *R x + v *R y)
  by (simp add: algebra_simps)
also have ... = a + (u *R x + v *R y)
  using ⟨u + v = 1⟩ by auto
ultimately have a + (u *R x + v *R y) ∈ (λx. a + x) ‘ S
  using h1 by auto
then have u *R x + v *R y ∈ S by auto
}
then show ?thesis unfolding affine_def by auto
qed

```

```

lemma affine_translation:
  affine S ↔ affine ((+) a ‘ S) for a :: 'a::real_vector
by (metis affine_translation_aux translation_galois)

```

```

lemma parallel_is_affine:
  fixes S T :: 'a::real_vector set
  assumes affine S affine_parallel S T
  shows affine T
by (metis affine_parallel_def affine_translation assms)

```

```

lemma subspace_imp_affine: subspace s ⇒ affine s
unfolding subspace_def affine_def by auto

```

```

lemma affine_hull_subset_span: (affine hull s) ⊆ (span s)
by (metis hull_minimal span_superset subspace_imp_affine subspace_span)

```

Subspace parallel to an affine set

```

lemma subspace_affine: subspace S ↔ affine S ∧ 0 ∈ S
by (metis add_cancel_right_left affine_alt diff_add_cancel mem_affine_3 scaleR_eq_0_iff
  subspace_def vector_space_assms(4))

```

```

lemma affine_diffs_subspace:
  assumes affine S a ∈ S
  shows subspace ((λx. (-a)+x) ‘ S)
by (metis ab_left_minus affine_translation assms image_eqI subspace_affine)

```

```

lemma affine_diffs_subspace_subtract:
  subspace ((λx. x - a) ‘ S) if affine S a ∈ S
using that affine_diffs_subspace [of _ a] by simp

```

```

lemma parallel_subspace_explicit:

```

```

assumes affine S
and a ∈ S
assumes L ≡ {y. ∃ x ∈ S. (-a) + x = y}
shows subspace L ∧ affine_parallel S L
by (smt (verit) Collect_cong ab_left_minus affine_parallel_def assms image_def
mem_Collect_eq parallel_is_affine subspace_affine)

```

```

lemma parallel_subspace_aux:
assumes subspace A
and subspace B
and affine_parallel A B
shows A ⊇ B
by (metis add.right_neutral affine_parallel_expl assms subsetI subspace_def)

```

```

lemma parallel_subspace:
assumes subspace A
and subspace B
and affine_parallel A B
shows A = B
by (simp add: affine_parallel_commute assms parallel_subspace_aux subset_antisym)

```

```

lemma affine_parallel_subspace:
assumes affine S S ≠ {}
shows ∃!L. subspace L ∧ affine_parallel S L
by (meson affine_parallel_assoc affine_parallel_commute assms equalsOI parallel_subspace_parallel_subspace_explicit)

```

1.6.2 Affine Dependence

Formalized by Lars Schewe.

```

definition affine_dependent :: 'a::real_vector set ⇒ bool
where affine_dependent S ⟷ (∃ x ∈ S. x ∈ affine_hull (S - {x}))

```

```

lemma affine_dependent_imp_dependent: affine_dependent S ⇒ dependent S
unfolding affine_dependent_def dependent_def
using affine_hull_subset_span by auto

```

```

lemma affine_dependent_subset:
[[affine_dependent S; S ⊆ T]] ⇒ affine_dependent T
using hull_mono [OF Diff_mono [OF_subset_refl]]
by (smt (verit) affine_dependent_def subsetD)

```

```

lemma affine_independent_subset:
shows [[¬ affine_dependent T; S ⊆ T]] ⇒ ¬ affine_dependent S
by (metis affine_dependent_subset)

```

```

lemma affine_independent_Diff:
¬ affine_dependent S ⇒ ¬ affine_dependent(S - T)
by (meson Diff_subset affine_dependent_subset)

```

proposition *affine_dependent_explicit*:

affine_dependent $p \longleftrightarrow$
 $(\exists S U. \text{finite } S \wedge S \subseteq p \wedge \text{sum } U S = 0 \wedge (\exists v \in S. U v \neq 0) \wedge \text{sum } (\lambda v. U v *_{\mathbb{R}} v) S = 0)$

proof –

have $\exists S U. \text{finite } S \wedge S \subseteq p \wedge \text{sum } U S = 0 \wedge (\exists v \in S. U v \neq 0) \wedge (\sum_{w \in S} U w *_{\mathbb{R}} w) = 0$

if $(\sum_{w \in S} U w *_{\mathbb{R}} w) = x \ x \in p \ \text{finite } S \ S \neq \{\} \ S \subseteq p - \{x\} \ \text{sum } U S = 1$
for $x \ S \ U$

proof (*intro exI conjI*)

have $x \notin S$

using *that by auto*

then show $(\sum_{v \in \text{insert } x \ S} \text{if } v = x \ \text{then } -1 \ \text{else } U v) = 0$

using *that by (simp add: sum_delta_notmem)*

show $(\sum_{w \in \text{insert } x \ S} (\text{if } w = x \ \text{then } -1 \ \text{else } U w) *_{\mathbb{R}} w) = 0$

using *that <x ∉ S> by (simp add: if_smult sum_delta_notmem cong: if_cong)*

qed (*use that in auto*)

moreover have $\exists x \in p. \exists S U. \text{finite } S \wedge S \neq \{\} \wedge S \subseteq p - \{x\} \wedge \text{sum } U S = 1 \wedge (\sum_{v \in S} U v *_{\mathbb{R}} v) = x$

if $(\sum_{v \in S} U v *_{\mathbb{R}} v) = 0 \ \text{finite } S \ S \subseteq p \ \text{sum } U S = 0 \ v \in S \ U v \neq 0$ **for** $S \ U \ v$

proof (*intro bexI exI conjI*)

have $S \neq \{v\}$

using *that by auto*

then show $S - \{v\} \neq \{\}$

using *that by auto*

show $(\sum_{x \in S - \{v\}} -(1 / U v) * U x) = 1$

unfolding *sum_distrib_left[symmetric] sum_diff1[OF <finite S>]* **by** (*simp add: that*)

show $(\sum_{x \in S - \{v\}} -(1 / U v) * U x) *_{\mathbb{R}} x = v$

unfolding *sum_distrib_left [symmetric] scaleR_scaleR[symmetric]*
scaleR_right.sum [symmetric] sum_diff1[OF <finite S>]

using *that by auto*

show $S - \{v\} \subseteq p - \{v\}$

using *that by auto*

qed (*use that in auto*)

ultimately show *?thesis*

unfolding *affine_dependent_def affine_hull_explicit* **by auto**

qed

lemma *affine_dependent_explicit_finite*:

fixes $S :: 'a :: \text{real_vector set}$

assumes *finite S*

shows *affine_dependent S* \longleftrightarrow

$(\exists U. \text{sum } U S = 0 \wedge (\exists v \in S. U v \neq 0) \wedge \text{sum } (\lambda v. U v *_{\mathbb{R}} v) S = 0)$

(**is** *?lhs = ?rhs*)

proof

have $*$: $\bigwedge vt \ U v. (\text{if } vt \ \text{then } U v \ \text{else } 0) *_{\mathbb{R}} v = (\text{if } vt \ \text{then } (U v) *_{\mathbb{R}} v \ \text{else } 0 :: 'a)$

```

    by auto
  assume ?lhs
  then obtain  $T U v$  where
    finite  $T$   $T \subseteq S$   $\text{sum } U T = 0$   $v \in T$   $U v \neq 0$   $(\sum v \in T. U v *_{\mathbb{R}} v) = 0$ 
    unfolding affine_dependent_explicit by auto
  then show ?rhs
    apply (rule_tac  $x = \lambda x. \text{if } x \in T \text{ then } U x \text{ else } 0$  in  $exI$ )
    apply (auto simp: * sum.inter_restrict[OF assms, symmetric] Int_absorb1[OF
  ‹ $T \subseteq S$ ›])
    done
next
  assume ?rhs
  then obtain  $U v$  where  $\text{sum } U S = 0$   $v \in S$   $U v \neq 0$   $(\sum v \in S. U v *_{\mathbb{R}} v) = 0$ 
  by auto
  then show ?lhs unfolding affine_dependent_explicit
  using assms by auto
qed

lemma dependent_imp_affine_dependent:
  assumes dependent  $\{x - a \mid x. x \in S\}$ 
  and  $a \notin S$ 
  shows affine_dependent (insert a S)
proof -
  from assms(1)[unfolded dependent_explicit] obtain  $S' U v$ 
  where  $S$ : finite  $S'$   $S' \subseteq \{x - a \mid x. x \in S\}$   $v \in S'$   $U v \neq 0$   $(\sum v \in S'. U v *_{\mathbb{R}} v) = 0$ 
  by auto
  define  $T$  where  $T = (\lambda x. x + a) ` S'$ 
  have inj: inj_on  $(\lambda x. x + a)$   $S'$ 
  unfolding inj_on_def by auto
  have  $0 \notin S'$ 
  using  $S(2)$  assms(2) unfolding subset_eq by auto
  have fin: finite  $T$  and  $T \subseteq S$ 
  unfolding  $T\_def$  using  $S(1,2)$  by auto
  then have finite (insert a T) and insert a  $T \subseteq \text{insert a } S$ 
  by auto
  moreover have *:  $\bigwedge P Q. (\sum x \in T. (\text{if } x = a \text{ then } P x \text{ else } Q x)) = (\sum x \in T. Q x)$ 
  by (smt (verit, best) ‹ $T \subseteq S$ › assms(2) subsetD sum.cong)
  have  $(\sum x \in \text{insert a } T. \text{if } x = a \text{ then } - (\sum x \in T. U (x - a)) \text{ else } U (x - a)) = 0$ 
  by (smt (verit) ‹ $T \subseteq S$ › assms(2) fin insert_absorb insert_subset sum.insert sum_mono)
  moreover have  $\exists v \in \text{insert a } T. (\text{if } v = a \text{ then } - (\sum x \in T. U (x - a)) \text{ else } U (v - a)) \neq 0$ 
  using  $S(3,4)$  ‹ $0 \notin S'$ ›
  by (rule_tac  $x = v + a$  in  $bexI$ ) (auto simp:  $T\_def$ )
  moreover have *:  $\bigwedge P Q. (\sum x \in T. (\text{if } x = a \text{ then } P x \text{ else } Q x) *_{\mathbb{R}} x) = (\sum x \in T. Q x *_{\mathbb{R}} x)$ 

```

```

    using ⟨a∉S⟩ ⟨T⊆S⟩ by (auto intro!: sum.cong)
  have (∑ x∈T. U (x - a)) *R a = (∑ v∈T. U (v - a) *R v)
    unfolding scaleR_left.sum
  unfolding T_def and sum.reindex[OF inj] and o_def
  using S(5)
  by (auto simp: sum.distrib scaleR_right_distrib)
  then have (∑ v∈insert a T. (if v = a then - (∑ x∈T. U (x - a)) else U (v -
a)) *R v) = 0
    unfolding sum_clauses(2)[OF fin] using ⟨a∉S⟩ ⟨T⊆S⟩ by (auto simp: *)
  ultimately show ?thesis
    unfolding affine_dependent_explicit
  by (force intro!: exI[where x=insert a T])
qed

```

lemma affine_dependent_biggerset:

```

  fixes S :: 'a::euclidean_space set
  assumes finite S card S ≥ DIM('a) + 2
  shows affine_dependent S

```

proof -

```

  have S ≠ {} using assms by auto
  then obtain a where a∈S by auto
  have *: {x - a | x. x ∈ S - {a}} = (λx. x - a) ` (S - {a})
    by auto
  have card {x - a | x. x ∈ S - {a}} = card (S - {a})
    unfolding * by (simp add: card_image inj_on_def)
  also have ... > DIM('a) using assms(2)
    unfolding card_Diff_singleton[OF ⟨a∈S⟩] by auto
  finally have affine_dependent (insert a (S - {a}))
    using dependent_biggerset dependent_imp_affine_dependent by blast
  then show ?thesis
    by (simp add: ⟨a ∈ S⟩ insert_absorb)

```

qed

lemma affine_dependent_biggerset_general:

```

  assumes finite (S :: 'a::euclidean_space set)
  and card S ≥ dim S + 2
  shows affine_dependent S

```

proof -

```

  from assms(2) have S ≠ {} by auto
  then obtain a where a∈S by auto
  have *: {x - a | x. x ∈ S - {a}} = (λx. x - a) ` (S - {a})
    by auto
  have **: card {x - a | x. x ∈ S - {a}} = card (S - {a})
    by (metis (no_types, lifting) * card_image diff_add_cancel inj_on_def)
  have dim {x - a | x. x ∈ S - {a}} ≤ dim S
    using ⟨a∈S⟩ by (auto simp: span_base span_diff intro: subset_le_dim)
  also have ... < dim S + 1 by auto
  also have ... ≤ card (S - {a})
    using assms card_Diff_singleton[OF ⟨a∈S⟩] by auto

```



```

finally have affine_dependent (insert a (S - {a}))
  by (smt (verit) Collect_cong dependent_imp_affine_dependent dependent_biggerset_general
** Diff_iff insertCI)
  then show ?thesis
    by (simp add: ⟨a ∈ S⟩ insert_absorb)
qed

```

1.6.3 Some Properties of Affine Dependent Sets

```

lemma affine_independent_0 [simp]: ¬ affine_dependent {}
  by (simp add: affine_dependent_def)

```

```

lemma affine_independent_1 [simp]: ¬ affine_dependent {a}
  by (simp add: affine_dependent_def)

```

```

lemma affine_independent_2 [simp]: ¬ affine_dependent {a,b}
  by (simp add: affine_dependent_def insert_Diff_if hull_same)

```

```

lemma affine_hull_translation: affine_hull ((λx. a + x) ` S) = (λx. a + x) `
(affine_hull S)

```

proof –

```

  have affine ((λx. a + x) ` (affine_hull S))
    using affine_translation affine_affine_hull by blast
  moreover have (λx. a + x) ` S ⊆ (λx. a + x) ` (affine_hull S)
    using hull_subset[of S] by auto
  ultimately have h1: affine_hull ((λx. a + x) ` S) ⊆ (λx. a + x) ` (affine_hull
S)
    by (metis hull_minimal)
  have affine((λx. -a + x) ` (affine_hull ((λx. a + x) ` S)))
    using affine_translation affine_affine_hull by blast
  moreover have (λx. -a + x) ` (λx. a + x) ` S ⊆ (λx. -a + x) ` (affine_hull
((λx. a + x) ` S))
    using hull_subset[of (λx. a + x) ` S] by auto
  moreover have S = (λx. -a + x) ` (λx. a + x) ` S
    using translation_assoc[of -a a] by auto
  ultimately have (λx. -a + x) ` (affine_hull ((λx. a + x) ` S)) >= (affine_hull
S)
    by (metis hull_minimal)
  then have affine_hull ((λx. a + x) ` S) >= (λx. a + x) ` (affine_hull S)
    by auto
  then show ?thesis using h1 by auto
qed

```

```

lemma affine_dependent_translation:
  assumes affine_dependent S
  shows affine_dependent ((λx. a + x) ` S)

```

proof –

```

  obtain x where x: x ∈ S ∧ x ∈ affine_hull (S - {x})
    using assms affine_dependent_def by auto

```

have $(+)$ $a \in (S - \{x\}) = (+)$ $a \in S - \{a + x\}$
by *auto*
then have $a + x \in \text{affine hull } ((\lambda x. a + x) \in S - \{a + x\})$
using *affine_hull_translation*[of $a \in S - \{x\}$] x **by** *auto*
moreover have $a + x \in (\lambda x. a + x) \in S$
using x **by** *auto*
ultimately show *?thesis*
unfolding *affine_dependent_def* **by** *auto*
qed

lemma *affine_dependent_translation_eq*:
 $\text{affine_dependent } S \longleftrightarrow \text{affine_dependent } ((\lambda x. a + x) \in S)$
by (*metis affine_dependent_translation_translation_galois*)

lemma *affine_hull_0_dependent*:
assumes $0 \in \text{affine hull } S$
shows *dependent* S
proof –
obtain $s \ u$ **where** $s \ u$: $\text{finite } s \wedge s \neq \{\} \wedge s \subseteq S \wedge \text{sum } u \ s = 1 \wedge (\sum_{v \in s. u \ v *_{\mathbb{R}} v) = 0$
using *assms affine_hull_explicit*[of S] **by** *auto*
then have $\exists v \in s. u \ v \neq 0$ **by** *auto*
then have $\text{finite } s \wedge s \subseteq S \wedge (\exists v \in s. u \ v \neq 0 \wedge (\sum_{v \in s. u \ v *_{\mathbb{R}} v) = 0)$
using $s \ u$ **by** *auto*
then show *?thesis*
unfolding *dependent_explicit*[of S] **by** *auto*
qed

lemma *affine_dependent_imp_dependent2*:
assumes *affine_dependent* $(\text{insert } 0 \ S)$
shows *dependent* S
proof –
obtain x **where** x : $x \in \text{insert } 0 \ S \wedge x \in \text{affine hull } (\text{insert } 0 \ S - \{x\})$
using *affine_dependent_def*[of $(\text{insert } 0 \ S)$] *assms* **by** *blast*
then have $x \in \text{span } (\text{insert } 0 \ S - \{x\})$
using *affine_hull_subset_span* **by** *auto*
moreover have $\text{span } (\text{insert } 0 \ S - \{x\}) = \text{span } (S - \{x\})$
using *insert_Diff_if*[of $0 \ S \ \{x\}$] *span_insert_0*[of $S - \{x\}$] **by** *auto*
ultimately have $x \in \text{span } (S - \{x\})$ **by** *auto*
then have $x \neq 0 \implies \text{dependent } S$
using x *dependent_def* **by** *auto*
moreover have *dependent* S **if** $x = 0$
by (*metis that affine_hull_0_dependent Diff_insert_absorb dependent_zero x*)
ultimately show *?thesis* **by** *auto*
qed

lemma *affine_dependent_iff_dependent*:
assumes $a \notin S$
shows $\text{affine_dependent } (\text{insert } a \ S) \longleftrightarrow \text{dependent } ((\lambda x. -a + x) \in S)$

proof –

have $((+) (- a) ' S) = \{x - a \mid x . x \in S\}$ **by** *auto*
then show *?thesis*
using *affine_dependent_translation_eq*[of (insert a S) -a]
affine_dependent_imp_dependent2 *assms*
dependent_imp_affine_dependent[of a S]
by (*auto simp del: uminus_add_conv_diff*)

qed

lemma *affine_dependent_iff_dependent2*:

assumes $a \in S$
shows $\text{affine_dependent } S \longleftrightarrow \text{dependent } ((\lambda x. -a + x) ' (S - \{a\}))$
by (*metis Diff_iff affine_dependent_iff_dependent2 assms insert_Diff singletonI*)

lemma *affine_hull_insert_span_gen*:

$\text{affine_hull } (\text{insert } a \ S) = (\lambda x. a + x) ' \text{span } ((\lambda x. -a + x) ' S)$

proof –

have $h1: \{x - a \mid x . x \in S\} = ((\lambda x. -a + x) ' S)$
by *auto*
{
assume $a \notin S$
then have *?thesis*
using *affine_hull_insert_span*[of a S] *h1* **by** *auto*
}
moreover
{
assume $a1: a \in S$
then have $\text{insert } 0 \ ((\lambda x. -a + x) ' (S - \{a\})) = (\lambda x. -a + x) ' S$
by *auto*
then have $\text{span } ((\lambda x. -a + x) ' (S - \{a\})) = \text{span } ((\lambda x. -a + x) ' S)$
using *span_insert_0*[of (+) (- a) ' (S - {a})]
by *presburger*
moreover have $\{x - a \mid x . x \in (S - \{a\})\} = ((\lambda x. -a + x) ' (S - \{a\}))$
by *auto*
moreover have $\text{insert } a \ (S - \{a\}) = \text{insert } a \ S$
by *auto*
ultimately have *?thesis*
using *affine_hull_insert_span*[of a S - {a}] **by** *auto*
}
ultimately show *?thesis* **by** *auto*

qed

lemma *affine_hull_span2*:

assumes $a \in S$
shows $\text{affine_hull } S = (\lambda x. a + x) ' \text{span } ((\lambda x. -a + x) ' (S - \{a\}))$
by (*metis affine_hull_insert_span_gen assms insert_Diff*)

lemma *affine_hull_span_gen*:

assumes $a \in \text{affine_hull } S$

shows $\text{affine hull } S = (\lambda x. a+x) \text{ ' } \text{span } ((\lambda x. -a+x) \text{ ' } S)$
by (*metis affine_hull_insert_span_gen assms hull_redundant*)

lemma *affine_hull_span_0*:
assumes $0 \in \text{affine hull } S$
shows $\text{affine hull } S = \text{span } S$
using *affine_hull_span_gen[of 0 S] assms by auto*

lemma *extend_to_affine_basis_nonempty*:
fixes $S V :: 'n::\text{real_vector set}$
assumes $\neg \text{affine_dependent } S \ S \subseteq V \ S \neq \{\}$
shows $\exists T. \neg \text{affine_dependent } T \wedge S \subseteq T \wedge T \subseteq V \wedge \text{affine hull } T = \text{affine hull } V$

proof –

obtain a **where** $a: a \in S$
using *assms by auto*
then have $h0: \text{independent } ((\lambda x. -a + x) \text{ ' } (S - \{a\}))$
using *affine_dependent_iff_dependent2 assms by auto*
obtain B **where** B :
 $(\lambda x. -a+x) \text{ ' } (S - \{a\}) \subseteq B \wedge B \subseteq (\lambda x. -a+x) \text{ ' } V \wedge \text{independent } B \wedge (\lambda x. -a+x) \text{ ' } V \subseteq \text{span } B$
using *assms*
by (*blast intro: maximal_independent_subset_extend[OF _ h0, of (\lambda x. -a + x) ' V]*)
define T **where** $T = (\lambda x. a+x) \text{ ' } \text{insert } 0 B$
then have $T = \text{insert } a ((\lambda x. a+x) \text{ ' } B)$
by *auto*
then have $\text{affine hull } T = (\lambda x. a+x) \text{ ' } \text{span } B$
using *affine_hull_insert_span_gen[of a ((\lambda x. a+x) ' B)] translation_assoc[of -a a B]*
by *auto*
then have $V \subseteq \text{affine hull } T$
using *B assms translation_inverse_subset[of a V span B]*
by *auto*
moreover have $T \subseteq V$
using *T_def B a assms by auto*
ultimately have $\text{affine hull } T = \text{affine hull } V$
by (*metis Int_absorb1 Int_absorb2 hull_hull hull_mono*)
moreover have $S \subseteq T$
using *T_def B translation_inverse_subset[of a S - {a} B]*
by *auto*
moreover have $\neg \text{affine_dependent } T$
using *T_def affine_dependent_translation_eq[of insert 0 B] affine_dependent_imp_dependent2 B*
by *auto*
ultimately show *?thesis* **using** $\langle T \subseteq V \rangle$ **by** *auto*
qed

lemma *affine_basis_exists*:

fixes $V :: 'n::real_vector\ set$
shows $\exists B. B \subseteq V \wedge \neg \text{affine_dependent } B \wedge \text{affine hull } V = \text{affine hull } B$
by (*metis affine_dependent_def affine_independent_1 empty_subsetI extend_to_affine_basis_nonempty insert_subset order_refl*)

proposition *extend_to_affine_basis*:

fixes $S V :: 'n::real_vector\ set$
assumes $\neg \text{affine_dependent } S \ S \subseteq V$
obtains T **where** $\neg \text{affine_dependent } T \ S \subseteq T \ T \subseteq V \ \text{affine hull } T = \text{affine hull } V$
by (*metis affine_basis_exists assms(1) assms(2) bot.extremum extend_to_affine_basis_nonempty*)

1.6.4 Affine Dimension of a Set

definition *aff_dim* :: $('a::euclidean_space) \ set \Rightarrow \ int$

where $\text{aff_dim } V =$
(SOME $d :: \ int.$
 $\exists B. \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine_dependent } B \wedge \text{of_nat } (\text{card } B)$
 $= d + 1)$

lemma *aff_dim_basis_exists*:

fixes $V :: ('n::euclidean_space) \ set$
shows $\exists B. \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine_dependent } B \wedge \text{of_nat } (\text{card } B) = \text{aff_dim } V + 1$

proof –

obtain B **where** $\neg \text{affine_dependent } B \wedge \text{affine hull } B = \text{affine hull } V$
using *affine_basis_exists[of V]* **by** *auto*
then show *?thesis*
unfolding *aff_dim_def*
 $\text{some_eq_ex}[of \ \lambda d. \ \exists B. \ \text{affine hull } B = \text{affine hull } V \wedge \neg \text{affine_dependent } B \wedge \text{of_nat } (\text{card } B) = d + 1]$
apply *auto*
apply (*rule exI[of _ int (card B) - (1 :: int)]*)
apply (*rule exI[of _ B], auto*)
done

qed

lemma *affine_hull_eq_empty [simp]*: $\text{affine hull } S = \{\} \longleftrightarrow S = \{\}$

by (*metis affine_empty subset_empty subset_hull*)

lemma *empty_eq_affine_hull [simp]*: $\{\} = \text{affine hull } S \longleftrightarrow S = \{\}$

by (*metis affine_hull_eq_empty*)

lemma *aff_dim_parallel_subspace_aux*:

fixes $B :: 'n::euclidean_space \ set$
assumes $\neg \text{affine_dependent } B \ a \in B$
shows $\text{finite } B \wedge ((\text{card } B) - 1 = \text{dim } (\text{span } ((\lambda x. -a+x) \ ' (B-\{a\}))))$

proof –

have *independent* $((\lambda x. -a+x) \ ' (B-\{a\}))$

```

    using affine_dependent_iff_dependent2 assms by auto
  then have fin: dim (span ((λx. -a+x) ‘ (B- $\{a\}$ ))) = card ((λx. -a + x) ‘
(B- $\{a\}$ ))
    finite ((λx. -a + x) ‘ (B -  $\{a\}$ ))
    using indep_card_eq_dim_span[of (λx. -a+x) ‘ (B- $\{a\}$ )] by auto
  show ?thesis
  proof (cases (λx. -a + x) ‘ (B -  $\{a\}$ ) =  $\{\}$ )
    case True
      have B = insert a ((λx. a + x) ‘ (λx. -a + x) ‘ (B -  $\{a\}$ ))
        using translation_assoc[of a -a (B -  $\{a\}$ )] assms by auto
      then have B =  $\{a\}$  using True by auto
      then show ?thesis using assms fin by auto
    next
      case False
        then have card ((λx. -a + x) ‘ (B -  $\{a\}$ )) > 0
          using fin by auto
        moreover have h1: card ((λx. -a + x) ‘ (B- $\{a\}$ )) = card (B- $\{a\}$ )
          by (rule card_image) (use translate_inj_on in blast)
        ultimately have card (B- $\{a\}$ ) > 0 by auto
        then have *: finite (B -  $\{a\}$ )
          using card_gt_0_iff[of (B -  $\{a\}$ )] by auto
        then have card (B -  $\{a\}$ ) = card B - 1
          using card_Diff_singleton assms by auto
        with * show ?thesis using fin h1 by auto
      qed
    qed
  qed

```

lemma *aff_dim_parallel_subspace*:

```

  fixes V L :: 'n::euclidean_space set
  assumes V ≠  $\{\}$ 
  and subspace L
  and affine_parallel (affine hull V) L
  shows aff_dim V = int (dim L)
  proof -
    obtain B where
      B: affine hull B = affine hull V ∧ ¬ affine_dependent B ∧ int (card B) =
aff_dim V + 1
      using aff_dim_basis_exists by auto
    then have B ≠  $\{\}$ 
      using assms B
      by auto
    then obtain a where a: a ∈ B by auto
    define Lb where Lb = span ((λx. -a+x) ‘ (B- $\{a\}$ ))
    moreover have affine_parallel (affine hull B) Lb
      using Lb_def B assms affine_hull_span2[of a B] a
      affine_parallel_commute[of Lb (affine hull B)]
      unfolding affine_parallel_def
      by auto
    moreover have subspace Lb

```

```

    using Lb_def subspace_span by auto
  moreover have affine_hull B  $\neq$  {}
    using assms B by auto
  ultimately have L = Lb
    using assms affine_parallel_subspace[of affine_hull B] affine_affine_hull[of B]
  B
  by auto
  then have dim L = dim Lb
    by auto
  moreover have card B - 1 = dim Lb and finite B
    using Lb_def aff_dim_parallel_subspace_aux a B by auto
  ultimately show ?thesis
    using B  $\langle$  B  $\neq$  {}  $\rangle$  card_gt_0_iff[of B] by auto
qed

```

```

lemma aff_independent_finite:
  fixes B :: 'n::euclidean_space set
  assumes  $\neg$  affine_dependent B
  shows finite B
  using aff_dim_parallel_subspace_aux assms finite_simps by fastforce

```

```

lemma aff_dim_empty:
  fixes S :: 'n::euclidean_space set
  shows S = {}  $\longleftrightarrow$  aff_dim S = -1
proof -
  obtain B where *: affine_hull B = affine_hull S
    and  $\neg$  affine_dependent B
    and int (card B) = aff_dim S + 1
    using aff_dim_basis_exists by auto
  moreover
  from * have S = {}  $\longleftrightarrow$  B = {}
    by auto
  ultimately show ?thesis
    using aff_independent_finite[of B] card_gt_0_iff[of B] by auto
qed

```

```

lemma aff_dim_empty_eq [simp]: aff_dim ({} :: 'a::euclidean_space set) = -1
  using aff_dim_empty by blast

```

```

lemma aff_dim_affine_hull [simp]: aff_dim (affine_hull S) = aff_dim S
  unfolding aff_dim_def using hull_hull[of _ S] by auto

```

```

lemma aff_dim_affine_hull2:
  assumes affine_hull S = affine_hull T
  shows aff_dim S = aff_dim T
  unfolding aff_dim_def using assms by auto

```

```

lemma aff_dim_unique:

```

```

fixes B V :: 'n::euclidean_space set
assumes affine_hull B = affine_hull V  $\wedge$   $\neg$  affine_dependent B
shows of_nat (card B) = aff_dim V + 1
proof (cases B = {})
  case True
    then have V = {}
      using assms
      by auto
    then have aff_dim V = (-1::int)
      using aff_dim_empty by auto
    then show ?thesis
      using  $\langle B = \{\} \rangle$  by auto
  next
    case False
      then obtain a where a: a  $\in$  B by auto
      define Lb where Lb = span (( $\lambda x.$  -a+x) ` (B-{a}))
      have affine_parallel (affine_hull B) Lb
        using Lb_def affine_hull_span2[of a B] a
          affine_parallel_commute[of Lb (affine_hull B)]
        unfolding affine_parallel_def by auto
      moreover have subspace Lb
        using Lb_def subspace_span by auto
      ultimately have aff_dim B = int(dim Lb)
        using aff_dim_parallel_subspace[of B Lb]  $\langle B \neq \{\} \rangle$  by auto
      moreover have (card B) - 1 = dim Lb finite B
        using Lb_def aff_dim_parallel_subspace_aux a assms by auto
      ultimately have of_nat (card B) = aff_dim B + 1
        using  $\langle B \neq \{\} \rangle$  card_gt_0_iff[of B] by auto
      then show ?thesis
        using aff_dim_affine_hull2 assms by auto
qed

```

```

lemma aff_dim_affine_independent:
  fixes B :: 'n::euclidean_space set
  assumes  $\neg$  affine_dependent B
  shows of_nat (card B) = aff_dim B + 1
  using aff_dim_unique[of B B] assms by auto

```

```

lemma affine_independent_iff_card:
  fixes S :: 'a::euclidean_space set
  shows  $\neg$  affine_dependent S  $\iff$  finite S  $\wedge$  aff_dim S = int(card S) - 1 (is
  ?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (simp add: aff_dim_affine_independent affine_independent_finite)
  show ?rhs  $\implies$  ?lhs
    by (metis of_nat_eq_iff affine_basis_exists aff_dim_unique card_subset_eq
  diff_add_cancel)
qed

```



```

lemma aff_dim_sing [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a} = 0
  using aff_dim_affine_independent[of {a}] affine_independent_1 by auto

lemma aff_dim_2 [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a,b} = (if a = b then 0 else 1)
proof (clarsimp)
  assume a ≠ b
  then have aff_dim{a,b} = card{a,b} - 1
    using affine_independent_2 [of a b] aff_dim_affine_independent by fastforce
  also have ... = 1
    using ⟨a ≠ b⟩ by simp
  finally show aff_dim {a, b} = 1 .
qed

lemma aff_dim_inner_basis_exists:
  fixes V :: ('n::euclidean_space) set
  shows ∃ B. B ⊆ V ∧ affine_hull B = affine_hull V ∧
    ¬ affine_dependent B ∧ of_nat (card B) = aff_dim V + 1
  by (metis aff_dim_unique affine_basis_exists)

lemma aff_dim_le_card:
  fixes V :: 'n::euclidean_space set
  assumes finite V
  shows aff_dim V ≤ of_nat (card V) - 1
  by (metis aff_dim_inner_basis_exists assms card_mono le_diff_eq of_nat_le_iff)

lemma aff_dim_parallel_le:
  fixes S T :: 'n::euclidean_space set
  assumes affine_parallel (affine_hull S) (affine_hull T)
  shows aff_dim S ≤ aff_dim T
proof (cases S={} ∨ T={})
  case True
  then show ?thesis
    by (smt (verit, best) aff_dim_affine_hull2 affine_hull_empty affine_parallel_def
    assms empty_is_image)
  next
  case False
  then obtain L where L: subspace L affine_parallel (affine_hull T) L
    by (metis affine_affine_hull affine_hull_eq_empty affine_parallel_subspace)
  with False show ?thesis
    by (metis aff_dim_parallel_subspace affine_parallel_assoc assms dual_order.refl)
qed

lemma aff_dim_parallel_eq:
  fixes S T :: 'n::euclidean_space set

```

```

assumes affine_parallel (affine hull S) (affine hull T)
shows aff_dim S = aff_dim T
by (smt (verit, del_insts) aff_dim_parallel_le affine_parallel_commute assms)

```

```

lemma aff_dim_translation_eq:
  aff_dim ((+) a ' S) = aff_dim S for a :: 'n::euclidean_space
by (metis aff_dim_parallel_eq affine_hull_translation affine_parallel_def)

```

```

lemma aff_dim_translation_eq_subtract:
  aff_dim (( $\lambda$ x. x - a) ' S) = aff_dim S for a :: 'n::euclidean_space
using aff_dim_translation_eq [of - a] by (simp cong: image_cong_simp)

```

```

lemma aff_dim_affine:
  fixes S L :: 'n::euclidean_space set
assumes affine S subspace L affine_parallel S L S  $\neq$  {}
shows aff_dim S = int (dim L)
by (simp add: aff_dim_parallel_subspace assms hull_same)

```

```

lemma dim_affine_hull [simp]:
  fixes S :: 'n::euclidean_space set
shows dim (affine hull S) = dim S
by (metis affine_hull_subset_span dim_eq_span dim_mono hull_subset_span_eq_dim)

```

```

lemma aff_dim_subspace:
  fixes S :: 'n::euclidean_space set
assumes subspace S
shows aff_dim S = int (dim S)
by (metis aff_dim_affine affine_parallel_subspace assms empty_iff parallel_subspace
  subspace_affine)

```

```

lemma aff_dim_zero:
  fixes S :: 'n::euclidean_space set
assumes 0  $\in$  affine hull S
shows aff_dim S = int (dim S)
by (metis aff_dim_affine_hull aff_dim_subspace affine_hull_span_0 assms
  dim_span subspace_span)

```

```

lemma aff_dim_eq_dim:
  fixes S :: 'n::euclidean_space set
assumes a  $\in$  affine hull S
shows aff_dim S = int (dim ((+) (- a) ' S))
by (metis ab_left_minus aff_dim_translation_eq aff_dim_zero affine_hull_translation
  image_eqI assms)

```

```

lemma aff_dim_eq_dim_subtract:
  fixes S :: 'n::euclidean_space set
assumes a  $\in$  affine hull S
shows aff_dim S = int (dim (( $\lambda$ x. x - a) ' S))
using aff_dim_eq_dim assms by auto

```

lemma *aff_dim_UNIV* [simp]: $\text{aff_dim } (UNIV :: 'n::\text{euclidean_space set}) = \text{int}(\text{DIM}('n))$
by (*simp add: aff_dim_subspace*)

lemma *aff_dim_geq*:
fixes $V :: 'n::\text{euclidean_space set}$
shows $\text{aff_dim } V \geq -1$
by (*metis add_le_cancel_right aff_dim_basis_exists diff_self_of_nat_0_le_iff uminus_add_conv_diff*)

lemma *aff_dim_negative_iff* [simp]:
fixes $S :: 'n::\text{euclidean_space set}$
shows $\text{aff_dim } S < 0 \longleftrightarrow S = \{\}$
by (*metis aff_dim_empty aff_dim_geq diff_0_eq_iff zle_diff1_eq*)

lemma *aff_lowdim_subset_hyperplane*:
fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{aff_dim } S < \text{DIM}('a)$
obtains $a \ b$ **where** $a \neq 0 \ S \subseteq \{x. a \cdot x = b\}$
proof (*cases S={}*)
case *True*
moreover
have (*SOME b. b ∈ Basis*) $\neq 0$
by (*metis norm_some_Basis norm_zero zero_neq_one*)
ultimately show *?thesis*
using *that by blast*
next
case *False*
then obtain $c \ S'$ **where** $c \notin S' \ S = \text{insert } c \ S'$
by (*meson equals0I mk_disjoint_insert*)
have $\text{dim } ((+) (-c) ' S) < \text{DIM}('a)$
by (*metis ‹S = insert c S'› aff_dim_eq_dim assms hull_inc insertI1 of_nat_less_imp_less*)
then obtain a **where** $a \neq 0 \ \text{span } ((+) (-c) ' S) \subseteq \{x. a \cdot x = 0\}$
using *lowdim_subset_hyperplane by blast*
moreover
have $a \cdot w = a \cdot c$ **if** $\text{span } ((+) (-c) ' S) \subseteq \{x. a \cdot x = 0\} \ w \in S$ **for** w
proof –
have $w - c \in \text{span } ((+) (-c) ' S)$
by (*simp add: span_base ‹w ∈ S›*)
with *that* **have** $w - c \in \{x. a \cdot x = 0\}$
by *blast*
then show *?thesis*
by (*auto simp: algebra_simps*)
qed
ultimately have $S \subseteq \{x. a \cdot x = a \cdot c\}$
by *blast*
then show *?thesis*
by (*rule that[OF ‹a ≠ 0›]*)
qed

lemma *affine_independent_card_dim_diffs*:

fixes $S :: 'a :: \text{euclidean_space set}$

assumes $\neg \text{affine_dependent } S \ a \in S$

shows $\text{card } S = \text{dim } ((\lambda x. x - a) \text{ ` } S) + 1$

using *aff_dim_affine_independent aff_dim_eq_dim_subtract assms hull_subset*

by *fastforce*

lemma *independent_card_le_aff_dim*:

fixes $B :: 'n :: \text{euclidean_space set}$

assumes $B \subseteq V$

assumes $\neg \text{affine_dependent } B$

shows $\text{int } (\text{card } B) \leq \text{aff_dim } V + 1$

by (*metis aff_dim_unique aff_independent_finite assms card_mono extend_to_affine_basis of_nat_mono*)

lemma *aff_dim_subset*:

fixes $S \ T :: 'n :: \text{euclidean_space set}$

assumes $S \subseteq T$

shows $\text{aff_dim } S \leq \text{aff_dim } T$

by (*metis add_le_cancel_right aff_dim_inner_basis_exists assms dual_order.trans independent_card_le_aff_dim*)

lemma *aff_dim_le_DIM*:

fixes $S :: 'n :: \text{euclidean_space set}$

shows $\text{aff_dim } S \leq \text{int } (\text{DIM } ('n))$

by (*metis aff_dim_UNIV aff_dim_subset top_greatest*)

lemma *affine_dim_equal*:

fixes $S :: 'n :: \text{euclidean_space set}$

assumes $\text{affine } S \ \text{affine } T \ S \neq \{\} \ S \subseteq T \ \text{aff_dim } S = \text{aff_dim } T$

shows $S = T$

proof –

obtain a **where** $a \in S \ a \in T \ T \neq \{\}$ **using** *assms* **by** *auto*

define LS **where** $LS = \{y. \exists x \in S. (-a) + x = y\}$

then **have** ls : *subspace* LS *affine_parallel* $S \ LS$

using *assms parallel_subspace_explicit*[*of* $S \ a \ LS$] $\langle a \in S \rangle$ **by** *auto*

then **have** $h1$: $\text{int } (\text{dim } LS) = \text{aff_dim } S$

using *assms aff_dim_affine*[*of* $S \ LS$] **by** *auto*

define LT **where** $LT = \{y. \exists x \in T. (-a) + x = y\}$

then **have** lt : *subspace* $LT \wedge \text{affine_parallel } T \ LT$

using *assms parallel_subspace_explicit*[*of* $T \ a \ LT$] $\langle a \in T \rangle$ **by** *auto*

then **have** $\text{dim } LS = \text{dim } LT$

using *assms aff_dim_affine*[*of* $T \ LT$] $\langle T \neq \{\} \rangle \ h1$ **by** *auto*

moreover **have** $LS \leq LT$

using $LS_def \ LT_def$ *assms* **by** *auto*

ultimately **have** $LS = LT$

using *subspace_dim_equal*[*of* $LS \ LT$] $ls \ lt$ **by** *auto*

moreover **have** $S = \{x. \exists y \in LS. a+y=x\} \ T = \{x. \exists y \in LT. a+y=x\}$

```

    using LS_def LT_def by auto
    ultimately show ?thesis by auto
qed

```

```

lemma aff_dim_eq_0:
  fixes S :: 'a::euclidean_space set
  shows aff_dim S = 0  $\longleftrightarrow$  ( $\exists a. S = \{a\}$ )
proof (cases S = {})
  case False
  then obtain a where a  $\in$  S by auto
  show ?thesis
  proof safe
    assume 0: aff_dim S = 0
    have  $\neg \{a,b\} \subseteq S$  if b  $\neq$  a for b
      by (metis 0 aff_dim_2 aff_dim_subset not_one_le_zero that)
    then show  $\exists a. S = \{a\}$ 
      using  $\langle a \in S \rangle$  by blast
  qed auto
qed auto

```

```

lemma affine_hull_UNIV:
  fixes S :: 'n::euclidean_space set
  assumes aff_dim S = int(DIM('n))
  shows affine hull S = (UNIV :: ('n::euclidean_space) set)
  by (simp add: aff_dim_empty affine_dim_equal assms)

```

```

lemma disjoint_affine_hull:
  fixes S :: 'n::euclidean_space set
  assumes  $\neg$  affine_dependent S T  $\subseteq$  S U  $\subseteq$  S T  $\cap$  U = {}
  shows (affine hull T)  $\cap$  (affine hull U) = {}
proof -
  obtain finite S finite T finite U
  using assms by (simp add: aff_independent_finite finite_subset)
  have False if y  $\in$  affine hull T and y  $\in$  affine hull U for y
  proof -
    from that obtain a b
    where a1 [simp]: sum a T = 1
      and [simp]: sum ( $\lambda v. a v *_{\mathbb{R}} v$ ) T = y
      and [simp]: sum b U = 1 sum ( $\lambda v. b v *_{\mathbb{R}} v$ ) U = y
    by (auto simp: affine_hull_finite  $\langle$ finite T $\rangle$   $\langle$ finite U $\rangle$ )
    define c where c x = (if x  $\in$  T then a x else if x  $\in$  U then -(b x) else 0) for
x
  have [simp]: S  $\cap$  T = T S  $\cap$  - T  $\cap$  U = U
    using assms by auto
  have sum c S = 0
  by (simp add: c_def comm_monoid_add_class.sum.If_cases  $\langle$ finite S $\rangle$  sum_negf)
  moreover have  $\neg (\forall v \in S. c v = 0)$ 
  by (metis (no_types) IntD1  $\langle$ S  $\cap$  T = T $\rangle$  a1 c_def sum.neutral zero_neq_one)
  moreover have ( $\sum v \in S. c v *_{\mathbb{R}} v$ ) = 0

```

```

    by (simp add: c_def if_smult sum_negf comm_monoid_add_class.sum.If_cases
        ⟨finite S⟩)
    ultimately show ?thesis
      using assms(1) ⟨finite S⟩ by (auto simp: affine_dependent_explicit)
    qed
    then show ?thesis by blast
  qed
end

```

1.7 Convex Sets and Functions

```

theory Convex
imports
  Affine HOL-Library.Set_Algebras HOL-Library.FuncSet
begin

```

1.7.1 Convex Sets

```

definition convex :: 'a::real_vector set ⇒ bool
  where convex s ⇔ (∀ x∈s. ∀ y∈s. ∀ u≥0. ∀ v≥0. u + v = 1 ⟶ u *R x + v
    *R y ∈ s)

```

```

lemma convexI:
  assumes ∧x y u v. x ∈ s ⟹ y ∈ s ⟹ 0 ≤ u ⟹ 0 ≤ v ⟹ u + v = 1 ⟹
    u *R x + v *R y ∈ s
  shows convex s
  by (simp add: assms convex_def)

```

```

lemma convexD:
  assumes convex s and x ∈ s and y ∈ s and 0 ≤ u and 0 ≤ v and u + v = 1
  shows u *R x + v *R y ∈ s
  using assms unfolding convex_def by fast

```

```

lemma convex_alt: convex s ⇔ (∀ x∈s. ∀ y∈s. ∀ u. 0 ≤ u ∧ u ≤ 1 ⟶ ((1 -
    u) *R x + u *R y) ∈ s)
  (is _ ⇔ ?alt)
  by (smt (verit) convexD convexI)

```

```

lemma convexD_alt:
  assumes convex s a ∈ s b ∈ s 0 ≤ u u ≤ 1
  shows ((1 - u) *R a + u *R b) ∈ s
  using assms unfolding convex_alt by auto

```

```

lemma mem_convex_alt:
  assumes convex S x ∈ S y ∈ S u ≥ 0 v ≥ 0 u + v > 0
  shows ((u/(u+v)) *R x + (v/(u+v)) *R y) ∈ S
  using assms
  by (simp add: convex_def zero_le_divide_iff add_divide_distrib [symmetric])

```

```

lemma convex_empty[intro,simp]: convex {}
  unfolding convex_def by simp

lemma convex_singleton[intro,simp]: convex {a}
  unfolding convex_def by (auto simp: scaleR_left_distrib[symmetric])

lemma convex_UNIV[intro,simp]: convex UNIV
  unfolding convex_def by auto

lemma convex_Inter: ( $\bigwedge s. s \in f \implies \text{convex } s$ )  $\implies \text{convex}(\bigcap f)$ 
  unfolding convex_def by auto

lemma convex_Int:  $\text{convex } s \implies \text{convex } t \implies \text{convex } (s \cap t)$ 
  unfolding convex_def by auto

lemma convex_INT: ( $\bigwedge i. i \in A \implies \text{convex } (B \ i)$ )  $\implies \text{convex } (\bigcap_{i \in A} B \ i)$ 
  unfolding convex_def by auto

lemma convex_Times:  $\text{convex } s \implies \text{convex } t \implies \text{convex } (s \times t)$ 
  unfolding convex_def by auto

lemma convex_halfspace_le: convex {x. inner a x  $\leq$  b}
  unfolding convex_def
  by (auto simp: inner_add intro!: convex_bound_le)

lemma convex_halfspace_ge: convex {x. inner a x  $\geq$  b}
proof -
  have *: {x. inner a x  $\geq$  b} = {x. inner (-a) x  $\leq$  -b}
    by auto
  show ?thesis
    unfolding * using convex_halfspace_le[of -a -b] by auto
qed

lemma convex_halfspace_abs_le: convex {x. |inner a x|  $\leq$  b}
proof -
  have *: {x. |inner a x|  $\leq$  b} = {x. inner a x  $\leq$  b}  $\cap$  {x. -b  $\leq$  inner a x}
    by auto
  show ?thesis
    unfolding * by (simp add: convex_Int convex_halfspace_ge convex_halfspace_le)
qed

lemma convex_hyperplane: convex {x. inner a x = b}
proof -
  have *: {x. inner a x = b} = {x. inner a x  $\leq$  b}  $\cap$  {x. inner a x  $\geq$  b}
    by auto
  show ?thesis using convex_halfspace_le convex_halfspace_ge
    by (auto intro!: convex_Int simp: *)
qed

```

```

lemma convex_halfspace_lt: convex {x. inner a x < b}
  unfolding convex_def
  by (auto simp: convex_bound_lt inner_add)

lemma convex_halfspace_gt: convex {x. inner a x > b}
  using convex_halfspace_lt[of -a -b] by auto

lemma convex_halfspace_Re_ge: convex {x. Re x ≥ b}
  using convex_halfspace_ge[of b 1::complex] by simp

lemma convex_halfspace_Re_le: convex {x. Re x ≤ b}
  using convex_halfspace_le[of 1::complex b] by simp

lemma convex_halfspace_Im_ge: convex {x. Im x ≥ b}
  using convex_halfspace_ge[of b i] by simp

lemma convex_halfspace_Im_le: convex {x. Im x ≤ b}
  using convex_halfspace_le[of i b] by simp

lemma convex_halfspace_Re_gt: convex {x. Re x > b}
  using convex_halfspace_gt[of b 1::complex] by simp

lemma convex_halfspace_Re_lt: convex {x. Re x < b}
  using convex_halfspace_lt[of 1::complex b] by simp

lemma convex_halfspace_Im_gt: convex {x. Im x > b}
  using convex_halfspace_gt[of b i] by simp

lemma convex_halfspace_Im_lt: convex {x. Im x < b}
  using convex_halfspace_lt[of i b] by simp

lemma convex_real_interval [iff]:
  fixes a b :: real
  shows convex {a..} and convex {..b}
    and convex {a<..} and convex {..<b}
    and convex {a..b} and convex {a<..b}
    and convex {a..and convex {a<..proof -
  have {a..} = {x. a ≤ inner 1 x}
    by auto
  then show 1: convex {a..}
    by (simp only: convex_halfspace_ge)
  have {..b} = {x. inner 1 x ≤ b}
    by auto
  then show 2: convex {..b}
    by (simp only: convex_halfspace_le)
  have {a<..} = {x. a < inner 1 x}
    by auto

```



```

then show 3: convex {a<..}
  by (simp only: convex_halfspace_gt)
have {..<b} = {x. inner 1 x < b}
  by auto
then show 4: convex {..<b}
  by (simp only: convex_halfspace_lt)
have {a..>} = {a..>} ∩ {..<b}
  by auto
then show convex {a..>}
  by (simp only: convex_Int 1 2)
have {a<..>} = {a<..>} ∩ {..>}
  by auto
then show convex {a<..>}
  by (simp only: convex_Int 3 2)
have {a..<b} = {a..<} ∩ {..<b}
  by auto
then show convex {a..<b}
  by (simp only: convex_Int 1 4)
have {a<..<b} = {a<..>} ∩ {..<b}
  by auto
then show convex {a<..<b}
  by (simp only: convex_Int 3 4)
qed

```

```

lemma convex_Reals: convex ℝ
  by (simp add: convex_def scaleR_conv_of_real)

```

1.7.2 Explicit expressions for convexity in terms of arbitrary sums

```

lemma convex_sum:
  fixes C :: 'a::real_vector set
  assumes finite S
  and convex C
  and a: (∑ i ∈ S. a i) = 1 ∧ i. i ∈ S ⇒ a i ≥ 0
  and C: ∧i. i ∈ S ⇒ y i ∈ C
  shows (∑ j ∈ S. a j *R y j) ∈ C
  using ⟨finite S⟩ a C
proof (induction arbitrary: a set: finite)
  case empty
  then show ?case by simp
next
  case (insert i S)
  then have 0 ≤ sum a S
  by (simp add: sum_nonneg)
  have a i *R y i + (∑ j ∈ S. a j *R y j) ∈ C
  proof (cases sum a S = 0)
  case True with insert show ?thesis
  by (simp add: sum_nonneg_eq_0_iff)

```

```

next
  case False
  with ⟨0 ≤ sum a S⟩ have 0 < sum a S
  by simp
  then have (∑ j ∈ S. (a j / sum a S) *R y j) ∈ C
  using insert
  by (simp add: insert.IH flip: sum_divide_distrib)
  with ⟨convex C⟩ insert ⟨0 ≤ sum a S⟩
  have a i *R y i + sum a S *R (∑ j ∈ S. (a j / sum a S) *R y j) ∈ C
  by (simp add: convex_def)
  then show ?thesis
  by (simp add: scaleR_sum_right False)
qed
then show ?case using ⟨finite S⟩ and ⟨i ∉ S⟩
  by simp
qed

lemma convex:
  convex S ↔
    (∀ (k :: nat) u x. (∀ i. 1 ≤ i ∧ i ≤ k → 0 ≤ u i ∧ x i ∈ S) ∧ (sum u {1..k} = 1)
    → sum (λ i. u i *R x i) {1..k} ∈ S)
  (is ?lhs = ?rhs)
proof
  show ?lhs ⇒ ?rhs
  by (metis (full_types) atLeastAtMost_iff convex_sum finite_atLeastAtMost)
  assume *: ∀ k u x. (∀ i :: nat. 1 ≤ i ∧ i ≤ k → 0 ≤ u i ∧ x i ∈ S) ∧ sum u
  {1..k} = 1
  → (∑ i = 1..k. u i *R (x i :: 'a)) ∈ S
  {
  fix μ :: real
  fix x y :: 'a
  assume xy: x ∈ S y ∈ S
  assume mu: μ ≥ 0 μ ≤ 1
  let ?u = λ i. if (i :: nat) = 1 then μ else 1 - μ
  let ?x = λ i. if (i :: nat) = 1 then x else y
  have {1 :: nat .. 2} ∩ - {x. x = 1} = {2}
  by auto
  then have S: (∑ j ∈ {1..2}. ?u j *R ?x j) ∈ S
  using sum.If_cases[of {(1 :: nat) .. 2} λ x. x = 1 λ x. μ λ x. 1 - μ]
  using mu xy * by auto
  have grarr: (∑ j ∈ {Suc (Suc 0)..2}. ?u j *R ?x j) = (1 - μ) *R y
  using sum.atLeast_Suc_atMost[of Suc (Suc 0) 2 λ j. (1 - μ) *R y] by auto
  with sum.atLeast_Suc_atMost
  have (∑ j ∈ {1..2}. ?u j *R ?x j) = μ *R x + (1 - μ) *R y
  by (smt (verit, best) Suc_1 Suc_eq_plus1 add_0 le_add1)
  then have (1 - μ) *R y + μ *R x ∈ S
  using S by (auto simp: add commute)
  }
}
then show convex S

```

unfolding *convex_alt* **by** *auto*
qed

lemma *convex_explicit*:

fixes $S :: 'a::\text{real_vector_set}$

shows $\text{convex } S \longleftrightarrow$

$(\forall t u. \text{finite } t \wedge t \subseteq S \wedge (\forall x \in t. 0 \leq u x) \wedge \text{sum } u t = 1 \longrightarrow \text{sum } (\lambda x. u x *_{\mathbb{R}} x) t \in S)$

proof *safe*

fix t

fix $u :: 'a \Rightarrow \text{real}$

assume $\text{convex } S$

and $\text{finite } t$

and $t \subseteq S \ \forall x \in t. 0 \leq u x \ \text{sum } u t = 1$

then show $(\sum x \in t. u x *_{\mathbb{R}} x) \in S$

by (*simp add: convex_sum subsetD*)

next

assume $*: \forall t. \forall u. \text{finite } t \wedge t \subseteq S \wedge (\forall x \in t. 0 \leq u x) \wedge \text{sum } u t = 1 \longrightarrow (\sum x \in t. u x *_{\mathbb{R}} x) \in S$

show $\text{convex } S$

unfolding *convex_alt*

proof *safe*

fix $x y$

fix $\mu :: \text{real}$

assume $*: x \in S \ y \in S \ 0 \leq \mu \ \mu \leq 1$

show $(1 - \mu) *_{\mathbb{R}} x + \mu *_{\mathbb{R}} y \in S$

proof (*cases* $x = y$)

case *False*

then show *?thesis*

using $*[\text{rule_format}, \text{of } \{x, y\} \ \lambda z. \text{if } z = x \text{ then } 1 - \mu \text{ else } \mu] **$

by *auto*

next

case *True*

then show *?thesis*

using $*[\text{rule_format}, \text{of } \{x, y\} \ \lambda z. 1] **$

by (*auto simp: field_simps real_vector.scale_left_diff_distrib*)

qed

qed

qed

lemma *convex_finite*:

assumes $\text{finite } S$

shows $\text{convex } S \longleftrightarrow (\forall u. (\forall x \in S. 0 \leq u x) \wedge \text{sum } u S = 1 \longrightarrow \text{sum } (\lambda x. u x *_{\mathbb{R}} x) S \in S)$

(**is** *?lhs = ?rhs*)

proof

{ **have** *if_distrib_arg*: $\bigwedge P f g x. (\text{if } P \text{ then } f \text{ else } g) x = (\text{if } P \text{ then } f x \text{ else } g x)$

by *simp*

```

fix T :: 'a set and u :: 'a ⇒ real
assume sum: ∀ u. (∀ x ∈ S. 0 ≤ u x) ∧ sum u S = 1 ⟶ (∑ x ∈ S. u x *R x) ∈
S
assume *: ∀ x ∈ T. 0 ≤ u x sum u T = 1
assume T ⊆ S
then have S ∩ T = T by auto
with sum[THEN spec[where x = λ x. if x ∈ T then u x else 0]] *
have (∑ x ∈ T. u x *R x) ∈ S
by (auto simp: assms sum.If_cases if_distrib if_distrib_arg) }
moreover assume ?rhs
ultimately show ?lhs
unfolding convex_explicit by auto
qed (auto simp: convex_explicit assms)

```

1.7.3 Convex Functions on a Set

```

definition convex_on :: 'a::real_vector set ⇒ ('a ⇒ real) ⇒ bool
where convex_on S f ⟷ convex S ∧
(∀ x ∈ S. ∀ y ∈ S. ∀ u ≥ 0. ∀ v ≥ 0. u + v = 1 ⟶ f (u *R x + v *R y) ≤ u * f x
+ v * f y)

```

```

definition concave_on :: 'a::real_vector set ⇒ ('a ⇒ real) ⇒ bool
where concave_on S f ≡ convex_on S (λ x. - f x)

```

```

lemma convex_on_iff_concave: convex_on S f = concave_on S (λ x. - f x)
by (simp add: concave_on_def)

```

```

lemma concave_on_iff:
concave_on S f ⟷ convex S ∧
(∀ x ∈ S. ∀ y ∈ S. ∀ u ≥ 0. ∀ v ≥ 0. u + v = 1 ⟶ f (u *R x + v *R y) ≥ u * f x
+ v * f y)
by (auto simp: concave_on_def convex_on_def algebra_simps)

```

```

lemma concave_onD:
assumes concave_on A f
shows ∧ t x y. t ≥ 0 ⟹ t ≤ 1 ⟹ x ∈ A ⟹ y ∈ A ⟹
f ((1 - t) *R x + t *R y) ≥ (1 - t) * f x + t * f y
using assms by (auto simp: concave_on_iff)

```

```

lemma convex_onI [intro?]:
assumes ∧ t x y. t > 0 ⟹ t < 1 ⟹ x ∈ A ⟹ y ∈ A ⟹
f ((1 - t) *R x + t *R y) ≤ (1 - t) * f x + t * f y
and convex A
shows convex_on A f
unfolding convex_on_def
by (smt (verit, del_insts) assms mult_cancel_right1 mult_eq_0_iff scaleR_collapse
scaleR_eq_0_iff)

```

```

lemma convex_onD:

```

```

assumes convex_on A f
shows  $\bigwedge t x y. t \geq 0 \implies t \leq 1 \implies x \in A \implies y \in A \implies$ 
   $f ((1 - t) *_R x + t *_R y) \leq (1 - t) * f x + t * f y$ 
using assms by (auto simp: convex_on_def)

```

```

lemma convex_on_linorderI [intro?]:
fixes A :: ('a::{\linorder,real_vector}) set
assumes  $\bigwedge t x y. t > 0 \implies t < 1 \implies x \in A \implies y \in A \implies x < y \implies$ 
   $f ((1 - t) *_R x + t *_R y) \leq (1 - t) * f x + t * f y$ 
  and convex A
shows convex_on A f
by (smt (verit, best) add.commute assms convex_onI distrib_left linorder_cases
  mult.commute mult_cancel_left2 scaleR_collapse)

```

```

lemma concave_on_linorderI [intro?]:
fixes A :: ('a::{\linorder,real_vector}) set
assumes  $\bigwedge t x y. t > 0 \implies t < 1 \implies x \in A \implies y \in A \implies x < y \implies$ 
   $f ((1 - t) *_R x + t *_R y) \geq (1 - t) * f x + t * f y$ 
  and convex A
shows concave_on A f
by (smt (verit) assms concave_on_def convex_on_linorderI mult_minus_right)

```

```

lemma convex_on_imp_convex: convex_on A f  $\implies$  convex A
by (auto simp: convex_on_def)

```

```

lemma concave_on_imp_convex: concave_on A f  $\implies$  convex A
by (simp add: concave_on_def convex_on_imp_convex)

```

```

lemma convex_onD_Icc:
assumes convex_on {x..y} f  $x \leq (y :: \_ :: \{real\_vector,preorder\})$ 
shows  $\bigwedge t. t \geq 0 \implies t \leq 1 \implies$ 
   $f ((1 - t) *_R x + t *_R y) \leq (1 - t) * f x + t * f y$ 
using assms(2) by (intro convex_onD [OF assms(1)]) simp_all

```

```

lemma convex_on_subset:  $\llbracket$ convex_on T f;  $S \subseteq T$ ; convex S $\rrbracket \implies$  convex_on S
f
by (simp add: convex_on_def subset_iff)

```

```

lemma convex_on_add [intro]:
assumes convex_on S f
  and convex_on S g
shows convex_on S ( $\lambda x. f x + g x$ )

```

```

proof -
{
  fix x y
  assume  $x \in S y \in S$ 
  moreover
  fix u v :: real
  assume  $0 \leq u \leq v u + v = 1$ 

```

```

ultimately
  have  $f (u *_R x + v *_R y) + g (u *_R x + v *_R y) \leq (u * f x + v * f y) + (u$ 
 $* g x + v * g y)$ 
    using assms unfolding convex_on_def by (auto simp: add_mono)
  then have  $f (u *_R x + v *_R y) + g (u *_R x + v *_R y) \leq u * (f x + g x) + v$ 
 $* (f y + g y)$ 
    by (simp add: field_simps)
  }
with assms show ?thesis
  unfolding convex_on_def by auto
qed

```

```

lemma convex_on_ident: convex_on  $S (\lambda x. x) \longleftrightarrow \text{convex } S$ 
by (simp add: convex_on_def)

```

```

lemma concave_on_ident: concave_on  $S (\lambda x. x) \longleftrightarrow \text{convex } S$ 
by (simp add: concave_on_iff)

```

```

lemma convex_on_const: convex_on  $S (\lambda x. a) \longleftrightarrow \text{convex } S$ 
by (simp add: convex_on_def flip: distrib_right)

```

```

lemma concave_on_const: concave_on  $S (\lambda x. a) \longleftrightarrow \text{convex } S$ 
by (simp add: concave_on_iff flip: distrib_right)

```

```

lemma convex_on_diff:
  assumes convex_on  $S f$  and concave_on  $S g$ 
  shows convex_on  $S (\lambda x. f x - g x)$ 
  using assms concave_on_def convex_on_add by fastforce

```

```

lemma concave_on_diff:
  assumes concave_on  $S f$ 
  and convex_on  $S g$ 
  shows concave_on  $S (\lambda x. f x - g x)$ 
  using convex_on_diff assms concave_on_def by fastforce

```

```

lemma concave_on_add:
  assumes concave_on  $S f$ 
  and concave_on  $S g$ 
  shows concave_on  $S (\lambda x. f x + g x)$ 
  using assms convex_on_iff_concave concave_on_diff concave_on_def by fast-
force

```

```

lemma convex_on_mul:
  fixes  $S::\text{real set}$ 
  assumes convex_on  $S f$  convex_on  $S g$ 
  assumes mono_on  $S f$  mono_on  $S g$ 
  assumes fty:  $f \in S \rightarrow \{0..\}$  and gty:  $g \in S \rightarrow \{0..\}$ 
  shows convex_on  $S (\lambda x. f x * g x)$ 
proof (intro convex_on_linorderI)

```

```

show convex S
  using assms convex_on_imp_convex by auto
fix t::real and x y
assume t: 0 < t t < 1 and xy: x ∈ S y ∈ S x < y
have *: t*(1-t) * f x * g y + t*(1-t) * f y * g x ≤ t*(1-t) * f x * g x +
t*(1-t) * f y * g y
  using t ⟨mono_on S f⟩ ⟨mono_on S g⟩ xy
  by (smt (verit, ccfv_SIG) left_diff_distrib mono_onD mult_left_less_imp_less
zero_le_mult_iff)
have inS: (1-t)*x + t*y ∈ S
  using t xy ⟨convex S⟩ by (simp add: convex_alt)
then have f ((1-t)*x + t*y) * g ((1-t)*x + t*y) ≤ ((1-t) * f x + t * f y)*g
((1-t)*x + t*y)
  using convex_onD [OF ⟨convex_on S f⟩, of t x y] t xy fty gty
  by (intro mult_mono add_nonneg_nonneg) (auto simp: Pi_iff zero_le_mult_iff)
also have ... ≤ ((1-t) * f x + t * f y) * ((1-t)*g x + t*g y)
  using convex_onD [OF ⟨convex_on S g⟩, of t x y] t xy fty gty inS
  by (intro mult_mono add_nonneg_nonneg) (auto simp: Pi_iff zero_le_mult_iff)
also have ... ≤ (1-t) * (f x * g x) + t * (f y * g y)
  using * by (simp add: algebra_simps)
finally show f ((1-t) *R x + t *R y) * g ((1-t) *R x + t *R y) ≤ (1-t)*(f
x*g x) + t*(f y*g y)
  by simp
qed

```

```

lemma convex_on_cmul [intro]:
  fixes c :: real
  assumes 0 ≤ c
  and convex_on S f
  shows convex_on S (λx. c * f x)
proof -
  have *: u * (c * fx) + v * (c * fy) = c * (u * fx + v * fy)
  for u c fx v fy :: real
  by (simp add: field_simps)
  show ?thesis using assms(2) and mult_left_mono [OF __ assms(1)]
  unfolding convex_on_def and * by auto
qed

```

```

lemma convex_on_cdiv [intro]:
  fixes c :: real
  assumes 0 ≤ c and convex_on S f
  shows convex_on S (λx. f x / c)
  unfolding divide_inverse
  using convex_on_cmul [of inverse c S f] by (simp add: mult commute assms)

```

```

lemma convex_lower:
  assumes convex_on S f
  and x ∈ S
  and y ∈ S

```

```

    and 0 ≤ u
    and 0 ≤ v
    and u + v = 1
  shows f (u *R x + v *R y) ≤ max (f x) (f y)
proof -
  let ?m = max (f x) (f y)
  have u * f x + v * f y ≤ u * max (f x) (f y) + v * max (f x) (f y)
    using assms(4,5) by (auto simp: mult_left_mono add_mono)
  also have ... = max (f x) (f y)
    using assms(6) by (simp add: distrib_right [symmetric])
  finally show ?thesis
    using assms unfolding convex_on_def by fastforce
qed

```

```

lemma convex_on_dist [intro]:
  fixes S :: 'a::real_normed_vector set
  assumes convex S
  shows convex_on S (λx. dist a x)
unfolding convex_on_def dist_norm
proof (intro conjI strip)
  fix x y
  assume x ∈ S y ∈ S
  fix u v :: real
  assume 0 ≤ u
  assume 0 ≤ v
  assume u + v = 1
  have a = u *R a + v *R a
    by (metis ⟨u + v = 1⟩ scaleR_left.add scaleR_one)
  then have a - (u *R x + v *R y) = (u *R (a - x)) + (v *R (a - y))
    by (auto simp: algebra_simps)
  then show norm (a - (u *R x + v *R y)) ≤ u * norm (a - x) + v * norm (a - y)
    by (smt (verit, best) ⟨0 ≤ u⟩ ⟨0 ≤ v⟩ norm_scaleR norm_triangle_ineq)
qed (use assms in auto)

```

```

lemma concave_on_mul:
  fixes S::real set
  assumes f: concave_on S f and g: concave_on S g
  assumes mono_on S f antimono_on S g
  assumes fty: f ∈ S → {0..} and gty: g ∈ S → {0..}
  shows concave_on S (λx. f x * g x)
proof (intro concave_on_linorderI)
  show convex S
    using concave_on_imp_convex f by blast
  fix t::real and x y
  assume t: 0 < t < 1 and xy: x ∈ S y ∈ S x < y
  have inS: (1-t)*x + t*y ∈ S
    using t xy ⟨convex S⟩ by (simp add: convex_alt)
  have f x * g y + f y * g x ≥ f x * g x + f y * g y

```



```

  using ⟨mono_on S f⟩ ⟨antimono_on S g⟩
  unfolding monotone_on_def by (smt (verit, best) left_diff_distrib mult_left_mono
xy)
  with t have *: t*(1-t) * f x * g y + t*(1-t) * f y * g x ≥ t*(1-t) * f x * g x
+ t*(1-t) * f y * g y
  by (smt (verit, ccfv_SIG) distrib_left mult_left_mono diff_ge_0_iff_ge mult.assoc)
  have (1 - t) * (f x * g x) + t * (f y * g y) ≤ ((1-t) * f x + t * f y) * ((1-t)
* g x + t * g y)
  using * by (simp add: algebra_simps)
  also have ... ≤ ((1-t) * f x + t * f y)*g ((1-t)*x + t*y)
  using concave_onD [OF ⟨concave_on S g⟩, of t x y] t xy fty gty inS
  by (intro mult_mono add_nonneg_nonneg) (auto simp: Pi_iff zero_le_mult_iff)
  also have ... ≤ f ((1-t)*x + t*y) * g ((1-t)*x + t*y)
  using concave_onD [OF ⟨concave_on S f⟩, of t x y] t xy fty gty inS
  by (intro mult_mono add_nonneg_nonneg) (auto simp: Pi_iff zero_le_mult_iff)
  finally show (1 - t) * (f x * g x) + t * (f y * g y)
    ≤ f ((1 - t) *R x + t *R y) * g ((1 - t) *R x + t *R y)
    by simp
qed

```

```

lemma concave_on_cmul [intro]:
  fixes c :: real
  assumes 0 ≤ c and concave_on S f
  shows concave_on S (λx. c * f x)
  using assms convex_on_cmul [of c S λx. - f x]
  by (auto simp: concave_on_def)

```

```

lemma concave_on_cdiv [intro]:
  fixes c :: real
  assumes 0 ≤ c and concave_on S f
  shows concave_on S (λx. f x / c)
  unfolding divide_inverse
  using concave_on_cmul [of inverse c S f] by (simp add: mult.commute assms)

```

1.7.4 Arithmetic operations on sets preserve convexity

```

lemma convex_linear_image:
  assumes linear f and convex S
  shows convex (f ` S)
proof -
  interpret f: linear f by fact
  from ⟨convex S⟩ show convex (f ` S)
    by (simp add: convex_def f.scaleR [symmetric] f.add [symmetric])
qed

```

```

lemma convex_linear_vimage:
  assumes linear f and convex S
  shows convex (f -1 S)
proof -

```

interpret f : *linear* f **by fact**
from $\langle \text{convex } S \rangle$ **show** $\text{convex } (f - ' S)$
by (*simp add: convex_def f.add f.scaleR*)
qed

lemma *convex_scaling*:
assumes $\text{convex } S$
shows $\text{convex } ((\lambda x. c *_{\mathbb{R}} x) ' S)$
by (*simp add: assms convex_linear_image*)

lemma *convex_scaled*:
assumes $\text{convex } S$
shows $\text{convex } ((\lambda x. x *_{\mathbb{R}} c) ' S)$
by (*simp add: assms convex_linear_image*)

lemma *convex_negations*:
assumes $\text{convex } S$
shows $\text{convex } ((\lambda x. - x) ' S)$
by (*simp add: assms convex_linear_image module_hom_uminus*)

lemma *convex_sums*:
assumes $\text{convex } S$
and $\text{convex } T$
shows $\text{convex } (\bigcup x \in S. \bigcup y \in T. \{x + y\})$
proof –
have *linear* $(\lambda(x, y). x + y)$
by (*auto intro: linearI simp: scaleR_add_right*)
with *assms* **have** $\text{convex } ((\lambda(x, y). x + y) ' (S \times T))$
by (*intro convex_linear_image convex_Times*)
also **have** $((\lambda(x, y). x + y) ' (S \times T)) = (\bigcup x \in S. \bigcup y \in T. \{x + y\})$
by *auto*
finally **show** *?thesis* .
qed

lemma *convex_differences*:
assumes $\text{convex } S$ $\text{convex } T$
shows $\text{convex } (\bigcup x \in S. \bigcup y \in T. \{x - y\})$
proof –
have $\{x - y \mid x y. x \in S \wedge y \in T\} = \{x + y \mid x y. x \in S \wedge y \in \text{uminus } ' T\}$
by (*auto simp: diff_conv_add_uminus simp del: add_uminus_conv_diff*)
then **show** *?thesis*
using *convex_sums[OF assms(1) convex_negations[OF assms(2)]]* **by** *auto*
qed

lemma *convex_translation*:
 $\text{convex } ((+) a ' S)$ **if** $\text{convex } S$
proof –
have $(\bigcup x \in \{a\}. \bigcup y \in S. \{x + y\}) = (+) a ' S$
by *auto*

```

then show ?thesis
  using convex_sums [OF convex_singleton [of a] that] by auto
qed

```

```

lemma convex_translation_subtract:
  convex (( $\lambda b. b - a$ ) ' S) if convex S
  using convex_translation [of S - a] that by (simp cong: image_cong_simp)

```

```

lemma convex_affinity:
  assumes convex S
  shows convex (( $\lambda x. a + c *_R x$ ) ' S)
proof -
  have ( $\lambda x. a + c *_R x$ ) ' S = (+) a ' (*_R) c ' S
  by auto
  then show ?thesis
  using convex_translation[OF convex_scaling[OF assms], of a c] by auto
qed

```

```

lemma convex_on_sum:
  fixes a :: 'a  $\Rightarrow$  real
  and y :: 'a  $\Rightarrow$  'b::real_vector
  and f :: 'b  $\Rightarrow$  real
  assumes finite S S  $\neq$  {}
  and convex_on C f
  and ( $\sum i \in S. a i$ ) = 1
  and  $\bigwedge i. i \in S \implies a i \geq 0$ 
  and  $\bigwedge i. i \in S \implies y i \in C$ 
  shows f ( $\sum i \in S. a i *_R y i$ )  $\leq$  ( $\sum i \in S. a i * f (y i)$ )
  using assms
proof (induct S arbitrary: a rule: finite_ne_induct)
  case (singleton i)
  then show ?case
  by auto
next
  case (insert i S)
  then have  $y i \in C$   $a i \geq 0$ 
  by auto
  with insert have conv:  $\bigwedge x y \mu. x \in C \implies y \in C \implies 0 \leq \mu \implies \mu \leq 1 \implies$ 
    f ( $\mu *_R x + (1 - \mu) *_R y$ )  $\leq \mu * f x + (1 - \mu) * f y$ 
  by (simp add: convex_on_def)
  show ?case
proof (cases a i = 1)
  case True
  with insert have ( $\sum j \in S. a j$ ) = 0
  by auto
  with insert show ?thesis
  by (simp add: sum_nonneg_eq_0_iff)
next
  case False

```

```

then have ai1: a i < 1
  using sum_nonneg_leq_bound[of insert i S a] insert by force
then have i0: 1 - a i > 0
  by auto
let ?a = λj. a j / (1 - a i)
have a_nonneg: ?a j ≥ 0 if j ∈ S for j
  using i0 insert that by fastforce
have (∑ j ∈ insert i S. a j) = 1
  using insert by auto
then have (∑ j ∈ S. a j) = 1 - a i
  using sum.insert insert by fastforce
then have (∑ j ∈ S. a j) / (1 - a i) = 1
  using i0 by auto
then have a1: (∑ j ∈ S. ?a j) = 1
  unfolding sum_divide_distrib by simp
have convex C
  using ⟨convex_on C f⟩ by (simp add: convex_on_def)
have asum: (∑ j ∈ S. ?a j *R y j) ∈ C
  using insert convex_sum [OF ⟨finite S⟩ ⟨convex C⟩ a1 a_nonneg] by auto
have asum_le: f (∑ j ∈ S. ?a j *R y j) ≤ (∑ j ∈ S. ?a j * f (y j))
  using a_nonneg a1 insert by blast
have f (∑ j ∈ insert i S. a j *R y j) = f ((∑ j ∈ S. a j *R y j) + a i *R y i)
  by (simp add: add.commute insert.hyps)
also have ... = f (((1 - a i) * inverse (1 - a i)) *R (∑ j ∈ S. a j *R y j)
+ a i *R y i)
  using i0 by auto
also have ... = f ((1 - a i) *R (∑ j ∈ S. (a j * inverse (1 - a i)) *R y j)
+ a i *R y i)
  using scaleR_right.sum[of inverse (1 - a i) λ j. a j *R y j S, symmetric]
  by (auto simp: algebra_simps)
also have ... = f ((1 - a i) *R (∑ j ∈ S. ?a j *R y j) + a i *R y i)
  by (auto simp: divide_inverse)
also have ... ≤ (1 - a i) *R f ((∑ j ∈ S. ?a j *R y j)) + a i * f (y i)
  using ai1 by (smt (verit) asum conv real_scaleR_def yai)
also have ... ≤ (1 - a i) * (∑ j ∈ S. ?a j * f (y j)) + a i * f (y i)
  using asum_le i0 by fastforce
also have ... = (∑ j ∈ S. a j * f (y j)) + a i * f (y i)
  using i0 by (auto simp: sum_distrib_left)
finally show ?thesis
  using insert by auto
qed
qed

```

```

lemma concave_on_sum:
  fixes a :: 'a ⇒ real
  and y :: 'a ⇒ 'b::real_vector
  and f :: 'b ⇒ real
  assumes finite S S ≠ {}
  and concave_on C f

```

```

    and  $(\sum i \in S. a \ i) = 1$ 
    and  $\bigwedge i. i \in S \implies a \ i \geq 0$ 
    and  $\bigwedge i. i \in S \implies y \ i \in C$ 
  shows  $f (\sum i \in S. a \ i *_{\mathbb{R}} y \ i) \geq (\sum i \in S. a \ i * f (y \ i))$ 
proof -
  have  $(\text{uminus} \circ f) (\sum i \in S. a \ i *_{\mathbb{R}} y \ i) \leq (\sum i \in S. a \ i * (\text{uminus} \circ f) (y \ i))$ 
  proof (intro convex_on_sum)
    show convex_on C (uminus o f)
      by (simp add: assms convex_on_iff_concave)
    qed (use assms in auto)
  then show ?thesis
    by (simp add: sum_negf o_def)
qed

```

```

lemma convex_on_alt:
  fixes C :: 'a::real_vector set
  shows convex_on C f  $\longleftrightarrow$  convex C  $\wedge$ 
     $(\forall x \in C. \forall y \in C. \forall \mu :: \text{real}. \mu \geq 0 \wedge \mu \leq 1 \longrightarrow$ 
       $f (\mu *_{\mathbb{R}} x + (1 - \mu) *_{\mathbb{R}} y) \leq \mu * f x + (1 - \mu) * f y)$ 
  by (smt (verit) convex_on_def)

```

```

lemma convex_on_slope_le:
  fixes f :: real  $\Rightarrow$  real
  assumes f: convex_on I f
    and I:  $x \in I \ y \in I$ 
    and t:  $x < t \ t < y$ 
  shows  $(f x - f t) / (x - t) \leq (f x - f y) / (x - y)$ 
    and  $(f x - f y) / (x - y) \leq (f t - f y) / (t - y)$ 
proof -
  define a where  $a \equiv (t - y) / (x - y)$ 
  with t have  $0 \leq a \ 0 \leq 1 - a$ 
  by (auto simp: field_simps)
  with f  $\langle x \in I \rangle \langle y \in I \rangle$  have cvx:  $f (a * x + (1 - a) * y) \leq a * f x + (1 - a) * f y$ 
  by (auto simp: convex_on_def)
  have  $a * x + (1 - a) * y = a * (x - y) + y$ 
  by (simp add: field_simps)
  also have ... = t
  unfolding a_def using  $\langle x < t \rangle \langle t < y \rangle$  by simp
  finally have  $f t \leq a * f x + (1 - a) * f y$ 
  using cvx by simp
  also have ... =  $a * (f x - f y) + f y$ 
  by (simp add: field_simps)
  finally have  $f t - f y \leq a * (f x - f y)$ 
  by simp
  with t show  $(f x - f t) / (x - t) \leq (f x - f y) / (x - y)$ 
  by (simp add: le_divide_eq divide_le_eq field_simps a_def)
  with t show  $(f x - f y) / (x - y) \leq (f t - f y) / (t - y)$ 
  by (simp add: le_divide_eq divide_le_eq field_simps)

```

qed

lemma *pos_convex_function*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes *convex C*

and $\text{leq}: \bigwedge x y. x \in C \implies y \in C \implies f' x * (y - x) \leq f y - f x$

shows *convex_on C f*

unfolding *convex_on_alt*

using *assms*

proof *safe*

fix $x y \mu :: \text{real}$

let $?x = \mu *_R x + (1 - \mu) *_R y$

assume $*$: *convex C* $x \in C$ $y \in C$ $\mu \geq 0$ $\mu \leq 1$

then have $1 - \mu \geq 0$ by *auto*

then have $xpos$: $?x \in C$

using $*$ unfolding *convex_alt* by *fastforce*

have geq : $\mu * (f x - f ?x) + (1 - \mu) * (f y - f ?x) \geq$

$\mu * f' ?x * (x - ?x) + (1 - \mu) * f' ?x * (y - ?x)$

using *add_mono* [*OF mult_left_mono* [*OF leq* [*OF xpos *(2)*] $\langle \mu \geq 0 \rangle$]

mult_left_mono [*OF leq* [*OF xpos *(3)*] $\langle 1 - \mu \geq 0 \rangle$]

by *auto*

then have $\mu * f x + (1 - \mu) * f y - f ?x \geq 0$

by (*auto simp: field_simps*)

then show $f (\mu *_R x + (1 - \mu) *_R y) \leq \mu * f x + (1 - \mu) * f y$

by *auto*

qed

lemma *atMostAtLeast_subset_convex*:

fixes $C :: \text{real set}$

assumes *convex C*

and $x \in C$ $y \in C$ $x < y$

shows $\{x .. y\} \subseteq C$

proof *safe*

fix z assume z : $z \in \{x .. y\}$

have *less*: $z \in C$ if $*$: $x < z$ $z < y$

proof -

let $?\mu = (y - z) / (y - x)$

have $0 \leq ?\mu$ $?\mu \leq 1$

using *assms* $*$ by (*auto simp: field_simps*)

then have *comb*: $?\mu * x + (1 - ?\mu) * y \in C$

using *assms iffD1* [*OF convex_alt*, *rule_format*, *of C y x ?\mu*]

by (*simp add: algebra_simps*)

have $?\mu * x + (1 - ?\mu) * y = (y - z) * x / (y - x) + (1 - (y - z) / (y - x)) * y$

by (*auto simp: field_simps*)

also have $\dots = ((y - z) * x + (y - x - (y - z)) * y) / (y - x)$

using *assms* by (*simp only: add_divide_distrib*) (*auto simp: field_simps*)

also have $\dots = z$

using *assms* by (*auto simp: field_simps*)

```

    finally show ?thesis
      using comb by auto
qed
show z ∈ C
  using z less assms by (auto simp: le_less)
qed

lemma f''_imp_f':
  fixes f :: real ⇒ real
  assumes convex C
    and f':  $\bigwedge x. x \in C \implies \text{DERIV } f \ x \ :> (f' \ x)$ 
    and f'':  $\bigwedge x. x \in C \implies \text{DERIV } f' \ x \ :> (f'' \ x)$ 
    and pos:  $\bigwedge x. x \in C \implies f'' \ x \geq 0$ 
    and x:  $x \in C$ 
    and y:  $y \in C$ 
  shows  $f' \ x * (y - x) \leq f \ y - f \ x$ 
  using assms
proof -
  have  $f \ y - f \ x \geq f' \ x * (y - x) \wedge f' \ y * (x - y) \leq f \ x - f \ y$ 
  if *:  $x \in C \wedge y \in C \wedge y > x$  for  $x \ y :: real$ 
  proof -
    from * have ge:  $y - x > 0 \wedge y - x \geq 0$  and le:  $x - y < 0 \wedge x - y \leq 0$ 
    by auto
    then obtain z1 where  $z1: z1 > x \wedge z1 < y \wedge f \ y - f \ x = (y - x) * f' \ z1$ 
    using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨x ∈ C⟩ ⟨y ∈ C⟩ ⟨x < y⟩],
      THEN f', THEN MVT2[OF ⟨x < y⟩, rule_format, unfolded atLeastAtMost_iff[symmetric]]]
    by auto
    then have z1 ∈ C
    using atMostAtLeast_subset_convex ⟨convex C⟩ ⟨x ∈ C⟩ ⟨y ∈ C⟩ ⟨x < y⟩
    by fastforce
    obtain z2 where  $z2: z2 > x \wedge z2 < z1 \wedge f' \ z1 - f' \ x = (z1 - x) * f'' \ z2$ 
    using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨x ∈ C⟩ ⟨z1 ∈ C⟩ ⟨z1 < x⟩],
      THEN f'', THEN MVT2[OF ⟨x < z1⟩, rule_format, unfolded atLeastAtMost_iff[symmetric]]] z1
    by auto
    obtain z3 where  $z3: z3 > z1 \wedge z3 < y \wedge f' \ y - f' \ z1 = (y - z1) * f'' \ z3$ 
    using subsetD[OF atMostAtLeast_subset_convex[OF ⟨convex C⟩ ⟨z1 ∈ C⟩ ⟨y ∈ C⟩ ⟨z1 < y⟩],
      THEN f'', THEN MVT2[OF ⟨z1 < y⟩, rule_format, unfolded atLeastAtMost_iff[symmetric]]] z1
    by auto
    from z1 have  $f \ x - f \ y = (x - y) * f' \ z1$ 
    by (simp add: field_simps)
    then have cool':  $f' \ y - (f \ x - f \ y) / (x - y) = (y - z1) * f'' \ z3$ 
    using le(1) z3(3) by auto
    have z3 ∈ C

```

```

    using z3 * atMostAtLeast_subset_convex ⟨convex C⟩ ⟨x ∈ C⟩ ⟨z1 ∈ C⟩ ⟨x
< z1⟩
    by fastforce
    then have B': f'' z3 ≥ 0
    using assms by auto
    with cool' have f' y - (f x - f y) / (x - y) ≥ 0
    using z1 by auto
    then have res: f' y * (x - y) ≤ f x - f y
    by (meson diff_ge_0_iff_ge le(1) neg_divide_le_eq)
    have cool: (f y - f x) / (y - x) - f' x = (z1 - x) * f'' z2
    using le(1) z1(3) z2(3) by auto
    have z2 ∈ C
    using z2 z1 * atMostAtLeast_subset_convex ⟨convex C⟩ ⟨z1 ∈ C⟩ ⟨y ∈ C⟩
⟨z1 < y⟩
    by fastforce
    with z1 assms have (z1 - x) * f'' z2 ≥ 0
    by auto
    then show f y - f x ≥ f' x * (y - x) f' y * (x - y) ≤ f x - f y
    using that(3) z1(3) res cool by auto
qed
then show ?thesis
using x y by fastforce
qed

```

```

lemma f''_ge0_imp_convex:
  fixes f :: real ⇒ real
  assumes convex C
    and  $\bigwedge x. x \in C \implies \text{DERIV } f \ x \ := \ (f' \ x)$ 
    and  $\bigwedge x. x \in C \implies \text{DERIV } f' \ x \ := \ (f'' \ x)$ 
    and  $\bigwedge x. x \in C \implies f'' \ x \geq 0$ 
  shows convex_on C f
  by (metis assms f''_imp_f' pos_convex_function)

```

```

lemma f''_le0_imp_concave:
  fixes f :: real ⇒ real
  assumes convex C
    and  $\bigwedge x. x \in C \implies \text{DERIV } f \ x \ := \ (f' \ x)$ 
    and  $\bigwedge x. x \in C \implies \text{DERIV } f' \ x \ := \ (f'' \ x)$ 
    and  $\bigwedge x. x \in C \implies f'' \ x \leq 0$ 
  shows concave_on C f
  unfolding concave_on_def
  by (rule assms f''_ge0_imp_convex derivative_eq_intros | simp)+

```

```

lemma convex_power_even:
  assumes even n
  shows convex_on (UNIV::real set) ( $\lambda x. x^{\wedge} n$ )
proof (intro f''_ge0_imp_convex)
  show (( $\lambda x. x^{\wedge} n$ ) has_real_derivative of_nat n *  $x^{\wedge}(n-1)$ ) (at x) for x
  by (rule derivative_eq_intros | simp)+

```



```

show (( $\lambda x.$  of_nat  $n * x^{(n-1)}$ ) has_real_derivative of_nat  $n * of\_nat (n-1)$ 
*  $x^{(n-2)}$ ) (at  $x$ ) for  $x$ 
  by (rule derivative_eq_intros | simp add: eval_nat_numeral)+
show  $\bigwedge x. 0 \leq \text{real } n * \text{real } (n - 1) * x^{(n - 2)}$ 
  using assms by (auto simp: zero_le_mult_iff zero_le_even_power)
qed auto

```

```

lemma convex_power_odd:
  assumes odd  $n$ 
  shows convex_on {0::real..} ( $\lambda x. x^n$ )
proof (intro f''_ge0_imp_convex)
  show (( $\lambda x. x^n$ ) has_real_derivative of_nat  $n * x^{(n-1)}$ ) (at  $x$ ) for  $x$ 
    by (rule derivative_eq_intros | simp)+
  show (( $\lambda x.$  of_nat  $n * x^{(n-1)}$ ) has_real_derivative of_nat  $n * of\_nat (n-1)$ 
*  $x^{(n-2)}$ ) (at  $x$ ) for  $x$ 
    by (rule derivative_eq_intros | simp add: eval_nat_numeral)+
  show  $\bigwedge x. x \in \{0::\text{real}..\} \implies 0 \leq \text{real } n * \text{real } (n - 1) * x^{(n - 2)}$ 
    using assms by (auto simp: zero_le_mult_iff zero_le_even_power)
qed auto

```

```

lemma convex_power2: convex_on (UNIV::real set) power2
  by (simp add: convex_power_even)

```

```

lemma log_concave:
  fixes  $b :: \text{real}$ 
  assumes  $b > 1$ 
  shows concave_on {0<..} ( $\lambda x. \log b x$ )
  using assms
  by (intro f''_le0_imp_concave derivative_eq_intros | simp)+

```

```

lemma ln_concave: concave_on {0<..} ln
  unfolding log_ln by (simp add: log_concave)

```

```

lemma minus_log_convex:
  fixes  $b :: \text{real}$ 
  assumes  $b > 1$ 
  shows convex_on {0 <..} ( $\lambda x. - \log b x$ )
  using assms concave_on_def log_concave by blast

```

```

lemma powr_convex:
  assumes  $p \geq 1$  shows convex_on {0<..} ( $\lambda x. x \text{ powr } p$ )
  using assms
  by (intro f''_ge0_imp_convex derivative_eq_intros | simp)+

```

```

lemma exp_convex: convex_on UNIV exp
  by (intro f''_ge0_imp_convex derivative_eq_intros | simp)+

```

The AM-GM inequality: the arithmetic mean exceeds the geometric mean.

```

lemma arith_geom_mean:

```

```

fixes  $x :: 'a \Rightarrow \text{real}$ 
assumes  $\text{finite } S \ S \neq \{\}$ 
and  $x: \bigwedge i. i \in S \implies x\ i \geq 0$ 
shows  $(\sum i \in S. x\ i / \text{card } S) \geq (\prod i \in S. x\ i)^{\text{powr } (1 / \text{card } S)}$ 
proof (cases  $\exists i \in S. x\ i = 0$ )
  case True
    then have  $(\prod i \in S. x\ i) = 0$ 
      by (simp add: <finite S>)
    moreover have  $(\sum i \in S. x\ i / \text{card } S) \geq 0$ 
      by (simp add: sum_nonneg x)
    ultimately show ?thesis
      by simp
  next
    case False
    have  $\ln (\sum i \in S. (1 / \text{card } S) *_{\mathbb{R}} x\ i) \geq (\sum i \in S. (1 / \text{card } S) * \ln (x\ i))$ 
    proof (intro concave_on_sum)
      show concave_on  $\{0 <..\}$  ln
        by (simp add: ln_concave)
      show  $\bigwedge i. i \in S \implies x\ i \in \{0 <..\}$ 
        using False x by fastforce
    qed (use assms False in auto)
    moreover have  $(\sum i \in S. (1 / \text{card } S) *_{\mathbb{R}} x\ i) > 0$ 
      using False assms by (simp add: card_gt_0_iff less_eq_real_def sum_pos)
    ultimately have  $(\sum i \in S. (1 / \text{card } S) *_{\mathbb{R}} x\ i) \geq \exp (\sum i \in S. (1 / \text{card } S) * \ln (x\ i))$ 
      using ln_ge_iff by blast
    then have  $(\sum i \in S. x\ i / \text{card } S) \geq \exp (\sum i \in S. \ln (x\ i) / \text{card } S)$ 
      by (simp add: divide_simps)
    then show ?thesis
      using assms False
      by (smt (verit, ccfv_SIG) divide_inverse exp_ln exp_powr_real exp_sum inverse_eq_divide prod.cong prod_powr_distrib)
    qed

```

1.7.5 Convexity of real functions

```

lemma convex_on_realI:
  assumes connected A
    and  $\bigwedge x. x \in A \implies (f \text{ has\_real\_derivative } f'\ x) (at\ x)$ 
    and  $\bigwedge x\ y. x \in A \implies y \in A \implies x \leq y \implies f'\ x \leq f'\ y$ 
  shows convex_on A f
proof (rule convex_on_linorderI)
  show convex A
    using  $\langle \text{connected } A \rangle$  convex_real_interval interval_cases
    by (smt (verit, ccfv_SIG) connectedD_interval convex_UNIV convex_empty)
    — the equivalence of "connected" and "convex" for real intervals is proved later
next
  fix  $t\ x\ y :: \text{real}$ 
  assume  $t: t > 0 \ t < 1$ 

```

```

assume xy:  $x \in A \ y \in A \ x < y$ 
define z where  $z = (1 - t) * x + t * y$ 
with  $\langle \text{connected } A \rangle$  and xy have ivl:  $\{x..y\} \subseteq A$ 
  using connected_contains_Icc by blast

from xy t have xz:  $z > x$ 
  by (simp add: z_def algebra_simps)
have  $y - z = (1 - t) * (y - x)$ 
  by (simp add: z_def algebra_simps)
also from xy t have  $\dots > 0$ 
  by (intro mult_pos_pos simp_all)
finally have yz:  $z < y$ 
  by simp

from assms xz yz ivl t have  $\exists \xi. \xi > x \wedge \xi < z \wedge fz - fx = (z - x) * f' \xi$ 
  by (intro MVT2) (auto intro!: assms(2))
then obtain  $\xi$  where  $\xi: \xi > x \ \xi < z \ f' \xi = (fz - fx) / (z - x)$ 
  by auto
from assms xz yz ivl t have  $\exists \eta. \eta > z \wedge \eta < y \wedge fy - fz = (y - z) * f' \eta$ 
  by (intro MVT2) (auto intro!: assms(2))
then obtain  $\eta$  where  $\eta: \eta > z \ \eta < y \ f' \eta = (fy - fz) / (y - z)$ 
  by auto

from  $\eta(3)$  have  $(fy - fz) / (y - z) = f' \eta ..$ 
also from  $\xi \ \eta \ \text{ivl}$  have  $\xi \in A \ \eta \in A$ 
  by auto
with  $\xi \ \eta$  have  $f' \eta \geq f' \xi$ 
  by (intro assms(3)) auto
also from  $\xi(3)$  have  $f' \xi = (fz - fx) / (z - x) .$ 
finally have  $(fy - fz) * (z - x) \geq (fz - fx) * (y - z)$ 
  using xz yz by (simp add: field_simps)
also have  $z - x = t * (y - x)$ 
  by (simp add: z_def algebra_simps)
also have  $y - z = (1 - t) * (y - x)$ 
  by (simp add: z_def algebra_simps)
finally have  $(fy - fz) * t \geq (fz - fx) * (1 - t)$ 
  using xy by simp
then show  $(1 - t) * fx + t * fy \geq f((1 - t) * x + t * y)$ 
  by (simp add: z_def algebra_simps)
qed

lemma convex_on_inverse:
  fixes  $A :: \text{real set}$ 
  assumes  $A \subseteq \{0<..\}$  convex A
  shows convex_on A inverse
proof -
  have convex_on  $\{0::\text{real}<..\}$  inverse
  proof (intro convex_on_realI)
    fix  $u \ v :: \text{real}$ 

```

```

    assume  $u \in \{0<..\}$   $v \in \{0<..\}$   $u \leq v$ 
    with assms show  $-inverse (u^2) \leq -inverse (v^2)$ 
      by simp
  next
    show  $\bigwedge x. x \in \{0<..\} \implies (inverse \text{ has\_real\_derivative } - inverse (x^2)) (at x)$ 
      by (rule derivative_eq_intros | simp add: power2_eq_square)+
  qed auto
  then show ?thesis
    using assms convex_on_subset by blast
  qed

```

```

lemma convex_onD_Icc':
  assumes convex_on  $\{x..y\}$   $f c \in \{x..y\}$ 
  defines  $d \equiv y - x$ 
  shows  $f c \leq (f y - f x) / d * (c - x) + f x$ 
proof (cases  $x y$  rule: linorder_cases)
  case less
  then have  $d: d > 0$ 
    by (simp add: d_def)
  from assms(2) less have  $A: 0 \leq (c - x) / d (c - x) / d \leq 1$ 
    by (simp_all add: d_def field_split_simps)
  have  $f c = f (x + (c - x) * 1)$ 
    by simp
  also from less have  $1 = ((y - x) / d)$ 
    by (simp add: d_def)
  also from  $d$  have  $x + (c - x) * \dots = (1 - (c - x) / d) *_R x + ((c - x) / d) *_R y$ 
    by (simp add: field_simps)
  also have  $f \dots \leq (1 - (c - x) / d) * f x + (c - x) / d * f y$ 
    using assms less by (intro convex_onD_Icc) simp_all
  also from  $d$  have  $\dots = (f y - f x) / d * (c - x) + f x$ 
    by (simp add: field_simps)
  finally show ?thesis .
qed (use assms in auto)

```

```

lemma convex_onD_Icc'':
  assumes convex_on  $\{x..y\}$   $f c \in \{x..y\}$ 
  defines  $d \equiv y - x$ 
  shows  $f c \leq (f x - f y) / d * (y - c) + f y$ 
proof (cases  $x y$  rule: linorder_cases)
  case less
  then have  $d: d > 0$ 
    by (simp add: d_def)
  from assms(2) less have  $A: 0 \leq (y - c) / d (y - c) / d \leq 1$ 
    by (simp_all add: d_def field_split_simps)
  have  $f c = f (y - (y - c) * 1)$ 
    by simp
  also from less have  $1 = ((y - x) / d)$ 
    by (simp add: d_def)

```

```

also from  $d$  have  $y - (y - c) * \dots = (1 - (1 - (y - c) / d)) * x + (1 - (y - c) / d) * y$ 
by (simp add: field_simps)
also have  $f \dots \leq (1 - (1 - (y - c) / d)) * f x + (1 - (y - c) / d) * f y$ 
using assms less by (intro convex_onD_Icc) (simp_all add: field_simps)
also from  $d$  have  $\dots = (f x - f y) / d * (y - c) + f y$ 
by (simp add: field_simps)
finally show ?thesis .
qed (use assms in auto)

```

```

lemma concave_onD_Icc:
assumes concave_on { $x..y$ }  $f x \leq (y :: \_ :: \{real\_vector,preorder\})$ 
shows  $\bigwedge t. t \geq 0 \implies t \leq 1 \implies$ 
 $f ((1 - t) * x + t * y) \geq (1 - t) * f x + t * f y$ 
using assms(2) by (intro concave_onD [OF assms(1)] simp_all)

```

```

lemma concave_onD_Icc':
assumes concave_on { $x..y$ }  $f c \in \{x..y\}$ 
defines  $d \equiv y - x$ 
shows  $f c \geq (f y - f x) / d * (c - x) + f x$ 
proof -
have  $- f c \leq (f x - f y) / d * (c - x) - f x$ 
using assms convex_onD_Icc' [of x y \lambda x. - f x c]
by (simp add: concave_on_def)
then show ?thesis
by (smt (verit, best) divide_minus_left mult_minus_left)
qed

```

```

lemma concave_onD_Icc'':
assumes concave_on { $x..y$ }  $f c \in \{x..y\}$ 
defines  $d \equiv y - x$ 
shows  $f c \geq (f x - f y) / d * (y - c) + f y$ 
proof -
have  $- f c \leq (f y - f x) / d * (y - c) - f y$ 
using assms convex_onD_Icc'' [of x y \lambda x. - f x c]
by (simp add: concave_on_def)
then show ?thesis
by (smt (verit, best) divide_minus_left mult_minus_left)
qed

```

```

lemma convex_on_le_max:
fixes  $a::real$ 
assumes convex_on { $x..y$ }  $f$  and  $a: a \in \{x..y\}$ 
shows  $f a \leq \max (f x) (f y)$ 
proof -
have  $*$ :  $(f y - f x) * (a - x) \leq (f y - f x) * (y - x)$  if  $f x \leq f y$ 
using  $a$  that by (intro mult_left_mono) auto
have  $f a \leq (f y - f x) / (y - x) * (a - x) + f x$ 
using assms convex_onD_Icc' by blast

```

```

also have ... ≤ max (f x) (f y)
using a *
by (simp add: divide_le_0_iff mult_le_0_iff zero_le_mult_iff max_def add.commute
mult.commute scaling_mono)
finally show ?thesis .
qed

```

```

lemma concave_on_ge_min:
  fixes a::real
  assumes concave_on {x..y} f and a: a ∈ {x..y}
  shows f a ≥ min (f x) (f y)
proof -
  have *: (f y - f x) * (a - x) ≥ (f y - f x) * (y - x) if f x ≥ f y
    using a that by (intro mult_left_mono_neg) auto
  have min (f x) (f y) ≤ (f y - f x) / (y - x) * (a - x) + f x
    using a * apply (simp add: zero_le_divide_iff mult_le_0_iff zero_le_mult_iff
min_def)
  by (smt (verit, best) nonzero_eq_divide_eq pos_divide_le_eq)
  also have ... ≤ f a
    using assms concave_onD_Icc' by blast
  finally show ?thesis .
qed

```

1.7.6 Some inequalities: Applications of convexity

```

lemma Youngs_inequality_0:
  fixes a::real
  assumes 0 ≤ α 0 ≤ β α+β = 1 a>0 b>0
  shows a powr α * b powr β ≤ α*a + β*b
proof -
  have α * ln a + β * ln b ≤ ln (α * a + β * b)
    using assms ln_concave by (simp add: concave_on_iff)
  moreover have 0 < α * a + β * b
    using assms by (smt (verit) mult_pos_pos split_mult_pos_le)
  ultimately show ?thesis
    using assms by (simp add: powr_def mult_exp_exp flip: ln_ge_iff)
qed

```

```

lemma Youngs_inequality:
  fixes p::real
  assumes p>1 q>1 1/p + 1/q = 1 a≥0 b≥0
  shows a * b ≤ a powr p / p + b powr q / q
proof (cases a=0 ∨ b=0)
  case False
  then show ?thesis
    using Youngs_inequality_0 [of 1/p 1/q a powr p b powr q] assms
    by (simp add: powr_powr)
qed (use assms in auto)

```

```

lemma Cauchy_Schwarz_ineq_sum:
  fixes  $a :: 'a \Rightarrow 'b::\text{linordered\_field}$ 
  shows  $(\sum i \in I. a\ i * b\ i)^2 \leq (\sum i \in I. (a\ i)^2) * (\sum i \in I. (b\ i)^2)$ 
proof (cases  $(\sum i \in I. (b\ i)^2) > 0$ )
  case False
  then consider  $\bigwedge i. i \in I \implies b\ i = 0 \mid \text{infinite } I$ 
    by (metis (mono_tags, lifting) sum_pos2 zero_le_power2 zero_less_power2)
  thus ?thesis
    by fastforce
next
  case True
  define  $r$  where  $r \equiv (\sum i \in I. a\ i * b\ i) / (\sum i \in I. (b\ i)^2)$ 
  have  $0 \leq (\sum i \in I. (a\ i - r * b\ i)^2)$ 
    by (simp add: sum_nonneg)
  also have  $\dots = (\sum i \in I. (a\ i)^2) - 2 * r * (\sum i \in I. a\ i * b\ i) + r^2 * (\sum i \in I. (b\ i)^2)$ 
    by (simp add: algebra_simps power2_eq_square sum_distrib_left flip: sum.distrib)
  also have  $\dots = (\sum i \in I. (a\ i)^2) - ((\sum i \in I. a\ i * b\ i)^2 / (\sum i \in I. (b\ i)^2))$ 
    by (simp add: r_def power2_eq_square)
  finally have  $0 \leq (\sum i \in I. (a\ i)^2) - ((\sum i \in I. a\ i * b\ i)^2 / (\sum i \in I. (b\ i)^2))$ 
  hence  $((\sum i \in I. a\ i * b\ i)^2 / (\sum i \in I. (b\ i)^2)) \leq (\sum i \in I. (a\ i)^2)$ 
    by (simp add: le_diff_eq)
  thus  $((\sum i \in I. a\ i * b\ i)^2) \leq (\sum i \in I. (a\ i)^2) * (\sum i \in I. (b\ i)^2)$ 
    by (simp add: pos_divide_le_eq True)
qed

```

```

lemma sum_squared_le_sum_of_squares:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes  $\bigwedge i. i \in I \implies f\ i \geq 0$  finite I I \neq \{\}
  shows  $(\sum i \in I. f\ i)^2 \leq (\sum y \in I. (f\ y)^2) * \text{card } I$ 
proof (cases finite I \wedge I \neq \{\})
  case True
  have  $(\sum i \in I. f\ i / \text{real } (\text{card } I))^2 \leq (\sum i \in I. (f\ i)^2 / \text{real } (\text{card } I))$ 
    using assms convex_on_sum [OF _ _ convex_power2, where a = \lambda x. 1 / real(card I) and S=I]
    by simp
  then show ?thesis
    using assms
    by (simp add: divide_simps power2_eq_square split: if_split_asm flip: sum_divide_distrib)
qed auto

```

1.7.7 Misc related lemmas

```

lemma convex_translation_eq [simp]:
   $\text{convex } ((+) a \text{ ' } s) \longleftrightarrow \text{convex } s$ 
  by (metis convex_translation translation_galois)

```

```

lemma convex_translation_subtract_eq [simp]:
   $\text{convex } ((\lambda b. b - a) \text{ ' } s) \longleftrightarrow \text{convex } s$ 

```

using *convex_translation_eq* [*of - a*] **by** (*simp cong: image_cong_simp*)

lemma *convex_linear_image_eq* [*simp*]:

fixes $f :: 'a::\text{real_vector} \Rightarrow 'b::\text{real_vector}$

shows $\llbracket \text{linear } f; \text{inj } f \rrbracket \Longrightarrow \text{convex } (f \text{ ` } s) \longleftrightarrow \text{convex } s$

by (*metis (no_types) convex_linear_image convex_linear_vimage inj_vimage_image_eq*)

lemma *vector_choose_size*:

assumes $0 \leq c$

obtains $x :: 'a::\{\text{real_normed_vector}, \text{perfect_space}\}$ **where** $\text{norm } x = c$

proof –

obtain $a::'a$ **where** $a \neq 0$

using *UNIV_not_singleton UNIV_eq_I set_zero singletonI* **by** *fastforce*

then show *?thesis*

by (*rule_tac x=scaleR (c / norm a) a in that*) (*simp add: assms*)

qed

lemma *vector_choose_dist*:

assumes $0 \leq c$

obtains $y :: 'a::\{\text{real_normed_vector}, \text{perfect_space}\}$ **where** $\text{dist } x \ y = c$

by (*metis add_diff_cancel_left' assms dist_commute dist_norm vector_choose_size*)

lemma *sum_delta'*:

fixes $s::'a::\text{real_vector}$ *set*

assumes *finite s*

shows $(\sum x \in s. (\text{if } y = x \text{ then } f \ x \text{ else } 0) *_{\mathbb{R}} x) = (\text{if } y \in s \text{ then } (f \ y) *_{\mathbb{R}} y \text{ else } 0)$

proof –

have $*$: $\bigwedge x \ y. (\text{if } y = x \text{ then } f \ x \text{ else } (0::\text{real})) *_{\mathbb{R}} x = (\text{if } x=y \text{ then } (f \ x) *_{\mathbb{R}} x \text{ else } 0)$

by *auto*

show *?thesis*

unfolding $*$ **using** *sum.delta[OF assms, of y $\lambda x. f \ x *_{\mathbb{R}} x$]* **by** *auto*

qed

1.7.8 Cones

definition *cone* $:: 'a::\text{real_vector}$ *set* $\Rightarrow \text{bool}$

where $\text{cone } s \longleftrightarrow (\forall x \in s. \forall c \geq 0. c *_{\mathbb{R}} x \in s)$

lemma *cone_empty*[*intro, simp*]: $\text{cone } \{\}$

unfolding *cone_def* **by** *auto*

lemma *cone_univ*[*intro, simp*]: $\text{cone } \text{UNIV}$

unfolding *cone_def* **by** *auto*

lemma *cone_Inter*[*intro*]: $\forall s \in f. \text{cone } s \Longrightarrow \text{cone } (\bigcap f)$

unfolding *cone_def* **by** *auto*

lemma *subspace_imp_cone*: $\text{subspace } S \Longrightarrow \text{cone } S$

by (simp add: cone_def subspace_scale)

Conic hull

lemma cone_cone_hull: cone (cone hull S)
unfolding hull_def by auto

lemma cone_hull_eq: cone hull S = S \longleftrightarrow cone S
by (metis cone_cone_hull hull_same)

lemma mem_cone:
assumes cone S x \in S c \geq 0
shows c \ast_R x \in S
using assms cone_def[of S] by auto

lemma cone_contains_0:
assumes cone S
shows S \neq {} \longleftrightarrow 0 \in S
using assms mem_cone by fastforce

lemma cone_0: cone {0}
unfolding cone_def by auto

lemma cone_Union[intro]: ($\forall s \in f. \text{cone } s$) \longrightarrow cone ($\bigcup f$)
unfolding cone_def by blast

lemma cone_iff:
assumes S \neq {}
shows cone S \longleftrightarrow 0 \in S \wedge ($\forall c. c > 0 \longrightarrow ((\ast_R) c) \text{ ' } S = S$) (is _ = ?rhs)

proof

assume cone S
{
 fix c :: real
 assume c > 0
 have x \in ((\ast_R) c) ' S if x \in S for x
 using <cone S> <c>0 mem_cone[of S x 1/c] that
 exI[of ($\lambda t. t \in S \wedge x = c \ast_R t$) (1 / c) \ast_R x]
 by auto
 then have ((\ast_R) c) ' S = S
 using <0 < c> <cone S> mem_cone by fastforce
}

then show 0 \in S \wedge ($\forall c. c > 0 \longrightarrow ((\ast_R) c) \text{ ' } S = S$)
 using <cone S> cone_contains_0[of S] assms by auto

next

show ?rhs \implies cone S
 by (metis Convex.cone_def imageI order_neq_le_trans scaleR_zero_left)

qed

lemma cone_hull_empty: cone hull {} = {}

by (metis cone_empty cone_hull_eq)

lemma cone_hull_empty_iff: $S = \{\}$ \longleftrightarrow cone hull $S = \{\}$
 by (metis cone_hull_empty hull_subset subset_empty)

lemma cone_hull_contains_0: $S \neq \{\}$ \longleftrightarrow $0 \in$ cone hull S
 by (metis cone_cone_hull cone_contains_0 cone_hull_empty_iff)

lemma mem_cone_hull:
 assumes $x \in S$ $c \geq 0$
 shows $c *_R x \in$ cone hull S
 by (metis assms cone_cone_hull hull_inc mem_cone)

proposition cone_hull_expl: cone hull $S = \{c *_R x \mid c x. c \geq 0 \wedge x \in S\}$
 (is ?lhs = ?rhs)

proof
 have ?rhs \in Collect cone
 using Convex.cone_def by fastforce
 moreover have $S \subseteq$?rhs
 by (smt (verit) mem_Collect_eq scaleR_one subsetI)
 ultimately show ?lhs \subseteq ?rhs
 using hull_minimal by blast
qed (use mem_cone_hull in auto)

lemma convex_cone:
 $convex S \wedge cone S \longleftrightarrow (\forall x \in S. \forall y \in S. (x + y) \in S) \wedge (\forall x \in S. \forall c \geq 0. (c *_R x) \in S)$
 (is ?lhs = ?rhs)

proof –
 {
 fix $x y$
 assume $x \in S$ $y \in S$ and ?lhs
 then have $2 *_R x \in S$ $2 *_R y \in S$ convex S
 unfolding cone_def by auto
 then have $x + y \in S$
 using convexD [OF ⟨convex S ⟩, of $2 *_R x$ $2 *_R y$]
 by (smt (verit, ccfv_threshold) field_sum_of_halves scaleR_2 scaleR_half_double)
 }
 then show ?thesis
 unfolding convex_def cone_def by blast
qed

1.7.9 Connectedness of convex sets

lemma convex_connected:
 fixes $S :: 'a::real_normed_vector$ set
 assumes convex S
 shows connected S
proof (rule connectedI)

```

fix A B
assume open A open B A ∩ B ∩ S = {} S ⊆ A ∪ B
moreover
assume A ∩ S ≠ {} B ∩ S ≠ {}
then obtain a b where a: a ∈ A a ∈ S and b: b ∈ B b ∈ S by auto
define f where [abs_def]: f u = u *R a + (1 - u) *R b for u
then have continuous_on {0 .. 1} f
  by (auto intro: continuous_intros)
then have connected (f ' {0 .. 1})
  by (auto intro: connected_continuous_image)
note connectedD[OF this, of A B]
moreover have a ∈ A ∩ f ' {0 .. 1}
  using a by (auto intro: image_eqI[of _ _ 1] simp: f_def)
moreover have b ∈ B ∩ f ' {0 .. 1}
  using b by (auto intro: image_eqI[of _ _ 0] simp: f_def)
moreover have f ' {0 .. 1} ⊆ S
  using ⟨convex S⟩ a b unfolding convex_def f_def by auto
ultimately show False by auto
qed

```

```

corollary connected_UNIV[intro]: connected (UNIV :: 'a::real_normed_vector set)
  by (simp add: convex_connected)

```

```

lemma convex_prod:
  assumes ∧i. i ∈ Basis ⇒ convex {x. P i x}
  shows convex {x. ∀i∈Basis. P i (x·i)}
  using assms by (auto simp: inner_add_left convex_def)

```

```

lemma convex_positive_orthant: convex {x::'a::euclidean_space. (∀i∈Basis. 0 ≤ x·i)}
  by (rule convex_prod) (simp flip: atLeast_def)

```

1.7.10 Convex hull

```

lemma convex_convex_hull [iff]: convex (convex hull s)
  by (metis (mono_tags) convex_Inter hull_def mem_Collect_eq)

```

```

lemma convex_hull_subset:
  s ⊆ convex hull t ⇒ convex hull s ⊆ convex hull t
  by (simp add: subset_hull)

```

```

lemma convex_hull_eq: convex hull s = s ↔ convex s
  by (metis convex_convex_hull hull_same)

```

Convex hull is "preserved" by a linear function

```

lemma convex_hull_linear_image:
  assumes f: linear f
  shows f ' (convex hull S) = convex hull (f ' S)
proof

```

```

show convex hull (f ‘ S) ⊆ f ‘ (convex hull S)
  by (intro hull_minimal image_mono hull_subset convex_linear_image assms
convex_convex_hull)
show f ‘ (convex hull S) ⊆ convex hull (f ‘ S)
  by (meson convex_convex_hull convex_linear_vimage f hull_minimal hull_subset
image_subset_iff_subset_vimage)
qed

```

```

lemma in_convex_hull_linear_image:
  assumes linear f x ∈ convex hull S
  shows f x ∈ convex hull (f ‘ S)
  using assms convex_hull_linear_image image_eqI by blast

```

```

lemma convex_hull_Times:
  convex hull (S × T) = (convex hull S) × (convex hull T)
proof
show convex hull (S × T) ⊆ (convex hull S) × (convex hull T)
  by (intro hull_minimal Sigma_mono hull_subset convex_Times convex_convex_hull)
  have (x, y) ∈ convex hull (S × T) if x: x ∈ convex hull S and y: y ∈ convex
hull T for x y
  proof (rule hull_induct [OF x], rule hull_induct [OF y])
    fix x y assume x ∈ S and y ∈ T
    then show (x, y) ∈ convex hull (S × T)
      by (simp add: hull_inc)
  next
  fix x let ?S = ((λy. (0, y)) - ‘ (λp. (- x, 0) + p) ‘ (convex hull S × T))
  have convex ?S
    by (intro convex_linear_vimage convex_translation convex_convex_hull,
      simp add: linear_iff)
  also have ?S = {y. (x, y) ∈ convex hull (S × T)}
    by (auto simp: image_def Bex_def)
  finally show convex {y. (x, y) ∈ convex hull (S × T)} .
  next
  show convex {x. (x, y) ∈ convex hull S × T}
  proof -
    fix y let ?S = ((λx. (x, 0)) - ‘ (λp. (0, - y) + p) ‘ (convex hull S × T))
    have convex ?S
      by (intro convex_linear_vimage convex_translation convex_convex_hull,
        simp add: linear_iff)
    also have ?S = {x. (x, y) ∈ convex hull (S × T)}
      by (auto simp: image_def Bex_def)
    finally show convex {x. (x, y) ∈ convex hull (S × T)} .
  qed
qed
then show (convex hull S) × (convex hull T) ⊆ convex hull (S × T)
  unfolding subset_eq split_paired_Ball_Sigma by blast
qed

```

Stepping theorems for convex hulls of finite sets

lemma *convex_hull_empty*[simp]: *convex hull* {} = {}
 by (simp add: hull_same)

lemma *convex_hull_singleton*[simp]: *convex hull* {a} = {a}
 by (simp add: hull_same)

lemma *convex_hull_insert*:
 fixes $S :: 'a::\text{real_vector set}$
 assumes $S \neq \{\}$
 shows *convex hull* (insert a S) =
 $\{x. \exists u \geq 0. \exists v \geq 0. \exists b. (u + v = 1) \wedge b \in (\text{convex hull } S) \wedge (x = u *_R a + v *_R b)\}$
 (is _ = ?hull)
proof (intro equalityI hull_minimal subsetI)
 fix x
 assume $x \in \text{insert } a \ S$
 then show $x \in ?\text{hull}$
 unfolding insert_iff
 proof
 assume $x = a$
 then show ?thesis
 by (smt (verit, del_insts) add.right_neutral assms ex_in_conv hull_inc mem_Collect_eq scaleR_one scaleR_zero_left)
 next
 assume $x \in S$
 with hull_subset show ?thesis
 by force
 qed
 next
 fix x
 assume $x \in ?\text{hull}$
 then obtain u v b where obt: $u \geq 0 \ v \geq 0 \ u + v = 1 \ b \in \text{convex hull } S \ x = u *_R a + v *_R b$
 by auto
 have $a \in \text{convex hull insert } a \ S \ b \in \text{convex hull insert } a \ S$
 using hull_mono[of S insert a S convex] hull_mono[of {a} insert a S convex]
 and obt(4)
 by auto
 then show $x \in \text{convex hull insert } a \ S$
 unfolding obt(5) using obt(1-3)
 by (rule convexD [OF convex_convex_hull])
 next
 show convex ?hull
 proof (rule convexI)
 fix x y u v
 assume as: $(0::\text{real}) \leq u \ 0 \leq v \ u + v = 1$ and x: $x \in ?\text{hull}$ and y: $y \in ?\text{hull}$
 from x obtain u1 v1 b1 where
 obt1: $u1 \geq 0 \ v1 \geq 0 \ u1 + v1 = 1 \ b1 \in \text{convex hull } S$ and xeq: $x = u1 *_R a +$

```

v1 *R b1
  by auto
  from y obtain u2 v2 b2 where
    obt2: u2 ≥ 0 v2 ≥ 0 u2 + v2 = 1 b2 ∈ convex hull S and yeq: y = u2 *R a +
v2 *R b2
  by auto
  have *: ∧(x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R x
  by (auto simp: algebra_simps)
  have ∃ b ∈ convex hull S. u *R x + v *R y =
    (u * u1) *R a + (v * u2) *R a + (b - (u * u1) *R b - (v * u2) *R b)
  proof (cases u * v1 + v * v2 = 0)
    case True
      have *: ∧(x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R
x
      by (auto simp: algebra_simps)
      have eq0: u * v1 = 0 v * v2 = 0
      using True mult_nonneg_nonneg[OF ⟨u ≥ 0⟩ ⟨v1 ≥ 0⟩] mult_nonneg_nonneg[OF
⟨v ≥ 0⟩ ⟨v2 ≥ 0⟩]
      by arith+
      then have u * u1 + v * u2 = 1
      using as(3) obt1(3) obt2(3) by auto
      then show ?thesis
      using * eq0 as obt1(4) xeq yeq by auto
    next
      case False
      have 1 - (u * u1 + v * u2) = (u + v) - (u * u1 + v * u2)
      by (simp add: as(3))
      also have ... = u * v1 + v * v2
      by (smt (verit, ccfv_SIG) distrib_left mult_cancel_left1 obt1(3) obt2(3))
      finally have **: 1 - (u * u1 + v * u2) = u * v1 + v * v2 .
      let ?b = ((u * v1) / (u * v1 + v * v2)) *R b1 + ((v * v2) / (u * v1 + v *
v2)) *R b2
      have zeroes: 0 ≤ u * v1 + v * v2 0 ≤ u * v1 0 ≤ u * v1 + v * v2 0 ≤ v *
v2
      using as obt1 obt2 by auto
      show ?thesis
      proof
        show u *R x + v *R y = (u * u1) *R a + (v * u2) *R a + (?b - (u * u1)
*R ?b - (v * u2) *R ?b)
        unfolding xeq yeq * **
        using False by (auto simp: scaleR_left_distrib scaleR_right_distrib)
        show ?b ∈ convex hull S
        using False mem_convex_alt obt1(4) obt2(4) zeroes(2) zeroes(4) by
fastforce
      qed
    qed
  then obtain b where b: b ∈ convex hull S
  u *R x + v *R y = (u * u1) *R a + (v * u2) *R a + (b - (u * u1) *R b -
(v * u2) *R b) ..

```

```

obtain  $u1: u1 \leq 1$  and  $u2: u2 \leq 1$ 
  using obt1 obt2 by auto
have  $u1 * u + u2 * v \leq \max u1 u2 * u + \max u1 u2 * v$ 
  by (smt (verit, ccfv_SIG) as mult_right_mono)
also have  $\dots \leq 1$ 
  unfolding distrib_left[symmetric] and as(3) using  $u1 u2$  by auto
finally have  $le1: u1 * u + u2 * v \leq 1$  .
show  $u *_R x + v *_R y \in ?hull$ 
proof (intro CollectI exI conjI)
  show  $0 \leq u * u1 + v * u2$ 
    by (simp add: as obt1(1) obt2(1))
  show  $0 \leq 1 - u * u1 - v * u2$ 
    by (simp add: le1 diff_diff_add mult.commute)
qed (use b in <auto simp: algebra_simps>)
qed
qed

```

```

lemma convex_hull_insert_alt:
  convex hull (insert a S) =
    (if S = {} then {a}
     else {(1 - u) *_R a + u *_R x | x u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ convex hull S})
apply (simp add: convex_hull_insert)
using diff_add_cancel diff_ge_0_iff_ge
by (smt (verit, del_insts) Collect_cong)

```

Explicit expression for convex hull

```

proposition convex_hull_indexed:
  fixes  $S :: 'a::real\_vector\ set$ 
  shows convex hull S =
     $\{y. \exists k\ u\ x. (\forall i \in \{1::nat .. k\}. 0 \leq u\ i \wedge x\ i \in S) \wedge$ 
       $(\sum u\ \{1..k\} = 1) \wedge (\sum i = 1..k. u\ i *_R\ x\ i) = y\}$ 
    (is ?xyz = ?hull)
proof (rule hull_unique [OF _ convexI])
  show  $S \subseteq ?hull$ 
    by (clarsimp, rule_tac x=1 in exI, rule_tac x=λx. 1 in exI, auto)
next
  fix  $T$ 
  assume  $S \subseteq T$  convex T
  then show  $?hull \subseteq T$ 
    by (blast intro: convex_sum)
next
  fix  $x\ y\ u\ v$ 
  assume  $uv: 0 \leq u \leq v$   $u + v = (1::real)$ 
  assume  $xy: x \in ?hull\ y \in ?hull$ 
  from  $xy$  obtain  $k1\ u1\ x1$  where
     $x$  [rule_format]:  $\forall i \in \{1::nat..k1\}. 0 \leq u1\ i \wedge x1\ i \in S$ 
       $\sum u1\ \{Suc\ 0..k1\} = 1$   $(\sum i = Suc\ 0..k1. u1\ i *_R\ x1\ i) = x$ 
  by auto

```

from xy **obtain** $k2\ u2\ x2$ **where**
 y [*rule_format*]: $\forall i \in \{1::nat..k2\}. 0 \leq u2\ i \wedge x2\ i \in S$
 $sum\ u2\ \{Suc\ 0..k2\} = 1\ (\sum\ i = Suc\ 0..k2. u2\ i *R\ x2\ i) = y$
by *auto*
have $*$: $\bigwedge P\ (x::'a)\ y\ s\ t\ i. (if\ P\ i\ then\ s\ else\ t) *R\ (if\ P\ i\ then\ x\ else\ y) = (if\ P\ i\ then\ s *R\ x\ else\ t *R\ y)$
 $\{1..k1 + k2\} \cap \{1..k1\} = \{1..k1\}\ \{1..k1 + k2\} \cap -\ \{1..k1\} = (\lambda i. i + k1)\ ' \{1..k2\}$
by *auto*
have inj : $inj_on\ (\lambda i. i + k1)\ \{1..k2\}$
unfolding inj_on_def **by** *auto*
let $?uu = \lambda i. if\ i \in \{1..k1\}\ then\ u * u1\ i\ else\ v * u2\ (i - k1)$
let $?xx = \lambda i. if\ i \in \{1..k1\}\ then\ x1\ i\ else\ x2\ (i - k1)$
show $u *R\ x + v *R\ y \in ?hull$
proof (*intro CollectI exI conjI ballI*)
show $0 \leq ?uu\ i\ ?xx\ i \in S$ **if** $i \in \{1..k1+k2\}$ **for** i
using *that* **by** (*auto simp add: le_diff_conv uv(1) x(1) uv(2) y(1)*)
show $(\sum\ i = 1..k1 + k2. ?uu\ i) = 1\ (\sum\ i = 1..k1 + k2. ?uu\ i *R\ ?xx\ i) =$
 $u *R\ x + v *R\ y$
unfolding $*$ *sum.If_cases[OF finite_atLeastAtMost[of 1 k1 + k2]]*
 $sum.reindex[OF inj] Collect_mem_eq\ o_def$
unfolding $scaleR_scaleR[symmetric] scaleR_right.sum\ [symmetric] sum_distrib_left[symmetric]$
by (*simp_all add: sum_distrib_left[symmetric] x(2,3) y(2,3) uv(3)*)
qed
qed

lemma *convex_hull_finite*:
fixes $S :: 'a::real_vector\ set$
assumes *finite* S
shows $convex\ hull\ S = \{y. \exists u. (\forall x \in S. 0 \leq u\ x) \wedge sum\ u\ S = 1 \wedge sum\ (\lambda x. u\ x *R\ x)\ S = y\}$
(is $?HULL = _)$
proof (*rule hull_unique [OF __ convexI]; clarify*)
fix x
assume $x \in S$
then show $\exists u. (\forall x \in S. 0 \leq u\ x) \wedge sum\ u\ S = 1 \wedge (\sum\ x \in S. u\ x *R\ x) = x$
by (*rule_tac x= $\lambda y. if\ x=y\ then\ 1\ else\ 0$ in exI*) (*auto simp: sum.delta'[OF assms] sum_delta''[OF assms]*)
next
fix $u\ v :: real$
assume $uw: 0 \leq u\ 0 \leq v\ u + v = 1$
fix ux **assume** ux [*rule_format*]: $\forall x \in S. 0 \leq ux\ x\ sum\ ux\ S = (1::real)$
fix uy **assume** uy [*rule_format*]: $\forall x \in S. 0 \leq uy\ x\ sum\ uy\ S = (1::real)$
have $0 \leq u * ux\ x + v * uy\ x$ **if** $x \in S$ **for** x
by (*simp add: that uv ux(1) uy(1)*)
moreover
have $(\sum\ x \in S. u * ux\ x + v * uy\ x) = 1$
unfolding $sum.distrib$ **and** $sum_distrib_left[symmetric]$ $ux(2)\ uy(2)$
using $uw(3)$ **by** *auto*


```

moreover
have  $(\sum x \in S. (u * ux\ x + v * uy\ x) *_R x) = u *_R (\sum x \in S. ux\ x *_R x) + v *_R$ 
 $(\sum x \in S. uy\ x *_R x)$ 
unfolding scaleR_left_distrib sum.distrib scaleR_scaleR[symmetric] scaleR_right.sum
 $[symmetric]$ 
by auto
ultimately
show  $\exists uc. (\forall x \in S. 0 \leq uc\ x) \wedge sum\ uc\ S = 1 \wedge$ 
 $(\sum x \in S. uc\ x *_R x) = u *_R (\sum x \in S. ux\ x *_R x) + v *_R (\sum x \in S. uy\ x$ 
 $*_R x)$ 
by (rule_tac x=\lambda x. u * ux\ x + v * uy\ x in exI, auto)
qed (use assms in <auto simp: convex_explicit>)

```

Another formulation

Formalized by Lars Schewe.

lemma *convex_hull_explicit:*

fixes $p :: 'a::real_vector\ set$

shows *convex_hull* $p =$

$\{y. \exists S\ u. finite\ S \wedge S \subseteq p \wedge (\forall x \in S. 0 \leq u\ x) \wedge sum\ u\ S = 1 \wedge sum\ (\lambda v. u\ v$

$*_R\ v)\ S = y\}$

(*is ?lhs = ?rhs*)

proof (*intro subset_antisym subsetI*)

fix x

assume $x \in convex_hull\ p$

then obtain $k\ u\ y$ **where**

obt: $\forall i \in \{1..k\}. 0 \leq u\ i \wedge y\ i \in p\ sum\ u\ \{1..k\} = 1\ (\sum i = 1..k. u\ i *_R$

$y\ i) = x$

unfolding *convex_hull_indexed* **by auto**

have *fin*: *finite* $\{1..k\}$ **by auto**

{

fix j

assume $j \in \{1..k\}$

then have $y\ j \in p \wedge 0 \leq sum\ u\ \{i. Suc\ 0 \leq i \wedge i \leq k \wedge y\ i = y\ j\}$

by (*metis (mono_tags, lifting) One_nat_def atLeastAtMost_iff mem_Collect_eq*
obt(1) sum_nonneg)

}

moreover have $(\sum v \in y\ ' \{1..k\}. sum\ u\ \{i \in \{1..k\}. y\ i = v\}) = 1$

unfolding *sum.image_gen[OF fin, symmetric]* **using** *obt(2)* **by auto**

moreover have $(\sum v \in y\ ' \{1..k\}. sum\ u\ \{i \in \{1..k\}. y\ i = v\} *_R\ v) = x$

using *sum.image_gen[OF fin, of \lambda i. u\ i *_R\ y\ i\ y, symmetric]*

unfolding *scaleR_left.sum* **using** *obt(3)* **by auto**

ultimately

have $\exists S\ u. finite\ S \wedge S \subseteq p \wedge (\forall x \in S. 0 \leq u\ x) \wedge sum\ u\ S = 1 \wedge (\sum v \in S. u$

$v *_R\ v) = x$

by (*smt (verit, ccfv_SIG) imageE mem_Collect_eq obt(1) subsetI sum.cong*
sum.infinite sum_nonneg)

then show $x \in ?rhs$ **by auto**

next

```

fix y
assume y ∈ ?rhs
then obtain S u where
  S: finite S S ⊆ p ∀ x ∈ S. 0 ≤ u x sum u S = 1 (∑ v ∈ S. u v *R v) = y
  by auto
obtain f where f: inj_on f {1..card S} f ‘ {1..card S} = S
  using ex_bij_betw_nat_finite_1[OF S(1)] unfolding bij_betw_def by auto
then have 0 ≤ u (f i) f i ∈ p if i ∈ {1..card S} for i
  using S ⟨i ∈ {1..card S}⟩ by blast+
moreover
{
  fix y
  assume y ∈ S
  then obtain i where i ∈ {1..card S} f i = y
  by (metis f(2) image_iff)
  then have {x. Suc 0 ≤ x ∧ x ≤ card S ∧ f x = y} = {i}
  using f(1) inj_onD by fastforce
  then have (∑ x ∈ {x ∈ {1..card S}. f x = y}. u (f x)) = u y
    (∑ x ∈ {x ∈ {1..card S}. f x = y}. u (f x) *R f x) = u y *R y
  by (simp_all add: sum_constant_scaleR ⟨f i = y⟩)
}
then have (∑ x = 1..card S. u (f x)) = 1 (∑ i = 1..card S. u (f i) *R f i) = y
  by (metis (mono_tags, lifting) S(4,5) f sum.reindex_cong)+
ultimately
show y ∈ convex hull p
  unfolding convex_hull_indexed
  by (smt (verit, del_insts) mem_Collect_eq sum.cong)
qed

```

A stepping theorem for that expansion

lemma convex_hull_finite_step:

fixes S :: 'a::real_vector set

assumes finite S

shows

(∃ u. (∀ x ∈ insert a S. 0 ≤ u x) ∧ sum u (insert a S) = w ∧ sum (λx. u x *_R x) (insert a S) = y)

↔ (∃ v ≥ 0. ∃ u. (∀ x ∈ S. 0 ≤ u x) ∧ sum u S = w - v ∧ sum (λx. u x *_R x) S = y - v *_R a)

(**is** ?lhs = ?rhs)

proof (cases a ∈ S)

case True

then have *: insert a S = S **by** auto

show ?thesis

proof

assume ?lhs

then show ?rhs

unfolding * **by** force

next

```

have fin: finite (insert a S) using assms by auto
assume ?rhs
then obtain v u where uv:  $v \geq 0 \ \forall x \in S. 0 \leq u \ x \ \text{sum } u \ S = w - v \ (\sum_{x \in S} u \ x \ *_R \ x) = y - v \ *_R \ a$ 
by auto
then show ?lhs
using uv True assms
apply (rule_tac  $x = \lambda x. (\text{if } a = x \ \text{then } v \ \text{else } 0) + u \ x$  in exI)
apply (auto simp: sum_clauses scaleR_left_distrib sum.distrib sum_delta''[OF fin])
done
qed
next
case False
show ?thesis
proof
assume ?lhs
then obtain u where u:  $\forall x \in \text{insert } a \ S. 0 \leq u \ x \ \text{sum } u \ (\text{insert } a \ S) = w$ 
 $(\sum_{x \in \text{insert } a \ S} u \ x \ *_R \ x) = y$ 
by auto
then show ?rhs
using  $u \ \langle a \notin S \rangle$  by (rule_tac  $x = u \ a$  in exI) (auto simp: sum_clauses assms)
next
assume ?rhs
then obtain v u where uv:  $v \geq 0 \ \forall x \in S. 0 \leq u \ x \ \text{sum } u \ S = w - v \ (\sum_{x \in S} u \ x \ *_R \ x) = y - v \ *_R \ a$ 
by auto
moreover
have  $(\sum_{x \in S} \text{if } a = x \ \text{then } v \ \text{else } u \ x) = \text{sum } u \ S \ (\sum_{x \in S} (\text{if } a = x \ \text{then } v \ \text{else } u \ x) \ *_R \ x) = (\sum_{x \in S} u \ x \ *_R \ x)$ 
using False by (auto intro!: sum.cong)
ultimately show ?lhs
using False by (rule_tac  $x = \lambda x. \text{if } a = x \ \text{then } v \ \text{else } u \ x$  in exI) (auto simp: sum_clauses(2)[OF assms])
qed
qed

```

Hence some special cases

lemma *convex_hull_2*: $\text{convex hull } \{a, b\} = \{u \ *_R \ a + v \ *_R \ b \mid u \ v. 0 \leq u \wedge 0 \leq v \wedge u + v = 1\}$
(is ?lhs = ?rhs)

proof –

have **: *finite* $\{b\}$ **by** *auto*

have $\bigwedge x \ v \ u. [0 \leq v; v \leq 1; (1 - v) \ *_R \ b = x - v \ *_R \ a]$
 $\implies \exists u \ v. x = u \ *_R \ a + v \ *_R \ b \wedge 0 \leq u \wedge 0 \leq v \wedge u + v = 1$

by (*metis add commute diff_add_cancel diff_ge_0_iff_ge*)

moreover

have $\bigwedge u \ v. [0 \leq u; 0 \leq v; u + v = 1]$

$$\implies \exists p \geq 0. \exists q. 0 \leq q \wedge q b \wedge q b = 1 - p \wedge q b *_R b = u *_R a + v *_R b - p *_R a$$

```

apply (rule_tac x=u in exI, simp)
apply (rule_tac x= $\lambda x. v$  in exI, simp)
done
ultimately show ?thesis
using convex_hull_finite_step[OF **, of a 1]
by (auto simp add: convex_hull_finite)
qed

```

lemma convex_hull_2_alt: $\text{convex hull } \{a, b\} = \{a + u *_R (b - a) \mid u. 0 \leq u \wedge u \leq 1\}$

```

unfolding convex_hull_2
proof (rule Collect_cong)
have *:  $\bigwedge x y :: \text{real}. x + y = 1 \iff x = 1 - y$ 
by auto
fix x
show ( $\exists v u. x = v *_R a + u *_R b \wedge 0 \leq v \wedge 0 \leq u \wedge v + u = 1$ )  $\iff$ 
( $\exists u. x = a + u *_R (b - a) \wedge 0 \leq u \wedge u \leq 1$ )
apply (simp add: *)
by (rule ex_cong1) (auto simp: algebra_simps)
qed

```

lemma convex_hull_3:

$$\text{convex hull } \{a, b, c\} = \{u *_R a + v *_R b + w *_R c \mid u v w. 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1\}$$

```

proof -
have fin: finite {a,b,c} finite {b,c} finite {c}
by auto
have *:  $\bigwedge x y z :: \text{real}. x + y + z = 1 \iff x = 1 - y - z$ 
by (auto simp: field_simps)
show ?thesis
unfolding convex_hull_finite[OF fin(1)] and convex_hull_finite_step[OF fin(2)] and *
unfolding convex_hull_finite_step[OF fin(3)]
apply (rule Collect_cong, simp)
apply auto
apply (rule_tac x=va in exI)
apply (rule_tac x=u c in exI, simp)
apply (rule_tac x=1 - v - w in exI, simp)
apply (rule_tac x=v in exI, simp)
apply (rule_tac x= $\lambda x. w$  in exI, simp)
done
qed

```

lemma convex_hull_3_alt:

$$\text{convex hull } \{a, b, c\} = \{a + u *_R (b - a) + v *_R (c - a) \mid u v. 0 \leq u \wedge 0 \leq v \wedge u + v \leq 1\}$$

```

proof -

```

```

have *:  $\bigwedge x y z :: \text{real}. x + y + z = 1 \longleftrightarrow x = 1 - y - z$ 
  by auto
show ?thesis
  unfolding convex_hull_3
  apply (auto simp: *)
  apply (rule_tac x=v in exI)
  apply (rule_tac x=w in exI)
  apply (simp add: algebra_simps)
  apply (rule_tac x=u in exI)
  apply (rule_tac x=v in exI)
  apply (simp add: algebra_simps)
done
qed

```

1.7.11 Relations among closure notions and corresponding hulls

```

lemma affine_imp_convex: affine s  $\implies$  convex s
  unfolding affine_def convex_def by auto

```

```

lemma convex_affine_hull [simp]: convex (affine hull S)
  by (simp add: affine_imp_convex)

```

```

lemma subspace_imp_convex: subspace s  $\implies$  convex s
  using subspace_imp_affine affine_imp_convex by auto

```

```

lemma convex_hull_subset_span: (convex hull s)  $\subseteq$  (span s)
  by (metis hull_minimal span_superset subspace_imp_convex subspace_span)

```

```

lemma convex_hull_subset_affine_hull: (convex hull s)  $\subseteq$  (affine hull s)
  by (metis affine_affine_hull affine_imp_convex hull_minimal hull_subset)

```

```

lemma aff_dim_convex_hull:
  fixes S :: 'n::euclidean_space set
  shows aff_dim (convex hull S) = aff_dim S
  by (smt (verit) aff_dim_affine_hull aff_dim_subset convex_hull_subset_affine_hull
    hull_subset)

```

1.7.12 Caratheodory's theorem

```

lemma convex_hull_caratheodory_aff_dim:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p =
    {y.  $\exists S u. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{aff\_dim } p + 1 \wedge$ 
      ( $\forall x \in S. 0 \leq u x$ )  $\wedge \text{sum } u S = 1 \wedge \text{sum } (\lambda v. u v *_{\mathbb{R}} v) S = y$ }
  unfolding convex_hull_explicit_set_eq_iff mem_Collect_eq
proof (intro allI iffI)
  fix y
  let ?P =  $\lambda n. \exists S u. \text{finite } S \wedge \text{card } S = n \wedge S \subseteq p \wedge (\forall x \in S. 0 \leq u x) \wedge$ 

```

```

    sum u S = 1 ∧ (∑ v∈S. u v *R v) = y
  assume ∃ S u. finite S ∧ S ⊆ p ∧ (∀ x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧ (∑ v∈S.
u v *R v) = y
  then obtain N where ?P N by auto
  then have ∃ n≤N. (∀ k<n. ¬ ?P k) ∧ ?P n
    by (rule_tac ex_least_nat_le, auto)
  then obtain n where ?P n and smallest: ∀ k<n. ¬ ?P k
    by blast
  then obtain S u where S: finite S card S = n S⊆p
    and u: ∀ x∈S. 0 ≤ u x sum u S = 1 (∑ v∈S. u v *R v) = y by auto

  have card S ≤ aff_dim p + 1
  proof (rule ccontr, simp only: not_le)
    assume aff_dim p + 1 < card S
    then have affine_dependent S
      by (smt (verit) independent_card_le_aff_dim S(3))
    then obtain w v where wv: sum w S = 0 v∈S w v ≠ 0 (∑ v∈S. w v *R v)
= 0
      using affine_dependent_explicit_finite[OF S(1)] by auto
    define i where i = (λv. (u v) / (- w v)) ‘ {v∈S. w v < 0}
    define t where t = Min i
    have ∃ x∈S. w x < 0
      by (smt (verit, best) S(1) sum_pos2 wv)
    then have i ≠ {} unfolding i_def by auto
    then have t ≥ 0
      using Min_ge_iff[of i 0] and S(1) u[unfolded le_less]
      unfolding t_def i_def
      by (auto simp: divide_le_0_iff)
    have t: ∀ v∈S. u v + t * w v ≥ 0
  proof
    fix v
    assume v ∈ S
    then have v: 0 ≤ u v
      using u(1) by blast
    show 0 ≤ u v + t * w v
    proof (cases w v < 0)
      case False
      thus ?thesis using v ⟨t≥0⟩ by auto
    next
      case True
      then have t ≤ u v / (- w v)
        using ⟨v∈S⟩ S unfolding t_def i_def by (auto intro: Min_le)
      then show ?thesis
        unfolding real_0_le_add_iff
        using True neg_le_minus_divide_eq by auto
    qed
  qed
  obtain a where a ∈ S and t = (λv. (u v) / (- w v)) a and w a < 0
    using Min_in[OF _ ⟨i≠{}⟩] and S(1) unfolding i_def t_def by auto

```

```

then have  $a: a \in S \ u \ a + t * w \ a = 0$  by auto
have  $*$ :  $\bigwedge f. \text{sum } f \ (S - \{a\}) = \text{sum } f \ S - ((f \ a)::'b::\text{ab\_group\_add})$ 
unfolding sum.remove[OF S(1) ‹a∈S›] by auto
have  $(\sum v \in S. u \ v + t * w \ v) = 1$ 
by (metis add.right_neutral mult_zero_right sum.distrib sum_distrib_left
 $u(2) \ wv(1)$ )
moreover have  $(\sum v \in S. u \ v *_{R} \ v + (t * w \ v) *_{R} \ v) - (u \ a *_{R} \ a + (t * w \ a) *_{R} \ a) = y$ 
unfolding sum.distrib u(3) scaleR_scaleR[symmetric] scaleR_right.sum
 $[symmetric] \ wv(4)$ 
using  $a(2)$   $[THEN \ eq\_neg\_iff\_add\_eq\_0 \ [THEN \ iffD2]]$  by simp
ultimately have  $?P \ (n - 1)$ 
apply (rule_tac x=(S - {a}) in exI)
apply (rule_tac x=λv. u v + t * w v in exI)
using  $S \ t \ a$ 
apply (auto simp: * scaleR_left_distrib)
done
then show False
using smallest[THEN spec[where x=n - 1]] by auto
qed
then show  $\exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{aff\_dim } p + 1 \wedge$ 
 $(\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge (\sum v \in S. u \ v *_{R} \ v) = y$ 
using  $S \ u$  by auto
qed auto

```

lemma *caratheodory_aff_dim*:

```

fixes  $p :: ('a::\text{euclidean\_space}) \text{ set}$ 
shows  $\text{convex hull } p = \{x. \exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{aff\_dim } p + 1 \wedge x \in \text{convex hull } S\}$ 
(is ?lhs = ?rhs)

```

proof

```

have  $\bigwedge x \ S \ u. [\text{finite } S; S \subseteq p; \text{int } (\text{card } S) \leq \text{aff\_dim } p + 1; \forall x \in S. 0 \leq u \ x; \text{sum } u \ S = 1]$ 
 $\implies (\sum v \in S. u \ v *_{R} \ v) \in \text{convex hull } S$ 
by (metis (mono_tags, lifting) convex_convex_hull convex_explicit_hull_subset)
then show  $?lhs \subseteq ?rhs$ 
by (subst convex_hull_caratheodory_aff_dim, auto)
qed (use hull_mono in auto)

```

lemma *convex_hull_caratheodory*:

```

fixes  $p :: ('a::\text{euclidean\_space}) \text{ set}$ 
shows  $\text{convex hull } p =$ 
 $\{y. \exists S \ u. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1 \wedge$ 
 $(\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge \text{sum } (\lambda v. u \ v *_{R} \ v) \ S = y\}$ 
(is ?lhs = ?rhs)

```

proof (*intro set_eqI iffI*)

fix x

assume $x \in ?lhs$ **then show** $x \in ?rhs$

unfolding *convex_hull_caratheodory_aff_dim*

using *aff_dim_le_DIM* [of *p*] **by** *fastforce*
qed (*auto simp: convex_hull_explicit*)

theorem *caratheodory*:

convex hull p =
 $\{x::'a::\text{euclidean_space}. \exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1 \wedge x \in \text{convex hull } S\}$

proof *safe*

fix *x*

assume $x \in \text{convex hull } p$

then obtain *S u* **where** $\text{finite } S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1$

$\forall x \in S. 0 \leq u \cdot x \wedge \sum u \cdot S = 1 \wedge (\sum v \in S. u \cdot v *_{\mathbb{R}} v) = x$

unfolding *convex_hull_caratheodory* **by** *auto*

then show $\exists S. \text{finite } S \wedge S \subseteq p \wedge \text{card } S \leq \text{DIM}('a) + 1 \wedge x \in \text{convex hull } S$

using *convex_hull_finite* **by** *fastforce*

qed (*use hull_mono in force*)

1.7.13 Some Properties of subset of standard basis

lemma *affine_hull_substd_basis*:

assumes $d \subseteq \text{Basis}$

shows $\text{affine hull } (\text{insert } 0 \ d) = \{x::'a::\text{euclidean_space}. \forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0\}$

(**is** $\text{affine hull } (\text{insert } 0 \ ?A) = ?B$)

proof *–*

have $*$: $\bigwedge A. (+) (0::'a) \cdot A = A \wedge A. (+) (- (0::'a)) \cdot A = A$

by *auto*

show *?thesis*

unfolding *affine_hull_insert_span_gen span_substd_basis* [*OF assms,symmetric*]

** ..*

qed

lemma *affine_hull_convex_hull* [*simp*]: $\text{affine hull } (\text{convex hull } S) = \text{affine hull } S$

by (*metis Int_absorb1 Int_absorb2 convex_hull_subset_affine_hull hull_hull hull_mono hull_subset*)

1.7.14 Moving and scaling convex hulls

lemma *convex_hull_set_plus*:

$\text{convex hull } (S + T) = \text{convex hull } S + \text{convex hull } T$

by (*simp add: set_plus_image linear_iff scaleR_right_distrib convex_hull_Times*

flip: convex_hull_linear_image)

lemma *translation_eq_singleton_plus*: $(\lambda x. a + x) \cdot T = \{a\} + T$

unfolding *set_plus_def* **by** *auto*

lemma *convex_hull_translation*:

$\text{convex hull } ((\lambda x. a + x) \cdot S) = (\lambda x. a + x) \cdot (\text{convex hull } S)$

by (simp add: convex_hull_set_plus translation_eq_singleton_plus)

lemma convex_hull_scaling:

$\text{convex_hull } ((\lambda x. c *_{\mathbb{R}} x) \text{ ` } S) = (\lambda x. c *_{\mathbb{R}} x) \text{ ` } (\text{convex_hull } S)$

by (simp add: convex_hull_linear_image)

lemma convex_hull_affinity:

$\text{convex_hull } ((\lambda x. a + c *_{\mathbb{R}} x) \text{ ` } S) = (\lambda x. a + c *_{\mathbb{R}} x) \text{ ` } (\text{convex_hull } S)$

by (metis convex_hull_scaling convex_hull_translation image_image)

1.7.15 Convexity of cone hulls

lemma convex_cone_hull:

assumes convex S

shows convex (cone hull S)

proof (rule convexI)

fix x y

assume xy: $x \in \text{cone hull } S$ $y \in \text{cone hull } S$

then have $S \neq \{\}$

using cone_hull_empty_iff[of S] by auto

fix u v :: real

assume uv: $u \geq 0$ $v \geq 0$ $u + v = 1$

then have *: $u *_{\mathbb{R}} x \in \text{cone hull } S$ $v *_{\mathbb{R}} y \in \text{cone hull } S$

by (simp_all add: cone_cone_hull mem_cone uv xy)

then obtain cx :: real and xx

and cy :: real and yy where $x: u *_{\mathbb{R}} x = cx *_{\mathbb{R}} xx$ $cx \geq 0$ $xx \in S$

and $y: v *_{\mathbb{R}} y = cy *_{\mathbb{R}} yy$ $cy \geq 0$ $yy \in S$

using cone_hull_expl[of S] by auto

have $u *_{\mathbb{R}} x + v *_{\mathbb{R}} y \in \text{cone hull } S$ if $cx + cy \leq 0$

using *(1) nless_le that x(2) y by fastforce

moreover

have $u *_{\mathbb{R}} x + v *_{\mathbb{R}} y \in \text{cone hull } S$ if $cx + cy > 0$

proof –

have $(cx / (cx + cy)) *_{\mathbb{R}} xx + (cy / (cx + cy)) *_{\mathbb{R}} yy \in S$

using assms mem_convex_alt[of S xx yy cx cy] x y that by auto

then have $cx *_{\mathbb{R}} xx + cy *_{\mathbb{R}} yy \in \text{cone hull } S$

using mem_cone_hull[of $(cx / (cx + cy)) *_{\mathbb{R}} xx + (cy / (cx + cy)) *_{\mathbb{R}} yy$ S $cx + cy$]

$\langle cx + cy > 0 \rangle$

by (auto simp: scaleR_right_distrib)

then show ?thesis

using x y by auto

qed

moreover have $cx + cy \leq 0 \vee cx + cy > 0$ by auto

ultimately show $u *_{\mathbb{R}} x + v *_{\mathbb{R}} y \in \text{cone hull } S$ by blast

qed

lemma cone_convex_hull:

assumes cone S

shows *cone* (*convex hull S*)
by (*metis* (*no_types*, *lifting*) *affine_hull_convex_hull* *affine_hull_eq_empty*
assms *cone_iff* *convex_hull_scaling* *hull_inc*)

1.8 Conic sets and conic hull

definition *conic* :: 'a::real_vector set \Rightarrow bool
where *conic S* $\equiv \forall x c. x \in S \longrightarrow 0 \leq c \longrightarrow (c *_R x) \in S$

lemma *conicD*: $\llbracket \text{conic } S; x \in S; 0 \leq c \rrbracket \Longrightarrow (c *_R x) \in S$
by (*meson* *conic_def*)

lemma *subspace_imp_conic*: *subspace S* \Longrightarrow *conic S*
by (*simp* *add*: *conic_def* *subspace_def*)

lemma *conic_empty* [*simp*]: *conic* {}
using *conic_def* **by** *blast*

lemma *conic_UNIV*: *conic UNIV*
by (*simp* *add*: *conic_def*)

lemma *conic_Inter*: $(\bigwedge S. S \in \mathcal{F} \Longrightarrow \text{conic } S) \Longrightarrow \text{conic}(\bigcap \mathcal{F})$
by (*simp* *add*: *conic_def*)

lemma *conic_linear_image*:
 $\llbracket \text{conic } S; \text{linear } f \rrbracket \Longrightarrow \text{conic}(f ' S)$
by (*smt* (*verit*) *conic_def* *image_iff* *linear.scaleR*)

lemma *conic_linear_image_eq*:
 $\llbracket \text{linear } f; \text{inj } f \rrbracket \Longrightarrow \text{conic}(f ' S) \longleftrightarrow \text{conic } S$
by (*smt* (*verit*) *conic_def* *conic_linear_image* *inj_image_mem_iff* *linear_cmul*)

lemma *conic_mul*: $\llbracket \text{conic } S; x \in S; 0 \leq c \rrbracket \Longrightarrow (c *_R x) \in S$
using *conic_def* **by** *blast*

lemma *conic_conic_hull*: *conic*(*conic hull S*)
by (*metis* (*no_types*, *lifting*) *conic_Inter* *hull_def* *mem_Collect_eq*)

lemma *conic_hull_eq*: (*conic hull S* = *S*) \longleftrightarrow *conic S*
by (*metis* *conic_conic_hull* *hull_same*)

lemma *conic_hull_UNIV* [*simp*]: *conic hull UNIV* = *UNIV*
by *simp*

lemma *conic_negations*: *conic S* \Longrightarrow *conic* (*image uminus S*)
by (*auto* *simp*: *conic_def* *image_iff*)

lemma *conic_span* [*iff*]: *conic*(*span S*)
by (*simp* *add*: *subspace_imp_conic*)

lemma *conic_hull_explicit*:

conic hull $S = \{c *_R x \mid c x. 0 \leq c \wedge x \in S\}$

proof (*rule hull_unique*)

show $S \subseteq \{c *_R x \mid c x. 0 \leq c \wedge x \in S\}$

by (*metis (no_types) cone_hull_expl hull_subset*)

show *conic* $\{c *_R x \mid c x. 0 \leq c \wedge x \in S\}$

using *mult_nonneg_nonneg* **by** (*force simp: conic_def*)

qed (*auto simp: conic_def*)

lemma *conic_hull_as_image*:

conic hull $S = (\lambda z. \text{fst } z *_R \text{snd } z) \text{ ' } (\{0..\} \times S)$

by (*force simp: conic_hull_explicit*)

lemma *conic_hull_linear_image*:

linear $f \implies \text{conic hull } f \text{ ' } S = f \text{ ' } (\text{conic hull } S)$

by (*force simp: conic_hull_explicit image_iff set_eq_iff linear_scale*)

lemma *conic_hull_image_scale*:

assumes $\bigwedge x. x \in S \implies 0 < c x$

shows *conic hull* $(\lambda x. c x *_R x) \text{ ' } S = \text{conic hull } S$

proof

show *conic hull* $(\lambda x. c x *_R x) \text{ ' } S \subseteq \text{conic hull } S$

proof (*rule hull_minimal*)

show $(\lambda x. c x *_R x) \text{ ' } S \subseteq \text{conic hull } S$

using *assms conic_hull_explicit* **by** *fastforce*

qed (*simp add: conic_conic_hull*)

show *conic hull* $S \subseteq \text{conic hull } (\lambda x. c x *_R x) \text{ ' } S$

proof (*rule hull_minimal*)

show $S \subseteq \text{conic hull } (\lambda x. c x *_R x) \text{ ' } S$

proof *clarsimp*

fix x

assume $x \in S$

then have $x = \text{inverse}(c x) *_R c x *_R x$

using *assms* **by** *fastforce*

then show $x \in \text{conic hull } (\lambda x. c x *_R x) \text{ ' } S$

by (*smt (verit, best) <x ∈ S> assms conic_conic_hull conic_mul hull_inc image_eqI inverse_nonpositive_iff_nonpositive*)

qed

qed (*simp add: conic_conic_hull*)

qed

lemma *convex_conic_hull*:

assumes *convex* S

shows *convex* $(\text{conic hull } S)$

proof (*clarsimp simp add: conic_hull_explicit convex_alt*)

fix $c x d y$ **and** $u :: \text{real}$

assume $\S: (0::\text{real}) \leq c x \in S \ (0::\text{real}) \leq d y \in S \ 0 \leq u \ u \leq 1$

show $\exists c'' x''. ((1 - u) * c) *_R x + (u * d) *_R y = c'' *_R x'' \wedge 0 \leq c'' \wedge x''$

```

∈ S
proof (cases (1 - u) * c = 0)
  case True
    with ⟨0 ≤ d⟩ ⟨y ∈ S⟩ ⟨0 ≤ u⟩
    show ?thesis by force
  next
    case False
    define ξ where ξ ≡ (1 - u) * c + u * d
    have *: c * u ≤ c
      by (simp add: § mult_left_le)
    have ξ > 0
      using False § by (smt (verit, best) ξ_def split_mult_pos_le)
    then have **: c + d * u = ξ + c * u
      by (simp add: ξ_def mult.commute right_diff_distrib')
    show ?thesis
    proof (intro exI conjI)
      show 0 ≤ ξ
        using ⟨0 < ξ⟩ by auto
      show ((1 - u) * c) *R x + (u * d) *R y = ξ *R (((1 - u) * c / ξ) *R x +
(u * d / ξ) *R y)
        using ⟨ξ > 0⟩ by (simp add: algebra_simps diff_divide_distrib)
      show ((1 - u) * c / ξ) *R x + (u * d / ξ) *R y ∈ S
        using ⟨0 < ξ⟩
        by (intro convexD [OF assms]) (auto simp: § field_split_simps * **)
    qed
  qed
qed

lemma conic_halfspace_le: conic {x. a · x ≤ 0}
  by (auto simp: conic_def mult_le_0_iff)

lemma conic_halfspace_ge: conic {x. a · x ≥ 0}
  by (auto simp: conic_def mult_le_0_iff)

lemma conic_hull_empty [simp]: conic hull {} = {}
  by (simp add: conic_hull_eq)

lemma conic_contains_0: conic S ⇒ (0 ∈ S ↔ S ≠ {})
  by (simp add: Convex.cone_def cone_contains_0 conic_def)

lemma conic_hull_eq_empty: conic hull S = {} ↔ (S = {})
  using conic_hull_explicit by fastforce

lemma conic_sums: [[conic S; conic T]] ⇒ conic (⋃ x ∈ S. ⋃ y ∈ T. {x + y})
  by (simp add: conic_def) (metis scaleR_right_distrib)

lemma conic_Times: [[conic S; conic T]] ⇒ conic(S × T)
  by (auto simp: conic_def)

```

```

lemma conic_Times_eq:
  conic(S × T) ↔ S = {} ∨ T = {} ∨ conic S ∧ conic T (is ?lhs = ?rhs)
proof
  show ?lhs ⇒ ?rhs
    by (force simp: conic_def)
  show ?rhs ⇒ ?lhs
    by (force simp: conic_Times)
qed

lemma conic_hull_0 [simp]: conic hull {0} = {0}
  by (simp add: conic_hull_eq subspace_imp_conic)

lemma conic_hull_contains_0 [simp]: 0 ∈ conic hull S ↔ (S ≠ {})
  by (simp add: conic_conic_hull conic_contains_0 conic_hull_eq_empty)

lemma conic_hull_eq_sing:
  conic hull S = {x} ↔ S = {0} ∧ x = 0
proof
  show conic hull S = {x} ⇒ S = {0} ∧ x = 0
    by (metis conic_conic_hull conic_contains_0 conic_def conic_hull_eq hull_inc
      insert_not_empty singleton_iff)
qed simp

lemma conic_hull_Int_affine_hull:
  assumes T ⊆ S 0 ∉ affine hull S
  shows (conic hull T) ∩ (affine hull S) = T
proof -
  have TaffS: T ⊆ affine hull S
    using ⟨T ⊆ S⟩ hull_subset by fastforce
  moreover
  have conic hull T ∩ affine hull S ⊆ T
proof (clarsimp simp: conic_hull_explicit)
  fix c x
  assume c *R x ∈ affine hull S
    and 0 ≤ c
    and x ∈ T
  show c *R x ∈ T
proof (cases c=1)
  case True
  then show ?thesis
    by (simp add: ⟨x ∈ T⟩)
  next
  case False
  then have x /R (1 - c) = x + (c * inverse (1 - c)) *R x
    by (smt (verit, ccfv_SIG) diff_add_cancel mult commute real_vector_affinity_eq
      scaleR_collapse scaleR_scaleR)
  then have 0 = inverse(1 - c) *R c *R x + (1 - inverse(1 - c)) *R x
    by (simp add: algebra_simps)
  then have 0 ∈ affine hull S

```

```

      by (smt (verit) ⟨c *R x ∈ affine hull S⟩ ⟨x ∈ T⟩ affine_affine_hull TaffS
in_mono mem_affine)
    then show ?thesis
      using assms by auto
    qed
  qed
  ultimately show ?thesis
    by (auto simp: hull_inc)
  qed

```

1.9 Convex cones and corresponding hulls

definition *convex_cone* :: '*a*::*real_vector set* ⇒ *bool*
 where *convex_cone* ≡ λ*S*. *S* ≠ {} ∧ *convex* *S* ∧ *conic* *S*

lemma *convex_cone_iff*:
convex_cone *S* ↔
 $0 \in S \wedge (\forall x \in S. \forall y \in S. x + y \in S) \wedge (\forall x \in S. \forall c \geq 0. c *_{\mathbb{R}} x \in S)$
 by (*metis* *cone_def* *conic_contains_0* *conic_def* *convex_cone* *convex_cone_def*)

lemma *convex_cone_add*: $\llbracket \text{convex_cone } S; x \in S; y \in S \rrbracket \implies x + y \in S$
 by (*simp* *add*: *convex_cone_iff*)

lemma *convex_cone_scaleR*: $\llbracket \text{convex_cone } S; 0 \leq c; x \in S \rrbracket \implies c *_{\mathbb{R}} x \in S$
 by (*simp* *add*: *convex_cone_iff*)

lemma *convex_cone_nonempty*: *convex_cone* *S* ⇒ *S* ≠ {}
 by (*simp* *add*: *convex_cone_def*)

lemma *convex_cone_linear_image*:
convex_cone *S* ∧ *linear* *f* ⇒ *convex_cone*(*f* ' *S*)
 by (*simp* *add*: *conic_linear_image* *convex_cone_def* *convex_linear_image*)

lemma *convex_cone_linear_image_eq*:
 $\llbracket \text{linear } f; \text{inj } f \rrbracket \implies (\text{convex_cone}(f ' S) \longleftrightarrow \text{convex_cone } S)$
 by (*simp* *add*: *conic_linear_image_eq* *convex_cone_def*)

lemma *convex_cone_halfspace_ge*: *convex_cone* {*x*. *a* · *x* ≥ 0}
 by (*simp* *add*: *convex_cone_iff* *inner_simps*(2))

lemma *convex_cone_halfspace_le*: *convex_cone* {*x*. *a* · *x* ≤ 0}
 by (*simp* *add*: *convex_cone_iff* *inner_right_distrib* *mult_nonneg_nonpos*)

lemma *convex_cone_contains_0*: *convex_cone* *S* ⇒ 0 ∈ *S*
 using *convex_cone_iff* by *blast*

lemma *convex_cone_Inter*:
 $(\bigwedge S. S \in f \implies \text{convex_cone } S) \implies \text{convex_cone}(\bigcap f)$
 by (*simp* *add*: *convex_cone_iff*)

```

lemma convex_cone_convex_cone_hull: convex_cone(convex_cone hull S)
  by (metis (no_types, lifting) convex_cone_Inter hull_def mem_Collect_eq)

lemma convex_convex_cone_hull: convex(convex_cone hull S)
  by (meson convex_cone_convex_cone_hull convex_cone_def)

lemma conic_convex_cone_hull: conic(convex_cone hull S)
  by (metis convex_cone_convex_cone_hull convex_cone_def)

lemma convex_cone_hull_nonempty: convex_cone hull S  $\neq$  {}
  by (simp add: convex_cone_convex_cone_hull convex_cone_nonempty)

lemma convex_cone_hull_contains_0: 0  $\in$  convex_cone hull S
  by (simp add: convex_cone_contains_0 convex_cone_convex_cone_hull)

lemma convex_cone_hull_add:
   $\llbracket x \in \text{convex\_cone hull } S; y \in \text{convex\_cone hull } S \rrbracket \implies x + y \in \text{convex\_cone hull } S$ 
  by (simp add: convex_cone_add convex_cone_convex_cone_hull)

lemma convex_cone_hull_mul:
   $\llbracket x \in \text{convex\_cone hull } S; 0 \leq c \rrbracket \implies (c *_{\mathbb{R}} x) \in \text{convex\_cone hull } S$ 
  by (simp add: conic_convex_cone_hull conic_mul)

thm convex_sums
lemma convex_cone_sums:
   $\llbracket \text{convex\_cone } S; \text{convex\_cone } T \rrbracket \implies \text{convex\_cone } (\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\})$ 
  by (simp add: convex_cone_def conic_sums convex_sums)

lemma convex_cone_Times:
   $\llbracket \text{convex\_cone } S; \text{convex\_cone } T \rrbracket \implies \text{convex\_cone}(S \times T)$ 
  by (simp add: conic_Times convex_Times convex_cone_def)

lemma convex_cone_Times_D1: convex_cone (S  $\times$  T)  $\implies$  convex_cone S
  by (metis Times_empty conic_Times_eq convex_cone_def convex_convex_hull
convex_hull_Times hull_same_times_eq_iff)

lemma convex_cone_Times_eq:
  convex_cone(S  $\times$  T)  $\longleftrightarrow$  convex_cone S  $\wedge$  convex_cone T
proof (cases S={ }  $\vee$  T={ })
  case True
  then show ?thesis
  by (auto dest: convex_cone_nonempty)
next
  case False
  then have convex_cone (S  $\times$  T)  $\implies$  convex_cone T
  by (metis conic_Times_eq convex_cone_def convex_convex_hull convex_hull_Times
hull_same_times_eq_iff)

```

then show *?thesis*
using *convex_cone_Times convex_cone_Times_D1* **by** *blast*
qed

lemma *convex_cone_hull_Un*:
 $convex_cone\ hull(S \cup T) = (\bigcup x \in convex_cone\ hull\ S. \bigcup y \in convex_cone\ hull\ T. \{x + y\})$
(is *?lhs = ?rhs*)
proof
show *?lhs \subseteq ?rhs*
proof (*rule hull_minimal*)
show $S \cup T \subseteq (\bigcup x \in convex_cone\ hull\ S. \bigcup y \in convex_cone\ hull\ T. \{x + y\})$
apply (*clarsimp simp: subset_iff*)
by (*metis add_0 convex_cone_hull_contains_0 group_cancel.rule0 hull_inc*)
show $convex_cone (\bigcup x \in convex_cone\ hull\ S. \bigcup y \in convex_cone\ hull\ T. \{x + y\})$
by (*simp add: convex_cone_convex_cone_hull convex_cone_sums*)
qed
next
show *?rhs \subseteq ?lhs*
by *clarify (metis convex_cone_hull_add hull_mono le_sup_iff subsetD subsetI)*
qed

lemma *convex_cone_singleton [iff]*: $convex_cone \{0\}$
by (*simp add: convex_cone_iff*)

lemma *convex_hull_subset_convex_cone_hull*:
 $convex\ hull\ S \subseteq convex_cone\ hull\ S$
by (*simp add: convex_convex_cone_hull hull_minimal hull_subset*)

lemma *conic_hull_subset_convex_cone_hull*:
 $conic\ hull\ S \subseteq convex_cone\ hull\ S$
by (*simp add: conic_convex_cone_hull hull_minimal hull_subset*)

lemma *subspace_imp_convex_cone*: $subspace\ S \implies convex_cone\ S$
by (*simp add: convex_cone_iff subspace_def*)

lemma *convex_cone_span*: $convex_cone(span\ S)$
by (*simp add: subspace_imp_convex_cone*)

lemma *convex_cone_negations*:
 $convex_cone\ S \implies convex_cone\ (image\ uminus\ S)$
by (*simp add: convex_cone_linear_image_module_hom_uminus*)

lemma *subspace_convex_cone_symmetric*:
 $subspace\ S \iff convex_cone\ S \wedge (\forall x \in S. -x \in S)$
by (*smt (verit) convex_cone_iff scaleR_left.minus subspace_def subspace_neg*)


```

lemma convex_cone_hull_separate_nonempty:
  assumes  $S \neq \{\}$ 
  shows  $\text{convex\_cone hull } S = \text{conic hull } (\text{convex hull } S)$  (is ?lhs = ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs
    by (metis assms conic_conic_hull convex_cone_def convex_conic_hull convex_convex_hull hull_subset subset_empty subset_hull)
  show ?rhs  $\subseteq$  ?lhs
    by (simp add: conic_convex_cone_hull convex_hull_subset_convex_cone_hull subset_hull)
qed

```

```

lemma convex_cone_hull_empty [simp]:  $\text{convex\_cone hull } \{\} = \{0\}$ 
  by (metis convex_cone_hull_contains_0 convex_cone_singleton hull_redundant hull_same)

```

```

lemma convex_cone_hull_separate:
   $\text{convex\_cone hull } S = \text{insert } 0 (\text{conic hull } (\text{convex hull } S))$ 
proof(cases  $S = \{\}$ )
  case False
  then show ?thesis
    using convex_cone_hull_contains_0 convex_cone_hull_separate_nonempty
by blast
qed auto

```

```

lemma convex_cone_hull_convex_hull_nonempty:
   $S \neq \{\} \implies \text{convex\_cone hull } S = (\bigcup x \in \text{convex hull } S. \bigcup c \in \{0..\}. \{c *_R x\})$ 
  by (force simp: convex_cone_hull_separate_nonempty conic_hull_as_image)

```

```

lemma convex_cone_hull_convex_hull:
   $\text{convex\_cone hull } S = \text{insert } 0 (\bigcup x \in \text{convex hull } S. \bigcup c \in \{0..\}. \{c *_R x\})$ 
  by (force simp: convex_cone_hull_separate conic_hull_as_image)

```

```

lemma convex_cone_hull_linear_image:
   $\text{linear } f \implies \text{convex\_cone hull } (f ' S) = \text{image } f (\text{convex\_cone hull } S)$ 
  by (metis (no_types, lifting) conic_hull_linear_image convex_cone_hull_separate convex_hull_linear_image image_insert linear_0)

```

1.9.1 Radon's theorem

Formalized by Lars Schewe.

```

lemma Radon_ex_lemma:
  assumes finite c affine_dependent c
  shows  $\exists u. \text{sum } u \ c = 0 \wedge (\exists v \in c. u \ v \neq 0) \wedge \text{sum } (\lambda v. u \ v *_R \ v) \ c = 0$ 
  using affine_dependent_explicit_finite assms by blast

```

```

lemma Radon_s_lemma:
  assumes finite S
  and  $\text{sum } f \ S = (0::\text{real})$ 

```

shows $\text{sum } f \{x \in S. 0 < f x\} = - \text{sum } f \{x \in S. f x < 0\}$
proof –
have $\bigwedge x. (\text{if } f x < 0 \text{ then } f x \text{ else } 0) + (\text{if } 0 < f x \text{ then } f x \text{ else } 0) = f x$
by *auto*
then show *?thesis*
using *assms* **by** (*simp* *add: sum.inter_filter flip: sum.distrib add_eq_0_iff*)
qed

lemma *Radon_v_lemma*:
assumes *finite S*
and $\text{sum } f S = 0$
and $\forall x. g x = (0::\text{real}) \longrightarrow f x = (0::\text{'a}::\text{euclidean_space})$
shows $(\text{sum } f \{x \in S. 0 < g x\}) = - \text{sum } f \{x \in S. g x < 0\}$
proof –
have $\bigwedge x. (\text{if } 0 < g x \text{ then } f x \text{ else } 0) + (\text{if } g x < 0 \text{ then } f x \text{ else } 0) = f x$
using *assms* **by** *auto*
then show *?thesis*
using *assms* **by** (*simp* *add: sum.inter_filter eq_neg_iff_add_eq_0 flip: sum.distrib add_eq_0_iff*)
qed

lemma *Radon_partition*:
assumes *finite C affine_dependent C*
shows $\exists M P. M \cap P = \{\} \wedge M \cup P = C \wedge (\text{convex hull } M) \cap (\text{convex hull } P) \neq \{\}$
proof –
obtain *u v* **where** $uv: \text{sum } u C = 0 \ v \in C \ u \ v \neq 0 \ (\sum v \in C. u \ v *_R v) = 0$
using *Radon_ex_lemma[OF assms]* **by** *auto*
have *fin*: $\text{finite } \{x \in C. 0 < u x\} \ \text{finite } \{x \in C. 0 > u x\}$
using *assms(1)* **by** *auto*
define *z* **where** $z = \text{inverse } (\text{sum } u \{x \in C. u x > 0\}) *_R \text{sum } (\lambda x. u x *_R x) \{x \in C. u x > 0\}$
have $\text{sum } u \{x \in C. 0 < u x\} \neq 0$
proof (*cases u v ≥ 0*)
case *False*
then have $u \ v < 0$ **by** *auto*
then show *?thesis*
by (*smt (verit) assms(1) fin(1) mem_Collect_eq sum.neutral_const sum_mono_inv uv*)
next
case *True*
with *fin uv* **show** $\text{sum } u \{x \in C. 0 < u x\} \neq 0$
by (*smt (verit) fin(1) mem_Collect_eq sum_nonneg_eq_0_iff uv*)
qed
then have $*$: $\text{sum } u \{x \in C. u x > 0\} > 0$
unfolding *less_le* **by** (*metis (no_types, lifting) mem_Collect_eq sum_nonneg*)
moreover have $\text{sum } u (\{x \in C. 0 < u x\} \cup \{x \in C. u x < 0\}) = \text{sum } u C$
 $(\sum x \in \{x \in C. 0 < u x\} \cup \{x \in C. u x < 0\}. u x *_R x) = (\sum x \in C. u x *_R x)$
using *assms(1)*

```

  by (rule_tac[!] sum.mono_neutral_left, auto)
then have  $\text{sum } u \{x \in C. 0 < u x\} = - \text{sum } u \{x \in C. 0 > u x\}$ 
  ( $\sum_{x \in \{x \in C. 0 < u x\}} u x *_R x = - (\sum_{x \in \{x \in C. 0 > u x\}} u x *_R x)$ )
  unfolding eq_neg_iff_add_eq_0
  using uv(1,4)
  by (auto simp: sum.union_inter_neutral[OF fin, symmetric])
moreover have  $\forall x \in \{v \in C. u v < 0\}. 0 \leq \text{inverse } (\text{sum } u \{x \in C. 0 < u x\})$ 
* - u x
  using * by (fastforce intro: mult_nonneg_nonneg)
ultimately have  $z \in \text{convex hull } \{v \in C. u v \leq 0\}$ 
  unfolding convex_hull_explicit mem_Collect_eq
  apply (rule_tac x={v ∈ C. u v < 0} in exI)
  apply (rule_tac x=λy. inverse (sum u {x ∈ C. u x > 0}) * - u y in exI)
  using assms(1) unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric]

  by (auto simp: z_def sum_negf sum_distrib_left[symmetric])
moreover have  $\forall x \in \{v \in C. 0 < u v\}. 0 \leq \text{inverse } (\text{sum } u \{x \in C. 0 < u x\})$ 
* u x
  using * by (fastforce intro: mult_nonneg_nonneg)
then have  $z \in \text{convex hull } \{v \in C. u v > 0\}$ 
  unfolding convex_hull_explicit mem_Collect_eq
  apply (rule_tac x={v ∈ C. 0 < u v} in exI)
  apply (rule_tac x=λy. inverse (sum u {x ∈ C. u x > 0}) * u y in exI)
  using assms(1)
  unfolding scaleR_scaleR[symmetric] scaleR_right.sum [symmetric]
  using * by (auto simp: z_def sum_negf sum_distrib_left[symmetric])
ultimately show ?thesis
  apply (rule_tac x={v ∈ C. u v ≤ 0} in exI)
  apply (rule_tac x={v ∈ C. u v > 0} in exI, auto)
done
qed

```

theorem Radon:

```

assumes affine_dependent c
obtains  $M P$  where  $M \subseteq c P \subseteq c M \cap P = \{\}$   $(\text{convex hull } M) \cap (\text{convex hull } P) \neq \{\}$ 
  by (smt (verit) Radon_partition affine_dependent_explicit affine_dependent_explicit_finite
  assms le_sup_iff)

```

1.9.2 Helly's theorem

lemma Helly_induct:

```

fixes  $\mathcal{F} :: 'a::\text{euclidean\_space}$   $\text{set set}$ 
assumes  $\text{card } \mathcal{F} = n$ 
  and  $n \geq \text{DIM}('a) + 1$ 
  and  $\forall S \in \mathcal{F}. \text{convex } S \forall T \subseteq \mathcal{F}. \text{card } T = \text{DIM}('a) + 1 \longrightarrow \bigcap T \neq \{\}$ 
shows  $\bigcap \mathcal{F} \neq \{\}$ 
using assms
proof (induction n arbitrary:  $\mathcal{F}$ )

```

```

    case 0
    then show ?case by auto
next
case (Suc n)
have finite  $\mathcal{F}$ 
  using  $\langle \text{card } \mathcal{F} = \text{Suc } n \rangle$  by (auto intro: card_ge_0_finite)
show  $\bigcap \mathcal{F} \neq \{\}$ 
proof (cases  $n = \text{DIM}('a)$ )
  case True
  then show ?thesis
    by (simp add: Suc.prem)
next
case False
have  $\bigcap (\mathcal{F} - \{S\}) \neq \{\}$  if  $S \in \mathcal{F}$  for  $S$ 
proof (rule Suc.IH[rule_format])
  show  $\text{card } (\mathcal{F} - \{S\}) = n$ 
    by (simp add: Suc.prem(1)  $\langle \text{finite } \mathcal{F} \rangle$  that)
  show  $\text{DIM}('a) + 1 \leq n$ 
    using False Suc.prem(2) by linarith
  show  $\bigwedge t. \llbracket t \subseteq \mathcal{F} - \{S\}; \text{card } t = \text{DIM}('a) + 1 \rrbracket \implies \bigcap t \neq \{\}$ 
    by (simp add: Suc.prem(4) subset_Diff_insert)
qed (use Suc in auto)
then have  $\forall S \in \mathcal{F}. \exists x. x \in \bigcap (\mathcal{F} - \{S\})$ 
  by blast
then obtain  $X$  where  $X: \bigwedge S. S \in \mathcal{F} \implies X S \in \bigcap (\mathcal{F} - \{S\})$ 
  by metis
show ?thesis
proof (cases inj_on  $X \mathcal{F}$ )
  case False
  then obtain  $S T$  where  $S \neq T$  and  $st: S \in \mathcal{F} T \in \mathcal{F} X S = X T$ 
    unfolding inj_on_def by auto
  then have  $*$ :  $\bigcap \mathcal{F} = \bigcap (\mathcal{F} - \{S\}) \cap \bigcap (\mathcal{F} - \{T\})$  by auto
  show ?thesis
    by (metis *  $X$  disjoint_iff_not_equal st)
next
case True
  then obtain  $M P$  where  $mp: M \cap P = \{\} M \cup P = X \text{ ' } \mathcal{F}$  convex hull  $M$ 
     $\cap$  convex hull  $P \neq \{\}$ 
    using Radon_partition[of  $X \text{ ' } \mathcal{F}$ ] and affine_dependent_biggerset[of  $X \text{ ' } \mathcal{F}$ ]
    unfolding card_image[OF True] and  $\langle \text{card } \mathcal{F} = \text{Suc } n \rangle$ 
    using Suc(3)  $\langle \text{finite } \mathcal{F} \rangle$  and False
    by auto
  have  $M \subseteq X \text{ ' } \mathcal{F} P \subseteq X \text{ ' } \mathcal{F}$ 
    using mp(2) by auto
  then obtain  $\mathcal{G} \mathcal{H}$  where  $gh: M = X \text{ ' } \mathcal{G} P = X \text{ ' } \mathcal{H} \mathcal{G} \subseteq \mathcal{F} \mathcal{H} \subseteq \mathcal{F}$ 
    unfolding subset_image_iff by auto
  then have  $\mathcal{F} \cup (\mathcal{G} \cup \mathcal{H}) = \mathcal{F}$  by auto
  then have  $\mathcal{F}: \mathcal{F} = \mathcal{G} \cup \mathcal{H}$ 
    using inj_on_Un_image_eq_iff[of  $X \mathcal{F} \mathcal{G} \cup \mathcal{H}$ ] and True

```

```

    unfolding mp(2)[unfolded image_Un[symmetric] gh]
  by auto
  have *:  $\mathcal{G} \cap \mathcal{H} = \{\}$ 
    using gh local.mp(1) by blast
  have convex hull  $(X \text{ ' } \mathcal{H}) \subseteq \bigcap \mathcal{G}$  convex hull  $(X \text{ ' } \mathcal{G}) \subseteq \bigcap \mathcal{H}$ 
    by (rule hull_minimal; use X *  $\mathcal{F}$  in ‹auto simp: Suc.prem(3) convex_Inter›)+
  then show ?thesis
    unfolding  $\mathcal{F}$  using mp(3)[unfolded gh] by blast
qed
qed
qed

```

theorem *Helly*:

```

  fixes  $\mathcal{F} :: 'a::euclidean\_space \text{ set set}$ 
  assumes  $\text{card } \mathcal{F} \geq \text{DIM}('a) + 1 \ \forall s \in \mathcal{F}. \text{convex } s$ 
    and  $\bigwedge t. [\![t \subseteq \mathcal{F}; \text{card } t = \text{DIM}('a) + 1]\!] \implies \bigcap t \neq \{\}$ 
  shows  $\bigcap \mathcal{F} \neq \{\}$ 
  using Helly_induct assms by blast

```

1.9.3 Epigraphs of convex functions

definition *epigraph* S ($f :: _ \Rightarrow \text{real}$) = $\{xy. \text{fst } xy \in S \wedge f(\text{fst } xy) \leq \text{snd } xy\}$

lemma *mem_epigraph*: $(x, y) \in \text{epigraph } S f \iff x \in S \wedge f x \leq y$

unfolding *epigraph_def* by auto

lemma *convex_epigraph*: $\text{convex}(\text{epigraph } S f) \iff \text{convex_on } S f$

proof *safe*

assume $L: \text{convex}(\text{epigraph } S f)$

then show $\text{convex_on } S f$

by (*fastforce simp: convex_def convex_on_def epigraph_def*)

next

assume $\text{convex_on } S f$

then show $\text{convex}(\text{epigraph } S f)$

unfolding *convex_def convex_on_def epigraph_def*

apply *safe*

apply (*rule_tac [2] $y = u * f a + v * f aa$ in order_trans*)

apply (*auto intro!: mult_left_mono add_mono*)

done

qed

lemma *convex_epigraphI*: $\text{convex_on } S f \implies \text{convex}(\text{epigraph } S f)$

unfolding *convex_epigraph* by auto

lemma *convex_epigraph_convex*: $\text{convex_on } S f \iff \text{convex}(\text{epigraph } S f)$

by (*simp add: convex_epigraph*)

Use this to derive general bound property of convex function

```

lemma convex_on:
  assumes convex S
  shows convex_on S f  $\longleftrightarrow$ 
     $(\forall k\ u\ x. (\forall i \in \{1..k::nat\}. 0 \leq u\ i \wedge x\ i \in S) \wedge \text{sum } u\ \{1..k\} = 1 \longrightarrow$ 
       $f (\text{sum } (\lambda i. u\ i *_{R}\ x\ i)\ \{1..k\}) \leq \text{sum } (\lambda i. u\ i * f(x\ i))\ \{1..k\})$ 
    (is ?lhs =  $(\forall k\ u\ x. ?rhs\ k\ u\ x)$ )
proof
  assume ?lhs
  then have §: convex  $\{xy. \text{fst } xy \in S \wedge f(\text{fst } xy) \leq \text{snd } xy\}$ 
    by (metis assms convex_epigraph epigraph_def)
  show  $\forall k\ u\ x. ?rhs\ k\ u\ x$ 
  proof (intro allI)
    fix k u x
    show ?rhs k u x
      using §
      unfolding convex mem_Collect_eq fst_sum snd_sum
      apply safe
      apply (drule_tac x=k in spec)
      apply (drule_tac x=u in spec)
      apply (drule_tac x= $\lambda i. (x\ i, f(x\ i))$  in spec)
      apply simp
    done
  qed
next
  assume  $\forall k\ u\ x. ?rhs\ k\ u\ x$ 
  then show ?lhs
  unfolding convex_epigraph_convex convex epigraph_def Ball_def mem_Collect_eq
fst_sum snd_sum
  using assms[unfolded convex] apply clarsimp
  apply (rule_tac y= $\sum i = 1..k. u\ i * f(\text{fst } (x\ i))$  in order_trans)
  by (auto simp add: mult_left_mono intro: sum_mono)
qed

```

1.9.4 A bound within a convex hull

```

lemma convex_on_convex_hull_bound:
  assumes convex_on (convex hull S) f
  and  $\forall x \in S. f\ x \leq b$ 
  shows  $\forall x \in \text{convex hull } S. f\ x \leq b$ 
proof
  fix x
  assume  $x \in \text{convex hull } S$ 
  then obtain k u v where
     $u: \forall i \in \{1..k::nat\}. 0 \leq u\ i \wedge v\ i \in S \text{ sum } u\ \{1..k\} = 1 (\sum i = 1..k. u\ i *_{R}\ v$ 
     $i) = x$ 
  unfolding convex_hull_indexed mem_Collect_eq by auto
  have  $(\sum i = 1..k. u\ i * f(v\ i)) \leq b$ 
  using sum_mono[of  $\{1..k\}$   $\lambda i. u\ i * f(v\ i)$   $\lambda i. u\ i * b$ ]

```

```

  unfolding sum_distrib_right[symmetric] u(2) mult_1
  using assms(2) mult_left_mono u(1) by blast
  then show  $f x \leq b$ 
  using assms(1)[unfolded convex_on[OF convex_convex_hull], rule_format, of
k u v]
  using hull_inc u by fastforce
qed

```

lemma *convex_set_plus*:

```

  assumes convex S and convex T shows convex (S + T)
  by (metis assms convex_hull_eq convex_hull_set_plus)

```

lemma *convex_set_sum*:

```

  assumes  $\bigwedge i. i \in A \implies \text{convex } (B i)$ 
  shows convex  $(\sum_{i \in A} B i)$ 
  using assms
  by (induction A rule: infinite_finite_induct) (auto simp: convex_set_plus)

```

lemma *finite_set_sum*:

```

  assumes  $\forall i \in A. \text{finite } (B i)$  shows finite  $(\sum_{i \in A} B i)$ 
  using assms
  by (induction A rule: infinite_finite_induct) (auto simp: finite_set_plus)

```

lemma *box_eq_set_sum_Basis*:

```

 $\{x. \forall i \in \text{Basis}. x \cdot i \in B i\} = (\sum_{i \in \text{Basis}} (\lambda x. x *_R i) ` (B i))$  (is ?lhs = ?rhs)

```

proof –

```

  have  $\bigwedge x. \forall i \in \text{Basis}. x \cdot i \in B i \implies$ 
 $\exists s. x = \text{sum } s \text{ Basis} \wedge (\forall i \in \text{Basis}. s i \in (\lambda x. x *_R i) ` B i)$ 
  by (metis (mono_tags, lifting) euclidean_representation_image_iff)
  moreover

```

have $\text{sum } f \text{ Basis} \cdot i \in B i$ **if** $i \in \text{Basis}$ **and** $f: \forall i \in \text{Basis}. f i \in (\lambda x. x *_R i) ` B$
i for i f

proof –

```

  have  $(\sum_{x \in \text{Basis} - \{i\}} f x \cdot i) = 0$ 
  proof (intro strip sum.neutral)
  show  $f x \cdot i = 0$  if  $x \in \text{Basis} - \{i\}$  for  $x$ 
  using that  $f \langle i \in \text{Basis} \rangle$  inner_Basis that by fastforce

```

qed

```

  then have  $(\sum_{x \in \text{Basis}} f x \cdot i) = f i \cdot i$ 

```

by (metis (no_types) $\langle i \in \text{Basis} \rangle$ add.right_neutral sum.remove [OF finite_Basis])

```

  then have  $(\sum_{x \in \text{Basis}} f x \cdot i) \in B i$ 

```

```

  using f that(1) by auto

```

```

  then show ?thesis

```

```

  by (simp add: inner_sum_left)

```

qed

ultimately show ?thesis

```

  by (subst set_sum_alt [OF finite_Basis]) auto

```

qed

```

lemma convex_hull_set_sum:
  convex_hull ( $\sum_{i \in A} B\ i$ ) = ( $\sum_{i \in A} \text{convex\_hull } (B\ i)$ )
  by (induction A rule: infinite_finite_induct) (auto simp: convex_hull_set_plus)

end

```

1.10 Definition of Finite Cartesian Product Type

```

theory Finite_Cartesian_Product
imports
  Euclidean_Space
  L2_Norm
  HOL-Library.Numeral_Type
  HOL-Library.Countable_Set
  HOL-Library.FuncSet
begin

```

1.10.1 Finite Cartesian products, with indexing and lambdas

```

typedef ('a, 'b) vec = UNIV :: ('b::finite  $\Rightarrow$  'a) set
  morphisms vec_nth vec_lambda ..

```

```

declare vec_lambda_inject [simplified, simp]

```

```

bundle vec_syntax begin
notation
  vec_nth (infixl $ 90) and
  vec_lambda (binder  $\chi$  10)
end

```

```

bundle no_vec_syntax begin
no_notation
  vec_nth (infixl $ 90) and
  vec_lambda (binder  $\chi$  10)
end

```

```

unbundle vec_syntax

```

Concrete syntax for ('a, 'b) *vec*:

- $'a \sim 'b$ becomes ('a, 'b::finite) *vec*
- $'a \sim 'b :: _$ becomes ('a, 'b) *vec* without extra sort-constraint

```

syntax _vec_type :: type  $\Rightarrow$  type  $\Rightarrow$  type (infixl  $\wedge$  15)

```

```

parse_translation <

```

```

  let

```

```

    fun vec t u = Syntax.const type_syntax <vec> $ t $ u;

```



```

fun finite_vec_tr [t, u] =
  (case Term_Position.strip_positions u of
   v as Free (x, _) =>
     if Lexicon.is_tid x then
       vec t (Syntax.const syntax_const <_ofsort> $ v $
        Syntax.const class_syntax <finite>)
     else vec t u
   | _ => vec t u)
in
  [(syntax_const <_vec_type>, K finite_vec_tr)]
end
>

```

lemma *vec_eq_iff*: $(x = y) \longleftrightarrow (\forall i. x\$i = y\$i)$
by (*simp add: vec_nth_inject [symmetric] fun_eq_iff*)

lemma *vec_lambda_beta* [*simp*]: $\text{vec_lambda } g \$ i = g i$
by (*simp add: vec_lambda_inverse*)

lemma *vec_lambda_unique*: $(\forall i. f\$i = g i) \longleftrightarrow \text{vec_lambda } g = f$
by (*auto simp add: vec_eq_iff*)

lemma *vec_lambda_eta* [*simp*]: $(\chi i. (g\$i)) = g$
by (*simp add: vec_eq_iff*)

1.10.2 Cardinality of vectors

instance *vec* :: (finite, finite) finite

proof

show *finite* (UNIV :: ('a, 'b) vec set)

proof (*subst bij_betw_finite*)

show *bij_betw* *vec_nth* UNIV (*Pi* (UNIV :: 'b set) ($\lambda_.$ UNIV :: 'a set))

by (*intro bij_betwI*[of $____ \text{vec_lambda}$]) (*auto simp: vec_eq_iff*)

have *finite* (*PiE* (UNIV :: 'b set) ($\lambda_.$ UNIV :: 'a set))

by (*intro finite_PiE*) *auto*

also have (*PiE* (UNIV :: 'b set) ($\lambda_.$ UNIV :: 'a set)) = *Pi* UNIV ($\lambda_.$ UNIV)

by *auto*

finally show *finite*

qed

qed

lemma *countable_PiE*:

$\text{finite } I \implies (\bigwedge i. i \in I \implies \text{countable } (F i)) \implies \text{countable } (\text{Pi}_E I F)$

by (*induct I arbitrary: F rule: finite_induct*) (*auto simp: PiE_insert_eq*)

instance *vec* :: (countable, finite) countable

proof

have *countable* (UNIV :: ('a, 'b) vec set)

proof (*rule countableI_bij2*)

```

show bij_betw_vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
  by (intro bij_betwI[of ___ vec_lambda]) (auto simp: vec_eq_iff)
have countable (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a set))
  by (intro countable_PiE) auto
also have (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a set)) = Pi UNIV (λ_. UNIV)
  by auto
finally show countable ... .
qed
thus ∃ t::('a, 'b) vec ⇒ nat.inj t
  by (auto elim!: countableE)
qed

```

```

lemma infinite_UNIV_vec:
  assumes infinite (UNIV :: 'a set)
  shows infinite (UNIV :: ('a^b) set)
proof (subst bij_betw_finite)
  show bij_betw_vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
    by (intro bij_betwI[of ___ vec_lambda]) (auto simp: vec_eq_iff)
  have infinite (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a set)) (is infinite ?A)
  proof
    assume finite ?A
    hence finite ((λf. f undefined) ' ?A)
      by (rule finite_imageI)
    also have (λf. f undefined) ' ?A = UNIV
      by auto
    finally show False
      using <infinite (UNIV :: 'a set)> by contradiction
  qed
also have ?A = Pi UNIV (λ_. UNIV)
  by auto
finally show infinite (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set)) .
qed

```

```

proposition CARD_vec [simp]:
  CARD('a^b) = CARD('a) ^ CARD('b)
proof (cases finite (UNIV :: 'a set))
  case True
  show ?thesis
  proof (subst bij_betw_same_card)
    show bij_betw_vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
      by (intro bij_betwI[of ___ vec_lambda]) (auto simp: vec_eq_iff)
    have CARD('a) ^ CARD('b) = card (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a
  set))
      (is _ = card ?A)
      by (subst card_PiE) (auto)
    also have ?A = Pi UNIV (λ_. UNIV)
      by auto
    finally show card ... = CARD('a) ^ CARD('b) ..
  qed

```

```

qed (simp_all add: infinite_UNIV_vec)

lemma countable_vector:
  fixes B:: 'n::finite  $\Rightarrow$  'a set
  assumes  $\bigwedge i. \text{countable } (B\ i)$ 
  shows countable  $\{V. \forall i::'n. V\ \$\ i \in B\ i\}$ 
proof -
  have  $f \in (\$) \text{ ' } \{V. \forall i. V\ \$\ i \in B\ i\}$  if  $f \in \text{Pi}_E\ \text{UNIV}\ B$  for  $f$ 
  proof -
    have  $\exists W. (\forall i. W\ \$\ i \in B\ i) \wedge (\$) W = f$ 
      by (metis that  $\text{Pi}_E\_iff\ \text{UNIV}_I\ \text{vec\_lambda\_inverse}$ )
    then show  $f \in (\$) \text{ ' } \{v. \forall i. v\ \$\ i \in B\ i\}$ 
      by blast
  qed
  then have  $\text{Pi}_E\ \text{UNIV}\ B = \text{vec\_nth} \text{ ' } \{V. \forall i::'n. V\ \$\ i \in B\ i\}$ 
    by blast
  then have countable  $(\text{vec\_nth} \text{ ' } \{V. \forall i. V\ \$\ i \in B\ i\})$ 
    by (metis finite_class.finite_UNIV countable_PiE assms)
  then have countable  $(\text{vec\_lambda} \text{ ' } \text{vec\_nth} \text{ ' } \{V. \forall i. V\ \$\ i \in B\ i\})$ 
    by auto
  then show ?thesis
    by (simp add: image_comp o_def vec_nth_inverse)
qed

```

1.10.3 Group operations and class instances

```

instantiation vec :: (zero, finite) zero
begin
  definition 0  $\equiv$  ( $\chi\ i. 0$ )
  instance ..
end

instantiation vec :: (plus, finite) plus
begin
  definition (+)  $\equiv$  ( $\lambda\ x\ y. (\chi\ i. x\ \$\ i + y\ \$\ i)$ )
  instance ..
end

instantiation vec :: (minus, finite) minus
begin
  definition (-)  $\equiv$  ( $\lambda\ x\ y. (\chi\ i. x\ \$\ i - y\ \$\ i)$ )
  instance ..
end

instantiation vec :: (uminus, finite) uminus
begin
  definition uminus  $\equiv$  ( $\lambda\ x. (\chi\ i. - (x\ \$\ i))$ )
  instance ..
end

```

```

lemma zero_index [simp]: 0 $ i = 0
  unfolding zero_vec_def by simp

lemma vector_add_component [simp]: (x + y)$i = x$i + y$i
  unfolding plus_vec_def by simp

lemma vector_minus_component [simp]: (x - y)$i = x$i - y$i
  unfolding minus_vec_def by simp

lemma vector_uminus_component [simp]: (- x)$i = - (x$i)
  unfolding uminus_vec_def by simp

instance vec :: (semigroup_add, finite) semigroup_add
  by standard (simp add: vec_eq_iff add.assoc)

instance vec :: (ab_semigroup_add, finite) ab_semigroup_add
  by standard (simp add: vec_eq_iff add.commute)

instance vec :: (monoid_add, finite) monoid_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (comm_monoid_add, finite) comm_monoid_add
  by standard (simp add: vec_eq_iff)

instance vec :: (cancel_semigroup_add, finite) cancel_semigroup_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (cancel_ab_semigroup_add, finite) cancel_ab_semigroup_add
  by standard (simp_all add: vec_eq_iff diff_diff_eq)

instance vec :: (cancel_comm_monoid_add, finite) cancel_comm_monoid_add
  ..

instance vec :: (group_add, finite) group_add
  by standard (simp_all add: vec_eq_iff)

instance vec :: (ab_group_add, finite) ab_group_add
  by standard (simp_all add: vec_eq_iff)

1.10.4 Basic componentwise operations on vectors

instantiation vec :: (times, finite) times
begin

definition (*) ≡ (λ x y. (χ i. (x$i) * (y$i)))
instance ..

end

```

instantiation *vec* :: (one, finite) one
begin

definition $1 \equiv (\chi i. 1)$
instance ..

end

instantiation *vec* :: (ord, finite) ord
begin

definition $x \leq y \longleftrightarrow (\forall i. x\$i \leq y\$i)$
definition $x < (y::'a^'b) \longleftrightarrow x \leq y \wedge \neg y \leq x$
instance ..

end

The ordering on one-dimensional vectors is linear.

instance *vec*:: (order, finite) order
by *standard* (auto simp: less_eq_vec_def less_vec_def vec_eq_iff
intro: order.trans order.antisym order.strict_implies_order)

instance *vec* :: (linorder, CARD_1) linorder
proof

obtain $a :: 'b$ **where** $\bigwedge P. (\forall i. P i) \longleftrightarrow P a$

proof –

have $CARD\ 'b = 1$ **by** (rule CARD_1)

then obtain $b :: 'b$ **where** $UNIV = \{b\}$ **by** (auto iff: card_Suc_eq)

then have $\bigwedge P. (\forall i \in UNIV. P i) \longleftrightarrow P b$ **by** auto

then show *thesis* **by** (auto intro: that)

qed

fix $x\ y :: 'a^'b::CARD_1$

note [simp] = less_eq_vec_def less_vec_def all_vec_eq_iff field_simps

show $x \leq y \vee y \leq x$ **by** auto

qed

Constant Vectors

definition *vec* $x = (\chi i. x)$

Also the scalar-vector multiplication.

definition *vector_scalar_mult*:: $'a::times \Rightarrow 'a \wedge 'n \Rightarrow 'a \wedge 'n$ (infixl *s 70)
where $c *s x = (\chi i. c * (x\$i))$

scalar product

definition *scalar_product* :: $'a :: semiring_1 \wedge 'n \Rightarrow 'a \wedge 'n \Rightarrow 'a$ **where**
scalar_product $v\ w = (\sum i \in UNIV. v\ \$\ i * w\ \$\ i)$

1.10.5 Real vector space

instantiation *vec* :: (*real_vector*, *finite*) *real_vector*
begin

definition *scaleR* $\equiv (\lambda r x. (\chi i. \text{scaleR } r (x\$i)))$

lemma *vector_scaleR_component* [*simp*]: (*scaleR* *r* *x*)\$*i* = *scaleR* *r* (*x*\$*i*)
unfolding *scaleR_vec_def* **by** *simp*

instance

by *standard* (*simp_all* *add*: *vec_eq_iff* *scaleR_left_distrib* *scaleR_right_distrib*)

end

1.10.6 Topological space

instantiation *vec* :: (*topological_space*, *finite*) *topological_space*
begin

definition [*code del*]:

open (*S* :: ('*a* ^ '*b*) *set*) \longleftrightarrow
 $(\forall x \in S. \exists A. (\forall i. \text{open } (A \ i) \wedge x\$i \in A \ i) \wedge$
 $(\forall y. (\forall i. y\$i \in A \ i) \longrightarrow y \in S))$

instance proof

show *open* (*UNIV* :: ('*a* ^ '*b*) *set*)
unfolding *open_vec_def* **by** *auto*

next

fix *S T* :: ('*a* ^ '*b*) *set*
assume *open S open T* **thus** *open* (*S* \cap *T*)
unfolding *open_vec_def*
apply *clarify*
apply (*drule* (1) *bspec*)
apply (*clarify*, *rename_tac* *Sa Ta*)
apply (*rule_tac* *x*= $\lambda i. Sa \ i \cap Ta \ i$ **in** *exI*)
apply (*simp* *add*: *open_Int*)
done

next

fix *K* :: ('*a* ^ '*b*) *set set*
assume $\forall S \in K. \text{open } S$ **thus** *open* ($\bigcup K$)
unfolding *open_vec_def*
by (*metis* *Union_iff*)

qed

end

lemma *open_vector_box*: $\forall i. \text{open } (S \ i) \implies \text{open } \{x. \forall i. x \ \$ \ i \in S \ i\}$
unfolding *open_vec_def* **by** *auto*

lemma *open_vimage_vec_nth*: $open\ S \implies open\ ((\lambda x. x\ \$\ i) -' S)$
unfolding *open_vec_def*
apply *clarify*
apply (*rule_tac* $x = \lambda k. \text{if } k = i \text{ then } S \text{ else } UNIV$ **in** *exI, simp*)
done

lemma *closed_vimage_vec_nth*: $closed\ S \implies closed\ ((\lambda x. x\ \$\ i) -' S)$
unfolding *closed_open_vimage_Cmpl* [*symmetric*]
by (*rule open_vimage_vec_nth*)

lemma *closed_vector_box*: $\forall i. closed\ (S\ i) \implies closed\ \{x. \forall i. x\ \$\ i \in S\ i\}$
proof –
have $\{x. \forall i. x\ \$\ i \in S\ i\} = (\bigcap i. (\lambda x. x\ \$\ i) -' S\ i)$ **by** *auto*
thus $\forall i. closed\ (S\ i) \implies closed\ \{x. \forall i. x\ \$\ i \in S\ i\}$
by (*simp add: closed_INT closed_vimage_vec_nth*)
qed

lemma *tendsto_vec_nth* [*tendsto_intros*]:
assumes $((\lambda x. f\ x) \longrightarrow a)$ *net*
shows $((\lambda x. f\ x\ \$\ i) \longrightarrow a\ \$\ i)$ *net*
proof (*rule topological_tendstoI*)
fix S **assume** $open\ S\ a\ \$\ i \in S$
then have $open\ ((\lambda y. y\ \$\ i) -' S)$ $a \in ((\lambda y. y\ \$\ i) -' S)$
by (*simp_all add: open_vimage_vec_nth*)
with *assms* **have** *eventually* $(\lambda x. f\ x \in (\lambda y. y\ \$\ i) -' S)$ *net*
by (*rule topological_tendstoD*)
then show *eventually* $(\lambda x. f\ x\ \$\ i \in S)$ *net*
by *simp*
qed

lemma *isCont_vec_nth* [*simp*]: $isCont\ f\ a \implies isCont\ (\lambda x. f\ x\ \$\ i)\ a$
unfolding *isCont_def* **by** (*rule tendsto_vec_nth*)

lemma *vec_tendstoI*:
assumes $\bigwedge i. ((\lambda x. f\ x\ \$\ i) \longrightarrow a\ \$\ i)$ *net*
shows $((\lambda x. f\ x) \longrightarrow a)$ *net*
proof (*rule topological_tendstoI*)
fix S **assume** $open\ S$ **and** $a \in S$
then obtain A **where** $A: \bigwedge i. open\ (A\ i) \bigwedge i. a\ \$\ i \in A\ i$
and $S: \bigwedge y. \forall i. y\ \$\ i \in A\ i \implies y \in S$
unfolding *open_vec_def* **by** *metis*
have $\bigwedge i. \text{eventually } (\lambda x. f\ x\ \$\ i \in A\ i)$ *net*
using *assms* A **by** (*rule topological_tendstoD*)
hence *eventually* $(\lambda x. \forall i. f\ x\ \$\ i \in A\ i)$ *net*
by (*rule eventually_all_finite*)
thus *eventually* $(\lambda x. f\ x \in S)$ *net*
by (*rule eventually_mono, simp add: S*)
qed

lemma *tendsto_vec_lambda* [*tendsto_intros*]:
assumes $\bigwedge i. ((\lambda x. f x i) \longrightarrow a i)$ *net*
shows $((\lambda x. \chi i. f x i) \longrightarrow (\chi i. a i))$ *net*
using *assms* **by** (*simp add: vec_tendstoI*)

lemma *open_image_vec_nth*: **assumes** *open S* **shows** *open (($\lambda x. x \$ i$) ' S)*
proof (*rule openI*)
fix *a* **assume** $a \in (\lambda x. x \$ i) ' S$
then obtain *z* **where** $a = z \$ i$ **and** $z \in S$..
then obtain *A* **where** $A: \forall i. \text{open } (A i) \wedge z \$ i \in A i$
and $S: \forall y. (\forall i. y \$ i \in A i) \longrightarrow y \in S$
using $\langle \text{open } S \rangle$ **unfolding** *open_vec_def* **by** *auto*
hence $A i \subseteq (\lambda x. x \$ i) ' S$
by (*clarsimp, rule_tac x= χj . if $j = i$ then x else $z \$ j$ in image_eqI,*
simp_all)
hence $\text{open } (A i) \wedge a \in A i \wedge A i \subseteq (\lambda x. x \$ i) ' S$
using $A \langle a = z \$ i \rangle$ **by** *simp*
then show $\exists T. \text{open } T \wedge a \in T \wedge T \subseteq (\lambda x. x \$ i) ' S$ **by** $-$ (*rule exI*)
qed

instance *vec* :: (*perfect_space, finite*) *perfect_space*

proof
fix $x :: 'a \wedge 'b$ **show** $\neg \text{open } \{x\}$
proof
assume *open {x}*
hence $\forall i. \text{open } ((\lambda x. x \$ i) ' \{x\})$ **by** (*fast intro: open_image_vec_nth*)
hence $\forall i. \text{open } \{x \$ i\}$ **by** *simp*
thus *False* **by** (*simp add: not_open_singleton*)
qed
qed

1.10.7 Metric space

instantiation *vec* :: (*metric_space, finite*) *dist*
begin

definition

dist $x y = L2_set (\lambda i. \text{dist } (x \$ i) (y \$ i)) \text{ UNIV}$

instance ..
end

instantiation *vec* :: (*metric_space, finite*) *uniformity_dist*
begin

definition [*code del*]:

(*uniformity* :: (($'a \wedge 'b :: _$) \times ($'a \wedge 'b :: _$)) *filter*) =
 $(\text{INF } e \in \{0 <.. \}. \text{principal } \{(x, y). \text{dist } x y < e\})$


```

instance
  by standard (rule uniformity_vec_def)
end

declare uniformity_Abort[where 'a='a :: metric_space ^ 'b :: finite, code]

instantiation vec :: (metric_space, finite) metric_space
begin

proposition dist_vec_nth_le:  $\text{dist } (x \$ i) (y \$ i) \leq \text{dist } x y$ 
  unfolding dist_vec_def by (rule member_le_L2_set) simp_all

instance proof
  fix  $x y :: 'a ^ 'b$ 
  show  $\text{dist } x y = 0 \iff x = y$ 
    unfolding dist_vec_def
    by (simp add: L2_set_eq_0_iff vec_eq_iff)
next
  fix  $x y z :: 'a ^ 'b$ 
  show  $\text{dist } x y \leq \text{dist } x z + \text{dist } y z$ 
    unfolding dist_vec_def
    apply (rule order_trans [OF _ L2_set_triangle_ineq])
    apply (simp add: L2_set_mono dist_triangle2)
    done
next
  fix  $S :: ('a ^ 'b) \text{ set}$ 
  have *:  $\text{open } S \iff (\forall x \in S. \exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in S)$ 
  proof
    assume  $\text{open } S$  show  $\forall x \in S. \exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in S$ 
    proof
      fix  $x$  assume  $x \in S$ 
      obtain  $A$  where  $A: \forall i. \text{open } (A i) \wedge \forall i. x \$ i \in A i$ 
        and  $S: \forall y. (\forall i. y \$ i \in A i) \longrightarrow y \in S$ 
        using  $\langle \text{open } S \rangle$  and  $\langle x \in S \rangle$  unfolding open_vec_def by metis
      have  $\forall i \in \text{UNIV}. \exists r > 0. \forall y. \text{dist } y (x \$ i) < r \longrightarrow y \in A i$ 
        using  $A$  unfolding open_dist by simp
      hence  $\exists r. \forall i \in \text{UNIV}. 0 < r i \wedge (\forall y. \text{dist } y (x \$ i) < r i \longrightarrow y \in A i)$ 
        by (rule finite_set_choice [OF finite])
      then obtain  $r$  where  $r1: \forall i. 0 < r i$ 
        and  $r2: \forall i y. \text{dist } y (x \$ i) < r i \longrightarrow y \in A i$  by fast
      have  $0 < \text{Min } (\text{range } r) \wedge (\forall y. \text{dist } y x < \text{Min } (\text{range } r) \longrightarrow y \in S)$ 
        by (simp add: r1 r2 S le_less_trans [OF dist_vec_nth_le])
      thus  $\exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in S$  ..
    qed
next
assume *:  $\forall x \in S. \exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in S$  show  $\text{open } S$ 
proof (unfold open_vec_def, rule)
  fix  $x$  assume  $x \in S$ 
  then obtain  $e$  where  $0 < e$  and  $S: \forall y. \text{dist } y x < e \longrightarrow y \in S$ 

```

```

    using * by fast
    define r where [abs_def]: r i = e / sqrt (of_nat CARD('b)) for i :: 'b
    from ⟨0 < e⟩ have r: ∀ i. 0 < r i
    unfolding r_def by simp_all
    from ⟨0 < e⟩ have e: e = L2_set r UNIV
    unfolding r_def by (simp add: L2_set_constant)
    define A where A i = {y. dist (x $ i) y < r i} for i
    have ∀ i. open (A i) ∧ x $ i ∈ A i
    unfolding A_def by (simp add: open_ball r)
    moreover have ∀ y. (∀ i. y $ i ∈ A i) → y ∈ S
    by (simp add: A_def S dist_vec_def e L2_set_strict_mono dist_commute)
    ultimately show ∃ A. (∀ i. open (A i) ∧ x $ i ∈ A i) ∧
      (∀ y. (∀ i. y $ i ∈ A i) → y ∈ S) by metis
  qed
  qed
  show open S = (∀ x ∈ S. ∀F (x', y) in uniformity. x' = x → y ∈ S)
    unfolding * eventually_uniformity_metric
    by (simp del: split_paired_All add: dist_vec_def dist_commute)
  qed
end

```

lemma *Cauchy_vec_nth*:

```

  Cauchy (λ n. X n) ⇒ Cauchy (λ n. X n $ i)
  unfolding Cauchy_def by (fast intro: le_less_trans [OF dist_vec_nth_le])

```

lemma *vec_CauchyI*:

```

  fixes X :: nat ⇒ 'a::metric_space ^ 'n
  assumes X: ∧ i. Cauchy (λ n. X n $ i)
  shows Cauchy (λ n. X n)
  proof (rule metric_CauchyI)
    fix r :: real assume 0 < r
    hence 0 < r / of_nat CARD('n) (is 0 < ?s) by simp
    define N where N i = (LEAST N. ∀ m ≥ N. ∀ n ≥ N. dist (X m $ i) (X n $ i)
      < ?s) for i
    define M where M = Max (range N)
    have ∧ i. ∃ N. ∀ m ≥ N. ∀ n ≥ N. dist (X m $ i) (X n $ i) < ?s
      using X ⟨0 < ?s⟩ by (rule metric_CauchyD)
    hence ∧ i. ∀ m ≥ N i. ∀ n ≥ N i. dist (X m $ i) (X n $ i) < ?s
      unfolding N_def by (rule LeastI_ex)
    hence M: ∧ i. ∀ m ≥ M. ∀ n ≥ M. dist (X m $ i) (X n $ i) < ?s
      unfolding M_def by simp
    {
      fix m n :: nat
      assume M ≤ m M ≤ n
      have dist (X m) (X n) = L2_set (λ i. dist (X m $ i) (X n $ i)) UNIV
        unfolding dist_vec_def ..
      also have ... ≤ sum (λ i. dist (X m $ i) (X n $ i)) UNIV
        by (rule L2_set_le_sum [OF zero_le_dist])
    }
  }

```

```

    also have ... < sum ( $\lambda i::'n. ?s$ ) UNIV
      by (rule sum_strict_mono, simp_all add:  $M \langle M \leq m \rangle \langle M \leq n \rangle$ )
    also have ... = r
      by simp
    finally have  $\text{dist } (X\ m) (X\ n) < r$  .
  }
  hence  $\forall m \geq M. \forall n \geq M. \text{dist } (X\ m) (X\ n) < r$ 
    by simp
  then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (X\ m) (X\ n) < r$  ..
qed

```

instance *vec* :: (complete_space, finite) complete_space

proof

```

  fix  $X :: \text{nat} \Rightarrow 'a \wedge 'b$  assume Cauchy  $X$ 
  have  $\bigwedge i. (\lambda n. X\ n\ \$\ i) \longrightarrow \text{lim } (\lambda n. X\ n\ \$\ i)$ 
    using Cauchy_vec_nth [OF  $\langle \text{Cauchy } X \rangle$ ]
  by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  hence  $X \longrightarrow \text{vec\_lambda } (\lambda i. \text{lim } (\lambda n. X\ n\ \$\ i))$ 
    by (simp add: vec_tendstoI)
  then show convergent  $X$ 
    by (rule convergentI)

```

qed

1.10.8 Normed vector space

instantiation *vec* :: (real_normed_vector, finite) real_normed_vector
begin

definition $\text{norm } x = L2_set (\lambda i. \text{norm } (x\$i))$ UNIV

definition $\text{sgn } (x::'a \wedge 'b) = \text{scaleR } (\text{inverse } (\text{norm } x))\ x$

instance proof

```

  fix  $a :: \text{real}$  and  $x\ y :: 'a \wedge 'b$ 
  show  $\text{norm } x = 0 \longleftrightarrow x = 0$ 
    unfolding norm_vec_def
    by (simp add: L2_set_eq_0_iff vec_eq_iff)
  show  $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$ 
    unfolding norm_vec_def
    apply (rule order_trans [OF _ L2_set_triangle_ineq])
    apply (simp add: L2_set_mono norm_triangle_ineq)
    done
  show  $\text{norm } (\text{scaleR } a\ x) = |a| * \text{norm } x$ 
    unfolding norm_vec_def
    by (simp add: L2_set_right_distrib)
  show  $\text{sgn } x = \text{scaleR } (\text{inverse } (\text{norm } x))\ x$ 
    by (rule sgn_vec_def)
  show  $\text{dist } x\ y = \text{norm } (x - y)$ 
    unfolding dist_vec_def norm_vec_def

```

by (*simp add: dist_norm*)
qed

end

lemma *norm_nth_le*: $\text{norm } (x \$ i) \leq \text{norm } x$
unfolding *norm_vec_def*
by (*rule member_le_L2_set*) *simp_all*

lemma *norm_le_componentwise_cart*:
fixes $x :: 'a::\text{real_normed_vector}^n$
assumes $\bigwedge i. \text{norm}(x\$i) \leq \text{norm}(y\$i)$
shows $\text{norm } x \leq \text{norm } y$
unfolding *norm_vec_def*
by (*rule L2_set_mono*) (*auto simp: assms*)

lemma *component_le_norm_cart*: $|x\$i| \leq \text{norm } x$
by (*metis norm_nth_le real_norm_def*)

lemma *norm_bound_component_le_cart*: $\text{norm } x \leq e \implies |x\$i| \leq e$
by (*metis component_le_norm_cart order_trans*)

lemma *norm_bound_component_lt_cart*: $\text{norm } x < e \implies |x\$i| < e$
by (*metis component_le_norm_cart le_less_trans*)

lemma *norm_le_l1_cart*: $\text{norm } x \leq \text{sum}(\lambda i. |x\$i|)$ *UNIV*
by (*simp add: norm_vec_def L2_set_le_sum*)

lemma *bounded_linear_vec_nth[intro]*: *bounded_linear* $(\lambda x. x \$ i)$
proof
show $\exists K. \forall x. \text{norm } (x \$ i) \leq \text{norm } x * K$
by (*metis mult.commute mult.left_neutral norm_nth_le*)
qed *auto*

instance *vec* :: (*banach, finite*) *banach* ..

1.10.9 Inner product space

instantiation *vec* :: (*real_inner, finite*) *real_inner*
begin

definition *inner* $x y = \text{sum } (\lambda i. \text{inner } (x\$i) (y\$i))$ *UNIV*

instance **proof**

fix $r :: \text{real}$ **and** $x y z :: 'a \wedge 'b$

show $\text{inner } x y = \text{inner } y x$

unfolding *inner_vec_def*

by (*simp add: inner_commute*)

show $\text{inner } (x + y) z = \text{inner } x z + \text{inner } y z$

```

    unfolding inner_vec_def
  by (simp add: inner_add_left sum.distrib)
show inner (scaleR r x) y = r * inner x y
  unfolding inner_vec_def
  by (simp add: sum_distrib_left)
show 0 ≤ inner x x
  unfolding inner_vec_def
  by (simp add: sum_nonneg)
show inner x x = 0 ↔ x = 0
  unfolding inner_vec_def
  by (simp add: vec_eq_iff sum_nonneg_eq_0_iff)
show norm x = sqrt (inner x x)
  unfolding inner_vec_def norm_vec_def L2_set_def
  by (simp add: power2_norm_eq_inner)
qed
end

```

1.10.10 Euclidean space

Vectors pointing along a single axis.

definition $axis\ k\ x = (\chi\ i.\ \text{if } i = k \text{ then } x \text{ else } 0)$

lemma $axis_nth$ [simp]: $axis\ i\ x\ \$\ i = x$
unfolding $axis_def$ **by** $simp$

lemma $axis_eq_axis$: $axis\ i\ x = axis\ j\ y \leftrightarrow x = y \wedge i = j \vee x = 0 \wedge y = 0$
unfolding $axis_def\ vec_eq_iff$ **by** $auto$

lemma $inner_axis_axis$:
 $inner\ (axis\ i\ x)\ (axis\ j\ y) = (\text{if } i = j \text{ then } inner\ x\ y \text{ else } 0)$
by (simp add: inner_vec_def axis_def sum.neutral sum.remove [of _ j])

lemma $inner_axis$: $inner\ x\ (axis\ i\ y) = inner\ (x\ \$\ i)\ y$
by (simp add: inner_vec_def axis_def sum.remove [where $x=i$])

lemma $inner_axis'$: $inner\ (axis\ i\ y)\ x = inner\ y\ (x\ \$\ i)$
by (simp add: inner_axis inner_commute)

instantiation $vec :: (euclidean_space, finite)\ euclidean_space$
begin

definition $Basis = (\bigcup i.\ \bigcup u \in Basis.\ \{axis\ i\ u\})$

instance proof
show $(Basis :: ('a \wedge 'b)\ set) \neq \{\}$
unfolding $Basis_vec_def$ **by** $simp$
next
show $finite\ (Basis :: ('a \wedge 'b)\ set)$

```

    unfolding Basis_vec_def by simp
next
  fix u v :: 'a ^ 'b
  assume u ∈ Basis and v ∈ Basis
  thus inner u v = (if u = v then 1 else 0)
    unfolding Basis_vec_def
    by (auto simp add: inner_axis_axis axis_eq_axis inner_Basis)
next
  fix x :: 'a ^ 'b
  show (∀ u ∈ Basis. inner x u = 0) ↔ x = 0
    unfolding Basis_vec_def
    by (simp add: inner_axis euclidean_all_zero_iff vec_eq_iff)
qed

```

proposition *DIM_cart* [simp]: $DIM('a \wedge 'b) = CARD('b) * DIM('a)$

proof –

```

  have card (⋃ i::'b. ⋃ u::'a ∈ Basis. {axis i u}) = (∑ i::'b ∈ UNIV. card (⋃ u::'a ∈ Basis.
    {axis i u}))

```

```

  by (rule card_UN_disjoint) (auto simp: axis_eq_axis)

```

```

  also have ... = CARD('b) * DIM('a)

```

```

  by (subst card_UN_disjoint) (auto simp: axis_eq_axis)

```

```

  finally show ?thesis

```

```

  by (simp add: Basis_vec_def)

```

qed

end

lemma *norm_axis_1* [simp]: $norm (axis m (1::real)) = 1$

```

  by (simp add: inner_axis' norm_eq_1)

```

lemma *sum_norm_allsubsets_bound_cart*:

```

  fixes f:: 'a ⇒ real ^ n

```

```

  assumes fP: finite P and fPs: ⋀ Q. Q ⊆ P ⇒ norm (sum f Q) ≤ e

```

```

  shows sum (λ x. norm (f x)) P ≤ 2 * real CARD('n) * e

```

```

  using sum_norm_allsubsets_bound[OF assms]

```

```

  by simp

```

lemma *cart_eq_inner_axis*: $a \ \$ \ i = inner \ a \ (axis \ i \ 1)$

```

  by (simp add: inner_axis)

```

lemma *axis_eq_0_iff* [simp]:

```

  shows axis m x = 0 ↔ x = 0

```

```

  by (simp add: axis_def vec_eq_iff)

```

lemma *axis_in_Basis_iff* [simp]: $axis \ i \ a \in Basis \longleftrightarrow a \in Basis$

```

  by (auto simp: Basis_vec_def axis_eq_axis)

```

Mapping each basis element to the corresponding finite index

definition *axis_index* :: $('a::comm_ring_1) \wedge n \Rightarrow 'n$ **where** *axis_index* v ≡

SOME i . $v = \text{axis } i \ 1$

lemma *axis_inverse*:

fixes $v :: \text{real}^n$

assumes $v \in \text{Basis}$

shows $\exists i. v = \text{axis } i \ 1$

proof –

have $v \in (\bigcup n. \bigcup r \in \text{Basis}. \{\text{axis } n \ r\})$

using *assms Basis_vec_def* **by** *blast*

then show *?thesis*

by (*force simp add: vec_eq_iff*)

qed

lemma *axis_index*:

fixes $v :: \text{real}^n$

assumes $v \in \text{Basis}$

shows $v = \text{axis } (\text{axis_index } v) \ 1$

by (*metis (mono_tags) assms axis_inverse axis_index_def someI_ex*)

lemma *axis_index_axis* [*simp*]:

fixes $UU :: \text{real}^n$

shows $(\text{axis_index } (\text{axis } u \ 1 :: \text{real}^n)) = (u :: 'n)$

by (*simp add: axis_eq_axis axis_index_def*)

1.10.11 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space

lemma *sum_cong_aux*:

$(\bigwedge x. x \in A \implies f \ x = g \ x) \implies \text{sum } f \ A = \text{sum } g \ A$

by (*auto intro: sum.cong*)

hide_fact (**open**) *sum_cong_aux*

method_setup *vector* = ‹

let

val *ss1* =

simpset_of (*put_simpset HOL_basic_ss context*

addsimps [*@{thm sum.distrib} RS sym*,

@{thm sum_subtractf} RS sym, *@{thm sum_distrib_left}*,

@{thm sum_distrib_right}, *@{thm sum_negf} RS sym*])

val *ss2* =

simpset_of (**context** *addsimps*

[*@{thm plus_vec_def}*], *@{thm times_vec_def}*,

@{thm minus_vec_def}], *@{thm uminus_vec_def}*,

@{thm one_vec_def}], *@{thm zero_vec_def}*], *@{thm vec_def}*,

@{thm scaleR_vec_def}], *@{thm vector_scalar_mult_def}*])

fun *vector_arith_tac* *ctxt* *ths* =

simp_tac (*put_simpset ss1 ctxt*)

THEN' (*fn* *i* => *resolve_tac ctxt* [*@{thms Finite_Cartesian_Product.sum_cong_aux}*])

```

i
  ORELSE resolve_tac ctxt @{thms sum.neutral} i
  ORELSE simp_tac (put_simpset HOL_basic_ss ctxt addsimps [@{thm
vec_eq_iff}]) i)
  (* THEN' TRY o clarify_tac HOL_cs THEN' (TRY o rtac @{thm iffI}) *)
  THEN' asm_full_simp_tac (put_simpset ss2 ctxt addsimps ths)
in
  Attrib.thms >> (fn ths => fn ctxt => SIMPLE_METHOD' (vector_arith_tac
ctxt ths))
end
› lift trivial vector statements to real arith statements

```

lemma *vec_0[simp]: vec 0 = 0 by vector*

lemma *vec_1[simp]: vec 1 = 1 by vector*

lemma *vec_inj[simp]: vec x = vec y \longleftrightarrow x = y by vector*

lemma *vec_in_image_vec: vec x \in (vec ' S) \longleftrightarrow x \in S by auto*

lemma *vec_add: vec(x + y) = vec x + vec y by vector*

lemma *vec_sub: vec(x - y) = vec x - vec y by vector*

lemma *vec_cmul: vec(c * x) = c *s vec x by vector*

lemma *vec_neg: vec(- x) = - vec x by vector*

lemma *vec_scaleR: vec(c * x) = c *_R vec x*
by vector

lemma *vec_sum:*

assumes *finite S*

shows *vec(sum f S) = sum (vec o f) S*

using *assms*

proof *induct*

case *empty*

then show *?case by simp*

next

case *insert*

then show *?case by (auto simp add: vec_add)*

qed

Obvious "component-pushing".

lemma *vec_component [simp]: vec x \$ i = x*
by vector

lemma *vector_mult_component [simp]: (x * y)\$i = x\$i * y\$i*
by vector

lemma *vector_smult_component [simp]: (c *s y)\$i = c * (y\$i)*
by vector

lemma *cond_component*: $(if\ b\ then\ x\ else\ y)\$i = (if\ b\ then\ x\$i\ else\ y\$i)$ **by** *vector*

lemmas *vector_component* =

vec_component vector_add_component vector_mult_component
vector_smult_component vector_minus_component vector_uminus_component
vector_scaleR_component cond_component

1.10.12 Some frequently useful arithmetic lemmas over vectors

instance *vec* :: $(semigroup_mult, finite)$ *semigroup_mult*
by *standard* (*vector mult.assoc*)

instance *vec* :: $(monoid_mult, finite)$ *monoid_mult*
by *standard* *vector+*

instance *vec* :: $(ab_semigroup_mult, finite)$ *ab_semigroup_mult*
by *standard* (*vector mult.commute*)

instance *vec* :: $(comm_monoid_mult, finite)$ *comm_monoid_mult*
by *standard* *vector*

instance *vec* :: $(semiring, finite)$ *semiring*
by *standard* (*vector field_simps*)**+**

instance *vec* :: $(semiring_0, finite)$ *semiring_0*
by *standard* (*vector field_simps*)**+**

instance *vec* :: $(semiring_1, finite)$ *semiring_1*
by *standard* *vector*

instance *vec* :: $(comm_semiring, finite)$ *comm_semiring*
by *standard* (*vector field_simps*)**+**

instance *vec* :: $(comm_semiring_0, finite)$ *comm_semiring_0* ..

instance *vec* :: $(semiring_0_cancel, finite)$ *semiring_0_cancel* ..

instance *vec* :: $(comm_semiring_0_cancel, finite)$ *comm_semiring_0_cancel* ..

instance *vec* :: $(ring, finite)$ *ring* ..

instance *vec* :: $(semiring_1_cancel, finite)$ *semiring_1_cancel* ..

instance *vec* :: $(comm_semiring_1, finite)$ *comm_semiring_1* ..

instance *vec* :: $(ring_1, finite)$ *ring_1* ..

instance *vec* :: $(real_algebra, finite)$ *real_algebra*
by *standard* (*simp_all add: vec_eq_iff*)

instance *vec* :: $(real_algebra_1, finite)$ *real_algebra_1* ..

lemma *of_nat_index*: $(of_nat\ n :: 'a::semiring_1\ ^n)\$i = of_nat\ n$

proof (*induct* *n*)

case *0*

```

    then show ?case by vector
  next
    case Suc
    then show ?case by vector
qed

```

```

lemma one_index [simp]: (1 :: 'a :: one ^ 'n) $ i = 1
  by vector

```

```

lemma neg_one_index [simp]: (- 1 :: 'a :: {one, uminus} ^ 'n) $ i = - 1
  by vector

```

```

instance vec :: (semiring_char_0, finite) semiring_char_0
proof
  fix m n :: nat
  show inj (of_nat :: nat => 'a ^ 'b)
    by (auto intro!: injI simp add: vec_eq_iff of_nat_index)
qed

```

```

instance vec :: (numeral, finite) numeral ..
instance vec :: (semiring_numeral, finite) semiring_numeral ..

```

```

lemma numeral_index [simp]: numeral w $ i = numeral w
  by (induct w) (simp_all only: numeral_simps vector_add_component one_index)

```

```

lemma neg_numeral_index [simp]: - numeral w $ i = - numeral w
  by (simp only: vector_uminus_component numeral_index)

```

```

instance vec :: (comm_ring_1, finite) comm_ring_1 ..
instance vec :: (ring_char_0, finite) ring_char_0 ..

```

```

lemma vector_smult_assoc: a * s (b * s x) = ((a::'a::semigroup_mult) * b) * s x
  by (vector mult.assoc)

```

```

lemma vector_sadd_rdistrib: ((a::'a::semiring) + b) * s x = a * s x + b * s x
  by (vector field_simps)

```

```

lemma vector_add_ldistrib: (c::'a::semiring) * s (x + y) = c * s x + c * s y
  by (vector field_simps)

```

```

lemma vector_smult_lzero[simp]: (0::'a::mult_zero) * s x = 0 by vector

```

```

lemma vector_smult_lid[simp]: (1::'a::monoid_mult) * s x = x by vector

```

```

lemma vector_ssub_ldistrib: (c::'a::ring) * s (x - y) = c * s x - c * s y
  by (vector field_simps)

```

```

lemma vector_smult_neg: (c::'a::ring) * s -x = -(c * s x) by vector

```

```

lemma vector_smult_lneg: -(c::'a::ring) * s x = -(c * s x) by vector

```

```

lemma vector_sneg_minus1: -x = (-1::'a::ring_1) * s x by vector

```

```

lemma vector_smult_rzero[simp]: c * s 0 = (0::'a::mult_zero ^ 'n) by vector

```

```

lemma vector_sub_rdistrib: ((a::'a::ring) - b) * s x = a * s x - b * s x
  by (vector field_simps)

```

```

lemma vec_eq[simp]: (vec m = vec n) <=> (m = n)

```

by (simp add: vec_eq_iff)

lemma *Vector_Spaces_linear_vec* [simp]: *Vector_Spaces.linear* (*) *vector_scalar_mult*
vec
 by *unfold_locales* (*vector algebra_simps*)+

lemma *vector_mul_eq_0* [simp]: $(a * s x = 0) \longleftrightarrow a = (0::'a::idom) \vee x = 0$
 by *vector*

lemma *vector_mul_lcancel* [simp]: $a * s x = a * s y \longleftrightarrow a = (0::'a::field) \vee x = y$
 by (*metis eq_iff_diff_eq_0 vector_mul_eq_0 vector_ssub_ldistrib*)

lemma *vector_mul_rcancel* [simp]: $a * s x = b * s x \longleftrightarrow (a::'a::field) = b \vee x = 0$
 by (*subst eq_iff_diff_eq_0, subst vector_sub_rdistrib [symmetric]*) *simp*

lemma *scalar_mult_eq_scaleR* [abs_def]: $c * s x = c *_R x$
 unfolding *scaleR_vec_def vector_scalar_mult_def* by *simp*

lemma *dist_mul* [simp]: $dist (c * s x) (c * s y) = |c| * dist x y$
 unfolding *dist_norm scalar_mult_eq_scaleR*
 unfolding *scaleR_right_diff_distrib [symmetric]* by *simp*

lemma *sum_component* [simp]:
 fixes $f:: 'a \Rightarrow ('b::comm_monoid_add) ^n$
 shows $(sum f S)\$i = sum (\lambda x. (f x)\$i) S$
proof (*cases finite S*)
 case *True*
 then show ?thesis by *induct simp_all*
next
 case *False*
 then show ?thesis by *simp*
qed

lemma *sum_eq*: $sum f S = (\chi i. sum (\lambda x. (f x)\$i) S)$
 by (*simp add: vec_eq_iff*)

lemma *sum_cmul*:
 fixes $f:: 'c \Rightarrow ('a::semiring_1) ^n$
 shows $sum (\lambda x. c * s f x) S = c * s sum f S$
 by (*simp add: vec_eq_iff sum_distrib_left*)

lemma *linear_vec* [simp]: *linear vec*
 using *Vector_Spaces_linear_vec*
 by *unfold_locales* (*vector algebra_simps*)+

1.10.13 Matrix operations

Matrix notation. NB: an $M \times N$ matrix is of type $(('a, 'n) vec, 'm) vec$, not $(('a, 'm) vec, 'n) vec$

definition $\text{map_matrix}::('a \Rightarrow 'b) \Rightarrow (('a, 'i::\text{finite})\text{vec}, 'j::\text{finite})\text{vec} \Rightarrow (('b, 'i)\text{vec}, 'j)\text{vec}$ **where**

$$\text{map_matrix } f \ x = (\chi \ i \ j. f \ (x \ \$ \ i \ \$ \ j))$$

lemma $\text{nth_map_matrix}[\text{simp}]$: $\text{map_matrix } f \ x \ \$ \ i \ \$ \ j = f \ (x \ \$ \ i \ \$ \ j)$

by $(\text{simp add: map_matrix_def})$

definition $\text{matrix_matrix_mult} :: ('a::\text{semiring_1})^{\wedge n} \wedge^m \Rightarrow 'a \wedge^p \wedge^n \Rightarrow 'a \wedge^p \wedge^m$

(infixl ** 70)

where $m \ ** \ m' == (\chi \ i \ j. \text{sum } (\lambda k. ((m \$ i) \$ k) * ((m' \$ k) \$ j)) \ (UNIV :: 'n \ \text{set})) :: 'a \wedge^p \wedge^m$

definition $\text{matrix_vector_mult} :: ('a::\text{semiring_1})^{\wedge n} \wedge^m \Rightarrow 'a \wedge^n \Rightarrow 'a \wedge^m$

(infixl *v 70)

where $m \ *v \ x \equiv (\chi \ i. \text{sum } (\lambda j. ((m \$ i) \$ j) * (x \$ j)) \ (UNIV :: 'n \ \text{set})) :: 'a \wedge^m$

definition $\text{vector_matrix_mult} :: 'a \wedge^m \Rightarrow ('a::\text{semiring_1})^{\wedge n} \wedge^m \Rightarrow 'a \wedge^n$

(infixl v* 70)

where $v \ v* \ m == (\chi \ j. \text{sum } (\lambda i. ((m \$ i) \$ j) * (v \$ i)) \ (UNIV :: 'm \ \text{set})) :: 'a \wedge^n$

definition $(\text{mat}::'a::\text{zero} \Rightarrow 'a \wedge^n \wedge^n) \ k = (\chi \ i \ j. \text{if } i = j \ \text{then } k \ \text{else } 0)$

definition $\text{transpose}::'a \wedge^n \wedge^m \Rightarrow 'a \wedge^m \wedge^n$ **where**

$$(\text{transpose}::'a \wedge^n \wedge^m \Rightarrow 'a \wedge^m \wedge^n) \ A = (\chi \ i \ j. ((A \$ j) \$ i))$$

definition $(\text{row}::'m \Rightarrow 'a \wedge^n \wedge^m \Rightarrow 'a \wedge^n) \ i \ A = (\chi \ j. ((A \$ i) \$ j))$

definition $(\text{column}::'n \Rightarrow 'a \wedge^n \wedge^m \Rightarrow 'a \wedge^m) \ j \ A = (\chi \ i. ((A \$ i) \$ j))$

definition $\text{rows}(A::'a \wedge^n \wedge^m) = \{ \text{row } i \ A \mid i. i \in (UNIV :: 'm \ \text{set}) \}$

definition $\text{columns}(A::'a \wedge^n \wedge^m) = \{ \text{column } i \ A \mid i. i \in (UNIV :: 'n \ \text{set}) \}$

lemma $\text{times0_left}[\text{simp}]$: $(0::'a::\text{semiring_1})^{\wedge n} \wedge^m ** (A::'a \wedge^p \wedge^n) = 0$

by $(\text{simp add: matrix_matrix_mult_def zero_vec_def})$

lemma $\text{times0_right}[\text{simp}]$: $(A::'a::\text{semiring_1})^{\wedge n} \wedge^m ** (0::'a \wedge^p \wedge^n) = 0$

by $(\text{simp add: matrix_matrix_mult_def zero_vec_def})$

lemma $\text{mat_0}[\text{simp}]$: $\text{mat } 0 = 0$ **by** (vector mat_def)

lemma $\text{matrix_add_ldistrib}$: $(A ** (B + C)) = (A ** B) + (A ** C)$

by $(\text{vector matrix_matrix_mult_def sum.distrib[symmetric] field_simps})$

lemma $\text{matrix_mul_lid}[\text{simp}]$:

fixes $A :: 'a::\text{semiring_1}^{\wedge m} \wedge^n$

shows $\text{mat } 1 ** A = A$

unfolding $\text{matrix_matrix_mult_def mat_def}$

by $(\text{auto simp: if_distrib if_distribR sum.delta'[OF finite] cong: if_cong})$

lemma $\text{matrix_mul_rid}[\text{simp}]$:

fixes $A :: 'a::\text{semiring_1}^{\wedge m} \wedge^n$

shows $A ** \text{mat } 1 = A$

unfolding $\text{matrix_matrix_mult_def mat_def}$

by (auto simp: if_distrib if_distribR sum.delta'[OF finite] cong: if_cong)

proposition *matrix_mul_assoc*: $A ** (B ** C) = (A ** B) ** C$
apply (vector_matrix_matrix_mult_def sum_distrib_left sum_distrib_right mult.assoc)
apply (subst sum.swap)
apply simp
done

proposition *matrix_vector_mul_assoc*: $A *v (B *v x) = (A ** B) *v x$
apply (vector_matrix_matrix_mult_def matrix_vector_mult_def
sum_distrib_left sum_distrib_right mult.assoc)
apply (subst sum.swap)
apply simp
done

proposition *scalar_matrix_assoc*:
fixes $A :: ('a::real_algebra_1)^m^n$
shows $k *_R (A ** B) = (k *_R A) ** B$
by (simp add: matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff
scaleR_sum_right)

proposition *matrix_scalar_ac*:
fixes $A :: ('a::real_algebra_1)^m^n$
shows $A ** (k *_R B) = k *_R A ** B$
by (simp add: matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff)

lemma *matrix_vector_mul_lid* [simp]: $mat\ 1 *v x = (x::'a::semiring_1^n)$
apply (vector_matrix_vector_mult_def mat_def)
apply (simp add: if_distrib if_distribR cong del: if_weak_cong)
done

lemma *matrix_transpose_mul*:
 $transpose(A ** B) = transpose\ B ** transpose\ (A::'a::comm_semiring_1^{_}_{_})$
by (simp add: matrix_matrix_mult_def transpose_def vec_eq_iff mult.commute)

lemma *matrix_mult_transpose_dot_column*:
shows $transpose\ A ** A = (\chi\ i\ j.\ inner\ (column\ i\ A)\ (column\ j\ A))$
by (simp add: matrix_matrix_mult_def vec_eq_iff transpose_def column_def
inner_vec_def)

lemma *matrix_mult_transpose_dot_row*:
shows $A ** transpose\ A = (\chi\ i\ j.\ inner\ (row\ i\ A)\ (row\ j\ A))$
by (simp add: matrix_matrix_mult_def vec_eq_iff transpose_def row_def in-
ner_vec_def)

lemma *matrix_eq*:
fixes $A\ B :: 'a::semiring_1^n^m$
shows $A = B \longleftrightarrow (\forall x.\ A *v x = B *v x)$ (is ?lhs \longleftrightarrow ?rhs)
proof

```

assume ?rhs
then show ?lhs
  apply (subst vec_eq_iff)
  apply (clarsimp simp add: matrix_vector_mult_def if_distrib if_distribR
vec_eq_iff cong: if_cong)
  apply (erule_tac x=axis ia 1 in allE)
  apply (erule_tac x=i in allE)
  apply (auto simp add: if_distrib if_distribR axis_def
sum.delta[OF finite] cong del: if_weak_cong)
done
qed auto

```

```

lemma matrix_vector_mul_component:  $(A * v x) \$k = \text{inner } (A \$k) x$ 
by (simp add: matrix_vector_mult_def inner_vec_def)

```

```

lemma dot_lm mul_matrix:  $\text{inner } ((x::\text{real } \wedge) v * A) y = \text{inner } x (A * v y)$ 
apply (simp add: inner_vec_def matrix_vector_mult_def vector_matrix_mult_def
sum_distrib_right sum_distrib_left ac_simps)
apply (subst sum.swap)
apply simp
done

```

```

lemma transpose_mat [simp]:  $\text{transpose } (\text{mat } n) = \text{mat } n$ 
by (vector transpose_def mat_def)

```

```

lemma transpose_transpose [simp]:  $\text{transpose } (\text{transpose } A) = A$ 
by (vector transpose_def)

```

```

lemma row_transpose [simp]:  $\text{row } i (\text{transpose } A) = \text{column } i A$ 
by (simp add: row_def column_def transpose_def vec_eq_iff)

```

```

lemma column_transpose [simp]:  $\text{column } i (\text{transpose } A) = \text{row } i A$ 
by (simp add: row_def column_def transpose_def vec_eq_iff)

```

```

lemma rows_transpose [simp]:  $\text{rows } (\text{transpose } A) = \text{columns } A$ 
by (auto simp add: rows_def columns_def intro: set_eqI)

```

```

lemma columns_transpose [simp]:  $\text{columns } (\text{transpose } A) = \text{rows } A$ 
by (metis transpose_transpose rows_transpose)

```

```

lemma transpose_scalar:  $\text{transpose } (k *_R A) = k *_R \text{transpose } A$ 
unfolding transpose_def
by (simp add: vec_eq_iff)

```

```

lemma transpose_iff [iff]:  $\text{transpose } A = \text{transpose } B \iff A = B$ 
by (metis transpose_transpose)

```

```

lemma matrix_mult_sum:
 $(A::\text{'a}::\text{comm\_semiring } 1 \wedge n \wedge m) * v x = \text{sum } (\lambda i. (x \$i) * s \text{ column } i A)$  (UNIV::

```

'n set)

by (simp add: matrix_vector_mult_def vec_eq_iff column_def mult.commute)

lemma vector_componentwise:

(x::'a::ring_1ⁿ) = (χ j. $\sum_{i \in UNIV} (x\$i) * (axis\ i\ 1 :: 'a^n) \$ j$)

by (simp add: axis_def if_distrib sum.If_cases vec_eq_iff)

lemma basis_expansion: $\text{sum } (\lambda i. (x\$i) * s\ axis\ i\ 1)\ UNIV = (x::('a::ring_1)^n)$

by (auto simp add: axis_def vec_eq_iff if_distrib sum.If_cases cong del: if_weak_cong)

Correspondence between matrices and linear operators.

definition matrix :: ('a::{plus,times, one, zero})^m \Rightarrow 'aⁿ \Rightarrow 'a^mⁿ

where matrix f = ($\chi\ i\ j. (f(axis\ j\ 1))\i)

lemma matrix_id_mat_1: matrix id = mat 1

by (simp add: mat_def matrix_def axis_def)

lemma matrix_scaleR: (matrix ((*_R) r)) = mat r

by (simp add: mat_def matrix_def axis_def if_distrib cong: if_cong)

lemma matrix_vector_mul_linear[intro, simp]: linear ($\lambda x. A * v (x::'a::real_algebra_1$
^{^_)})

by (simp add: linear_iff matrix_vector_mult_def vec_eq_iff field_simps sum_distrib_left
sum.distrib scaleR_right.sum)

lemma vector_matrix_left_distrib [algebra_simps]:

shows (x + y) v * A = x v * A + y v * A

unfolding vector_matrix_mult_def

by (simp add: algebra_simps sum.distrib vec_eq_iff)

lemma matrix_vector_right_distrib [algebra_simps]:

A * v (x + y) = A * v x + A * v y

by (vector matrix_vector_mult_def sum.distrib distrib_left)

lemma matrix_vector_mult_diff_distrib [algebra_simps]:

fixes A :: 'a::ring_1ⁿ^m

shows A * v (x - y) = A * v x - A * v y

by (vector matrix_vector_mult_def sum_subtractf right_diff_distrib)

lemma matrix_vector_mult_scaleR [algebra_simps]:

fixes A :: realⁿ^m

shows A * v (c *_R x) = c *_R (A * v x)

using linear_iff matrix_vector_mul_linear by blast

lemma matrix_vector_mult_0_right [simp]: A * v 0 = 0

by (simp add: matrix_vector_mult_def vec_eq_iff)

lemma matrix_vector_mult_0 [simp]: 0 * v w = 0

by (simp add: matrix_vector_mult_def vec_eq_iff)

lemma *matrix_vector_mult_add_rdistrib* [*algebra_simps*]:
 $(A + B) * v x = (A * v x) + (B * v x)$
by (*vector_matrix_vector_mult_def sum.distrib distrib_right*)

lemma *matrix_vector_mult_diff_rdistrib* [*algebra_simps*]:
fixes $A :: 'a :: \text{ring } 1 \wedge n \wedge m$
shows $(A - B) * v x = (A * v x) - (B * v x)$
by (*vector_matrix_vector_mult_def sum_subtractf left_diff_distrib*)

lemma *matrix_vector_column*:
 $(A :: 'a :: \text{comm_semiring } 1 \wedge n \wedge _) * v x = \text{sum } (\lambda i. (x \$ i) * s ((\text{transpose } A) \$ i))$
(*UNIV :: 'n set*)
by (*simp add: matrix_vector_mult_def transpose_def vec_eq_iff mult.commute*)

1.10.14 Inverse matrices (not necessarily square)

definition
 $\text{invertible}(A :: 'a :: \text{semiring } 1 \wedge n \wedge m) \longleftrightarrow (\exists A' :: 'a \wedge m \wedge n. A ** A' = \text{mat } 1 \wedge A' ** A = \text{mat } 1)$

definition
 $\text{matrix_inv}(A :: 'a :: \text{semiring } 1 \wedge n \wedge m) =$
(*SOME A' :: 'a \wedge m \wedge n. A ** A' = mat 1 \wedge A' ** A = mat 1*)

lemma *inj_matrix_vector_mult*:
fixes $A :: 'a :: \text{field } \wedge n \wedge m$
assumes *invertible A*
shows *inj ((*v) A)*
by (*metis assms inj_on_inverseI invertible_def matrix_vector_mul_assoc matrix_vector_mul_lid*)

lemma *scalar_invertible*:
fixes $A :: ('a :: \text{real_algebra } 1) \wedge m \wedge n$
assumes $k \neq 0$ **and** *invertible A*
shows *invertible (k *_R A)*

proof –
obtain A' **where** $A ** A' = \text{mat } 1$ **and** $A' ** A = \text{mat } 1$
using *assms unfolding invertible_def by auto*
with $\langle k \neq 0 \rangle$
have $(k *_{\mathbb{R}} A) ** ((1/k) *_{\mathbb{R}} A') = \text{mat } 1$ $((1/k) *_{\mathbb{R}} A') ** (k *_{\mathbb{R}} A) = \text{mat } 1$
by (*simp_all add: assms matrix_scalar_ac*)
thus *invertible (k *_R A)*
unfolding *invertible_def by auto*
qed

proposition *scalar_invertible_iff*:
fixes $A :: ('a :: \text{real_algebra } 1) \wedge m \wedge n$
assumes $k \neq 0$ **and** *invertible A*

shows $\text{invertible } (k *_{\mathbb{R}} A) \longleftrightarrow k \neq 0 \wedge \text{invertible } A$
by (*simp add: assms scalar_invertible*)

lemma *vector_transpose_matrix* [*simp*]: $x v * \text{transpose } A = A * v x$
unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*
by *simp*

lemma *transpose_matrix_vector* [*simp*]: $\text{transpose } A * v x = x v * A$
unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*
by *simp*

lemma *vector_scalar_commute*:
fixes $A :: 'a::\{\text{field}\}^m \wedge^n$
shows $A * v (c * s x) = c * s (A * v x)$
by (*simp add: vector_scalar_mult_def matrix_vector_mult_def mult_ac sum_distrib_left*)

lemma *scalar_vector_matrix_assoc*:
fixes $k :: 'a::\{\text{field}\}$ **and** $x :: 'a::\{\text{field}\}^n$ **and** $A :: 'a^m \wedge^n$
shows $(k * s x) v * A = k * s (x v * A)$
by (*metis transpose_matrix_vector vector_scalar_commute*)

lemma *vector_matrix_mult_0* [*simp*]: $0 v * A = 0$
unfolding *vector_matrix_mult_def* **by** (*simp add: zero_vec_def*)

lemma *vector_matrix_mult_0_right* [*simp*]: $x v * 0 = 0$
unfolding *vector_matrix_mult_def* **by** (*simp add: zero_vec_def*)

lemma *vector_matrix_mul_lid* [*simp*]:
fixes $v :: ('a::\{\text{semiring}_1\})^n$
shows $v v * \text{mat } 1 = v$
by (*metis matrix_vector_mul_lid transpose_mat vector_transpose_matrix*)

lemma *scaleR_vector_matrix_assoc*:
fixes $k :: \text{real}$ **and** $x :: \text{real}^n$ **and** $A :: \text{real}^m \wedge^n$
shows $(k *_{\mathbb{R}} x) v * A = k *_{\mathbb{R}} (x v * A)$
by (*metis matrix_vector_mult_scaleR transpose_matrix_vector*)

proposition *vector_scaleR_matrix_ac*:
fixes $k :: \text{real}$ **and** $x :: \text{real}^n$ **and** $A :: \text{real}^m \wedge^n$
shows $x v * (k *_{\mathbb{R}} A) = k *_{\mathbb{R}} (x v * A)$

proof –

have $x v * (k *_{\mathbb{R}} A) = (k *_{\mathbb{R}} x) v * A$
unfolding *vector_matrix_mult_def*
by (*simp add: algebra_simps*)
with *scaleR_vector_matrix_assoc*
show $x v * (k *_{\mathbb{R}} A) = k *_{\mathbb{R}} (x v * A)$
by *auto*

qed

end

1.11 Linear Algebra on Finite Cartesian Products

theory *Cartesian_Space*

imports

HOL-Combinatorics.Transposition

Finite_Cartesian_Product

Linear_Algebra

begin

1.11.1 Type $(\prime a, \prime n)$ *vec* and fields as vector spaces

definition *cart_basis* = $\{axis\ i\ 1\ \mid\ i.\ i \in UNIV\}$

lemma *finite_cart_basis*: *finite* (*cart_basis*) **unfolding** *cart_basis_def*
using *finite_Atleast_Atmost_nat* **by** *fastforce*

lemma *card_cart_basis*: *card* (*cart_basis::('a::zero_neq_oneⁱ) set*) = *CARD('i)*
unfolding *cart_basis_def Setcompr_eq_image*
by (*rule card_image*) (*auto simp: inj_on_def axis_eq_axis*)

interpretation *vec*: *vector_space* (*s)
by *unfold_locales* (*vector_algebra_simps*)+

lemma *independent_cart_basis*: *vec.independent* (*cart_basis*)

proof (*rule vec.independent_if_scalars_zero*)

show *finite* (*cart_basis*) **using** *finite_cart_basis* .

fix *f::('a, 'b) vec* \Rightarrow '*a* **and** *x::('a, 'b) vec*

assume *eq_0*: $(\sum x \in cart_basis. f\ x\ *s\ x) = 0$ **and** *x_in*: $x \in cart_basis$

obtain *i* **where** *x*: $x = axis\ i\ 1$ **using** *x_in* **unfolding** *cart_basis_def* **by** *auto*

have *sum_eq_0*: $(\sum x \in (cart_basis) - \{x\}. f\ x\ * (x\ \$\ i)) = 0$

proof (*intro sum.neutral ballI*)

fix *y* **assume** *y*: $y \in cart_basis - \{x\}$

obtain *a* **where** *a*: $y = axis\ a\ 1$ **and** *a_not_i*: $a \neq i$

using *y* *x* **unfolding** *cart_basis_def* **by** *auto*

have $y\ \$\ i = 0$ **unfolding** *a* *axis_def* **using** *a_not_i* **by** *auto*

thus $f\ y\ * y\ \$\ i = 0$ **by** *simp*

qed

have $0 = (\sum x \in cart_basis. f\ x\ *s\ x)\ \$\ i$ **using** *eq_0* **by** *simp*

also **have** $\dots = (\sum x \in cart_basis. (f\ x\ *s\ x)\ \$\ i)$ **unfolding** *sum_component* ..

also **have** $\dots = (\sum x \in cart_basis. f\ x\ * (x\ \$\ i))$ **unfolding** *vector_smult_component*

..

also **have** $\dots = f\ x\ * (x\ \$\ i) + (\sum x \in (cart_basis) - \{x\}. f\ x\ * (x\ \$\ i))$

by (*rule sum.remove[OF finite_cart_basis x_in]*)

also **have** $\dots = f\ x\ * (x\ \$\ i)$ **unfolding** *sum_eq_0* **by** *simp*

also **have** $\dots = f\ x$ **unfolding** *x* *axis_def* **by** *auto*

finally **show** $f\ x = 0$..

qed

```

lemma span_cart_basis [simp]: vec.span (cart_basis) = UNIV
proof -
  have x ∈ vec.span cart_basis for x :: ('a, 'b) vec
  proof -
    let ?f=λv. x $ (THE i. v = axis i 1)
    have x $ i = (∑ v∈cart_basis. x $ (THE i. v = axis i 1) *s v) $ i for i::'b
    proof -
      let ?w = axis i (1::'a)
      have the_eq_i: (THE a. ?w = axis a 1) = i
      by (rule the_equality, auto simp: axis_eq_axis)
      have sum_eq_0: (∑ v∈(cart_basis) - {?w}. x $ (THE i. v = axis i 1) * v
      $ i) = 0
      proof (intro sum.neutral ballI)
        fix y:: ('a, 'b) vec
        assume y: y ∈ cart_basis - {?w}
        obtain j where j: y = axis j 1 and i_not_j: i ≠ j
        using y unfolding cart_basis_def by auto
        have the_eq_j: (THE i. y = axis i 1) = j
        by (simp add: axis_eq_axis j)
        show x $ (THE i. y = axis i 1) * y $ i = 0
        by (simp add: axis_def i_not_j j)
      qed
      have (∑ v∈cart_basis. x $ (THE i. v = axis i 1) *s v) $ i
      = (∑ v∈cart_basis. x $ (THE i. v = axis i 1) * v $ i)
      by force
      also have ... = x $ (THE a. ?w = axis a 1) * ?w $ i + (∑ v∈(cart_basis)
      - {?w}. x $ (THE i. v = axis i 1) * v $ i)
      by (rule sum.remove[OF finite_cart_basis], auto simp add: cart_basis_def)
      also have ... = x $ (THE a. ?w = axis a 1) * ?w $ i
      unfolding sum_eq_0 by simp
      also have ... = x $ i
      unfolding the_eq_i unfolding axis_def by auto
      finally show ?thesis by simp
    qed
  then show x ∈ vec.span (cart_basis)
  by (metis (no_types, lifting) vec.span_base vec.span_scale vec.span_sum
  vec_eq_iff)
  qed
  then show ?thesis by auto
qed

```

```

interpretation vec: finite_dimensional_vector_space (*s) cart_basis
  by (unfold_locales, auto simp add: finite_cart_basis independent_cart_basis
  span_cart_basis)

```

```

lemma matrix_vector_mul_linear_gen[intro, simp]:
  Vector_Spaces.linear (*s) (*s) ((*v) A)

```

by *unfold locales*
 (vector_matrix_vector_mult_def sum.distrib algebra_simps)+

lemma *span_vec_eq*: $vec.span\ X = span\ X$
and *dim_vec_eq*: $vec.dim\ X = dim\ X$
and *dependent_vec_eq*: $vec.dependent\ X = dependent\ X$
and *subspace_vec_eq*: $vec.subspace\ X = subspace\ X$
for $X::(real^n)\ set$
unfolding *span_raw_def dim_raw_def dependent_raw_def subspace_raw_def*
by (auto simp: scalar_mult_eq_scaleR)

lemma *linear_componentwise*:
fixes $f::'a::field\ ^m \Rightarrow 'a\ ^n$
assumes $lf: Vector_Spaces.linear\ (*s)\ (*s)\ f$
shows $(f\ x)\$j = sum\ (\lambda i. (x\$i) * (f\ (axis\ i\ 1)\$j))\ (UNIV::'m\ set)\ (is\ ?lhs = ?rhs)$
proof –
interpret $lf: Vector_Spaces.linear\ (*s)\ (*s)\ f$
using *lf* .
let $?M = (UNIV::'m\ set)$
let $?N = (UNIV::'n\ set)$
have $fM: finite\ ?M$ **by** *simp*
have $?rhs = (sum\ (\lambda i. (x\$i) * s\ (f\ (axis\ i\ 1))))\ ?M)\j
unfolding *sum_component* **by** *simp*
then show *?thesis*
unfolding *lf.sum[symmetric] lf.scale[symmetric]*
unfolding *basis_expansion* **by** *auto*
qed

interpretation *vec*: $Vector_Spaces.linear\ (*s)\ (*s)\ (*v)\ A$
using *matrix_vector_mul_linear_gen*.

interpretation *vec*: $finite_dimensional_vector_space_pair\ (*s)\ cart_basis\ (*s)\ cart_basis\ ..$

lemma *matrix_works*:
assumes $lf: Vector_Spaces.linear\ (*s)\ (*s)\ f$
shows $matrix\ f * v\ x = f\ (x::'a::field\ ^n)$
proof –
have $\forall i. (\sum_{j \in UNIV} x\ \$j * f\ (axis\ j\ 1)\ \$i) = f\ x\ \i
by (*simp add: Cartesian_Space.linear_componentwise lf*)
then show *?thesis*
by (*simp add: matrix_def matrix_vector_mult_def vec_eq_iff mult commute*)
qed

lemma *matrix_of_matrix_vector_mul[simp]*: $matrix(\lambda x. A * v\ (x::'a::field\ ^n)) = A$
by (*simp add: matrix_eq matrix_works*)

lemma *matrix_compose_gen*:

assumes *lf*: *Vector_Spaces.linear* (*s) (*s) (*f*::'a::{field}ⁿ ⇒ 'a^m)
and *lg*: *Vector_Spaces.linear* (*s) (*s) (*g*::'a^m ⇒ 'a^l)
shows *matrix* (g o f) = *matrix* g ** *matrix* f
using *lf lg* *Vector_Spaces.linear_compose[OF lf lg]* *matrix_works[OF Vector_Spaces.linear_compose[OF lf lg]]*
by (*simp add: matrix_eq matrix_works matrix_vector_mul_assoc[symmetric]*
o_def)

lemma *matrix_compose*:

assumes *linear* (*f*::realⁿ ⇒ real^m) *linear* (*g*::real^m ⇒ real^l)
shows *matrix* (g o f) = *matrix* g ** *matrix* f
using *matrix_compose_gen[of f g]* *assms*
by (*simp add: linear_def scalar_mult_eq_scaleR*)

lemma *left_invertible_transpose*:

(∃ (B). B ** *transpose* (A) = *mat* (1::'a::comm_semiring_1)) ⟷ (∃ (B). A **
B = *mat* 1)
by (*metis matrix_transpose_mul transpose_mat transpose_transpose*)

lemma *right_invertible_transpose*:

(∃ (B). *transpose* (A) ** B = *mat* (1::'a::comm_semiring_1)) ⟷ (∃ (B). B **
A = *mat* 1)
by (*metis matrix_transpose_mul transpose_mat transpose_transpose*)

lemma *linear_matrix_vector_mul_eq*:

Vector_Spaces.linear (*s) (*s) *f* ⟷ *linear* (*f* :: realⁿ ⇒ real^m)
by (*simp add: scalar_mult_eq_scaleR linear_def*)

lemma *matrix_vector_mul[simp]*:

Vector_Spaces.linear (*s) (*s) *g* ⟹ (λy. *matrix* g *v y) = *g*
linear *f* ⟹ (λx. *matrix* f *v x) = *f*
bounded_linear *f* ⟹ (λx. *matrix* f *v x) = *f*
for *f* :: realⁿ ⇒ real^m
by (*simp_all add: ext matrix_works linear_matrix_vector_mul_eq linear_linear*)

lemma *matrix_left_invertible_injective*:

fixes A :: 'a::fieldⁿ^m
shows (∃ B. B ** A = *mat* 1) ⟷ *inj* ((*v) A)

proof *safe*

fix B

assume B: B ** A = *mat* 1

show *inj* ((*v) A)

unfolding *inj_on_def*

by (*metis B matrix_vector_mul_assoc matrix_vector_mul_lid*)

next

assume *inj* ((*v) A)

from *vec.linear_injective_left_inverse[OF matrix_vector_mul_linear_gen this]*

obtain *g* **where** *Vector_Spaces.linear* (*s) (*s) *g* **and** *g* o (*v) A = *id*

by *blast*
then have $\text{matrix } g ** A = \text{mat } 1$
by (*metis matrix_compose_gen matrix_id_mat_1 matrix_of_matrix_vector_mul*
vec.linear_axioms)
then show $\exists B. B ** A = \text{mat } 1$
by *metis*
qed

lemma *matrix_left_invertible_ker*:
 $(\exists B. (B :: 'a :: \{\text{field}\}^{\sim m \sim n}) ** (A :: 'a :: \{\text{field}\}^{\sim n \sim m}) = \text{mat } 1) \longleftrightarrow (\forall x. A * v$
 $x = 0 \longrightarrow x = 0)$
by (*simp add: matrix_left_invertible_injective vec.inj_iff_eq_0*)

lemma *matrix_right_invertible_surjective*:
 $(\exists B. (A :: 'a :: \{\text{field}\}^{\sim n \sim m}) ** (B :: 'a :: \{\text{field}\}^{\sim m \sim n}) = \text{mat } 1) \longleftrightarrow \text{surj } (\lambda x. A * v x)$
proof –
have $\bigwedge B x. A ** B = \text{mat } 1 \implies \exists y. x = A * v y$
by (*metis matrix_vector_mul_assoc matrix_vector_mul_lid*)
moreover have $\forall x. \exists xa. x = A * v xa \implies \exists B. A ** B = \text{mat } 1$
by (*metis (mono_tags, lifting) matrix_compose_gen matrix_id_mat_1 ma-*
trix_of_matrix_vector_mul surj_def vec.linear_axioms vec.linear_surjective_right_inverse)
ultimately show *?thesis*
by (*auto simp: image_def set_eq_iff*)
qed

lemma *matrix_left_invertible_independent_columns*:
fixes $A :: 'a :: \{\text{field}\}^{\sim n \sim m}$
shows $(\exists (B :: 'a :: \{\text{field}\}^{\sim m \sim n}). B ** A = \text{mat } 1) \longleftrightarrow$
 $(\forall c. \text{sum } (\lambda i. c i * s \text{ column } i A) (\text{UNIV} :: 'n \text{ set}) = 0 \longrightarrow (\forall i. c i = 0))$
(is ?lhs \longleftrightarrow ?rhs)
proof –
let $?U = \text{UNIV} :: 'n \text{ set}$
have $c i = 0$
if $\forall x. A * v x = 0 \longrightarrow x = 0 \text{ sum } (\lambda i. c i * s \text{ column } i A) ?U = 0$ **for** $c i$
by (*metis (no_types) UNIV_I matrix_mult_sum vec_lambda_eta vec_nth_cases*
zero_vec_def that)
moreover have $x = 0$ **if** $A * v x = 0$ *?rhs* **for** x
by (*metis (full_types) matrix_mult_sum that vec_eq_iff zero_index*)
ultimately show *?thesis*
unfolding *matrix_left_invertible_ker* **by** *auto*
qed

lemma *matrix_right_invertible_independent_rows*:
fixes $A :: 'a :: \{\text{field}\}^{\sim n \sim m}$
shows $(\exists (B :: 'a :: \{\text{field}\}^{\sim m \sim n}). A ** B = \text{mat } 1) \longleftrightarrow$
 $(\forall c. \text{sum } (\lambda i :: 'm. c i * s \text{ row } i A) \text{ UNIV} = 0 \longrightarrow (\forall i. c i = 0))$
by (*simp add: matrix_left_invertible_independent_columns flip: left_invertible_transpose*)

lemma *matrix_right_invertible_span_columns*:

```

( $\exists (B::'a::field \wedge n \wedge m). (A::'a \wedge m \wedge n) ** B = \text{mat } 1$ )  $\longleftrightarrow$ 
   $\text{vec.span (columns } A) = \text{UNIV (is } ?lhs = ?rhs)$ 
proof -
  let ?U = UNIV :: 'm set
  have fU: finite ?U by simp
  have lhseq: ?lhs  $\longleftrightarrow$  ( $\forall y. \exists (x::'a \wedge m). \text{sum } (\lambda i. (x\$i) *s \text{column } i \text{ } A) \text{ } ?U = y$ )
    unfolding matrix_right_invertible_surjective_matrix_mult_sum_surj_def
    by (simp add: eq_commute)
  have rhseq: ?rhs  $\longleftrightarrow$  ( $\forall x. x \in \text{vec.span (columns } A)$ ) by blast
  { assume h: ?lhs
    { fix x:: 'a  $\wedge$  m
      obtain y :: 'a  $\wedge$  m where y:  $\text{sum } (\lambda i. (y\$i) *s \text{column } i \text{ } A) \text{ } ?U = x$ 
        using h lhseq by blast
      then have x  $\in \text{vec.span (columns } A)$ 
        by (metis (mono_tags, lifting) columns_def mem_Collect_eq vec.span_base
vec.span_scale vec.span_sum)
    }
    then have ?rhs unfolding rhseq by blast }
  moreover
  { assume h: ?rhs
    let ?P =  $\lambda (y::'a \wedge n). \exists (x::'a \wedge m). \text{sum } (\lambda i. (x\$i) *s \text{column } i \text{ } A) \text{ } ?U = y$ 
      { fix y
        have y  $\in \text{vec.span (columns } A)$ 
          unfolding h by blast
        then have ?P y
          proof (induction rule: vec.span_induct_alt)
            case base
            then show ?case
              by (metis (full_types) matrix_mult_sum_matrix_vector_mult_0_right)
          next
            case (step c y1 y2)
            from step obtain i where i:  $i \in ?U \text{ } y1 = \text{column } i \text{ } A$ 
              unfolding columns_def by blast
            obtain x:: 'a  $\wedge$  m where x:  $\text{sum } (\lambda i. (x\$i) *s \text{column } i \text{ } A) \text{ } ?U = y2$ 
              using step by blast
            let ?x = ( $\chi \text{ } j. \text{if } j = i \text{ then } c + (x\$i) \text{ else } (x\$j)$ )::'a  $\wedge$  m
            show ?case
              proof (rule exI[where x = ?x], vector, auto simp add: i x[symmetric]
if_distrib_distrib_left if_distribR cong del: if_weak_cong)
                fix j
                have th:  $\forall xa \in ?U. (\text{if } xa = i \text{ then } (c + (x\$i)) * ((\text{column } xa \text{ } A)\$j)$ 
                   $\text{else } (x\$xa) * ((\text{column } xa \text{ } A)\$j)) = (\text{if } xa = i \text{ then } c * ((\text{column } i \text{ } A)\$j)$ 
 $\text{else } 0) + ((x\$xa) * ((\text{column } xa \text{ } A)\$j))$ 
                  using i(1) by (simp add: field_simps)
                have sum  $(\lambda xa. \text{if } xa = i \text{ then } (c + (x\$i)) * ((\text{column } xa \text{ } A)\$j)$ 
                   $\text{else } (x\$xa) * ((\text{column } xa \text{ } A)\$j)) \text{ } ?U = \text{sum } (\lambda xa. (\text{if } xa = i \text{ then } c * ((\text{column } i \text{ } A)\$j)$ 
 $\text{else } 0) + ((x\$xa) * ((\text{column } xa \text{ } A)\$j))) \text{ } ?U$ 
                  using th by force
                also have ... =  $\text{sum } (\lambda xa. \text{if } xa = i \text{ then } c * ((\text{column } i \text{ } A)\$j) \text{ else } 0) \text{ } ?U$ 

```

```

+ sum (λxa. ((x$xa) * ((column xa A)$j))) ?U
  by (simp add: sum.distrib)
  also have ... = c * ((column i A)$j) + sum (λxa. ((x$xa) * ((column xa
A)$j))) ?U
    unfolding sum.delta[OF fU] using i(1) by simp
    finally show sum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
      else (x$xa) * ((column xa A)$j))) ?U
      = c * ((column i A)$j) + sum (λxa. ((x$xa) * ((column xa
A)$j))) ?U .
  qed
  qed
}
then have ?lhs unfolding lhseq ..
}
ultimately show ?thesis by blast
qed

```

lemma *matrix_left_invertible_span_rows_gen*:

$(\exists (B::'a^{m \times n}). B ** (A::'a::\text{field}^{n \times m}) = \text{mat } 1) \longleftrightarrow \text{vec.span (rows } A) = \text{UNIV}$

by (metis *columns_transpose matrix_right_invertible_span_columns right_invertible_transpose*)

lemma *matrix_left_invertible_span_rows*:

$(\exists (B::\text{real}^{m \times n}). B ** (A::\text{real}^{n \times m}) = \text{mat } 1) \longleftrightarrow \text{span (rows } A) = \text{UNIV}$
 using *matrix_left_invertible_span_rows_gen*[of A] by (simp add: *span_vec_eq*)

lemma *matrix_left_right_inverse*:

fixes $A A' :: 'a::\{\text{field}\}^{n \times n}$

shows $A ** A' = \text{mat } 1 \longleftrightarrow A' ** A = \text{mat } 1$

proof –

{ fix $A A' :: 'a^{n \times n}$

assume $AA': A ** A' = \text{mat } 1$

have $sA: \text{surj } ((*) A)$

using *AA' matrix_right_invertible_surjective* by auto

obtain $f' :: 'a^{n \times n} \Rightarrow 'a^{n \times n}$

where $f': \text{Vector_Spaces.linear } (*) (*) f' \forall x. f' (A *v x) = x \forall x. A *v f' x = x$

using *sA vec.linear_surjective_isomorphism* by blast

have $\text{matrix } f' ** A = \text{mat } 1$

by (metis *f' matrix_eq matrix_vector_mul_assoc matrix_vector_mul_lid matrix_works*)

hence $A' ** A = \text{mat } 1$

by (metis *AA' matrix_mul_assoc matrix_mul_lid*)

}
 then show ?thesis by blast

qed

lemma *invertible_left_inverse*:

fixes $A :: 'a::\{\text{field}\}^{n \times n}$

shows $\text{invertible } A \longleftrightarrow (\exists (B::'a^{n \times n}). B ** A = \text{mat } 1)$
by (metis invertible_def matrix_left_right_inverse)

lemma *invertible_right_inverse*:

fixes $A :: 'a::\{\text{field}\}^{n \times n}$
shows $\text{invertible } A \longleftrightarrow (\exists (B::'a^{n \times n}). A ** B = \text{mat } 1)$
by (metis invertible_def matrix_left_right_inverse)

lemma *invertible_mult*:

assumes $\text{inv_}A: \text{invertible } A$
and $\text{inv_}B: \text{invertible } B$
shows $\text{invertible } (A ** B)$

proof –

obtain A' **where** $AA': A ** A' = \text{mat } 1$ **and** $A'A: A' ** A = \text{mat } 1$
using $\text{inv_}A$ **unfolding** *invertible_def* **by** *blast*
obtain B' **where** $BB': B ** B' = \text{mat } 1$ **and** $B'B: B' ** B = \text{mat } 1$
using $\text{inv_}B$ **unfolding** *invertible_def* **by** *blast*
have $A ** B ** (B' ** A') = \text{mat } 1$
by (metis AA' BB' *matrix_mul_assoc* *matrix_mul_rid*)
moreover have $B' ** A' ** (A ** B) = \text{mat } 1$
by (metis $A'A$ $B'B$ *matrix_mul_assoc* *matrix_mul_rid*)
ultimately show *?thesis*
using *invertible_def* **by** *blast*

qed

lemma *transpose_invertible*:

fixes $A :: \text{real}^{n \times n}$
assumes $\text{invertible } A$
shows $\text{invertible } (\text{transpose } A)$
by (meson *assms* *invertible_def* *matrix_left_right_inverse* *right_invertible_transpose*)

lemma *vector_matrix_mul_assoc*:

fixes $v :: ('a::\text{comm_semiring_1})^n$
shows $(v v * M) v * N = v v * (M ** N)$
by (metis (no_types, opaque_lifting) *matrix_transpose_mul* *matrix_vector_mul_assoc* *transpose_matrix_vector*)

lemma *matrix_scaleR_vector_ac*:

fixes $A :: \text{real}^{(m::\text{finite}) \times n}$
shows $A * v (k *_R v) = k *_R A * v v$
by (metis *matrix_vector_mult_scaleR* *transpose_scalar* *vector_scaleR_matrix_ac* *vector_transpose_matrix*)

lemma *scaleR_matrix_vector_assoc*:

fixes $A :: \text{real}^{(m::\text{finite}) \times n}$
shows $k *_R (A * v v) = k *_R A * v v$
by (metis *matrix_scaleR_vector_ac* *matrix_vector_mult_scaleR*)

1.11.2 Some interesting theorems and interpretations

```

locale linear_first_finite_dimensional_vector_space =
  l?: Vector_Spaces.linear scaleB scaleC f +
  B?: finite_dimensional_vector_space scaleB BasisB
  for scaleB :: ('a::field => 'b::ab_group_add => 'b) (infixr *b 75)
  and scaleC :: ('a => 'c::ab_group_add => 'c) (infixr *c 75)
  and BasisB :: ('b set)
  and f :: ('b=>'c)

```

```

lemma vec_dim_card: vec.dim (UNIV::('a::{field}^n) set) = CARD ('n)
  by (simp add: card_cart_basis)

```

```

interpretation vector_space_over_itself: vector_space (*) :: 'a::field => 'a => 'a
  by unfold_locales (simp_all add: algebra_simps)

```

```

lemmas [simp del] = vector_space_over_itself.scale_scale

```

```

interpretation vector_space_over_itself: finite_dimensional_vector_space
  (*) :: 'a::field => 'a => 'a {1}
  by unfold_locales (auto simp: vector_space_over_itself.span_singleton)

```

```

lemma dimension_eq_1[code_unfold]: vector_space_over_itself.dimension TYPE('a::field)=
  1
  unfolding vector_space_over_itself.dimension_def by simp

```

```

lemma dim_subset_UNIV_cart_gen:
  fixes S :: ('a::field^n) set
  shows vec.dim S ≤ CARD('n)
  by (metis vec.dim_eq_full vec.dim_subset_UNIV vec.span_UNIV vec.dim_card)

```

```

lemma dim_subset_UNIV_cart:
  fixes S :: (real^n) set
  shows dim S ≤ CARD('n)
  using dim_subset_UNIV_cart_gen[of S] by (simp add: dim_vec_eq)

```

Two sometimes fruitful ways of looking at matrix-vector multiplication.

```

lemma matrix_mult_dot: A *v x = (χ i. inner (A$i) x)
  by (simp add: matrix_vector_mult_def inner_vec_def)

```

```

lemma adjoint_matrix: adjoint(λx. (A::real^n^m) *v x) = (λx. transpose A *v
  x)
  by (metis adjoint_unique dot_lmul_matrix vector_transpose_matrix)

```

```

lemma matrix_adjoint:
  assumes lf: linear (f :: real^n => real ^m)
  shows matrix(adjoint f) = transpose(matrix f)
  by (metis adjoint_matrix assms_matrix_of_matrix_vector_mul matrix_vector_mul(2))

```

1.11.3 Rank of a matrix

Equivalence of row and column rank is taken from George Mackiw's paper, Mathematics Magazine 1995, p. 285.

lemma *matrix_vector_mult_in_columnspace_gen*:

fixes $A :: 'a::field^{n \times m}$

shows $(A * v \ x) \in \text{vec.span}(\text{columns } A)$

unfolding *columns_def*

by (*metis (mono_tags, lifting) matrix_mult_sum mem_Collect_eq vec.span_base vec.span_scale vec.span_sum*)

lemma *matrix_vector_mult_in_columnspace*:

fixes $A :: \text{real}^{n \times m}$

shows $(A * v \ x) \in \text{span}(\text{columns } A)$

using *matrix_vector_mult_in_columnspace_gen* [of $A \ x$] **by** (*simp add: span_vec_eq*)

lemma *subspace_orthogonal_to_vector*: *subspace* $\{y. \text{orthogonal } x \ y\}$

by (*simp add: subspace_def orthogonal_clauses*)

lemma *orthogonal_nullspace_rowspace*:

fixes $A :: \text{real}^{n \times m}$

assumes $0: A * v \ x = 0$ **and** $y: y \in \text{span}(\text{rows } A)$

shows *orthogonal* $x \ y$

using y

proof (*induction rule: span_induct*)

case *base*

then show *?case*

by (*simp add: subspace_orthogonal_to_vector*)

next

case (*step v*)

then obtain i **where** $v = \text{row } i \ A$

by (*auto simp: rows_def*)

with 0 **show** *?case*

unfolding *orthogonal_def inner_vec_def matrix_vector_mult_def row_def*

by (*simp add: mult.commute*) (*metis (no_types) vec_lambda_beta zero_index*)

qed

lemma *nullspace_inter_rowspace*:

fixes $A :: \text{real}^{n \times m}$

shows $A * v \ x = 0 \wedge x \in \text{span}(\text{rows } A) \iff x = 0$

using *orthogonal_nullspace_rowspace orthogonal_self span_zero matrix_vector_mult_0_right*

by *blast*

lemma *matrix_vector_mul_injective_on_rowpace*:

fixes $A :: \text{real}^{n \times m}$

shows $\llbracket A * v \ x = A * v \ y; x \in \text{span}(\text{rows } A); y \in \text{span}(\text{rows } A) \rrbracket \implies x = y$

using *nullspace_inter_rowpace* [of $A \ x \ y$]

by (*metis diff_eq_diff_eq diff_self matrix_vector_mult_diff_distrib span_diff*)

definition *rank* :: 'a::fieldⁿ^m=>nat
where *row_rank_def_gen*: $\text{rank } A \equiv \text{vec.dim}(\text{rows } A)$

lemma *row_rank_def*: $\text{rank } A = \text{dim}(\text{rows } A)$ **for** $A::\text{real}^n{}^m$
by (*auto simp: row_rank_def_gen dim_vec_eq*)

lemma *dim_rows_le_dim_columns*:

fixes $A :: \text{real}^n{}^m$

shows $\text{dim}(\text{rows } A) \leq \text{dim}(\text{columns } A)$

proof –

have $\text{dim}(\text{span}(\text{rows } A)) \leq \text{dim}(\text{span}(\text{columns } A))$

proof –

obtain B **where** *independent* B $\text{span}(\text{rows } A) \subseteq \text{span } B$

and $B: B \subseteq \text{span}(\text{rows } A)$ $\text{card } B = \text{dim}(\text{span}(\text{rows } A))$

using *basis_exists* [of $\text{span}(\text{rows } A)$] **by** *metis*

with *span_subspace* **have** $\text{eq}: \text{span } B = \text{span}(\text{rows } A)$

by *auto*

then have *inj*: *inj_on* $((*) A)$ $(\text{span } B)$

by (*simp add: inj_on_def matrix_vector_mul_injective_on_rowspace*)

then have *ind*: *independent* $((*) A \text{ ' } B)$

by (*rule linear_independent_injective_image* [*OF Finite_Cartesian_Product.matrix_vector_mult_linear_independent B*])

have $\text{dim}(\text{span}(\text{rows } A)) \leq \text{card}((*) A \text{ ' } B)$

by (*metis B(2) card_image inj inj_on_subset order_refl span_superset*)

also have $\dots \leq \text{dim}(\text{span}(\text{columns } A))$

using *ind*

by (*rule independent_card_le_dim*) (*auto intro!: matrix_vector_mult_in_columnspace*)

finally show *?thesis* .

qed

then show *?thesis*

by (*simp*)

qed

lemma *column_rank_def*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A = \text{dim}(\text{columns } A)$

unfolding *row_rank_def*

by (*metis columns_transpose dim_rows_le_dim_columns le_antisym rows_transpose*)

lemma *rank_transpose*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank}(\text{transpose } A) = \text{rank } A$

by (*metis column_rank_def row_rank_def rows_transpose*)

lemma *matrix_vector_mult_basis*:

fixes $A :: \text{real}^n{}^m$

shows $A * v$ (*axis* k 1) = *column* k A

by (*simp add: cart_eq_inner_axis column_def matrix_mult_dot*)

lemma *columns_image_basis*:

fixes $A :: \text{real}^n{}^m$

shows $\text{columns } A = (*v) A \text{ ' (range } (\lambda i. \text{axis } i \ 1))$

by (*force simp: columns_def matrix_vector_mult_basis [symmetric]*)

lemma *rank_dim_range*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A = \text{dim}(\text{range } (\lambda x. A *v x))$

unfolding *column_rank_def*

by (*smt (verit, best) columns_image_basis dim_span_image_subset_iff iso_tuple_UNIV_I matrix_vector_mult_in_columnspace span_eq*)

lemma *rank_bound*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A \leq \min \text{CARD}'m) (\text{CARD}'n)$

by (*metis (mono_tags, lifting) dim_subset_UNIV_cart min.bounded_iff column_rank_def row_rank_def*)

lemma *full_rank_injective*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A = \text{CARD}'n) \longleftrightarrow \text{inj } ((*v) A)$

by (*simp add: matrix_left_invertible_injective [symmetric] matrix_left_invertible_span_rows row_rank_def*

dim_eq_full [symmetric] card_cart_basis vec.dimension_def)

lemma *full_rank_surjective*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A = \text{CARD}'m) \longleftrightarrow \text{surj } ((*v) A)$

by (*metis (no_types, opaque_lifting) dim_eq_full dim_vec_eq rank_dim_range span_vec_eq vec.span_UNIV vec.span_image vec_dim_card*)

lemma *rank_I*: $\text{rank}(\text{mat } 1 :: \text{real}^n{}^n) = \text{CARD}'n)$

by (*simp add: full_rank_injective inj_on_def*)

lemma *less_rank_noninjective*:

fixes $A :: \text{real}^n{}^m$

shows $\text{rank } A < \text{CARD}'n) \longleftrightarrow \neg \text{inj } ((*v) A)$

using *less_le rank_bound* **by** (*auto simp: full_rank_injective [symmetric]*)

lemma *matrix_nonfull_linear_equations_eq*:

fixes $A :: \text{real}^n{}^m$

shows $(\exists x. (x \neq 0) \wedge A *v x = 0) \longleftrightarrow \text{rank } A \neq \text{CARD}'n)$

by (*meson matrix_left_invertible_injective full_rank_injective matrix_left_invertible_ker*)

lemma *rank_eq_0*: $\text{rank } A = 0 \longleftrightarrow A = 0$ **and** *rank_0* [*simp*]: $\text{rank } (0 :: \text{real}^n{}^m) = 0$

for $A :: \text{real}^n{}^m$

by (*auto simp: rank_dim_range matrix_eq*)

```

lemma rank_mul_le_right:
  fixes A :: realnm and B :: realpn
  shows rank(A ** B) ≤ rank B
proof -
  have rank(A ** B) ≤ dim ((*v) A ' range ((*v) B))
  by (auto simp: rank_dim_range image_comp o_def matrix_vector_mul_assoc)
  also have ... ≤ rank B
  by (simp add: rank_dim_range dim_image_le)
  finally show ?thesis .
qed

```

```

lemma rank_mul_le_left:
  fixes A :: realnm and B :: realpn
  shows rank(A ** B) ≤ rank A
  by (metis matrix_transpose_mul rank_mul_le_right rank_transpose)

```

1.11.4 Lemmas for working on $\text{real}^{1/2/3/4}$

```

lemma exhaust_2:
  fixes x :: 2
  shows x = 1 ∨ x = 2
proof (induct x)
  case (of_int z)
  then have z = 0 | z = 1
  by fastforce
  then show ?case
  by auto
qed

```

```

lemma forall_2: (∀ i::2. P i) ↔ P 1 ∧ P 2
  by (metis exhaust_2)

```

```

lemma exhaust_3:
  fixes x :: 3
  shows x = 1 ∨ x = 2 ∨ x = 3
proof (induct x)
  case (of_int z)
  then have z = 0 ∨ z = 1 ∨ z = 2 by fastforce
  then show ?case by auto
qed

```

```

lemma forall_3: (∀ i::3. P i) ↔ P 1 ∧ P 2 ∧ P 3
  by (metis exhaust_3)

```

```

lemma exhaust_4:
  fixes x :: 4
  shows x = 1 ∨ x = 2 ∨ x = 3 ∨ x = 4
proof (induct x)
  case (of_int z)

```

then have $z = 0 \vee z = 1 \vee z = 2 \vee z = 3$ **by** *fastforce*
then show *?case* **by** *auto*
qed

lemma *forall_4*: $(\forall i::4. P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3 \wedge P\ 4$
by (*metis exhaust_4*)

lemma *UNIV_1* [*simp*]: $UNIV = \{1::1\}$
by (*auto simp add: num1_eq_iff*)

lemma *UNIV_2*: $UNIV = \{1::2, 2::2\}$
using *exhaust_2* **by** *auto*

lemma *UNIV_3*: $UNIV = \{1::3, 2::3, 3::3\}$
using *exhaust_3* **by** *auto*

lemma *UNIV_4*: $UNIV = \{1::4, 2::4, 3::4, 4::4\}$
using *exhaust_4* **by** *auto*

lemma *sum_1*: $sum\ f\ (UNIV::1\ set) = f\ 1$
unfolding *UNIV_1* **by** *simp*

lemma *sum_2*: $sum\ f\ (UNIV::2\ set) = f\ 1 + f\ 2$
unfolding *UNIV_2* **by** *simp*

lemma *sum_3*: $sum\ f\ (UNIV::3\ set) = f\ 1 + f\ 2 + f\ 3$
unfolding *UNIV_3* **by** (*simp add: ac_simps*)

lemma *sum_4*: $sum\ f\ (UNIV::4\ set) = f\ 1 + f\ 2 + f\ 3 + f\ 4$
unfolding *UNIV_4* **by** (*simp add: ac_simps*)

1.11.5 The collapse of the general concepts to dimension one

lemma *vector_one*: $(x::'a\ \hat{1}) = (\chi\ i. (x\$1))$
by (*simp add: vec_eq_iff*)

lemma *forall_one*: $(\forall (x::'a\ \hat{1}). P\ x) \longleftrightarrow (\forall x. P(\chi\ i. x))$
by (*metis vector_one*)

lemma *norm_vector_1*: $norm\ (x::_ \hat{1}) = norm\ (x\$1)$
by (*simp add: norm_vec_def*)

lemma *dist_vector_1*:
fixes $x :: 'a::real_normed_vector\ \hat{1}$
shows $dist\ x\ y = dist\ (x\$1)\ (y\$1)$
by (*simp add: dist_norm norm_vector_1*)

lemma *norm_real*: $norm(x::real\ \hat{1}) = |x\$1|$
by (*simp add: norm_vector_1*)

lemma *dist_real*: $\text{dist}(x::\text{real} \hat{1}) y = |(x\$1) - (y\$1)|$
by (*auto simp add: norm_real dist_norm*)

1.11.6 Routine results connecting the types (*real*, 1) *vec* and *real*

lemma *vector_one_nth* [*simp*]:
fixes $x :: 'a \hat{1}$ **shows** $\text{vec } (x \$ 1) = x$
by (*metis vec_def vector_one*)

lemma *tendsto_at_within_vector_1*:
fixes $S :: 'a :: \text{metric_space set}$
assumes $(f \longrightarrow fx) \text{ (at } x \text{ within } S)$
shows $((\lambda y::'a \hat{1}. \chi \ i. f (y \$ 1)) \longrightarrow (\text{vec } fx::'a \hat{1})) \text{ (at } (\text{vec } x) \text{ within } \text{vec } ' S)$

proof (*rule topological_tendstoI*)

fix $T :: ('a \hat{1}) \text{ set}$

assume $\text{open } T \ \text{vec } fx \in T$

have $\forall_F x \text{ in at } x \text{ within } S. f x \in (\lambda x. x \$ 1) ' T$

using $\langle \text{open } T \rangle \langle \text{vec } fx \in T \rangle$ *assms open_image_vec_nth tendsto_def* **by**
fastforce

then show $\forall_F x::'a \hat{1} \text{ in at } (\text{vec } x) \text{ within } \text{vec } ' S. (\chi \ i. f (x \$ 1)) \in T$

unfolding *eventually_at dist_norm* [*symmetric*]

by (*rule ex_forward*)

(use $\langle \text{open } T \rangle$ **in**

$\langle \text{fastforce simp: dist_norm dist_vec_def L2_set_def image_iff vector_one open_vec_def} \rangle$)

qed

lemma *has_derivative_vector_1*:

assumes *der_g*: $(g \text{ has_derivative } (\lambda x. x * g' a)) \text{ (at } a \text{ within } S)$

shows $((\lambda x. \text{vec } (g (x \$ 1))) \text{ has_derivative } (*_R) (g' a))$

$\text{(at } ((\text{vec } a)::\text{real} \hat{1}) \text{ within } \text{vec } ' S)$

using *der_g*

apply (*clarsimp simp: Deriv.has_derivative_within bounded_linear_scaleR_right norm_vector_1*)

apply (*drule tendsto_at_within_vector_1, vector*)

apply (*auto simp: algebra_simps eventually_at tendsto_def*)

done

1.11.7 Explicit vector construction from lists

definition *vector* $l = (\chi \ i. \text{foldr } (\lambda x f n. \text{fun_upd } (f (n+1)) n x) l (\lambda n x. 0) 1 \ i)$

lemma *vector_1* [*simp*]: $(\text{vector}[x]) \$1 = x$

unfolding *vector_def* **by** *simp*

lemma *vector_2* [*simp*]: $(\text{vector}[x,y]) \$1 = x \ (\text{vector}[x,y] :: 'a \hat{2}) \$2 = (y::'a::\text{zero})$

unfolding *vector_def* **by** *simp_all*

lemma *vector_3* [*simp*]:
 $(\text{vector } [x,y,z] :: ('a::\text{zero})^3)\$1 = x$
 $(\text{vector } [x,y,z] :: ('a::\text{zero})^3)\$2 = y$
 $(\text{vector } [x,y,z] :: ('a::\text{zero})^3)\$3 = z$
unfolding *vector_def* **by** *simp_all*

lemma *forall_vector_1*: $(\forall v::'a::\text{zero}^1. P v) \longleftrightarrow (\forall x. P(\text{vector}[x]))$
by (*metis vector_1 vector_one*)

lemma *forall_vector_2*: $(\forall v::'a::\text{zero}^2. P v) \longleftrightarrow (\forall x y. P(\text{vector}[x, y]))$
proof –
have $P v$ **if** $\bigwedge x y. P(\text{vector } [x, y])$ **for** v
proof –
have $\text{vector } [v\$1, v\$2] = v$
by (*smt (verit, best) exhaust_2 vec_eq_iff vector_2*)
then show *?thesis*
by (*metis that*)
qed
then show *?thesis* **by** *auto*
qed

lemma *forall_vector_3*: $(\forall v::'a::\text{zero}^3. P v) \longleftrightarrow (\forall x y z. P(\text{vector}[x, y, z]))$
proof –
have $P v$ **if** $\bigwedge x y z. P(\text{vector } [x, y, z])$ **for** v
proof –
have $\text{vector } [v\$1, v\$2, v\$3] = v$
by (*smt (verit, best) exhaust_3 vec_eq_iff vector_3*)
then show *?thesis*
by (*metis that*)
qed
then show *?thesis* **by** *auto*
qed

1.11.8 lambda skolemization on cartesian products

lemma *lambda_skolem*: $(\forall i. \exists x. P i x) \longleftrightarrow (\exists x::'a \wedge 'n. \forall i. P i (x \$ i))$
by (*metis vec_lambda_beta*)

The same result in terms of square matrices.

Considering an n-element vector as an n-by-1 or 1-by-n matrix.

definition *rowvector* $v = (\chi \ i \ j. (v\$j))$

definition *columnvector* $v = (\chi \ i \ j. (v\$i))$

lemma *transpose_columnvector*: $\text{transpose}(\text{columnvector } v) = \text{rowvector } v$
by (*simp add: transpose_def rowvector_def columnvector_def vec_eq_iff*)

lemma *transpose_rowvector*: $\text{transpose}(\text{rowvector } v) = \text{columnvector } v$
 by (*simp add: transpose_def columnvector_def rowvector_def vec_eq_iff*)

lemma *dot_rowvector_columnvector*: $\text{columnvector } (A * v) = A ** \text{columnvector } v$
 by (*vector columnvector_def matrix_matrix_mult_def matrix_vector_mult_def*)

lemma *dot_matrix_product*:
 $(x :: \text{real}^n) \cdot y = (((\text{rowvector } x :: \text{real}^{n^1}) ** (\text{columnvector } y :: \text{real}^{1^n})) \$1) \$1$
 by (*vector matrix_matrix_mult_def rowvector_def columnvector_def inner_vec_def*)

lemma *dot_matrix_vector_mul*:
 fixes $A B :: \text{real}^{n^1}$ and $x y :: \text{real}^n$
 shows $(A * v x) \cdot (B * v y) =$
 $(((\text{rowvector } x :: \text{real}^{n^1}) ** ((\text{transpose } A ** B) ** (\text{columnvector } y :: \text{real}^{1^n}))) \$1) \$1$
 by (*metis dot_lmul_matrix dot_matrix_product dot_rowvector_columnvector matrix_mul_assoc vector_transpose_matrix*)

lemma *dim_substandard_cart*: $\text{vec.dim } \{x :: 'a :: \text{field}^n. \forall i. i \notin d \longrightarrow x \$ i = 0\} = \text{card } d$

(*is vec.dim ?A = _*)
proof (*rule vec.dim_unique*)
 let $?B = ((\lambda x. \text{axis } x 1) ^ d)$
 have *subset_basis*: $?B \subseteq \text{cart_basis}$
 by (*auto simp: cart_basis_def*)
 show $?B \subseteq ?A$
 by (*auto simp: axis_def*)
 show *vec.independent* $((\lambda x. \text{axis } x 1) ^ d)$
 using *subset_basis*
 by (*rule vec.independent_mono[OF vec.independent_Basis]*)
 have $x \in \text{vec.span } ?B$ if $\forall i. i \notin d \longrightarrow x \$ i = 0$ for $x :: 'a^1$
proof –
 have *finite* $?B$
 using *subset_basis finite_cart_basis*
 by (*rule finite_subset*)
 have $x = (\sum_{i \in \text{UNIV}} x \$ i * s \text{axis } i 1)$
 by (*rule basis_expansion[symmetric]*)
 also have $\dots = (\sum_{i \in d} (x \$ i) * s \text{axis } i 1)$
 by (*rule sum.mono_neutral_cong_right*) (*auto simp: that*)
 also have $\dots \in \text{vec.span } ?B$
 by (*simp add: vec.span_sum vec.span_clauses*)
 finally show $x \in \text{vec.span } ?B$.
qed
 then show $?A \subseteq \text{vec.span } ?B$ by *auto*
qed (*simp add: card_image inj_on_def axis_eq_axis*)

lemma *affinity_inverses*:

```

assumes m0:  $m \neq (0::'a::field)$ 
shows  $(\lambda x. m *s x + c) \circ (\lambda x. inverse(m) *s x + -(inverse(m) *s c)) = id$ 
 $(\lambda x. inverse(m) *s x + -(inverse(m) *s c)) \circ (\lambda x. m *s x + c) = id$ 
using m0
by (auto simp add: fun_eq_iff vector_add_ldistrib diff_conv_add_uminus simp
del: add_uminus_conv_diff)

```

```

lemma vector_affinity_eq:
assumes m0:  $(m::'a::field) \neq 0$ 
shows  $m *s x + c = y \longleftrightarrow x = inverse\ m *s y + -(inverse\ m *s c)$ 
proof
assume h:  $m *s x + c = y$ 
hence  $m *s x = y - c$  by (simp add: field_simps)
hence  $inverse\ m *s (m *s x) = inverse\ m *s (y - c)$  by simp
then show  $x = inverse\ m *s y + -(inverse\ m *s c)$ 
by (simp add: m0 vec.scale_right_diff_distrib)
next
assume h:  $x = inverse\ m *s y + -(inverse\ m *s c)$ 
show  $m *s x + c = y$  unfolding h
using m0 by (simp add: vector_smult_assoc vector_ssub_ldistrib)
qed

```

```

lemma vector_eq_affinity:
 $(m::'a::field) \neq 0 \implies (y = m *s x + c \longleftrightarrow inverse(m) *s y + -(inverse(m) *s c) = x)$ 
by (metis vector_affinity_eq)

```

```

lemma vector_cart:
fixes f ::  $real^{\wedge}n \Rightarrow real$ 
shows  $(\chi\ i. f (axis\ i\ 1)) = (\sum\ i \in Basis. f\ i *_{\mathbb{R}}\ i)$ 
unfolding euclidean_eq_iff [where 'a=real^{\wedge}n]
by simp (simp add: Basis_vec_def inner_axis)

```

```

lemma const_vector_cart:  $((\chi\ i. d)::real^{\wedge}n) = (\sum\ i \in Basis. d *_{\mathbb{R}}\ i)$ 
by (rule vector_cart)

```

1.11.9 Explicit formulas for low dimensions

```

lemma prod_neutral_const:  $prod\ f\ \{(1::nat)..1\} = f\ 1$ 
by simp

```

```

lemma prod_2:  $prod\ f\ \{(1::nat)..2\} = f\ 1 * f\ 2$ 
by (simp add: eval_nat_numeral atLeastAtMostSuc_conv mult.commute)

```

```

lemma prod_3:  $prod\ f\ \{(1::nat)..3\} = f\ 1 * f\ 2 * f\ 3$ 
by (simp add: eval_nat_numeral atLeastAtMostSuc_conv mult.commute)

```

1.11.10 Orthogonality of a matrix

```

definition orthogonal_matrix  $(Q::'a::semiring_1^{\wedge}n^{\wedge}n) \longleftrightarrow$ 

```

$\text{transpose } Q ** Q = \text{mat } 1 \wedge Q ** \text{transpose } Q = \text{mat } 1$

lemma *orthogonal_matrix*: $\text{orthogonal_matrix } (Q :: \text{real } ^n ^n) \longleftrightarrow \text{transpose } Q ** Q = \text{mat } 1$
by (*metis matrix_left_right_inverse orthogonal_matrix_def*)

lemma *orthogonal_matrix_id*: $\text{orthogonal_matrix } (\text{mat } 1 :: \text{real } ^n ^n)$
by (*simp add: orthogonal_matrix_def*)

proposition *orthogonal_matrix_mul*:
fixes $A :: \text{real } ^n ^n$
assumes *orthogonal_matrix* A *orthogonal_matrix* B
shows $\text{orthogonal_matrix}(A ** B)$
using *assms*
by (*simp add: orthogonal_matrix_matrix_transpose_mul matrix_left_right_inverse matrix_mul_assoc*)

proposition *orthogonal_transformation_matrix*:
fixes $f :: \text{real } ^n \Rightarrow \text{real } ^n$
shows $\text{orthogonal_transformation } f \longleftrightarrow \text{linear } f \wedge \text{orthogonal_matrix}(\text{matrix } f)$
(is ?lhs \longleftrightarrow ?rhs)

proof –
let $?mf = \text{matrix } f$
let $?ot = \text{orthogonal_transformation } f$
let $?U = \text{UNIV} :: 'n \text{ set}$
have fU : *finite* $?U$ **by** *simp*
let $?m1 = \text{mat } 1 :: \text{real } ^n ^n$
{
assume ot : $?ot$
from ot **have** lf : *Vector_Spaces.linear* $(*)$ $(*)$ f **and** fd : $\bigwedge v w. f v \cdot f w = v \cdot w$
unfolding *orthogonal_transformation_def orthogonal_matrix_linear_def scalar_mult_eq_scaleR*
by *blast+*
{
fix $i j$
let $?A = \text{transpose } ?mf ** ?mf$
have $th0$: $\bigwedge b (x :: 'a :: \text{comm_ring}_1). (\text{if } b \text{ then } 1 \text{ else } 0) * x = (\text{if } b \text{ then } x \text{ else } 0)$
 $\bigwedge b (x :: 'a :: \text{comm_ring}_1). x * (\text{if } b \text{ then } 1 \text{ else } 0) = (\text{if } b \text{ then } x \text{ else } 0)$
by *simp_all*
from fd [*of axis i 1 axis j 1, simplified matrix_works[OF lf, symmetric] dot_matrix_vector_mul*]
have $?A \$ i \$ j = ?m1 \$ i \$ j$
by (*simp add: inner_vec_def matrix_matrix_mult_def columnvector_def rowvector_def th0 sum.delta[OF fU] mat_def axis_def*)
}
then have *orthogonal_matrix* $?mf$

```

    unfolding orthogonal_matrix
  by vector
with lf have ?rhs
  unfolding linear_def scalar_mult_eq_scaleR
  by blast
}
moreover
have ?lhs if Vector_Spaces.linear (*s) (*s) f and orthogonal_matrix ?mf
  using that unfolding orthogonal_matrix_def norm_eq orthogonal_transformation
  by (metis dot_matrix_product dot_matrix_vector_mul linear_matrix_vector_mul_eq
matrix_mul_lid matrix_vector_mul(2))
ultimately show ?thesis
  by (auto simp: linear_def scalar_mult_eq_scaleR)
qed

```

1.11.11 Finding an Orthogonal Matrix

We can find an orthogonal matrix taking any unit vector to any other.

```

lemma orthogonal_matrix_transpose [simp]:
  orthogonal_matrix(transpose A)  $\longleftrightarrow$  orthogonal_matrix A
  by (auto simp: orthogonal_matrix_def)

```

```

lemma orthogonal_matrix_orthonormal_columns:
  fixes A :: realn
  shows orthogonal_matrix A  $\longleftrightarrow$ 
    ( $\forall i. \text{norm}(\text{column } i \ A) = 1$ )  $\wedge$ 
    ( $\forall i \ j. i \neq j \longrightarrow \text{orthogonal}(\text{column } i \ A) (\text{column } j \ A)$ )
  by (auto simp: orthogonal_matrix matrix_mult_transpose_dot_column vec_eq_iff
mat_def norm_eq_1 orthogonal_def)

```

```

lemma orthogonal_matrix_orthonormal_rows:
  fixes A :: realn
  shows orthogonal_matrix A  $\longleftrightarrow$ 
    ( $\forall i. \text{norm}(\text{row } i \ A) = 1$ )  $\wedge$ 
    ( $\forall i \ j. i \neq j \longrightarrow \text{orthogonal}(\text{row } i \ A) (\text{row } j \ A)$ )
  using orthogonal_matrix_orthonormal_columns [of transpose A] by simp

```

```

proposition orthogonal_matrix_exists_basis:
  fixes a :: realn
  assumes norm a = 1
  obtains A where orthogonal_matrix A A * v (axis k 1) = a
proof -
  obtain S where a  $\in$  S pairwise_orthogonal S and noS:  $\bigwedge x. x \in S \implies \text{norm } x = 1$ 
  and independent S card S = CARD('n) span S = UNIV
  using vector_in_orthonormal_basis assms by force
  then obtain f0 where bij_betw f0 (UNIV::'n set) S
  by (metis finite_class.finite_UNIV finite_same_card_bij finiteI_independent)
  then obtain f where f: bij_betw f (UNIV::'n set) S and a: a = f k

```

```

    using bij_swap_iff [of f0 k inv f0 a]
  by (metis UNIV_I <a ∈ S> bij_betw_inv_into_right bij_betw_swap_iff swap_apply(1))
show thesis
proof
  have [simp]:  $\bigwedge i. \text{norm } (f\ i) = 1$ 
    using bij_betwE [OF <bij_betw f UNIV S>] by (blast intro: noS)
  have [simp]:  $\bigwedge i\ j. i \neq j \implies \text{orthogonal } (f\ i) (f\ j)$ 
    using <pairwise orthogonal S> <bij_betw f UNIV S>
    by (auto simp: pairwise_def bij_betw_def inj_on_def)
  show orthogonal_matrix ( $\chi\ i\ j. f\ j\ \$\ i$ )
    by (simp add: orthogonal_matrix_orthonormal_columns column_def)
  show  $(\chi\ i\ j. f\ j\ \$\ i) *v\ \text{axis}\ k\ 1 = a$ 
    by (simp add: matrix_vector_mult_def axis_def a_if_distrib cong: if_cong)
qed
qed

```

```

lemma orthogonal_transformation_exists_1:
  fixes a b ::  $\text{real}^n$ 
  assumes  $\text{norm } a = 1$   $\text{norm } b = 1$ 
  obtains f where orthogonal_transformation f f a = b
proof -
  obtain k A B where AB: orthogonal_matrix A orthogonal_matrix B and eq:
A *v (axis k 1) = a B *v (axis k 1) = b
    using orthogonal_matrix_exists_basis assms by metis
  let ?f =  $\lambda x. (B ** \text{transpose } A) *v\ x$ 
  show thesis
  proof
    show orthogonal_transformation ?f
      by (simp add: AB orthogonal_matrix_mul orthogonal_transformation_matrix)
  next
    show ?f a = b
      using <orthogonal_matrix A> unfolding orthogonal_matrix_def
      by (metis eq matrix_mul_rid matrix_vector_mul_assoc)
  qed
qed

```

```

proposition orthogonal_transformation_exists:
  fixes a b ::  $\text{real}^n$ 
  assumes  $\text{norm } a = \text{norm } b$ 
  obtains f where orthogonal_transformation f f a = b
proof (cases a = 0 ∨ b = 0)
  case True
  with assms show ?thesis
    using that by force
  next
  case False
  then obtain f where f: orthogonal_transformation f and eq: f (a /R norm a)
  =  $(b /_R \text{norm } b)$ 
    by (auto intro: orthogonal_transformation_exists_1 [of a /R norm a b /R norm

```

```

b)
  show ?thesis
    using False assms eq f orthogonal_transformation_scaleR that by fastforce
qed

```

1.11.12 Scaling and isometry

proposition *scaling_linear*:

```

fixes f :: 'a::real_inner ⇒ 'a::real_inner
assumes f0: f 0 = 0
  and fd: ∀ x y. dist (f x) (f y) = c * dist x y
shows linear f
proof -
  {
    fix v w
    have norm (f x) = c * norm x for x
      by (metis dist_0_norm f0 fd)
    then have f v · f w = c2 * (v · w)
      unfolding dot_norm_neg dist_norm[symmetric]
      by (simp add: fd power2_eq_square field_simps)
  }
  then show ?thesis
    unfolding linear_iff vector_eq[where 'a='a] scalar_mult_eq_scaleR
    by (simp add: inner_add field_simps)
qed

```

lemma *isometry_linear*:

```

f (0::'a::real_inner) = (0::'a) ⇒ ∀ x y. dist(f x) (f y) = dist x y ⇒ linear f
by (rule scaling_linear[where c=1]) simp_all

```

Hence another formulation of orthogonal transformation

proposition *orthogonal_transformation_isometry*:

```

orthogonal_transformation f ⇔ f(0::'a::real_inner) = (0::'a) ∧ (∀ x y. dist(f
x) (f y) = dist x y)
unfolding orthogonal_transformation
by (metis dist_0_norm dist_norm isometry_linear linear_0 linear_diff)

```

Can extend an isometry from unit sphere:

lemma *isometry_sphere_extend*:

```

fixes f :: 'a::real_inner ⇒ 'a
assumes f1: ∀ x. norm x = 1 ⇒ norm (f x) = 1
  and fd1: ∀ x y. [norm x = 1; norm y = 1] ⇒ dist (f x) (f y) = dist x y
shows ∃ g. orthogonal_transformation g ∧ (∀ x. norm x = 1 → g x = f x)
proof -
  {
    fix x y x' y' u v u' v' :: 'a
    assume H: x = norm x *R u y = norm y *R v
      x' = norm x *R u' y' = norm y *R v'

```

```

    and J: norm u = 1 norm u' = 1 norm v = 1 norm v' = 1 norm(u' - v') =
norm(u - v)
    then have *: u · v = u' · v' + v' · u' - v · u
    by (simp add: norm_eq norm_eq_1 inner_add inner_diff)
    have norm (norm x *R u' - norm y *R v') = norm (norm x *R u - norm y
*R v)
    using J by (simp add: norm_eq norm_eq_1 inner_diff * field_simps)
    then have norm(x' - y') = norm(x - y)
    using H by metis
  }
note norm_eq = this
let ?g = λx. if x = 0 then 0 else norm x *R f (x /R norm x)
have thfg: ?g x = f x if norm x = 1 for x
  using that by auto
have thd: dist (?g x) (?g y) = dist x y for x y
proof (cases x=0 ∨ y=0)
  case False
  show dist (?g x) (?g y) = dist x y
  unfolding dist_norm
  proof (rule norm_eq)
    show x = norm x *R (x /R norm x) y = norm y *R (y /R norm y)
      norm (f (x /R norm x)) = 1 norm (f (y /R norm y)) = 1
    using False f1 by auto
  qed (use False in ⟨auto simp: field_simps intro: f1 fd1[unfolded dist_norm]⟩)
qed (auto simp: f1)
show ?thesis
  unfolding orthogonal_transformation_isometry
  by (rule exI[where x= ?g]) (metis thfg thd)
qed

```

1.11.13 Induction on matrix row operations

lemma induct_matrix_row_operations:

fixes $P :: \text{real}^n \Rightarrow \text{bool}$

assumes zero_row: $\bigwedge A i. \text{row } i \ A = 0 \implies P \ A$

and diagonal: $\bigwedge A. (\bigwedge i \ j. i \neq j \implies A[i][j] = 0) \implies P \ A$

and swap_cols: $\bigwedge A \ m \ n. \llbracket P \ A; \ m \neq n \rrbracket \implies P(\chi \ i \ j. A \ \$ \ i \ \$ \ \text{Transposition.transpose } m \ n \ j)$

and row_op: $\bigwedge A \ m \ n \ c. \llbracket P \ A; \ m \neq n \rrbracket$

$\implies P(\chi \ i. \text{if } i = m \text{ then row } m \ A + c \ *_{\mathbb{R}} \ \text{row } n \ A \ \text{else row } i \ A)$

shows $P \ A$

proof –

have $P \ A$ if $(\bigwedge i \ j. \llbracket j \in -K; \ i \neq j \rrbracket \implies A[i][j] = 0)$ for $A \ K$

proof –

have finite K

by simp

then show ?thesis using that

proof (induction arbitrary: A rule: finite_induct)

case empty


```

    with diagonal show ?case
      by simp
  next
  case (insert k K)
  note insertK = insert
  have P A if kk: A $ k $ k  $\neq$  0
    and 0:  $\bigwedge i j. \llbracket j \in - \text{insert } k \text{ } K; i \neq j \rrbracket \implies A \$ i \$ j = 0$ 
       $\bigwedge i. \llbracket i \in -L; i \neq k \rrbracket \implies A \$ i \$ k = 0$  for A L
  proof -
    have finite L
      by simp
    then show ?thesis using 0 kk
    proof (induction arbitrary: A rule: finite_induct)
      case (empty B)
      show ?case
      proof (rule insertK)
        fix i j
        assume  $i \in - K$   $j \neq i$ 
        show B $ j $ i = 0
          using  $\langle j \neq i \rangle \langle i \in - K \rangle$  empty
        by (metis ComplD ComplI Compl_eq_Diff_UNIV Diff_empty UNIV_I
insert_iff)
      qed
    next
    case (insert l L B)
    show ?case
    proof (cases k = l)
      case True
      with insert show ?thesis
      by auto
    next
    case False
    let ?C =  $\chi$  i. if  $i = l$  then row l B - (B $ l $ k / B $ k $ k) *R row k
      B else row i B
    have 1:  $\llbracket j \in - \text{insert } k \text{ } K; i \neq j \rrbracket \implies ?C \$ i \$ j = 0$  for j i
      by (auto simp: insert.premis(1) row_def)
    have 2: ?C $ i $ k = 0
      if  $i \in - L$   $i \neq k$  for i
    proof (cases i=l)
      case True
      with that insert.premis show ?thesis
      by (simp add: row_def)
    next
    case False
    with that show ?thesis
      by (simp add: insert.premis(2) row_def)
    qed
    have 3: ?C $ k $ k  $\neq$  0
      by (auto simp: insert.premis row_def  $\langle k \neq l \rangle$ )

```

```

      have PC: P ?C
      using insert.IH [OF 1 2 3] by auto
      have eqB: (χ i. if i = l then row l ?C + (B $ l $ k / B $ k $ k) *R row
k ?C else row i ?C) = B
      using ⟨k ≠ l⟩ by (simp add: vec_eq_iff row_def)
      show ?thesis
      using row_op [OF PC, of l k, where c = B $ l $ k / B $ k $ k] eqB ⟨k ≠ l⟩
      by (simp add: cong: if_cong)
    qed
  qed
  then have nonzero_hyp: P A
  if kk: A $ k $ k ≠ 0 and zeroes: ∧ i j. j ∈ - insert k K ∧ i ≠ j ⇒ A $ i $ j = 0
for A
  by (auto simp: intro!: kk zeroes)
  show ?case
  proof (cases row k A = 0)
    case True
    with zero_row show ?thesis by auto
  next
  case False
  then obtain l where l: A $ k $ l ≠ 0
  by (auto simp: row_def zero_vec_def vec_eq_iff)
  show ?thesis
  proof (cases k = l)
    case True
    with l nonzero_hyp insert.prem1 show ?thesis
    by blast
  next
  case False
  have *: A $ i $ Transposition.transpose k l j = 0 if j ≠ k j ∉ K i ≠ j for
i j
  using False l insert.prem1 that
  by (auto simp add: Transposition.transpose_def)
  have P (χ i j. (χ i j. A $ i $ Transposition.transpose k l j) $ i $
Transposition.transpose k l j)
  by (rule swap_cols [OF nonzero_hyp False]) (auto simp: l *)
  moreover
  have (χ i j. (χ i j. A $ i $ Transposition.transpose k l j) $ i $ Transposi-
tion.transpose k l j) = A
  by simp
  ultimately show ?thesis
  by simp
  qed
  qed
  qed
  then show ?thesis
  by blast

```

qed

lemma *induct_matrix_elementary*:

```

fixes P :: realn ⇒ bool
assumes mult:  $\bigwedge A B. \llbracket P A; P B \rrbracket \implies P(A ** B)$ 
and zero_row:  $\bigwedge A i. \text{row } i A = 0 \implies P A$ 
and diagonal:  $\bigwedge A. (\bigwedge i j. i \neq j \implies A\$i\$j = 0) \implies P A$ 
and swap1:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m$ 
n j)
and idplus:  $\bigwedge m n c. m \neq n \implies P(\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool}$ 
(i = j))
shows P A
proof -
have swap: P ( $\chi i j. A \$ i \$ \text{Transposition.transpose } m n j$ ) (is P ?C)
if P A m ≠ n for A m n
proof -
have A ** ( $\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m n j$ ) = ?C
by (simp add: matrix_matrix_mult_def mat_def vec_eq_iff if_distrib sum.delta_remove)
then show ?thesis
using mult swap1 that by metis
qed
have row: P ( $\chi i. \text{if } i = m \text{ then row } m A + c *_R \text{ row } n A \text{ else row } i A$ ) (is P
?C)
if P A m ≠ n for A m n c
proof -
let ?B =  $\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool } (i = j)$ 
have ?B ** A = ?C
using <m ≠ n> unfolding matrix_matrix_mult_def row_def of_bool_def
by (auto simp: vec_eq_iff if_distrib [of  $\lambda x. x * y$  for y] sum.remove cong:
if_cong)
then show ?thesis
by (rule subst) (auto simp: that mult idplus)
qed
show ?thesis
by (rule induct_matrix_row_operations [OF zero_row diagonal swap row])
qed

```

lemma *induct_matrix_elementary_alt*:

```

fixes P :: realn ⇒ bool
assumes mult:  $\bigwedge A B. \llbracket P A; P B \rrbracket \implies P(A ** B)$ 
and zero_row:  $\bigwedge A i. \text{row } i A = 0 \implies P A$ 
and diagonal:  $\bigwedge A. (\bigwedge i j. i \neq j \implies A\$i\$j = 0) \implies P A$ 
and swap1:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{mat } 1 \$ i \$ \text{Transposition.transpose } m$ 
n j)
and idplus:  $\bigwedge m n. m \neq n \implies P(\chi i j. \text{of\_bool } (i = m \wedge j = n \vee i = j))$ 
shows P A
proof -
have *: P ( $\chi i j. \text{if } i = m \wedge j = n \text{ then } c \text{ else of\_bool } (i = j)$ )
if m ≠ n for m n c

```

```

proof (cases c = 0)
  case True
  with diagonal show ?thesis by auto
next
  case False
  then have eq: ( $\chi$  i j. if i = m  $\wedge$  j = n then c else of_bool (i = j)) =
    ( $\chi$  i j. if i = j then (if j = n then inverse c else 1) else 0) **
    ( $\chi$  i j. of_bool (i = m  $\wedge$  j = n  $\vee$  i = j)) **
    ( $\chi$  i j. if i = j then if j = n then c else 1 else 0)
    using <m  $\neq$  n>
  apply (simp add: matrix_matrix_mult_def vec_eq_iff of_bool_def if_distrib
[of  $\lambda x. y * x$  for y] cong: if_cong)
  apply (simp add: if_if_eq_conj sum.neutral conj_commute cong: conj_cong)
  done
  show ?thesis
  unfolding eq by (intro mult_idplus that) (auto intro: diagonal)
qed
show ?thesis
  by (rule induct_matrix_elementary) (auto intro: assms *)
qed

```

lemma matrix_vector_mult_matrix_matrix_mult_compose:

```

(*v) (A ** B) = (*v) A  $\circ$  (*v) B
by (auto simp: matrix_vector_mul_assoc)

```

lemma induct_linear_elementary:

```

fixes f :: realn  $\Rightarrow$  realn
assumes linear f
  and comp:  $\bigwedge f g. \llbracket \text{linear } f; \text{linear } g; P f; P g \rrbracket \Longrightarrow P(f \circ g)$ 
  and zeroes:  $\bigwedge f i. \llbracket \text{linear } f; \bigwedge x. (f x) \$ i = 0 \rrbracket \Longrightarrow P f$ 
  and const:  $\bigwedge c. P(\lambda x. \chi i. c i * x \$ i)$ 
  and swap:  $\bigwedge m n::'n. m \neq n \Longrightarrow P(\lambda x. \chi i. x \$ Transposition.transpose m n i)$ 
  and idplus:  $\bigwedge m n::'n. m \neq n \Longrightarrow P(\lambda x. \chi i. \text{if } i = m \text{ then } x \$ m + x \$ n \text{ else } x \$ i)$ 
shows P f
proof -
  have P ((*v) A) for A
  proof (rule induct_matrix_elementary_alt)
    fix A B
    assume P ((*v) A) and P ((*v) B)
    then show P ((*v) (A ** B))
    by (auto simp add: matrix_vector_mult_matrix_matrix_mult_compose intro!:
comp)
  next
  fix A :: realn  $\wedge$  n and i
  assume row i A = 0
  show P ((*v) A)
  using matrix_vector_mul_linear
  by (rule zeroes[where i=i])

```

```

      (metis ‹row i A = 0› inner_zero_left matrix_vector_mul_component
row_def vec_lambda_eta)
    next
      fix A :: real^'n^'n
      assume 0: ‹∧i j. i ≠ j ⇒ A $ i $ j = 0›
      have A $ i $ i * x $ i = (∑j∈UNIV. A $ i $ j * x $ j) for x and i :: 'n
        by (simp add: 0 comm_monoid_add_class.sum.remove [where x=i])
      then have (λx. χ i. A $ i $ i * x $ i) = ((*v) A)
        by (auto simp: 0 matrix_vector_mult_def)
      then show P ((*v) A)
        using const [of λi. A $ i $ i] by simp
    next
      fix m n :: 'n
      assume m ≠ n
      have eq: (∑j∈UNIV. if i = Transposition.transpose m n j then x $ j else 0) =
        (∑j∈UNIV. if j = Transposition.transpose m n i then x $ j else 0)
        for i and x :: real^'n
      by (rule sum.cong) (auto simp add: swap_id_eq)
      have (λx::real^'n. χ i. x $ Transposition.transpose m n i) = ((*v) (χ i j. if i
= Transposition.transpose m n j then 1 else 0))
      by (auto simp: mat_def matrix_vector_mult_def eq_if_distrib [of λx. x * y
for y] cong: if_cong)
      with swap [OF ‹m ≠ n›] show P ((*v) (χ i j. mat 1 $ i $ Transposi-
tion.transpose m n j))
      by (simp add: mat_def matrix_vector_mult_def)
    next
      fix m n :: 'n
      assume m ≠ n
      then have x $ m + x $ n = (∑j∈UNIV. of_bool (j = n ∨ m = j) * x $ j)
for x :: real^'n
      by (auto simp: of_bool_def if_distrib [of λx. x * y for y] sum.remove cong:
if_cong)
      then have (λx::real^'n. χ i. if i = m then x $ m + x $ n else x $ i) =
        ((*v) (χ i j. of_bool (i = m ∧ j = n ∨ i = j)))
      unfolding matrix_vector_mult_def of_bool_def
      by (auto simp: vec_eq_iff if_distrib [of λx. x * y for y] cong: if_cong)
      then show P ((*v) (χ i j. of_bool (i = m ∧ j = n ∨ i = j)))
        using idplus [OF ‹m ≠ n›] by simp
    qed
  then show ?thesis
    by (metis ‹linear f› matrix_vector_mul(2))
  qed
end

```

1.12 Traces and Determinants of Square Matrices

```

theory Determinants
imports

```

HOL-Combinatorics.Permutations
Cartesian_Space

begin

1.12.1 Trace

definition $trace :: 'a::semiring_1 \hat{\ }^n \hat{\ }^n \Rightarrow 'a$
where $trace\ A = sum\ (\lambda i. (A\$i)\$i)\ (UNIV::'n\ set)$

lemma $trace_0: trace\ (mat\ 0) = 0$
by (*simp add: trace_def mat_def*)

lemma $trace_I: trace\ (mat\ 1 :: 'a::semiring_1 \hat{\ }^n \hat{\ }^n) = of_nat(CARD('n))$
by (*simp add: trace_def mat_def*)

lemma $trace_add: trace\ ((A::'a::comm_semiring_1 \hat{\ }^n \hat{\ }^n) + B) = trace\ A + trace\ B$
by (*simp add: trace_def sum.distrib*)

lemma $trace_sub: trace\ ((A::'a::comm_ring_1 \hat{\ }^n \hat{\ }^n) - B) = trace\ A - trace\ B$
by (*simp add: trace_def sum_subtractf*)

lemma $trace_mul_sym: trace\ ((A::'a::comm_semiring_1 \hat{\ }^n \hat{\ }^m) ** B) = trace\ (B**A)$
apply (*simp add: trace_def matrix_matrix_mult_def*)
apply (*subst sum.swap*)
apply (*simp add: mult.commute*)
done

Definition of determinant

definition $det :: 'a::comm_ring_1 \hat{\ }^n \hat{\ }^n \Rightarrow 'a$ **where**
 $det\ A =$
 $sum\ (\lambda p. of_int\ (sign\ p) * prod\ (\lambda i. A\$i\$p\ i)\ (UNIV :: 'n\ set))$
 $\{p. p\ permutes\ (UNIV :: 'n\ set)\}$

Basic determinant properties

lemma $det_transpose$ [*simp*]: $det\ (transpose\ A) = det\ (A::'a::comm_ring_1 \hat{\ }^n \hat{\ }^n)$

proof –

let $?di = \lambda A\ i\ j. A\$i\$j$

let $?U = (UNIV :: 'n\ set)$

have $fU: finite\ ?U$ **by** *simp*

{

fix p

assume $p: p \in \{p. p\ permutes\ ?U\}$

from p **have** $pU: p\ permutes\ ?U$

by *blast*

have $sth: sign\ (inv\ p) = sign\ p$

by (*metis sign_inverse fU p mem_Collect_eq permutation_permutes*)

```

from permutes_inj[OF pU]
have pi: inj_on p ?U
  by (blast intro: subset_inj_on)
from permutes_image[OF pU]
have prod ( $\lambda i. ?di$  (transpose A) i (inv p i)) ?U =
  prod ( $\lambda i. ?di$  (transpose A) i (inv p i)) (p ' ?U)
  by simp
also have ... = prod (( $\lambda i. ?di$  (transpose A) i (inv p i))  $\circ$  p) ?U
  unfolding prod.reindex[OF pi] ..
also have ... = prod ( $\lambda i. ?di$  A i (p i)) ?U
proof -
  have (( $\lambda i. ?di$  (transpose A) i (inv p i))  $\circ$  p) i = ?di A i (p i) if i  $\in$  ?U for i
    using that permutes_inv_o[OF pU] permutes_in_image[OF pU]
    unfolding transpose_def by (simp add: fun_eq_iff)
  then show prod (( $\lambda i. ?di$  (transpose A) i (inv p i))  $\circ$  p) ?U = prod ( $\lambda i. ?di$ 
A i (p i)) ?U
    by (auto intro: prod.cong)
  qed
finally have of_int (sign (inv p)) * (prod ( $\lambda i. ?di$  (transpose A) i (inv p i))
?U) =
  of_int (sign p) * (prod ( $\lambda i. ?di$  A i (p i)) ?U)
  using sth by simp
}
then show ?thesis
  unfolding det_def
  by (subst sum_permutations_inverse) (blast intro: sum.cong)
qed

```

lemma det_lowerdiagonal:

fixes A :: 'a::comm_ring_1[^]($n::\{finite,wellorder\}$)[^]($n::\{finite,wellorder\}$)

assumes ld: $\bigwedge i j. i < j \implies A\$i\$j = 0$

shows det A = prod ($\lambda i. A\$i\i) (UNIV::'n set)

proof -

let ?U = UNIV::'n set

let ?PU = {p. p permutes ?U}

let ?pp = $\lambda p. of_int$ (sign p) * prod ($\lambda i. A\$i\$p i$) (UNIV :: 'n set)

have fU: finite ?U

by simp

have id0: {id} \subseteq ?PU

by (auto simp: permutes_id)

have p0: $\forall p \in ?PU - \{id\}. ?pp p = 0$

proof

fix p

assume p \in ?PU - {id}

then **obtain** i **where** i: p i > i

by clarify (meson leI permutes_natset_le)

from ld[OF i] **have** $\exists i \in ?U. A\$i\$p i = 0$

by blast

with prod_zero[OF fU] **show** ?pp p = 0

```

      by force
    qed
  from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
  ?thesis
    unfolding det_def by (simp add: sign_id)
  qed

```

```

lemma det_upperdiagonal:
  fixes A :: 'a::comm_ring_1 ^ n :: {finite, wellorder} ^ n :: {finite, wellorder}
  assumes ld:  $\bigwedge i j. i > j \implies A\$i\$j = 0$ 
  shows det A = prod ( $\lambda i. A\$i\$i$ ) (UNIV :: 'n set)
proof -
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = ( $\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A\$i\$p\ i)$ ) (UNIV :: 'n set)
  have fU: finite ?U
    by simp
  have id0: {id}  $\subseteq$  ?PU
    by (auto simp: permutes_id)
  have p0:  $\forall p \in ?PU - \{id\}. ?pp\ p = 0$ 
  proof
    fix p
    assume p:  $p \in ?PU - \{id\}$ 
    then obtain i where i:  $p\ i < i$ 
      by clarify (meson leI permutes_natset_ge)
    from ld[OF i] have  $\exists i \in ?U. A\$i\$p\ i = 0$ 
      by blast
    with prod_zero[OF fU] show ?pp p = 0
      by force
  qed
  from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
  ?thesis
    unfolding det_def by (simp add: sign_id)
  qed

```

```

proposition det_diagonal:
  fixes A :: 'a::comm_ring_1 ^ n ^ n
  assumes ld:  $\bigwedge i j. i \neq j \implies A\$i\$j = 0$ 
  shows det A = prod ( $\lambda i. A\$i\$i$ ) (UNIV :: 'n set)
proof -
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp =  $\lambda p. \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. A\$i\$p\ i)$  (UNIV :: 'n set)
  have fU: finite ?U by simp
  from finite_permutations[OF fU] have fPU: finite ?PU .
  have id0: {id}  $\subseteq$  ?PU
    by (auto simp: permutes_id)
  have p0:  $\forall p \in ?PU - \{id\}. ?pp\ p = 0$ 
  proof

```



```

fix p
assume p: p ∈ ?PU - {id}
then obtain i where i: p i ≠ i
  by fastforce
with ld have ∃ i ∈ ?U. A $ i $ p i = 0
  by (metis UNIV_I)
with prod_zero [OF fU] show ?pp p = 0
  by force
qed
from sum.mono_neutral_cong_left[OF fPU id0 p0] show ?thesis
unfolding det_def by (simp add: sign_id)
qed

lemma det_I [simp]: det (mat 1 :: 'a::comm_ring_1 ^n ^n) = 1
  by (simp add: det_diagonal mat_def)

lemma det_0 [simp]: det (mat 0 :: 'a::comm_ring_1 ^n ^n) = 0
  by (simp add: det_def prod_zero power_0_left)

lemma det_permute_rows:
  fixes A :: 'a::comm_ring_1 ^n ^n
  assumes p: p permutes (UNIV :: 'n::finite set)
  shows det (χ i. A $ p i :: 'a ^n ^n) = of_int (sign p) * det A
proof -
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  have *: (∑ q ∈ ?PU. of_int (sign (q ∘ p)) * (∏ i ∈ ?U. A $ p i $ (q ∘ p) i)) =
    (∑ n ∈ ?PU. of_int (sign p) * of_int (sign n) * (∏ i ∈ ?U. A $ i $ n i))
  proof (rule sum.cong)
    fix q
    assume qPU: q ∈ ?PU
    have fU: finite ?U
      by simp
    from qPU have q: q permutes ?U
      by blast
    have prod (λi. A $ p i $ (q ∘ p) i) ?U = prod ((λi. A $ p i $ (q ∘ p) i) ∘ inv p) ?U
      by (simp only: prod.permute[OF permutes_inv[OF p], symmetric])
    also have ... = prod (λi. A $ (p ∘ inv p) i $ (q ∘ (p ∘ inv p)) i) ?U
      by (simp only: o_def)
    also have ... = prod (λi. A $ i $ q i) ?U
      by (simp only: o_def permutes_inverses[OF p])
    finally have thp: prod (λi. A $ p i $ (q ∘ p) i) ?U = prod (λi. A $ i $ q i) ?U
      by blast
    from p q have pp: permutation p and qp: permutation q
      by (metis fU permutation_permutes)+
    show of_int (sign (q ∘ p)) * prod (λi. A $ p i $ (q ∘ p) i) ?U =
      of_int (sign p) * of_int (sign q) * prod (λi. A $ i $ q i) ?U
      by (simp only: thp sign_compose[OF qp pp] mult.commute of_int_mult)
  qed auto

```

```

show ?thesis
  apply (simp add: det_def sum_distrib_left mult.assoc[symmetric])
  apply (subst sum_permutations_compose_right[OF p])
  apply (rule *)
  done
qed

lemma det_permute_columns:
  fixes A :: 'a::comm_ring_1n
  assumes p: p permutes (UNIV :: 'n set)
  shows det( $\chi$  i j. A $ i $ p j :: 'an) = of_int (sign p) * det A
proof -
  let ?Ap =  $\chi$  i j. A $ i $ p j :: 'an
  let ?At = transpose A
  have of_int (sign p) * det A = det (transpose ( $\chi$  i. transpose A $ p i))
    unfolding det_permute_rows[OF p, of ?At] det_transpose ..
  moreover
  have ?Ap = transpose ( $\chi$  i. transpose A $ p i)
    by (simp add: transpose_def vec_eq_iff)
  ultimately show ?thesis
    by simp
qed

lemma det_identical_columns:
  fixes A :: 'a::comm_ring_1n
  assumes jk: j  $\neq$  k
  and r: column j A = column k A
  shows det A = 0
proof -
  let ?U = UNIV :: 'n set
  let ?t_jk = Transposition.transpose j k
  let ?PU = {p. p permutes ?U}
  let ?S1 = {p. p  $\in$  ?PU  $\wedge$  evenperm p}
  let ?S2 = {(?t_jk  $\circ$  p) | p. p  $\in$  ?S1}
  let ?f =  $\lambda$ p. of_int (sign p) * ( $\prod$  i  $\in$  UNIV. A $ i $ p i)
  let ?g =  $\lambda$ p. ?t_jk  $\circ$  p
  have g_S1: ?S2 = ?g' ?S1 by auto
  have inj_g: inj_on ?g ?S1
proof (unfold inj_on_def, auto)
  fix x y assume x: x permutes ?U and even_x: evenperm x
  and y: y permutes ?U and even_y: evenperm y and eq: ?t_jk  $\circ$  x = ?t_jk
 $\circ$  y
  show x = y by (metis (opaque_lifting, no_types) comp_assoc eq_id_comp
    swap_id_idempotent)
qed
  have tjk_permutes: ?t_jk permutes ?U
  by (auto simp add: permutes_def dest: transpose_eq_imp_eq) (meson trans-
    pose_involutory)
  have tjk_eq:  $\forall$  i l. A $ i $ ?t_jk l = A $ i $ l

```

```

using r jk
unfolding column_def vec_eq_iff by (simp add: Transposition.transpose_def)

have sign_tjk: sign ?t_jk = -1 using sign_swap_id[of j k] jk by auto
{fix x
  assume x: x ∈ ?S1
  have sign (?t_jk ∘ x) = sign (?t_jk) * sign x
    by (metis (lifting) finite_class.finite_UNIV mem_Collect_eq
      permutation_permutes permutation_swap_id sign_compose x)
  also have ... = - sign x using sign_tjk by simp
  also have ... ≠ sign x unfolding sign_def by simp
  finally have sign (?t_jk ∘ x) ≠ sign x and (?t_jk ∘ x) ∈ ?S2
    using x by force+
}
hence disjoint: ?S1 ∩ ?S2 = {}
  by (force simp: sign_def)
have PU_decomposition: ?PU = ?S1 ∪ ?S2
proof (auto)
  fix x
  assume x: x permutes ?U and ∀ p. p permutes ?U → x = Transposition.transpose j k ∘ p → ¬ evenperm p
  then obtain p where p: p permutes UNIV and x_eq: x = Transposition.transpose j k ∘ p
    and odd_p: ¬ evenperm p
  by (metis (mono_tags) id_o_o_assoc permutes_compose swap_id_idempotent tjk_permutes)
  thus evenperm x
    by (meson evenperm_comp evenperm_swap finite_class.finite_UNIV
      jk_permutation_permutes permutation_swap_id)
next
  fix p assume p: p permutes ?U
  show Transposition.transpose j k ∘ p permutes UNIV by (metis p permutes_compose tjk_permutes)
qed
have sum ?f ?S2 = sum ((λp. of_int (sign p) * (∏ i ∈ UNIV. A $ i $ p i))
  ∘ (∘) (Transposition.transpose j k)) {p ∈ {p. p permutes UNIV}. evenperm p}
  unfolding g_S1 by (rule sum.reindex[OF inj_g])
also have ... = sum (λp. of_int (sign (?t_jk ∘ p)) * (∏ i ∈ UNIV. A $ i $ p i))
  ?S1
  unfolding o_def by (rule sum.cong, auto simp: tjk_eq)
also have ... = sum (λp. - ?f p) ?S1
proof (rule sum.cong, auto)
  fix x assume x: x permutes ?U
  and even_x: evenperm x
  hence perm_x: permutation x and perm_tjk: permutation ?t_jk
    using permutation_permutes[of x] permutation_permutes[of ?t_jk] permutation_swap_id
  by (metis finite_code)+
  have (sign (?t_jk ∘ x)) = - (sign x)

```

```

unfolding sign_compose[OF perm_tjk perm_x] sign_tjk by auto
thus of_int (sign (?t_jk ◦ x)) * (∏ i∈UNIV. A $ i $ x i)
  = - (of_int (sign x)) * (∏ i∈UNIV. A $ i $ x i)
by auto
qed
also have ... = - sum ?f ?S1 unfolding sum_negf ..
finally have *: sum ?f ?S2 = - sum ?f ?S1 .
have det A = (∑ p | p permutes UNIV. of_int (sign p)) * (∏ i∈UNIV. A $ i $
p i)
unfolding det_def ..
also have ... = sum ?f ?S1 + sum ?f ?S2
by (subst PU_decomposition, rule sum.union_disjoint[OF __ disjoint], auto)
also have ... = sum ?f ?S1 - sum ?f ?S1 unfolding * by auto
also have ... = 0 by simp
finally show det A = 0 by simp
qed

```

```

lemma det_identical_rows:
fixes A :: 'a::comm_ring_1n
assumes ij: i ≠ j and r: row i A = row j A
shows det A = 0
by (metis column_transpose det_identical_columns det_transpose ij r)

```

```

lemma det_zero_row:
fixes A :: 'a::{idom, ring_char_0}n and F :: 'b::{field}m
shows row i A = 0 ⇒ det A = 0 and row j F = 0 ⇒ det F = 0
by (force simp: row_def det_def vec_eq_iff sign_nz intro!: sum.neutral)+

```

```

lemma det_zero_column:
fixes A :: 'a::{idom, ring_char_0}n and F :: 'b::{field}m
shows column i A = 0 ⇒ det A = 0 and column j F = 0 ⇒ det F = 0
unfolding atomize_conj atomize_imp
by (metis det_transpose det_zero_row row_transpose)

```

```

lemma det_row_add:
fixes a b c :: 'n::finite ⇒ n
shows det((λ i. if i = k then a i + b i else c i)::'a::comm_ring_1n) =
  det((λ i. if i = k then a i else c i)::'a::comm_ring_1n) +
  det((λ i. if i = k then b i else c i)::'a::comm_ring_1n)
unfolding det_def vec_lambda_beta sum.distrib[symmetric]
proof (rule sum.cong)
let ?U = UNIV :: 'n set
let ?pU = {p. p permutes ?U}
let ?f = (λ i. if i = k then a i + b i else c i)::'n ⇒ 'a::comm_ring_1n
let ?g = (λ i. if i = k then a i else c i)::'n ⇒ 'a::comm_ring_1n
let ?h = (λ i. if i = k then b i else c i)::'n ⇒ 'a::comm_ring_1n
fix p
assume p: p ∈ ?pU
let ?Uk = ?U - {k}

```

```

from  $p$  have  $pU$ :  $p$  permutes  $?U$ 
  by blast
have  $kU$ :  $?U = \text{insert } k \ ?Uk$ 
  by blast
have  $eq$ :  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk$ 
            $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i) \ ?Uk$ 
  by auto
have  $Uk$ : finite  $?Uk$   $k \notin ?Uk$ 
  by auto
have  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) (\text{insert } k \ ?Uk)$ 
  unfolding  $kU$ [symmetric] ..
also have ... =  $?f \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk$ 
  by (rule prod.insert) auto
also have ... =  $(a \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk) + (b \ k \ \$ \ p \ k * \text{prod } (\lambda i.$ 
 $?f \ i \ \$ \ p \ i) \ ?Uk)$ 
  by (simp add: field_simps)
also have ... =  $(a \ k \ \$ \ p \ k * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk) + (b \ k \ \$ \ p \ k * \text{prod } (\lambda i.$ 
 $?h \ i \ \$ \ p \ i) \ ?Uk)$ 
  by (metis eq)
also have ... =  $\text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) (\text{insert } k \ ?Uk) + \text{prod } (\lambda i. ?h \ i \ \$ \ p \ i)$ 
 $(\text{insert } k \ ?Uk)$ 
  unfolding  $\text{prod.insert}[OF \ Uk]$  by simp
finally have  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U + \text{prod } (\lambda i. ?h$ 
 $i \ \$ \ p \ i) \ ?U$ 
  unfolding  $kU$ [symmetric] .
then show of_int (sign  $p$ ) *  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?U =$ 
 $\text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?U + \text{of\_int } (\text{sign } p) * \text{prod } (\lambda i. ?h \ i \ \$$ 
 $p \ i) \ ?U$ 
  by (simp add: field_simps)
qed auto

```

lemma *det_row_mul*:

```

fixes  $a \ b :: 'n::\text{finite} \Rightarrow \_ \wedge 'n$ 
shows  $\text{det}((\chi \ i. \text{if } i = k \ \text{then } c * s \ a \ i \ \text{else } b \ i)::'a::\text{comm\_ring\_1}^{\wedge n}) =$ 
 $c * \text{det}((\chi \ i. \text{if } i = k \ \text{then } a \ i \ \text{else } b \ i)::'a::\text{comm\_ring\_1}^{\wedge n})$ 
unfolding  $\text{det\_def } \text{vec\_lambda\_beta } \text{sum\_distrib\_left}$ 
proof (rule sum.cong)
  let  $?U = \text{UNIV} :: 'n \ \text{set}$ 
  let  $?pU = \{p. p \ \text{permutes } ?U\}$ 
  let  $?f = (\lambda i. \text{if } i = k \ \text{then } c * s \ a \ i \ \text{else } b \ i)::'n \Rightarrow 'a::\text{comm\_ring\_1}^{\wedge n}$ 
  let  $?g = (\lambda i. \text{if } i = k \ \text{then } a \ i \ \text{else } b \ i)::'n \Rightarrow 'a::\text{comm\_ring\_1}^{\wedge n}$ 
  fix  $p$ 
  assume  $p: p \in ?pU$ 
  let  $?Uk = ?U - \{k\}$ 
  from  $p$  have  $pU$ :  $p$  permutes  $?U$ 
    by blast
  have  $kU$ :  $?U = \text{insert } k \ ?Uk$ 
    by blast
  have  $eq$ :  $\text{prod } (\lambda i. ?f \ i \ \$ \ p \ i) \ ?Uk = \text{prod } (\lambda i. ?g \ i \ \$ \ p \ i) \ ?Uk$ 

```

```

    by auto
  have Uk: finite ?Uk k ∉ ?Uk
    by auto
  have prod (λi. ?f i $ p i) ?U = prod (λi. ?f i $ p i) (insert k ?Uk)
    unfolding kU[symmetric] ..
  also have ... = ?f k $ p k * prod (λi. ?f i $ p i) ?Uk
    by (rule prod.insert) auto
  also have ... = (c*s a k) $ p k * prod (λi. ?f i $ p i) ?Uk
    by (simp add: field_simps)
  also have ... = c* (a k $ p k * prod (λi. ?g i $ p i) ?Uk)
    unfolding eq by (simp add: ac_simps)
  also have ... = c* (prod (λi. ?g i $ p i) (insert k ?Uk))
    unfolding prod.insert[OF Uk] by simp
  finally have prod (λi. ?f i $ p i) ?U = c* (prod (λi. ?g i $ p i) ?U)
    unfolding kU[symmetric] .
  then show of_int (sign p) * prod (λi. ?f i $ p i) ?U = c * (of_int (sign p) *
prod (λi. ?g i $ p i) ?U)
    by (simp add: field_simps)
qed auto

```

```

lemma det_row_0:
  fixes b :: 'n::finite ⇒ _ ^ 'n
  shows det((χ i. if i = k then 0 else b i)::'a::comm_ring_1 ^ 'n ^ 'n) = 0
  using det_row_mul[of k 0 λi. 1 b]
  apply simp
  apply (simp only: vector_smult_lzero)
  done

```

```

lemma det_row_operation:
  fixes A :: 'a::{comm_ring_1} ^ 'n ^ 'n
  assumes ij: i ≠ j
  shows det (χ k. if k = i then row i A + c * s row j A else row k A) = det A
proof -
  let ?Z = (χ k. if k = i then row j A else row k A) :: 'a ^ 'n ^ 'n
  have th: row i ?Z = row j ?Z by (vector row_def)
  have th2: ((χ k. if k = i then row i A else row k A) :: 'a ^ 'n ^ 'n) = A
    by (vector row_def)
  show ?thesis
    unfolding det_row_add [of i] det_row_mul[of i] det_identical_rows[OF ij th]
    th2
    by simp
qed

```

```

lemma det_row_span:
  fixes A :: 'a::{field} ^ 'n ^ 'n
  assumes x: x ∈ vec.span {row j A | j. j ≠ i}
  shows det (χ k. if k = i then row i A + x else row k A) = det A
  using x
proof (induction rule: vec.span_induct_alt)

```

```

case base
have (if k = i then row i A + 0 else row k A) = row k A for k
  by simp
then show ?case
  by (simp add: row_def)
next
case (step c z y)
then obtain j where j: z = row j A i ≠ j
  by blast
let ?w = row i A + y
have th0: row i A + (c*s z + y) = ?w + c*s z
  by vector
let ?d = λx. det (χ k. if k = i then x else row k A)
have thz: ?d z = 0
  apply (rule det_identical_rows[OF j(2)])
  using j
  apply (vector row_def)
  done
have ?d (row i A + (c*s z + y)) = ?d (?w + c*s z)
  unfolding th0 ..
then have ?d (row i A + (c*s z + y)) = det A
  unfolding thz step.IH det_row_mul[of i] det_row_add[of i] by simp
then show ?case
  unfolding scalar_mult_eq_scaleR .
qed

```

```

lemma matrix_id [simp]: det (matrix id) = 1
  by (simp add: matrix_id_mat_1)

```

```

proposition det_matrix_scaleR [simp]: det (matrix (((*_R) r)) :: real^n^n) = r
  ^ CARD('n::finite)
  apply (subst det_diagonal)
  apply (auto simp: matrix_def mat_def)
  apply (simp add: cart_eq_inner_axis inner_axis_axis)
  done

```

May as well do this, though it's a bit unsatisfactory since it ignores exact duplicates by considering the rows/columns as a set.

```

lemma det_dependent_rows:
  fixes A:: 'a::{field}^n^n
  assumes d: vec.dependent (rows A)
  shows det A = 0
proof -
  let ?U = UNIV :: 'n set
  from d obtain i where i: row i A ∈ vec.span (rows A - {row i A})
  unfolding vec.dependent_def rows_def by blast
  show ?thesis
  proof (cases ∀ i j. i ≠ j → row i A ≠ row j A)
  case True

```

```

with i have vec.span (rows A - {row i A}) ⊆ vec.span {row j A | j. j ≠ i}
  by (auto simp: rows_def intro!: vec.span_mono)
then have - row i A ∈ vec.span {row j A | j. j ≠ i}
  by (meson i subsetCE vec.span_neg)
from det_row_span[OF this]
have det A = det (χ k. if k = i then 0 * s 1 else row k A)
  unfolding right_minus vector_smult_lzero ..
with det_row_mul[of i 0 λi. 1]
show ?thesis by simp
next
case False
then obtain j k where jk: j ≠ k row j A = row k A
  by auto
from det_identical_rows[OF jk] show ?thesis .
qed
qed

```

```

lemma det_dependent_columns:
  assumes d: vec.dependent (columns (A::realnn))
  shows det A = 0
  by (metis d det_dependent_rows rows_transpose det_transpose)

```

Multilinearity and the multiplication formula

```

lemma Cart_lambda_cong: (∧x. f x = g x) ⇒ (vec_lambda f::'an) = (vec_lambda
g :: 'an)
  by auto

```

```

lemma det_linear_row_sum:
  assumes fS: finite S
  shows det ((χ i. if i = k then sum (a i) S else c i)::'a::comm_ring_1nn) =
    sum (λj. det ((χ i. if i = k then a i j else c i)::'ann)) S
  using fS by (induct rule: finite_induct; simp add: det_row_0 det_row_add
cong: if_cong)

```

```

lemma finite_bounded_functions:
  assumes fS: finite S
  shows finite {f. (∀i ∈ {1..(k::nat)}. f i ∈ S) ∧ (∀i. i ∉ {1..k} ⇒ f i = i)}
proof (induct k)
  case 0
  have *: {f. ∀i. f i = i} = {id}
    by auto
  show ?case
    by (auto simp: *)
next
  case (Suc k)
  let ?f = λ(y::nat,g) i. if i = Suc k then y else g i
  let ?S = ?f ' (S × {f. (∀i ∈ {1..k}. f i ∈ S) ∧ (∀i. i ∉ {1..k} ⇒ f i = i)})
  have ?S = {f. (∀i ∈ {1..Suc k}. f i ∈ S) ∧ (∀i. i ∉ {1..Suc k} ⇒ f i = i)}
  apply (auto simp: image_iff)

```



```

  apply (rename_tac f)
  apply (rule_tac x=f (Suc k) in bexI)
  apply (rule_tac x =  $\lambda i. \text{if } i = \text{Suc } k \text{ then } i \text{ else } f \ i$  in exI, auto)
  done
with finite_imageI[OF finite_cartesian_product[OF fS Suc.hyps(1)], of ?f]
show ?case
  by metis
qed

lemma det_linear_rows_sum_lemma:
  assumes fS: finite S
  and fT: finite T
  shows det (( $\chi \ i. \text{if } i \in T \text{ then } \text{sum } (a \ i) \ S \ \text{else } c \ i$ ):: 'a::comm_ring_1'^n^n) =
    sum ( $\lambda f. \text{det}((\chi \ i. \text{if } i \in T \text{ then } a \ i \ (f \ i) \ \text{else } c \ i)$ :: 'a'^n^n))
    {f. ( $\forall i \in T. f \ i \in S$ )  $\wedge$  ( $\forall i. i \notin T \longrightarrow f \ i = i$ )}
  using fT
proof (induct T arbitrary: a c set: finite)
  case empty
  have th0:  $\bigwedge x \ y. (\chi \ i. \text{if } i \in \{\} \text{ then } x \ i \ \text{else } y \ i) = (\chi \ i. y \ i)$ 
  by vector
  from empty.prem show ?case
  unfolding th0 by (simp add: eq_id_iff)
next
  case (insert z T a c)
  let ?F =  $\lambda T. \{f. (\forall i \in T. f \ i \in S) \wedge (\forall i. i \notin T \longrightarrow f \ i = i)\}$ 
  let ?h =  $\lambda (y,g) \ i. \text{if } i = z \text{ then } y \ \text{else } g \ i$ 
  let ?k =  $\lambda h. (h(z), (\lambda i. \text{if } i = z \text{ then } i \ \text{else } h \ i))$ 
  let ?s =  $\lambda k \ a \ c \ f. \text{det}((\chi \ i. \text{if } i \in T \text{ then } a \ i \ (f \ i) \ \text{else } c \ i)$ :: 'a'^n^n)
  let ?c =  $\lambda j \ i. \text{if } i = z \text{ then } a \ i \ j \ \text{else } c \ i$ 
  have thif:  $\bigwedge a \ b \ c \ d. (\text{if } a \vee b \ \text{then } c \ \text{else } d) = (\text{if } a \ \text{then } c \ \text{else if } b \ \text{then } c \ \text{else } d)$ 
  by simp
  have thif2:  $\bigwedge a \ b \ c \ d \ e. (\text{if } a \ \text{then } b \ \text{else if } c \ \text{then } d \ \text{else } e) =$ 
    ( $\text{if } c \ \text{then } (\text{if } a \ \text{then } b \ \text{else } d) \ \text{else } (\text{if } a \ \text{then } b \ \text{else } e)$ )
  by simp
  from  $\langle z \notin T \rangle$  have nz:  $\bigwedge i. i \in T \implies i \neq z$ 
  by auto
  have det ( $\chi \ i. \text{if } i \in \text{insert } z \ T \text{ then } \text{sum } (a \ i) \ S \ \text{else } c \ i$ ) =
    det ( $\chi \ i. \text{if } i = z \text{ then } \text{sum } (a \ i) \ S \ \text{else if } i \in T \text{ then } \text{sum } (a \ i) \ S \ \text{else } c \ i$ )
  unfolding insert_iff thif ..
  also have ... = ( $\sum j \in S. \text{det}(\chi \ i. \text{if } i \in T \text{ then } \text{sum } (a \ i) \ S \ \text{else if } i = z \text{ then } a$ 
 $i \ j \ \text{else } c \ i)$ )
  unfolding det_linear_row_sum[OF fS]
  by (subst thif2) (simp add: nz cong: if_cong)
  finally have tha:
    det ( $\chi \ i. \text{if } i \in \text{insert } z \ T \text{ then } \text{sum } (a \ i) \ S \ \text{else } c \ i$ ) =
      ( $\sum (j, f) \in S \times ?F \ T. \text{det}(\chi \ i. \text{if } i \in T \text{ then } a \ i \ (f \ i)$ 
        else if  $i = z$  then  $a \ i \ j$ 
        else  $c \ i$ ))

```

```

    unfolding insert.hyps unfolding sum.cartesian_product by blast
  show ?case unfolding tha
    using ⟨z ∉ T⟩
    by (intro sum.reindex_bij_witness[where i=?k and j=?h])
      (auto intro!: cong[OF refl[of det]] simp: vec_eq_iff)
qed

lemma det_linear_rows_sum:
  fixes S :: 'n::finite set
  assumes fS: finite S
  shows det (χ i. sum (a i) S) =
    sum (λf. det (χ i. a i (f i) :: 'a::comm_ring_1 ^ 'n ^ 'n)) {f. ∀ i. f i ∈ S}
proof -
  have th0: ∧ x y. ((χ i. if i ∈ (UNIV :: 'n set) then x i else y i) :: 'a ^ 'n ^ 'n) = (χ
i. x i)
    by vector
  from det_linear_rows_sum_lemma[OF fS, of UNIV :: 'n set a, unfolded th0,
OF finite]
  show ?thesis by simp
qed

lemma matrix_mul_sum_alt:
  fixes A B :: 'a::comm_ring_1 ^ 'n ^ 'n
  shows A ** B = (χ i. sum (λk. A $ i $ k *s B $ k) (UNIV :: 'n set))
  by (vector matrix_matrix_mult_def sum_component)

lemma det_rows_mul:
  det((χ i. c i *s a i) :: 'a::comm_ring_1 ^ 'n ^ 'n) =
    prod (λi. c i) (UNIV :: 'n set) * det((χ i. a i) :: 'a ^ 'n ^ 'n)
proof (simp add: det_def sum_distrib_left cong add: prod.cong, rule sum.cong)
  let ?U = UNIV :: 'n set
  let ?PU = {p. p permutes ?U}
  fix p
  assume pU: p ∈ ?PU
  let ?s = of_int (sign p)
  from pU have p: p permutes ?U
    by blast
  have prod (λi. c i * a i $ p i) ?U = prod c ?U * prod (λi. a i $ p i) ?U
    unfolding prod.distrib ..
  then show ?s * (∏ xa ∈ ?U. c xa * a xa $ p xa) =
    prod c ?U * (?s * (∏ xa ∈ ?U. a xa $ p xa))
    by (simp add: field_simps)
qed rule

proposition det_mul:
  fixes A B :: 'a::comm_ring_1 ^ 'n ^ 'n
  shows det (A ** B) = det A * det B
proof -
  let ?U = UNIV :: 'n set

```

```

let ?F = {f. (∀ i ∈ ?U. f i ∈ ?U) ∧ (∀ i. i ∉ ?U → f i = i)}
let ?PU = {p. p permutes ?U}
have p ∈ ?F if p permutes ?U for p
  by simp
then have PUF: ?PU ⊆ ?F by blast
{
  fix f
  assume fPU: f ∈ ?F - ?PU
  have fUU: f ` ?U ⊆ ?U
    using fPU by auto
  from fPU have f: ∀ i ∈ ?U. f i ∈ ?U ∧ i ∉ ?U → f i = i - (∀ y. ∃ !x. f x
= y)
    unfolding permutes_def by auto

  let ?A = (χ i. A$i$f i *s B$f i) :: 'a^n~n
  let ?B = (χ i. B$f i) :: 'a^n~n
  {
    assume fni: ¬ inj_on f ?U
    then obtain i j where ij: f i = f j i ≠ j
      unfolding inj_on_def by blast
    then have row i ?B = row j ?B
      by (vector row_def)
    with det_identical_rows[OF ij(2)]
    have det (χ i. A$i$f i *s B$f i) = 0
      unfolding det_rows_mul by force
  }
  moreover
  {
    assume fi: inj_on f ?U
    from f fi have fith: ∧ i j. f i = f j ⇒ i = j
      unfolding inj_on_def by metis
    note fs = fi[unfolded surjective_iff_injective_gen[OF finite finite refl fUU,
symmetric]]
    have ∃!x. f x = y for y
      using fith fs by blast
    with f(3) have det (χ i. A$i$f i *s B$f i) = 0
      by blast
  }
  ultimately have det (χ i. A$i$f i *s B$f i) = 0
    by blast
}
then have zth: ∀ f ∈ ?F - ?PU. det (χ i. A$i$f i *s B$f i) = 0
  by simp
{
  fix p
  assume pU: p ∈ ?PU
  from pU have p: p permutes ?U
    by blast
  let ?s = λp. of_int (sign p)

```

```

let ?f = λq. ?s p * (∏ i ∈ ?U. A $ i $ p i) * (?s q * (∏ i ∈ ?U. B $ i $ q i))
have (sum (λq. ?s q *
  (∏ i ∈ ?U. (χ i. A $ i $ p i * s B $ p i :: 'an'n) $ i $ q i)) ?PU) =
  (sum (λq. ?s p * (∏ i ∈ ?U. A $ i $ p i) * (?s q * (∏ i ∈ ?U. B $ i $ q i)))
?PU)
  unfolding sum_permutations_compose_right[OF permutes_inv[OF p], of ?f]
proof (rule sum.cong)
  fix q
  assume qU: q ∈ ?PU
  then have q: q permutes ?U
    by blast
  from p q have pp: permutation p and pq: permutation q
    unfolding permutation_permutes by auto
  have th00: of_int (sign p) * of_int (sign p) = (1::'a)
    ∧ a. of_int (sign p) * (of_int (sign p) * a) = a
    unfolding mult.assoc[symmetric]
    unfolding of_int_mult[symmetric]
    by (simp_all add: sign_idempotent)
  have ths: ?s q = ?s p * ?s (q ∘ inv p)
    using pp pq permutation_inverse[OF pp] sign_inverse[OF pp]
    by (simp add: th00 ac_simps sign_idempotent sign_compose)
  have th001: prod (λi. B $ i $ q (inv p i)) ?U = prod ((λi. B $ i $ q (inv p i)) ∘
p) ?U
    by (rule prod.permute[OF p])
  have thp: prod (λi. (χ i. A $ i $ p i * s B $ p i :: 'an'n) $ i $ q i) ?U =
    prod (λi. A $ i $ p i) ?U * prod (λi. B $ i $ q (inv p i)) ?U
    unfolding th001 prod.distrib[symmetric] o_def permutes_inverses[OF p]
    apply (rule prod.cong[OF refl])
    using permutes_in_image[OF q]
    apply vector
    done
  show ?s q * prod (λi. ((χ i. A $ i $ p i * s B $ p i :: 'an'n) $ i $ q i)) ?U =
    ?s p * (prod (λi. A $ i $ p i) ?U) * (?s (q ∘ inv p) * prod (λi. B $ i $ (q ∘ inv p
i) ?U))
    using ths thp pp pq permutation_inverse[OF pp] sign_inverse[OF pp]
    by (simp add: sign_nz th00 field_simps sign_idempotent sign_compose)
qed rule
}
then have th2: sum (λf. det (χ i. A $ i $ f i * s B $ f i)) ?PU = det A * det B
  unfolding det_def sum_product
  by (rule sum.cong [OF refl])
have det (A**B) = sum (λf. det (χ i. A $ i $ f i * s B $ f i)) ?F
  unfolding matrix_mul_sum_alt det_linear_rows_sum[OF finite]
  by simp
also have ... = sum (λf. det (χ i. A $ i $ f i * s B $ f i)) ?PU
  using sum.mono_neutral_cong_left[OF finite PUF zth, symmetric]
  unfolding det_rows_mul by auto
finally show ?thesis unfolding th2 .
qed

```

1.12.2 Relation to invertibility

proposition *invertible_det_nz*:
fixes $A :: 'a :: \{\text{field}\}^n$
shows $\text{invertible } A \longleftrightarrow \det A \neq 0$
proof (*cases invertible A*)
case *True*
then obtain $B :: 'a^n$ **where** $B: A ** B = \text{mat } 1$
unfolding *invertible_right_inverse* **by** *blast*
then have $\det (A ** B) = \det (\text{mat } 1 :: 'a^n)$
by *simp*
then show *?thesis*
by (*metis True det_I det_mul mult_zero_left one_neq_zero*)
next
case *False*
let $?U = \text{UNIV} :: 'n \text{ set}$
have $fU: \text{finite } ?U$
by *simp*
from *False* **obtain** $c \ i$ **where** $c: \text{sum } (\lambda i. c \ i * s \ \text{row } i \ A) \ ?U = 0$ **and** $iU: i \in ?U$ **and** $ci: c \ i \neq 0$
unfolding *invertible_right_inverse matrix_right_invertible_independent_rows*
by *blast*
have $\text{thr}0: - \ \text{row } i \ A = \text{sum } (\lambda j. (1 / c \ i) * s \ (c \ j * s \ \text{row } j \ A)) \ (?U - \{i\})$
unfolding *sum_cmul* **using** $c \ ci$
by (*auto simp: sum.remove[OF fU iU] eq_vector_fraction_iff add_eq_0_iff*)
have $\text{thr}: - \ \text{row } i \ A \in \text{vec.span } \{\text{row } j \ A \mid j. j \neq i\}$
unfolding *thr0* **by** (*auto intro: vec.span_base vec.span_scale vec.span_sum*)
let $?B = (\chi \ k. \text{if } k = i \ \text{then } 0 \ \text{else } \text{row } k \ A) :: 'a^n$
have $\text{thrb}: \text{row } i \ ?B = 0$ **using** iU **by** (*vector row_def*)
have $\det A = 0$
unfolding *det_row_span[OF thr, symmetric] right_minus*
unfolding *det_zero_row(2)[OF thrb] ..*
then show *?thesis*
by (*simp add: False*)
qed

lemma *det_nz_iff_inj_gen*:
fixes $f :: 'a :: \text{field}^n \Rightarrow 'a :: \text{field}^n$
assumes *Vector_Spaces.linear (*s) (*s) f*
shows $\det (\text{matrix } f) \neq 0 \longleftrightarrow \text{inj } f$
proof
assume $\det (\text{matrix } f) \neq 0$
then show *inj f*
using *assms invertible_det_nz inj_matrix_vector_mult* **by** *force*
next
assume *inj f*
show $\det (\text{matrix } f) \neq 0$
using *vec.linear_injective_left_inverse [OF assms <inj f>]*
by (*metis assms invertible_det_nz invertible_left_inverse matrix_compose_gen*)

matrix_id_mat_1)

qed

lemma *det_nz_iff_inj*:
fixes $f :: \text{real}^n \Rightarrow \text{real}^n$
assumes *linear f*
shows $\det (\text{matrix } f) \neq 0 \iff \text{inj } f$
using *det_nz_iff_inj_gen[of f] assms*
unfolding *linear_matrix_vector_mul_eq* .

lemma *det_eq_0_rank*:
fixes $A :: \text{real}^n$
shows $\det A = 0 \iff \text{rank } A < \text{CARD}(n)$
using *invertible_det_nz [of A]*
by (*auto simp: matrix_left_invertible_injective invertible_left_inverse less_rank_noninjective*)

Invertibility of matrices and corresponding linear functions

lemma *matrix_left_invertible_gen*:
fixes $f :: 'a::\text{field}^m \Rightarrow 'a::\text{field}^n$
assumes *Vector_Spaces.linear (*s) (*s) f*
shows $((\exists B. B ** \text{matrix } f = \text{mat } 1) \iff (\exists g. \text{Vector_Spaces.linear } (*s) (*s) g \wedge g \circ f = \text{id}))$
g \wedge g \circ f = id)
proof safe
fix B
assume $1: B ** \text{matrix } f = \text{mat } 1$
show $\exists g. \text{Vector_Spaces.linear } (*s) (*s) g \wedge g \circ f = \text{id}$
proof (*intro exI conjI*)
show *Vector_Spaces.linear (*s) (*s) ($\lambda y. B *v y$)*
by simp
show $((*v) B) \circ f = \text{id}$
unfolding *o_def*
by (*metis assms 1 eq_id_iff_matrix_vector_mul(1) matrix_vector_mul_assoc matrix_vector_mul_lid*)
qed
next
fix g
assume *Vector_Spaces.linear (*s) (*s) g g \circ f = id*
then have *matrix g ** matrix f = mat 1*
by (*metis assms matrix_compose_gen matrix_id_mat_1*)
then show $\exists B. B ** \text{matrix } f = \text{mat } 1 ..$
qed

lemma *matrix_left_invertible*:
 $\text{linear } f \implies ((\exists B. B ** \text{matrix } f = \text{mat } 1) \iff (\exists g. \text{linear } g \wedge g \circ f = \text{id}))$ **for**
 $f :: \text{real}^m \Rightarrow \text{real}^n$
using *matrix_left_invertible_gen[of f]*
by (*auto simp: linear_matrix_vector_mul_eq*)

```

lemma matrix_right_invertible_gen:
  fixes  $f :: 'a::field^m \Rightarrow 'a^n$ 
  assumes Vector_Spaces.linear  $(*s) (*s) f$ 
  shows  $((\exists B. \text{matrix } f ** B = \text{mat } 1) \longleftrightarrow (\exists g. \text{Vector\_Spaces.linear } (*s) (*s) g \wedge f \circ g = \text{id}))$ 
proof safe
  fix  $B$ 
  assume  $1: \text{matrix } f ** B = \text{mat } 1$ 
  show  $\exists g. \text{Vector\_Spaces.linear } (*s) (*s) g \wedge f \circ g = \text{id}$ 
  proof (intro exI conjI)
    show Vector_Spaces.linear  $(*s) (*s) ((*v) B)$ 
      by simp
    show  $f \circ ((*v) B) = \text{id}$ 
      using  $1$  assms comp_apply eq_id_iff vec.linear_id matrix_id_mat_1 matrix_vector_mul_assoc matrix_works
      by (metis (no_types, opaque_lifting))
  qed
next
  fix  $g$ 
  assume Vector_Spaces.linear  $(*s) (*s) g$  and  $f \circ g = \text{id}$ 
  then have matrix  $f ** \text{matrix } g = \text{mat } 1$ 
    by (metis assms matrix_compose_gen matrix_id_mat_1)
  then show  $\exists B. \text{matrix } f ** B = \text{mat } 1 ..$ 
qed

```

```

lemma matrix_right_invertible:
  linear  $f \implies ((\exists B. \text{matrix } f ** B = \text{mat } 1) \longleftrightarrow (\exists g. \text{linear } g \wedge f \circ g = \text{id}))$  for
 $f::\text{real}^m \Rightarrow \text{real}^n$ 
  using matrix_right_invertible_gen[of  $f$ ]
  by (auto simp: linear_matrix_vector_mul_eq)

```

```

lemma matrix_invertible_gen:
  fixes  $f :: 'a::field^m \Rightarrow 'a^n$ 
  assumes Vector_Spaces.linear  $(*s) (*s) f$ 
  shows  $\text{invertible } (\text{matrix } f) \longleftrightarrow (\exists g. \text{Vector\_Spaces.linear } (*s) (*s) g \wedge f \circ g = \text{id} \wedge g \circ f = \text{id})$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis assms invertible_def left_right_inverse_eq matrix_left_invertible_gen matrix_right_invertible_gen)
next
  assume ?rhs then show ?lhs
    by (metis assms invertible_def matrix_compose_gen matrix_id_mat_1)
qed

```

```

lemma matrix_invertible:
  linear  $f \implies \text{invertible } (\text{matrix } f) \longleftrightarrow (\exists g. \text{linear } g \wedge f \circ g = \text{id} \wedge g \circ f = \text{id})$ 
for  $f::\text{real}^m \Rightarrow \text{real}^n$ 

```

```

using matrix_invertible_gen[of f]
by (auto simp: linear_matrix_vector_mul_eq)

```

```

lemma invertible_eq_bij:
  fixes m :: 'a::{field}^m^n
  shows invertible m  $\longleftrightarrow$  bij ((*v) m)
  using matrix_invertible_gen[OF matrix_vector_mul_linear_gen, of m, simplified matrix_of_matrix_vector_mul]
  by (metis bij_betw_def left_right_inverse_eq matrix_vector_mul_linear_gen o_bij
    vec.linear_injective_left_inverse vec.linear_surjective_right_inverse)

```

1.12.3 Cramer's rule

```

lemma cramer_lemma_transpose:
  fixes A :: 'a::{field}^n^n
  and x :: 'a::{field}^n
  shows det (( $\chi$  i. if i = k then sum ( $\lambda$ i. xs *s row i A) (UNIV::'n set)
    else row i A)::'a::{field}^n^n) = x$k * det A
  (is ?lhs = ?rhs)
proof -
  let ?U = UNIV :: 'n set
  let ?Uk = ?U - {k}
  have U: ?U = insert k ?Uk
  by blast
  have kUk: k  $\notin$  ?Uk
  by simp
  have th00:  $\bigwedge$  k s. x$k *s row k A + s = (x$k - 1) *s row k A + row k A + s
  by (vector field_simps)
  have th001:  $\bigwedge$  f k . ( $\lambda$ x. if x = k then f k else f x) = f
  by auto
  have ( $\chi$  i. row i A) = A by (vector row_def)
  then have thd1: det ( $\chi$  i. row i A) = det A
  by simp
  have thd0: det ( $\chi$  i. if i = k then row k A + ( $\sum$  i  $\in$  ?Uk. x $ i *s row i A) else
row i A) = det A
  by (force intro: det_row_span vec.span_sum vec.span_scale vec.span_base)
  show ?lhs = x$k * det A
  apply (subst U)
  unfolding sum.insert[OF finite kUk]
  apply (subst th00)
  unfolding add.assoc
  apply (subst det_row_add)
  unfolding thd0
  unfolding det_row_mul
  unfolding th001[of k  $\lambda$ i. row i A]
  unfolding thd1
  apply (simp add: field_simps)
done

```


qed

proposition *cramer_lemma*:

fixes $A :: 'a::\{field\}^{n \times n}$
shows $\det((\chi \ i \ j. \text{if } j = k \text{ then } (A * v \ x)\$i \text{ else } A\$i\$j)) :: 'a::\{field\}^{n \times n} = x\$k * \det A$

proof –

let $?U = UNIV :: 'n \text{ set}$
have $*$: $\bigwedge c. \text{sum } (\lambda i. c \ i * s \ \text{row } i \ (\text{transpose } A)) \ ?U = \text{sum } (\lambda i. c \ i * s \ \text{column } i \ A) \ ?U$
by (*auto intro: sum.cong*)
show *?thesis*
unfolding *matrix_mult_sum*
unfolding *cramer_lemma_transpose*[*of k x transpose A, unfolded det_transpose, symmetric*]
unfolding $*$ [*of \lambda i. x\\$i*]
apply (*subst det_transpose[symmetric]*)
apply (*rule cong[OF refl[of det]]*)
apply (*vector_transpose_def column_def row_def*)
done

qed

proposition *cramer*:

fixes $A :: 'a::\{field\}^{n \times n}$
assumes $d0: \det A \neq 0$
shows $A * v \ x = b \iff x = (\chi \ k. \det(\chi \ i \ j. \text{if } j=k \text{ then } b\$i \text{ else } A\$i\$j) / \det A)$

proof –

from $d0$ **obtain** B **where** $B: A ** B = \text{mat } 1 \ B ** A = \text{mat } 1$
unfolding *invertible_det_nz[symmetric]* *invertible_def*
by *blast*
have $(A ** B) * v \ b = b$
by (*simp add: B*)
then have $A * v \ (B * v \ b) = b$
by (*simp add: matrix_vector_mul_assoc*)
then have x : $\exists x. A * v \ x = b$
by *blast*
{
fix x
assume $x: A * v \ x = b$
have $x = (\chi \ k. \det(\chi \ i \ j. \text{if } j=k \text{ then } b\$i \text{ else } A\$i\$j) / \det A)$
unfolding x [*symmetric*]
using $d0$ **by** (*simp add: vec_eq_iff cramer_lemma field_simps*)
}
with x **show** *?thesis*
by *auto*

qed

lemma *det_1*: $\det (A :: 'a::\text{comm_ring_1}^{1 \times 1}) = A\$1\$1$
by (*simp add: det_def sign_id*)

lemma *det_2*: $\det (A::'a::\text{comm_ring_1}^{\wedge 2\wedge 2}) = A\$1\$1 * A\$2\$2 - A\$1\$2 * A\$2\$1$

proof –

have *f12*: *finite* {2::2} 1 \notin {2::2} **by** *auto*

show *?thesis*

unfolding *det_def UNIV_2*

unfolding *sum_over_permutations_insert[OF f12]*

unfolding *permutes_sing*

by (*simp add: sign_swap_id sign_id swap_id_eq*)

qed

lemma *det_3*:

$\det (A::'a::\text{comm_ring_1}^{\wedge 3\wedge 3}) =$

$A\$1\$1 * A\$2\$2 * A\$3\$3 +$

$A\$1\$2 * A\$2\$3 * A\$3\$1 +$

$A\$1\$3 * A\$2\$1 * A\$3\$2 -$

$A\$1\$1 * A\$2\$3 * A\$3\$2 -$

$A\$1\$2 * A\$2\$1 * A\$3\$3 -$

$A\$1\$3 * A\$2\$2 * A\$3\1

proof –

have *f123*: *finite* {2::3, 3} 1 \notin {2::3, 3}

by *auto*

have *f23*: *finite* {3::3} 2 \notin {3::3}

by *auto*

show *?thesis*

unfolding *det_def UNIV_3*

unfolding *sum_over_permutations_insert[OF f123]*

unfolding *sum_over_permutations_insert[OF f23]*

unfolding *permutes_sing*

by (*simp add: sign_swap_id permutation_swap_id sign_compose sign_id swap_id_eq*)

qed

proposition *det_orthogonal_matrix*:

fixes *Q*:: 'a::*linordered_idom*^{^n^}^{^n^}

assumes *oQ*: *orthogonal_matrix* *Q*

shows $\det Q = 1 \vee \det Q = -1$

proof –

have $Q ** \text{transpose } Q = \text{mat } 1$

by (*metis oQ orthogonal_matrix_def*)

then have $\det (Q ** \text{transpose } Q) = \det (\text{mat } 1:: 'a^{\wedge n^{\wedge n}})$

by *simp*

then have $\det Q * \det Q = 1$

by (*simp add: det_mul*)

then show *?thesis*

by (*simp add: square_eq_1_iff*)

qed

proposition *orthogonal_transformation_det* [simp]:
fixes $f :: \text{real}^n \Rightarrow \text{real}^n$
shows $\text{orthogonal_transformation } f \implies |\det (\text{matrix } f)| = 1$
using *det_orthogonal_matrix orthogonal_transformation_matrix* **by** *fastforce*

1.12.4 Rotation, reflection, rotoinversion

definition *rotation_matrix* $Q \longleftrightarrow \text{orthogonal_matrix } Q \wedge \det Q = 1$

definition *rotoinversion_matrix* $Q \longleftrightarrow \text{orthogonal_matrix } Q \wedge \det Q = -1$

lemma *orthogonal_rotation_or_rotoinversion*:

fixes $Q :: 'a::\text{linordered_idom}^n$

shows $\text{orthogonal_matrix } Q \longleftrightarrow \text{rotation_matrix } Q \vee \text{rotoinversion_matrix } Q$
by (*metis rotoinversion_matrix_def rotation_matrix_def det_orthogonal_matrix*)

Slightly stronger results giving rotation, but only in two or more dimensions

lemma *rotation_matrix_exists_basis*:

fixes $a :: \text{real}^n$

assumes $2: 2 \leq \text{CARD}(n)$ **and** $\text{norm } a = 1$

obtains A **where** $\text{rotation_matrix } A \wedge A * v (\text{axis } k \ 1) = a$

proof –

obtain A **where** $\text{orthogonal_matrix } A$ **and** $A: A * v (\text{axis } k \ 1) = a$

using *orthogonal_matrix_exists_basis* *assms* **by** *metis*

with *orthogonal_rotation_or_rotoinversion*

consider $\text{rotation_matrix } A \mid \text{rotoinversion_matrix } A$

by *metis*

then show *thesis*

proof *cases*

assume $\text{rotation_matrix } A$

then show *?thesis*

using $\langle A * v \text{axis } k \ 1 = a \rangle$ **that** **by** *auto*

next

from *obtain_subset_with_card_n[OF 2]* **obtain** $h :: 'n$ **where** $h \neq i$

by (*fastforce simp add: eval_nat_numeral card_Suc_eq*)

then obtain j **where** $j \neq k$

by (*metis (full_types)*)

let $?TA = \text{transpose } A$

let $?A = \chi \ i. \ \text{if } i = j \ \text{then } -1 *_{\mathbb{R}} (?TA \ \$ \ i) \ \text{else } ?TA \ \$ \ i$

assume $\text{rotoinversion_matrix } A$

then have [simp]: $\det A = -1$

by (*simp add: rotoinversion_matrix_def*)

show *?thesis*

proof

have [simp]: $\text{row } i (\chi \ i. \ \text{if } i = j \ \text{then } -1 *_{\mathbb{R}} ?TA \ \$ \ i \ \text{else } ?TA \ \$ \ i) = (\text{if } i = j \ \text{then } - \ \text{row } i \ ?TA \ \text{else } \text{row } i \ ?TA)$ **for** i

by (*auto simp: row_def*)

have $\text{orthogonal_matrix } ?A$

unfolding *orthogonal_matrix_orthonormal_rows*

```

    using ⟨orthogonal_matrix A⟩ by (auto simp: orthogonal_matrix_orthonormal_columns
orthogonal_clauses)
    then show rotation_matrix (transpose ?A)
      unfolding rotation_matrix_def
      by (simp add: det_row_mul[of j _ λi. ?TA $ i, unfolded scalar_mult_eq_scaleR])
    show transpose ?A *v axis k 1 = a
      using ⟨j ≠ k⟩ A by (simp add: matrix_vector_column_axis_def scalar_mult_eq_scaleR
if_distrib [of λz. z *R c for c] cong: if_cong)
    qed
  qed
qed

```

```

lemma rotation_exists_1:
  fixes a :: real'n
  assumes 2 ≤ CARD('n) norm a = 1 norm b = 1
  obtains f where orthogonal_transformation f det(matrix f) = 1 f a = b
proof -
  obtain k::'n where True
    by simp
  obtain A B where AB: rotation_matrix A rotation_matrix B
    and eq: A *v (axis k 1) = a B *v (axis k 1) = b
    using rotation_matrix_exists_basis assms by metis
  let ?f = λx. (B ** transpose A) *v x
  show thesis
proof
  show orthogonal_transformation ?f
    using AB orthogonal_matrix_mul orthogonal_transformation_matrix rotation_matrix_def matrix_vector_mul_linear by force
  show det (matrix ?f) = 1
    using AB by (auto simp: det_mul rotation_matrix_def)
  show ?f a = b
    using AB unfolding orthogonal_matrix_def rotation_matrix_def
    by (metis eq matrix_mul_rid matrix_vector_mul_assoc)
qed
qed

```

```

lemma rotation_exists:
  fixes a :: real'n
  assumes 2: 2 ≤ CARD('n) and eq: norm a = norm b
  obtains f where orthogonal_transformation f det(matrix f) = 1 f a = b
proof (cases a = 0 ∨ b = 0)
  case True
  with assms have a = 0 b = 0
    by auto
  then show ?thesis
    by (metis eq_id_iff matrix_id orthogonal_transformation_id that)
next
  case False
  then obtain f where f: orthogonal_transformation f det (matrix f) = 1

```

```

    and f': f (a /R norm a) = b /R norm b
    using rotation_exists_1 [of a /R norm a b /R norm b, OF 2] by auto
    then interpret linear f by (simp add: orthogonal_transformation)
    have f a = b
      using f' False
      by (simp add: eq scale)
    with f show thesis ..
qed

lemma rotation_rightward_line:
  fixes a :: realn
  obtains f where orthogonal_transformation f 2 ≤ CARD('n) ⇒ det(matrix
f) = 1
      f(norm a *R axis k 1) = a
proof (cases CARD('n) = 1)
  case True
  obtain f where orthogonal_transformation f f (norm a *R axis k (1::real)) = a
  proof (rule orthogonal_transformation_exists)
    show norm (norm a *R axis k (1::real)) = norm a
      by simp
  qed auto
  then show thesis
    using True that by auto
next
  case False
  obtain f where orthogonal_transformation f det(matrix f) = 1 f (norm a *R
axis k 1) = a
  proof (rule rotation_exists)
    show 2 ≤ CARD('n)
      using False one_le_card_finite [where 'a='n] by linarith
    show norm (norm a *R axis k (1::real)) = norm a
      by simp
  qed auto
  then show thesis
    using that by blast
qed

end

```

1.13 Operators involving abstract topology

```

theory Abstract_Topology
  imports
    Complex_Main
    HOL-Library.Set_Idioms
    HOL-Library.FuncSet
begin

```

1.13.1 General notion of a topology as a value

definition *istopology* :: ('a set \Rightarrow bool) \Rightarrow bool **where**
istopology L \equiv ($\forall S T. L S \longrightarrow L T \longrightarrow L (S \cap T)$) \wedge ($\forall \mathcal{K}. (\forall K \in \mathcal{K}. L K) \longrightarrow L (\bigcup \mathcal{K})$)

typedef 'a topology = {L::('a set) \Rightarrow bool. *istopology* L}
morphisms *openin* topology
unfolding *istopology_def* **by** *blast*

lemma *istopology_openin*[*iff*]: *istopology*(*openin* U)
using *openin*[*of* U] **by** *blast*

lemma *istopology_open*[*iff*]: *istopology* open
by (*auto simp: istopology_def*)

lemma *topology_inverse'* [*simp*]: *istopology* U \Longrightarrow *openin* (topology U) = U
using *topology_inverse*[*unfolded mem_Collect_eq*].

lemma *topology_inverse_iff*: *istopology* U \longleftrightarrow *openin* (topology U) = U
by (*metis istopology_openin topology_inverse'*)

lemma *topology_eq*: T1 = T2 \longleftrightarrow ($\forall S. \text{openin } T1 S \longleftrightarrow \text{openin } T2 S$)
proof

assume T1 = T2

then show $\forall S. \text{openin } T1 S \longleftrightarrow \text{openin } T2 S$ **by** *simp*

next

assume H: $\forall S. \text{openin } T1 S \longleftrightarrow \text{openin } T2 S$

then have *openin* T1 = *openin* T2 **by** (*simp add: fun_eq_iff*)

then have topology (*openin* T1) = topology (*openin* T2) **by** *simp*

then show T1 = T2 **unfolding** *openin_inverse*.

qed

The "universe": the union of all sets in the topology.

definition *topspace* T = $\bigcup \{S. \text{openin } T S\}$

Main properties of open sets

proposition *openin_clauses*:

fixes U :: 'a topology

shows

openin U {}

$\bigwedge S T. \text{openin } U S \Longrightarrow \text{openin } U T \Longrightarrow \text{openin } U (S \cap T)$

$\bigwedge K. (\forall S \in K. \text{openin } U S) \Longrightarrow \text{openin } U (\bigcup K)$

using *openin*[*of* U] **unfolding** *istopology_def* **by** *auto*

lemma *openin_subset*: *openin* U S \Longrightarrow S \subseteq *topspace* U
unfolding *topspace_def* **by** *blast*

lemma *openin_empty*[*simp*]: *openin* U {}

```

  by (rule openin_clauses)

lemma openin_Int[intro]: openin U S  $\implies$  openin U T  $\implies$  openin U (S  $\cap$  T)
  by (rule openin_clauses)

lemma openin_Union[intro]: ( $\bigwedge S. S \in K \implies$  openin U S)  $\implies$  openin U ( $\bigcup K$ )
  using openin_clauses by blast

lemma openin_Un[intro]: openin U S  $\implies$  openin U T  $\implies$  openin U (S  $\cup$  T)
  using openin_Union[of {S,T} U] by auto

lemma openin_topspace[intro, simp]: openin U (topspace U)
  by (force simp: openin_Union topspace_def)

lemma openin_subopen: openin U S  $\longleftrightarrow$  ( $\forall x \in S. \exists T. openin U T \wedge x \in T \wedge T \subseteq S$ )
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs
  then show ?rhs by auto
next
  assume H: ?rhs
  let ?t =  $\bigcup \{T. openin U T \wedge T \subseteq S\}$ 
  have openin U ?t by (force simp: openin_Union)
  also have ?t = S using H by auto
  finally show openin U S .
qed

lemma openin_INT [intro]:
  assumes finite I
     $\bigwedge i. i \in I \implies openin T (U i)$ 
  shows openin T (( $\bigcap i \in I. U i$ )  $\cap$  topspace T)
  using assms by (induct, auto simp: inf_sup_aci(2) openin_Int)

lemma openin_INT2 [intro]:
  assumes finite I I  $\neq$  {}
     $\bigwedge i. i \in I \implies openin T (U i)$ 
  shows openin T ( $\bigcap i \in I. U i$ )
proof -
  have ( $\bigcap i \in I. U i$ )  $\subseteq$  topspace T
    using  $\langle I \neq \{\} \rangle$  openin_subset[OF assms(3)] by auto
  then show ?thesis
    using openin_INT[of _ _ U, OF assms(1) assms(3)] by (simp add: inf_absorb2 inf_commute)
qed

lemma openin_Inter [intro]:
  assumes finite  $\mathcal{F}$   $\mathcal{F} \neq \{\}$   $\bigwedge X. X \in \mathcal{F} \implies openin T X$  shows openin T ( $\bigcap \mathcal{F}$ )
  by (metis (full_types) assms openin_INT2 image_ident)

```

lemma *openin_Int_Inter*:
assumes *finite* \mathcal{F} *openin* $T\ U \ \bigwedge X. X \in \mathcal{F} \implies \text{openin } T\ X$ **shows** *openin* $T\ (U \cap \bigcap \mathcal{F})$
using *openin_Inter* [of *insert* $U\ \mathcal{F}$] *assms* **by** *auto*

Closed sets

definition *closedin* :: '*a* topology \implies '*a* set \implies bool **where**
closedin $U\ S \longleftrightarrow S \subseteq \text{topspace } U \wedge \text{openin } U\ (\text{topspace } U - S)$

lemma *closedin_subset*: *closedin* $U\ S \implies S \subseteq \text{topspace } U$
by (*metis* *closedin_def*)

lemma *closedin_empty*[*simp*]: *closedin* $U\ \{\}$
by (*simp* *add*: *closedin_def*)

lemma *closedin_topspace*[*intro*, *simp*]: *closedin* $U\ (\text{topspace } U)$
by (*simp* *add*: *closedin_def*)

lemma *closedin_Un*[*intro*]: *closedin* $U\ S \implies \text{closedin } U\ T \implies \text{closedin } U\ (S \cup T)$
by (*auto* *simp*: *Diff_Un* *closedin_def*)

lemma *Diff_Inter*[*intro*]: $A - \bigcap S = \bigcup \{A - s \mid s \in S\}$
by *auto*

lemma *closedin_Union*:
assumes *finite* $S \ \bigwedge T. T \in S \implies \text{closedin } U\ T$
shows *closedin* $U\ (\bigcup S)$
using *assms* **by** *induction* *auto*

lemma *closedin_Inter*[*intro*]:
assumes *Ke*: $K \neq \{\}$
and *Kc*: $\bigwedge S. S \in K \implies \text{closedin } U\ S$
shows *closedin* $U\ (\bigcap K)$
using *Ke* *Kc* **unfolding** *closedin_def* *Diff_Inter* **by** *auto*

lemma *closedin_INT*[*intro*]:
assumes $A \neq \{\} \ \bigwedge x. x \in A \implies \text{closedin } U\ (B\ x)$
shows *closedin* $U\ (\bigcap_{x \in A} B\ x)$
using *assms* **by** *blast*

lemma *closedin_Int*[*intro*]: *closedin* $U\ S \implies \text{closedin } U\ T \implies \text{closedin } U\ (S \cap T)$
using *closedin_Inter*[of $\{S, T\}\ U$] **by** *auto*

lemma *openin_closedin_eq*: *openin* $U\ S \longleftrightarrow S \subseteq \text{topspace } U \wedge \text{closedin } U\ (\text{topspace } U - S)$

by (metis Diff_subset closedin_def double_diff equalityD1 openin_subset)

lemma topology_finer_closedin:

$\text{topspace } X = \text{topspace } Y \implies (\forall S. \text{openin } Y S \longrightarrow \text{openin } X S) \longleftrightarrow (\forall S. \text{closedin } Y S \longrightarrow \text{closedin } X S)$

by (metis closedin_def openin_closedin_eq)

lemma openin_closedin: $S \subseteq \text{topspace } U \implies (\text{openin } U S \longleftrightarrow \text{closedin } U (\text{topspace } U - S))$

by (simp add: openin_closedin_eq)

lemma openin_diff[intro]:

assumes $oS: \text{openin } U S$

and $cT: \text{closedin } U T$

shows $\text{openin } U (S - T)$

by (metis Int_Diff cT closedin_def inf.orderE oS openin_Int openin_subset)

lemma closedin_diff[intro]:

assumes $oS: \text{closedin } U S$

and $cT: \text{openin } U T$

shows $\text{closedin } U (S - T)$

by (metis Int_Diff cT closedin_Int closedin_subset inf.orderE oS openin_closedin_eq)

lemma all_openin: $(\forall U. \text{openin } X U \longrightarrow P U) \longleftrightarrow (\forall U. \text{closedin } X U \longrightarrow P (\text{topspace } X - U))$

by (metis Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq)

lemma all_closedin: $(\forall U. \text{closedin } X U \longrightarrow P U) \longleftrightarrow (\forall U. \text{openin } X U \longrightarrow P (\text{topspace } X - U))$

by (metis Diff_Diff_Int closedin_subset inf.absorb_iff2 openin_closedin_eq)

lemma ex_openin: $(\exists U. \text{openin } X U \wedge P U) \longleftrightarrow (\exists U. \text{closedin } X U \wedge P (\text{topspace } X - U))$

by (metis Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq)

lemma ex_closedin: $(\exists U. \text{closedin } X U \wedge P U) \longleftrightarrow (\exists U. \text{openin } X U \wedge P (\text{topspace } X - U))$

by (metis Diff_Diff_Int closedin_subset inf.absorb_iff2 openin_closedin_eq)

1.13.2 The discrete topology

definition discrete_topology where $\text{discrete_topology } U \equiv \text{topology } (\lambda S. S \subseteq U)$

lemma openin_discrete_topology [simp]: $\text{openin } (\text{discrete_topology } U) S \longleftrightarrow S \subseteq U$

proof –

have $\text{istopology } (\lambda S. S \subseteq U)$

by (auto simp: istopology_def)

then show ?thesis

by (*simp add: discrete_topology_def topology_inverse'*)
qed

lemma *topspace_discrete_topology* [*simp*]: $\text{topspace}(\text{discrete_topology } U) = U$
by (*meson openin_discrete_topology openin_subset openin_topspace order_refl subset_antisym*)

lemma *closedin_discrete_topology* [*simp*]: $\text{closedin}(\text{discrete_topology } U) S \longleftrightarrow S \subseteq U$
by (*simp add: closedin_def*)

lemma *discrete_topology_unique*:
 $\text{discrete_topology } U = X \longleftrightarrow \text{topspace } X = U \wedge (\forall x \in U. \text{openin } X \{x\})$ (**is**
?lhs = ?rhs)
proof
assume *R*: *?rhs*
then have *openin X S* if $S \subseteq U$ for *S*
using *openin_subopen subsetD* that **by** *fastforce*
then show *?lhs*
by (*metis R openin_discrete_topology openin_subset topology_eq*)
qed *auto*

lemma *discrete_topology_unique_alt*:
 $\text{discrete_topology } U = X \longleftrightarrow \text{topspace } X \subseteq U \wedge (\forall x \in U. \text{openin } X \{x\})$
using *openin_subset*
by (*auto simp: discrete_topology_unique*)

lemma *subtopology_eq_discrete_topology_empty*:
 $X = \text{discrete_topology } \{\} \longleftrightarrow \text{topspace } X = \{\}$
using *discrete_topology_unique* [*of* $\{\}$ *X*] **by** *auto*

lemma *subtopology_eq_discrete_topology_sing*:
 $X = \text{discrete_topology } \{a\} \longleftrightarrow \text{topspace } X = \{a\}$
by (*metis discrete_topology_unique openin_topspace singletonD*)

abbreviation *trivial_topology* **where** $\text{trivial_topology} \equiv \text{discrete_topology } \{\}$

lemma *null_topspace_iff_trivial* [*simp*]: $\text{topspace } X = \{\} \longleftrightarrow X = \text{trivial_topology}$
by (*simp add: subtopology_eq_discrete_topology_empty*)

1.13.3 Subspace topology

definition *subtopology* :: $'a \text{ topology} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ topology}$
where $\text{subtopology } U V = \text{topology } (\lambda T. \exists S. T = S \cap V \wedge \text{openin } U S)$

lemma *istopology_subtopology*: $\text{istopology } (\lambda T. \exists S. T = S \cap V \wedge \text{openin } U S)$
(**is** *istopology ?L*)

proof –
have *?L* $\{\}$ **by** *blast*

```

{
  fix A B
  assume A: ?L A and B: ?L B
  from A B obtain Sa and Sb where Sa: openin U Sa A = Sa ∩ V and Sb:
openin U Sb B = Sb ∩ V
  by blast
  have A ∩ B = (Sa ∩ Sb) ∩ V openin U (Sa ∩ Sb)
  using Sa Sb by blast+
  then have ?L (A ∩ B) by blast
}
moreover
{
  fix K
  assume K: K ⊆ Collect ?L
  have th0: Collect ?L = (λS. S ∩ V) ‘ Collect (openin U)
  by blast
  from K[unfolded th0 subset_image_iff]
  obtain Sk where Sk: Sk ⊆ Collect (openin U) K = (λS. S ∩ V) ‘ Sk
  by blast
  have ⋃ K = (⋃ Sk) ∩ V
  using Sk by auto
  moreover have openin U (⋃ Sk)
  using Sk by (auto simp: subset_eq)
  ultimately have ?L (⋃ K) by blast
}
ultimately show ?thesis
  unfolding subset_eq mem_Collect_eq istopology_def by auto
qed

```

lemma *openin_subtopology*: $openin (subtopology U V) S \longleftrightarrow (\exists T. openin U T \wedge S = T \cap V)$
unfolding *subtopology_def topology_inverse* [OF *istopology_subtopology*]
by *auto*

lemma *subset_openin_subtopology*:
 $\llbracket openin X S; S \subseteq T \rrbracket \implies openin (subtopology X T) S$
by (*metis inf.orderE openin_subtopology*)

lemma *openin_subtopology_Int*:
 $openin X S \implies openin (subtopology X T) (S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_Int2*:
 $openin X T \implies openin (subtopology X S) (S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_diff_closed*:
 $\llbracket S \subseteq topspace X; closedin X T \rrbracket \implies openin (subtopology X S) (S - T)$
unfolding *closedin_def openin_subtopology*

by (*rule_tac* $x = \text{topspace } X - T$ **in** *exI*) *auto*

lemma *openin_relative_to*: (*openin* X *relative_to* S) = *openin* (*subtopology* X S)
by (*force simp: relative_to_def openin_subtopology*)

lemma *topspace_subtopology* [*simp*]: *topspace* (*subtopology* U V) = *topspace* $U \cap V$
by (*auto simp: topspace_def openin_subtopology*)

lemma *topspace_subtopology_subset*:
 $S \subseteq \text{topspace } X \implies \text{topspace}(\text{subtopology } X S) = S$
by (*simp add: inf.absorb_iff2*)

lemma *closedin_subtopology*: *closedin* (*subtopology* U V) $S \longleftrightarrow (\exists T. \text{closedin } U T \wedge S = T \cap V)$
unfolding *closedin_def topspace_subtopology*
by (*auto simp: openin_subtopology*)

lemma *closedin_subtopology_Int_closed*:
 $\text{closedin } X T \implies \text{closedin}(\text{subtopology } X S) (S \cap T)$
using *closedin_subtopology inf_commute* **by** *blast*

lemma *closedin_subset_topspace*:
 $\llbracket \text{closedin } X S; S \subseteq T \rrbracket \implies \text{closedin}(\text{subtopology } X T) S$
using *closedin_subtopology* **by** *fastforce*

lemma *closedin_relative_to*:
 $(\text{closedin } X \text{ relative_to } S) = \text{closedin}(\text{subtopology } X S)$
by (*force simp: relative_to_def closedin_subtopology*)

lemma *openin_subtopology_refl*: *openin* (*subtopology* U V) $V \longleftrightarrow V \subseteq \text{topspace } U$
unfolding *openin_subtopology*
by *auto (metis IntD1 in_mono openin_subset)*

lemma *subtopology_trivial_iff*: *subtopology* X $S = \text{trivial_topology} \longleftrightarrow X = \text{trivial_topology} \vee \text{topspace } X \cap S = \{\}$
by (*auto simp flip: null_topspace_iff_trivial*)

lemma *subtopology_subtopology*:
 $\text{subtopology}(\text{subtopology } X S) T = \text{subtopology } X (S \cap T)$

proof –

have *eq*: $\bigwedge T'. (\exists S'. T' = S' \cap T \wedge (\exists T. \text{openin } X T \wedge S' = T \cap S)) = (\exists Sa. T' = Sa \cap (S \cap T) \wedge \text{openin } X Sa)$

by (*metis inf_assoc*)

have *subtopology* (*subtopology* X S) $T = \text{topology } (\lambda Ta. \exists Sa. Ta = Sa \cap T \wedge \text{openin}(\text{subtopology } X S) Sa)$

by (*simp add: subtopology_def*)

also have $\dots = \text{subtopology } X (S \cap T)$

by (simp add: openin_subtopology eq) (simp add: subtopology_def)
 finally show ?thesis .
 qed

lemma openin_subtopology_alt:
 openin (subtopology X U) S \longleftrightarrow S \in ($\lambda T. U \cap T$) ‘Collect (openin X)’
 by (simp add: image_iff inf_commute openin_subtopology)

lemma closedin_subtopology_alt:
 closedin (subtopology X U) S \longleftrightarrow S \in ($\lambda T. U \cap T$) ‘Collect (closedin X)’
 by (simp add: image_iff inf_commute closedin_subtopology)

lemma subtopology_superset:
 assumes UV: topspace U \subseteq V
 shows subtopology U V = U

proof –
 { fix S
 have openin U S if openin U T S = T \cap V for T
 by (metis Int_subset_iff assms inf.orderE openin_subset that)
 then have ($\exists T. \text{openin } U T \wedge S = T \cap V$) \longleftrightarrow openin U S
 by (metis assms inf.orderE inf_assoc openin_subset)
 }
 then show ?thesis
 unfolding topology_eq openin_subtopology by blast
 qed

lemma subtopology_topspace[simp]: subtopology U (topspace U) = U
 by (simp add: subtopology_superset)

lemma subtopology_UNIV[simp]: subtopology U UNIV = U
 by (simp add: subtopology_superset)

lemma subtopology_empty_iff_trivial [simp]: subtopology X {} = trivial_topology
 by (simp add: subtopology_eq_discrete_topology_empty)

lemma subtopology_restrict:
 subtopology X (topspace X \cap S) = subtopology X S
 by (metis subtopology_subtopology subtopology_topspace)

lemma openin_subtopology_empty:
 openin (subtopology U {}) S \longleftrightarrow S = {}
 by (metis Int_empty_right openin_empty openin_subtopology)

lemma closedin_subtopology_empty:
 closedin (subtopology U {}) S \longleftrightarrow S = {}
 by (metis Int_empty_right closedin_empty closedin_subtopology)

lemma closedin_subtopology_refl [simp]:
 closedin (subtopology U X) X \longleftrightarrow X \subseteq topspace U

by (metis closedin_def closedin_topspace inf.absorb_iff2 le_inf_iff topspace_subtopology)

lemma closedin_topspace_empty [simp]: closedin trivial_topology S \longleftrightarrow S = {}
by (simp add: closedin_def)

lemma openin_topspace_empty [simp]:
openin trivial_topology S \longleftrightarrow S = {}
by (simp add: openin_closedin_eq)

lemma openin_imp_subset:
openin (subtopology U S) T \implies T \subseteq S
by (metis Int_iff openin_subtopology subsetI)

lemma closedin_imp_subset:
closedin (subtopology U S) T \implies T \subseteq S
by (simp add: closedin_def)

lemma openin_open_subtopology:
openin X S \implies openin (subtopology X S) T \longleftrightarrow openin X T \wedge T \subseteq S
by (metis inf.orderE openin_Int openin_imp_subset openin_subtopology)

lemma closedin_closed_subtopology:
closedin X S \implies (closedin (subtopology X S) T \longleftrightarrow closedin X T \wedge T \subseteq S)
by (metis closedin_Int closedin_imp_subset closedin_subtopology inf.orderE)

lemma closedin_trans_full:
[[closedin (subtopology X U) S; closedin X U]] \implies closedin X S
using closedin_closed_subtopology by blast

lemma openin_subtopology_Un:
[[openin (subtopology X T) S; openin (subtopology X U) S]]
 \implies openin (subtopology X (T \cup U)) S
by (simp add: openin_subtopology) blast

lemma closedin_subtopology_Un:
[[closedin (subtopology X T) S; closedin (subtopology X U) S]]
 \implies closedin (subtopology X (T \cup U)) S
by (simp add: closedin_subtopology) blast

lemma openin_trans_full:
[[openin (subtopology X U) S; openin X U]] \implies openin X S
by (simp add: openin_open_subtopology)

1.13.4 The canonical topology from the underlying type class

abbreviation euclidean :: 'a::topological_space topology
where euclidean \equiv topology open

abbreviation top_of_set :: 'a::topological_space set \Rightarrow 'a topology

where $top_of_set \equiv subtopology (topology\ open)$

lemma $open_openin$: $open\ S \longleftrightarrow openin\ euclidean\ S$
 by *simp*

declare $open_openin$ [*symmetric*, *simp*]

lemma $topspace_euclidean$ [*simp*]: $topspace\ euclidean = UNIV$
 by (*force simp: topspace_def*)

lemma $topspace_euclidean_subtopology$ [*simp*]: $topspace (top_of_set\ S) = S$
 by (*simp*)

lemma $closed_closedin$: $closed\ S \longleftrightarrow closedin\ euclidean\ S$
 by (*simp add: closed_def closedin_def Compl_eq_Diff_UNIV*)

declare $closed_closedin$ [*symmetric*, *simp*]

lemma $openin_subtopology_self$ [*simp*]: $openin (top_of_set\ S)\ S$
 by (*metis openin_topspace topspace_euclidean_subtopology*)

lemma $euclidean_nontrivial$ [*simp*]: $euclidean \neq trivial_topology$
 by (*simp add: subtopology_eq_discrete_topology_empty*)

The most basic facts about the usual topology and metric on \mathbb{R}

abbreviation $euclideanreal$:: *real topology*
 where $euclideanreal \equiv topology\ open$

1.13.5 Basic "localization" results are handy for connectedness.

lemma $openin_open$: $openin (top_of_set\ U)\ S \longleftrightarrow (\exists T. open\ T \wedge (S = U \cap T))$
 by (*auto simp: openin_subtopology*)

lemma $openin_Int_open$:
 $[[openin (top_of_set\ U)\ S; open\ T]]$
 $\implies openin (top_of_set\ U)\ (S \cap T)$
 by (*metis open_Int Int_assoc openin_open*)

lemma $openin_open_Int$ [*intro*]: $open\ S \implies openin (top_of_set\ U)\ (U \cap S)$
 by (*auto simp: openin_open*)

lemma $open_openin_trans$ [*trans*]:
 $open\ S \implies open\ T \implies T \subseteq S \implies openin (top_of_set\ S)\ T$
 by (*metis Int_absorb1 openin_open_Int*)

lemma $open_subset$: $S \subseteq T \implies open\ S \implies openin (top_of_set\ T)\ S$
 by (*auto simp: openin_open*)

lemma *closedin_closed*: $\text{closedin } (\text{top_of_set } U) S \longleftrightarrow (\exists T. \text{closed } T \wedge S = U \cap T)$

by (*simp add: closedin_subtopology Int_ac*)

lemma *closedin_closed_Int*: $\text{closed } S \implies \text{closedin } (\text{top_of_set } U) (U \cap S)$

by (*metis closedin_closed*)

lemma *closed_subset*: $S \subseteq T \implies \text{closed } S \implies \text{closedin } (\text{top_of_set } T) S$

by (*auto simp: closedin_closed*)

lemma *closedin_closed_subset*:

$\llbracket \text{closedin } (\text{top_of_set } U) V; T \subseteq U; S = V \cap T \rrbracket$

$\implies \text{closedin } (\text{top_of_set } T) S$

by (*metis (no_types, lifting) Int_assoc Int_commute closedin_closed inf.orderE*)

lemma *finite_imp_closedin*:

fixes $S :: 'a::t1_space \text{ set}$

shows $\llbracket \text{finite } S; S \subseteq T \rrbracket \implies \text{closedin } (\text{top_of_set } T) S$

by (*simp add: finite_imp_closed closed_subset*)

lemma *closedin_singleton [simp]*:

fixes $a :: 'a::t1_space$

shows $\text{closedin } (\text{top_of_set } U) \{a\} \longleftrightarrow a \in U$

using *closedin_subset* **by** (*force intro: closed_subset*)

lemma *openin_euclidean_subtopology_iff*:

fixes $S U :: 'a::metric_space \text{ set}$

shows $\text{openin } (\text{top_of_set } U) S \longleftrightarrow$

$S \subseteq U \wedge (\forall x \in S. \exists e > 0. \forall x' \in U. \text{dist } x' x < e \longrightarrow x' \in S)$

(*is ?lhs \longleftrightarrow ?rhs*)

proof

assume *?lhs*

then show *?rhs*

unfolding *openin_open open_dist* **by** *blast*

next

define T **where** $T = \{x. \exists a \in S. \exists d > 0. (\forall y \in U. \text{dist } y a < d \longrightarrow y \in S) \wedge \text{dist } x a < d\}$

have $1: \forall x \in T. \exists e > 0. \forall y. \text{dist } y x < e \longrightarrow y \in T$

unfolding *T_def*

apply *clarsimp*

apply (*rule_tac x=d - dist x a in exI*)

by (*metis add_0_left dist_commute dist_triangle_lt less_diff_eq*)

assume *?rhs* **then have** $2: S = U \cap T$

unfolding *T_def*

by *auto (metis dist_self)*

from $1\ 2$ **show** *?lhs*

unfolding *openin_open open_dist* **by** *fast*

qed

lemma *connected_openin*:

connected S \longleftrightarrow

$\neg(\exists E1 E2. \text{openin } (\text{top_of_set } S) E1 \wedge$
 $\text{openin } (\text{top_of_set } S) E2 \wedge$
 $S \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$

unfolding *connected_def openin_open disjoint_iff_not_equal* **by** *blast*

lemma *connected_openin_eq*:

connected S \longleftrightarrow

$\neg(\exists E1 E2. \text{openin } (\text{top_of_set } S) E1 \wedge$
 $\text{openin } (\text{top_of_set } S) E2 \wedge$
 $E1 \cup E2 = S \wedge E1 \cap E2 = \{\} \wedge$
 $E1 \neq \{\} \wedge E2 \neq \{\})$

unfolding *connected_openin*

by (*metis* (*no_types*, *lifting*) *Un_subset_iff openin_imp_subset subset_antisym*)

lemma *connected_closedin*:

connected S \longleftrightarrow

$(\nexists E1 E2.$
 $\text{closedin } (\text{top_of_set } S) E1 \wedge$
 $\text{closedin } (\text{top_of_set } S) E2 \wedge$
 $S \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$
(is ?lhs = ?rhs)

proof

assume *?lhs*

then show *?rhs*

by (*auto simp add: connected_closed closedin_closed*)

next

assume *R: ?rhs*

then show *?lhs*

proof (*clarsimp simp add: connected_closed closedin_closed*)

fix *A B*

assume *s_sub*: $S \subseteq A \cup B$ $B \cap S \neq \{\}$

and *disj*: $A \cap B \cap S = \{\}$

and *cl*: *closed A closed B*

have $S - A = B \cap S$

using *Diff_subset_conv Un_Diff_Int disj s_sub(1)* **by** *auto*

then show $A \cap S = \{\}$

by (*metis* *Int_Diff_Un Int_Diff_disjoint R cl closedin_closed_Int dual_order.refl inf_commute s_sub(2)*)

qed

qed

lemma *connected_closedin_eq*:

connected S \longleftrightarrow

$\neg(\exists E1 E2.$
 $\text{closedin } (\text{top_of_set } S) E1 \wedge$
 $\text{closedin } (\text{top_of_set } S) E2 \wedge$

$$E1 \cup E2 = S \wedge E1 \cap E2 = \{\} \wedge \\ E1 \neq \{\} \wedge E2 \neq \{\}$$

unfolding *connected_closedin*
by (*metis Un_subset_iff closedin_imp_subset subset_antisym*)

These "transitivity" results are handy too

lemma *openin_trans[trans]*:
 $openin (top_of_set T) S \implies openin (top_of_set U) T \implies$
 $openin (top_of_set U) S$
by (*metis openin_Int_open openin_open*)

lemma *openin_open_trans*: $openin (top_of_set T) S \implies open T \implies open S$
by (*auto simp: openin_open intro: openin_trans*)

lemma *closedin_trans[trans]*:
 $closedin (top_of_set T) S \implies closedin (top_of_set U) T \implies$
 $closedin (top_of_set U) S$
by (*auto simp: closedin_closed closed_Inter Int_assoc*)

lemma *closedin_closed_trans*: $closedin (top_of_set T) S \implies closed T \implies closed S$
by (*auto simp: closedin_closed intro: closedin_trans*)

lemma *openin_subtopology_Int_subset*:
 $\llbracket openin (top_of_set u) (u \cap S); v \subseteq u \rrbracket \implies openin (top_of_set v) (v \cap S)$
by (*auto simp: openin_subtopology*)

lemma *openin_open_eq*: $open s \implies (openin (top_of_set s) t \longleftrightarrow open t \wedge t \subseteq s)$
using *open_subset openin_open_trans openin_subset* **by** *fastforce*

1.13.6 Derived set (set of limit points)

definition *derived_set_of* :: $'a topology \Rightarrow 'a set \Rightarrow 'a set$ (**infixl** *derived'_set'_of* 80)

where $X \text{ derived_set_of } S \equiv$
 $\{x \in \text{topspace } X.$
 $(\forall T. x \in T \wedge openin X T \longrightarrow (\exists y \neq x. y \in S \wedge y \in T))\}$

lemma *derived_set_of_restrict [simp]*:
 $X \text{ derived_set_of } (\text{topspace } X \cap S) = X \text{ derived_set_of } S$
by (*simp add: derived_set_of_def*) (*metis openin_subset subset_iff*)

lemma *in_derived_set_of*:
 $x \in X \text{ derived_set_of } S \longleftrightarrow x \in \text{topspace } X \wedge (\forall T. x \in T \wedge openin X T \longrightarrow$
 $(\exists y \neq x. y \in S \wedge y \in T))$
by (*simp add: derived_set_of_def*)

lemma *derived_set_of_subset_topspace*:

X derived_set_of $S \subseteq \text{topspace } X$
by (auto simp add: derived_set_of_def)

lemma derived_set_of_subtopology:
 $(\text{subtopology } X \ U) \text{ derived_set_of } S = U \cap (X \text{ derived_set_of } (U \cap S))$
by (simp add: derived_set_of_def openin_subtopology) blast

lemma derived_set_of_subset_subtopology:
 $(\text{subtopology } X \ S) \text{ derived_set_of } T \subseteq S$
by (simp add: derived_set_of_subtopology)

lemma derived_set_of_empty [simp]: $X \text{ derived_set_of } \{\} = \{\}$
by (auto simp: derived_set_of_def)

lemma derived_set_of_mono:
 $S \subseteq T \implies X \text{ derived_set_of } S \subseteq X \text{ derived_set_of } T$
unfolding derived_set_of_def **by** blast

lemma derived_set_of_Un:
 $X \text{ derived_set_of } (S \cup T) = X \text{ derived_set_of } S \cup X \text{ derived_set_of } T$ (is
 ?lhs = ?rhs)

proof
show ?lhs \subseteq ?rhs
by (clarsimp simp: in_derived_set_of) (metis IntE IntI openin_Int)
show ?rhs \subseteq ?lhs
by (simp add: derived_set_of_mono)
qed

lemma derived_set_of_Union:
 $\text{finite } \mathcal{F} \implies X \text{ derived_set_of } (\bigcup \mathcal{F}) = (\bigcup S \in \mathcal{F}. X \text{ derived_set_of } S)$
proof (induction \mathcal{F} rule: finite_induct)
case (insert $S \ \mathcal{F}$)
then show ?case
by (simp add: derived_set_of_Un)
qed auto

lemma derived_set_of_topspace:
 $X \text{ derived_set_of } (\text{topspace } X) = \{x \in \text{topspace } X. \neg \text{openin } X \ \{x\}\}$ (is ?lhs =
 ?rhs)
proof
show ?lhs \subseteq ?rhs
by (auto simp: in_derived_set_of)
show ?rhs \subseteq ?lhs
by (clarsimp simp: in_derived_set_of) (metis openin_closedin_eq openin_subopen
 singletonD subset_eq)
qed

lemma discrete_topology_unique_derived_set:
 $\text{discrete_topology } U = X \iff \text{topspace } X = U \wedge X \text{ derived_set_of } U = \{\}$

by (auto simp: discrete_topology_unique_derived_set_of_topspace)

lemma *subtopology_eq_discrete_topology_eq*:

subtopology X U = discrete_topology U \longleftrightarrow *U* \subseteq *topspace X* \wedge *U* \cap *X* *derived_set_of U* = {}

using *discrete_topology_unique_derived_set* [of *U* *subtopology X U*]

by (auto simp: eq_commute *derived_set_of_subtopology*)

lemma *subtopology_eq_discrete_topology*:

S \subseteq *topspace X* \wedge *S* \cap *X* *derived_set_of S* = {}

\implies *subtopology X S* = *discrete_topology S*

by (*simp add: subtopology_eq_discrete_topology_eq*)

lemma *subtopology_eq_discrete_topology_gen*:

assumes *S* \cap *X* *derived_set_of S* = {}

shows *subtopology X S* = *discrete_topology(topspace X* \cap *S)*

proof –

have *subtopology X S* = *subtopology X* (*topspace X* \cap *S*)

by (*simp add: subtopology_restrict*)

then show ?thesis

using *assms* by (*simp add: inf.assoc subtopology_eq_discrete_topology_eq*)

qed

lemma *subtopology_discrete_topology* [*simp*]:

subtopology (*discrete_topology U*) *S* = *discrete_topology(U* \cap *S)*

proof –

have $(\lambda T. \exists Sa. T = Sa \cap S \wedge Sa \subseteq U) = (\lambda Sa. Sa \subseteq U \wedge Sa \subseteq S)$

by *force*

then show ?thesis

by (*simp add: subtopology_def*) (*simp add: discrete_topology_def*)

qed

lemma *openin_Int_derived_set_of_subset*:

openin X S \implies *S* \cap *X* *derived_set_of T* \subseteq *X* *derived_set_of* (*S* \cap *T*)

by (auto simp: *derived_set_of_def*)

lemma *openin_Int_derived_set_of_eq*:

assumes *openin X S*

shows *S* \cap *X* *derived_set_of T* = *S* \cap *X* *derived_set_of* (*S* \cap *T*) (is ?lhs = ?rhs)

proof

show ?lhs \subseteq ?rhs

by (*simp add: assms openin_Int_derived_set_of_subset*)

show ?rhs \subseteq ?lhs

by (*metis derived_set_of_mono inf_commute inf_le1 inf_mono order_refl*)

qed

1.13.7 Closure with respect to a topological space

definition *closure_of* :: 'a topology \Rightarrow 'a set \Rightarrow 'a set (**infixr** *closure'_of* 80)
where $X \text{ closure_of } S \equiv \{x \in \text{topspace } X. \forall T. x \in T \wedge \text{openin } X T \longrightarrow (\exists y \in S. y \in T)\}$

lemma *closure_of_restrict*: $X \text{ closure_of } S = X \text{ closure_of } (\text{topspace } X \cap S)$
unfolding *closure_of_def*
using *openin_subset* **by** *blast*

lemma *in_closure_of*:
 $x \in X \text{ closure_of } S \iff$
 $x \in \text{topspace } X \wedge (\forall T. x \in T \wedge \text{openin } X T \longrightarrow (\exists y. y \in S \wedge y \in T))$
by (*auto simp: closure_of_def*)

lemma *closure_of*: $X \text{ closure_of } S = \text{topspace } X \cap (S \cup X \text{ derived_set_of } S)$
by (*fastforce simp: in_closure_of in_derived_set_of*)

lemma *closure_of_alt*: $X \text{ closure_of } S = \text{topspace } X \cap S \cup X \text{ derived_set_of } S$
using *derived_set_of_subset_topspace* [*of X S*]
unfolding *closure_of_def in_derived_set_of*
by *safe (auto simp: in_derived_set_of)*

lemma *derived_set_of_subset_closure_of*:
 $X \text{ derived_set_of } S \subseteq X \text{ closure_of } S$
by (*fastforce simp: closure_of_def in_derived_set_of*)

lemma *closure_of_subtopology*:
 $(\text{subtopology } X U) \text{ closure_of } S = U \cap (X \text{ closure_of } (U \cap S))$
unfolding *closure_of_def topspace_subtopology openin_subtopology*
by *safe (metis (full_types) IntI Int_iff inf.commute)+*

lemma *closure_of_empty* [*simp*]: $X \text{ closure_of } \{\} = \{\}$
by (*simp add: closure_of_alt*)

lemma *closure_of_topspace* [*simp*]: $X \text{ closure_of } \text{topspace } X = \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_UNIV* [*simp*]: $X \text{ closure_of } \text{UNIV} = \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_subset_topspace*: $X \text{ closure_of } S \subseteq \text{topspace } X$
by (*simp add: closure_of*)

lemma *closure_of_subset_subtopology*: $(\text{subtopology } X S) \text{ closure_of } T \subseteq S$
by (*simp add: closure_of_subtopology*)

lemma *closure_of_mono*: $S \subseteq T \implies X \text{ closure_of } S \subseteq X \text{ closure_of } T$
by (*fastforce simp add: closure_of_def*)

lemma *closure_of_subtopology_subset*:

(*subtopology X U*) *closure_of S* \subseteq (*X closure_of S*)

unfolding *closure_of_subtopology*

by *clarsimp (meson closure_of_mono contra_subsetD inf.cobounded2)*

lemma *closure_of_subtopology_mono*:

$T \subseteq U \implies$ (*subtopology X T*) *closure_of S* \subseteq (*subtopology X U*) *closure_of S*

unfolding *closure_of_subtopology*

by *auto (meson closure_of_mono inf_mono subset_iff)*

lemma *closure_of_Un [simp]*: $X \text{ closure_of } (S \cup T) = X \text{ closure_of } S \cup X \text{ closure_of } T$

by (*simp add: Un_assoc Un_left_commute closure_of_alt derived_set_of_Un inf_sup_distrib1*)

lemma *closure_of_Union*:

finite F $\implies X \text{ closure_of } (\bigcup \mathcal{F}) = \bigcup S \in \mathcal{F}. X \text{ closure_of } S$

by (*induction F rule: finite_induct*) *auto*

lemma *closure_of_subset*: $S \subseteq \text{topspace } X \implies S \subseteq X \text{ closure_of } S$

by (*auto simp: closure_of_def*)

lemma *closure_of_subset_Int*: $\text{topspace } X \cap S \subseteq X \text{ closure_of } S$

by (*auto simp: closure_of_def*)

lemma *closure_of_subset_eq*: $S \subseteq \text{topspace } X \wedge X \text{ closure_of } S \subseteq S \iff \text{closedin } X S$

proof –

have *openin X (topspace X - S)*

if $\bigwedge x. \llbracket x \in \text{topspace } X; \forall T. x \in T \wedge \text{openin } X T \longrightarrow S \cap T \neq \{\} \rrbracket \implies x \in S$

apply (*subst openin_subopen*)

by (*metis Diff_iff Diff_mono Diff_triv inf commute openin_subset order_refl that*)

then show *?thesis*

by (*auto simp: closedin_def closure_of_def disjoint_iff_not_equal*)

qed

lemma *closure_of_eq*: $X \text{ closure_of } S = S \iff \text{closedin } X S$

by (*metis closure_of_subset closure_of_subset_eq closure_of_subset_topspace subset_antisym*)

lemma *closedin_contains_derived_set*:

$\text{closedin } X S \iff X \text{ derived_set_of } S \subseteq S \wedge S \subseteq \text{topspace } X$

proof (*intro iffI conjI*)

show $\text{closedin } X S \implies X \text{ derived_set_of } S \subseteq S$

using *closure_of_eq derived_set_of_subset_closure_of* **by** *fastforce*

show $\text{closedin } X S \implies S \subseteq \text{topspace } X$

using *closedin_subset* **by** *blast*

show $X \text{ derived_set_of } S \subseteq S \wedge S \subseteq \text{topspace } X \implies \text{closedin } X S$

by (metis closure_of_closure_of_eq inf.absorb_iff2 sup.orderE)
qed

lemma *derived_set_subset_gen*:

$X \text{ derived_set_of } S \subseteq S \longleftrightarrow \text{closedin } X (\text{topspace } X \cap S)$

by (simp add: closedin_contains_derived_set derived_set_of_subset_topspace)

lemma *derived_set_subset*: $S \subseteq \text{topspace } X \implies (X \text{ derived_set_of } S \subseteq S \longleftrightarrow \text{closedin } X S)$

by (simp add: closedin_contains_derived_set)

lemma *closedin_derived_set*:

$\text{closedin } (\text{subtopology } X T) S \longleftrightarrow$

$S \subseteq \text{topspace } X \wedge S \subseteq T \wedge (\forall x. x \in X \text{ derived_set_of } S \wedge x \in T \longrightarrow x \in S)$

by (auto simp: closedin_contains_derived_set derived_set_of_subtopology Int_absorb1)

lemma *closedin_Int_closure_of*:

$\text{closedin } (\text{subtopology } X S) T \longleftrightarrow S \cap X \text{ closure_of } T = T$

by (metis Int_left_absorb closure_of_eq closure_of_subtopology)

lemma *closure_of_closedin*: $\text{closedin } X S \implies X \text{ closure_of } S = S$

by (simp add: closure_of_eq)

lemma *closure_of_eq_diff*: $X \text{ closure_of } S = \text{topspace } X - \bigcup \{T. \text{openin } X T \wedge \text{disjnt } S T\}$

by (auto simp: closure_of_def disjnt_iff)

lemma *closedin_closure_of* [simp]: $\text{closedin } X (X \text{ closure_of } S)$

unfolding closure_of_eq_diff by blast

lemma *closure_of_closure_of* [simp]: $X \text{ closure_of } (X \text{ closure_of } S) = X \text{ closure_of } S$

by (simp add: closure_of_eq)

lemma *closure_of_hull*:

assumes $S \subseteq \text{topspace } X$ shows $X \text{ closure_of } S = (\text{closedin } X) \text{ hull } S$

by (metis assms closedin_closure_of closure_of_eq closure_of_mono closure_of_subset hull_unique)

lemma *closure_of_minimal*:

$\llbracket S \subseteq T; \text{closedin } X T \rrbracket \implies (X \text{ closure_of } S) \subseteq T$

by (metis closure_of_eq closure_of_mono)

lemma *closure_of_minimal_eq*:

$\llbracket S \subseteq \text{topspace } X; \text{closedin } X T \rrbracket \implies (X \text{ closure_of } S) \subseteq T \longleftrightarrow S \subseteq T$

by (meson closure_of_minimal closure_of_subset subset_trans)

lemma *closure_of_unique*:

$$\begin{aligned} & \llbracket S \subseteq T; \text{closedin } X T; \\ & \quad \bigwedge T'. \llbracket S \subseteq T'; \text{closedin } X T' \rrbracket \implies T \subseteq T' \rrbracket \\ & \implies X \text{ closure_of } S = T \end{aligned}$$
by (*meson closedin_closure_of closedin_subset closure_of_minimal closure_of_subset eq_iff order.trans*)

lemma *closure_of_eq_empty_gen*: $X \text{ closure_of } S = \{\} \longleftrightarrow \text{disjnt } (\text{topspace } X) S$

unfolding *disjnt_def closure_of_restrict* [**where** $S=S$]
using *closure_of* **by** *fastforce*

lemma *closure_of_eq_empty*: $S \subseteq \text{topspace } X \implies X \text{ closure_of } S = \{\} \longleftrightarrow S = \{\}$

using *closure_of_subset* **by** *fastforce*

lemma *openin_Int_closure_of_subset*:

assumes *openin* $X S$

shows $S \cap X \text{ closure_of } T \subseteq X \text{ closure_of } (S \cap T)$

proof –

have $S \cap X \text{ derived_set_of } T = S \cap X \text{ derived_set_of } (S \cap T)$

by (*meson assms openin_Int_derived_set_of_eq*)

moreover have $S \cap (S \cap T) = S \cap T$

by *fastforce*

ultimately show *?thesis*

by (*metis closure_of_alt inf.cobounded2 inf_left_commute inf_sup_distrib1*)

qed

lemma *closure_of_openin_Int_closure_of*:

assumes *openin* $X S$

shows $X \text{ closure_of } (S \cap X \text{ closure_of } T) = X \text{ closure_of } (S \cap T)$

proof

show $X \text{ closure_of } (S \cap X \text{ closure_of } T) \subseteq X \text{ closure_of } (S \cap T)$

by (*simp add: assms closure_of_minimal openin_Int_closure_of_subset*)

next

show $X \text{ closure_of } (S \cap T) \subseteq X \text{ closure_of } (S \cap X \text{ closure_of } T)$

by (*metis Int_subset_iff assms closure_of_alt closure_of_mono inf_mono openin_subset subset_refl sup.coboundedI1*)

qed

lemma *openin_Int_closure_of_eq*:

assumes *openin* $X S$ **shows** $S \cap X \text{ closure_of } T = S \cap X \text{ closure_of } (S \cap T)$
(is *?lhs = ?rhs*)

proof

show *?lhs* \subseteq *?rhs*

by (*simp add: assms openin_Int_closure_of_subset*)

show *?rhs* \subseteq *?lhs*

by (*metis closure_of_mono inf_commute inf_le1 inf_mono order_refl*)

qed

lemma *openin_Int_closure_of_eq_empty:*

assumes *openin X S* **shows** $S \cap X \text{ closure_of } T = \{\}$ \longleftrightarrow $S \cap T = \{\}$ (**is** $?lhs = ?rhs$)

proof

show $?lhs \implies ?rhs$

unfolding *disjoint_iff*

by (*meson assms in_closure_of_in_mono openin_subset*)

show $?rhs \implies ?lhs$

by (*simp add: assms openin_Int_closure_of_eq*)

qed

lemma *closure_of_openin_Int_superset:*

openin X S $\wedge S \subseteq X \text{ closure_of } T$

$\implies X \text{ closure_of } (S \cap T) = X \text{ closure_of } S$

by (*metis closure_of_openin_Int_closure_of_inf.orderE*)

lemma *closure_of_openin_subtopology_Int_closure_of:*

assumes *S: openin (subtopology X U) S* **and** $T \subseteq U$

shows $X \text{ closure_of } (S \cap X \text{ closure_of } T) = X \text{ closure_of } (S \cap T)$ (**is** $?lhs = ?rhs$)

proof

obtain *S0* **where** *S0: openin X S0* $S = S0 \cap U$

using *assms* **by** (*auto simp: openin_subtopology*)

then show $?lhs \subseteq ?rhs$

proof –

have $S0 \cap X \text{ closure_of } T = S0 \cap X \text{ closure_of } (S0 \cap T)$

by (*meson S0(1) openin_Int_closure_of_eq*)

moreover have $S0 \cap T = S0 \cap U \cap T$

using $\langle T \subseteq U \rangle$ **by** *fastforce*

ultimately have $S \cap X \text{ closure_of } T \subseteq X \text{ closure_of } (S \cap T)$

using *S0(2)* **by** *auto*

then show *thesis*

by (*meson closedin_closure_of_closure_of_minimal*)

qed

next

show $?rhs \subseteq ?lhs$

proof –

have $T \cap S \subseteq T \cup X \text{ derived_set_of } T$

by *force*

then show *thesis*

by (*smt (verit, del_insts) Int_iff in_closure_of_inf.orderE openin_subset subsetI*)

qed

qed

lemma *closure_of_subtopology_open:*

openin X U $\vee S \subseteq U \implies (\text{subtopology X U}) \text{ closure_of } S = U \cap X \text{ closure_of } S$

by (*metis closure_of_subtopology_inf_absorb2 openin_Int_closure_of_eq*)

lemma *discrete_topology_closure_of*:
 (*discrete_topology U*) *closure_of S* = $U \cap S$
by (*metis closedin_discrete_topology closure_of_restrict closure_of_unique discrete_topology_unique inf_sup_ord(1) order_refl*)

Interior with respect to a topological space.

definition *interior_of* :: 'a topology \Rightarrow 'a set \Rightarrow 'a set (**infixr** *interior'_of* 80)
where $X \text{ interior_of } S \equiv \{x. \exists T. \text{openin } X T \wedge x \in T \wedge T \subseteq S\}$

lemma *interior_of_restrict*:
 $X \text{ interior_of } S = X \text{ interior_of } (\text{topspace } X \cap S)$
using *openin_subset* **by** (*auto simp: interior_of_def*)

lemma *interior_of_eq*: $(X \text{ interior_of } S = S) \longleftrightarrow \text{openin } X S$
unfolding *interior_of_def* **using** *openin_subopen* **by** *blast*

lemma *interior_of_openin*: $\text{openin } X S \Longrightarrow X \text{ interior_of } S = S$
by (*simp add: interior_of_eq*)

lemma *interior_of_empty* [*simp*]: $X \text{ interior_of } \{\} = \{\}$
by (*simp add: interior_of_eq*)

lemma *interior_of_topspace* [*simp*]: $X \text{ interior_of } (\text{topspace } X) = \text{topspace } X$
by (*simp add: interior_of_eq*)

lemma *openin_interior_of* [*simp*]: $\text{openin } X (X \text{ interior_of } S)$
unfolding *interior_of_def*
using *openin_subopen* **by** *fastforce*

lemma *interior_of_interior_of* [*simp*]:
 $X \text{ interior_of } X \text{ interior_of } S = X \text{ interior_of } S$
by (*simp add: interior_of_eq*)

lemma *interior_of_subset*: $X \text{ interior_of } S \subseteq S$
by (*auto simp: interior_of_def*)

lemma *interior_of_subset_closure_of*: $X \text{ interior_of } S \subseteq X \text{ closure_of } S$
by (*metis closure_of_subset_Int dual_order.trans interior_of_restrict interior_of_subset*)

lemma *subset_interior_of_eq*: $S \subseteq X \text{ interior_of } S \longleftrightarrow \text{openin } X S$
by (*metis interior_of_eq interior_of_subset subset_antisym*)

lemma *interior_of_mono*: $S \subseteq T \Longrightarrow X \text{ interior_of } S \subseteq X \text{ interior_of } T$
by (*auto simp: interior_of_def*)

lemma *interior_of_maximal*: $[[T \subseteq S; \text{openin } X T]] \Longrightarrow T \subseteq X \text{ interior_of } S$
by (*auto simp: interior_of_def*)

lemma *interior_of_maximal_eq*: $\text{openin } X \ T \implies T \subseteq X \ \text{interior_of } S \iff T \subseteq S$

by (*meson interior_of_maximal interior_of_subset order_trans*)

lemma *interior_of_unique*:

$\llbracket T \subseteq S; \text{openin } X \ T; \bigwedge T'. \llbracket T' \subseteq S; \text{openin } X \ T' \rrbracket \implies T' \subseteq T \rrbracket \implies X \ \text{interior_of } S = T$

by (*simp add: interior_of_maximal_eq interior_of_subset subset_antisym*)

lemma *interior_of_subset_topspace*: $X \ \text{interior_of } S \subseteq \text{topspace } X$

by (*simp add: openin_subset*)

lemma *interior_of_subset_subtopology*: $(\text{subtopology } X \ S) \ \text{interior_of } T \subseteq S$

by (*meson openin_imp_subset openin_interior_of*)

lemma *interior_of_Int*: $X \ \text{interior_of } (S \cap T) = X \ \text{interior_of } S \cap X \ \text{interior_of } T$ (*is ?lhs = ?rhs*)

proof

show $?lhs \subseteq ?rhs$

by (*simp add: interior_of_mono*)

show $?rhs \subseteq ?lhs$

by (*meson inf_mono interior_of_maximal interior_of_subset openin_Int openin_interior_of*)

qed

lemma *interior_of_Inter_subset*: $X \ \text{interior_of } (\bigcap \mathcal{F}) \subseteq (\bigcap S \in \mathcal{F}. X \ \text{interior_of } S)$

by (*simp add: INT_greatest Inf_lower interior_of_mono*)

lemma *union_interior_of_subset*:

$X \ \text{interior_of } S \cup X \ \text{interior_of } T \subseteq X \ \text{interior_of } (S \cup T)$

by (*simp add: interior_of_mono*)

lemma *interior_of_eq_empty*:

$X \ \text{interior_of } S = \{\} \iff (\forall T. \text{openin } X \ T \wedge T \subseteq S \longrightarrow T = \{\})$

by (*metis bot.extremum_uniqueI interior_of_maximal interior_of_subset openin_interior_of*)

lemma *interior_of_eq_empty_alt*:

$X \ \text{interior_of } S = \{\} \iff (\forall T. \text{openin } X \ T \wedge T \neq \{\} \longrightarrow T - S \neq \{\})$

by (*auto simp: interior_of_eq_empty*)

lemma *interior_of_Union_openin_subsets*:

$\bigcup \{T. \text{openin } X \ T \wedge T \subseteq S\} = X \ \text{interior_of } S$

by (*rule interior_of_unique [symmetric] auto*)

lemma *interior_of_complement*:

$X \ \text{interior_of } (\text{topspace } X - S) = \text{topspace } X - X \ \text{closure_of } S$

by (*auto simp: interior_of_def closure_of_def*)

lemma *interior_of_closure_of*:

$X \text{ interior_of } S = \text{topspace } X - X \text{ closure_of } (\text{topspace } X - S)$

unfolding *interior_of_complement* [*symmetric*]

by (*metis Diff_Diff_Int interior_of_restrict*)

lemma *closure_of_interior_of*:

$X \text{ closure_of } S = \text{topspace } X - X \text{ interior_of } (\text{topspace } X - S)$

by (*simp add: interior_of_complement Diff_Diff_Int closure_of*)

lemma *closure_of_complement*: $X \text{ closure_of } (\text{topspace } X - S) = \text{topspace } X - X \text{ interior_of } S$

unfolding *interior_of_def closure_of_def*

by (*blast dest: openin_subset*)

lemma *interior_of_eq_empty_complement*:

$X \text{ interior_of } S = \{\} \iff X \text{ closure_of } (\text{topspace } X - S) = \text{topspace } X$

using *interior_of_subset_topspace* [*of X S*] *closure_of_complement* **by** *fastforce*

lemma *closure_of_eq_topspace*:

$X \text{ closure_of } S = \text{topspace } X \iff X \text{ interior_of } (\text{topspace } X - S) = \{\}$

using *closure_of_subset_topspace* [*of X S*] *interior_of_complement* **by** *fastforce*

lemma *interior_of_subtopology_subset*:

$U \cap X \text{ interior_of } S \subseteq (\text{subtopology } X U) \text{ interior_of } S$

by (*auto simp: interior_of_def openin_subtopology*)

lemma *interior_of_subtopology_subsets*:

$T \subseteq U \implies T \cap (\text{subtopology } X U) \text{ interior_of } S \subseteq (\text{subtopology } X T) \text{ interior_of } S$

by (*metis inf.absorb_iff2 interior_of_subtopology_subset subtopology_subtopology*)

lemma *interior_of_subtopology_mono*:

$\llbracket S \subseteq T; T \subseteq U \rrbracket \implies (\text{subtopology } X U) \text{ interior_of } S \subseteq (\text{subtopology } X T) \text{ interior_of } S$

by (*metis dual_order.trans inf.orderE inf_commute interior_of_subset interior_of_subtopology_subsets*)

lemma *interior_of_subtopology_open*:

assumes *openin X U*

shows $(\text{subtopology } X U) \text{ interior_of } S = U \cap X \text{ interior_of } S$ (**is** *?lhs = ?rhs*)

proof

show *?lhs* \subseteq *?rhs*

by (*meson assms interior_of_maximal interior_of_subset le_infI openin_interior_of openin_open_subtopology*)

show *?rhs* \subseteq *?lhs*

by (*simp add: interior_of_subtopology_subset*)

qed

lemma *dense_intersects_open*:

$X \text{ closure_of } S = \text{topspace } X \iff (\forall T. \text{openin } X T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$

proof –

have $X \text{ closure_of } S = \text{topspace } X \iff (\text{topspace } X - X \text{ interior_of } (\text{topspace } X - S) = \text{topspace } X)$

by (*simp add: closure_of_interior_of*)

also have $\dots \iff X \text{ interior_of } (\text{topspace } X - S) = \{\}$

by (*simp add: closure_of_complement interior_of_eq_empty_complement*)

also have $\dots \iff (\forall T. \text{openin } X T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$

unfolding *interior_of_eq_empty_alt*

using *openin_subset* **by** *fastforce*

finally show *?thesis* .

qed

lemma *interior_of_closedin_union_empty_interior_of*:

assumes *closedin* $X S$ **and** *disj*: $X \text{ interior_of } T = \{\}$

shows $X \text{ interior_of } (S \cup T) = X \text{ interior_of } S$

proof –

have $X \text{ closure_of } (\text{topspace } X - T) = \text{topspace } X$

by (*metis Diff_Diff_Int disj closure_of_eq_topspace closure_of_restrict interior_of_closure_of*)

then show *?thesis*

unfolding *interior_of_closure_of*

by (*metis Diff_Un Diff_subset assms(1) closedin_def closure_of_openin_Int_superset*)

qed

lemma *interior_of_union_eq_empty*:

closedin $X S$

$\implies (X \text{ interior_of } (S \cup T) = \{\} \iff$

$X \text{ interior_of } S = \{\} \wedge X \text{ interior_of } T = \{\})$

by (*metis interior_of_closedin_union_empty_interior_of le_sup_iff subset_empty union_interior_of_subset*)

lemma *discrete_topology_interior_of* [*simp*]:

(*discrete_topology* U) $\text{interior_of } S = U \cap S$

by (*simp add: interior_of_restrict [of _ S] interior_of_eq*)

1.13.8 Frontier with respect to topological space

definition *frontier_of* :: 'a topology \Rightarrow 'a set \Rightarrow 'a set (*infixr frontier'_of* 80)

where $X \text{ frontier_of } S \equiv X \text{ closure_of } S - X \text{ interior_of } S$

lemma *frontier_of_closures*:

$X \text{ frontier_of } S = X \text{ closure_of } S \cap X \text{ closure_of } (\text{topspace } X - S)$

by (*metis Diff_Diff_Int closure_of_complement closure_of_subset_topspace double_diff frontier_of_def interior_of_subset_closure_of*)

lemma *interior_of_union_frontier_of* [*simp*]:

$X \text{ interior_of } S \cup X \text{ frontier_of } S = X \text{ closure_of } S$

by (simp add: frontier_of_def interior_of_subset_closure_of subset_antisym)

lemma frontier_of_restrict: $X \text{ frontier_of } S = X \text{ frontier_of } (\text{topspace } X \cap S)$
 by (metis closure_of_restrict frontier_of_def interior_of_restrict)

lemma closedin_frontier_of: $\text{closedin } X (X \text{ frontier_of } S)$
 by (simp add: closedin_Int frontier_of_closures)

lemma frontier_of_subset_topspace: $X \text{ frontier_of } S \subseteq \text{topspace } X$
 by (simp add: closedin_frontier_of closedin_subset)

lemma frontier_of_subset_subtopology: $(\text{subtopology } X S) \text{ frontier_of } T \subseteq S$
 by (metis (no_types) closedin_derived_set closedin_frontier_of)

lemma frontier_of_subtopology_subset:
 $U \cap (\text{subtopology } X U) \text{ frontier_of } S \subseteq (X \text{ frontier_of } S)$

proof –

have $U \cap X \text{ interior_of } S - \text{subtopology } X U \text{ interior_of } S = \{\}$

by (simp add: interior_of_subtopology_subset)

moreover have $X \text{ closure_of } S \cap \text{subtopology } X U \text{ closure_of } S = \text{subtopology } X U \text{ closure_of } S$

by (meson closure_of_subtopology_subset inf.absorb_iff2)

ultimately show ?thesis

unfolding frontier_of_def

by blast

qed

lemma frontier_of_subtopology_mono:
 $\llbracket S \subseteq T; T \subseteq U \rrbracket \implies (\text{subtopology } X T) \text{ frontier_of } S \subseteq (\text{subtopology } X U) \text{ frontier_of } S$

by (simp add: frontier_of_def Diff_mono closure_of_subtopology_mono interior_of_subtopology_mono)

lemma clopenin_eq_frontier_of:

$\text{closedin } X S \wedge \text{openin } X S \iff S \subseteq \text{topspace } X \wedge X \text{ frontier_of } S = \{\}$

proof (cases $S \subseteq \text{topspace } X$)

case True

then show ?thesis

by (metis Diff_eq_empty_iff closure_of_eq closure_of_subset_eq frontier_of_def interior_of_eq interior_of_subset interior_of_union_frontier_of sup_bot_right)

next

case False

then show ?thesis

by (simp add: frontier_of_closures openin_closedin_eq)

qed

lemma frontier_of_eq_empty:

$S \subseteq \text{topspace } X \implies (X \text{ frontier_of } S = \{\}) \iff \text{closedin } X S \wedge \text{openin } X S$

by (simp add: clopenin_eq_frontier_of)

lemma *frontier_of_openin*:

openin $X S \implies X \text{frontier_of } S = X \text{closure_of } S - S$

by (*metis* (*no_types*) *frontier_of_def interior_of_eq*)

lemma *frontier_of_openin_straddle_Int*:

assumes *openin* $X U U \cap X \text{frontier_of } S \neq \{\}$

shows $U \cap S \neq \{\}$ $U - S \neq \{\}$

proof –

have $U \cap (X \text{closure_of } S \cap X \text{closure_of } (\text{topspace } X - S)) \neq \{\}$

using *assms* **by** (*simp* *add*: *frontier_of_closures*)

then show $U \cap S \neq \{\}$

using *assms* *openin_Int_closure_of_eq_empty* **by** *fastforce*

show $U - S \neq \{\}$

proof –

have $\exists A. X \text{closure_of } (A - S) \cap U \neq \{\}$

using $\langle U \cap (X \text{closure_of } S \cap X \text{closure_of } (\text{topspace } X - S)) \neq \{\} \rangle$ **by**

blast

then have $\neg U \subseteq S$

by (*metis* *Diff_disjoint Diff_eq_empty_iff Int_Diff assms*(1) *inf_commute* *openin_Int_closure_of_eq_empty*)

then show *?thesis*

by *blast*

qed

qed

lemma *frontier_of_subset_closedin*: *closedin* $X S \implies (X \text{frontier_of } S) \subseteq S$

using *closure_of_eq frontier_of_def* **by** *fastforce*

lemma *frontier_of_empty* [*simp*]: $X \text{frontier_of } \{\} = \{\}$

by (*simp* *add*: *frontier_of_def*)

lemma *frontier_of_topspace* [*simp*]: $X \text{frontier_of } \text{topspace } X = \{\}$

by (*simp* *add*: *frontier_of_def*)

lemma *frontier_of_subset_eq*:

assumes $S \subseteq \text{topspace } X$

shows $(X \text{frontier_of } S) \subseteq S \longleftrightarrow \text{closedin } X S$

proof

show $X \text{frontier_of } S \subseteq S \implies \text{closedin } X S$

by (*metis* *assms* *closure_of_subset_eq interior_of_subset interior_of_union_frontier_of le_sup_iff*)

show $\text{closedin } X S \implies X \text{frontier_of } S \subseteq S$

by (*simp* *add*: *frontier_of_subset_closedin*)

qed

lemma *frontier_of_complement*: $X \text{frontier_of } (\text{topspace } X - S) = X \text{frontier_of } S$

by (*metis* *Diff_Diff_Int closure_of_restrict frontier_of_closures inf_commute*)

lemma *frontier_of_disjoint_eq*:
assumes $S \subseteq \text{topspace } X$
shows $((X \text{ frontier_of } S) \cap S = \{\}) \longleftrightarrow \text{openin } X S$
proof
assume $X \text{ frontier_of } S \cap S = \{\}$
then have $\text{closedin } X (\text{topspace } X - S)$
using *assms closure_of_subset frontier_of_def interior_of_eq interior_of_subset*
by *fastforce*
then show $\text{openin } X S$
using *assms by (simp add: openin_closedin)*
next
show $\text{openin } X S \implies X \text{ frontier_of } S \cap S = \{\}$
by *(simp add: Diff_Diff_Int closedin_def frontier_of_openin inf.absorb_iff2 inf_commute)*
qed

lemma *frontier_of_disjoint_eq_alt*:
 $S \subseteq (\text{topspace } X - X \text{ frontier_of } S) \longleftrightarrow \text{openin } X S$
proof *(cases $S \subseteq \text{topspace } X$)*
case *True*
show *?thesis*
using *True frontier_of_disjoint_eq by auto*
next
case *False*
then show *?thesis*
by *(meson Diff_subset openin_subset subset_trans)*
qed

lemma *frontier_of_Int*:
 $X \text{ frontier_of } (S \cap T) =$
 $X \text{ closure_of } (S \cap T) \cap (X \text{ frontier_of } S \cup X \text{ frontier_of } T)$
proof *-*
have $*$: $U \subseteq S \wedge U \subseteq T \implies U \cap (S \cap A \cup T \cap B) = U \cap (A \cup B)$ **for** $U S$
 $T A B :: 'a \text{ set}$
by *blast*
show *?thesis*
by *(simp add: frontier_of_closures closure_of_mono Diff_Int * flip: closure_of_Un)*
qed

lemma *frontier_of_Int_subset*: $X \text{ frontier_of } (S \cap T) \subseteq X \text{ frontier_of } S \cup X \text{ frontier_of } T$
by *(simp add: frontier_of_Int)*

lemma *frontier_of_Int_closedin*:
assumes $\text{closedin } X S \text{ closedin } X T$
shows $X \text{ frontier_of } (S \cap T) = X \text{ frontier_of } S \cap T \cup S \cap X \text{ frontier_of } T$ (**is**
?lhs = ?rhs)


```

proof
  show ?lhs  $\subseteq$  ?rhs
    using assms by (force simp add: frontier_of_Int closedin_Int closure_of_closedin)
  show ?rhs  $\subseteq$  ?lhs
    using assms frontier_of_subset_closedin
    by (auto simp add: frontier_of_Int closedin_Int closure_of_closedin)
qed

```

```

lemma frontier_of_Un_subset:  $X \text{ frontier\_of}(S \cup T) \subseteq X \text{ frontier\_of } S \cup X \text{ frontier\_of } T$ 
by (metis Diff_Un frontier_of_Int_subset frontier_of_complement)

```

```

lemma frontier_of_Union_subset:
  finite  $\mathcal{F} \implies X \text{ frontier\_of} (\bigcup \mathcal{F}) \subseteq (\bigcup T \in \mathcal{F}. X \text{ frontier\_of } T)$ 
proof (induction  $\mathcal{F}$  rule: finite_induct)
  case (insert A  $\mathcal{F}$ )
  then show ?case
    using frontier_of_Un_subset by fastforce
qed simp

```

```

lemma frontier_of_frontier_of_subset:
   $X \text{ frontier\_of} (X \text{ frontier\_of } S) \subseteq X \text{ frontier\_of } S$ 
by (simp add: closedin_frontier_of_frontier_of_subset_closedin)

```

```

lemma frontier_of_subtopology_open:
  openin  $X U \implies (\text{subtopology } X U) \text{ frontier\_of } S = U \cap X \text{ frontier\_of } S$ 
by (simp add: Diff_Int_distrib closure_of_subtopology_open frontier_of_def
interior_of_subtopology_open)

```

```

lemma discrete_topology_frontier_of [simp]:
  (discrete_topology  $U$ ) frontier_of  $S = \{\}$ 
by (simp add: Diff_eq discrete_topology_closure_of_frontier_of_closures)

```

```

lemma openin_subset_topospace_eq:
  assumes disjoint  $S (X \text{ frontier\_of } U)$ 
  shows openin (subtopology  $X U$ )  $S \iff \text{openin } X S \wedge S \subseteq U$ 

```

```

proof
  assume  $S: \text{openin} (\text{subtopology } X U) S$ 
  show openin  $X S \wedge S \subseteq U$ 
proof
  show  $S \subseteq U$ 
    using  $S$  openin_imp_subset by blast
  have disjoint  $S (X \text{ frontier\_of} (\text{topspace } X \cap U))$ 
    by (metis assms frontier_of_restrict)
  moreover
  have openin (subtopology  $X (\text{topspace } X \cap U)) S$ 
    by (simp add:  $S$  subtopology_restrict)
  moreover
  have openin  $X S$ 

```

```

if dis: disjnt S (X frontier_of U) and ope: openin (subtopology X U) S and
U  $\subseteq$  topspace X
for S U
proof –
  obtain T where T: openin X T S = T  $\cap$  U
  using ope by (auto simp: openin_subtopology)
  have T  $\cap$  U = T  $\cap$  X interior_of U
  using that T interior_of_subset_in_closure_of by (fastforce simp: disjnt_iff
frontier_of_def)
  then show ?thesis
  using  $\langle S = T \cap U \rangle$   $\langle openin\ X\ T \rangle$  by auto
qed
ultimately show openin X S
by blast
qed
qed (metis inf.absorb_iff1 openin_subtopology_Int)

```

1.13.9 Locally finite collections

```

definition locally_finite_in
where
locally_finite_in X A  $\longleftrightarrow$ 
  ( $\bigcup \mathcal{A} \subseteq$  topspace X)  $\wedge$ 
  ( $\forall x \in$  topspace X.  $\exists V$ . openin X V  $\wedge$   $x \in V \wedge$  finite {U  $\in$  A. U  $\cap$  V  $\neq$ 
  {}})

```

```

lemma finite_imp_locally_finite_in:
  [finite A;  $\bigcup \mathcal{A} \subseteq$  topspace X]  $\implies$  locally_finite_in X A
by (auto simp: locally_finite_in_def)

```

```

lemma locally_finite_in_subset:
assumes locally_finite_in X A  $\mathcal{B} \subseteq \mathcal{A}$ 
shows locally_finite_in X B
proof –
have finite ( $\mathcal{A} \cap \{U. U \cap V \neq \{\}\}$ )  $\implies$  finite ( $\mathcal{B} \cap \{U. U \cap V \neq \{\}\}$ ) for V
by (meson  $\langle \mathcal{B} \subseteq \mathcal{A} \rangle$  finite_subset_inf_le1 inf_le2 le_inf_iff_subset_trans)
then show ?thesis
using assms unfolding locally_finite_in_def Int_def by fastforce
qed

```

```

lemma locally_finite_in_refinement:
assumes A: locally_finite_in X A and f:  $\bigwedge S. S \in \mathcal{A} \implies f\ S \subseteq S$ 
shows locally_finite_in X (f ‘ A)
proof –
show ?thesis
unfolding locally_finite_in_def
proof safe
fix x
assume  $x \in$  topspace X

```

```

then obtain V where openin X V x ∈ V finite {U ∈ A. U ∩ V ≠ {}}
  using A unfolding locally_finite_in_def by blast
moreover have {U ∈ A. f U ∩ V ≠ {}} ⊆ {U ∈ A. U ∩ V ≠ {}} for V
  using f by blast
ultimately have finite {U ∈ A. f U ∩ V ≠ {}}
  using finite_subset by blast
moreover have f ' {U ∈ A. f U ∩ V ≠ {}} = {U ∈ f ' A. U ∩ V ≠ {}}
  by blast
ultimately have finite {U ∈ f ' A. U ∩ V ≠ {}}
  by (metis (no_types, lifting) finite_imageI)
then show ∃ V. openin X V ∧ x ∈ V ∧ finite {U ∈ f ' A. U ∩ V ≠ {}}
  using ⟨openin X V⟩ ⟨x ∈ V⟩ by blast
next
show ∧ x xa. [xa ∈ A; x ∈ f xa] ⇒ x ∈ topspace X
  by (meson Sup_upper A f locally_finite_in_def subset_iff)
qed
qed

lemma locally_finite_in_subtopology:
  assumes A: locally_finite_in X A ∪ A ⊆ S
  shows locally_finite_in (subtopology X S) A
  unfolding locally_finite_in_def
proof safe
  fix x
  assume x: x ∈ topspace (subtopology X S)
  then obtain V where openin X V x ∈ V and fin: finite {U ∈ A. U ∩ V ≠ {}}
  using A unfolding locally_finite_in_def topspace_subtopology by blast
  show ∃ V. openin (subtopology X S) V ∧ x ∈ V ∧ finite {U ∈ A. U ∩ V ≠ {}}
proof (intro exI conjI)
  show openin (subtopology X S) (S ∩ V)
    by (simp add: ⟨openin X V⟩ openin_subtopology_Int2)
  have {U ∈ A. U ∩ (S ∩ V) ≠ {}} ⊆ {U ∈ A. U ∩ V ≠ {}}
    by auto
  with fin show finite {U ∈ A. U ∩ (S ∩ V) ≠ {}}
    using finite_subset by auto
  show x ∈ S ∩ V
    using x ⟨x ∈ V⟩ by (simp)
qed
next
show ∧ x A. [x ∈ A; A ∈ A] ⇒ x ∈ topspace (subtopology X S)
  using assms unfolding locally_finite_in_def topspace_subtopology by blast
qed

```

```

lemma closedin_locally_finite_Union:
  assumes clo: ∧ S. S ∈ A ⇒ closedin X S and A: locally_finite_in X A
  shows closedin X (∪ A)
  using A unfolding locally_finite_in_def closedin_def

```

```

proof clarify
  show openin  $X$  (topspace  $X - \bigcup \mathcal{A}$ )
  proof (subst openin_subopen, clarify)
    fix  $x$ 
    assume  $x \in \text{topspace } X$  and  $x \notin \bigcup \mathcal{A}$ 
    then obtain  $V$  where openin  $X V x \in V$  and fin: finite  $\{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
  using  $\mathcal{A}$  unfolding locally_finite_in_def by blast
  let  $?T = V - \bigcup \{S \in \mathcal{A}. S \cap V \neq \{\}\}$ 
  show  $\exists T. \text{openin } X T \wedge x \in T \wedge T \subseteq \text{topspace } X - \bigcup \mathcal{A}$ 
  proof (intro exI conjI)
    show openin  $X ?T$ 
    by (metis (no_types, lifting) fin <openin X V> clo closedin_Union mem_Collect_eq openin_diff)
    show  $x \in ?T$ 
    using  $\langle x \notin \bigcup \mathcal{A} \rangle \langle x \in V \rangle$  by auto
    show  $?T \subseteq \text{topspace } X - \bigcup \mathcal{A}$ 
    using  $\langle \text{openin } X V \rangle$  openin_subset by auto
  qed
qed
qed

```

```

lemma locally_finite_in_closure:
  assumes  $\mathcal{A}$ : locally_finite_in  $X \mathcal{A}$ 
  shows locally_finite_in  $X ((\lambda S. X \text{closure\_of } S) \text{ ' } \mathcal{A})$ 
  using  $\mathcal{A}$  unfolding locally_finite_in_def
proof (intro conjI; clarsimp)
  fix  $x A$ 
  assume  $x \in X \text{closure\_of } A$ 
  then show  $x \in \text{topspace } X$ 
  by (meson in_closure_of)
next
  fix  $x$ 
  assume  $x \in \text{topspace } X$  and  $\bigcup \mathcal{A} \subseteq \text{topspace } X$ 
  then obtain  $V$  where  $V$ : openin  $X V x \in V$  and fin: finite  $\{U \in \mathcal{A}. U \cap V \neq \{\}\}$ 
  using  $\mathcal{A}$  unfolding locally_finite_in_def by blast
  have eq:  $\{y \in f \text{ ' } \mathcal{A}. Q y\} = f \text{ ' } \{x. x \in \mathcal{A} \wedge Q(f x)\}$  for  $f$  and  $Q :: \text{'a set} \Rightarrow \text{bool}$ 
  by blast
  have eq2:  $\{A \in \mathcal{A}. X \text{closure\_of } A \cap V \neq \{\}\} = \{A \in \mathcal{A}. A \cap V \neq \{\}\}$ 
  using openin_Int_closure_of_eq_empty V by blast
  have finite  $\{U \in (\text{closure\_of}) X \text{ ' } \mathcal{A}. U \cap V \neq \{\}\}$ 
  by (simp add: eq eq2 fin)
  with  $V$  show  $\exists V. \text{openin } X V \wedge x \in V \wedge \text{finite } \{U \in (\text{closure\_of}) X \text{ ' } \mathcal{A}. U \cap V \neq \{\}\}$ 
  by blast
qed

```

lemma *closedin_Union_locally_finite_closure*:

locally_finite_in $X \mathcal{A} \implies \text{closedin } X (\bigcup ((\lambda S. X \text{ closure_of } S) \text{ ` } \mathcal{A}))$
by (*metis* (*mono_tags*) *closedin_closure_of* *closedin_locally_finite_Union_imageE* *locally_finite_in_closure*)

lemma *closure_of_Union_subset*: $\bigcup ((\lambda S. X \text{ closure_of } S) \text{ ` } \mathcal{A}) \subseteq X \text{ closure_of } (\bigcup \mathcal{A})$

by (*simp* *add*: *SUP_le_iff* *Sup_upper* *closure_of_mono*)

lemma *closure_of_locally_finite_Union*:

assumes *locally_finite_in* $X \mathcal{A}$

shows $X \text{ closure_of } (\bigcup \mathcal{A}) = \bigcup ((\lambda S. X \text{ closure_of } S) \text{ ` } \mathcal{A})$

proof (*rule* *closure_of_unique*)

show $\bigcup \mathcal{A} \subseteq \bigcup ((\text{closure_of}) X \text{ ` } \mathcal{A})$

using *assms* **by** (*simp* *add*: *SUP_upper2* *Sup_le_iff* *closure_of_subset_locally_finite_in_def*)

show $\text{closedin } X (\bigcup ((\text{closure_of}) X \text{ ` } \mathcal{A}))$

using *assms* **by** (*simp* *add*: *closedin_Union_locally_finite_closure*)

show $\bigwedge T'. [\bigcup \mathcal{A} \subseteq T'; \text{closedin } X T'] \implies \bigcup ((\text{closure_of}) X \text{ ` } \mathcal{A}) \subseteq T'$

by (*simp* *add*: *Sup_le_iff* *closure_of_minimal*)

qed

1.13.10 Continuous maps

We will need to deal with continuous maps in terms of topologies and not in terms of type classes, as defined below.

definition *continuous_map* **where**

continuous_map $X Y f \equiv$

$f \in \text{topspace } X \rightarrow \text{topspace } Y \wedge$

$(\forall U. \text{openin } Y U \longrightarrow \text{openin } X \{x \in \text{topspace } X. f x \in U\})$

lemma *continuous_map*:

continuous_map $X Y f \longleftrightarrow$

$f \text{ ` } (\text{topspace } X) \subseteq \text{topspace } Y \wedge (\forall U. \text{openin } Y U \longrightarrow \text{openin } X \{x \in \text{topspace } X. f x \in U\})$

by (*auto* *simp*: *continuous_map_def*)

lemma *continuous_map_image_subset_topspace*:

continuous_map $X Y f \implies f \text{ ` } (\text{topspace } X) \subseteq \text{topspace } Y$

by (*auto* *simp*: *continuous_map_def*)

lemma *continuous_map_funspace*:

continuous_map $X Y f \implies f \in \text{topspace } X \rightarrow \text{topspace } Y$

by (*auto* *simp*: *continuous_map_def*)

lemma *continuous_map_on_empty* [*simp*]: *continuous_map* *trivial_topology* $Y f$

by (*auto* *simp*: *continuous_map_def*)

lemma *continuous_map_on_empty2* [*simp*]: *continuous_map* $X \text{ trivial_topology}$

$f \leftarrow X = \text{trivial_topology}$
using *continuous_map_image_subset_topspace* **by** *fastforce*

lemma *continuous_map_closedin*:
 $\text{continuous_map } X \ Y \ f \leftarrow$
 $f \in \text{topspace } X \rightarrow \text{topspace } Y \wedge$
 $(\forall C. \text{closedin } Y \ C \longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\})$

proof –
have $(\forall U. \text{openin } Y \ U \longrightarrow \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}) =$
 $(\forall C. \text{closedin } Y \ C \longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\})$
if $f \in \text{topspace } X \rightarrow \text{topspace } Y$
proof –
have $\text{eq: } \{x \in \text{topspace } X. f \ x \in \text{topspace } Y \wedge f \ x \notin C\} = (\text{topspace } X - \{x$
 $\in \text{topspace } X. f \ x \in C\})$ **for** C
using *that* **by** *blast*
show *?thesis*
proof (*intro iffI allI impI*)
fix C
assume $\forall U. \text{openin } Y \ U \longrightarrow \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ **and**
 $\text{closedin } Y \ C$
then show $\text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\}$
by (*auto simp add: closedin_def eq*)
next
fix U
assume $\forall C. \text{closedin } Y \ C \longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\}$ **and**
 $\text{openin } Y \ U$
then show $\text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$
by (*auto simp add: openin_closedin_eq eq*)
qed
qed
then show *?thesis*
by (*auto simp: continuous_map_def*)
qed

lemma *openin_continuous_map_preimage*:
 $\llbracket \text{continuous_map } X \ Y \ f; \text{openin } Y \ U \rrbracket \Longrightarrow \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$
by (*simp add: continuous_map_def*)

lemma *closedin_continuous_map_preimage*:
 $\llbracket \text{continuous_map } X \ Y \ f; \text{closedin } Y \ C \rrbracket \Longrightarrow \text{closedin } X \ \{x \in \text{topspace } X. f \ x \in C\}$
by (*simp add: continuous_map_closedin*)

lemma *openin_continuous_map_preimage_gen*:
assumes $\text{continuous_map } X \ Y \ f \ \text{openin } X \ U \ \text{openin } Y \ V$
shows $\text{openin } X \ \{x \in U. f \ x \in V\}$
proof –
have $\text{eq: } \{x \in U. f \ x \in V\} = U \cap \{x \in \text{topspace } X. f \ x \in V\}$
using *assms(2) openin_closedin_eq* **by** *fastforce*

```

show ?thesis
  unfolding eq
  using assms openin_continuous_map_preimage by fastforce
qed

```

```

lemma closedin_continuous_map_preimage_gen:
  assumes continuous_map X Y f closedin X U closedin Y V
  shows closedin X {x ∈ U. f x ∈ V}
proof –
  have eq: {x ∈ U. f x ∈ V} = U ∩ {x ∈ topspace X. f x ∈ V}
    using assms(2) closedin_def by fastforce
  show ?thesis
    unfolding eq
    using assms closedin_continuous_map_preimage by fastforce
qed

```

```

lemma continuous_map_image_closure_subset:
  assumes continuous_map X Y f
  shows  $f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S$ 
proof –
  have *:  $f' (\text{topspace } X) \subseteq \text{topspace } Y$ 
    by (meson assms continuous_map)
  have  $X \text{ closure\_of } T \subseteq \{x \in X \text{ closure\_of } T. f x \in Y \text{ closure\_of } (f' T)\}$ 
    if  $T \subseteq \text{topspace } X$  for  $T$ 
  proof (rule closure_of_minimal)
    show  $T \subseteq \{x \in X \text{ closure\_of } T. f x \in Y \text{ closure\_of } f' T\}$ 
      using closure_of_subset * that by (fastforce simp: in_closure_of)
  next
    show closedin X {x ∈ X closure_of T. f x ∈ Y closure_of f' T}
      using assms closedin_continuous_map_preimage_gen by fastforce
  qed
  then show ?thesis
    by (smt (verit, ccfv_threshold) assms continuous_map image_eqI image_subset_iff in_closure_of mem_Collect_eq)
qed

```

```

lemma continuous_map_subset_aux1: continuous_map X Y f  $\implies$ 
   $(\forall S. f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S)$ 
  using continuous_map_image_closure_subset by blast

```

```

lemma continuous_map_subset_aux2:
  assumes  $\forall S. S \subseteq \text{topspace } X \implies f' (X \text{ closure\_of } S) \subseteq Y \text{ closure\_of } f' S$ 
  shows continuous_map X Y f
  unfolding continuous_map_closedin
proof (intro conjI ballI allI impI)
  show  $f \in \text{topspace } X \rightarrow \text{topspace } Y$ 
    using assms closure_of_subset_topspace by fastforce
next
  fix  $C$ 

```

```

assume closedin Y C
then show closedin X {x ∈ topspace X. f x ∈ C}
proof (clarsimp simp flip: closure_of_subset_eq, intro conjI)
  fix x
  assume x: x ∈ X closure_of {x ∈ topspace X. f x ∈ C}
    and C ⊆ topspace Y and Y closure_of C ⊆ C
  show x ∈ topspace X
    by (meson x in_closure_of)
  have {a ∈ topspace X. f a ∈ C} ⊆ topspace X
    by simp
  moreover have Y closure_of f ‘ {a ∈ topspace X. f a ∈ C} ⊆ C
    by (simp add: <closedin Y C> closure_of_minimal_image_subset_iff)
  ultimately show f x ∈ C
    using x assms by blast
qed
qed

```

```

lemma continuous_map_eq_image_closure_subset:
  continuous_map X Y f ↔ (∀ S. f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S)
using continuous_map_subset_aux1 continuous_map_subset_aux2 by metis

```

```

lemma continuous_map_eq_image_closure_subset_alt:
  continuous_map X Y f ↔ (∀ S. S ⊆ topspace X → f ‘ (X closure_of S) ⊆
  Y closure_of f ‘ S)
using continuous_map_subset_aux1 continuous_map_subset_aux2 by metis

```

```

lemma continuous_map_eq_image_closure_subset_gen:
  continuous_map X Y f ↔
  f ∈ topspace X → topspace Y ∧
  (∀ S. f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S)
by (meson Pi_iff continuous_map continuous_map_eq_image_closure_subset
image_subset_iff)

```

```

lemma continuous_map_closure_preimage_subset:
  continuous_map X Y f
  ⇒ X closure_of {x ∈ topspace X. f x ∈ T}
  ⊆ {x ∈ topspace X. f x ∈ Y closure_of T}
unfolding continuous_map_closedin
by (rule closure_of_minimal) (use in_closure_of in <fastforce+>)

```

```

lemma continuous_map_frontier_frontier_preimage_subset:
assumes continuous_map X Y f
shows X frontier_of {x ∈ topspace X. f x ∈ T} ⊆ {x ∈ topspace X. f x ∈ Y
frontier_of T}
proof –
  have eq: topspace X – {x ∈ topspace X. f x ∈ T} = {x ∈ topspace X. f x ∈
  topspace Y – T}
  using assms unfolding continuous_map_def by blast

```



```

have X closure_of {x ∈ topspace X. f x ∈ T} ⊆ {x ∈ topspace X. f x ∈ Y
closure_of T}
  by (simp add: assms continuous_map_closure_preimage_subset)
moreover
have X closure_of (topspace X - {x ∈ topspace X. f x ∈ T}) ⊆ {x ∈ topspace
X. f x ∈ Y closure_of (topspace Y - T)}
  using continuous_map_closure_preimage_subset [OF assms] eq by presburger
ultimately show ?thesis
  by (auto simp: frontier_of_closures)
qed

```

```

lemma topology_finer_continuous_id:
  assumes topspace X = topspace Y
  shows (∀ S. openin X S ⟶ openin Y S) ⟷ continuous_map Y X id (is ?lhs
= ?rhs)
proof
  show ?lhs ⟹ ?rhs
    unfolding continuous_map_def
    using assms openin_subopen openin_subset by fastforce
  show ?rhs ⟹ ?lhs
    unfolding continuous_map_def
    using assms openin_subopen topspace_def by fastforce
qed

```

```

lemma continuous_map_const [simp]:
  continuous_map X Y (λx. C) ⟷ X = trivial_topology ∨ C ∈ topspace Y
proof (cases X = trivial_topology)
  case nontriv: False
  show ?thesis
  proof (cases C ∈ topspace Y)
  case True
    with openin_subopen show ?thesis
    by (auto simp: continuous_map_def)
  next
  case False
    with nontriv show ?thesis
    using continuous_map_image_subset_topspace discrete_topology_unique
image_subset_iff by fastforce
  qed
qed auto

```

```

declare continuous_map_const [THEN iffD2, continuous_intros]

```

```

lemma continuous_map_compose [continuous_intros]:
  assumes f: continuous_map X X' f and g: continuous_map X' X'' g
  shows continuous_map X X'' (g ∘ f)
  unfolding continuous_map_def
proof (intro conjI ballI allI impI)
  show g ∘ f ∈ topspace X → topspace X''

```

```

    using assms unfolding continuous_map_def by force
next
  fix U
  assume openin X'' U
  have eq: {x ∈ topspace X. (g ∘ f) x ∈ U} = {x ∈ topspace X. f x ∈ {y. y ∈
topspace X' ∧ g y ∈ U}}
    using continuous_map_image_subset_topspace f by force
  show openin X {x ∈ topspace X. (g ∘ f) x ∈ U}
    unfolding eq
    using assms unfolding continuous_map_def
    using ⟨openin X'' U⟩ by blast
qed

```

```

lemma continuous_map_eq:
  assumes continuous_map X X' f and  $\bigwedge x. x \in \text{topspace } X \implies f x = g x$ 
  shows continuous_map X X' g
proof -
  have eq: {x ∈ topspace X. f x ∈ U} = {x ∈ topspace X. g x ∈ U} for U
    using assms by auto
  show ?thesis
    using assms by (force simp add: continuous_map_def) eq
qed

```

```

lemma restrict_continuous_map [simp]:
  topspace X ⊆ S  $\implies$  continuous_map X X' (restrict f S)  $\longleftrightarrow$  continuous_map
X X' f
  by (auto simp: elim!: continuous_map_eq)

```

```

lemma continuous_map_in_subtopology:
  continuous_map X (subtopology X' S) f  $\longleftrightarrow$  continuous_map X X' f ∧ f '
(topspace X) ⊆ S
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof -
    have  $\bigwedge A. f '(X \text{ closure\_of } A) \subseteq \text{subtopology } X' S \text{ closure\_of } f ' A$ 
      by (meson L continuous_map_image_closure_subset)
    then show ?thesis
      by (metis (no_types) closure_of_subset_subtopology closure_of_subtopology_subset
closure_of_topspace continuous_map_eq_image_closure_subset order.trans)
  qed
next
  assume R: ?rhs
  then have eq: {x ∈ topspace X. f x ∈ U} = {x ∈ topspace X. f x ∈ U ∧ f x ∈
S} for U
    by auto
  show ?lhs
    using R

```

unfolding *continuous_map*
by (*auto simp: openin_subtopology eq*)
qed

lemma *continuous_map_from_subtopology*:
 $continuous_map\ X\ Y\ f \implies continuous_map\ (subtopology\ X\ S)\ Y\ f$
by (*auto simp: continuous_map openin_subtopology*)

lemma *continuous_map_into_fulltopology*:
 $continuous_map\ X\ (subtopology\ Y\ T)\ f \implies continuous_map\ X\ Y\ f$
by (*auto simp: continuous_map_in_subtopology*)

lemma *continuous_map_into_subtopology*:
 $\llbracket continuous_map\ X\ Y\ f; f \in\ topspace\ X \rightarrow T \rrbracket \implies continuous_map\ X\ (subtopology\ Y\ T)\ f$
by (*auto simp: continuous_map_in_subtopology*)

lemma *continuous_map_from_subtopology_mono*:
 $\llbracket continuous_map\ (subtopology\ X\ T)\ Y\ f; S \subseteq T \rrbracket \implies continuous_map\ (subtopology\ X\ S)\ Y\ f$
by (*metis inf.absorb_iff2 continuous_map_from_subtopology subtopology_subtopology*)

lemma *continuous_map_from_discrete_topology* [*simp*]:
 $continuous_map\ (discrete_topology\ U)\ X\ f \iff f \in U \rightarrow topspace\ X$
by (*auto simp: continuous_map_def*)

lemma *continuous_map_iff_continuous* [*simp*]: *continuous_map* (*top_of_set* *S*)
euclidean g = continuous_on S g
by (*fastforce simp add: continuous_map openin_subtopology continuous_on_open_invariant*)

lemma *continuous_map_iff_continuous2* [*simp*]: *continuous_map euclidean euclidean g = continuous_on UNIV g*
by (*metis continuous_map_iff_continuous subtopology_UNIV*)

lemma *continuous_map_openin_preimage_eq*:
 $continuous_map\ X\ Y\ f \iff f \in (topspace\ X) \rightarrow topspace\ Y \wedge (\forall U. openin\ Y\ U \longrightarrow openin\ X\ (topspace\ X \cap f^{-1}\ U))$
by (*auto simp: continuous_map_def vimage_def Int_def*)

lemma *continuous_map_closedin_preimage_eq*:
 $continuous_map\ X\ Y\ f \iff f \in (topspace\ X) \rightarrow topspace\ Y \wedge (\forall U. closedin\ Y\ U \longrightarrow closedin\ X\ (topspace\ X \cap f^{-1}\ U))$
by (*auto simp: continuous_map_closedin vimage_def Int_def*)

lemma *continuous_map_square_root*: *continuous_map euclideanreal euclideanreal sqrt*

by (simp add: continuous_at_imp_continuous_on isCont_real_sqrt)

lemma continuous_map_sqrt [continuous_intros]:

continuous_map X euclideanreal f \implies continuous_map X euclideanreal ($\lambda x.$
sqrt(f x))

by (meson continuous_map_compose continuous_map_eq continuous_map_square_root
o_apply)

lemma continuous_map_id [simp, continuous_intros]: continuous_map X X id

unfolding continuous_map_def using openin_subopen topspace_def by fastforce

declare continuous_map_id [unfolded id_def, simp, continuous_intros]

lemma continuous_map_id_subt [simp]: continuous_map (subtopology X S) X id

by (simp add: continuous_map_from_subtopology)

declare continuous_map_id_subt [unfolded id_def, simp]

lemma continuous_map_alt:

continuous_map T1 T2 f

= (($\forall U.$ openin T2 U \longrightarrow openin T1 (f $^{-}$ U \cap topspace T1)) \wedge f \in topspace
T1 \rightarrow topspace T2)

by (auto simp: continuous_map_def vimage_def image_def Collect_conj_eq
inf_commute)

lemma continuous_map_open [intro]:

continuous_map T1 T2 f \implies openin T2 U \implies openin T1 (f $^{-}$ U \cap topspace(T1))

unfolding continuous_map_alt by auto

lemma continuous_map_preimage_topspace [intro]:

assumes continuous_map T1 T2 f

shows f $^{-}$ (topspace T2) \cap topspace T1 = topspace T1

using assms unfolding continuous_map_def by auto

1.13.11 Open and closed maps (not a priori assumed continuous)

definition open_map :: 'a topology \Rightarrow 'b topology \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool

where open_map X1 X2 f $\equiv \forall U.$ openin X1 U \longrightarrow openin X2 (f $^{-}$ U)

definition closed_map :: 'a topology \Rightarrow 'b topology \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool

where closed_map X1 X2 f $\equiv \forall U.$ closedin X1 U \longrightarrow closedin X2 (f $^{-}$ U)

lemma open_map_imp_subset_topspace:

open_map X1 X2 f \implies f \in (topspace X1) \rightarrow topspace X2

unfolding open_map_def using openin_subset by fastforce

lemma *open_map_on_empty* [simp]: *open_map trivial_topology Y f*
by (*simp add: open_map_def*)

lemma *closed_map_on_empty*:
closed_map trivial_topology Y f
by (*simp add: closed_map_def closedin_topospace_empty*)

lemma *closed_map_const*:
closed_map X Y ($\lambda x. c$) \longleftrightarrow X = trivial_topology \vee closedin Y {c}
by (*metis closed_map_def closed_map_on_empty closedin_topospace discrete_topology_unique equals0D image_constant_conv*)

lemma *open_map_imp_subset*:
 $\llbracket \text{open_map } X1\ X2\ f; S \subseteq \text{topspace } X1 \rrbracket \implies f \in S \rightarrow \text{topspace } X2$
using *open_map_imp_subset_topospace* **by** *fastforce*

lemma *topology_finer_open_id*:
 $(\forall S. \text{openin } X\ S \longrightarrow \text{openin } X'\ S) \longleftrightarrow \text{open_map } X\ X'\ \text{id}$
unfolding *open_map_def* **by** *auto*

lemma *open_map_id*: *open_map X X id*
unfolding *open_map_def* **by** *auto*

lemma *open_map_eq*:
 $\llbracket \text{open_map } X\ X'\ f; \bigwedge x. x \in \text{topspace } X \implies f\ x = g\ x \rrbracket \implies \text{open_map } X\ X'\ g$
unfolding *open_map_def*
by (*metis image_cong openin_subset subset_iff*)

lemma *open_map_inclusion_eq*:
open_map (subtopology X S) X id \longleftrightarrow openin X (topspace X \cap S)
by (*metis openin_topospace openin_trans_full subtopology_restrict topology_finer_open_id topspace_subtopology*)

lemma *open_map_inclusion*:
 $\text{openin } X\ S \implies \text{open_map } (\text{subtopology } X\ S)\ X\ \text{id}$
by (*simp add: open_map_inclusion_eq openin_Int*)

lemma *open_map_compose*:
 $\llbracket \text{open_map } X\ X'\ f; \text{open_map } X'\ X''\ g \rrbracket \implies \text{open_map } X\ X''\ (g \circ f)$
by (*metis (no_types, lifting) image_comp open_map_def*)

lemma *closed_map_imp_subset_topospace*:
 $\text{closed_map } X1\ X2\ f \implies f \in (\text{topspace } X1) \rightarrow \text{topspace } X2$
by (*simp add: closed_map_def closedin_def image_subset_iff_funcset*)

lemma *closed_map_imp_subset*:
 $\llbracket \text{closed_map } X1\ X2\ f; S \subseteq \text{topspace } X1 \rrbracket \implies f \in S \rightarrow \text{topspace } X2$
using *closed_map_imp_subset_topospace* **by** *blast*

lemma *topology_finer_closed_id*:

$(\forall S. \text{closedin } X \ S \longrightarrow \text{closedin } X' \ S) \longleftrightarrow \text{closed_map } X \ X' \ \text{id}$

by (*simp add: closed_map_def*)

lemma *closed_map_id*: $\text{closed_map } X \ X \ \text{id}$

by (*simp add: closed_map_def*)

lemma *closed_map_eq*:

$\llbracket \text{closed_map } X \ X' \ f; \bigwedge x. x \in \text{topspace } X \implies f \ x = g \ x \rrbracket \implies \text{closed_map } X \ X' \ g$

unfolding *closed_map_def*

by (*metis image_cong closedin_subset subset_iff*)

lemma *closed_map_compose*:

$\llbracket \text{closed_map } X \ X' \ f; \text{closed_map } X' \ X'' \ g \rrbracket \implies \text{closed_map } X \ X'' \ (g \circ f)$

by (*metis (no_types, lifting) closed_map_def image_comp*)

lemma *closed_map_inclusion_eq*:

$\text{closed_map } (\text{subtopology } X \ S) \ X \ \text{id} \longleftrightarrow$
 $\text{closedin } X \ (\text{topspace } X \cap S)$

proof –

have *: $\text{closedin } X \ (T \cap S)$ **if** $\text{closedin } X \ (S \cap \text{topspace } X)$ $\text{closedin } X \ T$ **for** T

by (*smt (verit, best) closedin_Int closure_of_subset_eq inf_sup_aci le_iff_inf that*)

then show *?thesis*

by (*fastforce simp add: closed_map_def Int_commute closedin_subtopology_alt intro: **)

qed

lemma *closed_map_inclusion*: $\text{closedin } X \ S \implies \text{closed_map } (\text{subtopology } X \ S) \ X \ \text{id}$

by (*simp add: closed_map_inclusion_eq closedin_Int*)

lemma *open_map_into_subtopology*:

$\llbracket \text{open_map } X \ X' \ f; f \in \text{topspace } X \rightarrow S \rrbracket \implies \text{open_map } X \ (\text{subtopology } X' \ S)$

unfolding *open_map_def openin_subtopology*

using *openin_subset* **by** *fastforce*

lemma *closed_map_into_subtopology*:

$\llbracket \text{closed_map } X \ X' \ f; f \in \text{topspace } X \rightarrow S \rrbracket \implies \text{closed_map } X \ (\text{subtopology } X' \ S)$

unfolding *closed_map_def closedin_subtopology*

using *closedin_subset* **by** *fastforce*

lemma *open_map_into_discrete_topology*:

$\text{open_map } X \ (\text{discrete_topology } U) \ f \longleftrightarrow f \in (\text{topspace } X) \rightarrow U$

unfolding *open_map_def openin_discrete_topology* **using** *openin_subset* **by** *blast*

lemma *closed_map_into_discrete_topology*:

$\text{closed_map } X \text{ (discrete_topology } U) f \longleftrightarrow f \in (\text{topspace } X) \rightarrow U$

unfolding *closed_map_def closedin_discrete_topology using closedin_subset*
by *blast*

lemma *bijjective_open_imp_closed_map*:

$\llbracket \text{open_map } X X' f; f' (\text{topspace } X) = \text{topspace } X'; \text{inj_on } f (\text{topspace } X) \rrbracket$

$\implies \text{closed_map } X X' f$

unfolding *open_map_def closed_map_def closedin_def*

by *auto (metis Diff_subset inj_on_image_set_diff)*

lemma *bijjective_closed_imp_open_map*:

$\llbracket \text{closed_map } X X' f; f' (\text{topspace } X) = \text{topspace } X'; \text{inj_on } f (\text{topspace } X) \rrbracket$

$\implies \text{open_map } X X' f$

unfolding *closed_map_def open_map_def openin_closedin_eq*

by *auto (metis Diff_subset inj_on_image_set_diff)*

lemma *open_map_from_subtopology*:

$\llbracket \text{open_map } X X' f; \text{openin } X U \rrbracket \implies \text{open_map (subtopology } X U) X' f$

unfolding *open_map_def openin_subtopology_alt* **by** *blast*

lemma *closed_map_from_subtopology*:

$\llbracket \text{closed_map } X X' f; \text{closedin } X U \rrbracket \implies \text{closed_map (subtopology } X U) X' f$

unfolding *closed_map_def closedin_subtopology_alt* **by** *blast*

lemma *open_map_restriction*:

assumes $f: \text{open_map } X X' f$ **and** $U: \{x \in \text{topspace } X. f x \in V\} = U$

shows $\text{open_map (subtopology } X U) (\text{subtopology } X' V) f$

unfolding *open_map_def*

proof *clarsimp*

fix W

assume $\text{openin (subtopology } X U) W$

then obtain T **where** $\text{openin } X T W = T \cap U$

by (*meson openin_subtopology*)

with $f U$ **have** $f' W = (f' T) \cap V$

unfolding *open_map_def openin_closedin_eq* **by** *auto*

then show $\text{openin (subtopology } X' V) (f' W)$

by (*metis <openin X T> f open_map_def openin_subtopology_Int*)

qed

lemma *closed_map_restriction*:

assumes $f: \text{closed_map } X X' f$ **and** $U: \{x \in \text{topspace } X. f x \in V\} = U$

shows $\text{closed_map (subtopology } X U) (\text{subtopology } X' V) f$

unfolding *closed_map_def*

proof *clarsimp*

fix W

assume $\text{closedin (subtopology } X U) W$

then obtain T **where** $\text{closedin } X T W = T \cap U$

by (*meson* *closedin_subtopology*)
 with $f \ U$ have $f \ ' \ W = (f \ ' \ T) \cap V$
 unfolding *closed_map_def* *closedin_def* by *auto*
 then show *closedin* (*subtopology* $X \ ' \ V$) ($f \ ' \ W$)
 by (*metis* \langle *closedin* $X \ T \rangle$ *closed_map_def* *closedin_subtopology* f)
 qed

lemma *closed_map_closure_of_image*:

$closed_map \ X \ Y \ f \longleftrightarrow$
 $f \in \mathit{topspace} \ X \rightarrow \mathit{topspace} \ Y \wedge$
 $(\forall S. S \subseteq \mathit{topspace} \ X \longrightarrow Y \ \mathit{closure_of} \ (f \ ' \ S) \subseteq f \ ' \ (X \ \mathit{closure_of} \ S))$ (is
?lhs=?rhs)

proof

assume *?lhs*

then show *?rhs*

by (*simp* *add*: *closed_map_def* *closed_map_imp_subset_topspace* *closure_of_minimal*
closure_of_subset *image_mono*)

next

assume *?rhs*

then show *?lhs*

by (*metis* *closed_map_def* *closed_map_into_discrete_topology* *closure_of_eq*

closure_of_subset_eq *topspace_discrete_topology*)

qed

lemma *open_map_interior_of_image_subset*:

$open_map \ X \ Y \ f \longleftrightarrow (\forall S. \mathit{image} \ f \ (X \ \mathit{interior_of} \ S) \subseteq Y \ \mathit{interior_of} \ (f \ ' \ S))$
 by (*metis* *image_mono* *interior_of_eq* *interior_of_maximal* *interior_of_subset*
open_map_def *openin_interior_of* *subset_antisym*)

lemma *open_map_interior_of_image_subset_alt*:

$open_map \ X \ Y \ f \longleftrightarrow (\forall S \subseteq \mathit{topspace} \ X. f \ ' \ (X \ \mathit{interior_of} \ S) \subseteq Y \ \mathit{interior_of} \ f \ ' \ S)$

by (*metis* *interior_of_eq* *open_map_def* *open_map_interior_of_image_subset*
openin_subset *subset_interior_of_eq*)

lemma *open_map_interior_of_image_subset_gen*:

$open_map \ X \ Y \ f \longleftrightarrow$
 $(f \in \mathit{topspace} \ X \rightarrow \mathit{topspace} \ Y \wedge (\forall S. f \ ' \ (X \ \mathit{interior_of} \ S) \subseteq Y \ \mathit{interior_of} \ f \ ' \ S))$

by (*metis* *open_map_imp_subset_topspace* *open_map_interior_of_image_subset*)

lemma *open_map_preimage_neighbourhood*:

$open_map \ X \ Y \ f \longleftrightarrow$
 $(f \in \mathit{topspace} \ X \rightarrow \mathit{topspace} \ Y \wedge$
 $(\forall U \ T. \mathit{closedin} \ X \ U \wedge T \subseteq \mathit{topspace} \ Y \wedge$
 $\{x \in \mathit{topspace} \ X. f \ x \in T\} \subseteq U \longrightarrow$
 $(\exists V. \mathit{closedin} \ Y \ V \wedge T \subseteq V \wedge \{x \in \mathit{topspace} \ X. f \ x \in V\} \subseteq U)))$ (is
?lhs=?rhs)


```

proof
  assume L: ?lhs
  show ?rhs
  proof (intro conjI strip)
    show  $f \in \text{topspace } X \rightarrow \text{topspace } Y$ 
      by (simp add: ⟨open_map X Y f⟩ open_map_imp_subset_topspace)
  next
    fix U T
    assume UT:  $\text{closedin } X \ U \wedge T \subseteq \text{topspace } Y \wedge \{x \in \text{topspace } X. f \ x \in T\} \subseteq U$ 
    have  $\text{closedin } Y \ (\text{topspace } Y - f \ ' \ (\text{topspace } X - U))$ 
      by (meson UT L open_map_def openin_closedin_eq openin_diff openin_topspace)
    with UT
    show  $\exists V. \text{closedin } Y \ V \wedge T \subseteq V \wedge \{x \in \text{topspace } X. f \ x \in V\} \subseteq U$ 
      using image_iff by auto
    qed
  next
    assume R: ?rhs
    show ?lhs
    unfolding open_map_def
    proof (intro strip)
      fix U assume openin X U
      have  $\{x \in \text{topspace } X. f \ x \in \text{topspace } Y - f \ ' \ U\} \subseteq \text{topspace } X - U$ 
        by blast
      then obtain V where V:  $\text{closedin } Y \ V$ 
      and sub:  $\text{topspace } Y - f \ ' \ U \subseteq V \wedge \{x \in \text{topspace } X. f \ x \in V\} \subseteq \text{topspace } X - U$ 
        using R ⟨openin X U⟩ by (meson Diff_subset openin_closedin_eq)
      then have  $f \ ' \ U \subseteq \text{topspace } Y - V$ 
        using R ⟨openin X U⟩ openin_subset by fastforce
      with sub have  $f \ ' \ U = \text{topspace } Y - V$ 
        by auto
      then show openin Y (f ' U)
        using V(1) by auto
    qed
  qed

```

lemma closed_map_preimage_neighbourhood:

$$\begin{aligned}
 & \text{closed_map } X \ Y \ f \longleftrightarrow \\
 & \quad f \in \text{topspace } X \rightarrow \text{topspace } Y \wedge \\
 & \quad (\forall U \ T. \text{openin } X \ U \wedge T \subseteq \text{topspace } Y \wedge \\
 & \quad \quad \{x \in \text{topspace } X. f \ x \in T\} \subseteq U \\
 & \quad \longrightarrow (\exists V. \text{openin } Y \ V \wedge T \subseteq V \wedge \\
 & \quad \quad \{x \in \text{topspace } X. f \ x \in V\} \subseteq U)) \quad (\text{is } ?lhs=?rhs)
 \end{aligned}$$

```

proof
  assume L: ?lhs
  show ?rhs
  proof (intro conjI strip)

```

```

    show  $f \in \text{topspace } X \rightarrow \text{topspace } Y$ 
      by (simp add:  $L \text{ closed\_map\_imp\_subset\_topspace}$ )
  next
    fix  $U T$ 
    assume  $UT: \text{openin } X U \wedge T \subseteq \text{topspace } Y \wedge \{x \in \text{topspace } X. f x \in T\} \subseteq U$ 
    then have  $\text{openin } Y (\text{topspace } Y - f^{-1}(\text{topspace } X - U))$ 
      by (meson  $L \text{ closed\_map\_def closedin\_def closedin\_diff closedin\_topspace}$ )
    then show  $\exists V. \text{openin } Y V \wedge T \subseteq V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq U$ 
      using  $UT \text{ image\_iff}$  by auto
    qed
  next
    assume  $R: ?rhs$ 
    show  $?lhs$ 
      unfolding  $\text{closed\_map\_def}$ 
    proof (intro strip)
      fix  $U$  assume  $\text{closedin } X U$ 
      have  $\{x \in \text{topspace } X. f x \in \text{topspace } Y - f^{-1} U\} \subseteq \text{topspace } X - U$ 
        by blast
      then obtain  $V$  where  $V: \text{openin } Y V$ 
        and  $\text{sub}: \text{topspace } Y - f^{-1} U \subseteq V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq \text{topspace } X - U$ 
        using  $R \text{ Diff\_subset } \langle \text{closedin } X U \rangle$  unfolding  $\text{closedin\_def}$ 
        by (smt (verit, ccfv_threshold)  $\text{Collect\_mem\_eq Collect\_mono\_iff}$ )
      then have  $f^{-1} U \subseteq \text{topspace } Y - V$ 
        using  $R \langle \text{closedin } X U \rangle \text{ closedin\_subset}$  by fastforce
      with  $\text{sub}$  have  $f^{-1} U = \text{topspace } Y - V$ 
        by auto
      with  $V$  show  $\text{closedin } Y (f^{-1} U)$ 
        by auto
    qed
  qed

lemma  $\text{closed\_map\_fibre\_neighbourhood}$ :
   $\text{closed\_map } X Y f \iff$ 
   $f \in (\text{topspace } X) \rightarrow \text{topspace } Y \wedge$ 
   $(\forall U y. \text{openin } X U \wedge y \in \text{topspace } Y \wedge \{x \in \text{topspace } X. f x = y\} \subseteq U$ 
   $\implies (\exists V. \text{openin } Y V \wedge y \in V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq U))$ 
  unfolding  $\text{closed\_map\_preimage\_neighbourhood}$ 
proof (intro conj_cong refl all_cong1)
  fix  $U$ 
  assume  $f \in \text{topspace } X \rightarrow \text{topspace } Y$ 
  show  $(\forall T. \text{openin } X U \wedge T \subseteq \text{topspace } Y \wedge \{x \in \text{topspace } X. f x \in T\} \subseteq U$ 
     $\implies (\exists V. \text{openin } Y V \wedge T \subseteq V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq U))$ 
    =  $(\forall y. \text{openin } X U \wedge y \in \text{topspace } Y \wedge \{x \in \text{topspace } X. f x = y\} \subseteq U$ 
     $\implies (\exists V. \text{openin } Y V \wedge y \in V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq U))$ 
    (is  $(\forall T. ?P T) \iff (\forall y. ?Q y)$ )
  proof
    assume  $L [rule\_format]: \forall T. ?P T$ 
    show  $\forall y. ?Q y$ 

```

```

proof
  fix  $y$  show  $?Q\ y$ 
    using  $L$  [of  $\{y\}$ ] by blast
qed
next
  assume  $R: \forall y. ?Q\ y$ 
  show  $\forall T. ?P\ T$ 
  proof (cases openin X U)
    case True
      note  $[[unify\_search\_bound=3]]$ 
      obtain  $V$  where  $V: \bigwedge y. \llbracket y \in \text{topspace } Y; \{x \in \text{topspace } X. f\ x = y\} \subseteq U \rrbracket$ 
 $\implies$ 
         $\text{openin } Y\ (V\ y) \wedge y \in V\ y \wedge \{x \in \text{topspace } X. f\ x \in V\ y\} \subseteq U$ 
        using  $R$  by (simp add: True) meson
        show  $?thesis$ 
        proof clarify
          fix  $T$ 
          assume  $\text{openin } X\ U$  and  $T \subseteq \text{topspace } Y$  and  $\{x \in \text{topspace } X. f\ x \in T\}$ 
 $\subseteq U$ 
          with  $V$  show  $\exists V. \text{openin } Y\ V \wedge T \subseteq V \wedge \{x \in \text{topspace } X. f\ x \in V\} \subseteq U$ 
          by (rule_tac x= $\bigcup y \in T. V\ y$  in exI) fastforce
        qed
      qed auto
    qed
  qed

```

lemma *open_map_in_subtopology*:

```

   $\text{openin } Y\ S$ 
 $\implies \text{open\_map } X\ (\text{subtopology } Y\ S)\ f \longleftrightarrow \text{open\_map } X\ Y\ f \wedge f \in \text{topspace } X \rightarrow S$ 
  by (metis image_subset_iff_funcset open_map_def open_map_into_subtopology openin_imp_subset openin_topospace openin_trans_full)

```

lemma *open_map_from_open_subtopology*:

```

 $\llbracket \text{openin } Y\ S; \text{open\_map } X\ (\text{subtopology } Y\ S)\ f \rrbracket \implies \text{open\_map } X\ Y\ f$ 
using open_map_in_subtopology by blast

```

lemma *closed_map_in_subtopology*:

```

   $\text{closedin } Y\ S$ 
 $\implies \text{closed\_map } X\ (\text{subtopology } Y\ S)\ f \longleftrightarrow (\text{closed\_map } X\ Y\ f \wedge f \in \text{topspace } X \rightarrow S)$ 
by (metis closed_map_def closed_map_imp_subset_topospace closed_map_into_subtopology)

```

$\text{closedin_closed_subtopology closedin_subset topspace_subtopology_subset}$)

lemma *closed_map_from_closed_subtopology*:

```

 $\llbracket \text{closedin } Y\ S; \text{closed\_map } X\ (\text{subtopology } Y\ S)\ f \rrbracket \implies \text{closed\_map } X\ Y\ f$ 
using closed_map_in_subtopology by blast

```

lemma *closed_map_from_composition_left*:
assumes *cmf*: *closed_map* $X\ Z$ $(g \circ f)$ **and** *contf*: *continuous_map* $X\ Y\ f$ **and**
fm: $f \text{ ' } \textit{topspace}\ X = \textit{topspace}\ Y$
shows *closed_map* $Y\ Z\ g$
unfolding *closed_map_def*
proof (*intro strip*)
fix U **assume** *closedin* $Y\ U$
then have *closedin* $X\ \{x \in \textit{topspace}\ X. f\ x \in U\}$
using *contf* *closedin_continuous_map_preimage* **by** *blast*
then have *closedin* $Z\ ((g \circ f) \text{ ' } \{x \in \textit{topspace}\ X. f\ x \in U\})$
using *cmf* *closed_map_def* **by** *blast*
moreover
have $\bigwedge y. y \in U \implies \exists x \in \textit{topspace}\ X. f\ x \in U \wedge g\ y = g\ (f\ x)$
by (*smt* (*verit*, *ccfv_SIG*) $\langle \textit{closedin}\ Y\ U \rangle$ *closedin_subset_fm_image_iff_subsetD*)
then have $(g \circ f) \text{ ' } \{x \in \textit{topspace}\ X. f\ x \in U\} = g \text{ ' } U$ **by** *auto*
ultimately show *closedin* $Z\ (g \text{ ' } U)$
by *metis*
qed

identical proof as the above

lemma *open_map_from_composition_left*:
assumes *cmf*: *open_map* $X\ Z\ (g \circ f)$ **and** *contf*: *continuous_map* $X\ Y\ f$ **and**
fm: $f \text{ ' } \textit{topspace}\ X = \textit{topspace}\ Y$
shows *open_map* $Y\ Z\ g$
unfolding *open_map_def*
proof (*intro strip*)
fix U **assume** *openin* $Y\ U$
then have *openin* $X\ \{x \in \textit{topspace}\ X. f\ x \in U\}$
using *contf* *openin_continuous_map_preimage* **by** *blast*
then have *openin* $Z\ ((g \circ f) \text{ ' } \{x \in \textit{topspace}\ X. f\ x \in U\})$
using *cmf* *open_map_def* **by** *blast*
moreover
have $\bigwedge y. y \in U \implies \exists x \in \textit{topspace}\ X. f\ x \in U \wedge g\ y = g\ (f\ x)$
by (*smt* (*verit*, *ccfv_SIG*) $\langle \textit{openin}\ Y\ U \rangle$ *openin_subset_fm_image_iff_subsetD*)
then have $(g \circ f) \text{ ' } \{x \in \textit{topspace}\ X. f\ x \in U\} = g \text{ ' } U$ **by** *auto*
ultimately show *openin* $Z\ (g \text{ ' } U)$
by *metis*
qed

lemma *closed_map_from_composition_right*:
assumes *cmf*: *closed_map* $X\ Z\ (g \circ f)$ $f \in \textit{topspace}\ X \rightarrow \textit{topspace}\ Y$ *continuous_map* $Y\ Z\ g$ *inj_on* $g\ (\textit{topspace}\ Y)$
shows *closed_map* $X\ Y\ f$
unfolding *closed_map_def*
proof (*intro strip*)
fix C **assume** *closedin* $X\ C$
have $\bigwedge y\ c. \llbracket y \in \textit{topspace}\ Y; g\ y = g\ (f\ c); c \in C \rrbracket \implies y \in f \text{ ' } C$

```

    using ‹closedin X C› assms closedin_subset inj_onD by fastforce
  then
  have f' C = {x ∈ topspace Y. g x ∈ (g ∘ f) ' C}
    using ‹closedin X C› assms(2) closedin_subset by fastforce
  moreover have closedin Z ((g ∘ f) ' C)
    using ‹closedin X C› cmf closed_map_def by blast
  ultimately show closedin Y (f ' C)
    using assms(3) closedin_continuous_map_preimage by fastforce
qed

```

identical proof as the above

```

lemma open_map_from_composition_right:
  assumes open_map X Z (g ∘ f) f ∈ topspace X → topspace Y continuous_map
  Y Z g inj_on g (topspace Y)
  shows open_map X Y f
  unfolding open_map_def
proof (intro strip)
  fix C assume openin X C
  have ∧y c. [y ∈ topspace Y; g y = g (f c); c ∈ C] ⇒ y ∈ f ' C
    using ‹openin X C› assms openin_subset inj_onD by fastforce
  then
  have f' C = {x ∈ topspace Y. g x ∈ (g ∘ f) ' C}
    using ‹openin X C› assms(2) openin_subset by fastforce
  moreover have openin Z ((g ∘ f) ' C)
    using ‹openin X C› assms(1) open_map_def by blast
  ultimately show openin Y (f ' C)
    using assms(3) openin_continuous_map_preimage by fastforce
qed

```

1.13.12 Quotient maps

definition *quotient_map* where

```

quotient_map X X' f ‹↔›
  f ' (topspace X) = topspace X' ∧
  (∀ U. U ⊆ topspace X' → (openin X {x. x ∈ topspace X ∧ f x ∈ U} ‹↔›
  openin X' U))

```

lemma *quotient_map_eq*:

```

assumes quotient_map X X' f ∧x. x ∈ topspace X ⇒ f x = g x
shows quotient_map X X' g
by (smt (verit) Collect_cong assms image_cong quotient_map_def)

```

lemma *quotient_map_compose*:

```

assumes f: quotient_map X X' f and g: quotient_map X' X'' g
shows quotient_map X X'' (g ∘ f)
unfolding quotient_map_def
proof (intro conjI allI impI)
  show (g ∘ f) ' topspace X = topspace X''
    using assms by (simp only: image_comp [symmetric]) (simp add: quotient_map_def)

```

```

next
  fix U''
  assume U'': U'' ⊆ topspace X''
  define U' where U' ≡ {y ∈ topspace X'. g y ∈ U''}
  have U' ⊆ topspace X'
    by (auto simp add: U'_def)
  then have U': openin X {x ∈ topspace X. f x ∈ U'} = openin X' U'
    using assms unfolding quotient_map_def by simp
  have {x ∈ topspace X. f x ∈ topspace X' ∧ g (f x) ∈ U''} = {x ∈ topspace X.
    (g ∘ f) x ∈ U''}
    using f quotient_map_def by fastforce
  then show openin X {x ∈ topspace X. (g ∘ f) x ∈ U''} = openin X'' U''
    by (smt (verit, best) Collect_cong U' U'_def U'' g mem_Collect_eq quo-
    tient_map_def)
qed

```

```

lemma quotient_map_from_composition:
  assumes f: continuous_map X X' f and g: continuous_map X' X'' g and gf:
  quotient_map X X'' (g ∘ f)
  shows quotient_map X' X'' g
  unfolding quotient_map_def
proof (intro conjI allI impI)
  show g ' topspace X' = topspace X''
    using assms unfolding continuous_map_def quotient_map_def by fastforce
next
  fix U'' :: 'c set
  assume U'': U'' ⊆ topspace X''
  have eq: {x ∈ topspace X. g (f x) ∈ U''} = {x ∈ topspace X. f x ∈ {y. y ∈
  topspace X' ∧ g y ∈ U''}}
    using continuous_map_def f by fastforce
  show openin X' {x ∈ topspace X'. g x ∈ U''} = openin X'' U''
    using assms unfolding continuous_map_def quotient_map_def
    by (metis (mono_tags, lifting) Collect_cong U'' comp_apply eq)
qed

```

```

lemma quotient_imp_continuous_map:
  quotient_map X X' f ⇒ continuous_map X X' f
  by (simp add: continuous_map openin_subset quotient_map_def)

```

```

lemma quotient_imp_surjective_map:
  quotient_map X X' f ⇒ f ' (topspace X) = topspace X'
  by (simp add: quotient_map_def)

```

```

lemma quotient_map_closedin:
  quotient_map X X' f ↔
  f ' (topspace X) = topspace X' ∧
  (∀ U. U ⊆ topspace X' → (closedin X {x. x ∈ topspace X ∧ f x ∈ U} ↔
  closedin X' U))
proof -

```

```

have eq: (topspace X - {x ∈ topspace X. f x ∈ U}) = {x ∈ topspace X. f x ∈
topspace X' ∧ f x ∉ U}
  if f ' topspace X = topspace X' U' ⊆ topspace X' for U'
  using that by auto
have (∀ U ⊆ topspace X'. openin X {x ∈ topspace X. f x ∈ U} = openin X' U)
=
  (∀ U ⊆ topspace X'. closedin X {x ∈ topspace X. f x ∈ U} = closedin X'
U)
  if f ' topspace X = topspace X'
proof (rule iffI; intro allI impI subsetI)
  fix U'
  assume *[rule_format]: ∀ U ⊆ topspace X'. openin X {x ∈ topspace X. f x ∈
U} = openin X' U
  and U': U' ⊆ topspace X'
  show closedin X {x ∈ topspace X. f x ∈ U'} = closedin X' U'
  using U' by (auto simp add: closedin_def simp flip: * [of topspace X' - U']
eq [OF that])
  next
  fix U' :: 'b set
  assume *[rule_format]: ∀ U ⊆ topspace X'. closedin X {x ∈ topspace X. f x ∈
U} = closedin X' U
  and U': U' ⊆ topspace X'
  show openin X {x ∈ topspace X. f x ∈ U'} = openin X' U'
  using U' by (auto simp add: openin_closedin_eq simp flip: * [of topspace
X' - U'] eq [OF that])
  qed
then show ?thesis
  unfolding quotient_map_def by force
qed

```

lemma continuous_open_imp_quotient_map:

```

assumes continuous_map X X' f and om: open_map X X' f and feq: f '
(topspace X) = topspace X'
shows quotient_map X X' f
proof -
  { fix U
    assume U: U ⊆ topspace X' and openin X {x ∈ topspace X. f x ∈ U}
    then have ope: openin X' (f ' {x ∈ topspace X. f x ∈ U})
    using om unfolding open_map_def by blast
    then have openin X' U
    using U feq by (subst openin_subopen) force
  }
  moreover have openin X {x ∈ topspace X. f x ∈ U} if U ⊆ topspace X' and
openin X' U for U
  using that assms unfolding continuous_map_def by blast
  ultimately show ?thesis
  unfolding quotient_map_def using assms by blast
qed

```

lemma *continuous_closed_imp_quotient_map*:

assumes *continuous_map* $X X' f$ **and** *om*: *closed_map* $X X' f$ **and** *feq*: $f' (topspace X) = topspace X'$

shows *quotient_map* $X X' f$

proof –

have $f' \{x \in topspace X. f x \in U\} = U$ **if** $U \subseteq topspace X'$ **for** U

using *that feq by auto*

with *assms show ?thesis*

unfolding *quotient_map_closedin closed_map_def continuous_map_closedin*

by *auto*

qed

lemma *continuous_open_quotient_map*:

$\llbracket \text{continuous_map } X X' f; \text{ open_map } X X' f \rrbracket \implies \text{quotient_map } X X' f \longleftrightarrow f' (topspace X) = topspace X'$

by (*meson continuous_open_imp_quotient_map quotient_map_def*)

lemma *continuous_closed_quotient_map*:

$\llbracket \text{continuous_map } X X' f; \text{ closed_map } X X' f \rrbracket \implies \text{quotient_map } X X' f \longleftrightarrow f' (topspace X) = topspace X'$

by (*meson continuous_closed_imp_quotient_map quotient_map_def*)

lemma *injective_quotient_map*:

assumes *inj_on* $f (topspace X)$

shows *quotient_map* $X X' f \longleftrightarrow$

$\text{continuous_map } X X' f \wedge \text{open_map } X X' f \wedge \text{closed_map } X X' f \wedge f' (topspace X) = topspace X'$

(*is ?lhs = ?rhs*)

(*is ?lhs = ?rhs*)

proof

assume L : *?lhs*

have *om*: *open_map* $X X' f$

proof (*clarsimp simp add: open_map_def*)

fix U

assume *openin* $X U$

then have $U \subseteq topspace X$

by (*simp add: openin_subset*)

moreover have $\{x \in topspace X. f x \in f' U\} = U$

using $\langle U \subseteq topspace X \rangle$ *assms inj_onD* **by** *fastforce*

ultimately show *openin* $X' (f' U)$

using L **unfolding** *quotient_map_def*

by (*metis (no_types, lifting) Collect_cong <openin X U> image_mono*)

qed

then have *closed_map* $X X' f$

by (*simp add: L assms bijective_open_imp_closed_map quotient_imp_surjective_map*)

then show *?rhs*

using L *om* **by** (*simp add: quotient_imp_continuous_map quotient_imp_surjective_map*)

next

assume *?rhs*

then show *?lhs*

by (simp add: continuous_closed_imp_quotient_map)
qed

lemma continuous_compose_quotient_map:

assumes f : quotient_map $X X'$ f and g : continuous_map $X X''$ $(g \circ f)$
shows continuous_map $X' X''$ g
unfolding quotient_map_def continuous_map_def
proof (intro conjI ballI allI impI)
show $g \in \text{topspace } X' \rightarrow \text{topspace } X''$
using assms unfolding quotient_map_def Pi_iff
by (metis (no_types, opaque_lifting) continuous_map_image_subset_topspace
image_comp image_subset_iff)
next
fix $U'' :: 'c \text{ set}$
assume U'' : openin $X'' U''$
have $f' \text{ topspace } X = \text{topspace } X'$
by (simp add: f quotient_imp_surjective_map)
then have eq: $\{x \in \text{topspace } X. f x \in \text{topspace } X' \wedge g (f x) \in U\} = \{x \in \text{topspace } X. g (f x) \in U\}$ for U
by auto
have openin $X \{x \in \text{topspace } X. f x \in \text{topspace } X' \wedge g (f x) \in U''\}$
unfolding eq using U'' g openin_continuous_map_preimage by fastforce
then have *: openin $X \{x \in \text{topspace } X. f x \in \{x \in \text{topspace } X'. g x \in U''\}\}$
by auto
show openin $X' \{x \in \text{topspace } X'. g x \in U''\}$
using f unfolding quotient_map_def
by (metis (no_types) Collect_subset *)
qed

lemma continuous_compose_quotient_map_eq:

quotient_map $X X' f \implies \text{continuous_map } X X'' (g \circ f) \iff \text{continuous_map } X' X'' g$
using continuous_compose_quotient_map continuous_map_compose quotient_imp_continuous_map
by blast

lemma quotient_map_compose_eq:

quotient_map $X X' f \implies \text{quotient_map } X X'' (g \circ f) \iff \text{quotient_map } X' X'' g$
by (meson continuous_compose_quotient_map_eq quotient_imp_continuous_map
quotient_map_compose quotient_map_from_composition)

lemma quotient_map_restriction:

assumes quo: quotient_map $X Y f$ and U : $\{x \in \text{topspace } X. f x \in V\} = U$ and
disj: openin $Y V \vee \text{closedin } Y V$
shows quotient_map (subtopology $X U$) (subtopology $Y V$) f
using disj
proof
assume V : openin $Y V$
with U have sub: $U \subseteq \text{topspace } X \ V \subseteq \text{topspace } Y$

```

    by (auto simp: openin_subset)
  have fim: f ' topspace X = topspace Y
    and Y:  $\bigwedge U. U \subseteq \text{topspace } Y \implies \text{openin } X \{x \in \text{topspace } X. f x \in U\} =$ 
    openin Y U
    using quo unfolding quotient_map_def by auto
  have openin X U
    using U V Y sub(2) by blast
  show ?thesis
    unfolding quotient_map_def
  proof (intro conjI allI impI)
    show f ' topspace (subtopology X U) = topspace (subtopology Y V)
      using sub U fim by (auto)
    next
      fix Y' :: 'b set
      assume Y'  $\subseteq$  topspace (subtopology Y V)
      then have Y'  $\subseteq$  topspace Y Y'  $\subseteq$  V
        by (simp_all)
      then have eq:  $\{x \in \text{topspace } X. x \in U \wedge f x \in Y'\} = \{x \in \text{topspace } X. f x \in$ 
      Y'\}
        using U by blast
      then show openin (subtopology X U)  $\{x \in \text{topspace } (subtopology X U). f x \in$ 
      Y'\} = openin (subtopology Y V) Y'
        using U V Y  $\langle$ openin X U $\rangle$   $\langle$ Y'  $\subseteq$  topspace Y $\rangle$   $\langle$ Y'  $\subseteq$  V $\rangle$ 
        by (simp add: openin_open_subtopology eq) (auto simp: openin_closedin_eq)
      qed
    next
      assume V: closedin Y V
      with U have sub: U  $\subseteq$  topspace X V  $\subseteq$  topspace Y
        by (auto simp: closedin_subset)
      have fim: f ' topspace X = topspace Y
        and Y:  $\bigwedge U. U \subseteq \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x \in U\} =$ 
        closedin Y U
        using quo unfolding quotient_map_closedin by auto
      have closedin X U
        using U V Y sub(2) by blast
      show ?thesis
        unfolding quotient_map_closedin
      proof (intro conjI allI impI)
        show f ' topspace (subtopology X U) = topspace (subtopology Y V)
          using sub U fim by (auto)
        next
          fix Y' :: 'b set
          assume Y'  $\subseteq$  topspace (subtopology Y V)
          then have Y'  $\subseteq$  topspace Y Y'  $\subseteq$  V
            by (simp_all)
          then have eq:  $\{x \in \text{topspace } X. x \in U \wedge f x \in Y'\} = \{x \in \text{topspace } X. f x \in$ 
          Y'\}
            using U by blast
          then show closedin (subtopology X U)  $\{x \in \text{topspace } (subtopology X U). f x \in$ 

```

```


$$Y\} = \text{closedin } (\text{subtopology } Y \ V) \ Y'$$

using  $U \ V \ Y \ \langle \text{closedin } X \ U \rangle \ \langle Y' \subseteq \text{topspace } Y \rangle \ \langle Y' \subseteq V \rangle$ 
by (simp add: closedin_closed_subtopology eq) (auto simp: closedin_def)
qed
qed

lemma quotient_map_saturated_open:
  quotient_map  $X \ Y \ f \ \longleftrightarrow$ 
  continuous_map  $X \ Y \ f \ \wedge \ f' (\text{topspace } X) = \text{topspace } Y \ \wedge$ 
  ( $\forall U. \text{openin } X \ U \ \wedge \ \{x \in \text{topspace } X. f \ x \in f' \ U\} \subseteq U \ \longrightarrow \text{openin } Y \ (f' \ U)$ )
  (is ?lhs = ?rhs)
proof
  assume  $L: ?lhs$ 
  then have  $\text{fim}: f' \ \text{topspace } X = \text{topspace } Y$ 
  and  $Y: \bigwedge U. U \subseteq \text{topspace } Y \ \Longrightarrow \text{openin } Y \ U = \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ 
  unfolding quotient_map_def by auto
  show ?rhs
  proof (intro conjI allI impI)
    show continuous_map  $X \ Y \ f$ 
    by (simp add: L quotient_imp_continuous_map)
    show  $f' \ \text{topspace } X = \text{topspace } Y$ 
    by (simp add: fim)
  next
    fix  $U :: 'a \ \text{set}$ 
    assume  $U: \text{openin } X \ U \ \wedge \ \{x \in \text{topspace } X. f \ x \in f' \ U\} \subseteq U$ 
    then have  $\text{sub}: f' \ U \subseteq \text{topspace } Y$  and  $\text{eq}: \{x \in \text{topspace } X. f \ x \in f' \ U\} = U$ 
    using fim openin_subset by fastforce+
    show  $\text{openin } Y \ (f' \ U)$ 
    by (simp add: sub Y eq U)
  qed
next
  assume ?rhs
  then have  $YX: \bigwedge U. \text{openin } Y \ U \ \Longrightarrow \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ 
  and  $\text{fim}: f' \ \text{topspace } X = \text{topspace } Y$ 
  and  $XY: \bigwedge U. \llbracket \text{openin } X \ U; \{x \in \text{topspace } X. f \ x \in f' \ U\} \subseteq U \rrbracket \Longrightarrow \text{openin } Y \ (f' \ U)$ 
  by (auto simp: quotient_map_def continuous_map_def)
  show ?lhs
  proof (simp add: quotient_map_def fim, intro allI impI iffI)
    fix  $U :: 'b \ \text{set}$ 
    assume  $U \subseteq \text{topspace } Y$  and  $X: \text{openin } X \ \{x \in \text{topspace } X. f \ x \in U\}$ 
    have  $\text{feq}: f' \ \{x \in \text{topspace } X. f \ x \in U\} = U$ 
    using  $\langle U \subseteq \text{topspace } Y \rangle$  fim by auto
    show  $\text{openin } Y \ U$ 
    using XY [OF X] by (simp add: feq)
  next

```

```

    fix U :: 'b set
    assume U ⊆ topspace Y and Y: openin Y U
    show openin X {x ∈ topspace X. f x ∈ U}
      by (metis YX [OF Y])
  qed
qed

lemma quotient_map_saturated_closed:
  quotient_map X Y f ⟷
    continuous_map X Y f ∧ f ' (topspace X) = topspace Y ∧
    (∀ U. closedin X U ∧ {x ∈ topspace X. f x ∈ f ' U} ⊆ U ⟶ closedin Y (f
  ' U))
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then obtain fim: f ' topspace X = topspace Y
    and Y: ⋀ U. U ⊆ topspace Y ⟹ closedin Y U = closedin X {x ∈ topspace
  X. f x ∈ U}
    by (simp add: quotient_map_closedin)
  show ?rhs
  proof (intro conjI allI impI)
    show continuous_map X Y f
      by (simp add: L quotient_imp_continuous_map)
    show f ' topspace X = topspace Y
      by (simp add: fim)
  next
    fix U :: 'a set
    assume U: closedin X U ∧ {x ∈ topspace X. f x ∈ f ' U} ⊆ U
    then have sub: f ' U ⊆ topspace Y and eq: {x ∈ topspace X. f x ∈ f ' U} =
  U
      using fim closedin_subset by fastforce+
    show closedin Y (f ' U)
      by (simp add: sub Y eq U)
  qed
next
  assume ?rhs
  then obtain YX: ⋀ U. closedin Y U ⟹ closedin X {x ∈ topspace X. f x ∈ U}
    and fim: f ' topspace X = topspace Y
    and XY: ⋀ U. [closedin X U; {x ∈ topspace X. f x ∈ f ' U} ⊆ U] ⟹ closedin
  Y (f ' U)
    by (simp add: continuous_map_closedin)
  show ?lhs
  proof (simp add: quotient_map_closedin fim, intro allI impI iffI)
    fix U :: 'b set
    assume U ⊆ topspace Y and X: closedin X {x ∈ topspace X. f x ∈ U}
    have feq: f ' {x ∈ topspace X. f x ∈ U} = U
      using ⟨U ⊆ topspace Y⟩ fim by auto
    show closedin Y U
      using XY [OF X] by (simp add: feq)
  qed

```

```

next
  fix U :: 'b set
  assume U  $\subseteq$  topspace Y and Y: closedin Y U
  show closedin X {x  $\in$  topspace X. f x  $\in$  U}
    by (metis YX [OF Y])
qed
qed

lemma quotient_map_onto_image:
  assumes f ' topspace X  $\subseteq$  topspace Y and U:  $\bigwedge$ U. U  $\subseteq$  topspace Y  $\implies$  openin
  X {x  $\in$  topspace X. f x  $\in$  U} = openin Y U
  shows quotient_map X (subtopology Y (f ' topspace X)) f
  unfolding quotient_map_def topspace_subtopology
proof (intro conjI strip)
  fix U
  assume U  $\subseteq$  topspace Y  $\cap$  f ' topspace X
  with U have openin X {x  $\in$  topspace X. f x  $\in$  U}  $\implies$   $\exists$ x. openin Y x  $\wedge$  U =
  f ' topspace X  $\cap$  x
    by fastforce
  moreover have  $\exists$ x. openin Y x  $\wedge$  U = f ' topspace X  $\cap$  x  $\implies$  openin X {x  $\in$ 
  topspace X. f x  $\in$  U}
    by (metis (mono_tags, lifting) Collect_cong IntE IntI U image_eqI openin_subset)
  ultimately show openin X {x  $\in$  topspace X. f x  $\in$  U} = openin (subtopology Y
  (f ' topspace X)) U
    by (force simp: openin_subtopology_alt image_iff)
qed (use assms in auto)

lemma quotient_map_lift_exists:
  assumes f: quotient_map X Y f and h: continuous_map X Z h
  and  $\bigwedge$ x y.  $\llbracket$ x  $\in$  topspace X; y  $\in$  topspace X; f x = f y $\rrbracket \implies$  h x = h y
  obtains g where continuous_map Y Z g g ' topspace Y = h ' topspace X
   $\bigwedge$ x. x  $\in$  topspace X  $\implies$  g(f x) = h x
proof -
  obtain g where g:  $\bigwedge$ x. x  $\in$  topspace X  $\implies$  h x = g(f x)
  using function_factors_left_gen[of  $\lambda$ x. x  $\in$  topspace X f h] assms by blast
  show ?thesis
  proof
    show g ' topspace Y = h ' topspace X
      using f g by (force dest!: quotient_imp_surjective_map)
    show continuous_map Y Z g
      by (smt (verit) f g h continuous_compose_quotient_map_eq continuous_map_eq o_def)
  qed (simp add: g)
qed

```

1.13.13 Separated Sets

```

definition separatedin :: 'a topology  $\Rightarrow$  'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool
  where separatedin X S T  $\equiv$ 

```

$$S \subseteq \text{topspace } X \wedge T \subseteq \text{topspace } X \wedge \\ S \cap X \text{ closure_of } T = \{\} \wedge T \cap X \text{ closure_of } S = \{\}$$

lemma *separatedin_empty* [*simp*]:

$$\text{separatedin } X \ S \ \{\} \longleftrightarrow S \subseteq \text{topspace } X \\ \text{separatedin } X \ \{\} \ S \longleftrightarrow S \subseteq \text{topspace } X$$

by (*simp_all add: separatedin_def*)

lemma *separatedin_refl* [*simp*]:

$$\text{separatedin } X \ S \ S \longleftrightarrow S = \{\}$$

by (*metis closure_of_subset empty_subsetI inf.orderE separatedin_def*)

lemma *separatedin_sym*:

$$\text{separatedin } X \ S \ T \longleftrightarrow \text{separatedin } X \ T \ S$$

by (*auto simp: separatedin_def*)

lemma *separatedin_imp_disjoint*:

$$\text{separatedin } X \ S \ T \implies \text{disjnt } S \ T$$

by (*meson closure_of_subset disjnt_def disjnt_subset2 separatedin_def*)

lemma *separatedin_mono*:

$$\llbracket \text{separatedin } X \ S \ T; S' \subseteq S; T' \subseteq T \rrbracket \implies \text{separatedin } X \ S' \ T'$$

unfolding *separatedin_def*

using *closure_of_mono* **by** *blast*

lemma *separatedin_open_sets*:

$$\llbracket \text{openin } X \ S; \text{openin } X \ T \rrbracket \implies \text{separatedin } X \ S \ T \longleftrightarrow \text{disjnt } S \ T$$

unfolding *disjnt_def separatedin_def*

by (*auto simp: openin_Int_closure_of_eq_empty openin_subset*)

lemma *separatedin_closed_sets*:

$$\llbracket \text{closedin } X \ S; \text{closedin } X \ T \rrbracket \implies \text{separatedin } X \ S \ T \longleftrightarrow \text{disjnt } S \ T$$

unfolding *closure_of_eq disjnt_def separatedin_def*

by (*metis closedin_def closure_of_eq inf_commute*)

lemma *separatedin_subtopology*:

$$\text{separatedin } (\text{subtopology } X \ U) \ S \ T \longleftrightarrow S \subseteq U \wedge T \subseteq U \wedge \text{separatedin } X \ S \ T$$

by (*auto simp: separatedin_def closure_of_subtopology Int_ac disjoint_iff elim!: inf.orderE*)

lemma *separatedin_discrete_topology*:

$$\text{separatedin } (\text{discrete_topology } U) \ S \ T \longleftrightarrow S \subseteq U \wedge T \subseteq U \wedge \text{disjnt } S \ T$$

by (*metis openin_discrete_topology separatedin_def separatedin_open_sets topspace_discrete_topology*)

lemma *separated_eq_distinguishable*:

$$\text{separatedin } X \ \{x\} \ \{y\} \longleftrightarrow$$

$$x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$$

$$(\exists U. \text{openin } X \ U \wedge x \in U \wedge (y \notin U)) \wedge$$

$$(\exists v. \text{openin } X \ v \wedge y \in v \wedge (x \notin v))$$

by (force simp: separatedin_def closure_of_def)

lemma separatedin_Un [simp]:

$separatedin\ X\ S\ (T \cup U) \longleftrightarrow separatedin\ X\ S\ T \wedge separatedin\ X\ S\ U$
 $separatedin\ X\ (S \cup T)\ U \longleftrightarrow separatedin\ X\ S\ U \wedge separatedin\ X\ T\ U$
by (auto simp: separatedin_def)

lemma separatedin_Union:

$finite\ \mathcal{F} \implies separatedin\ X\ S\ (\bigcup \mathcal{F}) \longleftrightarrow S \subseteq topspace\ X \wedge (\forall T \in \mathcal{F}. separatedin\ X\ S\ T)$
 $finite\ \mathcal{F} \implies separatedin\ X\ (\bigcup \mathcal{F})\ S \longleftrightarrow (\forall T \in \mathcal{F}. separatedin\ X\ S\ T) \wedge S \subseteq topspace\ X$
by (auto simp: separatedin_def closure_of_Union)

lemma separatedin_openin_diff:

$\llbracket openin\ X\ S; openin\ X\ T \rrbracket \implies separatedin\ X\ (S - T)\ (T - S)$
unfolding separatedin_def
by (metis Diff_Int_distrib2 Diff_disjoint Diff_empty Diff_mono empty_Diff empty_subsetI openin_Int_closure_of_eq_empty openin_subset)

lemma separatedin_closedin_diff:

assumes closedin X S closedin X T
shows separatedin X (S - T) (T - S)
proof -
have $S - T \subseteq topspace\ X$ $T - S \subseteq topspace\ X$
using assms closedin_subset **by** auto
with assms **show** ?thesis
by (simp add: separatedin_def Diff_Int_distrib2 closure_of_minimal inf_absorb2)
qed

lemma separation_closedin_Un_gen:

$separatedin\ X\ S\ T \longleftrightarrow$
 $S \subseteq topspace\ X \wedge T \subseteq topspace\ X \wedge disjnt\ S\ T \wedge$
 $closedin\ (subtopology\ X\ (S \cup T))\ S \wedge$
 $closedin\ (subtopology\ X\ (S \cup T))\ T$
by (auto simp add: separatedin_def closedin_Int_closure_of_disjnt_iff dest: closure_of_subset)

lemma separation_openin_Un_gen:

$separatedin\ X\ S\ T \longleftrightarrow$
 $S \subseteq topspace\ X \wedge T \subseteq topspace\ X \wedge disjnt\ S\ T \wedge$
 $openin\ (subtopology\ X\ (S \cup T))\ S \wedge$
 $openin\ (subtopology\ X\ (S \cup T))\ T$
unfolding openin_closedin_eq_topspace_subtopology separation_closedin_Un_gen disjnt_def
by (auto simp: Diff_triv Int_commute Un_Diff inf_absorb1 topspace_def)

lemma separatedin_full:

$S \cup T = topspace\ X$

$\implies \text{separatedin } X S T \iff \text{disjnt } S T \wedge \text{closedin } X S \wedge \text{openin } X S \wedge \text{closedin } X T \wedge \text{openin } X T$
by (*metis separatedin_open_sets separation_closedin_Un_gen separation_openin_Un_gen subtopology_topspace*)

1.13.14 Homeomorphisms

(1-way and 2-way versions may be useful in places)

definition *homeomorphic_map* :: 'a topology \Rightarrow 'b topology \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool
where
homeomorphic_map X Y f \equiv *quotient_map* X Y f \wedge *inj_on* f (*topspace* X)

definition *homeomorphic_maps* :: 'a topology \Rightarrow 'b topology \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow bool
where
homeomorphic_maps X Y f g \equiv
continuous_map X Y f \wedge *continuous_map* Y X g \wedge
 $(\forall x \in \text{topspace } X. g(f\ x) = x) \wedge (\forall y \in \text{topspace } Y. f(g\ y) = y)$

lemma *homeomorphic_map_eq*:
 $[[\text{homeomorphic_map } X Y f; \bigwedge x. x \in \text{topspace } X \implies f\ x = g\ x]] \implies \text{homeomorphic_map } X Y g$
by (*meson homeomorphic_map_def inj_on_cong quotient_map_eq*)

lemma *homeomorphic_maps_eq*:
 $[[\text{homeomorphic_maps } X Y f g; \bigwedge x. x \in \text{topspace } X \implies f\ x = f'\ x; \bigwedge y. y \in \text{topspace } Y \implies g\ y = g'\ y]] \implies \text{homeomorphic_maps } X Y f' g'$
unfolding *homeomorphic_maps_def*
by (*metis continuous_map_eq continuous_map_image_subset_topspace image_subset_iff*)

lemma *homeomorphic_maps_sym*:
 $\text{homeomorphic_maps } X Y f g \iff \text{homeomorphic_maps } Y X g f$
by (*auto simp: homeomorphic_maps_def*)

lemma *homeomorphic_maps_id*:
 $\text{homeomorphic_maps } X Y \text{id id} \iff Y = X$ (**is** ?lhs = ?rhs)

proof

assume L: ?lhs

then have *topspace* X = *topspace* Y

by (*auto simp: homeomorphic_maps_def continuous_map_def*)

with L **show** ?rhs

unfolding *homeomorphic_maps_def*

by (*metis topology_finer_continuous_id topology_eq*)

next

assume ?rhs

then show ?lhs

unfolding *homeomorphic_maps_def* **by** *auto*
qed

lemma *homeomorphic_map_id* [*simp*]: $\text{homeomorphic_map } X \ Y \ \text{id} \longleftrightarrow Y = X$
 (**is** *?lhs = ?rhs*)

proof

assume *L: ?lhs*

then have *eq: topspace X = topspace Y*

by (*auto simp: homeomorphic_map_def continuous_map_def quotient_map_def*)

then have $\bigwedge S. \text{openin } X \ S \longrightarrow \text{openin } Y \ S$

by (*meson L homeomorphic_map_def injective_quotient_map topology_finer_open_id*)

then show *?rhs*

using *L* **unfolding** *homeomorphic_map_def*

by (*metis eq quotient_imp_continuous_map topology_eq topology_finer_continuous_id*)

next

assume *?rhs*

then show *?lhs*

unfolding *homeomorphic_map_def*

by (*simp add: closed_map_id continuous_closed_imp_quotient_map*)

qed

lemma *homeomorphic_map_compose*:

assumes *homeomorphic_map X Y f homeomorphic_map Y X'' g*

shows *homeomorphic_map X X'' (g ∘ f)*

proof –

have *inj_on g (f ` topspace X)*

by (*metis (no_types) assms homeomorphic_map_def quotient_imp_surjective_map*)

then show *?thesis*

using *assms* **by** (*meson comp_inj_on homeomorphic_map_def quotient_map_compose_eq*)

qed

lemma *homeomorphic_maps_compose*:

homeomorphic_maps X Y f h ∧

homeomorphic_maps Y X'' g k

$\implies \text{homeomorphic_maps } X \ X'' \ (g \circ f) \ (h \circ k)$

unfolding *homeomorphic_maps_def*

by (*auto simp: continuous_map_compose; simp add: continuous_map_def Pi_iff*)

lemma *homeomorphic_eq_everything_map*:

homeomorphic_map X Y f \longleftrightarrow

continuous_map X Y f ∧ open_map X Y f ∧ closed_map X Y f ∧

f ` (topspace X) = topspace Y ∧ inj_on f (topspace X)

unfolding *homeomorphic_map_def*

by (*force simp: injective_quotient_map intro: injective_quotient_map*)

lemma *homeomorphic_imp_continuous_map*:

homeomorphic_map X Y f $\implies \text{continuous_map } X \ Y \ f$

by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_open_map*:

homeomorphic_map X Y f \implies *open_map X Y f*
by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_closed_map*:

homeomorphic_map X Y f \implies *closed_map X Y f*
by (*simp add: homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_surjective_map*:

homeomorphic_map X Y f \implies *f ' topspace X = topspace Y*
using *homeomorphic_eq_everything_map* **by** *fastforce*

lemma *homeomorphic_imp_injective_map*:

homeomorphic_map X Y f \implies *inj_on f (topspace X)*
by (*simp add: homeomorphic_eq_everything_map*)

lemma *bijjective_open_imp_homeomorphic_map*:

\llbracket *continuous_map X Y f*; *open_map X Y f*; *f ' (topspace X) = topspace Y*;
*inj_on f (topspace X) \rrbracket
 \implies *homeomorphic_map X Y f*
by (*simp add: homeomorphic_map_def continuous_open_imp_quotient_map*)*

lemma *bijjective_closed_imp_homeomorphic_map*:

\llbracket *continuous_map X Y f*; *closed_map X Y f*; *f ' (topspace X) = topspace Y*;
*inj_on f (topspace X) \rrbracket
 \implies *homeomorphic_map X Y f*
by (*simp add: continuous_closed_quotient_map homeomorphic_map_def*)*

lemma *open_eq_continuous_inverse_map*:

assumes *X*: $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y \wedge g(f x) = x$
and *Y*: $\bigwedge y. y \in \text{topspace } Y \implies g y \in \text{topspace } X \wedge f(g y) = y$
shows *open_map X Y f* \longleftrightarrow *continuous_map Y X g*

proof –

have *eq*: $\{x \in \text{topspace } Y. g x \in U\} = f ' U$ **if** *openin X U* **for** *U*
using *openin_subset [OF that]* **by** (*force simp: X Y image_iff*)
show *?thesis*
by (*auto simp: Y open_map_def continuous_map_def eq*)

qed

lemma *closed_eq_continuous_inverse_map*:

assumes *X*: $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y \wedge g(f x) = x$
and *Y*: $\bigwedge y. y \in \text{topspace } Y \implies g y \in \text{topspace } X \wedge f(g y) = y$
shows *closed_map X Y f* \longleftrightarrow *continuous_map Y X g*

proof –

have *eq*: $\{x \in \text{topspace } Y. g x \in U\} = f ' U$ **if** *closedin X U* **for** *U*
using *closedin_subset [OF that]* **by** (*force simp: X Y image_iff*)
show *?thesis*
by (*auto simp: Y closed_map_def continuous_map_def eq*)

qed

lemma *homeomorphic_maps_map*:

homeomorphic_maps $X Y f g \longleftrightarrow$
 $\text{homeomorphic_map } X Y f \wedge \text{homeomorphic_map } Y X g \wedge$
 $(\forall x \in \text{topspace } X. g(f x) = x) \wedge (\forall y \in \text{topspace } Y. f(g y) = y)$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then have L : *continuous_map* $X Y f$ *continuous_map* $Y X g \forall x \in \text{topspace } X. g$
 $(f x) = x \forall x' \in \text{topspace } Y. f (g x') = x'$

by (*auto simp: homeomorphic_maps_def*)

show ?rhs

proof (*intro conjI bijective_open_imp_homeomorphic_map L*)

show *open_map* $X Y f$

using L **using** *open_eq_continuous_inverse_map* [*of concl: X Y f g*]

by (*simp add: continuous_map_def Pi_iff*)

show *open_map* $Y X g$

using L **using** *open_eq_continuous_inverse_map* [*of concl: Y X g f*]

by (*simp add: continuous_map_def Pi_iff*)

show $f ' \text{topspace } X = \text{topspace } Y g ' \text{topspace } Y = \text{topspace } X$

using L **by** (*force simp: continuous_map_closedin Pi_iff*)+

show *inj_on* f (*topspace* X) *inj_on* g (*topspace* Y)

using L **unfolding** *inj_on_def* **by** *metis*+

qed

next

assume ?rhs

then show ?lhs

by (*auto simp: homeomorphic_maps_def homeomorphic_imp_continuous_map*)

qed

lemma *homeomorphic_maps_imp_map*:

homeomorphic_maps $X Y f g \implies \text{homeomorphic_map } X Y f$
using *homeomorphic_maps_map* **by** *blast*

lemma *homeomorphic_map_maps*:

homeomorphic_map $X Y f \longleftrightarrow (\exists g. \text{homeomorphic_maps } X Y f g)$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then have L : *continuous_map* $X Y f$ *open_map* $X Y f$ *closed_map* $X Y f$
 $f ' (\text{topspace } X) = \text{topspace } Y$ *inj_on* f (*topspace* X)

by (*auto simp: homeomorphic_eq_everything_map*)

have X : $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y \wedge \text{inv_into } (\text{topspace } X) f (f$
 $x) = x$

using L **by** *auto*

have Y : $\bigwedge y. y \in \text{topspace } Y \implies \text{inv_into } (\text{topspace } X) f y \in \text{topspace } X \wedge f$
 $(\text{inv_into } (\text{topspace } X) f y) = y$

by (*simp add: L f_inv_into_f inv_into_into*)

have *homeomorphic_maps* $X Y f$ (*inv_into* (*topspace* X) f)

```

unfolding homeomorphic_maps_def
proof (intro conjI L)
  show continuous_map Y X (inv_into (topspace X) f)
    by (simp add: L X Y flip: open_eq_continuous_inverse_map [where f=f])
next
  show  $\forall x \in \text{topspace } X. \text{inv\_into } (\text{topspace } X) f (f x) = x$ 
     $\forall y \in \text{topspace } Y. f (\text{inv\_into } (\text{topspace } X) f y) = y$ 
    using X Y by auto
qed
then show ?rhs
  by metis
next
  assume ?rhs
  then show ?lhs
    using homeomorphic_maps_map by blast
qed

```

```

lemma homeomorphic_maps_involution:
   $[[\text{continuous\_map } X X f; \bigwedge x. x \in \text{topspace } X \implies f(f x) = x]] \implies \text{homeomor-}$ 
   $\text{phic\_maps } X X f f$ 
  by (auto simp: homeomorphic_maps_def)

```

```

lemma homeomorphic_map_involution:
   $[[\text{continuous\_map } X X f; \bigwedge x. x \in \text{topspace } X \implies f(f x) = x]] \implies \text{homeomor-}$ 
   $\text{phic\_map } X X f$ 
  using homeomorphic_maps_involution homeomorphic_maps_map by blast

```

```

lemma homeomorphic_map_openness:
  assumes hom: homeomorphic_map X Y f and U: U  $\subseteq$  topspace X
  shows openin Y (f ' U)  $\longleftrightarrow$  openin X U
proof –
  obtain g where homeomorphic_maps X Y f g
    using assms by (auto simp: homeomorphic_map_maps)
  then have g: homeomorphic_map Y X g and gf:  $\bigwedge x. x \in \text{topspace } X \implies g(f$ 
x) = x
    by (auto simp: homeomorphic_maps_map)
  then have openin X U  $\implies$  openin Y (f ' U)
    using hom homeomorphic_imp_open_map open_map_def by blast
  show openin Y (f ' U) = openin X U
proof
  assume L: openin Y (f ' U)
  have U = g ' (f ' U)
    using U gf by force
  then show openin X U
    by (metis L homeomorphic_imp_open_map open_map_def g)
next
  assume openin X U
  then show openin Y (f ' U)
    using hom homeomorphic_imp_open_map open_map_def by blast

```

qed
qed

lemma *homeomorphic_map_closedness*:

assumes *hom*: *homeomorphic_map* X Y f **and** U : $U \subseteq \text{topspace } X$
shows $\text{closedin } Y (f \text{ ` } U) \longleftrightarrow \text{closedin } X U$

proof –

obtain g **where** *homeomorphic_maps* X Y f g

using *assms* **by** (*auto simp: homeomorphic_map_maps*)

then have g : *homeomorphic_map* Y X g **and** gf : $\bigwedge x. x \in \text{topspace } X \implies g(f$
 $x) = x$

by (*auto simp: homeomorphic_maps_map*)

then have $\text{closedin } X U \implies \text{closedin } Y (f \text{ ` } U)$

using *hom* *homeomorphic_imp_closed_map* *closed_map_def* **by** *blast*

show $\text{closedin } Y (f \text{ ` } U) = \text{closedin } X U$

proof

assume L : $\text{closedin } Y (f \text{ ` } U)$

have $U = g \text{ ` } (f \text{ ` } U)$

using U gf **by** *force*

then show $\text{closedin } X U$

by (*metis* L *homeomorphic_imp_closed_map* *closed_map_def* g)

next

assume $\text{closedin } X U$

then show $\text{closedin } Y (f \text{ ` } U)$

using *hom* *homeomorphic_imp_closed_map* *closed_map_def* **by** *blast*

qed

qed

lemma *homeomorphic_map_openness_eq*:

homeomorphic_map X Y $f \implies \text{openin } X U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{openin } Y$
 $(f \text{ ` } U)$

by (*meson* *homeomorphic_map_openness* *openin_closedin_eq*)

lemma *homeomorphic_map_closedness_eq*:

homeomorphic_map X Y $f \implies \text{closedin } X U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{closedin}$
 $Y (f \text{ ` } U)$

by (*meson* *closedin_subset* *homeomorphic_map_closedness*)

lemma *all_openin_homeomorphic_image*:

assumes *homeomorphic_map* X Y f

shows $(\forall V. \text{openin } Y V \longrightarrow P V) \longleftrightarrow (\forall U. \text{openin } X U \longrightarrow P(f \text{ ` } U))$

by (*metis* (*no_types*, *lifting*) *assms* *homeomorphic_eq_everything_map* *homeomorphic_map_openness* *openin_subset* *subset_image_iff*)

lemma *all_closedin_homeomorphic_image*:

assumes *homeomorphic_map* X Y f

shows $(\forall V. \text{closedin } Y V \longrightarrow P V) \longleftrightarrow (\forall U. \text{closedin } X U \longrightarrow P(f \text{ ` } U))$

by (*metis* (*no_types*, *lifting*) *assms* *closedin_subset* *homeomorphic_eq_everything_map*)

homeomorphic_map_closedness_subset_image_iff)

lemma *homeomorphic_map_derived_set_of*:

assumes *hom*: *homeomorphic_map* *X Y f* **and** *S*: $S \subseteq \text{topspace } X$

shows $Y \text{ derived_set_of } (f \text{ ' } S) = f \text{ ' } (X \text{ derived_set_of } S)$

proof –

have *fim*: $f \text{ ' } (\text{topspace } X) = \text{topspace } Y$ **and** *inj*: *inj_on* *f* (*topspace* *X*)

using *hom* **by** (*auto simp: homeomorphic_eq_everything_map*)

have *iff*: $(\forall T. x \in T \wedge \text{openin } X T \longrightarrow (\exists y. y \neq x \wedge y \in S \wedge y \in T)) =$

$(\forall T. T \subseteq \text{topspace } Y \longrightarrow f x \in T \longrightarrow \text{openin } Y T \longrightarrow (\exists y. y \neq f x \wedge y \in f \text{ ' } S \wedge y \in T))$

if $x \in \text{topspace } X$ **for** *x*

proof –

have \S : $(x \in T \wedge \text{openin } X T) = (T \subseteq \text{topspace } X \wedge f x \in f \text{ ' } T \wedge \text{openin } Y (f \text{ ' } T))$ **for** *T*

by (*meson hom homeomorphic_map_openness_eq inj inj_on_image_mem_iff that*)

moreover **have** $(\exists y. y \neq x \wedge y \in S \wedge y \in T) = (\exists y. y \neq f x \wedge y \in f \text{ ' } S \wedge y \in f \text{ ' } T)$ (*is ?lhs = ?rhs*)

if $T \subseteq \text{topspace } X \wedge f x \in f \text{ ' } T \wedge \text{openin } Y (f \text{ ' } T)$ **for** *T*

by (*smt (verit, del_insts) S \langle x \in \text{topspace } X \rangle image_iff inj inj_on_def subsetD that*)

ultimately **show** *?thesis*

by (*auto simp flip: fim simp: all_subset_image*)

qed

have $*$: $\llbracket T = f \text{ ' } S; \bigwedge x. x \in S \implies P x \longleftrightarrow Q(f x) \rrbracket \implies \{y. y \in T \wedge Q y\} = f \text{ ' } \{x \in S. P x\}$ **for** *T S P Q*

by *auto*

show *?thesis*

unfolding *derived_set_of_def*

by (*rule **) (*use fim iff openin_subset in force*)+

qed

lemma *homeomorphic_map_closure_of*:

assumes *hom*: *homeomorphic_map* *X Y f* **and** *S*: $S \subseteq \text{topspace } X$

shows $Y \text{ closure_of } (f \text{ ' } S) = f \text{ ' } (X \text{ closure_of } S)$

unfolding *closure_of*

using *homeomorphic_imp_surjective_map* [*OF hom*] *S*

by (*auto simp: in_derived_set_of homeomorphic_map_derived_set_of* [*OF assms*])

lemma *homeomorphic_map_interior_of*:

assumes *hom*: *homeomorphic_map* *X Y f* **and** *S*: $S \subseteq \text{topspace } X$

shows $Y \text{ interior_of } (f \text{ ' } S) = f \text{ ' } (X \text{ interior_of } S)$

proof –

{ **fix** *y*

assume $y \in \text{topspace } Y$ **and** $y \notin Y \text{ closure_of } (\text{topspace } Y - f \text{ ' } S)$

then **have** $y \in f \text{ ' } (\text{topspace } X - X \text{ closure_of } (\text{topspace } X - S))$

```

    using homeomorphic_eq_everything_map [THEN iffD1, OF hom] homeo-
    morphic_map_closure_of [OF hom]
  by (metis DiffI Diff_subset S closure_of_subset_topspace inj_on_image_set_diff)
}
moreover
{ fix x
  assume  $x \in \text{topspace } X$ 
  then have  $f x \in \text{topspace } Y$ 
    using hom homeomorphic_imp_surjective_map by blast }
moreover
{ fix x
  assume  $x \in \text{topspace } X$  and  $x \notin X \text{ closure\_of } (\text{topspace } X - S)$  and  $f x \in Y$ 
  closure_of (topspace  $Y - f^{-1} S$ )
  then have False
    using homeomorphic_map_closure_of [OF hom] hom
    unfolding homeomorphic_eq_everything_map
    by (metis Diff_subset S closure_of_subset_topspace inj_on_image_mem_iff
    inj_on_image_set_diff)
}
ultimately show ?thesis
  by (auto simp: interior_of_closure_of)
qed

```

lemma *homeomorphic_map_frontier_of*:

```

  assumes hom: homeomorphic_map  $X Y f$  and  $S: S \subseteq \text{topspace } X$ 
  shows  $Y \text{ frontier\_of } (f^{-1} S) = f^{-1} (X \text{ frontier\_of } S)$ 
  unfolding frontier_of_def
proof (intro equalityI subsetI DiffI)
  fix y
  assume  $y \in Y \text{ closure\_of } f^{-1} S - Y \text{ interior\_of } f^{-1} S$ 
  then show  $y \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
    using S hom homeomorphic_map_closure_of homeomorphic_map_interior_of
  by fastforce
next
  fix y
  assume  $y \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
  then show  $y \in Y \text{ closure\_of } f^{-1} S$ 
    using S hom homeomorphic_map_closure_of by fastforce
next
  fix x
  assume  $x \in f^{-1} (X \text{ closure\_of } S - X \text{ interior\_of } S)$ 
  then obtain y where  $y: x = f y$   $y \in X \text{ closure\_of } S$   $y \notin X \text{ interior\_of } S$ 
    by blast
  then show  $x \notin Y \text{ interior\_of } f^{-1} S$ 
    using S hom homeomorphic_map_interior_of y(1)
    unfolding homeomorphic_map_def
    by (smt (verit, ccfv_SIG) in_closure_of inj_on_image_mem_iff interior_of_subset_topspace)
qed

```

lemma *homeomorphic_maps_subtopologies*:

$\llbracket \text{homeomorphic_maps } X Y f g; f' (\text{topspace } X \cap S) = \text{topspace } Y \cap T \rrbracket$
 $\implies \text{homeomorphic_maps } (\text{subtopology } X S) (\text{subtopology } Y T) f g$

unfolding *homeomorphic_maps_def*

by (*force simp: continuous_map_from_subtopology continuous_map_in_subtopology*)

lemma *homeomorphic_maps_subtopologies_alt*:

$\llbracket \text{homeomorphic_maps } X Y f g; f' (\text{topspace } X \cap S) \subseteq T; g' (\text{topspace } Y \cap T) \subseteq S \rrbracket$

$\implies \text{homeomorphic_maps } (\text{subtopology } X S) (\text{subtopology } Y T) f g$

unfolding *homeomorphic_maps_def*

by (*force simp: continuous_map_from_subtopology continuous_map_in_subtopology*)

lemma *homeomorphic_map_subtopologies*:

$\llbracket \text{homeomorphic_map } X Y f; f' (\text{topspace } X \cap S) = \text{topspace } Y \cap T \rrbracket$

$\implies \text{homeomorphic_map } (\text{subtopology } X S) (\text{subtopology } Y T) f$

by (*meson homeomorphic_map_maps homeomorphic_maps_subtopologies*)

lemma *homeomorphic_map_subtopologies_alt*:

assumes *hom: homeomorphic_map X Y f*

and $S: \bigwedge x. \llbracket x \in \text{topspace } X; f x \in \text{topspace } Y \rrbracket \implies f x \in T \longleftrightarrow x \in S$

shows *homeomorphic_map (subtopology X S) (subtopology Y T) f*

proof –

have *homeomorphic_maps (subtopology X S) (subtopology Y T) f g*

if *homeomorphic_maps X Y f g* **for** *g*

proof (*rule homeomorphic_maps_subtopologies [OF that]*)

have $f' (\text{topspace } X \cap S) \subseteq \text{topspace } Y \cap T$

using *S hom homeomorphic_imp_surjective_map* **by** *fastforce*

then show $f' (\text{topspace } X \cap S) = \text{topspace } Y \cap T$

using that unfolding homeomorphic_maps_def continuous_map_def Pi_iff

by (*smt (verit, del_insts) Int_iff S image_iff subsetI subset_antisym*)

qed

then show *?thesis*

using hom by (*meson homeomorphic_map_maps*)

qed

1.13.15 Relation of homeomorphism between topological spaces

definition *homeomorphic_space* (**infixr** *homeomorphic'_space* 50)

where $X \text{homeomorphic_space } Y \equiv \exists f g. \text{homeomorphic_maps } X Y f g$

lemma *homeomorphic_space_refl [iff]: X homeomorphic_space X*

by (*meson homeomorphic_maps_id homeomorphic_space_def*)

lemma *homeomorphic_space_sym*:

$X \text{homeomorphic_space } Y \longleftrightarrow Y \text{homeomorphic_space } X$

unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_sym*)

lemma *homeomorphic_space_trans* [trans]:

$\llbracket X1 \text{ homeomorphic_space } X2; X2 \text{ homeomorphic_space } X3 \rrbracket \implies X1 \text{ homeomorphic_space } X3$

unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_compose*)

lemma *homeomorphic_space*:

$X \text{ homeomorphic_space } Y \iff (\exists f. \text{homeomorphic_map } X Y f)$

by (*simp add: homeomorphic_map_maps homeomorphic_space_def*)

lemma *homeomorphic_maps_imp_homeomorphic_space*:

$\text{homeomorphic_maps } X Y f g \implies X \text{ homeomorphic_space } Y$

unfolding *homeomorphic_space_def* **by** *metis*

lemma *homeomorphic_map_imp_homeomorphic_space*:

$\text{homeomorphic_map } X Y f \implies X \text{ homeomorphic_space } Y$

unfolding *homeomorphic_map_maps*

using *homeomorphic_space_def* **by** *blast*

lemma *homeomorphic_empty_space*:

$X \text{ homeomorphic_space } Y \implies X = \text{trivial_topology} \iff Y = \text{trivial_topology}$

by (*meson continuous_map_on_empty2 homeomorphic_maps_def homeomorphic_space_def*)

lemma *homeomorphic_empty_space_eq*:

assumes $X = \text{trivial_topology}$

shows $X \text{ homeomorphic_space } Y \iff Y = \text{trivial_topology}$

using *assms funcset_mem*

by (*fastforce simp: homeomorphic_maps_def homeomorphic_space_def continuous_map_def*)

lemma *homeomorphic_space_unfold*:

assumes $X \text{ homeomorphic_space } Y$

obtains $f g$ **where** $\text{homeomorphic_map } X Y f \text{ homeomorphic_map } Y X g$

and $\bigwedge x. x \in \text{topspace } X \implies g(f x) = x \bigwedge y. y \in \text{topspace } Y \implies f(g y) = y$

and $f \in \text{topspace } X \rightarrow \text{topspace } Y \ g \in \text{topspace } Y \rightarrow \text{topspace } X$

using *assms unfolding homeomorphic_space_def homeomorphic_maps_map*

by (*smt (verit, best) Pi_I homeomorphic_imp_surjective_map image_eqI*)

1.13.16 Connected topological spaces

definition *connected_space* :: '*a* topology \Rightarrow bool **where**

connected_space $X \equiv$

$\neg(\exists E1 E2. \text{openin } X E1 \wedge \text{openin } X E2 \wedge \text{topspace } X \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$

definition *connectedin* :: '*a* topology \Rightarrow '*a* set \Rightarrow bool **where**

connectedin $X S \equiv S \subseteq \text{topspace } X \wedge \text{connected_space } (\text{subtopology } X S)$

lemma *connected_spaceD*:

$\llbracket \text{connected_space } X;$
 $\text{openin } X U; \text{openin } X V; \text{topspace } X \subseteq U \cup V; U \cap V = \{\}; U \neq \{\}; V \neq \{\} \rrbracket \implies \text{False}$
by (auto simp: connected_space_def)

lemma connectedin_subset_topspace: $\text{connectedin } X S \implies S \subseteq \text{topspace } X$
by (simp add: connectedin_def)

lemma connectedin_topspace:
 $\text{connectedin } X (\text{topspace } X) \longleftrightarrow \text{connected_space } X$
by (simp add: connectedin_def)

lemma connected_space_subtopology:
 $\text{connectedin } X S \implies \text{connected_space } (\text{subtopology } X S)$
by (simp add: connectedin_def)

lemma connectedin_subtopology:
 $\text{connectedin } (\text{subtopology } X S) T \longleftrightarrow \text{connectedin } X T \wedge T \subseteq S$
by (force simp: connectedin_def subtopology_subtopology_inf_absorb2)

lemma connected_space_eq:
 $\text{connected_space } X \longleftrightarrow$
 $(\nexists E1 E2. \text{openin } X E1 \wedge \text{openin } X E2 \wedge E1 \cup E2 = \text{topspace } X \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$
unfolding connected_space_def
by (metis openin_Un openin_subset subset_antisym)

lemma connected_space_closedin:
 $\text{connected_space } X \longleftrightarrow$
 $(\nexists E1 E2. \text{closedin } X E1 \wedge \text{closedin } X E2 \wedge \text{topspace } X \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$ (is ?lhs = ?rhs)

proof

assume ?lhs
then have $\bigwedge E1 E2. \llbracket \text{openin } X E1; E1 \cap E2 = \{\}; \text{topspace } X \subseteq E1 \cup E2; \text{openin } X E2 \rrbracket \implies E1 = \{\} \vee E2 = \{\}$
by (simp add: connected_space_def)
then show ?rhs
unfolding connected_space_def
by (metis disjoint_def separatedin_closed_sets separation_openin_Un_gen subtopology_superset)
next
assume R: ?rhs
then show ?lhs
unfolding connected_space_def
by (metis Diff_triv Int_commute separatedin_openin_diff separation_closedin_Un_gen subtopology_superset)
qed

lemma connected_space_closedin_eq:

$connected_space\ X \longleftrightarrow$
 $(\nexists E1\ E2. closedin\ X\ E1 \wedge closedin\ X\ E2 \wedge$
 $E1 \cup E2 = topspace\ X \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$
by (metis closedin_Un closedin_def connected_space_closedin_subset_antisym)

lemma connected_space_clopen_in:

$connected_space\ X \longleftrightarrow$
 $(\forall T. openin\ X\ T \wedge closedin\ X\ T \longrightarrow T = \{\} \vee T = topspace\ X)$

proof –

have eq: $openin\ X\ E1 \wedge openin\ X\ E2 \wedge E1 \cup E2 = topspace\ X \wedge E1 \cap E2 = \{\} \wedge P$

$\longleftrightarrow E2 = topspace\ X - E1 \wedge openin\ X\ E1 \wedge openin\ X\ E2 \wedge P$ **for** $E1\ E2$
 P

using openin_subset **by** blast

show ?thesis

unfolding connected_space_eq eq_closedin_def

by (auto simp: openin_closedin_eq)

qed

lemma connectedin:

$connectedin\ X\ S \longleftrightarrow$
 $S \subseteq topspace\ X \wedge$
 $(\nexists E1\ E2.$
 $openin\ X\ E1 \wedge openin\ X\ E2 \wedge$
 $S \subseteq E1 \cup E2 \wedge E1 \cap E2 \cap S = \{\} \wedge E1 \cap S \neq \{\} \wedge E2 \cap S \neq \{\})$

(is ?lhs = ?rhs)

proof –

have *: $(\exists E1:: 'a\ set. \exists E2:: 'a\ set. (\exists T1:: 'a\ set. P1\ T1 \wedge E1 = f1\ T1) \wedge$
 $(\exists T2:: 'a\ set. P2\ T2 \wedge E2 = f2\ T2) \wedge$

$R\ E1\ E2) \longleftrightarrow (\exists T1\ T2. P1\ T1 \wedge P2\ T2 \wedge R(f1\ T1)\ (f2\ T2))$ **for** $P1$
 $f1\ P2\ f2\ R$

by auto

show ?thesis

unfolding connectedin_def connected_space_def openin_subtopology topspace_subtopology

*

by (intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset)

qed

lemma connectedinD:

$\llbracket connectedin\ X\ S; openin\ X\ E1; openin\ X\ E2; S \subseteq E1 \cup E2; E1 \cap E2 \cap S =$
 $\{\}; E1 \cap S \neq \{\}; E2 \cap S \neq \{\} \rrbracket \Longrightarrow False$

by (meson connectedin)

lemma connectedin_iff_connected [simp]: $connectedin\ euclidean\ S \longleftrightarrow connected\ S$

by (simp add: connected_def connectedin)

lemma connectedin_closedin:

$connectedin\ X\ S \longleftrightarrow$

$$\begin{aligned}
& S \subseteq \text{topspace } X \wedge \\
& \neg(\exists E1 E2. \text{closedin } X E1 \wedge \text{closedin } X E2 \wedge \\
& \quad S \subseteq (E1 \cup E2) \wedge \\
& \quad (E1 \cap E2 \cap S = \{\}) \wedge \\
& \quad \neg(E1 \cap S = \{\}) \wedge \neg(E2 \cap S = \{\}))
\end{aligned}$$

proof –

have *: $(\exists E1:: 'a \text{ set. } \exists E2:: 'a \text{ set. } (\exists T1:: 'a \text{ set. } P1 T1 \wedge E1 = f1 T1) \wedge$
 $(\exists T2:: 'a \text{ set. } P2 T2 \wedge E2 = f2 T2) \wedge$

$R E1 E2) \longleftrightarrow (\exists T1 T2. P1 T1 \wedge P2 T2 \wedge R(f1 T1) (f2 T2))$ **for** $P1$
 $f1 P2 f2 R$

by *auto*

show *?thesis*

unfolding *connectedin_def connected_space_closedin closedin_subtopology topspace_subtopology*

*

by (*intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset*)

qed

lemma *connectedin_empty [simp]: connectedin X {}*

by (*simp add: connectedin*)

lemma *connected_space_trivial_topology [simp]: connected_space trivial_topology*

using *connectedin_topspace by fastforce*

lemma *connectedin_sing [simp]: connectedin X {a} $\longleftrightarrow a \in \text{topspace } X$*

by (*simp add: connectedin*)

lemma *connectedin_absolute [simp]:*

connectedin (subtopology X S) S \longleftrightarrow connectedin X S

by (*simp add: connectedin_subtopology*)

lemma *connectedin_Union:*

assumes $\mathcal{U}: \bigwedge S. S \in \mathcal{U} \implies \text{connectedin } X S$ **and** $ne: \bigcap \mathcal{U} \neq \{\}$

shows *connectedin X ($\bigcup \mathcal{U}$)*

proof –

have $\bigcup \mathcal{U} \subseteq \text{topspace } X$

using \mathcal{U} **by** (*simp add: Union_least connectedin_def*)

moreover **have** *False*

if *openin X E1 openin X E2* **and** *cover: $\bigcup \mathcal{U} \subseteq E1 \cup E2$* **and** *disj: $E1 \cap E2$*
 $\cap \bigcup \mathcal{U} = \{\}$

and *overlap1: $E1 \cap \bigcup \mathcal{U} \neq \{\}$* **and** *overlap2: $E2 \cap \bigcup \mathcal{U} \neq \{\}$*

for $E1 E2$

proof –

have *disjS: $E1 \cap E2 \cap S = \{\}$* **if** $S \in \mathcal{U}$ **for** S

using *Diff_triv that disj by auto*

have *coverS: $S \subseteq E1 \cup E2$* **if** $S \in \mathcal{U}$ **for** S

using *that cover by blast*

have $\mathcal{U} \neq \{\}$

using *overlap1 by blast*

obtain a **where** $a: \bigwedge U. U \in \mathcal{U} \implies a \in U$

```

    using ne by force
  with  $\langle \mathcal{U} \neq \{\} \rangle$  have  $a \in \bigcup \mathcal{U}$ 
    by blast
  then consider  $a \in E1 \mid a \in E2$ 
    using  $\langle \bigcup \mathcal{U} \subseteq E1 \cup E2 \rangle$  by auto
  then show False
  proof cases
    case 1
      then obtain  $b \ S$  where  $b \in E2 \ b \in S \ S \in \mathcal{U}$ 
        using overlap2 by blast
      then show ?thesis
        using 1  $\langle \text{openin } X \ E1 \rangle \langle \text{openin } X \ E2 \rangle \text{ disj } S \ \text{cover } S \ a \ [OF \ \langle S \in \mathcal{U} \rangle] \ \mathcal{U} [OF \ \langle S \in \mathcal{U} \rangle]$ 
          unfolding connectedin
          by (meson disjoint_iff_not_equal)
    next
      case 2
        then obtain  $b \ S$  where  $b \in E1 \ b \in S \ S \in \mathcal{U}$ 
          using overlap1 by blast
        then show ?thesis
          using 2  $\langle \text{openin } X \ E1 \rangle \langle \text{openin } X \ E2 \rangle \text{ disj } S \ \text{cover } S \ a \ [OF \ \langle S \in \mathcal{U} \rangle] \ \mathcal{U} [OF \ \langle S \in \mathcal{U} \rangle]$ 
            unfolding connectedin
            by (meson disjoint_iff_not_equal)
  qed
  qed
  ultimately show ?thesis
    unfolding connectedin by blast
  qed

```

lemma *connectedin_Un*:

```

   $[[\text{connectedin } X \ S; \text{connectedin } X \ T; S \cap T \neq \{\}]] \implies \text{connectedin } X \ (S \cup T)$ 
  using connectedin_Union [of  $\{S, T\}$ ] by auto

```

lemma *connected_space_subconnected*:

```

   $\text{connected\_space } X \longleftrightarrow (\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists S. \text{connectedin } X \ S \wedge x \in S \wedge y \in S)$  (is ?lhs = ?rhs)

```

proof

```

  assume ?lhs

```

```

  then show ?rhs

```

```

    using connectedin_topspace by blast

```

next

```

  assume R [rule_format]: ?rhs

```

```

  have False if  $\text{openin } X \ U \ \text{openin } X \ V$  and  $\text{disj: } U \cap V = \{\}$  and  $\text{cover: topspace } X \subseteq U \cup V$ 

```

```

    and  $U \neq \{\} \ V \neq \{\}$  for  $U \ V$ 

```

proof –

```

  obtain  $u \ v$  where  $u \in U \ v \in V$ 

```

```

    using  $\langle U \neq \{\} \rangle \langle V \neq \{\} \rangle$  by auto

```

```

then obtain  $T$  where  $u \in T$   $v \in T$  and  $T$ : connectedin  $X$   $T$ 
using  $R$  [of  $u$   $v$ ] that
by (meson  $\langle$ openin  $X$   $U$  $\rangle$   $\langle$ openin  $X$   $V$  $\rangle$  subsetD openin_subset)
then show False
using that unfolding connectedin
by (metis IntI  $\langle u \in U \rangle$   $\langle v \in V \rangle$  empty_iff inf_bot_left subset_trans)
qed
then show ?lhs
by (auto simp: connected_space_def)
qed

```

```

lemma connectedin_intermediate_closure_of:
assumes connectedin  $X$   $S$   $S \subseteq T$   $T \subseteq X$  closure_of  $S$ 
shows connectedin  $X$   $T$ 
proof -
have  $S$ :  $S \subseteq$  topspace  $X$  and  $T$ :  $T \subseteq$  topspace  $X$ 
using assms by (meson closure_of_subset_topspace dual_order.trans)+
have  $\S$ :  $\bigwedge E1$   $E2$ . [openin  $X$   $E1$ ; openin  $X$   $E2$ ;  $E1 \cap S = \{\}$   $\vee$   $E2 \cap S = \{\}$ ]
 $\implies E1 \cap T = \{\}$   $\vee$   $E2 \cap T = \{\}$ 
using assms unfolding disjoint_iff by (meson in_closure_of_subsetD)
then show ?thesis
using assms
unfolding connectedin_closure_of_subset_topspace  $S$   $T$ 
by (metis Int_empty_right  $T$  dual_order.trans inf.orderE inf_left_commute)
qed

```

```

lemma connectedin_closure_of:
 $\text{connectedin } X \ S \implies \text{connectedin } X \ (X \ \text{closure\_of } S)$ 
by (meson closure_of_subset connectedin_def connectedin_intermediate_closure_of
subset_refl)

```

```

lemma connectedin_separation:
 $\text{connectedin } X \ S \longleftrightarrow$ 
 $S \subseteq$  topspace  $X \wedge$ 
 $(\nexists C1$   $C2$ .  $C1 \cup C2 = S \wedge C1 \neq \{\}$   $\wedge$   $C2 \neq \{\}$   $\wedge$   $C1 \cap X$  closure_of  $C2$ 
 $= \{\}$   $\wedge$   $C2 \cap X$  closure_of  $C1 = \{\}$ )
unfolding connectedin_def connected_space_closedin_eq closedin_Int_closure_of
topspace_subtopology
apply (intro conj_cong refl arg_cong [where  $f = \text{Not}$ ])
apply (intro ex_cong1 iffI, blast)
using closure_of_subset_Int by force

```

```

lemma connectedin_eq_not_separated:
 $\text{connectedin } X \ S \longleftrightarrow$ 
 $S \subseteq$  topspace  $X \wedge$ 
 $(\nexists C1$   $C2$ .  $C1 \cup C2 = S \wedge C1 \neq \{\}$   $\wedge$   $C2 \neq \{\}$   $\wedge$  separatedin  $X$   $C1$   $C2$ )
unfolding separatedin_def by (metis connectedin_separation sup.boundedE)

```

```

lemma connectedin_eq_not_separated_subset:

```

$connectedin\ X\ S \longleftrightarrow$
 $S \subseteq topspace\ X \wedge (\nexists\ C1\ C2. S \subseteq C1 \cup C2 \wedge S \cap C1 \neq \{\} \wedge S \cap C2 \neq \{\})$
 $\wedge separatedin\ X\ C1\ C2)$

proof –

have $\forall\ C1\ C2. S \subseteq C1 \cup C2 \longrightarrow S \cap C1 = \{\} \vee S \cap C2 = \{\} \vee \neg separatedin\ X\ C1\ C2$

if $\bigwedge\ C1\ C2. C1 \cup C2 = S \longrightarrow C1 = \{\} \vee C2 = \{\} \vee \neg separatedin\ X\ C1\ C2$

proof (intro allI)

fix $C1\ C2$

show $S \subseteq C1 \cup C2 \longrightarrow S \cap C1 = \{\} \vee S \cap C2 = \{\} \vee \neg separatedin\ X\ C1\ C2$

using that [of $S \cap C1\ S \cap C2$]

by (auto simp: separatedin_mono)

qed

then show ?thesis

by (metis Un_Int_eq(1) Un_Int_eq(2) connectedin_eq_not_separated_order_refl)

qed

lemma *connected_space_eq_not_separated*:

$connected_space\ X \longleftrightarrow$

$(\nexists\ C1\ C2. C1 \cup C2 = topspace\ X \wedge C1 \neq \{\} \wedge C2 \neq \{\} \wedge separatedin\ X\ C1\ C2)$

by (simp add: connectedin_eq_not_separated_flip: connectedin_topspace)

lemma *connected_space_eq_not_separated_subset*:

$connected_space\ X \longleftrightarrow$

$(\nexists\ C1\ C2. topspace\ X \subseteq C1 \cup C2 \wedge C1 \neq \{\} \wedge C2 \neq \{\} \wedge separatedin\ X\ C1\ C2)$

by (metis connected_space_eq_not_separated le_sup_iff separatedin_def subset_antisym)

lemma *connectedin_subset_separated_union*:

$\llbracket connectedin\ X\ C; separatedin\ X\ S\ T; C \subseteq S \cup T \rrbracket \Longrightarrow C \subseteq S \vee C \subseteq T$

unfolding connectedin_eq_not_separated_subset **by** blast

lemma *connectedin_nonseparated_union*:

assumes $connectedin\ X\ S\ connectedin\ X\ T\ \neg separatedin\ X\ S\ T$

shows $connectedin\ X\ (S \cup T)$

proof –

have $\bigwedge\ C1\ C2. \llbracket T \subseteq C1 \cup C2; S \subseteq C1 \cup C2 \rrbracket \Longrightarrow$

$S \cap C1 = \{\} \wedge T \cap C1 = \{\} \vee S \cap C2 = \{\} \wedge T \cap C2 = \{\} \vee \neg separatedin\ X\ C1\ C2$

using *assms*

unfolding connectedin_eq_not_separated_subset

by (metis (no_types, lifting) *assms* connectedin_subset_separated_union inf.orderE separatedin_empty(1) separatedin_mono separatedin_sym)

then show ?thesis

unfolding connectedin_eq_not_separated_subset

by (simp add: assms connectedin_subset_topspace Int_Un_distrib2)
qed

lemma *connected_space_closures*:

connected_space $X \longleftrightarrow$
 $(\nexists e1\ e2. e1 \cup e2 = \text{topspace } X \wedge X \text{ closure_of } e1 \cap X \text{ closure_of } e2 = \{\})$
 $\wedge e1 \neq \{\} \wedge e2 \neq \{\}$)
(is ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

unfolding *connected_space_closedin_eq*

by (metis Un_upper1 Un_upper2 closedin_closure_of closure_of_Un closure_of_eq_empty closure_of_topspace)

next

assume ?rhs

then show ?lhs

unfolding *connected_space_closedin_eq*

by (metis closure_of_eq)

qed

lemma *connectedin_Int_frontier_of*:

assumes *connectedin* $X\ S\ S \cap T \neq \{\}\ S - T \neq \{\}$

shows $S \cap X \text{ frontier_of } T \neq \{\}$

proof -

have $S \subseteq \text{topspace } X$ and *:

$\bigwedge E1\ E2. \text{openin } X\ E1 \longrightarrow \text{openin } X\ E2 \longrightarrow E1 \cap E2 \cap S = \{\} \longrightarrow S \subseteq E1$
 $\cup E2 \longrightarrow E1 \cap S = \{\} \vee E2 \cap S = \{\}$

using $\langle \text{connectedin } X\ S \rangle$ by (auto simp: *connectedin*)

moreover

have $S - (\text{topspace } X \cap T) \neq \{\}$

using *assms*(3) by blast

moreover

have $S \cap \text{topspace } X \cap T \neq \{\}$

using *assms* *connectedin* by fastforce

moreover

have False if $S \cap T \neq \{\}\ S - T \neq \{\}\ T \subseteq \text{topspace } X\ S \cap X \text{ frontier_of } T = \{\}$ for T

proof -

have null: $S \cap (X \text{ closure_of } T - X \text{ interior_of } T) = \{\}$

using that **unfolding** *frontier_of_def* by blast

have $X \text{ interior_of } T \cap (\text{topspace } X - X \text{ closure_of } T) \cap S = \{\}$

by (metis *Diff_disjoint inf_bot_left interior_of_Int interior_of_complement interior_of_empty*)

moreover have $S \subseteq X \text{ interior_of } T \cup (\text{topspace } X - X \text{ closure_of } T)$

using that $\langle S \subseteq \text{topspace } X \rangle$ null by auto

moreover have $S \cap X \text{ interior_of } T \neq \{\}$

using *closure_of_subset* *that*(1) *that*(3) null by fastforce

ultimately have $S \cap X \text{ interior_of } (\text{topspace } X - T) = \{\}$


```

    by (metis * inf_commute interior_of_complement openin_interior_of)
  then have topspace (subtopology X S) ∩ X interior_of T = S
    using ⟨S ⊆ topspace X⟩ interior_of_complement null by fastforce
  then show ?thesis
    using that by (metis Diff_eq_empty_iff inf_le2 interior_of_subset subset_trans)
  qed
  ultimately show ?thesis
    by (metis Int_lower1 frontier_of_restrict inf_assoc)
  qed

```

lemma *connectedin_continuous_map_image:*

```

  assumes f: continuous_map X Y f and connectedin X S
  shows connectedin Y (f ` S)
  proof -
    have S ⊆ topspace X and *:
      ∧ E1 E2. openin X E1 → openin X E2 → E1 ∩ E2 ∩ S = {} → S ⊆ E1
    ∪ E2 → E1 ∩ S = {} ∨ E2 ∩ S = {}
    using ⟨connectedin X S⟩ by (auto simp: connectedin)
    show ?thesis
      unfolding connectedin connected_space_def
    proof (intro conjI notI; clarify)
      show f x ∈ topspace Y if x ∈ S for x
        using ⟨S ⊆ topspace X⟩ continuous_map_image_subset_topspace f that by
      blast
    next
      fix U V
      let ?U = {x ∈ topspace X. f x ∈ U}
      let ?V = {x ∈ topspace X. f x ∈ V}
      assume UV: openin Y U openin Y V f ` S ⊆ U ∪ V U ∩ V ∩ f ` S = {} U
      ∩ f ` S ≠ {} V ∩ f ` S ≠ {}
      then have 1: ?U ∩ ?V ∩ S = {}
        by auto
      have 2: openin X ?U openin X ?V
        using ⟨openin Y U⟩ ⟨openin Y V⟩ continuous_map f by fastforce+
      show False
        using * [of ?U ?V] UV ⟨S ⊆ topspace X⟩
        by (auto simp: 1 2)
    qed
  qed

```

lemma *connected_space_quotient_map_image:*

```

  [[quotient_map X X' q; connected_space X]] ⇒ connected_space X'
  by (metis connectedin_continuous_map_image connectedin_topspace quotient_imp_continuous_map
  quotient_imp_surjective_map)

```

lemma *homeomorphic_connected_space:*

```

  X homeomorphic_space Y ⇒ connected_space X ↔ connected_space Y
  unfolding homeomorphic_space_def homeomorphic_maps_def

```

by (metis connected_space_subconnected connectedin_continuous_map_image
connectedin_topspace continuous_map_image_subset_topspace image_eqI image_subset_iff)

lemma *homeomorphic_map_connectedness*:

assumes f : *homeomorphic_map* $X Y f$ and U : $U \subseteq \text{topspace } X$

shows $\text{connectedin } Y (f \text{ ` } U) \longleftrightarrow \text{connectedin } X U$

proof –

have 1: $f \text{ ` } U \subseteq \text{topspace } Y \longleftrightarrow U \subseteq \text{topspace } X$

using $U f$ *homeomorphic_imp_surjective_map* by blast

moreover have $\text{connected_space } (\text{subtopology } Y (f \text{ ` } U)) \longleftrightarrow \text{connected_space } (\text{subtopology } X U)$

proof (rule *homeomorphic_connected_space*)

have $f \text{ ` } U \subseteq \text{topspace } Y$

by (simp add: U 1)

then have $\text{topspace } Y \cap f \text{ ` } U = f \text{ ` } U$

by (simp add: subset_antisym)

then show $\text{subtopology } Y (f \text{ ` } U)$ *homeomorphic_space* $\text{subtopology } X U$

by (metis $U f$ *homeomorphic_map_imp_homeomorphic_space* *homeomorphic_map_subtopologies* *homeomorphic_space_sym* *inf.absorb_iff2*)

qed

ultimately show ?thesis

by (auto simp: *connectedin_def*)

qed

lemma *homeomorphic_map_connectedness_eq*:

homeomorphic_map $X Y f$

$\implies \text{connectedin } X U \longleftrightarrow$

$U \subseteq \text{topspace } X \wedge \text{connectedin } Y (f \text{ ` } U)$

using *homeomorphic_map_connectedness* *connectedin_subset_topspace* by metis

lemma *connectedin_discrete_topology*:

$\text{connectedin } (\text{discrete_topology } U) S \longleftrightarrow S \subseteq U \wedge (\exists a. S \subseteq \{a\})$

proof (cases $S \subseteq U$)

case True

show ?thesis

proof (cases $S = \{\}$)

case False

moreover have $\text{connectedin } (\text{discrete_topology } U) S \longleftrightarrow (\exists a. S = \{a\})$

proof

show $\text{connectedin } (\text{discrete_topology } U) S \implies \exists a. S = \{a\}$

using False *connectedin_Int_frontier_of_insert_Diff* by fastforce

qed (use True in auto)

ultimately show ?thesis

by auto

qed simp

next

case False

then show ?thesis

by (simp add: *connectedin_def*)

qed

lemma *connected_space_discrete_topology*:

$connected_space (discrete_topology U) \longleftrightarrow (\exists a. U \subseteq \{a\})$

by (*metis* *connectedin_discrete_topology connectedin_topspace order_refl topspace_discrete_topology*)

1.13.17 Compact sets

definition *compactin where*

$compactin X S \longleftrightarrow$

$S \subseteq topspace X \wedge$

$(\forall \mathcal{U}. (\forall U \in \mathcal{U}. openin X U) \wedge S \subseteq \bigcup \mathcal{U}$

$\longrightarrow (\exists \mathcal{F}. finite \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}))$

definition *compact_space where*

$compact_space X \equiv compactin X (topspace X)$

lemma *compact_space_alt*:

$compact_space X \longleftrightarrow$

$(\forall \mathcal{U}. (\forall U \in \mathcal{U}. openin X U) \wedge topspace X \subseteq \bigcup \mathcal{U}$

$\longrightarrow (\exists \mathcal{F}. finite \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge topspace X \subseteq \bigcup \mathcal{F}))$

by (*simp* *add: compact_space_def compactin_def*)

lemma *compact_space*:

$compact_space X \longleftrightarrow$

$(\forall \mathcal{U}. (\forall U \in \mathcal{U}. openin X U) \wedge \bigcup \mathcal{U} = topspace X$

$\longrightarrow (\exists \mathcal{F}. finite \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \bigcup \mathcal{F} = topspace X))$

unfolding *compact_space_alt*

using *openin_subset* **by** *fastforce*

lemma *compactinD*:

$\llbracket compactin X S; \bigwedge U. U \in \mathcal{U} \implies openin X U; S \subseteq \bigcup \mathcal{U} \rrbracket \implies \exists \mathcal{F}. finite \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$

by (*auto simp: compactin_def*)

lemma *compactin_euclidean_iff* [*simp*]: $compactin euclidean S \longleftrightarrow compact S$

by (*simp* *add: compact_eq_Heine_Borel compactin_def*) *meson*

lemma *compactin_absolute* [*simp*]:

$compactin (subtopology X S) S \longleftrightarrow compactin X S$

proof –

have *eq*: $(\forall U \in \mathcal{U}. \exists Y. openin X Y \wedge U = Y \cap S) \longleftrightarrow \mathcal{U} \subseteq (\lambda Y. Y \cap S) \cdot \{y. openin X y\}$ **for** \mathcal{U}

by *auto*

show *?thesis*

by (*auto simp: compactin_def openin_subtopology eq_imp_conjL all_subset_image ex_finite_subset_image*)

qed

lemma *compactin_subspace*: $\text{compactin } X \ S \longleftrightarrow S \subseteq \text{topspace } X \wedge \text{compact_space } (\text{subtopology } X \ S)$

unfolding *compact_space_def topspace_subtopology*
by (*metis compactin_absolute compactin_def inf.absorb2*)

lemma *compact_space_subtopology*: $\text{compactin } X \ S \implies \text{compact_space } (\text{subtopology } X \ S)$

by (*simp add: compactin_subspace*)

lemma *compactin_subtopology*: $\text{compactin } (\text{subtopology } X \ S) \ T \longleftrightarrow \text{compactin } X \ T \wedge T \subseteq S$

by (*metis compactin_subspace inf.absorb_iff2 le_inf_iff subtopology_subtopology topspace_subtopology*)

lemma *compactin_subset_topspace*: $\text{compactin } X \ S \implies S \subseteq \text{topspace } X$

by (*simp add: compactin_subspace*)

lemma *compactin_contractive*:

$\llbracket \text{compactin } X' \ S; \text{topspace } X' = \text{topspace } X; \bigwedge U. \text{openin } X \ U \implies \text{openin } X' \ U \rrbracket \implies \text{compactin } X \ S$
by (*simp add: compactin_def*)

lemma *finite_imp_compactin*:

$\llbracket S \subseteq \text{topspace } X; \text{finite } S \rrbracket \implies \text{compactin } X \ S$
by (*metis compactin_subspace compact_space finite_UnionD inf.absorb_iff2 order_refl topspace_subtopology*)

lemma *compactin_empty [iff]*: $\text{compactin } X \ \{\}$

by (*simp add: finite_imp_compactin*)

lemma *compact_space_trivial_topology [simp]*: $\text{compact_space } \text{trivial_topology}$

by (*simp add: compact_space_def*)

lemma *finite_imp_compactin_eq*:

$\text{finite } S \implies (\text{compactin } X \ S \longleftrightarrow S \subseteq \text{topspace } X)$
using *compactin_subset_topspace finite_imp_compactin* **by** *blast*

lemma *compactin_sing [simp]*: $\text{compactin } X \ \{a\} \longleftrightarrow a \in \text{topspace } X$

by (*simp add: finite_imp_compactin_eq*)

lemma *closed_compactin*:

assumes *XK*: $\text{compactin } X \ K$ **and** $C \subseteq K$ **and** *XC*: $\text{closedin } X \ C$
shows $\text{compactin } X \ C$

unfolding *compactin_def*

proof (*intro conjI allI impI*)

show $C \subseteq \text{topspace } X$

by (*simp add: XC closedin_subset*)

next

fix $U :: 'a \ \text{set}$

assume U : $\text{Ball } \mathcal{U} (\text{openin } X) \wedge C \subseteq \bigcup \mathcal{U}$
have $(\forall U \in \text{insert } (\text{topspace } X - C) \mathcal{U}. \text{openin } X U)$
using $XC \mathcal{U}$ **by** *blast*
moreover have $K \subseteq \bigcup (\text{insert } (\text{topspace } X - C) \mathcal{U})$
using $\mathcal{U} XK$ *compactin_subset_topspace* **by** *fastforce*
ultimately obtain \mathcal{F} **where** $\text{finite } \mathcal{F} \mathcal{F} \subseteq \text{insert } (\text{topspace } X - C) \mathcal{U} K \subseteq \bigcup \mathcal{F}$
using *assms unfolding compactin_def* **by** *metis*
moreover have $\text{openin } X (\text{topspace } X - C)$
using XC **by** *auto*
ultimately show $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge C \subseteq \bigcup \mathcal{F}$
using $\langle C \subseteq K \rangle$
by (*rule_tac* $x = \mathcal{F} - \{\text{topspace } X - C\}$ **in** *exI*) *auto*
qed

lemma *closed_compactin_Inter*:
 $\llbracket \text{compactin } X K; K \in \mathcal{K}; \bigwedge K. K \in \mathcal{K} \implies \text{closedin } X K \rrbracket \implies \text{compactin } X (\bigcap \mathcal{K})$
by (*metis* *Inf_lower closed_compactin closedin_Inter empty_iff*)

lemma *closedin_subtopology_Int_subset*:
 $\llbracket \text{closedin } (\text{subtopology } X U) (U \cap S); V \subseteq U \rrbracket \implies \text{closedin } (\text{subtopology } X V) (V \cap S)$
by (*smt* (*verit*, *ccfv_SIG*) *closedin_subtopology inf.left_commute inf.orderE inf_commute*)

lemma *closedin_subtopology_Int_closedin*:
 $\llbracket \text{closedin } (\text{subtopology } X U) S; \text{closedin } X T \rrbracket \implies \text{closedin } (\text{subtopology } X U) (S \cap T)$
by (*smt* (*verit*, *best*) *closedin_Int closedin_subtopology inf_assoc inf_commute*)

lemma *closedin_compact_space*:
 $\llbracket \text{compact_space } X; \text{closedin } X S \rrbracket \implies \text{compactin } X S$
by (*simp* *add: closed_compactin closedin_subset compact_space_def*)

lemma *compact_Int_closedin*:
assumes $\text{compactin } X S$ $\text{closedin } X T$ **shows** $\text{compactin } X (S \cap T)$
proof –
have $\text{compactin } (\text{subtopology } X S) (S \cap T)$
by (*metis* *assms closedin_compact_space closedin_subtopology compactin_subspace inf_commute*)
then show *?thesis*
by (*simp* *add: compactin_subtopology*)
qed

lemma *closed_Int_compactin*: $\llbracket \text{closedin } X S; \text{compactin } X T \rrbracket \implies \text{compactin } X (S \cap T)$
by (*metis* *compact_Int_closedin inf_commute*)

lemma *compactin_Un*:
assumes S : $\text{compactin } X S$ **and** T : $\text{compactin } X T$ **shows** $\text{compactin } X (S \cup T)$

T)

```

unfolding compactin_def
proof (intro conjI allI impI)
  show  $S \cup T \subseteq \text{topspace } X$ 
    using assms by (auto simp: compactin_def)
next
  fix  $\mathcal{U} :: 'a \text{ set set}$ 
  assume  $\mathcal{U}: \text{Ball } \mathcal{U} (\text{openin } X) \wedge S \cup T \subseteq \bigcup \mathcal{U}$ 
  with  $S$  obtain  $\mathcal{F}$  where  $\mathcal{V}: \text{finite } \mathcal{F} \ \mathcal{F} \subseteq \mathcal{U} \ S \subseteq \bigcup \mathcal{F}$ 
    unfolding compactin_def by (meson sup.bounded_iff)
  obtain  $\mathcal{W}$  where  $\text{finite } \mathcal{W} \ \mathcal{W} \subseteq \mathcal{U} \ T \subseteq \bigcup \mathcal{W}$ 
    using  $\mathcal{U} \ T$ 
  unfolding compactin_def by (meson sup.bounded_iff)
  with  $\mathcal{V}$  show  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge S \cup T \subseteq \bigcup \mathcal{V}$ 
    by (rule_tac  $x = \mathcal{F} \cup \mathcal{W}$  in exI) auto
qed

```

lemma compactin_Union:

```

 $\llbracket \text{finite } \mathcal{F}; \bigwedge S. S \in \mathcal{F} \implies \text{compactin } X \ S \rrbracket \implies \text{compactin } X (\bigcup \mathcal{F})$ 
by (induction rule: finite_induct) (simp_all add: compactin_Un)

```

lemma compactin_subtopology_imp_compact:

```

assumes compactin (subtopology  $X \ S$ )  $K$  shows compactin  $X \ K$ 
using assms
proof (clarsimp simp add: compactin_def)
  fix  $\mathcal{U}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. U \cap S) \text{ ` } \mathcal{U}$ 
  assume  $K \subseteq \text{topspace } X$  and  $K \subseteq S$  and  $\forall x \in \mathcal{U}. \text{openin } X \ x$  and  $K \subseteq \bigcup \mathcal{U}$ 
  then have  $\forall V \in \mathcal{V}. \text{openin } (\text{subtopology } X \ S) \ V \ K \subseteq \bigcup \mathcal{V}$ 
    unfolding  $\mathcal{V}$ _def by (auto simp: openin_subtopology)
  moreover
  assume  $\forall \mathcal{U}. (\forall x \in \mathcal{U}. \text{openin } (\text{subtopology } X \ S) \ x) \wedge K \subseteq \bigcup \mathcal{U} \longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F})$ 
  ultimately obtain  $\mathcal{F}$  where  $\text{finite } \mathcal{F} \ \mathcal{F} \subseteq \mathcal{V} \ K \subseteq \bigcup \mathcal{F}$ 
    by meson
  then have  $\mathcal{F}: \exists U. U \in \mathcal{U} \wedge V = U \cap S$  if  $V \in \mathcal{F}$  for  $V$ 
    unfolding  $\mathcal{V}$ _def using that by blast
  let  $?\mathcal{F} = (\lambda F. @U. U \in \mathcal{U} \wedge F = U \cap S) \text{ ` } \mathcal{F}$ 
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$ 
proof (intro exI conjI)
  show finite  $?\mathcal{F}$ 
    using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
  show  $?\mathcal{F} \subseteq \mathcal{U}$ 
    using someI_ex [OF  $\mathcal{F}$ ] by blast
  show  $K \subseteq \bigcup ?\mathcal{F}$ 
proof clarsimp
  fix  $x$ 
  assume  $x \in K$ 
  then show  $\exists V \in \mathcal{F}. x \in (\text{SOME } U. U \in \mathcal{U} \wedge V = U \cap S)$ 

```

```

    using ‹ $K \subseteq \bigcup \mathcal{F}$ › someI_ex [OF  $\mathcal{F}$ ]
  by (metis (no_types, lifting) IntD1 Union_iff subsetCE)
qed
qed
qed

lemma compact_imp_compactin_subtopology:
  assumes compactin  $X$   $K$   $K \subseteq S$  shows compactin (subtopology  $X$   $S$ )  $K$ 
  using assms
proof (clarsimp simp add: compactin_def)
  fix  $\mathcal{U} :: 'a$  set set
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv \{V. \text{openin } X \ V \wedge (\exists U \in \mathcal{U}. U = V \cap S)\}$ 
  assume  $K \subseteq S$  and  $K \subseteq \text{topspace } X$  and  $\forall U \in \mathcal{U}. \text{openin } (\text{subtopology } X \ S) \ U$ 
  and  $K \subseteq \bigcup \mathcal{U}$ 
  then have  $\forall V \in \mathcal{V}. \text{openin } X \ V \ K \subseteq \bigcup \mathcal{V}$ 
    unfolding  $\mathcal{V}$ _def by (fastforce simp: subset_eq openin_subtopology)+
  moreover
  assume  $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X \ U) \wedge K \subseteq \bigcup \mathcal{U} \longrightarrow (\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge$ 
 $K \subseteq \bigcup \mathcal{F})$ 
  ultimately obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$   $K \subseteq \bigcup \mathcal{F}$ 
  by meson
  let ? $\mathcal{F}$  =  $(\lambda F. F \cap S)$  ‘  $\mathcal{F}$ 
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$ 
  proof (intro exI conjI)
    show finite ? $\mathcal{F}$ 
      using ‹finite  $\mathcal{F}$ › by blast
    show ? $\mathcal{F} \subseteq \mathcal{U}$ 
      using  $\mathcal{V}$ _def ‹ $\mathcal{F} \subseteq \mathcal{V}$ › by blast
    show  $K \subseteq \bigcup ?\mathcal{F}$ 
      using ‹ $K \subseteq \bigcup \mathcal{F}$ › assms(2) by auto
  qed
qed

```

```

proposition compact_space_fip:
  compact_space  $X \longleftrightarrow$ 
  ( $\forall \mathcal{U}. (\forall C \in \mathcal{U}. \text{closedin } X \ C) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}) \longrightarrow$ 
 $\bigcap \mathcal{U} \neq \{\})$ 
  (is _ = ?rhs)
proof (cases  $X = \text{trivial\_topology}$ )
  case True
  then show ?thesis
    by (metis Pow_iff closedin_topospace_empty compact_space_trivial_topology
  finite.emptyI finite_Pow_iff finite_subset subsetI)
  next
  case False
  show ?thesis
  proof safe
    fix  $\mathcal{U} :: 'a$  set set

```

```

assume * [rule_format]:  $\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda S. \text{topspace } X - S) \text{ ' } \mathcal{U}$ 
assume clo:  $\forall C \in \mathcal{U}. \text{closedin } X C$  and [simp]:  $\bigcap \mathcal{U} = \{\}$ 
then have  $\forall V \in \mathcal{V}. \text{openin } X V$   $\text{topspace } X \subseteq \bigcup \mathcal{V}$ 
  by (auto simp:  $\mathcal{V}$ _def)
  moreover assume [unfolded compact_space_alt, rule_format, of  $\mathcal{V}$ ]: compact_space X
ultimately obtain  $\mathcal{F}$  where  $\mathcal{F}$ :  $\text{finite } \mathcal{F} \mathcal{F} \subseteq \mathcal{U} \text{topspace } X \subseteq \text{topspace } X - \bigcap \mathcal{F}$ 
  by (auto simp: ex_finite_subset_image  $\mathcal{V}$ _def)
moreover have  $\mathcal{F} \neq \{\}$ 
  using  $\mathcal{F}$  False by force
ultimately show False
  using * [of  $\mathcal{F}$ ]
  by auto (metis Diff_iff Inter_iff clo closedin_def subsetD)
next
assume R [rule_format]: ?rhs
show compact_space X
  unfolding compact_space_alt
proof clarify
  fix  $\mathcal{U} :: \text{'a set set}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda S. \text{topspace } X - S) \text{ ' } \mathcal{U}$ 
  assume  $\forall C \in \mathcal{U}. \text{openin } X C$  and  $\text{topspace } X \subseteq \bigcup \mathcal{U}$ 
  with False have *:  $\forall V \in \mathcal{V}. \text{closedin } X V$   $\mathcal{U} \neq \{\}$ 
  by (auto simp:  $\mathcal{V}$ _def)
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{F}$ 
  proof (rule ccontr; simp)
    assume  $\forall \mathcal{F} \subseteq \mathcal{U}. \text{finite } \mathcal{F} \longrightarrow \neg \text{topspace } X \subseteq \bigcup \mathcal{F}$ 
    then have  $\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{V} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
    by (simp add:  $\mathcal{V}$ _def all_finite_subset_image)
    with  $\langle \text{topspace } X \subseteq \bigcup \mathcal{U} \rangle$  show False
    using R [of  $\mathcal{V}$ ] * by (simp add:  $\mathcal{V}$ _def)
  qed
qed
qed
qed

```

corollary compactin_fip:

$$\text{compactin } X S \longleftrightarrow S \subseteq \text{topspace } X \wedge (\forall \mathcal{U}. (\forall C \in \mathcal{U}. \text{closedin } X C) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow S \cap \bigcap \mathcal{F} \neq \{\})) \longrightarrow S \cap \bigcap \mathcal{U} \neq \{\}$$

```

proof (cases  $S = \{\}$ )
case False
show ?thesis
proof (cases  $S \subseteq \text{topspace } X$ )
case True
then have compactin X S  $\longleftrightarrow$ 
   $(\forall \mathcal{U}. \mathcal{U} \subseteq (\lambda T. S \cap T) \text{ ' } \{T. \text{closedin } X T\} \longrightarrow$ 

```



```

      (∀F. finite F → F ⊆ U → ⋂ F ≠ {}) → ⋂ U ≠ {}
    by (simp add: compact_space_fip compactin_subspace closedin_subtopology
image_def subset_eq Int_commute imp_conjL)
    also have ... = (∀U ⊆ Collect (closedin X). (∀F. finite F → F ⊆ (⋂) S ' U
→ ⋂ F ≠ {}) → ⋂ ((⋂) S ' U) ≠ {})
    by (simp add: all_subset_image)
    also have ... = (∀U. (∀ C ∈ U. closedin X C) ∧ (∀F. finite F ∧ F ⊆ U →
S ∩ ⋂ F ≠ {}) → S ∩ ⋂ U ≠ {})
    proof -
      have eq: ((∀F. finite F ∧ F ⊆ U → ⋂ ((⋂) S ' F) ≠ {}) → ⋂ ((⋂) S '
U) ≠ {}) ↔
        ((∀F. finite F ∧ F ⊆ U → S ∩ ⋂ F ≠ {}) → S ∩ ⋂ U ≠ {}) for
U
      by simp (use ⟨S ≠ {}⟩ in blast)
    show ?thesis
      unfolding imp_conjL [symmetric] all_finite_subset_image eq by blast
    qed
  finally show ?thesis
    using True by simp
  qed (simp add: compactin_subspace)
qed force

```

corollary compact_space_imp_nest:

```

  fixes C :: nat ⇒ 'a set
  assumes compact_space X and clo: ⋀ n. closedin X (C n)
    and ne: ⋀ n. C n ≠ {} and dec: decseq C
  shows (⋂ n. C n) ≠ {}
  proof -
    let ?U = range (λ n. ⋂ m ≤ n. C m)
    have closedin X A if A ∈ ?U for A
      using that clo by auto
    moreover have (⋂ n ∈ K. ⋂ m ≤ n. C m) ≠ {} if finite K for K
    proof -
      obtain n where ⋀ k. k ∈ K ⇒ k ≤ n
      using Max.coboundedI ⟨finite K⟩ by blast
      with dec have C n ⊆ (⋂ n ∈ K. ⋂ m ≤ n. C m)
      unfolding decseq_def by blast
    with ne [of n] show ?thesis
      by blast
    qed
  ultimately show ?thesis
    using ⟨compact_space X⟩ [unfolded compact_space_fip, rule_format, of ?U]
    by (simp add: all_finite_subset_image INT_extend_simps UN_atMost_UNIV
del: INT_simps)
  qed

```

lemma compactin_discrete_topology:

```

  compactin (discrete_topology X) S ↔ S ⊆ X ∧ finite S (is ?lhs = ?rhs)
  proof (intro iffI conjI)

```

```

assume  $L$ : ?lhs
then show  $S \subseteq X$ 
  by (auto simp: compactin_def)
have *:  $\bigwedge \mathcal{U}. \text{Ball } \mathcal{U} \text{ (openin (discrete\_topology } X)) \wedge S \subseteq \bigcup \mathcal{U} \implies$ 
  ( $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$ )
  using  $L$  by (auto simp: compactin_def)
show finite  $S$ 
  using * [of ( $\lambda x. \{x\}$ ) '  $X$ ] '  $S \subseteq X$ 
  by clarsimp (metis UN_singleton finite_subset_image infinite_super)
next
assume ?rhs
then show ?lhs
  by (simp add: finite_imp_compactin)
qed

```

lemma compact_space_discrete_topology: compact_space(discrete_topology X) \longleftrightarrow finite X
by (simp add: compactin_discrete_topology compact_space_def)

lemma compact_space_imp_Bolzano_Weierstrass:
assumes compact_space X infinite S $S \subseteq \text{topspace } X$
shows X derived_set_of $S \neq \{\}$

```

proof
assume  $X$ :  $X$  derived_set_of  $S = \{\}$ 
then have closedin  $X$   $S$ 
  by (simp add: closedin_contains_derived_set assms)
then have compactin  $X$   $S$ 
  by (rule closedin_compact_space [OF '  $X$ ])
with  $X$  show False
  by (metis 'infinite  $S$ ') compactin_subspace compact_space_discrete_topology
  inf_bot_right subtopology_eq_discrete_topology_eq
qed

```

lemma compactin_imp_Bolzano_Weierstrass:
 $\llbracket \text{compactin } X \text{ } S; \text{infinite } T \wedge T \subseteq S \rrbracket \implies S \cap X \text{ derived_set_of } T \neq \{\}$
using compact_space_imp_Bolzano_Weierstrass [of subtopology X S]
by (simp add: compactin_subspace derived_set_of_subtopology inf_absorb2)

lemma compact_closure_of_imp_Bolzano_Weierstrass:
 $\llbracket \text{compactin } X \text{ (} X \text{ closure_of } S); \text{infinite } T; T \subseteq S; T \subseteq \text{topspace } X \rrbracket \implies X$
 $\text{derived_set_of } T \neq \{\}$
using closure_of_mono closure_of_subset compactin_imp_Bolzano_Weierstrass
by fastforce

lemma discrete_compactin_eq_finite:
 $S \cap X \text{ derived_set_of } S = \{\} \implies \text{compactin } X \text{ } S \longleftrightarrow S \subseteq \text{topspace } X \wedge \text{finite } S$
by (meson compactin_imp_Bolzano_Weierstrass finite_imp_compactin_eq_order_refl)

```

lemma discrete_compact_space_eq_finite:
   $X \text{ derived\_set\_of } (\text{topspace } X) = \{\} \implies (\text{compact\_space } X \longleftrightarrow \text{finite}(\text{topspace } X))$ 
  by (metis compact_space_discrete_topology discrete_topology_unique_derived_set)

lemma image_compactin:
  assumes cpt: compactin  $X$   $S$  and cont: continuous_map  $X$   $Y$   $f$ 
  shows compactin  $Y$   $(f \text{ ` } S)$ 
  unfolding compactin_def
proof (intro conjI allI impI)
  show  $f \text{ ` } S \subseteq \text{topspace } Y$ 
    using compactin_subset_topspace cont continuous_map_image_subset_topspace
  cpt by blast
next
  fix  $\mathcal{U} :: 'b \text{ set set}$ 
  assume  $\mathcal{U}$ :  $\text{Ball } \mathcal{U} (\text{openin } Y) \wedge f \text{ ` } S \subseteq \bigcup \mathcal{U}$ 
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. \{x \in \text{topspace } X. f \ x \in U\}) \text{ ` } \mathcal{U}$ 
  have  $S \subseteq \text{topspace } X$ 
    and  $*$ :  $\bigwedge \mathcal{U}. [\forall U \in \mathcal{U}. \text{openin } X \ U; S \subseteq \bigcup \mathcal{U}] \implies \exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$ 
    using cpt by (auto simp: compactin_def)
  obtain  $\mathcal{F}$  where  $\mathcal{F}$ : finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$   $S \subseteq \bigcup \mathcal{F}$ 
  proof –
    have  $1$ :  $\forall U \in \mathcal{V}. \text{openin } X \ U$ 
      unfolding  $\mathcal{V}$ _def using  $\mathcal{U}$  cont[unfolded continuous_map] by blast
    have  $2$ :  $S \subseteq \bigcup \mathcal{V}$ 
      unfolding  $\mathcal{V}$ _def using compactin_subset_topspace cpt  $\mathcal{U}$  by fastforce
    show thesis
      using  $*$  [OF  $1$   $2$ ] that by metis
  qed
  have  $\forall v \in \mathcal{V}. \exists U. U \in \mathcal{U} \wedge v = \{x \in \text{topspace } X. f \ x \in U\}$ 
    using  $\mathcal{V}$ _def by blast
  then obtain  $U$  where  $U$ :  $\forall v \in \mathcal{V}. U \ v \in \mathcal{U} \wedge v = \{x \in \text{topspace } X. f \ x \in U \ v\}$ 
    by metis
  show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge f \text{ ` } S \subseteq \bigcup \mathcal{F}$ 
proof (intro conjI exI)
  show finite  $(U \text{ ` } \mathcal{F})$ 
    by (simp add: <finite  $\mathcal{F}$ >)
next
  show  $U \text{ ` } \mathcal{F} \subseteq \mathcal{U}$ 
    using  $\langle \mathcal{F} \subseteq \mathcal{V} \rangle U$  by auto
next
  show  $f \text{ ` } S \subseteq \bigcup (U \text{ ` } \mathcal{F})$ 
    using  $\mathcal{F}(2-3)$   $U$  UnionE subset_eq  $U$  by fastforce
qed
qed

```

lemma *homeomorphic_compact_space*:
assumes X *homeomorphic_space* Y
shows $\text{compact_space } X \longleftrightarrow \text{compact_space } Y$
using *homeomorphic_space_sym*
by (*metis* *assms* *compact_space_def* *homeomorphic_eq_everything_map* *homeomorphic_space_image_compactin*)

lemma *homeomorphic_map_compactness*:
assumes hom : *homeomorphic_map* X Y f **and** U : $U \subseteq \text{topspace } X$
shows $\text{compactin } Y (f \text{ ` } U) \longleftrightarrow \text{compactin } X U$
proof –
have $f \text{ ` } U \subseteq \text{topspace } Y$
using hom U *homeomorphic_imp_surjective_map* **by** *blast*
moreover **have** *homeomorphic_map* (*subtopology* X U) (*subtopology* Y ($f \text{ ` } U$))
 f
using U hom *homeomorphic_imp_surjective_map* **by** (*blast* *intro*: *homeomorphic_map_subtopologies*)
then **have** $\text{compact_space } (\text{subtopology } Y (f \text{ ` } U)) = \text{compact_space } (\text{subtopology } X U)$
using *homeomorphic_compact_space* *homeomorphic_map_imp_homeomorphic_space*
by *blast*
ultimately **show** *?thesis*
by (*simp* *add*: *compactin_subspace* U)
qed

lemma *homeomorphic_map_compactness_eq*:
homeomorphic_map X Y f
 $\implies \text{compactin } X U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{compactin } Y (f \text{ ` } U)$
by (*meson* *compactin_subset_topspace* *homeomorphic_map_compactness*)

1.13.18 Embedding maps

definition *embedding_map*
where *embedding_map* X Y $f \equiv \text{homeomorphic_map } X (\text{subtopology } Y (f \text{ ` } (\text{topspace } X))) f$

lemma *embedding_map_eq*:
 $\llbracket \text{embedding_map } X Y f; \bigwedge x. x \in \text{topspace } X \implies f x = g x \rrbracket \implies \text{embedding_map } X Y g$
unfolding *embedding_map_def*
by (*metis* *homeomorphic_map_eq_image_cong*)

lemma *embedding_map_compose*:
assumes *embedding_map* X X' f *embedding_map* X' X'' g
shows *embedding_map* X X'' ($g \circ f$)
proof –
have hm : *homeomorphic_map* X (*subtopology* X' ($f \text{ ` } \text{topspace } X$)) f *homeomorphic_map* X' (*subtopology* X'' ($g \text{ ` } \text{topspace } X'$)) g
using *assms* **by** (*auto* *simp*: *embedding_map_def*)

```

then obtain  $C$  where  $g \text{ ' } \textit{topspace } X' \cap C = (g \circ f) \text{ ' } \textit{topspace } X$ 
using continuous_map_image_subset_topspace homeomorphic_imp_continuous_map
by fastforce
then have homeomorphic_map (subtopology  $X'$  ( $f \text{ ' } \textit{topspace } X$ )) (subtopology
 $X''$  ( $(g \circ f) \text{ ' } \textit{topspace } X$ ))  $g$ 
by (metis hm homeomorphic_eq_everything_map homeomorphic_map_subtopologies
image_comp subtopology_subtopology_topspace_subtopology)
then show ?thesis
unfolding embedding_map_def
using hm(1) homeomorphic_map_compose by blast
qed

```

lemma *surjective_embedding_map*:

```

embedding_map  $X$   $Y$   $f \wedge f \text{ ' } (\textit{topspace } X) = \textit{topspace } Y \iff \textit{homeomorphic_map}$ 
 $X$   $Y$   $f$ 
by (force simp: embedding_map_def homeomorphic_eq_everything_map)

```

lemma *embedding_map_in_subtopology*:

```

embedding_map  $X$  (subtopology  $Y$   $S$ )  $f \iff \textit{embedding_map}$   $X$   $Y$   $f \wedge f \text{ ' } (\textit{topspace}$ 
 $X) \subseteq S$  (is ?lhs = ?rhs)

```

proof

show *?lhs* \implies *?rhs*

unfolding *embedding_map_def*

by (*metis* *continuous_map_in_subtopology* *homeomorphic_imp_continuous_map*
inf_absorb2 *subtopology_subtopology*)

qed (*simp* *add*: *embedding_map_def* *inf.absorb_iff2* *subtopology_subtopology*)

lemma *injective_open_imp_embedding_map*:

```

 $\llbracket \textit{continuous_map } X$   $Y$   $f$ ; open_map  $X$   $Y$   $f$ ; inj_on  $f$  ( $\textit{topspace } X$ )  $\rrbracket \implies \textit{embed-}$ 
 $\textit{ding_map } X$   $Y$   $f$ 

```

unfolding *embedding_map_def* *homeomorphic_map_def*

by (*simp* *add*: *image_subset_iff_funcset* *continuous_map_in_subtopology*
continuous_open_quotient_map_eq_iff

open_map_imp_subset *open_map_into_subtopology*)

lemma *injective_closed_imp_embedding_map*:

```

 $\llbracket \textit{continuous_map } X$   $Y$   $f$ ; closed_map  $X$   $Y$   $f$ ; inj_on  $f$  ( $\textit{topspace } X$ )  $\rrbracket \implies \textit{embed-}$ 
 $\textit{ding_map } X$   $Y$   $f$ 

```

unfolding *embedding_map_def* *homeomorphic_map_def*

by (*simp* *add*: *closed_map_imp_subset* *closed_map_into_subtopology*
continuous_closed_quotient_map

continuous_map_in_subtopology *dual_order.eq_iff* *image_subset_iff_funcset*)

lemma *embedding_map_imp_homeomorphic_space*:

```

embedding_map  $X$   $Y$   $f \implies X$  homeomorphic_space (subtopology  $Y$  ( $f \text{ ' } (\textit{topspace}$ 
 $X)$ ))

```

unfolding *embedding_map_def*

using *homeomorphic_space* **by** *blast*

lemma *embedding_imp_closed_map*:

$\llbracket \text{embedding_map } X \ Y \ f; \text{closedin } Y \ (f \text{ ' } \text{topspace } X) \rrbracket \implies \text{closed_map } X \ Y \ f$

unfolding *closed_map_def*

by (*auto simp: closedin_closed_subtopology embedding_map_def homeomorphic_map_closedness_eq*)

lemma *embedding_imp_closed_map_eq*:

$\text{embedding_map } X \ Y \ f \implies (\text{closed_map } X \ Y \ f \longleftrightarrow \text{closedin } Y \ (f \text{ ' } \text{topspace } X))$

using *closed_map_def embedding_imp_closed_map* **by** *blast*

1.13.19 Retraction and section maps

definition *retraction_maps* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow \text{bool}$

where $\text{retraction_maps } X \ Y \ f \ g \equiv$

$\text{continuous_map } X \ Y \ f \wedge \text{continuous_map } Y \ X \ g \wedge (\forall x \in \text{topspace } Y. f(g \ x) = x)$

definition *section_map* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{bool}$

where $\text{section_map } X \ Y \ f \equiv \exists g. \text{retraction_maps } Y \ X \ g \ f$

definition *retraction_map* :: $'a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow ('a \Rightarrow 'b) \Rightarrow \text{bool}$

where $\text{retraction_map } X \ Y \ f \equiv \exists g. \text{retraction_maps } X \ Y \ f \ g$

lemma *retraction_maps_eq*:

$\llbracket \text{retraction_maps } X \ Y \ f \ g; \bigwedge x. x \in \text{topspace } X \implies f \ x = f' \ x; \bigwedge x. x \in \text{topspace } Y \implies g \ x = g' \ x \rrbracket$

$\implies \text{retraction_maps } X \ Y \ f' \ g'$

unfolding *retraction_maps_def*

by (*metis continuous_map_eq continuous_map_image_subset_topspace image_subset_iff*)

lemma *section_map_eq*:

$\llbracket \text{section_map } X \ Y \ f; \bigwedge x. x \in \text{topspace } X \implies f \ x = g \ x \rrbracket \implies \text{section_map } X \ Y \ g$

unfolding *section_map_def* **using** *retraction_maps_eq* **by** *blast*

lemma *retraction_map_eq*:

$\llbracket \text{retraction_map } X \ Y \ f; \bigwedge x. x \in \text{topspace } X \implies f \ x = g \ x \rrbracket \implies \text{retraction_map } X \ Y \ g$

unfolding *retraction_map_def* **using** *retraction_maps_eq* **by** *blast*

lemma *homeomorphic_imp_retraction_maps*:

$\text{homeomorphic_maps } X \ Y \ f \ g \implies \text{retraction_maps } X \ Y \ f \ g$

by (*simp add: homeomorphic_maps_def retraction_maps_def*)

lemma *section_and_retraction_eq_homeomorphic_map*:

$\text{section_map } X \ Y \ f \wedge \text{retraction_map } X \ Y \ f \longleftrightarrow \text{homeomorphic_map } X \ Y \ f$
(*is ?lhs = ?rhs*)

proof

```

assume ?lhs
then obtain g where homeomorphic_maps X Y f g
  unfolding homeomorphic_maps_def retraction_map_def section_map_def
  by (smt (verit) Pi_iff continuous_map_def retraction_maps_def)
then show ?rhs
  using homeomorphic_map_maps by blast
next
assume ?rhs
then show ?lhs
  unfolding retraction_map_def section_map_def
  by (meson homeomorphic_imp_retraction_maps homeomorphic_map_maps
homeomorphic_maps_sym)
qed

```

```

lemma section_imp_embedding_map:
  section_map X Y f  $\implies$  embedding_map X Y f
  unfolding section_map_def embedding_map_def homeomorphic_map_maps
retraction_maps_def homeomorphic_maps_def
  by (force simp: continuous_map_in_subtopology continuous_map_from_subtopology)

```

```

lemma retraction_imp_quotient_map:
  assumes retraction_map X Y f
  shows quotient_map X Y f
  unfolding quotient_map_def
proof (intro conjI subsetI allI impI)
  show f 'topspace X = topspace Y
    using assms by (force simp: retraction_map_def retraction_maps_def
continuous_map_def Pi_iff)
  next
  fix U
  assume U: U  $\subseteq$  topspace Y
  have openin Y U
    if  $\forall x \in \text{topspace } Y. g\ x \in \text{topspace } X \ \forall x \in \text{topspace } Y. f\ (g\ x) = x$ 
     $\text{openin } Y\ \{x \in \text{topspace } Y. g\ x \in \{x \in \text{topspace } X. f\ x \in U\}\}$  for g
    using openin_subopen U that by fastforce
  then show openin X {x  $\in$  topspace X. f x  $\in$  U} = openin Y U
    using assms by (auto simp: retraction_map_def retraction_maps_def
continuous_map_def Pi_iff)
qed

```

```

lemma retraction_maps_compose:
   $\llbracket \text{retraction\_maps } X\ Y\ f\ f'; \text{retraction\_maps } Y\ Z\ g\ g' \rrbracket \implies \text{retraction\_maps } X\ Z\ (g \circ f)\ (f' \circ g')$ 
  unfolding retraction_maps_def
  by (smt (verit, ccfv_threshold) comp_apply continuous_map_compose
continuous_map_image_subset_topospace image_subset_iff)

```

```

lemma retraction_map_compose:
   $\llbracket \text{retraction\_map } X\ Y\ f; \text{retraction\_map } Y\ Z\ g \rrbracket \implies \text{retraction\_map } X\ Z\ (g \circ f)$ 

```

f)
by (*meson retraction_map_def retraction_maps_compose*)

lemma *section_map_compose*:
 $\llbracket \text{section_map } X \ Y \ f; \text{ section_map } Y \ Z \ g \rrbracket \implies \text{section_map } X \ Z \ (g \circ f)$
by (*meson retraction_maps_compose section_map_def*)

lemma *surjective_section_eq_homeomorphic_map*:
 $\text{section_map } X \ Y \ f \wedge f^{-1}(\text{topspace } X) = \text{topspace } Y \iff \text{homeomorphic_map } X \ Y \ f$
by (*meson section_and_retraction_eq_homeomorphic_map section_imp_embedding_map surjective_embedding_map*)

lemma *surjective_retraction_or_section_map*:
 $f^{-1}(\text{topspace } X) = \text{topspace } Y \implies \text{retraction_map } X \ Y \ f \vee \text{section_map } X \ Y \ f \iff \text{retraction_map } X \ Y \ f$
using *section_and_retraction_eq_homeomorphic_map surjective_section_eq_homeomorphic_map*
by *fastforce*

lemma *retraction_imp_surjective_map*:
 $\text{retraction_map } X \ Y \ f \implies f^{-1}(\text{topspace } X) = \text{topspace } Y$
by (*simp add: retraction_imp_quotient_map quotient_imp_surjective_map*)

lemma *section_imp_injective_map*:
 $\llbracket \text{section_map } X \ Y \ f; x \in \text{topspace } X; y \in \text{topspace } X \rrbracket \implies f \ x = f \ y \iff x = y$
by (*metis (mono_tags, opaque_lifting) retraction_maps_def section_map_def*)

lemma *retraction_maps_to_retract_maps*:
 $\text{retraction_maps } X \ Y \ r \ s \implies \text{retraction_maps } X \ (\text{subtopology } X \ (s^{-1}(\text{topspace } Y))) \ (s \circ r) \ \text{id}$
unfolding *retraction_maps_def*
by (*auto simp: continuous_map_compose continuous_map_into_subtopology continuous_map_from_subtopology*)

1.13.20 Continuity

lemma *continuous_on_open*:
 $\text{continuous_on } S \ f \iff (\forall T. \text{openin } (\text{top_of_set } (f^{-1} S)) \ T \implies \text{openin } (\text{top_of_set } S) \ (S \cap f^{-1} T))$
unfolding *continuous_on_open_invariant openin_open Int_def vimage_def Int_commute*
by (*simp add: imp_ex imageI conj_commute eq_commute cong: conj_cong*)

lemma *continuous_on_closed*:
 $\text{continuous_on } S \ f \iff (\forall T. \text{closedin } (\text{top_of_set } (f^{-1} S)) \ T \implies \text{closedin } (\text{top_of_set } S) \ (S \cap f^{-1} T))$
unfolding *continuous_on_closed_invariant closedin_closed Int_def vimage_def Int_commute*

by (simp add: imp_ex imageI conj_commute eq_commute cong: conj_cong)

lemma *continuous_on_imp_closedin*:

assumes *continuous_on* S *f* *closedin* (*top_of_set* ($f^{-1} S$)) T

shows *closedin* (*top_of_set* S) ($S \cap f^{-1} T$)

using *assms* *continuous_on_closed* **by** *blast*

lemma *continuous_map_subtopology_eu* [*simp*]:

continuous_map (*top_of_set* S) (*subtopology euclidean* T) $h \longleftrightarrow$ *continuous_on* S $h \wedge h^{-1} S \subseteq T$

by (simp add: *continuous_map_in_subtopology*)

lemma *continuous_map_euclidean_top_of_set*:

assumes *eq*: $f^{-1} S = \text{UNIV}$ **and** *cont*: *continuous_on* UNIV f

shows *continuous_map euclidean* (*top_of_set* S) f

by (simp add: *cont* *continuous_map_in_subtopology* *eq_image_subset_iff_subset_vimage*)

1.13.21 Half-global and completely global cases

lemma *continuous_openin_preimage_gen*:

assumes *continuous_on* S *f* *open* T

shows *openin* (*top_of_set* S) ($S \cap f^{-1} T$)

proof –

have *: ($S \cap f^{-1} T$) = ($S \cap f^{-1} (T \cap f^{-1} S)$)

by *auto*

have *openin* (*top_of_set* ($f^{-1} S$)) ($T \cap f^{-1} S$)

using *openin_open_Int*[of T $f^{-1} S$, *OF* *assms*(2)] **unfolding** *openin_open* **by** *auto*

then show *?thesis*

using *assms*(1)[*unfolded* *continuous_on_open*, *THEN* *spec*[**where** $x = T \cap f^{-1} S$]]

using * **by** *auto*

qed

lemma *continuous_closedin_preimage*:

assumes *continuous_on* S *f* **and** *closed* T

shows *closedin* (*top_of_set* S) ($S \cap f^{-1} T$)

proof –

have *: ($S \cap f^{-1} T$) = ($S \cap f^{-1} (T \cap f^{-1} S)$)

by *auto*

have *closedin* (*top_of_set* ($f^{-1} S$)) ($T \cap f^{-1} S$)

using *closedin_closed_Int*[of T $f^{-1} S$, *OF* *assms*(2)]

by (simp add: *Int_commute*)

then show *?thesis*

using *assms*(1)[*unfolded* *continuous_on_closed*, *THEN* *spec*[**where** $x = T \cap f^{-1} S$]]

using * **by** *auto*

qed

lemma *continuous_openin_preimage_eq*:

continuous_on $S f \longleftrightarrow (\forall T. \text{open } T \longrightarrow \text{openin } (\text{top_of_set } S) (S \cap f^{-1} T))$

by (*metis Int_commute continuous_on_open_invariant open_openin openin_subtopology*)

lemma *continuous_closedin_preimage_eq*:

continuous_on $S f \longleftrightarrow$

$(\forall T. \text{closed } T \longrightarrow \text{closedin } (\text{top_of_set } S) (S \cap f^{-1} T))$

by (*metis Int_commute closedin_closed continuous_on_closed_invariant*)

lemma *continuous_open_preimage*:

assumes *conf*: *continuous_on* $S f$ **and** *open* S *open* T

shows *open* $(S \cap f^{-1} T)$

proof –

obtain U **where** *open* U $(S \cap f^{-1} T) = S \cap U$

using *continuous_openin_preimage_gen*[*OF* *conf* \langle *open* T \rangle]

unfolding *openin_open* **by** *auto*

then show *?thesis*

using *open_Int*[*of* $S U$, *OF* \langle *open* S \rangle] **by** *auto*

qed

lemma *continuous_closed_preimage*:

assumes *conf*: *continuous_on* $S f$ **and** *closed* S *closed* T

shows *closed* $(S \cap f^{-1} T)$

proof –

obtain U **where** *closed* U $(S \cap f^{-1} T) = S \cap U$

using *continuous_closedin_preimage*[*OF* *conf* \langle *closed* T \rangle]

unfolding *closedin_closed* **by** *auto*

then show *?thesis* **using** *closed_Int*[*of* $S U$, *OF* \langle *closed* S \rangle] **by** *auto*

qed

lemma *continuous_open_vimage*: *open* $S \implies (\bigwedge x. \text{continuous } (\text{at } x) f) \implies \text{open } (f^{-1} S)$

by (*metis continuous_on_eq_continuous_within open_vimage*)

lemma *continuous_closed_vimage*: *closed* $S \implies (\bigwedge x. \text{continuous } (\text{at } x) f) \implies \text{closed } (f^{-1} S)$

by (*simp add: closed_vimage continuous_on_eq_continuous_within*)

lemma *Times_in_interior_subtopology*:

assumes $(x, y) \in U$ *openin* $(\text{top_of_set } (S \times T)) U$

obtains $V W$ **where** *openin* $(\text{top_of_set } S) V$ $x \in V$

openin $(\text{top_of_set } T) W$ $y \in W$ $(V \times W) \subseteq U$

proof –

from *assms* **obtain** E **where** *open* E $U = S \times T \cap E$ $(x, y) \in E$ $x \in S$ $y \in T$

by (*auto simp: openin_open*)

from *open_prod_elim*[*OF* \langle *open* E \rangle \langle $(x, y) \in E$ \rangle]

obtain $E1 E2$ **where** *open* $E1$ *open* $E2$ $(x, y) \in E1 \times E2$ $E1 \times E2 \subseteq E$

by *blast*

show *?thesis*

```

proof
  show  $\text{openin } (\text{top\_of\_set } S) (E1 \cap S) \text{ openin } (\text{top\_of\_set } T) (E2 \cap T)$ 
    using  $\langle \text{open } E1 \rangle \langle \text{open } E2 \rangle$  by  $(\text{auto simp: openin\_open})$ 
  show  $x \in E1 \cap S \ y \in E2 \cap T$ 
    using  $\langle (x, y) \in E1 \times E2 \rangle \langle x \in S \rangle \langle y \in T \rangle$  by  $\text{auto}$ 
  show  $(E1 \cap S) \times (E2 \cap T) \subseteq U$ 
    using  $\langle E1 \times E2 \subseteq E \rangle \langle U = \_ \rangle$  by  $\text{auto}$ 
qed
qed

lemma  $\text{closedin\_Times}$ :
   $\text{closedin } (\text{top\_of\_set } S) S' \implies \text{closedin } (\text{top\_of\_set } T) T' \implies$ 
   $\text{closedin } (\text{top\_of\_set } (S \times T)) (S' \times T')$ 
  unfolding  $\text{closedin\_closed}$  using  $\text{closed\_Times}$  by  $\text{blast}$ 

lemma  $\text{openin\_Times}$ :
   $\text{openin } (\text{top\_of\_set } S) S' \implies \text{openin } (\text{top\_of\_set } T) T' \implies$ 
   $\text{openin } (\text{top\_of\_set } (S \times T)) (S' \times T')$ 
  unfolding  $\text{openin\_open}$  using  $\text{open\_Times}$  by  $\text{blast}$ 

lemma  $\text{openin\_Times\_eq}$ :
  fixes  $S :: 'a::\text{topological\_space set}$  and  $T :: 'b::\text{topological\_space set}$ 
  shows
     $\text{openin } (\text{top\_of\_set } (S \times T)) (S' \times T') \longleftrightarrow$ 
     $S' = \{\} \vee T' = \{\} \vee \text{openin } (\text{top\_of\_set } S) S' \wedge \text{openin } (\text{top\_of\_set } T) T'$ 
    (is ?lhs = ?rhs)
  proof  $(\text{cases } S' = \{\} \vee T' = \{\})$ 
    case  $\text{True}$ 
      then show  $?thesis$  by  $\text{auto}$ 
  next
    case  $\text{False}$ 
      then obtain  $x \ y$  where  $x \in S' \ y \in T'$ 
      by  $\text{blast}$ 
      show  $?thesis$ 
      proof
        assume  $?lhs$ 
        have  $\text{openin } (\text{top\_of\_set } S) S'$ 
        proof  $(\text{subst openin\_subopen, clarify})$ 
          show  $\exists U. \text{openin } (\text{top\_of\_set } S) U \wedge x \in U \wedge U \subseteq S'$  if  $x \in S'$  for  $x$ 
          using  $\text{that } \langle y \in T' \rangle \text{Times\_in\_interior\_subtopology } [OF\_ \langle ?lhs \rangle, \text{of } x \ y]$ 
          by  $\text{simp } (\text{metis mem\_Sigma\_iff subsetD subsetI})$ 
        qed
        moreover have  $\text{openin } (\text{top\_of\_set } T) T'$ 
        proof  $(\text{subst openin\_subopen, clarify})$ 
          show  $\exists U. \text{openin } (\text{top\_of\_set } T) U \wedge y \in U \wedge U \subseteq T'$  if  $y \in T'$  for  $y$ 
          using  $\text{that } \langle x \in S' \rangle \text{Times\_in\_interior\_subtopology } [OF\_ \langle ?lhs \rangle, \text{of } x \ y]$ 
          by  $\text{simp } (\text{metis mem\_Sigma\_iff subsetD subsetI})$ 
        qed
        ultimately show  $?rhs$ 

```

```

    by simp
  next
    assume ?rhs
    with False show ?lhs
      by (simp add: openin_Times)
  qed
qed

```

```

lemma Lim_transform_within_openin:
  assumes f: (f ⟶ l) (at a within T)
    and openin (top_of_set T) S a ∈ S
    and eq:  $\bigwedge x. [x \in S; x \neq a] \implies f x = g x$ 
  shows (g ⟶ l) (at a within T)
proof -
  have  $\forall_F x$  in at a within T.  $x \in T \wedge x \neq a$ 
    by (simp add: eventually_at_filter)
  moreover
  from ⟨openin _ _⟩ obtain U where open U S = T ∩ U
    by (auto simp: openin_open)
  then have a ∈ U using ⟨a ∈ S⟩ by auto
  from topological_tendstoD[OF tendsto_ident_at ⟨open U⟩ ⟨a ∈ U⟩]
  have  $\forall_F x$  in at a within T.  $x \in U$  by auto
  ultimately
  have  $\forall_F x$  in at a within T.  $f x = g x$ 
    by eventually_elim (auto simp: ⟨S = _⟩ eq)
  with f show ?thesis
    by (rule Lim_transform_eventually)
qed

```

```

lemma continuous_on_open_gen:
  assumes f ∈ S → T
  shows continuous_on S f ⟷
    ( $\forall U. \text{openin } (\text{top\_of\_set } T) U$ 
       $\longrightarrow \text{openin } (\text{top\_of\_set } S) (S \cap f^{-1} U)$ )
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  with assms show ?rhs
    apply (simp add: Pi_iff_continuous_openin_preimage_eq openin_open)
    by (metis inf.orderE inf_assoc subsetI vimageI vimage_Int)
next
  assume R [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp add: continuous_openin_preimage_eq)
    show  $\bigwedge T. \text{open } T \implies \text{openin } (\text{top\_of\_set } S) (S \cap f^{-1} T)$ 
      by (metis (no_types) R assms image_subset_iff funcset_image_subset_iff_subset_vimage
        inf.orderE inf_assoc openin_open_Int vimage_Int)
  qed
qed

```

lemma *continuous_openin_preimage*:

[[*continuous_on* S f ; $f \in S \rightarrow T$; *openin* (*top_of_set* T) U]]
 \implies *openin* (*top_of_set* S) ($S \cap f^{-1} U$)
by (*simp add: continuous_on_open_gen*)

lemma *continuous_on_closed_gen*:

assumes $f \in S \rightarrow T$
shows *continuous_on* S $f \longleftrightarrow$
 $(\forall U. \text{closedin} (\text{top_of_set } T) U \longrightarrow \text{closedin} (\text{top_of_set } S) (S \cap f^{-1} U))$

proof –

have *: $U \subseteq T \implies S \cap f^{-1} (T - U) = S - (S \cap f^{-1} U)$ **for** U

using *assms* **by** *blast*

then show ?*thesis*

unfolding *continuous_on_open_gen* [*OF assms*]

by (*metis closedin_def inf.cobounded1 openin_closedin_eq topspace_euclidean_subtopology*)

qed

lemma *continuous_closedin_preimage_gen*:

assumes *continuous_on* S f $f \in S \rightarrow T$ *closedin* (*top_of_set* T) U
shows *closedin* (*top_of_set* S) ($S \cap f^{-1} U$)
using *assms* *continuous_on_closed_gen* **by** *blast*

lemma *continuous_transform_within_openin*:

assumes *continuous* (*at a* *within* T) f
and *openin* (*top_of_set* T) S $a \in S$
and *eq*: $\bigwedge x. x \in S \implies f x = g x$
shows *continuous* (*at a* *within* T) g
using *assms* **by** (*simp add: Lim_transform_within_openin continuous_within*)

1.13.22 The topology generated by some (open) subsets

In the definition below of a generated topology, the *Empty* case is not necessary, as it follows from *UN* taking for K the empty set. However, it is convenient to have, and is never a problem in proofs, so I prefer to write it down explicitly.

We do not require *UNIV* to be an open set, as this will not be the case in applications. (We are thinking of a topology on a subset of *UNIV*, the remaining part of *UNIV* being irrelevant.)

inductive *generate_topology_on* **for** S **where**

Empty: *generate_topology_on* S $\{\}$

| *Int*: *generate_topology_on* S $a \implies$ *generate_topology_on* S $b \implies$ *generate_topology_on* S $(a \cap b)$

| *UN*: $(\bigwedge k. k \in K \implies$ *generate_topology_on* S $k) \implies$ *generate_topology_on* S $(\bigcup K)$

| *Basis*: $s \in S \implies$ *generate_topology_on* S s

lemma *istopology_generate_topology_on*:
istopology (generate_topology_on S)
unfolding *istopology_def* **by** (*auto intro: generate_topology_on.intros*)

The basic property of the topology generated by a set S is that it is the smallest topology containing all the elements of S :

lemma *generate_topology_on_coarsest*:
assumes T : *istopology T* \wedge $s. s \in S \implies T s$
and *gen: generate_topology_on S s0*
shows $T s0$
using *gen*
by (*induct rule: generate_topology_on.induct*) (*use T in <auto simp: istopology_def>*)

abbreviation *topology_generated_by*::('a set set) \Rightarrow ('a topology)
where *topology_generated_by S* \equiv *topology (generate_topology_on S)*

lemma *openin_topology_generated_by_iff*:
openin (topology_generated_by S) s \iff *generate_topology_on S s*
using *topology_inverse'[OF istopology_generate_topology_on[of S]]* **by** *simp*

lemma *openin_topology_generated_by*:
openin (topology_generated_by S) s \implies *generate_topology_on S s*
using *openin_topology_generated_by_iff* **by** *auto*

lemma *topology_generated_by_topspace [simp]*:
topspace (topology_generated_by S) = ($\bigcup S$)
proof
{
fix s **assume** *openin (topology_generated_by S) s*
then have *generate_topology_on S s* **by** (*rule openin_topology_generated_by*)
then have $s \subseteq ($\bigcup S$)$ **by** (*induct, auto*)
}
then show *topspace (topology_generated_by S) \subseteq ($\bigcup S$)*
unfolding *topspace_def* **by** *auto*
next
have *generate_topology_on S ($\bigcup S$)*
using *generate_topology_on.UN[OF generate_topology_on.Basis, of S S]* **by**
simp
then show $(\bigcup S) \subseteq$ *topspace (topology_generated_by S)*
unfolding *topspace_def* **using** *openin_topology_generated_by_iff* **by** *auto*
qed

lemma *topology_generated_by_Basis*:
 $s \in S \implies$ *openin (topology_generated_by S) s*
by (*simp add: Basis openin_topology_generated_by_iff*)

lemma *generate_topology_on_Inter*:
 \llbracket *finite F; $\wedge K. K \in \mathcal{F} \implies$ generate_topology_on S K; $\mathcal{F} \neq \{\}$* $\rrbracket \implies$ *generate_topology_on S ($\bigcap \mathcal{F}$)*

by (induction \mathcal{F} rule: finite_induct; force intro: generate_topology_on.intros)

1.13.23 Topology bases and sub-bases

lemma *istopology_base_alt*:

$istopology\ (arbitrary_union_of\ P) \longleftrightarrow$
 $(\forall S\ T. (arbitrary_union_of\ P)\ S \wedge (arbitrary_union_of\ P)\ T$
 $\longrightarrow (arbitrary_union_of\ P)\ (S \cap T))$

by (simp add: istopology_def) (blast intro: arbitrary_union_of_Union)

lemma *istopology_base_eq*:

$istopology\ (arbitrary_union_of\ P) \longleftrightarrow$
 $(\forall S\ T. P\ S \wedge P\ T \longrightarrow (arbitrary_union_of\ P)\ (S \cap T))$

by (simp add: istopology_base_alt arbitrary_union_of_Int_eq)

lemma *istopology_base*:

$(\bigwedge S\ T. \llbracket P\ S; P\ T \rrbracket \Longrightarrow P(S \cap T)) \Longrightarrow istopology\ (arbitrary_union_of\ P)$

by (simp add: arbitrary_def istopology_base_eq union_of_inc)

lemma *openin_topology_base_unique*:

$openin\ X = arbitrary_union_of\ P \longleftrightarrow$
 $(\forall V. P\ V \longrightarrow openin\ X\ V) \wedge (\forall U\ x. openin\ X\ U \wedge x \in U \longrightarrow (\exists V. P\ V$
 $\wedge x \in V \wedge V \subseteq U))$
 (is ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

by (auto simp: union_of_def arbitrary_def)

next

assume R : ?rhs

then have $*$: $\exists U \subseteq Collect\ P. \bigcup U = S$ if $openin\ X\ S$ for S

using that by (rule_tac $x = \{V. P\ V \wedge V \subseteq S\}$ in exI) fastforce
 from R show ?lhs

by (fastforce simp add: union_of_def arbitrary_def intro: $*$)

qed

lemma *topology_base_unique*:

assumes $\bigwedge S. P\ S \Longrightarrow openin\ X\ S$

$\bigwedge U\ x. \llbracket openin\ X\ U; x \in U \rrbracket \Longrightarrow \exists B. P\ B \wedge x \in B \wedge B \subseteq U$

shows $topology\ (arbitrary_union_of\ P) = X$

proof –

have $X = topology\ (openin\ X)$

by (simp add: openin_inverse)

also from $assms$ have $openin\ X = arbitrary_union_of\ P$

by (subst openin_topology_base_unique) auto

finally show ?thesis ..

qed

lemma *topology_bases_eq_aux*:

$$\begin{aligned} & \llbracket (\text{arbitrary_union_of } P) S; \\ & \quad \bigwedge U x. \llbracket P U; x \in U \rrbracket \implies \exists V. Q V \wedge x \in V \wedge V \subseteq U \rrbracket \\ & \implies (\text{arbitrary_union_of } Q) S \end{aligned}$$
by (*metis arbitrary_union_of_alt arbitrary_union_of_idempot*)

lemma topology_bases_eq:

$$\begin{aligned} & \llbracket \bigwedge U x. \llbracket P U; x \in U \rrbracket \implies \exists V. Q V \wedge x \in V \wedge V \subseteq U; \\ & \quad \bigwedge V x. \llbracket Q V; x \in V \rrbracket \implies \exists U. P U \wedge x \in U \wedge U \subseteq V \rrbracket \\ & \implies \text{topology } (\text{arbitrary_union_of } P) = \\ & \quad \text{topology } (\text{arbitrary_union_of } Q) \end{aligned}$$
by (*fastforce intro: arg_cong [where f=topology] elim: topology_bases_eq_aux*)

lemma istopology_subbase:

$$\text{istopology } (\text{arbitrary_union_of } (\text{finite_intersection_of } P \text{ relative_to } S))$$
by (*simp add: finite_intersection_of_Int istopology_base relative_to_Int*)

lemma openin_subbase:

$$\text{openin } (\text{topology } (\text{arbitrary_union_of } (\text{finite_intersection_of } B \text{ relative_to } U)))$$

$$S$$

$$\longleftrightarrow (\text{arbitrary_union_of } (\text{finite_intersection_of } B \text{ relative_to } U)) S$$
by (*simp add: istopology_subbase topology_inverse'*)

lemma topspace_subbase [simp]:

$$\text{topspace } (\text{topology } (\text{arbitrary_union_of } (\text{finite_intersection_of } B \text{ relative_to } U)))$$

$$= U \text{ (is ?lhs = _)}$$

proof

show $?lhs \subseteq U$

by (*metis arbitrary_union_of_relative_to openin_subbase openin_topspace relative_to_imp_subset*)

show $U \subseteq ?lhs$

by (*metis arbitrary_union_of_inc finite_intersection_of_empty inf.orderE istopology_subbase openin_subset relative_to_inc subset_UNIV topology_inverse'*)

qed

lemma minimal_topology_subbase:

assumes $X: \bigwedge S. P S \implies \text{openin } X S$ **and** $\text{openin } X U$

and $S: \text{openin } (\text{topology } (\text{arbitrary_union_of } (\text{finite_intersection_of } P \text{ relative_to } U))) S$

shows $\text{openin } X S$

proof –

have $(\text{arbitrary_union_of } (\text{finite_intersection_of } P \text{ relative_to } U)) S$

using S openin_subbase **by** *blast*

with $X \langle \text{openin } X U \rangle$ **show** $?thesis$

by (*force simp add: union_of_def intersection_of_def relative_to_def intro: openin_Int_Inter*)

qed

lemma istopology_subbase_UNIV:

istopology (arbitrary_union_of (finite_intersection_of P))
 by (simp add: istopology_base finite_intersection_of_Int)

lemma generate_topology_on_eq:

generate_topology_on S = arbitrary_union_of finite' intersection_of ($\lambda x. x \in S$) (is ?lhs = ?rhs)

proof (intro ext iffI)

fix A

assume ?lhs A

then show ?rhs A

proof induction

case (Int a b)

then show ?case

by (metis (mono_tags, lifting) istopology_base_alt finite'_intersection_of_Int istopology_base)

next

case (UN K)

then show ?case

by (simp add: arbitrary_union_of_Union)

next

case (Basis s)

then show ?case

by (simp add: Sup_upper arbitrary_union_of_inc finite'_intersection_of_inc relative_to_subset_inc)

qed auto

next

fix A

assume ?rhs A

then obtain \mathcal{U} **where** $\mathcal{U}: \bigwedge T. T \in \mathcal{U} \implies \exists \mathcal{F}. \text{finite}' \mathcal{F} \wedge \mathcal{F} \subseteq S \wedge \bigcap \mathcal{F} = T$

and eq: $A = \bigcup \mathcal{U}$

unfolding union_of_def intersection_of_def **by** auto

show ?lhs A

unfolding eq

proof (rule generate_topology_on.UN)

fix T

assume $T \in \mathcal{U}$

with \mathcal{U} **obtain** \mathcal{F} **where** $\text{finite}' \mathcal{F} \wedge \mathcal{F} \subseteq S \wedge \bigcap \mathcal{F} = T$

by blast

have *generate_topology_on* S ($\bigcap \mathcal{F}$)

proof (rule generate_topology_on_Inter)

show $\text{finite}' \mathcal{F} \wedge \mathcal{F} \neq \{\}$

by (auto simp: ‹finite' \mathcal{F} ›)

show $\bigwedge K. K \in \mathcal{F} \implies \text{generate_topology_on } S K$

by (metis ‹ $\mathcal{F} \subseteq S$ › generate_topology_on.simps subset_iff)

qed

then show *generate_topology_on* S T

using ‹ $\bigcap \mathcal{F} = T$ › by blast

qed

qed

lemma *continuous_on_generated_topo_iff*:

continuous_map *T1* (*topology_generated_by* *S*) *f* \longleftrightarrow

$((\forall U. U \in S \longrightarrow \text{openin } T1 (f^{-1}U \cap \text{topspace}(T1))) \wedge (f^{-1}(\text{topspace } T1) \subseteq (\bigcup S)))$

unfolding *continuous_map_alt* *topology_generated_by_topospace*

proof (*auto simp add: topology_generated_by_Basis*)

assume *H*: $\forall U. U \in S \longrightarrow \text{openin } T1 (f^{-1}U \cap \text{topspace } T1)$

fix *U* **assume** *openin* (*topology_generated_by* *S*) *U*

then have *generate_topology_on* *S* *U* **by** (*rule* *openin_topology_generated_by*)

then show *openin* *T1* ($f^{-1}U \cap \text{topspace } T1$)

proof (*induct*)

fix *a b*

assume *H*: *openin* *T1* ($f^{-1}a \cap \text{topspace } T1$) *openin* *T1* ($f^{-1}b \cap \text{topspace } T1$)

have $f^{-1}(a \cap b) \cap \text{topspace } T1 = (f^{-1}a \cap \text{topspace } T1) \cap (f^{-1}b \cap \text{topspace } T1)$

by *auto*

then show *openin* *T1* ($f^{-1}(a \cap b) \cap \text{topspace } T1$) **using** *H* **by** *auto*

next

fix *K*

assume *H*: *openin* *T1* ($f^{-1}k \cap \text{topspace } T1$) **if** *k* \in *K* **for** *k*

define *L* **where** $L = \{f^{-1}k \cap \text{topspace } T1 \mid k. k \in K\}$

have $*$: *openin* *T1* *l* **if** *l* \in *L* **for** *l* **using** *that H unfolding L_def* **by** *auto*

have *openin* *T1* $(\bigcup L)$ **using** *openin_Union[OF *]* **by** *simp*

moreover have $(\bigcup L) = (f^{-1}(\bigcup K \cap \text{topspace } T1))$ **unfolding** *L_def* **by** *auto*

ultimately show *openin* *T1* ($f^{-1}(\bigcup K \cap \text{topspace } T1)$) **by** *simp*

qed (*auto simp add: H*)

qed

lemma *continuous_on_generated_topo*:

assumes $\bigwedge U. U \in S \implies \text{openin } T1 (f^{-1}U \cap \text{topspace}(T1))$

$f^{-1}(\text{topspace } T1) \subseteq (\bigcup S)$

shows *continuous_map* *T1* (*topology_generated_by* *S*) *f*

using *assms continuous_on_generated_topo_iff* **by** *blast*

1.13.24 Continuity via bases/subbases, hence upper and lower semicontinuity

lemma *continuous_map_into_topology_base*:

assumes *P*: *openin* *Y* = *arbitrary_union_of* *P*

and *f*: $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y$

and *ope*: $\bigwedge U. P U \implies \text{openin } X \{x \in \text{topspace } X. f x \in U\}$

shows *continuous_map* *X* *Y* *f*

proof –

have $*$: $\bigwedge \mathcal{U}. (\bigwedge t. t \in \mathcal{U} \implies P t) \implies \text{openin } X \{x \in \text{topspace } X. \exists U \in \mathcal{U}. f x \in U\}$

by (*smt* (*verit*) *Ball_Collect* *ope* *mem_Collect_eq* *openin_subopen*)

```

show ?thesis
  using P by (auto simp: continuous_map_def arbitrary_def union_of_def
intro!: f *)
qed

```

```

lemma continuous_map_into_topology_base_eq:
  assumes P: openin Y = arbitrary_union_of P
  shows
    continuous_map X Y f  $\longleftrightarrow$ 
    f  $\in$  topspace X  $\rightarrow$  topspace Y  $\wedge$  ( $\forall U. P U \longrightarrow$  openin X {x  $\in$  topspace X. f x
 $\in U$ })
  (is ?lhs=?rhs)
proof
  assume L: ?lhs
  then have f  $\in$  topspace X  $\rightarrow$  topspace Y
    by (simp add: continuous_map_funspace)
  moreover have  $\bigwedge U. P U \implies$  openin X {x  $\in$  topspace X. f x  $\in U$ }
    using L assms continuous_map openin_topology_base_unique by fastforce
  ultimately show ?rhs by auto
qed (simp add: assms Pi_iff continuous_map_into_topology_base)

```

```

lemma continuous_map_into_topology_subbase:
  fixes U P
  defines Y  $\equiv$  topology(arbitrary_union_of (finite_intersection_of P relative_to
U))
  assumes f:  $\bigwedge x. x \in$  topspace X  $\implies$  f x  $\in$  topspace Y
  and ope:  $\bigwedge U. P U \implies$  openin X {x  $\in$  topspace X. f x  $\in U$ }
  shows continuous_map X Y f
proof (intro continuous_map_into_topology_base)
  show openin Y = arbitrary_union_of (finite_intersection_of P relative_to U)
    unfolding Y_def using istopology_subbase topology_inverse' by blast
  show openin X {x  $\in$  topspace X. f x  $\in V$ }
    if  $\S$ : (finite_intersection_of P relative_to U) V for V
  proof -
  define finv where finv  $\equiv$   $\lambda V. \{x \in$  topspace X. f x  $\in V\}$ 
  obtain  $\mathcal{U}$  where  $\mathcal{U}$ : finite  $\mathcal{U} \wedge \forall V. V \in \mathcal{U} \implies P V$ 
    {x  $\in$  topspace X. f x  $\in V$ } = ( $\bigcap V \in$  insert U  $\mathcal{U}. finv V$ )
  using  $\S$  by (fastforce simp: finv_def intersection_of_def relative_to_def)
  show ?thesis
  unfolding  $\mathcal{U}$ 
proof (intro openin_Inter ope)
  have U: U = topspace Y
    unfolding Y_def using topspace_subbase by fastforce
  fix V
  assume V: V  $\in$  finv 'insert U  $\mathcal{U}$ 
  with U f have openin X {x  $\in$  topspace X. f x  $\in U$ }
    by (auto simp: openin_subopen [of X Collect _])
  then show openin X V
    using V U(2) ope by (fastforce simp: finv_def)

```

```

    qed (use ⟨finite U⟩ in auto)
  qed
qed (use f in auto)

lemma continuous_map_into_topology_subbase_eq:
  assumes  $Y = \text{topology}(\text{arbitrary\_union\_of } (\text{finite\_intersection\_of } P \text{ relative\_to } U))$ 
  shows
    continuous_map  $X$   $Y$   $f \iff$ 
     $f \in \text{topspace } X \rightarrow \text{topspace } Y \wedge (\forall U. P U \longrightarrow \text{openin } X \{x \in \text{topspace } X. f x \in U\})$ 
    (is ?lhs=?rhs)
  proof
    assume  $L: ?lhs$ 
    show ?rhs
    proof (intro conjI strip)
      show  $f \in \text{topspace } X \rightarrow \text{topspace } Y$ 
        using  $L$  continuous_map_def by fastforce
      fix  $V$ 
      assume  $P V$ 
      have  $U = \text{topspace } Y$ 
        unfolding assms using topology_subbase by fastforce
      then have  $eq: \{x \in \text{topspace } X. f x \in V\} = \{x \in \text{topspace } X. f x \in U \cap V\}$ 
        using  $L$  by (auto simp: continuous_map)
      have  $\text{openin } Y (U \cap V)$ 
        unfolding assms openin_subbase
        by (meson ⟨ $P V$ ⟩ arbitrary_union_of_inc finite_intersection_of_inc relative_to_inc)
      show  $\text{openin } X \{x \in \text{topspace } X. f x \in V\}$ 
        using  $L$  ⟨ $\text{openin } Y (U \cap V)$ ⟩ continuous_map eq by fastforce
    qed
  qed
next
  show ?rhs  $\implies$  ?lhs
    unfolding assms
    by (intro continuous_map_into_topology_subbase) auto
qed

```

```

lemma subbase_subtopology_euclidean:
  fixes  $U :: 'a::\text{order\_topology\_set}$ 
  shows
    topology
      (arbitrary_union_of
        (finite_intersection_of  $(\lambda x. x \in \text{range } \text{greaterThan} \cup \text{range } \text{lessThan})$  relative_to  $U$ ))
    = subtopology_euclidean  $U$ 
  proof -
    have  $\exists V. (\text{finite\_intersection\_of } (\lambda x. x \in \text{range } \text{greaterThan} \vee x \in \text{range } \text{lessThan})) V \wedge x \in V \wedge V \subseteq W$ 
      if  $\text{open } W \ x \in W$  for  $W$  and  $x::'a$ 

```

```

    using ‹open W› [unfolded open_generated_order] ‹x ∈ W›
  proof (induct rule: generate_topology.induct)
    case UNIV
    then show ?case
      using finite_intersection_of_empty by blast
  next
    case (Int a b)
    then show ?case
      by (meson Int_iff finite_intersection_of_Int inf_mono)
  next
    case (UN K)
    then show ?case
      by (meson Union_iff subset_iff)
  next
    case (Basis s)
    then show ?case
      by (metis (no_types, lifting) Un_iff finite_intersection_of_inc order_refl)
  qed
  moreover
  have  $\bigwedge V::'a$  set. (finite_intersection_of ( $\lambda x. x \in \text{range greaterThan} \vee x \in \text{range lessThan}$ ))  $V \implies \text{open } V$ 
    by (force simp: intersection_of_def subset_iff)
  ultimately have *: openin (euclidean::'a topology) =
    (arbitrary_union_of (finite_intersection_of ( $\lambda x. x \in \text{range greaterThan} \vee x \in \text{range lessThan}$ )))
    by (smt (verit, best) openin_topology_base_unique open_openin)
  show ?thesis
    unfolding subtopology_def arbitrary_union_of_relative_to [symmetric]
    apply (simp add: relative_to_def flip: *)
    by (metis Int_commute)
  qed

lemma continuous_map_upper_lower_semicontinuous_lt_gen:
  fixes U :: 'a::order_topology set
  shows continuous_map X (subtopology euclidean U) f  $\longleftrightarrow$ 
    ( $\forall x \in \text{topspace } X. f x \in U$ )  $\wedge$ 
    ( $\forall a. \text{openin } X \{x \in \text{topspace } X. f x > a\}$ )  $\wedge$ 
    ( $\forall a. \text{openin } X \{x \in \text{topspace } X. f x < a\}$ )
  by (auto simp: continuous_map_into_topology_subbase_eq [OF subbase_subtopology_euclidean
    [symmetric, of U]]
    greaterThan_def lessThan_def image_iff simp flip: all_simps)

lemma continuous_map_upper_lower_semicontinuous_lt:
  fixes f :: 'a  $\Rightarrow$  'b::order_topology
  shows continuous_map X euclidean f  $\longleftrightarrow$ 
    ( $\forall a. \text{openin } X \{x \in \text{topspace } X. f x > a\}$ )  $\wedge$ 
    ( $\forall a. \text{openin } X \{x \in \text{topspace } X. f x < a\}$ )
  using continuous_map_upper_lower_semicontinuous_lt_gen [where U=UNIV]
  by auto

```

lemma *Int_Collect_imp_eq*: $A \cap \{x. x \in A \longrightarrow P x\} = \{x \in A. P x\}$
by *blast*

lemma *continuous_map_upper_lower_semicontinuous_le_gen*:
shows

continuous_map X (*subtopology euclideanreal* U) $f \longleftrightarrow$
 $(\forall x \in \text{topspace } X. f x \in U) \wedge$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \geq a\}) \wedge$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \leq a\})$

unfolding *continuous_map_upper_lower_semicontinuous_lt_gen*

by (*auto simp: closedin_def Diff_eq Compl_eq not_le Int_Collect_imp_eq*)

lemma *continuous_map_upper_lower_semicontinuous_le*:

continuous_map X *euclideanreal* $f \longleftrightarrow$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \geq a\}) \wedge$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \leq a\})$

using *continuous_map_upper_lower_semicontinuous_le_gen* [**where** $U = \text{UNIV}$]

by *auto*

lemma *continuous_map_upper_lower_semicontinuous_lte_gen*:

continuous_map X (*subtopology euclideanreal* U) $f \longleftrightarrow$
 $(\forall x \in \text{topspace } X. f x \in U) \wedge$
 $(\forall a. \text{openin } X \{x \in \text{topspace } X. f x < a\}) \wedge$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \leq a\})$

unfolding *continuous_map_upper_lower_semicontinuous_lt_gen*

by (*auto simp: closedin_def Diff_eq Compl_eq not_le Int_Collect_imp_eq*)

lemma *continuous_map_upper_lower_semicontinuous_lte*:

continuous_map X *euclideanreal* $f \longleftrightarrow$
 $(\forall a. \text{openin } X \{x \in \text{topspace } X. f x < a\}) \wedge$
 $(\forall a. \text{closedin } X \{x \in \text{topspace } X. f x \leq a\})$

using *continuous_map_upper_lower_semicontinuous_lte_gen* [**where** $U = \text{UNIV}$]

by *auto*

1.13.25 Pullback topology

Pulling back a topology by map gives again a topology. *subtopology* is a special case of this notion, pulling back by the identity. We introduce the general notion as we will need it to define the strong operator topology on the space of continuous linear operators, by pulling back the product topology on the space of all functions.

pullback_topology $A f T$ is the pullback of the topology T by the map f on the set A .

definition *pullback_topology*::('a set) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b topology) \Rightarrow ('a topology)

where *pullback_topology* $A f T = \text{topology } (\lambda S. \exists U. \text{openin } T U \wedge S = f^{-1}U \cap A)$

lemma *istopology_pullback_topology*:

istopology $(\lambda S. \exists U. \text{openin } T \ U \wedge S = f^{-1}U \cap A)$

unfolding *istopology_def* **proof** (*auto*)

fix *K* **assume** $\forall S \in K. \exists U. \text{openin } T \ U \wedge S = f^{-1}U \cap A$

then have $\exists U. \forall S \in K. \text{openin } T \ (U \cap S) \wedge S = f^{-1}(U \cap A)$

by (*rule bchoice*)

then obtain *U* **where** $U: \forall S \in K. \text{openin } T \ (U \cap S) \wedge S = f^{-1}(U \cap A)$

by *blast*

define *V* **where** $V = \bigcup_{S \in K} U \cap S$

have $\text{openin } T \ V \cup K = f^{-1}V \cap A$ **unfolding** *V_def* **using** *U* **by** *auto*

then show $\exists V. \text{openin } T \ V \wedge \bigcup K = f^{-1}V \cap A$ **by** *auto*

qed

lemma *openin_pullback_topology*:

openin $(\text{pullback_topology } A \ f \ T) \ S \longleftrightarrow (\exists U. \text{openin } T \ U \wedge S = f^{-1}U \cap A)$

unfolding *pullback_topology_def* *topology_inverse* [*OF istopology_pullback_topology*]

by *auto*

lemma *topspace_pullback_topology*:

topspace $(\text{pullback_topology } A \ f \ T) = f^{-1}(\text{topspace } T) \cap A$

by (*auto simp add: topspace_def openin_pullback_topology*)

proposition *continuous_map_pullback* [*intro*]:

assumes *continuous_map* *T1* *T2* *g*

shows *continuous_map* $(\text{pullback_topology } A \ f \ T1) \ T2 \ (g \circ f)$

unfolding *continuous_map_alt*

proof (*auto*)

fix *U*:*'b* *set* **assume** *openin* *T2* *U*

then have *openin* *T1* $(g^{-1}U \cap \text{topspace } T1)$

using *assms* **unfolding** *continuous_map_alt* **by** *auto*

have $(g \circ f)^{-1}U \cap \text{topspace } (\text{pullback_topology } A \ f \ T1) = (g \circ f)^{-1}U \cap A \cap f^{-1}(\text{topspace } T1)$

unfolding *topspace_pullback_topology* **by** *auto*

also have $\dots = f^{-1}(g^{-1}U \cap \text{topspace } T1) \cap A$

by *auto*

also have *openin* $(\text{pullback_topology } A \ f \ T1) \ (\dots)$

unfolding *openin_pullback_topology* **using** $\langle \text{openin } T1 \ (g^{-1}U \cap \text{topspace } T1) \rangle$

by *auto*

finally show *openin* $(\text{pullback_topology } A \ f \ T1) \ ((g \circ f)^{-1}U \cap \text{topspace } (\text{pullback_topology } A \ f \ T1))$

by *auto*

next

fix *x* **assume** $x \in \text{topspace } (\text{pullback_topology } A \ f \ T1)$

then have $f \ x \in \text{topspace } T1$

unfolding *topspace_pullback_topology* **by** *auto*

then show $g \ (f \ x) \in \text{topspace } T2$

using *assms* **unfolding** *continuous_map_def* **by** *auto*

qed

proposition *continuous_map_pullback'* [intro]:
assumes *continuous_map* $T1\ T2$ $(f\ o\ g)$ *topspace* $T1 \subseteq g^{-1}A$
shows *continuous_map* $T1$ $(\text{pullback_topology } A\ f\ T2)$ g
unfolding *continuous_map_alt*
proof (*auto*)
fix U **assume** *openin* $(\text{pullback_topology } A\ f\ T2)$ U
then have $\exists V.$ *openin* $T2$ $V \wedge U = f^{-1}V \cap A$
unfolding *openin_pullback_topology* **by** *auto*
then obtain V **where** *openin* $T2$ $V \wedge U = f^{-1}V \cap A$
by *blast*
then have $g^{-1}U \cap \text{topspace } T1 = g^{-1}(f^{-1}V \cap A) \cap \text{topspace } T1$
by *blast*
also have $\dots = (f\ o\ g)^{-1}V \cap (g^{-1}A \cap \text{topspace } T1)$
by *auto*
also have $\dots = (f\ o\ g)^{-1}V \cap \text{topspace } T1$
using *assms(2)* **by** *auto*
also have *openin* $T1$ (\dots)
using *assms(1)* $\langle \text{openin } T2\ V \rangle$ **by** *auto*
finally show *openin* $T1$ $(g^{-1}U \cap \text{topspace } T1)$ **by** *simp*
next
fix x **assume** $x \in \text{topspace } T1$
have $(f\ o\ g)\ x \in \text{topspace } T2$
using *assms(1)* $\langle x \in \text{topspace } T1 \rangle$ **unfolding** *continuous_map_def* **by** *auto*
then have $g\ x \in f^{-1}(\text{topspace } T2)$
unfolding *comp_def* **by** *blast*
moreover have $g\ x \in A$ **using** *assms(2)* $\langle x \in \text{topspace } T1 \rangle$ **by** *blast*
ultimately show $g\ x \in \text{topspace } (\text{pullback_topology } A\ f\ T2)$
unfolding *topspace_pullback_topology* **by** *blast*
qed

1.13.26 Proper maps (not a priori assumed continuous)

definition *proper_map*
where
proper_map $X\ Y\ f \equiv$
 $\text{closed_map } X\ Y\ f \wedge (\forall y \in \text{topspace } Y. \text{compactin } X\ \{x \in \text{topspace } X. f\ x = y\})$

lemma *proper_imp_closed_map*:
 $\text{proper_map } X\ Y\ f \implies \text{closed_map } X\ Y\ f$
by (*simp add: proper_map_def*)

lemma *proper_map_imp_subset_topspace*:
 $\text{proper_map } X\ Y\ f \implies f \in (\text{topspace } X) \rightarrow \text{topspace } Y$
by (*simp add: closed_map_imp_subset_topspace proper_map_def*)

lemma *proper_map_restriction*:
assumes *proper_map* $X\ Y\ f$ $\{x \in \text{topspace } X. f\ x \in V\} = U$


```

shows proper_map (subtopology X U) (subtopology Y V) f
proof -
  have [simp]: {x ∈ topspace X. f x ∈ V ∧ f x = y} = {x ∈ topspace X. f x = y}
    if y ∈ V for y
  using that by auto
  show ?thesis
    using assms unfolding proper_map_def using closed_map_restriction
    by (force simp: compactin_subtopology)
qed

```

```

lemma closed_injective_imp_proper_map:
  assumes f: closed_map X Y f and inj: inj_on f (topspace X)
  shows proper_map X Y f
  unfolding proper_map_def
proof (clarsimp simp: f)
  show compactin X {x ∈ topspace X. f x = y}
    if y ∈ topspace Y for y
  using inj_on_eq_iff [OF inj] that
  proof -
    have {x ∈ topspace X. f x = y} = {} ∨ (∃ a ∈ topspace X. {x ∈ topspace X. f
x = y} = {a})
    using inj_on_eq_iff [OF inj] by auto
    then show ?thesis
      using that by (metis (no_types, lifting) compactin_empty compactin_sing)
  qed
qed

```

```

lemma injective_imp_proper_eq_closed_map:
  inj_on f (topspace X) ⟹ (proper_map X Y f ⟷ closed_map X Y f)
  using closed_injective_imp_proper_map proper_imp_closed_map by blast

```

```

lemma homeomorphic_imp_proper_map:
  homeomorphic_map X Y f ⟹ proper_map X Y f
  by (simp add: closed_injective_imp_proper_map homeomorphic_eq_everything_map)

```

```

lemma compactin_proper_map_preimage:
  assumes f: proper_map X Y f and compactin Y K
  shows compactin X {x. x ∈ topspace X ∧ f x ∈ K}
proof -
  have f ∈ (topspace X) → topspace Y
    by (simp add: f proper_map_imp_subset_topspace)
  have *: ∧y. y ∈ topspace Y ⟹ compactin X {x ∈ topspace X. f x = y}
    using f by (auto simp: proper_map_def)
  show ?thesis
    unfolding compactin_def
  proof clarsimp
    show ∃F. finite F ∧ F ⊆ U ∧ {x ∈ topspace X. f x ∈ K} ⊆ ⋃F
      if U: ∀U ∈ U. openin X U and sub: {x ∈ topspace X. f x ∈ K} ⊆ ⋃U
      for U

```

```

proof –
  have  $\forall y \in K. \exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f x = y\} \subseteq \bigcup \mathcal{V}$ 
  proof
    fix  $y$ 
    assume  $y \in K$ 
    then have  $\text{compactin } X \{x \in \text{topspace } X. f x = y\}$ 
      by (metis *  $\langle \text{compactin } Y K \rangle \text{compactin\_subspace subsetD}$ )
    with  $\langle y \in K \rangle$  show  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f x = y\} \subseteq$ 
 $\bigcup \mathcal{V}$ 
      unfolding compactin_def using  $\mathcal{U}$  sub by fastforce
    qed
  then obtain  $\mathcal{V}$  where  $\mathcal{V}: \bigwedge y. y \in K \implies \text{finite } (\mathcal{V} y) \wedge \mathcal{V} y \subseteq \mathcal{U} \wedge \{x \in$ 
 $\text{topspace } X. f x = y\} \subseteq \bigcup (\mathcal{V} y)$ 
    by (metis (full_types))
  define  $F$  where  $F \equiv \lambda y. \text{topspace } Y - f^{-1}(\text{topspace } X - \bigcup (\mathcal{V} y))$ 
  have  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq F^{-1} K \wedge K \subseteq \bigcup \mathcal{F}$ 
  proof (rule compactinD [OF  $\langle \text{compactin } Y K \rangle$ ])
    have  $\bigwedge x. x \in K \implies \text{closedin } Y (f^{-1}(\text{topspace } X - \bigcup (\mathcal{V} x)))$ 
      using  $f$  unfolding proper_map_def closed_map_def
      by (meson  $\mathcal{U}$   $\mathcal{V}$  openin_Union openin_closedin_eq subsetD)
    then show  $\text{openin } Y U$  if  $U \in F^{-1} K$  for  $U$ 
      using that by (auto simp: F_def)
    show  $K \subseteq \bigcup (F^{-1} K)$ 
      using  $\mathcal{V}$   $\langle \text{compactin } Y K \rangle$  unfolding  $F\_def$  compactin_def by fastforce
    qed
  then obtain  $J$  where  $\text{finite } J \wedge J \subseteq K$  and  $J: K \subseteq \bigcup (F^{-1} J)$ 
    by (auto simp: ex_finite_subset_image)
  show ?thesis
    unfolding  $F\_def$ 
  proof (intro exI conjI)
    show  $\text{finite } (\bigcup (\mathcal{V}^{-1} J))$ 
      using  $\mathcal{V} \langle J \subseteq K \rangle \langle \text{finite } J \rangle$  by blast
    show  $\bigcup (\mathcal{V}^{-1} J) \subseteq \mathcal{U}$ 
      using  $\mathcal{V} \langle J \subseteq K \rangle$  by blast
    show  $\{x \in \text{topspace } X. f x \in K\} \subseteq \bigcup (\bigcup (\mathcal{V}^{-1} J))$ 
      using  $J \langle J \subseteq K \rangle$  unfolding  $F\_def$  by auto
    qed
  qed
qed
qed

```

lemma *compact_space_proper_map_preimage*:

assumes $f: \text{proper_map } X Y$ **and** $\text{fim}: f^{-1}(\text{topspace } X) = \text{topspace } Y$ **and** *compact_space Y*

shows *compact_space X*

proof –

have $\text{eq}: \text{topspace } X = \{x \in \text{topspace } X. f x \in \text{topspace } Y\}$
using *fim* **by** *blast*

```

moreover have compactin Y (topspace Y)
  using <compact_space Y> compact_space_def by auto
ultimately show ?thesis
  unfolding compact_space_def
  using eq f compactin_proper_map_preimage by fastforce
qed

```

```

lemma proper_map_alt:
  proper_map X Y f  $\longleftrightarrow$ 
  closed_map X Y f  $\wedge$  ( $\forall K$ . compactin Y K  $\longrightarrow$  compactin X {x. x  $\in$  topspace
  X  $\wedge$  f x  $\in$  K})
  proof (intro iffI conjI allI impI)
  show compactin X {x  $\in$  topspace X. f x  $\in$  K}
    if proper_map X Y f and compactin Y K for K
    using that by (simp add: compactin_proper_map_preimage)
  show proper_map X Y f
    if f: closed_map X Y f  $\wedge$  ( $\forall K$ . compactin Y K  $\longrightarrow$  compactin X {x  $\in$  topspace
  X. f x  $\in$  K})
  proof -
    have compactin X {x  $\in$  topspace X. f x = y} if y  $\in$  topspace Y for y
    proof -
      have compactin X {x  $\in$  topspace X. f x  $\in$  {y}}
        using f compactin_sing that by fastforce
      then show ?thesis
        by auto
    qed
  with f show ?thesis
    by (auto simp: proper_map_def)
  qed
qed (simp add: proper_imp_closed_map)

```

```

lemma proper_map_on_empty [simp]: proper_map trivial_topology Y f
  by (auto simp: proper_map_def closed_map_on_empty)

```

```

lemma proper_map_id [simp]:
  proper_map X X id
proof (clarsimp simp: proper_map_alt closed_map_id)
  fix K
  assume K: compactin X K
  then have {a  $\in$  topspace X. a  $\in$  K} = K
    by (simp add: compactin_subspace subset_antisym subset_iff)
  then show compactin X {a  $\in$  topspace X. a  $\in$  K}
    using K by auto
qed

```

```

lemma proper_map_compose:
  assumes proper_map X Y f proper_map Y Z g
  shows proper_map X Z (g  $\circ$  f)
proof -

```

```

have closed_map X Y f and f:  $\bigwedge K. \text{compactin } Y K \implies \text{compactin } X \{x \in \text{topspace } X. f x \in K\}$ 
and closed_map Y Z g and g:  $\bigwedge K. \text{compactin } Z K \implies \text{compactin } Y \{x \in \text{topspace } Y. g x \in K\}$ 
using assms by (auto simp: proper_map_alt)
show ?thesis
unfolding proper_map_alt
proof (intro conjI allI impI)
show closed_map X Z (g ∘ f)
using ⟨closed_map X Y f⟩ ⟨closed_map Y Z g⟩ closed_map_compose by
blast
have  $\{x \in \text{topspace } X. g (f x) \in K\} = \{x \in \text{topspace } X. f x \in \{b \in \text{topspace } Y. g b \in K\}\}$  for K
using ⟨closed_map X Y f⟩ closed_map_imp_subset_topspace by blast
then show compactin X  $\{x \in \text{topspace } X. (g \circ f) x \in K\}$ 
if compactin Z K for K
using f [OF g [OF that]] by auto
qed
qed

```

lemma *proper_map_const*:

$\text{proper_map } X Y (\lambda x. c) \iff \text{compact_space } X \wedge (X = \text{trivial_topology} \vee \text{closedin } Y \{c\})$

proof (*cases* X = *trivial_topology*)

case *True*

then show ?thesis

by *simp*

next

case *False*

have *: *compactin* X $\{x \in \text{topspace } X. c = y\}$ **if** *compact_space* X **for** y

using *that* **unfolding** *compact_space_def*

by (*metis* (*mono_tags*, *lifting*) *compactin_empty* *empty_subsetI* *mem_Collect_eq* *subsetI* *subset_antisym*)

then show ?thesis

using *closed_compactin* *closedin_subset*

by (*force simp: False proper_map_def closed_map_const compact_space_def*)

qed

lemma *proper_map_inclusion*:

$S \subseteq \text{topspace } X \implies \text{proper_map } (\text{subtopology } X S) X \text{id} \iff \text{closedin } X S \wedge (\forall k. \text{compactin } X k \longrightarrow \text{compactin } X (S \cap k))$

by (*metis* *closed_Int_compactin* *closed_map_inclusion_eq* *inf.absorb_iff2* *inj_on_id* *injective_imp_proper_eq_closed_map*)

lemma *proper_map_into_subtopology*:

$\llbracket \text{proper_map } X Y f; f \in \text{topspace } X \rightarrow C \rrbracket \implies \text{proper_map } X (\text{subtopology } Y C) f$

by (*simp add: closed_map_into_subtopology compactin_subtopology proper_map_alt*)

```

lemma proper_map_from_composition_left:
  assumes gf: proper_map X Z (g o f) and contf: continuous_map X Y f and
  fim: f ' topspace X = topspace Y
  shows proper_map Y Z g
  unfolding proper_map_def
proof (intro strip conjI)
  show closed_map Y Z g
  by (meson closed_map_from_composition_left gf contf fim proper_imp_closed_map)
  fix z assume z ∈ topspace Z
  have eq: {y ∈ topspace Y. g y = z} = f ' {x ∈ topspace X. (g o f) x = z}
  using fim by force
  show compactin Y {x ∈ topspace Y. g x = z}
  unfolding eq
  proof (rule image_compactin [OF _ contf])
    show compactin X {x ∈ topspace X. (g o f) x = z}
    using ⟨z ∈ topspace Z⟩ gf proper_map_def by fastforce
  qed
qed

```

```

lemma proper_map_from_composition_right_inj:
  assumes gf: proper_map X Z (g o f) and fim: f ∈ topspace X → topspace Y
  and contf: continuous_map Y Z g and inj: inj_on g (topspace Y)
  shows proper_map X Y f
  unfolding proper_map_def
proof (intro strip conjI)
  show closed_map X Y f
  by (meson closed_map_from_composition_right assms proper_imp_closed_map)
  fix y
  assume y ∈ topspace Y
  with fim inj have eq: {x ∈ topspace X. f x = y} = {x ∈ topspace X. (g o f) x
  = g y}
  by (auto simp: Pi_iff inj_onD)
  show compactin X {x ∈ topspace X. f x = y}
  using contf gf ⟨y ∈ topspace Y⟩
  unfolding eq continuous_map_def proper_map_def
  by blast
qed

```

1.13.27 Perfect maps (proper, continuous and surjective)

```

definition perfect_map
  where perfect_map X Y f ≡ continuous_map X Y f ∧ proper_map X Y f ∧ f
  ' (topspace X) = topspace Y

```

```

lemma homeomorphic_imp_perfect_map:
  homeomorphic_map X Y f ⇒ perfect_map X Y f
  by (simp add: homeomorphic_eq_everything_map homeomorphic_imp_proper_map
  perfect_map_def)

```

lemma *perfect_imp_quotient_map*:

perfect_map $X\ Y\ f \implies$ *quotient_map* $X\ Y\ f$

by (*simp* *add*: *continuous_closed_imp_quotient_map* *perfect_map_def* *proper_map_def*)

lemma *homeomorphic_eq_injective_perfect_map*:

homeomorphic_map $X\ Y\ f \iff$ *perfect_map* $X\ Y\ f \wedge$ *inj_on* f (*topspace* X)

using *homeomorphic_imp_perfect_map* *homeomorphic_map_def* *perfect_imp_quotient_map*
by *blast*

lemma *perfect_injective_eq_homeomorphic_map*:

perfect_map $X\ Y\ f \wedge$ *inj_on* f (*topspace* X) \iff *homeomorphic_map* $X\ Y\ f$

by (*simp* *add*: *homeomorphic_eq_injective_perfect_map*)

lemma *perfect_map_id* [*simp*]: *perfect_map* $X\ X\ id$

by (*simp* *add*: *homeomorphic_imp_perfect_map*)

lemma *perfect_map_compose*:

\llbracket *perfect_map* $X\ Y\ f$; *perfect_map* $Y\ Z\ g$ $\rrbracket \implies$ *perfect_map* $X\ Z\ (g \circ f)$

by (*meson* *continuous_map_compose* *perfect_imp_quotient_map* *perfect_map_def* *proper_map_compose* *quotient_map_compose_eq* *quotient_map_def*)

lemma *perfect_imp_continuous_map*:

perfect_map $X\ Y\ f \implies$ *continuous_map* $X\ Y\ f$

using *perfect_map_def* **by** *blast*

lemma *perfect_imp_closed_map*:

perfect_map $X\ Y\ f \implies$ *closed_map* $X\ Y\ f$

by (*simp* *add*: *perfect_map_def* *proper_map_def*)

lemma *perfect_imp_proper_map*:

perfect_map $X\ Y\ f \implies$ *proper_map* $X\ Y\ f$

by (*simp* *add*: *perfect_map_def*)

lemma *perfect_imp_surjective_map*:

perfect_map $X\ Y\ f \implies$ $f'(\text{topspace } X) = \text{topspace } Y$

by (*simp* *add*: *perfect_map_def*)

lemma *perfect_map_from_composition_left*:

assumes *perfect_map* $X\ Z\ (g \circ f)$ **and** *continuous_map* $X\ Y\ f$

and *continuous_map* $Y\ Z\ g$ **and** $f' \text{topspace } X = \text{topspace } Y$

shows *perfect_map* $Y\ Z\ g$

using *assms* **unfolding** *perfect_map_def*

by (*metis* *image_comp_proper_map_from_composition_left*)

end

1.14 F -Sigma and G -Delta sets in a Topological Space

An F -sigma set is a countable union of closed sets; a G -delta set is the dual.

theory *FSigma*

imports *Abstract_Topology*

begin

definition *fsigma_in*

where $fsigma_in\ X \equiv countable_union_of_closedin\ X$

definition *gdelta_in*

where $gdelta_in\ X \equiv (countable_intersection_of_openin\ X)\ relative_to\ topspace\ X$

lemma *fsigma_in_ascending*:

$fsigma_in\ X\ S \longleftrightarrow (\exists C. (\forall n. closedin\ X\ (C\ n)) \wedge (\forall n. C\ n \subseteq C(Suc\ n)) \wedge \bigcup (range\ C) = S)$

unfolding *fsigma_in_def*

by (*metis* *closedin_Un_countable_union_of_ascending_closedin_empty*)

lemma *gdelta_in_alt*:

$gdelta_in\ X\ S \longleftrightarrow S \subseteq topspace\ X \wedge (countable_intersection_of_openin\ X)\ S$

proof –

have $(countable_intersection_of_openin\ X)\ (topspace\ X)$

by (*simp* *add*: *countable_intersection_of_inc*)

then show *?thesis*

unfolding *gdelta_in_def*

by (*metis* *countable_intersection_of_inter_relative_to_def_relative_to_imp_subset_relative_to_subset_inc*)

qed

lemma *fsigma_in_subset*: $fsigma_in\ X\ S \Longrightarrow S \subseteq topspace\ X$

using *closedin_subset* **by** (*fastforce* *simp*: *fsigma_in_def* *union_of_def* *subset_iff*)

lemma *gdelta_in_subset*: $gdelta_in\ X\ S \Longrightarrow S \subseteq topspace\ X$

by (*simp* *add*: *gdelta_in_alt*)

lemma *closed_imp_fsigma_in*: $closedin\ X\ S \Longrightarrow fsigma_in\ X\ S$

by (*simp* *add*: *countable_union_of_inc_fsigma_in_def*)

lemma *open_imp_gdelta_in*: $openin\ X\ S \Longrightarrow gdelta_in\ X\ S$

by (*simp* *add*: *countable_intersection_of_inc_gdelta_in_alt* *openin_subset*)

lemma *fsigma_in_empty* [*iff*]: $fsigma_in\ X\ \{\}$

by (*simp* *add*: *closed_imp_fsigma_in*)

lemma *gdelta_in_empty* [iff]: *gdelta_in* X $\{\}$
 by (*simp* add: *open_imp_gdelta_in*)

lemma *fsigma_in_topspace* [iff]: *fsigma_in* X (*topspace* X)
 by (*simp* add: *closed_imp_fsigma_in*)

lemma *gdelta_in_topspace* [iff]: *gdelta_in* X (*topspace* X)
 by (*simp* add: *open_imp_gdelta_in*)

lemma *fsigma_in_Union*:
 $\llbracket \text{countable } T; \bigwedge S. S \in T \implies \text{fsigma_in } X S \rrbracket \implies \text{fsigma_in } X (\bigcup T)$
 by (*simp* add: *countable_union_of_Union fsigma_in_def*)

lemma *fsigma_in_Un*:
 $\llbracket \text{fsigma_in } X S; \text{fsigma_in } X T \rrbracket \implies \text{fsigma_in } X (S \cup T)$
 by (*simp* add: *countable_union_of_Un fsigma_in_def*)

lemma *fsigma_in_Int*:
 $\llbracket \text{fsigma_in } X S; \text{fsigma_in } X T \rrbracket \implies \text{fsigma_in } X (S \cap T)$
 by (*simp* add: *closedin_Int countable_union_of_Int fsigma_in_def*)

lemma *gdelta_in_Inter*:
 $\llbracket \text{countable } T; T \neq \{\}; \bigwedge S. S \in T \implies \text{gdelta_in } X S \rrbracket \implies \text{gdelta_in } X (\bigcap T)$
 by (*simp* add: *Inf_less_eq countable_intersection_of_Inter gdelta_in_alt*)

lemma *gdelta_in_Int*:
 $\llbracket \text{gdelta_in } X S; \text{gdelta_in } X T \rrbracket \implies \text{gdelta_in } X (S \cap T)$
 by (*simp* add: *countable_intersection_of_inter gdelta_in_alt le_infI2*)

lemma *gdelta_in_Un*:
 $\llbracket \text{gdelta_in } X S; \text{gdelta_in } X T \rrbracket \implies \text{gdelta_in } X (S \cup T)$
 by (*simp* add: *countable_intersection_of_union gdelta_in_alt openin_Un*)

lemma *fsigma_in_diff*:
 assumes *fsigma_in* X S *gdelta_in* X T
 shows *fsigma_in* X $(S - T)$
proof –
 have [*simp*]: $S - (S \cap T) = S - T$ for $S T :: 'a$ set
 by *auto*
 have [*simp*]: $\text{topspace } X - \bigcap \mathcal{T} = (\bigcup T \in \mathcal{T}. \text{topspace } X - T)$ for \mathcal{T}
 by *auto*
 have $\bigwedge \mathcal{T}. \llbracket \text{countable } \mathcal{T}; \mathcal{T} \subseteq \text{Collect } (\text{openin } X) \rrbracket \implies$
 $\text{countable_union_of_closedin } X (\bigcup ((-) (\text{topspace } X) ' \mathcal{T}))$
 by (*metis* *Ball_Collect countable_union_of_UN countable_union_of_inc openin_closedin_eq*)
 then have $\forall S. \text{gdelta_in } X S \longrightarrow \text{fsigma_in } X (\text{topspace } X - S)$
 by (*simp* add: *fsigma_in_def gdelta_in_def all_relative_to_all_intersection_of*
del: UN_simps)
 then show ?thesis

by (metis Diff_Int2 Diff_Int_distrib2 assms fsigma_in_Int fsigma_in_subset inf.absorb_iff2)

qed

lemma gdelta_in_diff:

assumes gdelta_in X S fsigma_in X T

shows gdelta_in X (S - T)

proof -

have [simp]: $\text{topspace } X - \bigcup \mathcal{T} = \text{topspace } X \cap (\bigcap T \in \mathcal{T}. \text{topspace } X - T)$ for \mathcal{T}

by auto

have $\bigwedge \mathcal{T}. \llbracket \text{countable } \mathcal{T}; \mathcal{T} \subseteq \text{Collect } (\text{closedin } X) \rrbracket$

$\implies (\text{countable_intersection_of_openin } X \text{ relative_to } \text{topspace } X)$

$(\text{topspace } X \cap \bigcap ((-) (\text{topspace } X) ' \mathcal{T}))$

by (metis Ball_Collect closedin_def countable_intersection_of_INT countable_intersection_of_inc relative_to_inc)

then have $\forall S. \text{fsigma_in } X S \implies \text{gdelta_in } X (\text{topspace } X - S)$

by (simp add: fsigma_in_def gdelta_in_def all_union_of_del: INT_simps)

then show ?thesis

by (metis Diff_Int2 Diff_Int_distrib2 assms gdelta_in_Int gdelta_in_alt inf.orderE inf_commute)

qed

lemma gdelta_in_fsigma_in:

$\text{gdelta_in } X S \iff S \subseteq \text{topspace } X \wedge \text{fsigma_in } X (\text{topspace } X - S)$

by (metis double_diff fsigma_in_diff fsigma_in_topspace gdelta_in_alt gdelta_in_diff gdelta_in_topspace)

lemma fsigma_in_gdelta_in:

$\text{fsigma_in } X S \iff S \subseteq \text{topspace } X \wedge \text{gdelta_in } X (\text{topspace } X - S)$

by (metis Diff_Diff_Int fsigma_in_subset gdelta_in_fsigma_in inf.absorb_iff2)

lemma gdelta_in_descending:

$\text{gdelta_in } X S \iff (\exists C. (\forall n. \text{openin } X (C n)) \wedge (\forall n. C(\text{Suc } n) \subseteq C n) \wedge \bigcap (\text{range } C) = S)$ (is ?lhs=?rhs)

proof

assume ?lhs

then obtain C where $C: S \subseteq \text{topspace } X \wedge \bigwedge n. \text{closedin } X (C n)$

$\bigwedge n. C n \subseteq C(\text{Suc } n) \cup (\text{range } C) = \text{topspace } X - S$

by (meson fsigma_in_ascending gdelta_in_fsigma_in)

define D where $D \equiv \lambda n. \text{topspace } X - C n$

have $\bigwedge n. \text{openin } X (D n) \wedge D(\text{Suc } n) \subseteq D n$

by (simp add: Diff_mono C D_def openin_diff)

moreover have $\bigcap (\text{range } D) = S$

by (simp add: Diff_Diff_Int Int_absorb1 C D_def)

ultimately show ?rhs

by metis

next

assume ?rhs

then obtain C **where** $S \subseteq \text{topspace } X$
and $C: \bigwedge n. \text{openin } X (C n) \wedge n. C(\text{Suc } n) \subseteq C n \cap (\text{range } C) = S$
using openin_subset **by** fastforce
define D **where** $D \equiv \lambda n. \text{topspace } X - C n$
have $\bigwedge n. \text{closedin } X (D n) \wedge D n \subseteq D(\text{Suc } n)$
by $(\text{simp add: Diff_mono } C \text{ closedin_diff } D_def)$
moreover have $\bigcup (\text{range } D) = \text{topspace } X - S$
using $C D_def$ **by** blast
ultimately show $?lhs$
by $(\text{metis } \langle S \subseteq \text{topspace } X \rangle \text{fsigma_in_ascending } g\text{delta_in_fsigma_in})$
qed

lemma $\text{homeomorphic_map_fsigmaness}$:

assumes $f: \text{homeomorphic_map } X Y$ **and** $U \subseteq \text{topspace } X$
shows $\text{fsigma_in } Y (f' U) \longleftrightarrow \text{fsigma_in } X U$ **(is** $?lhs=?rhs)$
proof –
obtain g **where** $g: \text{homeomorphic_maps } X Y$ **and** $g: \text{homeomorphic_map } Y X$
and $1: (\forall x \in \text{topspace } X. g(f x) = x)$ **and** $2: (\forall y \in \text{topspace } Y. f(g y) = y)$
using $\text{assms homeomorphic_map_maps}$ **by** $(\text{metis homeomorphic_maps_map})$
show $?thesis$
proof
assume $?lhs$
then obtain \mathcal{V} **where** $\text{countable } \mathcal{V}$ **and** $\mathcal{V}: \mathcal{V} \subseteq \text{Collect } (\text{closedin } Y) \cup \mathcal{V} = f'U$
by $(\text{force simp: fsigma_in_def union_of_def})$
define \mathcal{U} **where** $\mathcal{U} \equiv \text{image } (\text{image } g) \mathcal{V}$
have $\text{countable } \mathcal{U}$
using $\mathcal{U_def} \langle \text{countable } \mathcal{V} \rangle$ **by** blast
moreover have $\mathcal{U} \subseteq \text{Collect } (\text{closedin } X)$
using $f g \text{homeomorphic_map_closedness_eq } \mathcal{V}$ **unfolding** $\mathcal{U_def}$ **by** blast
moreover have $\bigcup \mathcal{U} \subseteq U$
unfolding $\mathcal{U_def}$ **by** $(\text{smt } (\text{verit}) \text{assms } 1 \mathcal{V} \text{image_Union image_iff in_mono subsetI})$
moreover have $U \subseteq \bigcup \mathcal{U}$
unfolding $\mathcal{U_def}$ **using** $\text{assms } \mathcal{V}$
by $(\text{smt } (\text{verit}, \text{del_insts}) 1 \text{imageI image_Union subset_iff})$
ultimately show $?rhs$
by $(\text{metis fsigma_in_def subset_antisym union_of_def})$
next
assume $?rhs$
then obtain \mathcal{V} **where** $\text{countable } \mathcal{V}$ **and** $\mathcal{V}: \mathcal{V} \subseteq \text{Collect } (\text{closedin } X) \cup \mathcal{V} = U$
by $(\text{auto simp: fsigma_in_def union_of_def})$
define \mathcal{U} **where** $\mathcal{U} \equiv \text{image } (\text{image } f) \mathcal{V}$
have $\text{countable } \mathcal{U}$
using $\mathcal{U_def} \langle \text{countable } \mathcal{V} \rangle$ **by** blast
moreover have $\mathcal{U} \subseteq \text{Collect } (\text{closedin } Y)$
using $f g \text{homeomorphic_map_closedness_eq } \mathcal{V}$ **unfolding** $\mathcal{U_def}$ **by** blast
moreover have $\bigcup \mathcal{U} = f'U$

```

    unfolding  $\mathcal{U}$ _def using  $\mathcal{V}$  by blast
  ultimately show ?lhs
    by (metis fsigma_in_def union_of_def)
qed
qed

```

```

lemma homeomorphic_map_fsigmaness_eq:
  homeomorphic_map  $X$   $Y$   $f$ 
   $\implies$  (fsigma_in  $X$   $U \iff U \subseteq \text{topspace } X \wedge \text{fsigma\_in } Y (f \text{ ` } U)$ )
  by (metis fsigma_in_subset homeomorphic_map_fsigmaness)

```

```

lemma homeomorphic_map_gdeltaness:
  assumes homeomorphic_map  $X$   $Y$   $f$   $U \subseteq \text{topspace } X$ 
  shows gdelta_in  $Y$  (f `  $U$ )  $\iff$  gdelta_in  $X$   $U$ 
proof -
  have  $\text{topspace } Y - f \text{ ` } U = f \text{ ` } (\text{topspace } X - U)$ 
  by (metis (no_types, lifting) Diff_subset assms homeomorphic_eq_everything_map
  inj_on_image_set_diff)
  then show ?thesis
    using assms homeomorphic_imp_surjective_map
    by (fastforce simp: gdelta_in_fsigma_in homeomorphic_map_fsigmaness_eq)
qed

```

```

lemma homeomorphic_map_gdeltaness_eq:
  homeomorphic_map  $X$   $Y$   $f$ 
   $\implies$  gdelta_in  $X$   $U \iff U \subseteq \text{topspace } X \wedge \text{gdelta\_in } Y (f \text{ ` } U)$ 
  by (meson gdelta_in_subset homeomorphic_map_gdeltaness)

```

```

lemma fsigma_in_relative_to:
  (fsigma_in  $X$  relative_to  $S$ ) = fsigma_in (subtopology  $X$   $S$ )
  by (simp add: fsigma_in_def countable_union_of_relative_to closedin_relative_to)

```

```

lemma fsigma_in_subtopology:
  fsigma_in (subtopology  $X$   $U$ )  $S \iff (\exists T. \text{fsigma\_in } X T \wedge S = T \cap U)$ 
  by (metis fsigma_in_relative_to inf_commute relative_to_def)

```

```

lemma gdelta_in_relative_to:
  (gdelta_in  $X$  relative_to  $S$ ) = gdelta_in (subtopology  $X$   $S$ )
  apply (simp add: gdelta_in_def)
  by (metis countable_intersection_of_relative_to openin_relative_to subtopology_restrict)

```

```

lemma gdelta_in_subtopology:
  gdelta_in (subtopology  $X$   $U$ )  $S \iff (\exists T. \text{gdelta\_in } X T \wedge S = T \cap U)$ 
  by (metis gdelta_in_relative_to inf_commute relative_to_def)

```

```

lemma fsigma_in_fsigma_subtopology:
  fsigma_in  $X$   $S \implies$  (fsigma_in (subtopology  $X$   $S$ )  $T \iff \text{fsigma\_in } X T \wedge T \subseteq S$ )

```

by (metis fsigma_in_Int fsigma_in_gdelta_in fsigma_in_subtopology inf.orderE topspace_subtopology_subset)

lemma *gdelta_in_gdelta_subtopology*:

$gdelta_in\ X\ S \implies (gdelta_in\ (subtopology\ X\ S)\ T \longleftrightarrow gdelta_in\ X\ T \wedge T \subseteq S)$

by (metis gdelta_in_Int gdelta_in_subset gdelta_in_subtopology inf.orderE topspace_subtopology_sub)

end

1.15 Disjoint sum of arbitrarily many spaces

theory *Sum_Topology*

imports *Abstract_Topology*

begin

definition *sum_topology* :: ('a \Rightarrow 'b topology) \Rightarrow 'a set \Rightarrow ('a \times 'b) topology **where**

sum_topology X I \equiv

topology ($\lambda U. U \subseteq \text{Sigma } I\ (\text{topspace } \circ X) \wedge (\forall i \in I. \text{openin } (X\ i)\ \{x. (i, x) \in U\})$)

lemma *is_sum_topology*: *istopology* ($\lambda U. U \subseteq \text{Sigma } I\ (\text{topspace } \circ X) \wedge (\forall i \in I. \text{openin } (X\ i)\ \{x. (i, x) \in U\})$)

proof –

have 1: $\{x. (i, x) \in S \cap T\} = \{x. (i, x) \in S\} \cap \{x::'b. (i, x) \in T\}$ **for** S T **and** $i::'a$

by *auto*

have 2: $\{x. (i, x) \in \bigcup K\} = (\bigcup K \in \mathcal{K}. \{x::'b. (i, x) \in K\})$ **for** K **and** $i::'a$

by *auto*

show ?thesis

unfolding *istopology_def* 1 2 **by** *blast*

qed

lemma *openin_sum_topology*:

$\text{openin } (sum_topology\ X\ I)\ U \longleftrightarrow$

$U \subseteq \text{Sigma } I\ (\text{topspace } \circ X) \wedge (\forall i \in I. \text{openin } (X\ i)\ \{x. (i, x) \in U\})$

by (auto simp: *sum_topology_def* *is_sum_topology*)

lemma *openin_disjoint_union*:

$\text{openin } (sum_topology\ X\ I)\ (\text{Sigma } I\ U) \longleftrightarrow (\forall i \in I. \text{openin } (X\ i)\ (U\ i))$

using *openin_subset* **by** (force simp: *openin_sum_topology*)

lemma *topspace_sum_topology* [*simp*]:

$\text{topspace } (sum_topology\ X\ I) = \text{Sigma } I\ (\text{topspace } \circ X)$

by (metis *comp_apply* *openin_disjoint_union* *openin_subset* *openin_sum_topology* *openin_topspace* *subset_antisym*)

lemma *openin_sum_topology_alt*:

$\text{openin } (\text{sum_topology } X I) U \longleftrightarrow (\exists T. U = \text{Sigma } I T \wedge (\forall i \in I. \text{openin } (X i) (T i)))$

by (*bestsimp simp: openin_sum_topology dest: openin_subset*)

lemma forall_openin_sum_topology:

$(\forall U. \text{openin } (\text{sum_topology } X I) U \longrightarrow P U) \longleftrightarrow (\forall T. (\forall i \in I. \text{openin } (X i) (T i)) \longrightarrow P(\text{Sigma } I T))$

by (*auto simp: openin_sum_topology_alt*)

lemma exists_openin_sum_topology:

$(\exists U. \text{openin } (\text{sum_topology } X I) U \wedge P U) \longleftrightarrow (\exists T. (\forall i \in I. \text{openin } (X i) (T i)) \wedge P(\text{Sigma } I T))$

by (*auto simp: openin_sum_topology_alt*)

lemma closedin_sum_topology:

$\text{closedin } (\text{sum_topology } X I) U \longleftrightarrow U \subseteq \text{Sigma } I (\text{topspace } \circ X) \wedge (\forall i \in I. \text{closedin } (X i) \{x. (i, x) \in U\})$

(**is** ?lhs = ?rhs)

proof

assume *L: ?lhs*

then have $U \subseteq \text{Sigma } I (\text{topspace } \circ X)$

using *closedin_subset* **by** *fastforce*

then have $\forall i \in I. \{x. (i, x) \in U\} \subseteq \text{topspace } (X i)$

by *fastforce*

moreover have $\text{openin } (X i) (\text{topspace } (X i) - \{x. (i, x) \in U\})$ **if** $i \in I$ **for** i

apply (*subst openin_subopen*)

using *L* **that** **unfolding** *closedin_def openin_sum_topology*

by (*bestsimp simp: o_def subset_iff*)

ultimately show ?rhs

by (*simp add: U closedin_def*)

next

assume *R: ?rhs*

then have $\text{openin } (X i) \{x. (i, x) \in \text{topspace } (\text{sum_topology } X I) - U\}$ **if** $i \in I$ **for** i

apply (*subst openin_subopen*)

using *that* **unfolding** *closedin_def openin_sum_topology*

by (*bestsimp simp: o_def subset_iff*)

then show ?lhs

by (*simp add: R closedin_def openin_sum_topology*)

qed

lemma closedin_disjoint_union:

$\text{closedin } (\text{sum_topology } X I) (\text{Sigma } I U) \longleftrightarrow (\forall i \in I. \text{closedin } (X i) (U i))$

using *closedin_subset* **by** (*force simp: closedin_sum_topology*)

lemma closedin_sum_topology_alt:

$\text{closedin } (\text{sum_topology } X I) U \longleftrightarrow (\exists T. U = \text{Sigma } I T \wedge (\forall i \in I. \text{closedin } (X i) (T i)))$

using *closedin_subset* **unfolding** *closedin_sum_topology* **by** *auto blast*

lemma forall_closedin_sum_topology:

$(\forall U. \text{closedin}(\text{sum_topology } X I) U \longrightarrow P U) \longleftrightarrow$
 $(\forall t. (\forall i \in I. \text{closedin}(X i) (t i)) \longrightarrow P(\text{Sigma } I t))$
by (metis closedin_sum_topology_alt)

lemma exists_closedin_sum_topology:

$(\exists U. \text{closedin}(\text{sum_topology } X I) U \wedge P U) \longleftrightarrow$
 $(\exists T. (\forall i \in I. \text{closedin}(X i) (T i)) \wedge P(\text{Sigma } I T))$
by (simp add: closedin_sum_topology_alt) blast

lemma open_map_component_injection:

$i \in I \implies \text{open_map}(X i) (\text{sum_topology } X I) (\lambda x. (i, x))$
unfolding open_map_def openin_sum_topology
using openin_subset openin_subopen **by** (force simp: image_iff)

lemma closed_map_component_injection:

assumes $i \in I$
shows $\text{closed_map}(X i) (\text{sum_topology } X I) (\lambda x. (i, x))$
proof –
have $\text{closedin}(X j) \{x. j = i \wedge x \in U\}$
if $\bigwedge U S. \text{closedin } U S \implies S \subseteq \text{topspace } U$ **and** $\text{closedin}(X i) U$ **and** $j \in I$
for $U j$
using that **by** (cases $j=i$) auto
then show ?thesis
using closedin_subset assms **by** (force simp: closed_map_def closedin_sum_topology image_iff)
qed

lemma continuous_map_component_injection:

$i \in I \implies \text{continuous_map}(X i) (\text{sum_topology } X I) (\lambda x. (i, x))$
apply (clarsimp simp: continuous_map_def openin_sum_topology)
by (smt (verit, best) Collect_cong mem_Collect_eq openin_subset subsetD)

lemma subtopology_sum_topology:

$\text{subtopology}(\text{sum_topology } X I) (\text{Sigma } I S) =$
 $\text{sum_topology}(\lambda i. \text{subtopology}(X i) (S i)) I$
unfolding topology_eq
proof (intro strip iffI)
fix T
assume $*$: $\text{openin}(\text{subtopology}(\text{sum_topology } X I) (\text{Sigma } I S)) T$
then obtain U **where** $U: \forall i \in I. \text{openin}(X i) (U i) \wedge T = \text{Sigma } I U \cap \text{Sigma } I S$
by (auto simp: openin_subtopology openin_sum_topology_alt)
have $T = (\text{SIGMA } i:I. U i \cap S i)$
proof
show $T \subseteq (\text{SIGMA } i:I. U i \cap S i)$
by (metis * SigmaE Sigma_Int_distrib2 U openin_imp_subset subset_iff)
show $(\text{SIGMA } i:I. U i \cap S i) \subseteq T$

```

    using U by fastforce
  qed
  then show openin (sum_topology ( $\lambda i.$  subtopology (X i) (S i)) I) T
    by (simp add: U openin_disjoint_union openin_subtopology_Int)
next
  fix T
  assume *: openin (sum_topology ( $\lambda i.$  subtopology (X i) (S i)) I) T
  then obtain U where  $\forall i \in I. \exists T. \text{openin } (X i) T \wedge U i = T \cap S i$  and Teq:
    T = Sigma I U
    by (auto simp: openin_subtopology openin_sum_topology_alt)
  then obtain B where  $\bigwedge i. i \in I \implies \text{openin } (X i) (B i) \wedge U i = B i \cap S i$ 
    by metis
  then show openin (subtopology (sum_topology X I) (Sigma I S)) T
    by (auto simp: openin_subtopology openin_sum_topology_alt Teq)
qed

lemma embedding_map_component_injection:
  i  $\in$  I  $\implies$  embedding_map (X i) (sum_topology X I) ( $\lambda x. (i, x)$ )
  by (metis injective_open_imp_embedding_map continuous_map_component_injection
    open_map_component_injection inj_onI prod.inject)

lemma topological_property_of_sum_component:
  assumes major: P (sum_topology X I)
    and minor:  $\bigwedge X S. \llbracket P X; \text{closedin } X S; \text{openin } X S \rrbracket \implies P(\text{subtopology } X S)$ 
    and PQ:  $\bigwedge X Y. X \text{ homeomorphic\_space } Y \implies (P X \longleftrightarrow Q Y)$ 
  shows ( $\forall i \in I. Q(X i)$ )
proof -
  have Q(X i) if i  $\in$  I for i
  proof -
    have P(subtopology (sum_topology X I) (Pair i 'topspace (X i)))
    by (meson closed_map_component_injection closed_map_def closedin_topspace
      major minor open_map_component_injection open_map_def openin_topspace that)
    then show ?thesis
    by (metis PQ embedding_map_component_injection embedding_map_imp_homeomorphic_space
      homeomorphic_space_sym that)
  qed
  then show ?thesis by metis
qed

end

```


Chapter 2

Topology

```
theory Elementary_Topology
imports
  HOL-Library.Set_Idioms
  HOL-Library.Disjoint_Sets
  Product_Vector
begin
```

2.1 Elementary Topology

Affine transformations of intervals

```
lemma real_affinity_le:  $0 < m \implies m * x + c \leq y \iff x \leq \text{inverse } m * y + - (c / m)$ 
for  $m :: 'a::\text{linordered\_field}$ 
by (simp add: field_simps)
```

```
lemma real_le_affinity:  $0 < m \implies y \leq m * x + c \iff \text{inverse } m * y + - (c / m) \leq x$ 
for  $m :: 'a::\text{linordered\_field}$ 
by (simp add: field_simps)
```

```
lemma real_affinity_lt:  $0 < m \implies m * x + c < y \iff x < \text{inverse } m * y + - (c / m)$ 
for  $m :: 'a::\text{linordered\_field}$ 
by (simp add: field_simps)
```

```
lemma real_lt_affinity:  $0 < m \implies y < m * x + c \iff \text{inverse } m * y + - (c / m) < x$ 
for  $m :: 'a::\text{linordered\_field}$ 
by (simp add: field_simps)
```

```
lemma real_affinity_eq:  $m \neq 0 \implies m * x + c = y \iff x = \text{inverse } m * y + - (c / m)$ 
for  $m :: 'a::\text{linordered\_field}$ 
```

by (simp add: field_simps)

lemma *real_eq_affinity*: $m \neq 0 \implies y = m * x + c \longleftrightarrow \text{inverse } m * y + - (c / m) = x$
 for $m :: 'a::\text{linordered_field}$
 by (simp add: field_simps)

2.1.1 Topological Basis

context *topological_space*

begin

definition *topological_basis* $B \longleftrightarrow$
 $(\forall b \in B. \text{open } b) \wedge (\forall x. \text{open } x \longrightarrow (\exists B'. B' \subseteq B \wedge \bigcup B' = x))$

lemma *topological_basis*:
 $\text{topological_basis } B \longleftrightarrow (\forall x. \text{open } x \longleftrightarrow (\exists B'. B' \subseteq B \wedge \bigcup B' = x))$
 (is ?lhs = ?rhs)

proof

show ?lhs \implies ?rhs

by (meson local.open_Union subsetD topological_basis_def)

show ?rhs \implies ?lhs

unfolding topological_basis_def

by (metis cSup_singleton empty_subsetI insert_subset)

qed

lemma *topological_basis_iff*:
 assumes $\bigwedge B'. B' \in B \implies \text{open } B'$
 shows $\text{topological_basis } B \longleftrightarrow (\forall O'. \text{open } O' \longrightarrow (\forall x \in O'. \exists B' \in B. x \in B' \wedge B' \subseteq O'))$
 (is $_ \longleftrightarrow$?rhs)

proof safe

fix O' and $x :: 'a$

assume H : $\text{topological_basis } B \text{ open } O' x \in O'$

then have $(\exists B' \subseteq B. \bigcup B' = O')$ by (simp add: topological_basis_def)

then obtain B' where $B' \subseteq B \ O' = \bigcup B'$ by auto

then show $\exists B' \in B. x \in B' \wedge B' \subseteq O'$ using H by auto

next

assume H : ?rhs

show $\text{topological_basis } B$

using *assms* unfolding topological_basis_def

proof safe

fix $O' :: 'a \text{ set}$

assume $\text{open } O'$

with H obtain f where $\forall x \in O'. f x \in B \wedge x \in f x \wedge f x \subseteq O'$

by (force intro: bchoice simp: Bex_def)

then show $\exists B' \subseteq B. \bigcup B' = O'$

by (auto intro: exI[where $x = \{f x \mid x. x \in O'\}$])

qed

qed

lemma *topological_basisI*:

assumes $\bigwedge B'. B' \in B \implies \text{open } B'$
 and $\bigwedge O' x. \text{open } O' \implies x \in O' \implies \exists B' \in B. x \in B' \wedge B' \subseteq O'$
 shows *topological_basis* B
 by (*simp add: assms topological_basis_iff*)

lemma *topological_basisE*:

fixes O'
 assumes *topological_basis* B
 and *open* O'
 and $x \in O'$
 obtains B' where $B' \in B$ $x \in B'$ $B' \subseteq O'$
 by (*metis assms topological_basis_def topological_basis_iff*)

lemma *topological_basis_open*:

assumes *topological_basis* B and $X \in B$
 shows *open* X
 using *assms* by (*simp add: topological_basis_def*)

lemma *topological_basis_imp_subbasis*:

assumes B: *topological_basis* B
 shows *open* = *generate_topology* B
 proof (*intro ext iffI*)
 fix S :: 'a set
 assume *open* S
 with B obtain B' where $B' \subseteq B$ $S = \bigcup B'$
 unfolding *topological_basis_def* by *blast*
 then show *generate_topology* B S
 by (*auto intro: generate_topology.intros dest: topological_basis_open*)
 next
 fix S :: 'a set
 assume *generate_topology* B S
 then show *open* S
 by *induct* (*auto dest: topological_basis_open[OF B]*)
 qed

lemma *basis_dense*:

fixes B :: 'a set set
 and f :: 'a set \Rightarrow 'a
 assumes *topological_basis* B and $\bigwedge B'. B' \neq \{\} \implies f B' \in B'$
 shows $\forall X. \text{open } X \longrightarrow X \neq \{\} \longrightarrow (\exists B' \in B. f B' \in X)$
 by (*metis assms equals0D in_mono topological_basisE*)

end

lemma *topological_basis_prod*:

assumes A: *topological_basis* A

```

    and B: topological_basis B
  shows topological_basis ((λ(a, b). a × b) ‘ (A × B))
proof –
  have ∃ X ⊆ A × B. (⋃(a, b) ∈ X. a × b) = S if open S for S
proof –
  have (x, y) ∈ (⋃(a, b) ∈ {X ∈ A × B. fst X × snd X ⊆ S}. a × b)
    if xy: (x, y) ∈ S for x y
proof –
  obtain a b where a: open a x ∈ a and b: open b y ∈ b and a × b ⊆ S
    by (metis open_prod_elim[OF ‹open S›] xy mem_Sigma_iff)
  moreover obtain A0 where A0 ∈ A x ∈ A0 A0 ⊆ a
    using A a b topological_basisE by blast
  moreover
  from B b obtain B0 where B0 ∈ B y ∈ B0 B0 ⊆ b
    by (rule topological_basisE)
  ultimately show ?thesis
    by (intro UN_I[of (A0, B0)]) auto
qed
  then have (⋃(a, b) ∈ {x ∈ A × B. fst x × snd x ⊆ S}. a × b) = S
    by force
  then show ?thesis
    using subset_eq by force
qed
with A B show ?thesis
  unfolding topological_basis_def
  by (smt (verit) SigmaE imageE image_mono open_Times case_prod_conv)
qed

```

2.1.2 Countable Basis

```

locale countable_basis = topological_space p for p::'a set ⇒ bool +
  fixes B :: 'a set set
  assumes is_basis: topological_basis B
    and countable_basis: countable B
begin

```

```

lemma open_countable_basis_ex:
  assumes p X
  shows ∃ B' ⊆ B. X = ⋃ B'
  using assms countable_basis is_basis
  unfolding topological_basis_def by blast

```

```

lemma open_countable_basisE:
  assumes p X
  obtains B' where B' ⊆ B X = ⋃ B'
  using assms open_countable_basis_ex by auto

```

```

lemma countable_dense_exists:
  ∃ D::'a set. countable D ∧ (∀ X. p X ⟶ X ≠ {} ⟶ (∃ d ∈ D. d ∈ X))

```

proof –

let $?f = (\lambda B'. \text{SOME } x. x \in B')$
have *countable* ($?f \text{ ' } B$) **using** *countable_basis* **by** *simp*
with *basis_dense*[*OF is_basis, of ?f*] **show** *?thesis*
by (*intro exI*[**where** $x=?f \text{ ' } B$]) (*metis* (*mono_tags*) *all_not_in_conv imageI someI*)
qed

lemma *countable_dense_setE*:

obtains $D :: \text{'a set}$
where *countable* $D \wedge X. p X \implies X \neq \{\}$ $\implies \exists d \in D. d \in X$
using *countable_dense_exists* **by** *blast*

end

lemma *countable_basis_openI*: *countable_basis open B*

if *countable B topological_basis B*
using *that*
by *unfold_locales*
(*simp_all add: topological_basis topological_space.topological_basis topological_space_axioms*)

lemma (**in** *first_countable_topology*) *first_countable_basisE*:

fixes $x :: \text{'a}$
obtains \mathcal{A} **where** *countable* $\mathcal{A} \wedge A. A \in \mathcal{A} \implies x \in A \wedge A. A \in \mathcal{A} \implies \text{open } A$
 $\wedge S. \text{open } S \implies x \in S \implies (\exists A \in \mathcal{A}. A \subseteq S)$

proof –

obtain \mathcal{A} **where** $\mathcal{A}: (\forall i::\text{nat}. x \in \mathcal{A } i \wedge \text{open } (\mathcal{A } i)) (\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. \mathcal{A } i \subseteq S))$

using *first_countable_basis[of x]* **by** *metis*

moreover **have** *countable* (*range* \mathcal{A})

by *simp*

ultimately **show** *thesis*

by (*smt* (*verit, best*) *imageE rangeI that*)

qed

lemma (**in** *first_countable_topology*) *first_countable_basis_Int_stableE*:

obtains \mathcal{A} **where** *countable* $\mathcal{A} \wedge A. A \in \mathcal{A} \implies x \in A \wedge A. A \in \mathcal{A} \implies \text{open } A$
 $\wedge S. \text{open } S \implies x \in S \implies (\exists A \in \mathcal{A}. A \subseteq S)$
 $\wedge A B. A \in \mathcal{A} \implies B \in \mathcal{A} \implies A \cap B \in \mathcal{A}$

proof *atomize_elim*

obtain \mathcal{B} **where** \mathcal{B} :

countable \mathcal{B}

$\wedge B. B \in \mathcal{B} \implies x \in B$

$\wedge B. B \in \mathcal{B} \implies \text{open } B$

$\wedge S. \text{open } S \implies x \in S \implies \exists B \in \mathcal{B}. B \subseteq S$

by (*rule first_countable_basisE*) *blast*

define \mathcal{A} **where** [*abs_def*]:

$\mathcal{A} = (\lambda N. \bigcap ((\lambda n. \text{from_nat_into } \mathcal{B } n) \text{ ' } N)) \text{ ' } (\text{Collect } \text{finite}::\text{nat set set})$

```

then show  $\exists A. \text{countable } \mathcal{A} \wedge (\forall A. A \in \mathcal{A} \longrightarrow x \in A) \wedge (\forall A. A \in \mathcal{A} \longrightarrow \text{open } A) \wedge$ 
 $(\forall S. \text{open } S \longrightarrow x \in S \longrightarrow (\exists A \in \mathcal{A}. A \subseteq S)) \wedge (\forall A B. A \in \mathcal{A} \longrightarrow B \in \mathcal{A} \longrightarrow A \cap B \in \mathcal{A})$ 
proof (safe intro!: exI[where  $x=A$ ])
  show countable  $\mathcal{A}$ 
    unfolding  $\mathcal{A\_def}$  by (intro countable_image countable_Collect_finite)
  fix  $A$ 
  assume  $A \in \mathcal{A}$ 
  then show  $x \in A \text{ open } A$ 
    using  $\mathcal{B}(4)$ [OF open_UNIV] by (auto simp: \mathcal{A\_def intro: \mathcal{B} from_nat_into)
next
  let  $?int = \lambda N. \bigcap (\text{from\_nat\_into } \mathcal{B} \text{ ' } N)$ 
  fix  $A B$ 
  assume  $A \in \mathcal{A} B \in \mathcal{A}$ 
  then obtain  $N M$  where  $A = ?int N B = ?int M$  finite  $(N \cup M)$ 
    by (auto simp: \mathcal{A\_def})
  then show  $A \cap B \in \mathcal{A}$ 
    by (auto simp: \mathcal{A\_def intro!: image_eqI[where  $x=N \cup M$ ])
next
  fix  $S$ 
  assume open  $S x \in S$ 
  then obtain  $a$  where  $a \in \mathcal{B} a \subseteq S$  using  $\mathcal{B}$  by blast
  moreover have  $a \in \mathcal{A}$ 
    unfolding  $\mathcal{A\_def}$ 
  proof (rule image_eqI)
    show  $a = \bigcap (\text{from\_nat\_into } \mathcal{B} \text{ ' } \{ \text{to\_nat\_on } \mathcal{B} a \})$ 
      by (simp add: \mathcal{B} a)
    qed auto
  ultimately show  $\exists a \in \mathcal{A}. a \subseteq S$ 
    by blast
qed
qed

```

lemma (*in topological_space*) *first_countableI*:

```

assumes countable  $\mathcal{A}$ 
  and  $1: \bigwedge A. A \in \mathcal{A} \Longrightarrow x \in A \bigwedge A. A \in \mathcal{A} \Longrightarrow \text{open } A$ 
  and  $2: \bigwedge S. \text{open } S \Longrightarrow x \in S \Longrightarrow \exists A \in \mathcal{A}. A \subseteq S$ 
shows  $\exists \mathcal{A}::\text{nat} \Rightarrow \text{'a set. } (\forall i. x \in \mathcal{A} i \wedge \text{open } (\mathcal{A} i)) \wedge (\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. \mathcal{A} i \subseteq S))$ 
proof (safe intro!: exI[of  $\_ \text{from\_nat\_into } \mathcal{A}$ ])
  fix  $i$ 
  have  $\mathcal{A} \neq \{ \}$  using  $2$ [of UNIV] by auto
  show  $x \in \text{from\_nat\_into } \mathcal{A} i \text{ open } (\text{from\_nat\_into } \mathcal{A} i)$ 
    using range_from_nat_into_subset[OF  $\langle \mathcal{A} \neq \{ \} \rangle$ ]  $1$  by auto
next
  fix  $S$ 
  assume open  $S x \in S$ 
  then show  $\exists i. \text{from\_nat\_into } \mathcal{A} i \subseteq S$ 

```

```

    by (metis 2 ‹countable  $\mathcal{A}$ › from_nat_into_surj)
qed

instance prod :: (first_countable_topology, first_countable_topology) first_countable_topology
proof
  fix x :: 'a × 'b
  obtain  $\mathcal{A}$  where  $\mathcal{A}$ :
    countable  $\mathcal{A}$ 
     $\bigwedge a. a \in \mathcal{A} \implies \text{fst } x \in a$ 
     $\bigwedge a. a \in \mathcal{A} \implies \text{open } a$ 
     $\bigwedge S. \text{open } S \implies \text{fst } x \in S \implies \exists a \in \mathcal{A}. a \subseteq S$ 
  by (rule first_countable_basisE[of fst x]) blast
  obtain  $\mathcal{B}$  where  $\mathcal{B}$ :
    countable  $\mathcal{B}$ 
     $\bigwedge a. a \in \mathcal{B} \implies \text{snd } x \in a$ 
     $\bigwedge a. a \in \mathcal{B} \implies \text{open } a$ 
     $\bigwedge S. \text{open } S \implies \text{snd } x \in S \implies \exists a \in \mathcal{B}. a \subseteq S$ 
  by (rule first_countable_basisE[of snd x]) blast
  show  $\exists \mathcal{A} :: \text{nat} \Rightarrow ('a \times 'b)$  set.
    ( $\forall i. x \in \mathcal{A} i \wedge \text{open } (\mathcal{A} i)$ )  $\wedge$  ( $\forall S. \text{open } S \wedge x \in S \longrightarrow (\exists i. \mathcal{A} i \subseteq S)$ )
  proof (rule first_countableI[of ( $\lambda(a, b). a \times b$ ) ' ( $\mathcal{A} \times \mathcal{B}$ )], safe)
    fix a b
    assume x:  $a \in \mathcal{A} \ b \in \mathcal{B}$ 
    show  $x \in a \times b$ 
      by (simp add:  $\mathcal{A}(2) \ \mathcal{B}(2)$  mem_Times_iff x)
    show open (a × b)
      by (simp add:  $\mathcal{A}(3) \ \mathcal{B}(3)$  open_Times x)
  next
  fix S
  assume open S x ∈ S
  then obtain a' b' where a'b': open a' open b' x ∈ a' × b' a' × b' ⊆ S
    by (rule open_prod_elim)
  moreover
  obtain a b where a ∈  $\mathcal{A}$  a ⊆ a' b ∈  $\mathcal{B}$  b ⊆ b'
    by (meson  $\mathcal{B}(4) \ \mathcal{A}(4)$  a'b' mem_Times_iff)
  ultimately
  show  $\exists a \in (\lambda(a, b). a \times b)$  ' ( $\mathcal{A} \times \mathcal{B}$ ). a ⊆ S
    by (auto intro!: bexI[of _ a × b] bexI[of _ a] bexI[of _ b])
  qed (simp add:  $\mathcal{A} \ \mathcal{B}$ )
qed

class second_countable_topology = topological_space +
  assumes ex_countable_subbasis:
     $\exists B :: 'a$  set set. countable B  $\wedge$  open = generate_topology B
begin

lemma ex_countable_basis:  $\exists B :: 'a$  set set. countable B  $\wedge$  topological_basis B
proof –
  from ex_countable_subbasis obtain B where B: countable B open = gener-
```

```

ate_topology B
  by blast
let ?B = Inter ' {b. finite b ∧ b ⊆ B }

show ?thesis
proof (intro exI conjI)
  show countable ?B
  by (intro countable_image countable_Collect_finite_subset B)
  {
  fix S
  assume open S
  then have ∃ B' ⊆ {b. finite b ∧ b ⊆ B}. (⋃ b ∈ B'. ⋂ b) = S
    unfolding B
  proof induct
    case UNIV
    show ?case by (intro exI[of _ {}]) simp
  next
  case (Int a b)
  then obtain x y where x: a = ⋃ (Inter ' x) ∧ i. i ∈ x ⇒ finite i ∧ i ⊆ B
    and y: b = ⋃ (Inter ' y) ∧ i. i ∈ y ⇒ finite i ∧ i ⊆ B
    by blast
  show ?case
    unfolding x y Int_UN_distrib2
    by (intro exI[of _ {i ∪ j | i j. i ∈ x ∧ j ∈ y}]) (auto dest: x(2) y(2))
  next
  case (UN K)
  then have ∀ k ∈ K. ∃ B' ⊆ {b. finite b ∧ b ⊆ B}. ⋃ (Inter ' B') = k by auto
  then obtain k where
    ∀ ka ∈ K. k ka ⊆ {b. finite b ∧ b ⊆ B} ∧ ⋃ (Inter ' (k ka)) = ka
    unfolding bchoice_iff ..
  then show ∃ B' ⊆ {b. finite b ∧ b ⊆ B}. ⋃ (Inter ' B') = ⋃ K
    by (intro exI[of _ ⋃ (k ' K)]) auto
  next
  case (Basis S)
  then show ?case
    by (intro exI[of _ {{S}}]) auto
  qed
  then have (∃ B' ⊆ Inter ' {b. finite b ∧ b ⊆ B}. ⋃ B' = S)
    unfolding subset_image_iff by blast }
  then show topological_basis ?B
  unfolding topological_basis_def
  by (safe intro!: open_Inter)
    (simp_all add: B generate_topology.Basis_subset_eq)
  qed
qed
end

```


lemma *univ_second_countable*:

obtains $\mathcal{B} :: 'a::\text{second_countable_topology set set}$
where *countable* $\mathcal{B} \wedge C. C \in \mathcal{B} \implies \text{open } C$
 $\wedge S. \text{open } S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$
by (*metis ex_countable_basis topological_basis_def*)

proposition *Lindelof*:

fixes $\mathcal{F} :: 'a::\text{second_countable_topology set set}$
assumes $\mathcal{F}: \wedge S. S \in \mathcal{F} \implies \text{open } S$
obtains \mathcal{F}' **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$
proof –
obtain $\mathcal{B} :: 'a \text{ set set}$
where *countable* $\mathcal{B} \wedge C. C \in \mathcal{B} \implies \text{open } C$
and $\mathcal{B}: \wedge S. \text{open } S \implies \exists U. U \subseteq \mathcal{B} \wedge S = \bigcup U$
using *univ_second_countable* **by** *blast*
define \mathcal{D} **where** $\mathcal{D} \equiv \{S. S \in \mathcal{B} \wedge (\exists U. U \in \mathcal{F} \wedge S \subseteq U)\}$
have *countable* \mathcal{D}
by (*simp add: D_def <countable B>*)
have $\wedge S. \exists U. S \in \mathcal{D} \longrightarrow U \in \mathcal{F} \wedge S \subseteq U$
by (*simp add: D_def*)
then obtain G **where** $G: \wedge S. S \in \mathcal{D} \longrightarrow G S \in \mathcal{F} \wedge S \subseteq G S$
by *metis*
have $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$
unfolding \mathcal{D}_def **by** (*blast dest: F B*)
moreover have $\bigcup \mathcal{D} \subseteq \bigcup \mathcal{F}$
using \mathcal{D}_def **by** *blast*
ultimately have $\text{eq1}: \bigcup \mathcal{F} = \bigcup \mathcal{D} ..$
moreover have $\bigcup \mathcal{D} = \bigcup (G ` \mathcal{D})$
using G eq1 **by** *auto*
ultimately show *?thesis*
by (*metis G <countable D> countable_image image_subset_iff that*)
qed

lemma *countable_disjoint_open_subsets*:

fixes $\mathcal{F} :: 'a::\text{second_countable_topology set set}$
assumes $\wedge S. S \in \mathcal{F} \implies \text{open } S$ **and** *pw: pairwise disjoint* \mathcal{F}
shows *countable* \mathcal{F}
proof –
obtain \mathcal{F}' **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$
by (*metis assms Lindelof*)
with *pw* **have** $\mathcal{F} \subseteq \text{insert } \{\} \mathcal{F}'$
by (*fastforce simp add: pairwise_def disjoint_iff*)
then show *?thesis*
by (*simp add: <countable F'> countable_subset*)
qed

sublocale *second_countable_topology* <

countable_basis open SOME B. countable B \wedge *topological_basis B*
using *someI_ex[OF ex_countable_basis]*

by *unfold_locales safe*

```

instance prod :: (second_countable_topology, second_countable_topology) second_countable_topology
proof
  obtain A :: 'a set set where countable A topological_basis A
    using ex_countable_basis by auto
  moreover
  obtain B :: 'b set set where countable B topological_basis B
    using ex_countable_basis by auto
  ultimately show  $\exists B::('a \times 'b)$  set set. countable B  $\wedge$  open = generate_topology
  B
    by (auto intro!: exI[of _  $(\lambda(a, b). a \times b)$ ] ' (A  $\times$  B)] topological_basis_prod
      topological_basis_imp_subbasis)
qed

```

```

instance second_countable_topology  $\subseteq$  first_countable_topology
proof
  fix x :: 'a
  define B :: 'a set set where B = (SOME B. countable B  $\wedge$  topological_basis B)
  then have B: countable B topological_basis B
    using countable_basis_is_basis
  by (auto simp: countable_basis_is_basis)
  then show  $\exists A::nat \Rightarrow 'a$  set.
    ( $\forall i. x \in A \ i \wedge$  open (A i))  $\wedge$  ( $\forall S. open \ S \wedge x \in S \longrightarrow (\exists i. A \ i \subseteq S)$ )
    by (intro first_countableI[of {b $\in$ B. x  $\in$  b}])
      (fastforce simp: topological_space_class.topological_basis_def)+
qed

```

```

instance nat :: second_countable_topology
proof
  show  $\exists B::nat$  set set. countable B  $\wedge$  open = generate_topology B
    by (intro exI[of _ range lessThan  $\cup$  range greaterThan]) (auto simp: open_nat_def)
qed

```

```

lemma countable_separating_set_linorder1:
  shows  $\exists B::('a::\{linorder\_topology, second\_countable\_topology\}$  set). countable
  B  $\wedge$  ( $\forall x \ y. x < y \longrightarrow (\exists b \in B. x < b \wedge b \leq y)$ )
proof -
  obtain A::'a set set where countable A topological_basis A using ex_countable_basis
by auto
  define B1 where B1 = {(LEAST x. x  $\in$  U) | U. U  $\in$  A}
  then have countable B1 using <countable A> by (simp add: Setcompr_eq_image)
  define B2 where B2 = {(SOME x. x  $\in$  U) | U. U  $\in$  A}
  then have countable B2 using <countable A> by (simp add: Setcompr_eq_image)
  have  $\exists b \in B1 \cup B2. x < b \wedge b \leq y$  if  $x < y$  for x y
  proof (cases)
    assume  $\exists z. x < z \wedge z < y$ 
    then obtain z where z:  $x < z \wedge z < y$  by auto

```

```

define U where U = {x<..

```

lemma countable_separating_set_linorder2:

shows $\exists B::('a::\{linorder_topology, second_countable_topology\} set). countable$
 $B \wedge (\forall x y. x < y \longrightarrow (\exists b \in B. x \leq b \wedge b < y))$

proof –

obtain A::'a set set where countable A topological_basis A using ex_countable_basis
by auto

define B1 where B1 = {(GREATEST x. x ∈ U) | U. U ∈ A}

then have countable B1 using ‹countable A› by (simp add: Setcompr_eq_image)

define B2 where B2 = {(SOME x. x ∈ U) | U. U ∈ A}

then have countable B2 using ‹countable A› by (simp add: Setcompr_eq_image)

have $\exists b \in B1 \cup B2. x \leq b \wedge b < y$ if $x < y$ for $x y$

proof (cases)

assume $\exists z. x < z \wedge z < y$

then obtain z where $z: x < z \wedge z < y$ by auto

define U where U = {x<..

then have open U by simp

moreover have z ∈ U using z U_def by simp

ultimately obtain V where V ∈ A z ∈ V V ⊆ U

```

    using topological_basisE[OF ‹topological_basis A›] by auto
    define w where w = (SOME x. x ∈ V)
    then have w ∈ V using ‹z ∈ V› by (metis someI2)
    then have  $x \leq w \wedge w < y$  using ‹w ∈ V› ‹V ⊆ U› U_def by fastforce
    moreover have  $w \in B1 \cup B2$  using w_def B2_def ‹V ∈ A› by auto
    ultimately show ?thesis by auto
next
assume  $\neg(\exists z. x < z \wedge z < y)$ 
then have *:  $\bigwedge z. z < y \implies z \leq x$  using leI by blast
define U where U = {.. $y$ }
then have open U by simp
moreover have  $x \in U$  using ‹x < y› U_def by simp
ultimately obtain V where V ∈ A x ∈ V V ⊆ U
    using topological_basisE[OF ‹topological_basis A›] by auto
have U = {.. $x$ } unfolding U_def using * ‹x < y› by auto
then have V ⊆ {.. $x$ } using ‹V ⊆ U› by simp
    then have (GREATEST x. x ∈ V) = x using ‹x ∈ V› by (meson Greatest_equality atMost_iff subsetCE)
    then have  $x \in B1 \cup B2$  using ‹V ∈ A› B1_def by auto
    moreover have  $x \leq x \wedge x < y$  using ‹x < y› by simp
    ultimately show ?thesis by auto
qed
moreover have countable (B1 ∪ B2) using ‹countable B1› ‹countable B2› by
simp
ultimately show ?thesis by auto
qed

lemma countable_separating_set_dense_linorder:
  shows  $\exists B::('a::\{linorder\_topology, dense\_linorder, second\_countable\_topology\}$ 
  set). countable B  $\wedge (\forall x y. x < y \implies (\exists b \in B. x < b \wedge b < y))$ 
proof -
  obtain B::'a set where B: countable B  $\wedge x y. x < y \implies (\exists b \in B. x < b \wedge b \leq$ 
  y)
    using countable_separating_set_linorder1 by auto
  have  $\exists b \in B. x < b \wedge b < y$  if  $x < y$  for x y
  proof -
    obtain z where  $x < z z < y$  using ‹x < y› dense by blast
    then obtain b where b ∈ B  $x < b \wedge b \leq z$  using B(2) by auto
    then have  $x < b \wedge b < y$  using ‹z < y› by auto
    then show ?thesis using ‹b ∈ B› by auto
  qed
  then show ?thesis using B(1) by auto
qed

```

2.1.3 Polish spaces

Textbooks define Polish spaces as completely metrizable. We assume the topology to be complete for a given metric.

```
class polish_space = complete_space + second_countable_topology
```

2.1.4 Limit Points

definition (in *topological_space*) *islimpt*:: 'a \Rightarrow 'a set \Rightarrow bool (**infixr** *islimpt* 60)
 where $x \text{ islimpt } S \longleftrightarrow (\forall T. x \in T \longrightarrow \text{open } T \longrightarrow (\exists y \in S. y \in T \wedge y \neq x))$

lemma *islimptI*:

assumes $\bigwedge T. x \in T \Longrightarrow \text{open } T \Longrightarrow \exists y \in S. y \in T \wedge y \neq x$
shows $x \text{ islimpt } S$
using *assms unfolding islimpt_def by auto*

lemma *islimptE*:

assumes $x \text{ islimpt } S$ **and** $x \in T$ **and** *open* T
obtains y **where** $y \in S$ **and** $y \in T$ **and** $y \neq x$
using *assms unfolding islimpt_def by auto*

lemma *islimpt_iff_eventually*: $x \text{ islimpt } S \longleftrightarrow \neg \text{eventually } (\lambda y. y \notin S) \text{ (at } x)$
unfolding *islimpt_def eventually_at_topological by auto*

lemma *islimpt_subset*: $x \text{ islimpt } S \Longrightarrow S \subseteq T \Longrightarrow x \text{ islimpt } T$
unfolding *islimpt_def by fast*

lemma *islimpt_UNIV_iff*: $x \text{ islimpt } UNIV \longleftrightarrow \neg \text{open } \{x\}$
unfolding *islimpt_def by (safe, fast, case_tac T = {x}, fast, fast)*

lemma *islimpt_punctured*: $x \text{ islimpt } S = x \text{ islimpt } (S - \{x\})$
unfolding *islimpt_def by blast*

A perfect space has no isolated points.

lemma *islimpt_UNIV [simp, intro]*: $x \text{ islimpt } UNIV$
for $x :: 'a::\text{perfect_space}$
unfolding *islimpt_UNIV_iff by (rule not_open_singleton)*

lemma *closed_limpt*: $\text{closed } S \longleftrightarrow (\forall x. x \text{ islimpt } S \longrightarrow x \in S)$
unfolding *closed_def open_subopen [of -S]*
by (*metis Compl_iff islimptE islimptI open_subopen subsetI*)

lemma *islimpt_EMPTY [simp]*: $\neg x \text{ islimpt } \{\}$
by (*auto simp: islimpt_def*)

lemma *islimpt_Un*: $x \text{ islimpt } (S \cup T) \longleftrightarrow x \text{ islimpt } S \vee x \text{ islimpt } T$
by (*simp add: islimpt_iff_eventually eventually_conj_iff*)

lemma *islimpt_finite_union_iff*:

assumes *finite* A
shows $z \text{ islimpt } (\bigcup x \in A. B x) \longleftrightarrow (\exists x \in A. z \text{ islimpt } B x)$
using *assms by (induction rule: finite_induct) (simp_all add: islimpt_Un)*

lemma *islimpt_insert*:

fixes $x :: 'a::t1_space$
shows $x \text{ islimpt } (\text{insert } a \text{ } S) \longleftrightarrow x \text{ islimpt } S$

proof

assume x *islimpt* (*insert* a S)

then show x *islimpt* S

by (*metis* *closed_limpt* *closed_singleton* *empty_iff* *insert_iff* *insert_is_Un* *islimpt_Un* *islimpt_def*)

next

assume x *islimpt* S

then show x *islimpt* (*insert* a S)

by (*rule* *islimpt_subset*) *auto*

qed

lemma *islimpt_finite*:

fixes $x :: 'a :: t1_space$

shows *finite* $S \implies \neg x$ *islimpt* S

by (*induct* *set*: *finite*) (*simp_all* *add*: *islimpt_insert*)

lemma *islimpt_Un_finite*:

fixes $x :: 'a :: t1_space$

shows *finite* $S \implies x$ *islimpt* ($S \cup T$) $\longleftrightarrow x$ *islimpt* T

by (*simp* *add*: *islimpt_Un* *islimpt_finite*)

lemma *islimpt_eq_acc_point*:

fixes $l :: 'a :: t1_space$

shows l *islimpt* $S \longleftrightarrow (\forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap S))$

proof (*safe intro!*: *islimptI*)

fix U

assume l *islimpt* S $l \in U$ *open* U *finite* ($U \cap S$)

then have l *islimpt* S $l \in (U - (U \cap S - \{l\}))$ *open* ($U - (U \cap S - \{l\})$)

by (*auto* *intro*: *finite_imp_closed*)

then show *False*

by (*rule* *islimptE*) *auto*

next

fix T

assume $*$: $\forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap S)$ $l \in T$ *open* T

then have $\exists x. x \in (T \cap S - \{l\})$

by (*metis* *ex_in_conv* *finite.emptyI* *infinite_remove*)

then show $\exists y \in S. y \in T \wedge y \neq l$

by *auto*

qed

lemma *acc_point_range_imp_convergent_subsequence*:

fixes $l :: 'a :: \text{first_countable_topology}$

assumes l : $\forall U. l \in U \longrightarrow \text{open } U \longrightarrow \text{infinite } (U \cap \text{range } f)$

shows $\exists r :: \text{nat} \Rightarrow \text{nat}. \text{strict_mono } r \wedge (f \circ r) \longrightarrow l$

proof –

from *countable_basis_at_decseq*[*of* l]

obtain A **where** A :

$\bigwedge i. \text{open } (A\ i)$

$\bigwedge i. l \in A\ i$

```

   $\bigwedge S. \text{open } S \implies l \in S \implies \text{eventually } (\lambda i. A \ i \subseteq S) \text{ sequentially}$ 
  by blast
define s where s n i = (SOME j. i < j  $\wedge$  f j  $\in$  A (Suc n)) for n i
{
  fix n i
  have infinite (A (Suc n)  $\cap$  range f - f'{'.. i})
    using l A by auto
  then have  $\exists x. x \in A$  (Suc n)  $\cap$  range f - f'{'.. i}
    by (metis all_not_in_conv finite.emptyI)
  then have  $\exists a. i < a \wedge f a \in A$  (Suc n)
    by (force simp: linorder_not_le)
  then have i < s n i f (s n i)  $\in$  A (Suc n)
    unfolding s_def by (auto intro: someI2_ex)
}
note s = this
define r where r = rec_nat (s 0 0) s
have strict_mono r
  by (auto simp: r_def s strict_mono_Suc_iff)
moreover
have  $(\lambda n. f (r n)) \longrightarrow l$ 
proof (rule topological_tendstoI)
  fix S
  assume open S l  $\in$  S
  with A(3) have eventually  $(\lambda i. A \ i \subseteq S)$  sequentially
    by auto
  moreover
  {
    fix i
    assume Suc 0  $\leq$  i
    then have f (r i)  $\in$  A i
      by (cases i) (simp_all add: r_def s)
  }
  then have eventually  $(\lambda i. f (r i) \in A \ i)$  sequentially
    by (auto simp: eventually_sequentially)
  ultimately show eventually  $(\lambda i. f (r i) \in S)$  sequentially
    by eventually_elim auto
qed
  ultimately show  $\exists r::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
    by (auto simp: convergent_def comp_def)
qed

```

```

lemma islimpt_range_imp_convergent_subsequence:
  fixes l :: 'a :: {t1_space, first_countable_topology}
  assumes l: l islimpt (range f)
  shows  $\exists r::\text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  using l unfolding islimpt_eq_acc_point
  by (rule acc_point_range_imp_convergent_subsequence)

```

```

lemma sequence_unique_limpt:

```

```

fixes  $f :: \text{nat} \Rightarrow 'a::t2\_space$ 
assumes  $f: (f \longrightarrow l)$  sequentially
  and  $l' \text{ islimpt } (\text{range } f)$ 
shows  $l' = l$ 
proof (rule ccontr)
  assume  $l' \neq l$ 
  obtain  $s\ t$  where  $\text{open } s\ \text{open } t\ l' \in s\ l \in t\ s \cap t = \{\}$ 
    using hausdorff [OF  $\langle l' \neq l \rangle$ ] by auto
  then obtain  $N$  where  $\forall n \geq N. f\ n \in t$ 
    by (meson f lim_explicit)

  have  $UNIV = \{..<N\} \cup \{N..\}$ 
    by auto
  then have  $l' \text{ islimpt } (f \text{ ' } (\{..<N\} \cup \{N..\}))$ 
    using assms(2) by simp
  then have  $l' \text{ islimpt } (f \text{ ' } \{..<N\} \cup f \text{ ' } \{N..\})$ 
    by (simp add: image_Un)
  then have  $l' \text{ islimpt } (f \text{ ' } \{N..\})$ 
    by (simp add: islimpt_Un_finite)
  then obtain  $y$  where  $y \in f \text{ ' } \{N..\}\ y \in s\ y \neq l'$ 
    using  $\langle l' \in s \rangle \langle \text{open } s \rangle$  by (rule islimptE)
  then obtain  $n$  where  $N \leq n\ f\ n \in s\ f\ n \neq l'$ 
    by auto
  with  $\langle \forall n \geq N. f\ n \in t \rangle \langle s \cap t = \{\} \rangle$  show False
    by blast
qed

```

lemma *islimpt_sequential*:

fixes $x :: 'a::\text{first_countable_topology}$

shows $x \text{ islimpt } S \longleftrightarrow (\exists f. (\forall n::\text{nat}. f\ n \in S - \{x\}) \wedge (f \longrightarrow x) \text{ sequentially})$

(**is** *?lhs = ?rhs*)

proof

assume *?lhs*

from *countable_basis_at_decseq*[*of* x] **obtain** A **where** A :

$\bigwedge i. \text{open } (A\ i)$

$\bigwedge i. x \in A\ i$

$\bigwedge S. \text{open } S \implies x \in S \implies \text{eventually } (\lambda i. A\ i \subseteq S) \text{ sequentially}$

by *blast*

define f **where** $f\ n = (\text{SOME } y. y \in S \wedge y \in A\ n \wedge x \neq y)$ **for** n

{

fix n

from (*?lhs*) **have** $\exists y. y \in S \wedge y \in A\ n \wedge x \neq y$

unfolding *islimpt_def* **using** *A*(1,2)[*of* n] **by** *auto*

then have $f\ n \in S \wedge f\ n \in A\ n \wedge x \neq f\ n$

unfolding *f_def* **by** (*rule someI_ex*)

then have $f\ n \in S\ f\ n \in A\ n\ x \neq f\ n$ **by** *auto*

}

then have $\forall n. f\ n \in S - \{x\}$ **by** *auto*


```

moreover have  $(\lambda n. f n) \longrightarrow x$ 
proof (rule topological_tendstoI)
  fix  $S$ 
  assume  $open\ S\ x \in S$ 
  from  $A(\beta)[OF\ this] \langle \bigwedge n. f\ n \in A\ n \rangle$ 
  show eventually  $(\lambda x. f\ x \in S)$  sequentially
    by (auto elim!: eventually_mono)
qed
ultimately show ?rhs by fast
next
assume ?rhs
then obtain  $f :: nat \Rightarrow 'a$  where  $f: \bigwedge n. f\ n \in S - \{x\}$  and  $lim: f \longrightarrow x$ 
  by auto
show ?lhs
  unfolding islimpt_def
proof safe
  fix  $T$ 
  assume  $open\ T\ x \in T$ 
  from  $lim[THEN\ topological_tendstoD, OF\ this] f$ 
  show  $\exists y \in S. y \in T \wedge y \neq x$ 
  unfolding eventually_sequentially by auto
qed
qed

lemma islimpt_isCont_image:
  fixes  $f :: 'a :: \{first\_countable\_topology, t2\_space\} \Rightarrow 'b :: \{first\_countable\_topology, t2\_space\}$ 
  assumes  $x\ islimpt\ A$  and  $isCont\ f\ x$  and  $ev: eventually\ (\lambda y. f\ y \neq f\ x)\ (at\ x)$ 
  shows  $f\ x\ islimpt\ f\ 'A$ 
proof -
  from  $assms(1)$  obtain  $g$  where  $g: g \longrightarrow x$   $range\ g \subseteq A - \{x\}$ 
  unfolding islimpt_sequential by blast
  have  $filterlim\ g\ (at\ x)$  sequentially
  using  $g$  by (auto simp: filterlim_at intro!: always_eventually)
  then obtain  $N$  where  $N: \bigwedge n. n \geq N \implies f\ (g\ n) \neq f\ x$ 
  by (metis (mono_tags, lifting) ev eventually_at_top_linorder filterlim_iff)
  have  $(\lambda x. g\ (x + N)) \longrightarrow x$ 
  using  $g(1)$  by (rule LIMSEQ_ignore_initial_segment)
  hence  $(\lambda x. f\ (g\ (x + N))) \longrightarrow f\ x$ 
  using  $assms(2)$   $isCont_tendsto_compose$  by blast
  moreover have  $range\ (\lambda x. f\ (g\ (x + N))) \subseteq f\ 'A - \{f\ x\}$ 
  using  $g(2)\ N$  by auto
  ultimately show ?thesis
  unfolding islimpt_sequential by (intro exI[of _  $\lambda x. f\ (g\ (x + N))$ ]) auto
qed

lemma islimpt_image:
  assumes  $z\ islimpt\ g - 'A \cap B$   $g\ z \notin A$   $z \in B$  continuous_on B g
  shows  $g\ z\ islimpt\ A$ 

```

by (smt (verit, best) IntD1 assms continuous_on_topological_inf_le2 islimpt_def subset_eq vimageE)

2.1.5 Interior of a Set

definition *interior* :: ('a::topological_space) set \Rightarrow 'a set **where**
interior $S = \bigcup \{T. \text{open } T \wedge T \subseteq S\}$

lemma *interiorI* [intro?]:
 assumes *open T and x \in T and T \subseteq S*
 shows *x \in interior S*
 using *assms unfolding interior_def* by fast

lemma *interiorE* [elim?]:
 assumes *x \in interior S*
 obtains *T where open T and x \in T and T \subseteq S*
 using *assms unfolding interior_def* by fast

lemma *open_interior* [simp, intro]: *open (interior S)*
 by (simp add: interior_def open_Union)

lemma *interior_subset*: *interior S \subseteq S*
 by (auto simp: interior_def)

lemma *interior_maximal*: *T \subseteq S \implies open T \implies T \subseteq interior S*
 by (auto simp: interior_def)

lemma *interior_open*: *open S \implies interior S = S*
 by (intro equalityI interior_subset interior_maximal subset_refl)

lemma *interior_eq*: *interior S = S \longleftrightarrow open S*
 by (metis open_interior interior_open)

lemma *open_subset_interior*: *open S \implies S \subseteq interior T \longleftrightarrow S \subseteq T*
 by (metis interior_maximal interior_subset subset_trans)

lemma *interior_empty* [simp]: *interior {} = {}*
 using *open_empty* by (rule interior_open)

lemma *interior_UNIV* [simp]: *interior UNIV = UNIV*
 using *open_UNIV* by (rule interior_open)

lemma *interior_interior* [simp]: *interior (interior S) = interior S*
 using *open_interior* by (rule interior_open)

lemma *interior_mono*: *S \subseteq T \implies interior S \subseteq interior T*
 by (auto simp: interior_def)

lemma *interior_unique*:

assumes $T \subseteq S$ **and** *open* T
assumes $\bigwedge T'. T' \subseteq S \implies \text{open } T' \implies T' \subseteq T$
shows *interior* $S = T$
by (*intro equalityI assms interior_subset open_interior interior_maximal*)

lemma *interior_singleton* [*simp*]: *interior* $\{a\} = \{\}$
for $a :: 'a::\text{perfect_space}$
by (*meson interior_eq interior_subset not_open_singleton subset_singletonD*)

lemma *interior_Int* [*simp*]: *interior* $(S \cap T) = \text{interior } S \cap \text{interior } T$
by (*meson Int_mono Int_subset_iff antisym_conv interior_maximal interior_subset open_Int open_interior*)

lemma *eventually_nhds_in_nhd*: $x \in \text{interior } s \implies \text{eventually } (\lambda y. y \in s) (\text{nhds } x)$
using *interior_subset*[*of s*] **by** (*subst eventually_nhds*) *blast*

lemma *interior_limit_point* [*intro*]:
fixes $x :: 'a::\text{perfect_space}$
assumes $x \in \text{interior } S$
shows $x \text{ islimpt } S$
proof –
obtain T **where** $x \in T$ $T \subseteq S$ *open* T
using *interior_subset* x **by** *blast*
with $x \text{ islimpt_UNIV}$ [*of x*] **show** *?thesis*
unfolding *islimpt_def* **by** (*metis (full_types) Int_iff open_Int subsetD*)
qed

lemma *open_imp_islimpt*:
fixes $x :: 'a::\text{perfect_space}$
assumes *open* S $x \in S$
shows $x \text{ islimpt } S$
using *assms interior_eq interior_limit_point* **by** *auto*

lemma *islimpt_Int_eventually*:
assumes $x \text{ islimpt } A$ *eventually* $(\lambda y. y \in B)$ (*at x*)
shows $x \text{ islimpt } A \cap B$
using *assms unfolding islimpt_def eventually_at_filter eventually_nhds*
by (*metis Int_iff UNIV_I open_Int*)

lemma *islimpt_conv_frequently_at*:
 $x \text{ islimpt } A \longleftrightarrow \text{frequently } (\lambda y. y \in A)$ (*at x*)
by (*simp add: frequently_def islimpt_iff_eventually*)

lemma *frequently_at_imp_islimpt*:
assumes *frequently* $(\lambda y. y \in A)$ (*at x*)
shows $x \text{ islimpt } A$
by (*simp add: assms islimpt_conv_frequently_at*)

```

lemma interior_closed_Un_empty_interior:
  assumes cS: closed S
    and iT: interior T = {}
  shows interior (S ∪ T) = interior S
proof
  show interior S ⊆ interior (S ∪ T)
    by (rule interior_mono) (rule Un_upper1)
  show interior (S ∪ T) ⊆ interior S
proof
  fix x
  assume x ∈ interior (S ∪ T)
  then obtain R where open R x ∈ R R ⊆ S ∪ T ..
  show x ∈ interior S
proof (rule ccontr)
  assume x ∉ interior S
  with  $\langle x \in R \rangle$   $\langle \text{open } R \rangle$  obtain y where y ∈ R - S
    unfolding interior_def by fast
  then show False
    by (metis Diff_subset_conv  $\langle R \subseteq S \cup T \rangle$   $\langle \text{open } R \rangle$  cS empty_iff iT interiorI
      open_Diff)
  qed
qed
qed

```

```

lemma interior_Times: interior (A × B) = interior A × interior B
proof (rule interior_unique)
  show interior A × interior B ⊆ A × B
    by (intro Sigma_mono interior_subset)
  show open (interior A × interior B)
    by (intro open_Times open_interior)
  fix T
  assume T ⊆ A × B and open T
  then show T ⊆ interior A × interior B
proof safe
  fix x y
  assume  $(x, y) \in T$ 
  then obtain C D where open C open D C × D ⊆ T x ∈ C y ∈ D
    using  $\langle \text{open } T \rangle$  unfolding open_prod_def by fast
  then have open D C ⊆ A D ⊆ B x ∈ C y ∈ D
    using  $\langle T \subseteq A \times B \rangle$  by auto
  then show x ∈ interior A and y ∈ interior B
    by (auto intro: interiorI)
  qed
qed

```

```

lemma interior_Ici:
  fixes x :: 'a :: {dense_linorder, linorder_topology}
  assumes b < x
  shows interior {x ..} = {x <..}

```

```

proof (rule interior_unique)
  fix T
  assume  $T \subseteq \{x \dots\}$  open T
  moreover have  $x \notin T$ 
  proof
    assume  $x \in T$ 
    obtain y where  $y < x$   $\{y < \dots x\} \subseteq T$ 
      using open_left[OF ‹open T› ‹ $x \in T$ › ‹ $b < x$ ›] by auto
    with dense[OF ‹ $y < x$ ›] obtain z where  $z \in T$   $z < x$ 
      by (auto simp: subset_eq Ball_def)
    with ‹ $T \subseteq \{x \dots\}$ › show False by auto
  qed
  ultimately show  $T \subseteq \{x < \dots\}$ 
    by (auto simp: subset_eq less_le)
qed auto

```

```

lemma interior_Iic:
  fixes x :: 'a :: {dense_linorder, linorder_topology}
  assumes  $x < b$ 
  shows interior  $\{.. x\} = \{.. < x\}$ 
proof (rule interior_unique)
  fix T
  assume  $T \subseteq \{.. x\}$  open T
  moreover have  $x \notin T$ 
  proof
    assume  $x \in T$ 
    obtain y where  $x < y$   $\{x .. < y\} \subseteq T$ 
      using open_right[OF ‹open T› ‹ $x \in T$ › ‹ $x < b$ ›] by auto
    with dense[OF ‹ $x < y$ ›] obtain z where  $z \in T$   $x < z$ 
      by (auto simp: subset_eq Ball_def less_le)
    with ‹ $T \subseteq \{.. x\}$ › show False by auto
  qed
  ultimately show  $T \subseteq \{.. < x\}$ 
    by (auto simp: subset_eq less_le)
qed auto

```

```

lemma countable_disjoint_nonempty_interior_subsets:
  fixes  $\mathcal{F} :: 'a :: \text{second\_countable\_topology}$  set set
  assumes pw: pairwise disjoint  $\mathcal{F}$  and int:  $\bigwedge S. \llbracket S \in \mathcal{F}; \text{interior } S = \{\} \rrbracket \implies S = \{\}$ 
  shows countable  $\mathcal{F}$ 
proof (rule countable_image_inj_on)
  have disjoint (interior ‘  $\mathcal{F}$ )
    using pw by (simp add: disjoint_image_subset interior_subset)
  then show countable (interior ‘  $\mathcal{F}$ )
    by (auto intro: countable_disjoint_open_subsets)
  show inj_on interior  $\mathcal{F}$ 
    using pw apply (clarsimp simp: inj_on_def pairwise_def)
    apply (metis disjoint_def disjoint_subset1 inf.orderE int interior_subset)

```

done
qed

2.1.6 Closure of a Set

definition *closure* :: ('a::topological_space) set \Rightarrow 'a set **where**
closure $S = S \cup \{x . x \text{ islimpt } S\}$

lemma *interior_closure*: $\text{interior } S = - (\text{closure } (- S))$
by (*auto simp: interior_def closure_def islimpt_def*)

lemma *closure_interior*: $\text{closure } S = - \text{interior } (- S)$
by (*simp add: interior_closure*)

lemma *closed_closure*[*simp, intro*]: $\text{closed } (\text{closure } S)$
by (*simp add: closure_interior closed_Compl*)

lemma *closure_subset*: $S \subseteq \text{closure } S$
by (*simp add: closure_def*)

lemma *closure_hull*: $\text{closure } S = \text{closed hull } S$
by (*auto simp: hull_def closure_interior interior_def*)

lemma *closure_eq*: $\text{closure } S = S \iff \text{closed } S$
unfolding *closure_hull* **using** *closed_Inter* **by** (*rule hull_eq*)

lemma *closure_closed* [*simp*]: $\text{closed } S \implies \text{closure } S = S$
by (*simp only: closure_eq*)

lemma *closure_closure* [*simp*]: $\text{closure } (\text{closure } S) = \text{closure } S$
unfolding *closure_hull* **by** (*rule hull_hull*)

lemma *closure_mono*: $S \subseteq T \implies \text{closure } S \subseteq \text{closure } T$
unfolding *closure_hull* **by** (*rule hull_mono*)

lemma *closure_minimal*: $S \subseteq T \implies \text{closed } T \implies \text{closure } S \subseteq T$
unfolding *closure_hull* **by** (*rule hull_minimal*)

lemma *closure_unique*:
assumes $S \subseteq T$
and $\text{closed } T$
and $\bigwedge T'. S \subseteq T' \implies \text{closed } T' \implies T \subseteq T'$
shows $\text{closure } S = T$
using *assms* **unfolding** *closure_hull* **by** (*rule hull_unique*)

lemma *closure_empty* [*simp*]: $\text{closure } \{\} = \{\}$
using *closed_empty* **by** (*rule closure_closed*)

lemma *closure_UNIV* [*simp*]: $\text{closure } \text{UNIV} = \text{UNIV}$

```

using closed_UNIV by (rule closure_closed)

lemma closure_Un [simp]: closure (S ∪ T) = closure S ∪ closure T
  by (simp add: closure_interior)

lemma closure_eq_empty [iff]: closure S = {} ↔ S = {}
  using closure_empty closure_subset[of S] by blast

lemma closure_subset_eq: closure S ⊆ S ↔ closed S
  using closure_eq[of S] closure_subset[of S] by simp

lemma open_Int_closure_eq_empty: open S ⇒ (S ∩ closure T) = {} ↔ S ∩
T = {}
  using open_subset_interior[of S - T]
  using interior_subset[of - T]
  by (auto simp: closure_interior)

lemma open_Int_closure_subset: open S ⇒ S ∩ closure T ⊆ closure (S ∩ T)
proof
  fix x
  assume *: open S x ∈ S ∩ closure T
  then have x islimpt (S ∩ T) if x islimpt T
    by (metis IntD1 eventually_at_in_open' inf_commute islimpt_Int_eventually
that)
  with * show x ∈ closure (S ∩ T)
    unfolding closure_def by blast
qed

lemma closure_complement: closure (- S) = - interior S
  by (simp add: closure_interior)

lemma interior_complement: interior (- S) = - closure S
  by (simp add: closure_interior)

lemma interior_diff: interior(S - T) = interior S - closure T
  by (simp add: Diff_eq interior_complement)

lemma closure_Times: closure (A × B) = closure A × closure B
proof (rule closure_unique)
  show A × B ⊆ closure A × closure B
    by (intro Sigma_mono closure_subset)
  show closed (closure A × closure B)
    by (intro closed_Times closed_closure)
fix T
assume A × B ⊆ T and closed T
then show closure A × closure B ⊆ T
  apply (simp add: closed_def open_prod_def, clarify)
  apply (rule ccontr)
  apply (drule_tac x=(a, b) in bspec, simp, clarify, rename_tac C D)

```

```

apply (simp add: closure_interior interior_def)
apply (drule_tac x=C in spec)
apply (drule_tac x=D in spec, auto)
done

```

qed

lemma closure_open_Int_superset:

assumes open S $S \subseteq \text{closure } T$

shows $\text{closure}(S \cap T) = \text{closure } S$

by (metis Int_Un_distrib Un_Int_eq(4) assms closure_Un closure_closure open_Int_closure_subset sup.orderE)

lemma closure_Int: $\text{closure}(\bigcap I) \subseteq \bigcap \{\text{closure } S \mid S. S \in I\}$

by (simp add: INF_greatest Inter_lower Setcompr_eq_image closure_mono)

lemma islimpt_in_closure: $(x \text{ islimpt } S) = (x \in \text{closure}(S - \{x\}))$

unfolding closure_def **using** islimpt_punctured **by** blast

lemma connected_imp_connected_closure: $\text{connected } S \implies \text{connected}(\text{closure } S)$

by (rule connectedI) (meson closure_subset open_Int open_Int_closure_eq_empty subset_trans connectedD)

lemma bdd_below_closure:

fixes A :: real set

assumes bdd_below A

shows bdd_below (closure A)

proof –

from assms **obtain** m **where** $\bigwedge x. x \in A \implies m \leq x$

by (auto simp: bdd_below_def)

then have $A \subseteq \{m..\}$ **by** auto

then have $\text{closure } A \subseteq \{m..\}$

using closed_real_atLeast **by** (rule closure_minimal)

then show ?thesis

by (auto simp: bdd_below_def)

qed

2.1.7 Frontier (also known as boundary)

definition frontier :: ('a::topological_space) set \Rightarrow 'a set **where**

frontier S = closure S – interior S

lemma frontier_closed [iff]: closed (frontier S)

by (simp add: frontier_def closed_Diff)

lemma frontier_closures: frontier S = closure S \cap closure (– S)

by (auto simp: frontier_def interior_closure)

lemma frontier_Int: frontier(S \cap T) = closure(S \cap T) \cap (frontier S \cup frontier T)

proof –

have $\text{closure } (S \cap T) \subseteq \text{closure } S \text{ closure } (S \cap T) \subseteq \text{closure } T$
by (*simp_all add: closure_mono*)
then show *?thesis*
by (*auto simp: frontier_closures*)

qed

lemma *frontier_Int_subset*: $\text{frontier}(S \cap T) \subseteq \text{frontier } S \cup \text{frontier } T$
by (*auto simp: frontier_Int*)

lemma *frontier_Int_closed*:
assumes *closed S closed T*
shows $\text{frontier}(S \cap T) = (\text{frontier } S \cap T) \cup (S \cap \text{frontier } T)$
by (*smt (verit, best) Diff_Int Int_Diff assms closed_Int closure_eq frontier_def inf_commute interior_Int*)

lemma *frontier_subset_closed*: $\text{closed } S \implies \text{frontier } S \subseteq S$
by (*metis frontier_def closure_closed Diff_subset*)

lemma *frontier_empty* [*simp*]: $\text{frontier } \{\} = \{\}$
by (*simp add: frontier_def*)

lemma *frontier_subset_eq*: $\text{frontier } S \subseteq S \iff \text{closed } S$
by (*metis Diff_subset_conv closure_subset_eq frontier_def interior_subset subset_Un_eq*)

lemma *frontier_complement* [*simp*]: $\text{frontier } (- S) = \text{frontier } S$
by (*auto simp: frontier_def closure_complement interior_complement*)

lemma *frontier_Un_subset*: $\text{frontier}(S \cup T) \subseteq \text{frontier } S \cup \text{frontier } T$
by (*metis compl_sup frontier_Int_subset frontier_complement*)

lemma *frontier_disjoint_eq*: $\text{frontier } S \cap S = \{\} \iff \text{open } S$
using *frontier_complement frontier_subset_eq*[*of - S*]
unfolding *open_closed* **by** *auto*

lemma *frontier_UNIV* [*simp*]: $\text{frontier } \text{UNIV} = \{\}$
using *frontier_complement frontier_empty* **by** *fastforce*

lemma *frontier_interiors*: $\text{frontier } s = - \text{interior}(s) - \text{interior}(-s)$
by (*simp add: Int_commute frontier_def interior_closure*)

lemma *frontier_interior_subset*: $\text{frontier}(\text{interior } S) \subseteq \text{frontier } S$
by (*simp add: Diff_mono frontier_interiors interior_mono interior_subset*)

lemma *closure_Un_frontier*: $\text{closure } S = S \cup \text{frontier } S$
by (*simp add: closure_def frontier_closures sup_inf_distrib1*)

2.1.8 Filters and the “eventually true” quantifier

Identify Trivial limits, where we can’t approach arbitrarily closely.

lemma *trivial_limit_within*: $\text{trivial_limit } (at\ a\ \text{within } S) \longleftrightarrow \neg\ a\ \text{islimpt } S$
unfolding *trivial_limit_def eventually_at_topological_islimpt_def*
by *blast*

lemma *trivial_limit_at_iff*: $\text{trivial_limit } (at\ a) \longleftrightarrow \neg\ a\ \text{islimpt } UNIV$
using *trivial_limit_within [of a UNIV]* **by** *simp*

lemma *trivial_limit_at*: $\neg\ \text{trivial_limit } (at\ a)$
for $a :: 'a::\text{perfect_space}$
by *(rule at_neq_bot)*

lemma *not_trivial_limit_within*: $\neg\ \text{trivial_limit } (at\ x\ \text{within } S) = (x \in \text{closure } (S - \{x\}))$
using *islimpt_in_closure* **by** *(metis trivial_limit_within)*

lemma *not_in_closure_trivial_limitI*:
 $x \notin \text{closure } S \implies \text{trivial_limit } (at\ x\ \text{within } S)$
using *not_trivial_limit_within[of x S]*
by *safe (metis Diff_empty Diff_insert0 closure_subset contra_subsetD)*

lemma *filterlim_at_within_closure_implies_filterlim*: $\text{filterlim } f\ l\ (at\ x\ \text{within } S)$
if $x \in \text{closure } S \implies \text{filterlim } f\ l\ (at\ x\ \text{within } S)$
by *(metis bot.extremum filterlim_iff_le_filtercomap not_in_closure_trivial_limitI that)*

lemma *at_within_eq_bot_iff*: $at\ c\ \text{within } A = \text{bot} \longleftrightarrow c \notin \text{closure } (A - \{c\})$
using *not_trivial_limit_within[of c A]* **by** *blast*

2.1.9 Limits

The expected monotonicity property.

lemma *Lim_Un*:
assumes $(f \longrightarrow l)\ (at\ x\ \text{within } S)\ (f \longrightarrow l)\ (at\ x\ \text{within } T)$
shows $(f \longrightarrow l)\ (at\ x\ \text{within } (S \cup T))$
using *assms unfolding at_within_union* **by** *(rule filterlim_sup)*

lemma *Lim_Un_univ*:
 $(f \longrightarrow l)\ (at\ x\ \text{within } S) \implies (f \longrightarrow l)\ (at\ x\ \text{within } T) \implies$
 $S \cup T = UNIV \implies (f \longrightarrow l)\ (at\ x)$
by *(metis Lim_Un)*

Interrelations between restricted and unrestricted limits.

lemma *Lim_at_imp_Lim_at_within*: $(f \longrightarrow l)\ (at\ x) \implies (f \longrightarrow l)\ (at\ x\ \text{within } S)$
by *(metis order_refl filterlim_mono subset_UNIV at_le)*

lemma *eventually_within_interior*:

assumes $x \in \text{interior } S$

shows $\text{eventually } P \text{ (at } x \text{ within } S) \longleftrightarrow \text{eventually } P \text{ (at } x)$

by (*metis* *assms* *at_within_open_subset* *interior_subset* *open_interior*)

lemma *at_within_interior*: $x \in \text{interior } S \implies \text{at } x \text{ within } S = \text{at } x$

unfolding *filter_eq_iff* **by** (*intro* *allI* *eventually_within_interior*)

lemma *Lim_within_LIMSEQ*:

fixes $a :: 'a :: \text{first_countable_topology}$

assumes $\forall S. (\forall n. S \ n \neq a \wedge S \ n \in T) \wedge S \longrightarrow a \longrightarrow (\lambda n. X \ (S \ n)) \longrightarrow$

L

shows $(X \longrightarrow L) \text{ (at } a \text{ within } T)$

using *assms* **unfolding** *tendsto_def* [**where** $l=L$]

by (*simp* *add*: *sequentially_imp_eventually_within*)

lemma *Lim_right_bound*:

fixes $f :: 'a :: \{\text{linorder_topology, conditionally_complete_linorder, no_top}\} \Rightarrow$

$'b :: \{\text{linorder_topology, conditionally_complete_linorder}\}$

assumes *mono*: $\bigwedge a \ b. a \in I \implies b \in I \implies x < a \implies a \leq b \implies f \ a \leq f \ b$

and *bnd*: $\bigwedge a. a \in I \implies x < a \implies K \leq f \ a$

shows $(f \longrightarrow \text{Inf } (f \ ' (\{x<..\} \cap I))) \text{ (at } x \text{ within } (\{x<..\} \cap I))$

proof (*cases* $\{x<..\} \cap I = \{\}$)

case *True*

then show *?thesis* **by** *simp*

next

case *False*

show *?thesis*

proof (*rule* *order_tendstoI*)

fix a

assume $a < \text{Inf } (f \ ' (\{x<..\} \cap I))$

{

fix y

assume $y \in \{x<..\} \cap I$

with *False* *bnd* **have** $\text{Inf } (f \ ' (\{x<..\} \cap I)) \leq f \ y$

by (*auto* *intro*!: *cInf_lower* *bdd_belowI2*)

with a **have** $a < f \ y$

by (*blast* *intro*: *less_le_trans*)

}

then show $\text{eventually } (\lambda x. a < f \ x) \text{ (at } x \text{ within } (\{x<..\} \cap I))$

by (*auto* *simp*: *eventually_at_filter* *intro*: *exI*[*of* $_ 1$] *zero_less_one*)

next

fix a

assume $\text{Inf } (f \ ' (\{x<..\} \cap I)) < a$

from *cInf_lessD*[*OF* $_ \text{this}$] *False* **obtain** y **where** $x < y \ y \in I \ f \ y < a$

by *auto*

then have $\text{eventually } (\lambda x. x \in I \longrightarrow f \ x < a) \text{ (at_right } x)$

unfolding *eventually_at_right*[*OF* $\langle x < y \rangle$] **by** (*metis* *less_imp_le* *le_less_trans*)

```

mono)
  then show eventually ( $\lambda x. f x < a$ ) (at  $x$  within ( $\{x < ..\} \cap I$ ))
    unfolding eventually_at_filter by eventually_elim simp
  qed
qed

```

These are special for limits out of the same topological space.

```

lemma Lim_within_id: ( $id \longrightarrow a$ ) (at  $a$  within  $s$ )
  unfolding id_def by (rule tendsto_ident_at)

```

```

lemma Lim_at_id: ( $id \longrightarrow a$ ) (at  $a$ )
  unfolding id_def by (rule tendsto_ident_at)

```

It's also sometimes useful to extract the limit point from the filter.

```

abbreviation netlimit :: ' $a::t2\_space$  filter  $\Rightarrow$  ' $a$ 
  where netlimit  $F \equiv Lim F (\lambda x. x)$ 

```

```

lemma netlimit_at [simp]:
  fixes  $a :: 'a::\{perfect\_space,t2\_space\}$ 
  shows netlimit (at  $a$ ) =  $a$ 
  using Lim_ident_at [of  $a UNIV$ ] by simp

```

```

lemma lim_within_interior:
   $x \in interior S \implies (f \longrightarrow l)$  (at  $x$  within  $S$ )  $\iff (f \longrightarrow l)$  (at  $x$ )
  by (metis at_within_interior)

```

```

lemma netlimit_within_interior:
  fixes  $x :: 'a::\{t2\_space,perfect\_space\}$ 
  assumes  $x \in interior S$ 
  shows netlimit (at  $x$  within  $S$ ) =  $x$ 
  using assms by (metis at_within_interior netlimit_at)

```

Useful lemmas on closure and set of possible sequential limits.

```

lemma closure_sequential:
  fixes  $l :: 'a::first\_countable\_topology$ 
  shows  $l \in closure S \iff (\exists x. (\forall n. x n \in S) \wedge (x \longrightarrow l)$  sequentially)
  by (metis Diff_empty Diff_insert0 Un_iff closure_def islimpt_sequential mem_Collect_eq tendsto_const)

```

```

lemma closed_sequential_limits:
  fixes  $S :: 'a::first\_countable\_topology$  set
  shows closed  $S \iff (\forall x l. (\forall n. x n \in S) \wedge (x \longrightarrow l)$  sequentially  $\implies l \in S$ )
  by (metis closure_sequential closure_subset_eq subset_iff)

```

```

lemma tendsto_If_within_closures:
  assumes  $f: x \in S \cup (closure S \cap closure T) \implies$ 
    ( $f \longrightarrow l x$ ) (at  $x$  within  $S \cup (closure S \cap closure T)$ )
  assumes  $g: x \in T \cup (closure S \cap closure T) \implies$ 
    ( $g \longrightarrow l x$ ) (at  $x$  within  $T \cup (closure S \cap closure T)$ )

```

```

  assumes  $x \in S \cup T$ 
  shows  $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } g x) \longrightarrow l x)$  (at  $x$  within  $S \cup T$ )
proof -
  have  $*(S \cup T) \cap \{x. x \in S\} = S$   $(S \cup T) \cap \{x. x \notin S\} = T - S$ 
  by auto
  have  $(f \longrightarrow l x)$  (at  $x$  within  $S$ )
  by (rule filterlim_at_within_closure_implies_filterlim)
  (use  $\langle x \in \_ \rangle$  in  $\langle$ auto simp: inf_commute closure_def intro: tendsto_within_subset[OF
f] $\rangle$ )
  moreover
  have  $(g \longrightarrow l x)$  (at  $x$  within  $T - S$ )
  by (rule filterlim_at_within_closure_implies_filterlim)
  (use  $\langle x \in \_ \rangle$  in
     $\langle$ auto intro!: tendsto_within_subset[OF g] simp: closure_def intro: is-
limpt_subset $\rangle$ )
  ultimately show ?thesis
  by (intro filterlim_at_within_If) (simp_all only: *)
qed

```

2.1.10 Compactness

```

lemma brouwer_compactness_lemma:
  fixes  $f :: 'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes compact  $S$ 
  and continuous_on  $S f$ 
  and  $\neg (\exists x \in S. f x = 0)$ 
  obtains  $d$  where  $0 < d$  and  $\forall x \in S. d \leq \text{norm } (f x)$ 
proof (cases  $S = \{\}$ )
case True
show thesis
  by (rule that [of 1]) (auto simp: True)
next
case False
have continuous_on  $S$  (norm  $\circ f$ )
  by (rule continuous_intros continuous_on_norm assms(2))+
with False obtain  $x$  where  $x: x \in S \ \forall y \in S. (\text{norm} \circ f) x \leq (\text{norm} \circ f) y$ 
using continuous_attains_inf[OF assms(1), of norm  $\circ f$ ]
unfolding o_def
  by auto
then show ?thesis
  by (metis assms(3) that comp_apply zero_less_norm_iff)
qed

```

Bolzano-Weierstrass property

```

proposition Heine_Borel_imp_Bolzano_Weierstrass:
  assumes compact  $S$ 
  and infinite  $T$ 
  and  $T \subseteq S$ 
  shows  $\exists x \in S. x \text{ islimpt } T$ 

```

proof (*rule ccontr*)
assume $\neg (\exists x \in S. x \text{ islimpt } T)$
then obtain f **where** $f: \forall x \in S. x \in f x \wedge \text{open } (f x) \wedge (\forall y \in T. y \in f x \longrightarrow y = x)$
unfolding *islimpt_def* **by** *metis*
obtain g **where** $g: g \subseteq \{T. \exists x. x \in S \wedge T = f x\}$ *finite* $g \subseteq \bigcup g$
using *assms(1)[unfolded compact_eq_Heine_Borel, THEN spec[where x={T. $\exists x. x \in S \wedge T = f x$ }]]*
using f **by** *auto*
then have $g': \forall x \in g. \exists y \in S. x = f y$
by *auto*
have *inj_on* f T
by (*smt* (*verit*, *best*) *assms(3)* f *inj_onI* *subsetD*)
then have *infinite* ($f' T$)
using *assms(2)* **using** *finite_imageD* **by** *auto*
moreover
have *False* **if** $x \in T$ $f x \notin g$ **for** x
by (*smt* (*verit*) *UnionE* *assms(3)* f g' $g(3)$ *subsetD* *that*)
then have $f' T \subseteq g$ **by** *auto*
ultimately show *False*
using $g(2)$ **using** *finite_subset* **by** *auto*
qed

lemma *sequence_infinite_lemma*:
fixes $f :: \text{nat} \Rightarrow 'a::t1_space$
assumes $\forall n. f n \neq l$
and ($f \longrightarrow l$) *sequentially*
shows *infinite* (*range* f)
proof
assume *finite* (*range* f)
then have $l \notin \text{range } f \wedge \text{closed } (\text{range } f)$
using $\langle \text{finite } (\text{range } f) \rangle$ *assms(1)* *finite_imp_closed* **by** *blast*
then have *eventually* ($\lambda n. f n \in - \text{range } f$) *sequentially*
by (*metis* *Compl_iff* *assms(2)* *open_Compl* *topological_tendstoD*)
then show *False*
unfolding *eventually_sequentially* **by** *auto*
qed

lemma *Bolzano_Weierstrass_imp_closed*:
fixes $S :: 'a::\{\text{first_countable_topology}, t2_space\}$ *set*
assumes $\forall T. \text{infinite } T \wedge T \subseteq S \longrightarrow (\exists x \in S. x \text{ islimpt } T)$
shows *closed* S
proof –
{
fix x l
assume *as*: $\forall n::\text{nat}. x n \in S$ ($x \longrightarrow l$) *sequentially*
have $l \in S$
proof (*cases* $\forall n. x n \neq l$)
case *False*

```

    then show  $l \in S$  using as(1) by auto
  next
  case True
  with as(2) have infinite (range x)
    using sequence_infinite_lemma[of x l] by auto
  then obtain l' where  $l' \in S$  l' islimpt (range x)
    using as(1) assms by auto
  then show  $l \in S$  using sequence_unique_limpt as True by auto
  qed
}
then show ?thesis
  unfolding closed_sequential_limits by fast
qed

```

```

lemma closure_insert:
  fixes  $x :: 'a::t1\_space$ 
  shows  $\text{closure} (\text{insert } x \ S) = \text{insert } x (\text{closure } S)$ 
  by (metis closed_singleton closure_Un closure_closed insert_is_Un)

```

```

lemma finite_not_islimpt_in_compact:
  assumes  $\text{compact } A \wedge z. z \in A \implies \neg z \text{ islimpt } B$ 
  shows finite ( $A \cap B$ )
  by (meson Heine_Borel_imp_Bolzano_Weierstrass assms inf_le1 inf_le2 islimpt_subset)

```

In particular, some common special cases.

```

lemma compact_Un [intro]:
  assumes compact S
  and compact T
  shows compact ( $S \cup T$ )
proof (rule compactI)
  fix f
  assume *: Ball f open  $S \cup T \subseteq \bigcup f$ 
  from *  $\langle \text{compact } S \rangle$  obtain s' where  $s' \subseteq f \wedge \text{finite } s' \wedge S \subseteq \bigcup s'$ 
  unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  moreover
  from *  $\langle \text{compact } T \rangle$  obtain t' where  $t' \subseteq f \wedge \text{finite } t' \wedge T \subseteq \bigcup t'$ 
  unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  ultimately show  $\exists f' \subseteq f. \text{finite } f' \wedge S \cup T \subseteq \bigcup f'$ 
  by (auto intro!: exI[of _ s' \cup t'])
qed

```

```

lemma compact_Union [intro]:  $\text{finite } S \implies (\bigwedge T. T \in S \implies \text{compact } T) \implies \text{compact} (\bigcup S)$ 
  by (induct set: finite) auto

```

```

lemma compact_UN [intro]:
   $\text{finite } A \implies (\bigwedge x. x \in A \implies \text{compact } (B \ x)) \implies \text{compact} (\bigcup_{x \in A} B \ x)$ 
  by blast

```

```

lemma closed_Int_compact [intro]:
  assumes closed S and compact T
  shows compact (S ∩ T)
  using compact_Int_closed [of T S] assms
  by (simp add: Int_commute)

```

```

lemma compact_Int [intro]:
  fixes S T :: 'a :: t2_space set
  assumes compact S and compact T
  shows compact (S ∩ T)
  using assms by (intro compact_Int_closed compact_imp_closed)

```

```

lemma compact_sing [simp]: compact {a}
  unfolding compact_eq Heine_Borel by auto

```

```

lemma compact_insert [simp]:
  assumes compact S
  shows compact (insert x S)
  by (metis assms compact_Un compact_sing insert_is_Un)

```

```

lemma finite_imp_compact: finite S ⇒ compact S
  by (induct set: finite) simp_all

```

```

lemma open_delete:
  fixes S :: 'a::t1_space set
  shows open S ⇒ open (S - {x})
  by (simp add: open_Diff)

```

Compactness expressed with filters

```

lemma closure_iff_nhds_not_empty:
  x ∈ closure X ↔ (∀ A. ∀ S ⊆ A. open S → x ∈ S → X ∩ A ≠ {})
proof safe
  assume x: x ∈ closure X
  fix S A
  assume  $\S$ : open S x ∈ S X ∩ A = {} S ⊆ A
  then have x ∉ closure (-S)
    by (simp add: closed_open)
  with x have x ∈ closure X - closure (-S)
    by auto
  with  $\S$  show False
    by (metis Compl_iff Diff_iff closure_mono disjoint_iff subsetD subsetI)
next
  assume  $\forall A S. S \subseteq A \rightarrow \text{open } S \rightarrow x \in S \rightarrow X \cap A \neq \{\}$ 
  then show x ∈ closure X
    by (metis ComplI Compl_disjoint closure_interior interior_subset open_interior)
qed

```

```

lemma compact_filter:

```



```

  compact U  $\longleftrightarrow$  ( $\forall F. F \neq \text{bot} \longrightarrow \text{eventually } (\lambda x. x \in U) F \longrightarrow (\exists x \in U. \text{inf}$ 
  ( $\text{nhds } x) F \neq \text{bot}$ ))
proof (intro allI iffI impI compact_fip[THEN iffD2] notI)
  fix F
  assume compact U
  assume F: F  $\neq$  bot eventually ( $\lambda x. x \in U$ ) F
  then have U  $\neq$  {}
    by (auto simp: eventually_False)

define Z where Z = closure ' {A. eventually ( $\lambda x. x \in A$ ) F}
then have  $\forall z \in Z. \text{closed } z$ 
  by auto
moreover
have ev_Z:  $\bigwedge z. z \in Z \implies \text{eventually } (\lambda x. x \in z) F$ 
  unfolding Z_def by (auto elim: eventually_mono intro: subsetD[OF closure_subset])
have ( $\forall B \subseteq Z. \text{finite } B \longrightarrow U \cap \bigcap B \neq \{\}$ )
proof (intro allI impI)
  fix B assume finite B  $B \subseteq Z$ 
  with <finite B> ev_Z F(2) have eventually ( $\lambda x. x \in U \cap (\bigcap B)$ ) F
    by (auto simp: eventually_ball_finite_distrib eventually_conj_iff)
  with F show  $U \cap \bigcap B \neq \{\}$ 
    by (intro notI) (simp add: eventually_False)
qed
ultimately have  $U \cap \bigcap Z \neq \{\}$ 
  using <compact U> unfolding compact_fip by blast
then obtain x where  $x \in U$  and  $x: \bigwedge z. z \in Z \implies x \in z$ 
  by auto

have  $\bigwedge P. \text{eventually } P (\text{inf } (\text{nhds } x) F) \implies P \neq \text{bot}$ 
  unfolding eventually_inf eventually_nhds
proof safe
  fix P Q R S
  assume eventually R F open S  $x \in S$ 
  with open_Int_closure_eq_empty[of S {x. R x}] x
  have  $S \cap \{x. R x\} \neq \{\}$  by (auto simp: Z_def)
  moreover assume Ball S Q  $\forall x. Q x \wedge R x \longrightarrow \text{bot } x$ 
  ultimately show False by (auto simp: set_eq_iff)
qed
with <x  $\in$  U> show  $\exists x \in U. \text{inf } (\text{nhds } x) F \neq \text{bot}$ 
  by (metis eventually_bot)
next
fix A
assume A:  $\forall a \in A. \text{closed } a \forall B \subseteq A. \text{finite } B \longrightarrow U \cap \bigcap B \neq \{\}$   $U \cap \bigcap A = \{\}$ 
define F where F = (INF a  $\in$  insert U A. principal a)
have F  $\neq$  bot
  unfolding F_def
proof (rule INF_filter_not_bot)
  fix X

```

```

assume  $X: X \subseteq \text{insert } U \text{ } A \text{ finite } X$ 
with  $A(\emptyset)[\text{THEN spec, of } X - \{U\}]$  have  $U \cap \bigcap (X - \{U\}) \neq \{\}$ 
  by auto
with  $X$  show  $(\text{INF } a \in X. \text{principal } a) \neq \text{bot}$ 
  by (auto simp: INF_principal_finite principal_eq_bot_iff)
qed
moreover
have  $F \leq \text{principal } U$ 
  unfolding  $F\_def$  by auto
then have eventually  $(\lambda x. x \in U) F$ 
  by (auto simp: le_filter_def eventually_principal)
moreover
assume  $\forall F. F \neq \text{bot} \longrightarrow \text{eventually } (\lambda x. x \in U) F \longrightarrow (\exists x \in U. \text{inf } (\text{nhds } x) F \neq \text{bot})$ 
ultimately obtain  $x$  where  $x \in U$  and  $x: \text{inf } (\text{nhds } x) F \neq \text{bot}$ 
  by auto

{ fix  $V$  assume  $V \in A$ 
  then have  $F \leq \text{principal } V$ 
    unfolding  $F\_def$  by (intro INF_lower2[of V]) auto
  then have  $V: \text{eventually } (\lambda x. x \in V) F$ 
    by (auto simp: le_filter_def eventually_principal)
  have  $x \in \text{closure } V$ 
    unfolding closure_iff_nhds_not_empty
  proof (intro impI allI)
    fix  $S A$ 
    assume open  $S x \in S S \subseteq A$ 
    then have eventually  $(\lambda x. x \in A) (\text{nhds } x)$ 
      by (auto simp: eventually_nhds)
    with  $V$  have eventually  $(\lambda x. x \in V \cap A) (\text{inf } (\text{nhds } x) F)$ 
      by (auto simp: eventually_inf)
    with  $x$  show  $V \cap A \neq \{\}$ 
      by (auto simp del: Int_iff simp add: trivial_limit_def)
  qed
  then have  $x \in V$ 
    using  $\langle V \in A \rangle A(1)$  by simp
}
with  $\langle x \in U \rangle$  have  $x \in U \cap \bigcap A$  by auto
with  $\langle U \cap \bigcap A = \{\} \rangle$  show False by auto
qed

```

definition *countably_compact* :: $(\text{'a}::\text{topological_space}) \text{ set} \Rightarrow \text{bool}$ **where**
countably_compact $U \iff$
 $(\forall A. \text{countable } A \longrightarrow (\forall a \in A. \text{open } a) \longrightarrow U \subseteq \bigcup A$
 $\longrightarrow (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T)$

lemma *countably_compactE*:

assumes *countably_compact* s **and** $\forall t \in C. \text{open } t$ **and** $s \subseteq \bigcup C$ *countable* C
obtains C' **where** $C' \subseteq C$ **and** *finite* C' **and** $s \subseteq \bigcup C'$

using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *countably_compactI*:

assumes $\bigwedge C. \forall t \in C. \text{open } t \implies s \subseteq \bigcup C \implies \text{countable } C \implies (\exists C' \subseteq C. \text{finite } C' \wedge s \subseteq \bigcup C')$

shows *countably_compact* *s*

using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *compact_imp_countably_compact*: *compact* *U* \implies *countably_compact* *U*

by (*auto simp: compact_eq_Heine_Borel countably_compact_def*)

lemma *countably_compact_imp_compact*:

assumes *countably_compact* *U*

and *ccover*: *countable* *B* $\forall b \in B. \text{open } b$

and *basis*: $\bigwedge T x. \text{open } T \implies x \in T \implies x \in U \implies \exists b \in B. x \in b \wedge b \cap U \subseteq$

T

shows *compact* *U*

using $\langle \text{countably_compact } U \rangle$

unfolding *compact_eq_Heine_Borel countably_compact_def*

proof *safe*

fix *A*

assume *A*: $\forall a \in A. \text{open } a \implies U \subseteq \bigcup A$

assume ***: $\forall A. \text{countable } A \longrightarrow (\forall a \in A. \text{open } a) \longrightarrow U \subseteq \bigcup A \longrightarrow (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T)$

moreover **define** *C* **where** $C = \{b \in B. \exists a \in A. b \cap U \subseteq a\}$

ultimately **have** *countable* *C* $\forall a \in C. \text{open } a$

unfolding *C_def* **using** *ccover* **by** *auto*

moreover

have $\bigcup A \cap U \subseteq \bigcup C$

proof *safe*

fix *x a*

assume $x \in U \implies x \in a \implies a \in A$

with *basis*[*of a x*] *A* **obtain** *b* **where** $b \in B \implies x \in b \wedge b \cap U \subseteq a$

by *blast*

with $\langle a \in A \rangle$ **show** $x \in \bigcup C$

unfolding *C_def* **by** *auto*

qed

then **have** $U \subseteq \bigcup C$ **using** $\langle U \subseteq \bigcup A \rangle$ **by** *auto*

ultimately **obtain** *T* **where** $T \subseteq C \implies \text{finite } T \implies U \subseteq \bigcup T$

using *** **by** *metis*

then **have** $\forall t \in T. \exists a \in A. t \cap U \subseteq a$

by (*auto simp: C_def*)

then **obtain** *f* **where** $\forall t \in T. f t \in A \wedge t \cap U \subseteq f t$

unfolding *bchoice_iff Bex_def* **..**

with *T* **show** $\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T$

unfolding *C_def* **by** (*intro exI[of _ f`T]*) *fastforce*

qed

proposition *countably_compact_imp_compact_second_countable*:

$countably_compact\ U \implies compact\ (U :: 'a :: second_countable_topology\ set)$
proof (rule *countably_compact_imp_compact*)
fix T **and** $x :: 'a$
assume *open* $T\ x \in T$
from *topological_basisE*[*OF is_basis this*] **obtain** b **where**
 $b \in (SOME\ B.\ countable\ B \wedge topological_basis\ B)\ x \in b\ b \subseteq T.$
then show $\exists b \in SOME\ B.\ countable\ B \wedge topological_basis\ B.\ x \in b \wedge b \cap U \subseteq T$
by *blast*
qed (*insert countable_basis topological_basis_open*[*OF is_basis*], *auto*)

lemma *countably_compact_eq_compact*:
 $countably_compact\ U \longleftrightarrow compact\ (U :: 'a :: second_countable_topology\ set)$
using *countably_compact_imp_compact* *compact_imp_countably_compact*
by *blast*

Sequential compactness

definition *seq_compact* :: $'a::topological_space\ set \Rightarrow bool$ **where**
 $seq_compact\ S \longleftrightarrow$
 $(\forall f. (\forall n. f\ n \in S) \longrightarrow (\exists l \in S. \exists r::nat \Rightarrow nat. strict_mono\ r \wedge (f \circ r) \longrightarrow l))$

lemma *seq_compactI*:
assumes $\bigwedge f. \forall n. f\ n \in S \implies \exists l \in S. \exists r::nat \Rightarrow nat. strict_mono\ r \wedge (f \circ r) \longrightarrow l$
shows *seq_compact* S
unfolding *seq_compact_def* **using** *assms* **by** *fast*

lemma *seq_compactE*:
assumes *seq_compact* $S\ \forall n. f\ n \in S$
obtains $l\ r$ **where** $l \in S\ strict_mono\ (r :: nat \Rightarrow nat)\ (f \circ r) \longrightarrow l$
using *assms* **unfolding** *seq_compact_def* **by** *fast*

lemma *seq_compact_Int_closed*:
assumes *seq_compact* S **and** *closed* T
shows *seq_compact* $(S \cap T)$
proof (rule *seq_compactI*)
fix f **assume** $\forall n::nat. f\ n \in S \cap T$
hence $\forall n. f\ n \in S$ **and** $\forall n. f\ n \in T$
by *simp_all*
from $\langle seq_compact\ S \rangle$ **and** $\langle \forall n. f\ n \in S \rangle$
obtain $l\ r$ **where** $l \in S$ **and** $r: strict_mono\ r$ **and** $l: (f \circ r) \longrightarrow l$
by (rule *seq_compactE*)
from $\langle \forall n. f\ n \in T \rangle$ **have** $\forall n. (f \circ r)\ n \in T$
by *simp*
with $\langle l \in S \rangle$ **and** r **and** l **show** $\exists l \in S \cap T. \exists r. strict_mono\ r \wedge (f \circ r) \longrightarrow l$
by (*metis Int_iff* *closed_T* *closed_sequentially*)

qed

lemma seq_compact_closed_subset:

assumes closed S and $S \subseteq T$ and seq_compact T

shows seq_compact S

using assms seq_compact_Int_closed [of $T S$] by (simp add: Int_absorb1)

lemma seq_compact_imp_countably_compact:

fixes $U :: 'a :: first_countable_topology$ set

assumes seq_compact U

shows countably_compact U

proof (safe intro!: countably_compactI)

fix A

assume $A: \forall a \in A. \text{open } a \implies U \subseteq \bigcup A$ countable A

have subseq: $\bigwedge X. \text{range } X \subseteq U \implies \exists r x. x \in U \wedge \text{strict_mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge (X \circ r) \longrightarrow x$

using $\langle \text{seq_compact } U \rangle$ by (fastforce simp: seq_compact_def subset_eq)

show $\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T$

proof cases

assume finite A

with A show ?thesis by auto

next

assume infinite A

then have $A \neq \{\}$ by auto

show ?thesis

proof (rule ccontr)

assume $\neg (\exists T \subseteq A. \text{finite } T \wedge U \subseteq \bigcup T)$

then have $\forall T. \exists x. T \subseteq A \wedge \text{finite } T \implies (x \in U - \bigcup T)$

by auto

then obtain X' where $T: \bigwedge T. T \subseteq A \implies \text{finite } T \implies X' T \in U - \bigcup T$

by metis

define X where $X n = X' (\text{from_nat_into } A \text{ ' } \{.. n\})$ for n

have $X: \bigwedge n. X n \in U - (\bigcup_{i \leq n} \text{from_nat_into } A i)$

using $\langle A \neq \{\} \rangle$ unfolding X_def by (intro T) (auto intro: from_nat_into)

then have range $X \subseteq U$

by auto

with subseq[of X] obtain $r x$ where $x \in U$ and $r: \text{strict_mono } r (X \circ r) \longrightarrow x$

by auto

from $\langle x \in U \rangle \langle U \subseteq \bigcup A \rangle$ from_nat_into_surj[OF $\langle \text{countable } A \rangle$]

obtain n where $x \in \text{from_nat_into } A n$ by auto

with $r(2)$ $A(1)$ from_nat_into[OF $\langle A \neq \{\} \rangle$]

have eventually $(\lambda i. X (r i) \in \text{from_nat_into } A n)$ sequentially

unfolding tendsto_def by fastforce

then obtain N where $\bigwedge i. N \leq i \implies X (r i) \in \text{from_nat_into } A n$

by (auto simp: eventually_sequentially)

moreover from X have $\bigwedge i. n \leq r i \implies X (r i) \notin \text{from_nat_into } A n$

by auto

moreover from $\langle \text{strict_mono } r \rangle$ [THEN seq_suble, of max $n N$] have $\exists i. n$

```

≤ r i ∧ N ≤ i
  by (auto intro!: exI[of _ max n N])
  ultimately show False
    by auto
qed
qed
qed

```

```

lemma compact_imp_seq_compact:
  fixes U :: 'a :: first_countable_topology set
  assumes compact U
  shows seq_compact U
  unfolding seq_compact_def
proof safe
  fix X :: nat ⇒ 'a
  assume ∀ n. X n ∈ U
  then have eventually (λx. x ∈ U) (filtermap X sequentially)
    by (auto simp: eventually_filtermap)
  moreover
  have filtermap X sequentially ≠ bot
    by (simp add: trivial_limit_def eventually_filtermap)
  ultimately
  obtain x where x ∈ U and x: inf (nhds x) (filtermap X sequentially) ≠ bot (is
  ?F ≠ _)
    using ⟨compact U⟩ by (auto simp: compact_filter)

  from countable_basis_at_decseq[of x]
  obtain A where A:
    ∧ i. open (A i)
    ∧ i. x ∈ A i
    ∧ S. open S ⇒ x ∈ S ⇒ eventually (λi. A i ⊆ S) sequentially
  by blast
  define s where s n i = (SOME j. i < j ∧ X j ∈ A (Suc n)) for n i
  {
    fix n i
    have ∃ a. i < a ∧ X a ∈ A (Suc n)
    proof (rule ccontr)
      assume ¬ (∃ a > i. X a ∈ A (Suc n))
      then have ∧ a. Suc i ≤ a ⇒ X a ∉ A (Suc n)
        by auto
      then have eventually (λx. x ∉ A (Suc n)) (filtermap X sequentially)
        by (auto simp: eventually_filtermap eventually_sequentially)
      moreover have eventually (λx. x ∈ A (Suc n)) (nhds x)
        using A(1,2)[of Suc n] by (auto simp: eventually_nhds)
      ultimately have eventually (λx. False) ?F
        by (auto simp: eventually_inf)
      with x show False
        by (simp add: eventually_False)
    qed
  }

```

```

    then have  $i < s\ n\ i$   $X\ (s\ n\ i) \in A\ (Suc\ n)$ 
      unfolding  $s\_def$  by (auto intro: someI2_ex)
  }
  note  $s = this$ 
  define  $r$  where  $r = rec\_nat\ (s\ 0\ 0)\ s$ 
  have  $strict\_mono\ r$ 
    by (auto simp:  $r\_def\ s\ strict\_mono\_Suc\_iff$ )
  moreover
  have  $(\lambda n. X\ (r\ n)) \longrightarrow x$ 
  proof (rule topological_tendstoI)
    fix  $S$ 
    assume  $open\ S\ x \in S$ 
    with  $A(3)$  have  $eventually\ (\lambda i. A\ i \subseteq S)$  sequentially
      by auto
    moreover
    {
      fix  $i$ 
      assume  $Suc\ 0 \leq i$ 
      then have  $X\ (r\ i) \in A\ i$ 
        by (cases  $i$ ) (simp_all add:  $r\_def\ s$ )
    }
    then have  $eventually\ (\lambda i. X\ (r\ i) \in A\ i)$  sequentially
      by (auto simp: eventually_sequentially)
    ultimately show  $eventually\ (\lambda i. X\ (r\ i) \in S)$  sequentially
      by eventually_elim auto
  qed
  ultimately show  $\exists x \in U. \exists r. strict\_mono\ r \wedge (X \circ r) \longrightarrow x$ 
    using  $\langle x \in U \rangle$  by (auto simp: convergent_def comp_def)
  qed

lemma countably_compact_imp_acc_point:
  assumes  $countably\_compact\ S$ 
  and  $countable\ T$ 
  and  $infinite\ T$ 
  and  $T \subseteq S$ 
  shows  $\exists x \in S. \forall U. x \in U \wedge open\ U \longrightarrow infinite\ (U \cap T)$ 
  proof (rule ccontr)
    define  $C$  where  $C = (\lambda F. interior\ (F \cup (-\ T)))\ \{F. finite\ F \wedge F \subseteq T\}$ 
    note  $\langle countably\_compact\ S \rangle$ 
    moreover have  $\forall T \in C. open\ T$ 
      by (auto simp:  $C\_def$ )
    moreover
    assume  $\neg (\exists x \in S. \forall U. x \in U \wedge open\ U \longrightarrow infinite\ (U \cap T))$ 
    then have  $S: \bigwedge x. x \in S \implies \exists U. x \in U \wedge open\ U \wedge finite\ (U \cap T)$  by metis
    have  $S \subseteq \bigcup C$ 
      using  $\langle T \subseteq S \rangle$ 
      unfolding  $C\_def$ 
      apply (safe dest!:  $S$ )
      apply (rule_tac  $a=U \cap T$  in  $UN_I$ )

```

```

    apply (auto intro!: interiorI simp add: finite_subset)
  done
  moreover
  from ⟨countable T⟩ have countable C
    unfolding C_def by (auto intro: countable_Collect_finite_subset)
  ultimately
  obtain D where D ⊆ C finite D S ⊆ ⋃ D
    by (rule countably_compactE)
  then obtain E where E: E ⊆ {F. finite F ∧ F ⊆ T} finite E
    and S: S ⊆ (⋃ F∈E. interior (F ∪ (- T)))
    by (metis (lifting) finite_subset_image C_def)
  from S ⟨T ⊆ S⟩ have T ⊆ ⋃ E
    using interior_subset by blast
  moreover have finite (⋃ E)
    using E by auto
  ultimately show False using ⟨infinite T⟩
    by (auto simp: finite_subset)
qed

lemma countable_acc_point_imp_seq_compact:
  fixes S :: 'a::first_countable_topology set
  assumes ⋀T. [infinite T; countable T; T ⊆ S] ⇒ ∃x∈S. ∀U. x∈U ∧ open U
  → infinite (U ∩ T)
  shows seq_compact S
  unfolding seq_compact_def
proof (intro strip)
  fix f :: nat ⇒ 'a
  assume f: ∀n. f n ∈ S
  show ∃l∈S. ∃r. strict_mono r ∧ ((f ∘ r) → l) sequentially
  proof (cases finite (range f))
  case True
    obtain l where infinite {n. f n = f l}
      using pigeonhole_infinite[OF True] by auto
    then obtain r :: nat ⇒ nat where strict_mono r and fr: ∀n. f (r n) = f l
      using infinite_enumerate by blast
    then have strict_mono r ∧ (f ∘ r) → f l
      by (simp add: fr o_def)
    with f show ∃l∈S. ∃r. strict_mono r ∧ (f ∘ r) → l
      by auto
  case False
  next
  case False
    with f assms obtain l where l ∈ S ∀U. l∈U ∧ open U → infinite (U ∩
  range f)
    by (metis image_subset_iff uncountable_def)
    with ⟨l ∈ S⟩ show ∃l∈S. ∃r. strict_mono r ∧ ((f ∘ r) → l) sequentially
      by (meson acc_point_range_imp_convergent_subsequence)
  qed
qed

```


lemma *seq_compact_eq_countably_compact*:
fixes $U :: 'a :: \text{first_countable_topology_set}$
shows $\text{seq_compact } U \longleftrightarrow \text{countably_compact } U$
by (*metis countable_acc_point_imp_seq_compact countably_compact_imp_acc_point seq_compact_imp_countably_compact*)

lemma *seq_compact_eq_acc_point*:
fixes $S :: 'a :: \text{first_countable_topology_set}$
shows $\text{seq_compact } S \longleftrightarrow$
 $(\forall T. \text{infinite } T \wedge \text{countable } T \wedge T \subseteq S \longrightarrow (\exists x \in S. \forall U. x \in U \wedge \text{open } U \longrightarrow$
 $\text{infinite } (U \cap T)))$
by (*metis countable_acc_point_imp_seq_compact countably_compact_imp_acc_point seq_compact_imp_countably_compact*)

lemma *seq_compact_eq_compact*:
fixes $U :: 'a :: \text{second_countable_topology_set}$
shows $\text{seq_compact } U \longleftrightarrow \text{compact } U$
using *seq_compact_eq_countably_compact countably_compact_eq_compact* **by**
blast

proposition *Bolzano_Weierstrass_imp_seq_compact*:
fixes $S :: 'a :: \{\text{t1_space, first_countable_topology}\}$ *set*
shows $(\bigwedge T. [\text{infinite } T; T \subseteq S] \Longrightarrow \exists x \in S. x \text{ islimpt } T) \Longrightarrow \text{seq_compact } S$
by (*rule countable_acc_point_imp_seq_compact*) (*metis islimpt_eq_acc_point*)

2.1.11 Cartesian products

lemma *seq_compact_Times*:
assumes $\text{seq_compact } S \text{ seq_compact } T$
shows $\text{seq_compact } (S \times T)$
unfolding *seq_compact_def*
proof *clarify*
fix $h :: \text{nat} \Rightarrow 'a \times 'b$
assume $\forall n. h \ n \in S \times T$
then have $*$: $\bigwedge n. (\text{fst} \circ h) \ n \in S \ \bigwedge n. (\text{snd} \circ h) \ n \in T$
by (*simp_all add: mem_Times_iff*)
then obtain lS **and** $rS :: \text{nat} \Rightarrow \text{nat}$
where $lS \in S \ \text{strict_mono } rS$ **and** $lS: (\text{fst} \circ h \circ rS) \longrightarrow lS$
using *assms seq_compact_def* **by** *metis*
then obtain lT **and** $rT :: \text{nat} \Rightarrow \text{nat}$
where $lT \in T \ \text{strict_mono } rT$ **and** $lT: (\text{snd} \circ h \circ rS \circ rT) \longrightarrow lT$
using *assms seq_compact_def* $*$
by (*metis (mono_tags, lifting) comp_apply*)
have $\text{strict_mono } (rS \circ rT)$
by (*simp add: <strict_mono rS> <strict_mono rT> strict_mono_o*)
moreover have $(h \circ (rS \circ rT)) \longrightarrow (lS, lT)$
using *tendsto_Pair [OF LIMSEQ_subseq LIMSEQ [OF lS <strict_mono rT>]*
 $lT]$
by (*simp add: o_def*)

ultimately show $\exists l \in S \times T. \exists r. \text{strict_mono } r \wedge (h \circ r) \longrightarrow l$
using $\langle lS \in S \rangle \langle lT \in T \rangle$ **by** *blast*
qed

lemma *compact_Times*:
assumes *compact S compact T*
shows *compact (S × T)*
proof (*rule compactI*)
fix C
assume $C: \forall T \in C. \text{open } T \ S \times T \subseteq \bigcup C$
have $\forall x \in S. \exists A. \text{open } A \wedge x \in A \wedge (\exists D \subseteq C. \text{finite } D \wedge A \times T \subseteq \bigcup D)$
proof
fix x
assume $x \in S$
have $\forall y \in T. \exists A \ B \ C. C \in C \wedge \text{open } A \wedge \text{open } B \wedge x \in A \wedge y \in B \wedge A \times B \subseteq C$
by (*smt (verit, ccfv_threshold) C UnionE* $\langle x \in S \rangle$ *mem_Sigma_iff open_prod_def subsetD*)
then obtain $a \ b \ c$ **where** $b: \bigwedge y. y \in T \implies \text{open } (b \ y)$
and $c: \bigwedge y. y \in T \implies c \ y \in C \wedge \text{open } (a \ y) \wedge \text{open } (b \ y) \wedge x \in a \ y \wedge y \in b \ y \wedge a \ y \times b \ y \subseteq c \ y$
by *metis*
then have $\forall y \in T. \text{open } (b \ y) \ T \subseteq (\bigcup y \in T. b \ y)$ **by** *auto*
with *compactE_image[OF* $\langle \text{compact } T \rangle$ **obtain** D **where** $D: D \subseteq T \ \text{finite } D \ T \subseteq (\bigcup y \in D. b \ y)$
by *metis*
moreover from $D \ c$ **have** $(\bigcap y \in D. a \ y) \times T \subseteq (\bigcup y \in D. c \ y)$
by (*fastforce simp: subset_eq*)
ultimately show $\exists a. \text{open } a \wedge x \in a \wedge (\exists d \subseteq C. \text{finite } d \wedge a \times T \subseteq \bigcup d)$
using c **by** (*intro exI[of _ c'D] exI[of _ $\bigcap (a'D)$] conjI*) (*auto intro!: open_INT*)
qed
then obtain $a \ d$ **where** $a: \bigwedge x. x \in S \implies \text{open } (a \ x) \ S \subseteq (\bigcup x \in S. a \ x)$
and $d: \bigwedge x. x \in S \implies d \ x \subseteq C \wedge \text{finite } (d \ x) \wedge a \ x \times T \subseteq \bigcup (d \ x)$
unfolding *subset_eq UN_iff* **by** *metis*
moreover
from *compactE_image[OF* $\langle \text{compact } S \rangle$ a
obtain e **where** $e \subseteq S \ \text{finite } e$ **and** $S: S \subseteq (\bigcup x \in e. a \ x)$
by *auto*
moreover
have $S \times T \subseteq (\bigcup x \in e. \bigcup (d \ x))$
by (*smt (verit, del_insts) S SigmaE UN_iff d e(1) mem_Sigma_iff subset_eq*)
ultimately show $\exists C' \subseteq C. \text{finite } C' \wedge S \times T \subseteq \bigcup C'$
by (*intro exI[of _ $\bigcup x \in e. d \ x$]*) (*auto simp: subset_eq*)
qed

lemma *tube_lemma*:
assumes *compact K*

```

assumes open W
assumes {x0} × K ⊆ W
shows ∃X0. x0 ∈ X0 ∧ open X0 ∧ X0 × K ⊆ W
proof -
  {
    fix y assume y ∈ K
    then have (x0, y) ∈ W using assms by auto
    with ⟨open W⟩
    have ∃X0 Y. open X0 ∧ open Y ∧ x0 ∈ X0 ∧ y ∈ Y ∧ X0 × Y ⊆ W
      by (rule open_prod_elim) blast
  }
  then obtain X0 Y where
    *: ∀y ∈ K. open (X0 y) ∧ open (Y y) ∧ x0 ∈ X0 y ∧ y ∈ Y y ∧ X0 y × Y y
    ⊆ W
    by metis
  from * have ∀t∈Y ' K. open t K ⊆ ⋃(Y ' K) by auto
  with ⟨compact K⟩ obtain CC where CC: CC ⊆ Y ' K finite CC K ⊆ ⋃ CC
    by (meson compactE)
  then obtain c where c: ∧C. C ∈ CC ⇒ c C ∈ K ∧ C = Y (c C)
    by (force intro!: choice)
  with * CC show ?thesis
    by (force intro!: exI[where x=⋂ C∈CC. X0 (c C)])
qed

```

lemma *continuous_on_prod_compactE*:

fixes *fx*::'a::topological_space × 'b::topological_space ⇒ 'c::metric_space
and *e*::real

assumes *cont_fx*: continuous_on (U × C) *fx*

assumes *compact C*

assumes [*intro*]: x0 ∈ U

notes [*continuous_intros*] = continuous_on_compose2[OF *cont_fx*]

assumes *e* > 0

obtains X0 where x0 ∈ X0 open X0

∀x∈X0 ∩ U. ∀t ∈ C. dist (fx (x, t)) (fx (x0, t)) ≤ e

proof -

define *psi* where *psi* = (λ(x, t). dist (fx (x, t)) (fx (x0, t)))

define W0 where W0 = {(x, t) ∈ U × C. *psi* (x, t) < e}

have W0_eq: W0 = *psi* -' {..*e*} ∩ U × C

by (auto simp: vimage_def W0_def)

have open {..*e*} by simp

have continuous_on (U × C) *psi*

by (auto intro!: continuous_intros simp: *psi_def split_beta'*)

then obtain W where W: open W W ∩ U × C = W0 ∩ U × C

unfolding W0_eq

by (metis ⟨open {..*e*}⟩ continuous_on_open_invariant inf_right_idem)

have {x0} × C ⊆ W ∩ U × C

unfolding W

by (auto simp: W0_def *psi_def* ⟨0 < e⟩)

then have {x0} × C ⊆ W by blast

```

from tube_lemma[OF ‹compact C› ‹open W› this]
obtain X0 where X0: x0 ∈ X0 open X0 X0 × C ⊆ W
by blast

have ∀x∈X0 ∩ U. ∀t ∈ C. dist (fx (x, t)) (fx (x0, t)) ≤ e
proof safe
  fix x assume x: x ∈ X0 x ∈ U
  fix t assume t: t ∈ C
  have dist (fx (x, t)) (fx (x0, t)) = psi (x, t)
    by (auto simp: psi_def)
  also have psi (x, t) < e
    using W(2) W0_def X0(3) t x by fastforce
  finally show dist (fx (x,t)) (fx (x0,t)) ≤ e by simp
qed
from X0(1,2) this show ?thesis ..
qed

```

2.1.12 Continuity

```

lemma continuous_at_imp_continuous_within:
  continuous (at x) f ⇒ continuous (at x within s) f
unfolding continuous_within continuous_at using Lim_at_imp_Lim_at_within
by auto

```

```

lemma Lim_trivial_limit: trivial_limit net ⇒ (f ⟶ l) net
by simp

```

lemmas continuous_on = continuous_on_def — legacy theorem name

```

lemma continuous_within_subset:
  continuous (at x within S) f ⇒ t ⊆ S ⇒ continuous (at x within t) f
unfolding continuous_within by(metis tendsto_within_subset)

```

```

lemma continuous_on_interior:
  continuous_on S f ⇒ x ∈ interior S ⇒ continuous (at x) f
by (metis continuous_on_eq_continuous_at continuous_on_subset interiorE)

```

```

lemma continuous_on_eq:
  [[continuous_on S f; ∧x. x ∈ S ⇒ f x = g x]] ⇒ continuous_on S g
unfolding continuous_on_def tendsto_def eventually_at_topological
by simp

```

Characterization of various kinds of continuity in terms of sequences.

```

lemma continuous_within_sequentiallyI:
  fixes f :: 'a::{first_countable_topology, t2_space} ⇒ 'b::topological_space
  assumes ∧u::nat ⇒ 'a. u ⟶ a ⇒ (∀n. u n ∈ S) ⇒ (λn. f (u n)) ⟶
  f a
  shows continuous (at a within S) f
  using assms unfolding continuous_within tendsto_def[where l = f a]

```

by (auto intro!: sequentially_imp_eventually_within)

lemma *continuous_within_tendsto_compose*:

fixes $f :: 'a :: t2_space \Rightarrow 'b :: topological_space$

assumes f : *continuous* (at a within S) f

and *eventually* $(\lambda n. x\ n \in S) F (x \longrightarrow a) F$

shows $((\lambda n. f (x\ n)) \longrightarrow f\ a) F$

proof –

have $*$: *filterlim* x (*inf* (*nhds* a) (*principal* S)) F

by (*simp* *add*: *assms* *filterlim_inf* *filterlim_principal*)

show *?thesis*

using $*$ *f* *continuous_within* *filterlim_compose* *tendsto_at_within_iff_tendsto_nhds*

by *blast*

qed

lemma *continuous_within_tendsto_compose'*:

fixes $f :: 'a :: t2_space \Rightarrow 'b :: topological_space$

assumes *continuous* (at a within S) f

$\bigwedge n. x\ n \in S$

$(x \longrightarrow a) F$

shows $((\lambda n. f (x\ n)) \longrightarrow f\ a) F$

using *always_eventually* *assms* *continuous_within_tendsto_compose* **by** *blast*

lemma *continuous_within_sequentially*:

fixes $f :: 'a :: \{first_countable_topology, t2_space\} \Rightarrow 'b :: topological_space$

shows *continuous* (at a within S) $f \longleftrightarrow$

$(\forall x. (\forall n :: nat. x\ n \in S) \wedge (x \longrightarrow a) \textit{sequentially}$

$\longrightarrow ((f \circ x) \longrightarrow f\ a) \textit{sequentially})$

using *continuous_within_tendsto_compose'*[*of* a S f *sequentially*]

continuous_within_sequentiallyI[*of* a S f]

by (*auto* *simp*: *o_def*)

lemma *continuous_at_sequentiallyI*:

fixes $f :: 'a :: \{first_countable_topology, t2_space\} \Rightarrow 'b :: topological_space$

assumes $\bigwedge u. u \longrightarrow a \Longrightarrow (\lambda n. f (u\ n)) \longrightarrow f\ a$

shows *continuous* (at a) f

using *continuous_within_sequentiallyI*[*of* a *UNIV* f] *assms* **by** *auto*

lemma *continuous_at_sequentially*:

fixes $f :: 'a :: metric_space \Rightarrow 'b :: topological_space$

shows *continuous* (at a) $f \longleftrightarrow$

$(\forall x. (x \longrightarrow a) \textit{sequentially} \longrightarrow ((f \circ x) \longrightarrow f\ a) \textit{sequentially})$

using *continuous_within_sequentially*[*of* a *UNIV* f] **by** *simp*

lemma *continuous_on_sequentiallyI*:

fixes $f :: 'a :: \{first_countable_topology, t2_space\} \Rightarrow 'b :: topological_space$

assumes $\bigwedge u\ a. (\forall n. u\ n \in S) \Longrightarrow a \in S \Longrightarrow u \longrightarrow a \Longrightarrow (\lambda n. f (u\ n)) \longrightarrow f\ a$

shows *continuous_on* S f

using *assms* **unfolding** *continuous_on_eq_continuous_within*
using *continuous_within_sequentiallyI*[*of _ S f*] **by** *auto*

lemma *continuous_on_sequentially*:

fixes $f :: 'a::\{first_countable_topology, t2_space\} \Rightarrow 'b::topological_space$

shows *continuous_on S f* \longleftrightarrow

$(\forall x. \forall a \in S. (\forall n. x(n) \in S) \wedge (x \longrightarrow a) \text{ sequentially}$
 $\longrightarrow ((f \circ x) \longrightarrow f a) \text{ sequentially})$

by (*meson continuous_on_eq_continuous_within continuous_within_sequentially*)

Continuity in terms of open preimages.

lemma *continuous_at_open*:

continuous (at x) f $\longleftrightarrow (\forall t. \text{open } t \wedge f x \in t \longrightarrow (\exists S. \text{open } S \wedge x \in S \wedge (\forall x' \in S. (f x') \in t)))$

by (*metis UNIV_I continuous_within_topological*)

lemma *continuous_imp_tendsto*:

assumes *continuous (at x0) f* **and** $x \longrightarrow x0$

shows $(f \circ x) \longrightarrow (f x0)$

proof (*rule topological_tendstoI*)

fix S

assume *open S f x0* $\in S$

then obtain T **where** $T_def: \text{open } T \ x0 \in T \ \forall x \in T. f x \in S$

using *assms continuous_at_open* **by** *metis*

then have *eventually* $(\lambda n. x n \in T)$ *sequentially*

using *assms T_def* **by** (*auto simp: tendsto_def*)

then show *eventually* $(\lambda n. (f \circ x) n \in S)$ *sequentially*

using T_def **by** (*auto elim!: eventually_mono*)

qed

2.1.13 Homeomorphisms

definition *homeomorphism S T f g* \longleftrightarrow

$(\forall x \in S. (g(f x) = x)) \wedge (f ' S = T) \wedge \text{continuous_on } S f \wedge$
 $(\forall y \in T. (f(g y) = y)) \wedge (g ' T = S) \wedge \text{continuous_on } T g$

lemma *homeomorphismI* [*intro?*]:

assumes *continuous_on S f continuous_on T g*

$f ' S \subseteq T \ g ' T \subseteq S \ \wedge x. x \in S \Longrightarrow g(f x) = x \ \wedge y. y \in T \Longrightarrow f(g y) = y$

shows *homeomorphism S T f g*

using *assms* **by** (*force simp: homeomorphism_def*)

lemma *homeomorphism_translation*:

fixes $a :: 'a :: real_normed_vector$

shows *homeomorphism* $((+) a ' S) S ((+) (- a)) ((+) a)$

unfolding *homeomorphism_def* **by** (*auto simp: algebra_simps continuous_intros*)

lemma *homeomorphism_ident*: *homeomorphism T T* $(\lambda a. a)$ $(\lambda a. a)$

by (*rule homeomorphismI*) *auto*

lemma *homeomorphism_compose*:

assumes *homeomorphism* $S\ T\ f\ g$ *homeomorphism* $T\ U\ h\ k$
shows *homeomorphism* $S\ U\ (h\ o\ f)\ (g\ o\ k)$
using *assms*
unfolding *homeomorphism_def*
by (*intro conjI ballI continuous_on_compose*) (*auto simp: image_iff*)

lemma *homeomorphism_cong*:

homeomorphism $X'\ Y'\ f'\ g'$
if *homeomorphism* $X\ Y\ f\ g$ $X' = X\ Y' = Y \wedge x. x \in X \implies f'\ x = f\ x \wedge y. y \in Y \implies g'\ y = g\ y$
using *that* **by** (*auto simp add: homeomorphism_def*)

lemma *homeomorphism_empty* [*simp*]:

homeomorphism $\{\}\ \{\}\ f\ g$
unfolding *homeomorphism_def* **by** *auto*

lemma *homeomorphism_symD*: *homeomorphism* $S\ t\ f\ g \implies \text{homeomorphism } t\ S\ g\ f$

by (*simp add: homeomorphism_def*)

lemma *homeomorphism_sym*: *homeomorphism* $S\ t\ f\ g = \text{homeomorphism } t\ S\ g\ f$

by (*force simp: homeomorphism_def*)

lemma *continuous_on_translation_eq*:

fixes $g :: 'a :: \text{real_normed_vector} \Rightarrow 'b :: \text{real_normed_vector}$
shows *continuous_on* $A\ ((+)\ a\ o\ g) = \text{continuous_on } A\ g$

proof –

have $g: g = (\lambda x. -a + x) \circ ((\lambda x. a + x) \circ g)$

by (*rule ext*) *simp*

show *?thesis*

by (*metis (no_types, opaque_lifting) g continuous_on_compose homeomorphism_def homeomorphism_translation*)

qed

definition *homeomorphic* :: $'a::\text{topological_space set} \Rightarrow 'b::\text{topological_space set} \Rightarrow \text{bool}$

(**infixr** *homeomorphic* 60)

where $s\ \text{homeomorphic } t \equiv (\exists f\ g. \text{homeomorphism } s\ t\ f\ g)$

lemma *homeomorphic_empty* [*iff*]:

$S\ \text{homeomorphic } \{\} \longleftrightarrow S = \{\}\ \{\} \text{homeomorphic } S \longleftrightarrow S = \{\}$

by (*auto simp: homeomorphic_def homeomorphism_def*)

lemma *homeomorphic_refl*: $S\ \text{homeomorphic } S$

using *homeomorphic_def homeomorphism_ident* **by** *fastforce*

lemma *homeomorphic_sym*: $S\ \text{homeomorphic } T \longleftrightarrow T\ \text{homeomorphic } S$

unfolding *homeomorphic_def homeomorphism_def*
by *blast*

lemma *homeomorphic_trans* [*trans*]:
assumes *S homeomorphic T and T homeomorphic U*
shows *S homeomorphic U*
using *assms unfolding homeomorphic_def*
by (*metis homeomorphism_compose*)

lemma *homeomorphic_minimal*:
S homeomorphic T \longleftrightarrow
 $(\exists f g. (\forall x \in S. f(x) \in T \wedge (g(f(x)) = x)) \wedge$
 $(\forall y \in T. g(y) \in S \wedge (f(g(y)) = y)) \wedge$
 $continuous_on\ S\ f \wedge continuous_on\ T\ g)$
by (*smt (verit, ccfv_threshold) homeomorphic_def homeomorphismI homeomorphism_def image_eqI image_subset_iff*)

lemma *homeomorphicI* [*intro?*]:
 $\llbracket f ' S = T; g ' T = S;$
 $continuous_on\ S\ f; continuous_on\ T\ g;$
 $\bigwedge x. x \in S \implies g(f(x)) = x;$
 $\bigwedge y. y \in T \implies f(g(y)) = y \rrbracket \implies S\ homeomorphic\ T$
unfolding *homeomorphic_def homeomorphism_def* **by** *metis*

lemma *homeomorphism_of_subsets*:
 $\llbracket homeomorphism\ S\ T\ f\ g; S' \subseteq S; T'' \subseteq T; f ' S' = T'' \rrbracket$
 $\implies homeomorphism\ S'\ T''\ f\ g$
by (*smt (verit, del_insts) continuous_on_subset homeomorphismI homeomorphism_def imageE subset_eq*)

lemma *homeomorphism_apply1*: $\llbracket homeomorphism\ S\ T\ f\ g; x \in S \rrbracket \implies g(f\ x) = x$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_apply2*: $\llbracket homeomorphism\ S\ T\ f\ g; x \in T \rrbracket \implies f(g\ x) = x$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_image1*: $homeomorphism\ S\ T\ f\ g \implies f ' S = T$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_image2*: $homeomorphism\ S\ T\ f\ g \implies g ' T = S$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_cont1*: $homeomorphism\ S\ T\ f\ g \implies continuous_on\ S\ f$
by (*simp add: homeomorphism_def*)

lemma *homeomorphism_cont2*: $homeomorphism\ S\ T\ f\ g \implies continuous_on\ T\ g$
by (*simp add: homeomorphism_def*)

lemma *continuous_on_no_limpt*:
 $(\bigwedge x. \neg x \text{ islimpt } S) \implies \text{continuous_on } S f$
unfolding *continuous_on_def*
by (*metis UNIV_I empty_iff eventually_at_topological islimptE open_UNIV tendsto_def trivial_limit_within*)

lemma *continuous_on_finite*:
fixes $S :: 'a::t1_space \text{ set}$
shows $\text{finite } S \implies \text{continuous_on } S f$
by (*metis continuous_on_no_limpt islimpt_finite*)

lemma *homeomorphic_finite*:
fixes $S :: 'a::t1_space \text{ set}$ **and** $T :: 'b::t1_space \text{ set}$
assumes *finite T*
shows $S \text{ homeomorphic } T \iff \text{finite } S \wedge \text{finite } T \wedge \text{card } S = \text{card } T$ (**is** *?lhs = ?rhs*)
proof
assume $S \text{ homeomorphic } T$
with *assms show ?rhs*
by (*metis (full_types) card_image_le finite_imageI homeomorphic_def homeomorphism_def le_antisym*)
next
assume $R: ?rhs$
with *finite_same_card_bij obtain h where bij_betw h S T*
by *auto*
with R **show** *?lhs*
apply (*simp only: homeomorphic_def homeomorphism_def continuous_on_finite*)
by (*smt (verit, ccfv_SIG) bij_betw_imp_surj_on bij_betw_inv_into bij_betw_inv_into_left bij_betw_inv_into_right*)
qed

Relatively weak hypotheses if a set is compact.

lemma *homeomorphism_compact*:
fixes $f :: 'a::topological_space \Rightarrow 'b::t2_space$
assumes $\text{compact } S \text{ continuous_on } S f f^{-1} S = T \text{ inj_on } f S$
shows $\exists g. \text{homeomorphism } S T f g$
proof –
obtain g **where** $g: \forall x \in S. g(f x) = x \forall x \in T. f(g x) = x g^{-1} T = S$
using *assms the_inv_into_f_f* **by** *fastforce*
with *assms show ?thesis*
unfolding *homeomorphism_def homeomorphic_def* **by** (*metis continuous_on_inv*)
qed

lemma *homeomorphic_compact*:
fixes $f :: 'a::topological_space \Rightarrow 'b::t2_space$
shows $\text{compact } S \implies \text{continuous_on } S f \implies (f^{-1} S = T) \implies \text{inj_on } f S \implies S \text{ homeomorphic } T$
unfolding *homeomorphic_def* **by** (*metis homeomorphism_compact*)

Preservation of topological properties.

lemma *homeomorphic_compactness*: S homeomorphic $T \implies (\text{compact } S \longleftrightarrow \text{compact } T)$
unfolding *homeomorphic_def* *homeomorphism_def*
by (*metis compact_continuous_image*)

2.1.14 On Linorder Topologies

lemma *islimpt_greaterThanLessThan1*:
fixes $a b::'a::\{\text{linorder_topology, dense_order}\}$
assumes $a < b$
shows a *islimpt* $\{a <..
proof (*rule islimptI*)
fix T
assume *open* T $a \in T$
then obtain c **where** $c: a < c \{a <..
by (*meson assms open_right*)
with *assms* *dense*[*of a min c b*]
show $\exists y \in \{a <..
by (*metis atLeastLessThan_iff greaterThanLessThan_iff min_less_iff_conj not_le order.strict_implies_order subset_eq*)
qed$$$

lemma *islimpt_greaterThanLessThan2*:
fixes $a b::'a::\{\text{linorder_topology, dense_order}\}$
assumes $a < b$
shows b *islimpt* $\{a <..
proof (*rule islimptI*)
fix T
assume *open* T $b \in T$
from *open_left*[*OF this* $\langle a < b \rangle$]
obtain c **where** $c: c < b \{c <.. **by** *auto*
with *assms* *dense*[*of max a c b*]
show $\exists y \in \{a <..
by (*metis greaterThanAtMost_iff greaterThanLessThan_iff max_less_iff_conj not_le order.strict_implies_order subset_eq*)
qed$$$

lemma *closure_greaterThanLessThan[simp]*:
fixes $a b::'a::\{\text{linorder_topology, dense_order}\}$
shows $a < b \implies \text{closure } \{a <.. (**is** $_ \implies ?l = ?r$)
proof
have $?l \subseteq \text{closure } ?r$
by (*rule closure_mono*) *auto*
thus $\text{closure } \{a <.. **by** *simp*
qed (*auto simp: closure_def order_order_iff_strict islimpt_greaterThanLessThan1 islimpt_greaterThanLessThan2*)$$

lemma *closure_greaterThan[simp]*:
fixes $a b::'a::\{\text{no_top, linorder_topology, dense_order}\}$

```

  shows closure {a<..} = {a..}
proof -
  from gt_ex obtain b where a < b by auto
  hence {a<..} = {a<..b} ∪ {b..} by auto
  also have closure ... = {a..} using ⟨a < b⟩ unfolding closure_Un
  by auto
  finally show ?thesis .
qed

```

```

lemma closure_lessThan[simp]:
  fixes b::'a::{no_bot, linorder_topology, dense_order}
  shows closure {..b} = {..b}
proof -
  from lt_ex obtain a where a < b by auto
  hence {..b} = {a<..b} ∪ {..a} by auto
  also have closure ... = {..b} using ⟨a < b⟩ unfolding closure_Un
  by auto
  finally show ?thesis .
qed

```

```

lemma closure_atLeastLessThan[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a ..< b} = {a .. b}
proof -
  from assms have {a ..< b} = {a} ∪ {a <..b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
  by (auto simp: assms less_imp_le)
  finally show ?thesis .
qed

```

```

lemma closure_greaterThanAtMost[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a <..b} = {a .. b}
proof -
  from assms have {a <..b} = {b} ∪ {a <..b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
  by (auto simp: assms less_imp_le)
  finally show ?thesis .
qed

```

```

end
theory Abstract_Limits
  imports
    Abstract_Topology
begin

```

2.1.15 nhdsin and atin

definition $nhdsin :: 'a \text{ topology} \Rightarrow 'a \Rightarrow 'a \text{ filter}$
where $nhdsin \ X \ a =$
(if $a \in \text{topspace } X$ then $(\text{INF } S \in \{S. \text{openin } X \ S \wedge a \in S\}. \text{principal } S)$
else bot)

definition $atin_within :: ['a \text{ topology}, 'a, 'a \text{ set}] \Rightarrow 'a \text{ filter}$
where $atin_within \ X \ a \ S = \text{inf} \ (nhdsin \ X \ a) \ (\text{principal} \ (\text{topspace } X \cap S - \{a\}))$

abbreviation $atin :: 'a \text{ topology} \Rightarrow 'a \Rightarrow 'a \text{ filter}$
where $atin \ X \ a \equiv atin_within \ X \ a \ UNIV$

lemma $atin_def: atin \ X \ a = \text{inf} \ (nhdsin \ X \ a) \ (\text{principal} \ (\text{topspace } X - \{a\}))$
by $(\text{simp} \ \text{add}: atin_within_def)$

lemma $nhdsin_degenerate \ [simp]: a \notin \text{topspace } X \Longrightarrow nhdsin \ X \ a = \text{bot}$
and $atin_degenerate \ [simp]: a \notin \text{topspace } X \Longrightarrow atin \ X \ a = \text{bot}$
by $(\text{simp_all} \ \text{add}: nhdsin_def \ atin_def)$

lemma $eventually_nhdsin:$
 $eventually \ P \ (nhdsin \ X \ a) \longleftrightarrow a \notin \text{topspace } X \vee (\exists S. \text{openin } X \ S \wedge a \in S \wedge$
 $(\forall x \in S. P \ x))$

proof $(\text{cases } a \in \text{topspace } X)$

case $True$

hence $nhdsin \ X \ a = (\text{INF } S \in \{S. \text{openin } X \ S \wedge a \in S\}. \text{principal } S)$

by $(\text{simp} \ \text{add}: nhdsin_def)$

also have $eventually \ P \ \dots \longleftrightarrow (\exists S. \text{openin } X \ S \wedge a \in S \wedge (\forall x \in S. P \ x))$

using $True$ **by** $(\text{subst} \ \text{eventually_INF_base}) \ (\text{auto} \ \text{simp}: \text{eventually_principal})$

finally show $?thesis$ **using** $True$ **by** simp

qed $auto$

lemma $eventually_atin_within:$

$eventually \ P \ (atin_within \ X \ a \ S) \longleftrightarrow a \notin \text{topspace } X \vee (\exists T. \text{openin } X \ T \wedge a \in T \wedge$
 $\in T \wedge (\forall x \in T. x \in S \wedge x \neq a \longrightarrow P \ x))$

proof $(\text{cases } a \in \text{topspace } X)$

case $True$

hence $eventually \ P \ (atin_within \ X \ a \ S) \longleftrightarrow$

$(\exists T. \text{openin } X \ T \wedge a \in T \wedge$

$(\forall x \in T. x \in \text{topspace } X \wedge x \in S \wedge x \neq a \longrightarrow P \ x))$

by $(\text{simp} \ \text{add}: atin_within_def \ \text{eventually_inf_principal} \ \text{eventually_nhdsin})$

also have $\dots \longleftrightarrow (\exists T. \text{openin } X \ T \wedge a \in T \wedge (\forall x \in T. x \in S \wedge x \neq a \longrightarrow P \ x))$

using openin_subset **by** $(\text{intro} \ \text{ex_cong}) \ \text{auto}$

finally show $?thesis$ **by** $(\text{simp} \ \text{add}: True)$

qed $(\text{simp} \ \text{add}: atin_within_def)$

lemma $eventually_atin:$

$eventually \ P \ (atin \ X \ a) \longleftrightarrow a \notin \text{topspace } X \vee$

$(\exists U. \text{openin } X \ U \wedge a \in U \wedge (\forall x \in U - \{a\}. P \ x))$

by (auto simp add: eventually_atin_within)

lemma *nontrivial_limit_atin*:

atin X a ≠ bot \longleftrightarrow *a* \in *X derived_set_of topspace X*

proof

assume *L*: *atin X a ≠ bot*

then have *a* \in *topspace X*

by (*meson atin_degenerate*)

moreover have \neg *openin X {a}*

using *L* by (*auto simp: eventually_atin trivial_limit_eq*)

ultimately

show *a* \in *X derived_set_of topspace X*

by (*auto simp: derived_set_of_topspace*)

next

assume *a*: *a* \in *X derived_set_of topspace X*

show *atin X a ≠ bot*

proof

assume *atin X a = bot*

then have *eventually* (λ _. *False*) (*atin X a*)

by *simp*

then show *False*

by (*smt* (*verit*, *best*) *a eventually_atin in_derived_set_of insertE insert_Diff*)

qed

qed

lemma *eventually_atin_subtopology*:

assumes *a* \in *topspace X*

shows *eventually P (atin (subtopology X S) a)* \longleftrightarrow

(a \in *S* \longrightarrow (\exists *U*. *openin (subtopology X S) U* \wedge *a* \in *U* \wedge (\forall *x* \in *U* - {*a*}. *P x*)))

using *assms* **by** (*simp add: eventually_atin*)

lemma *eventually_within_imp*:

eventually P (atin_within X a S) \longleftrightarrow *eventually* (λ *x*. *x* \in *S* \longrightarrow *P x*) (*atin X a*)

by (*auto simp add: eventually_atin_within eventually_atin*)

lemma *atin_subtopology_within*:

assumes *a* \in *S*

shows *atin (subtopology X S) a* = *atin_within X a S*

proof –

have *eventually P (atin (subtopology X S) a)* \longleftrightarrow *eventually P (atin_within X a S)* **for** *P*

unfolding *eventually_atin eventually_atin_within openin_subtopology*

using *assms* **by** *auto*

then show *?thesis*

by (*meson le_filter_def order.eq_iff*)

qed

lemma *atin_subtopology_within_if*:

shows $\text{atin}(\text{subtopology } X S) a = (\text{if } a \in S \text{ then } \text{atin_within } X a S \text{ else bot})$
by (*simp add: atin_subtopology_within*)

lemma *trivial_limit_atpointof_within*:
 $\text{trivial_limit}(\text{atin_within } X a S) \longleftrightarrow$
 $(a \notin X \text{ derived_set_of } S)$
by (*auto simp: trivial_limit_def eventually_atin_within_in_derived_set_of*)

lemma *derived_set_of_trivial_limit*:
 $a \in X \text{ derived_set_of } S \longleftrightarrow \neg \text{trivial_limit}(\text{atin_within } X a S)$
by (*simp add: trivial_limit_atpointof_within*)

2.1.16 Limits in a topological space

definition *limitin* :: $'a \text{ topology} \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b \text{ filter} \Rightarrow \text{bool}$ **where**
 $\text{limitin } X f l F \equiv l \in \text{topspace } X \wedge (\forall U. \text{openin } X U \wedge l \in U \longrightarrow \text{eventually}$
 $(\lambda x. f x \in U) F)$

lemma *limit_within_subset*:
 $\llbracket \text{limitin } X f l (\text{atin_within } Y a S); T \subseteq S \rrbracket \Longrightarrow \text{limitin } X f l (\text{atin_within } Y a T)$
by (*smt (verit) eventually_atin_within limitin_def subset_eq*)

lemma *limitinD*: $\llbracket \text{limitin } X f l F; \text{openin } X U; l \in U \rrbracket \Longrightarrow \text{eventually } (\lambda x. f x \in U) F$
by (*simp add: limitin_def*)

lemma *limitin_canonical_iff* [*simp*]: $\text{limitin euclidean } f l F \longleftrightarrow (f \longrightarrow l) F$
by (*auto simp: limitin_def tendsto_def*)

lemma *limitin_topspace*: $\text{limitin } X f l F \Longrightarrow l \in \text{topspace } X$
by (*simp add: limitin_def*)

lemma *limitin_const_iff* [*simp*]: $\text{limitin } X (\lambda a. l) l F \longleftrightarrow l \in \text{topspace } X$
by (*simp add: limitin_def*)

lemma *limitin_const*: $\text{limitin euclidean } (\lambda a. l) l F$
by *simp*

lemma *limitin_eventually*:
 $\llbracket l \in \text{topspace } X; \text{eventually } (\lambda x. f x = l) F \rrbracket \Longrightarrow \text{limitin } X f l F$
by (*auto simp: limitin_def eventually_mono*)

lemma *limitin_subsequence*:
 $\llbracket \text{strict_mono } r; \text{limitin } X f l \text{ sequentially} \rrbracket \Longrightarrow \text{limitin } X (f \circ r) l \text{ sequentially}$
unfolding *limitin_def* **using** *eventually_subseq* **by** *fastforce*

lemma *limitin_subtopology*:
 $\text{limitin}(\text{subtopology } X S) f l F$

```

   $\longleftrightarrow l \in S \wedge \text{eventually } (\lambda a. f a \in S) F \wedge \text{limitin } X f l F \text{ (is ?lhs = ?rhs)}$ 
proof (cases  $l \in S \cap \text{topspace } X$ )
  case True
  show ?thesis
  proof
    assume L: ?lhs
    with True
    have  $\forall_F b \text{ in } F. f b \in \text{topspace } X \cap S$ 
      by (metis (no_types) limitin_def openin_topspace topspace_subtopology)
    with L show ?rhs
    apply (clarsimp simp add: limitin_def eventually_mono openin_subtopology_alt)
    apply (drule_tac  $x=S \cap U$  in spec, force simp: elim: eventually_mono)
    done
  next
    assume ?rhs
    then show ?lhs
      using eventually_elim2
      by (fastforce simp add: limitin_def openin_subtopology_alt)
  qed
qed (auto simp: limitin_def)

```

```

lemma limitin_canonical_iff_gen [simp]:
  assumes open S
  shows  $\text{limitin } (\text{top\_of\_set } S) f l F \longleftrightarrow (f \longrightarrow l) F \wedge l \in S$ 
  using assms by (auto simp: limitin_subtopology tendsto_def)

```

```

lemma limitin_sequentially:
   $\text{limitin } X S l \text{ sequentially} \longleftrightarrow$ 
   $l \in \text{topspace } X \wedge (\forall U. \text{openin } X U \wedge l \in U \longrightarrow (\exists N. \forall n. N \leq n \longrightarrow S n \in U))$ 
  by (simp add: limitin_def eventually_sequentially)

```

```

lemma limitin_sequentially_offset:
   $\text{limitin } X f l \text{ sequentially} \implies \text{limitin } X (\lambda i. f (i + k)) l \text{ sequentially}$ 
  unfolding limitin_sequentially
  by (metis add.commute le_add2 order_trans)

```

```

lemma limitin_sequentially_offset_rev:
  assumes  $\text{limitin } X (\lambda i. f (i + k)) l \text{ sequentially}$ 
  shows  $\text{limitin } X f l \text{ sequentially}$ 
proof -
  have  $\exists N. \forall n \geq N. f n \in U$  if  $U: \text{openin } X U l \in U$  for  $U$ 
  proof -
    obtain  $N$  where  $\bigwedge n. n \geq N \implies f (n + k) \in U$ 
    using assms  $U$  unfolding limitin_sequentially by blast
    then have  $\forall n \geq N+k. f n \in U$ 
    by (metis add_leD2 le_add_diff_inverse ordered_cancel_comm_monoid_diff_class.le_diff_conv2
      add.commute)
  qed

```

then show *?thesis ..*
qed
with *assms show ?thesis*
unfolding *limitin_sequentially*
by *simp*
qed

lemma *limitin_atin*:
 $limitin\ Y\ f\ y\ (atin\ X\ x) \longleftrightarrow$
 $y \in\ topspace\ Y \wedge$
 $(x \in\ topspace\ X$
 $\longrightarrow (\forall V.\ openin\ Y\ V \wedge y \in V$
 $\longrightarrow (\exists U.\ openin\ X\ U \wedge x \in U \wedge f\ ' (U - \{x\}) \subseteq V)))$
by (*auto simp: limitin_def eventually_atin image_subset_iff*)

lemma *limitin_atin_self*:
 $limitin\ Y\ f\ (f\ a)\ (atin\ X\ a) \longleftrightarrow$
 $f\ a \in\ topspace\ Y \wedge$
 $(a \in\ topspace\ X$
 $\longrightarrow (\forall V.\ openin\ Y\ V \wedge f\ a \in V$
 $\longrightarrow (\exists U.\ openin\ X\ U \wedge a \in U \wedge f\ ' U \subseteq V)))$
unfolding *limitin_atin* **by** *fastforce*

lemma *limitin_trivial*:
 $\llbracket trivial_limit\ F; y \in\ topspace\ X \rrbracket \Longrightarrow limitin\ X\ f\ y\ F$
by (*simp add: limitin_def*)

lemma *limitin_transform_eventually*:
 $\llbracket eventually\ (\lambda x.\ f\ x = g\ x)\ F; limitin\ X\ f\ l\ F \rrbracket \Longrightarrow limitin\ X\ g\ l\ F$
unfolding *limitin_def* **using** *eventually_elim2* **by** *fastforce*

lemma *continuous_map_limit*:
assumes *continuous_map X Y g* **and** *f: limitin X f l F*
shows *limitin Y (g o f) (g l) F*
proof –
have $g\ l \in\ topspace\ Y$
by (*meson assms continuous_map f image_eqI in_mono limitin_def*)
moreover
have $\bigwedge U.\ \llbracket \forall V.\ openin\ X\ V \wedge l \in V \longrightarrow (\forall_F\ x\ in\ F.\ f\ x \in V); openin\ Y\ U; g\ l \in U \rrbracket$
 $\Longrightarrow \forall_F\ x\ in\ F.\ g\ (f\ x) \in U$
using *assms eventually_mono*
by (*fastforce simp: limitin_def dest!: openin_continuous_map_preimage*)
ultimately show *?thesis*
using *f* **by** (*fastforce simp add: limitin_def*)
qed

2.1.17 Pointwise continuity in topological spaces

definition *topcontinuous_at* where

$$\begin{aligned} \text{topcontinuous_at } X \ Y \ f \ x \ \longleftrightarrow \\ x \in \text{topspace } X \ \wedge \\ f \in \text{topspace } X \ \rightarrow \ \text{topspace } Y \ \wedge \\ (\forall V. \text{openin } Y \ V \ \wedge \ f \ x \in V \\ \longrightarrow (\exists U. \text{openin } X \ U \ \wedge \ x \in U \ \wedge \ (\forall y \in U. f \ y \in V))) \end{aligned}$$

lemma *topcontinuous_at_atin*:

$$\begin{aligned} \text{topcontinuous_at } X \ Y \ f \ x \ \longleftrightarrow \\ x \in \text{topspace } X \ \wedge \\ f \in \text{topspace } X \ \rightarrow \ \text{topspace } Y \ \wedge \\ \text{limitin } Y \ f \ (f \ x) \ (\text{atin } X \ x) \end{aligned}$$

unfolding *topcontinuous_at_def*

by (*fastforce simp add: limitin_atin*)⁺

lemma *continuous_map_eq_topcontinuous_at*:

$$\begin{aligned} \text{continuous_map } X \ Y \ f \ \longleftrightarrow \ (\forall x \in \text{topspace } X. \text{topcontinuous_at } X \ Y \ f \ x) \\ (\text{is ?lhs} = \text{?rhs}) \end{aligned}$$

proof

assume *?lhs*

then show *?rhs*

by (*auto simp: continuous_map_def topcontinuous_at_def*)

next

assume *R: ?rhs*

then show *?lhs*

apply (*auto simp: continuous_map_def topcontinuous_at_def*)

by (*smt (verit, del_insts) mem_Collect_eq openin_subopen openin_subset subset_eq*)

qed

lemma *continuous_map_atin*:

$$\text{continuous_map } X \ Y \ f \ \longleftrightarrow \ (\forall x \in \text{topspace } X. \text{limitin } Y \ f \ (f \ x) \ (\text{atin } X \ x))$$

by (*auto simp: limitin_def topcontinuous_at_atin continuous_map_eq_topcontinuous_at*)

lemma *limitin_continuous_map*:

$$\llbracket \text{continuous_map } X \ Y \ f; a \in \text{topspace } X; f \ a = b \rrbracket \implies \text{limitin } Y \ f \ b \ (\text{atin } X \ a)$$

by (*auto simp: continuous_map_atin*)

lemma *limit_continuous_map_within*:

$$\llbracket \text{continuous_map } (\text{subtopology } X \ S) \ Y \ f; a \in S; a \in \text{topspace } X \rrbracket$$

$$\implies \text{limitin } Y \ f \ (f \ a) \ (\text{atin_within } X \ a \ S)$$

by (*metis Int_iff atin_subtopology_within continuous_map_atin topspace_subtopology*)

2.1.18 Combining theorems for continuous functions into the reals

lemma *continuous_map_canonical_const* [*continuous_intros*]:

$$\text{continuous_map } X \ \text{euclidean} \ (\lambda x. c)$$

by *simp*

lemma *continuous_map_real_mult* [*continuous_intros*]:
 $\llbracket \text{continuous_map } X \text{ euclideanreal } f; \text{ continuous_map } X \text{ euclideanreal } g \rrbracket$
 $\implies \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x * g x)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_pow* [*continuous_intros*]:
 $\text{continuous_map } X \text{ euclideanreal } f \implies \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x ^ n)$
 by (*induction n*) (*auto simp: continuous_map_real_mult*)

lemma *continuous_map_real_mult_left*:
 $\text{continuous_map } X \text{ euclideanreal } f \implies \text{continuous_map } X \text{ euclideanreal } (\lambda x. c * f x)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_left_eq*:
 $\text{continuous_map } X \text{ euclideanreal } (\lambda x. c * f x) \longleftrightarrow c = 0 \vee \text{continuous_map } X \text{ euclideanreal } f$
proof (*cases c = 0*)
 case *False*
 have $\text{continuous_map } X \text{ euclideanreal } (\lambda x. c * f x) \implies \text{continuous_map } X \text{ euclideanreal } f$
 apply (*frule continuous_map_real_mult_left [where c=inverse c]*)
 apply (*simp add: field_simps False*)
 done
 with *False* show ?thesis
 using *continuous_map_real_mult_left* by blast
qed *simp*

lemma *continuous_map_real_mult_right*:
 $\text{continuous_map } X \text{ euclideanreal } f \implies \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x * c)$
 by (*simp add: continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_right_eq*:
 $\text{continuous_map } X \text{ euclideanreal } (\lambda x. f x * c) \longleftrightarrow c = 0 \vee \text{continuous_map } X \text{ euclideanreal } f$
 by (*simp add: mult.commute flip: continuous_map_real_mult_left_eq*)

lemma *continuous_map_minus* [*continuous_intros*]:
 fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_vector}$
 shows $\text{continuous_map } X \text{ euclidean } f \implies \text{continuous_map } X \text{ euclidean } (\lambda x. - f x)$
 by (*simp add: continuous_map_atin tendsto_minus*)

lemma *continuous_map_minus_eq* [*simp*]:
 fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_vector}$

shows $\text{continuous_map } X \text{ euclidean } (\lambda x. - f x) \longleftrightarrow \text{continuous_map } X \text{ euclidean } f$
using $\text{continuous_map_minus add.inverse_inverse continuous_map_eq}$ **by** *fast-force*

lemma $\text{continuous_map_add}$ [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_vector}$
shows $\llbracket \text{continuous_map } X \text{ euclidean } f; \text{continuous_map } X \text{ euclidean } g \rrbracket \Longrightarrow$
 $\text{continuous_map } X \text{ euclidean } (\lambda x. f x + g x)$
by (*simp add: continuous_map_atin tendsto_add*)

lemma $\text{continuous_map_diff}$ [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_vector}$
shows $\llbracket \text{continuous_map } X \text{ euclidean } f; \text{continuous_map } X \text{ euclidean } g \rrbracket \Longrightarrow$
 $\text{continuous_map } X \text{ euclidean } (\lambda x. f x - g x)$
by (*simp add: continuous_map_atin tendsto_diff*)

lemma $\text{continuous_map_norm}$ [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_vector}$
shows $\text{continuous_map } X \text{ euclidean } f \Longrightarrow \text{continuous_map } X \text{ euclidean } (\lambda x.$
 $\text{norm}(f x))$
by (*simp add: continuous_map_atin tendsto_norm*)

lemma $\text{continuous_map_real_abs}$ [*continuous_intros*]:
 $\text{continuous_map } X \text{ euclideanreal } f \Longrightarrow \text{continuous_map } X \text{ euclideanreal } (\lambda x.$
 $\text{abs}(f x))$
by (*simp add: continuous_map_atin tendsto_rabs*)

lemma $\text{continuous_map_real_max}$ [*continuous_intros*]:
 $\llbracket \text{continuous_map } X \text{ euclideanreal } f; \text{continuous_map } X \text{ euclideanreal } g \rrbracket$
 $\Longrightarrow \text{continuous_map } X \text{ euclideanreal } (\lambda x. \text{max}(f x) (g x))$
by (*simp add: continuous_map_atin tendsto_max*)

lemma $\text{continuous_map_real_min}$ [*continuous_intros*]:
 $\llbracket \text{continuous_map } X \text{ euclideanreal } f; \text{continuous_map } X \text{ euclideanreal } g \rrbracket$
 $\Longrightarrow \text{continuous_map } X \text{ euclideanreal } (\lambda x. \text{min}(f x) (g x))$
by (*simp add: continuous_map_atin tendsto_min*)

lemma $\text{continuous_map_sum}$ [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{real_normed_vector}$
shows $\llbracket \text{finite } I; \bigwedge i. i \in I \Longrightarrow \text{continuous_map } X \text{ euclidean } (\lambda x. f x i) \rrbracket$
 $\Longrightarrow \text{continuous_map } X \text{ euclidean } (\lambda x. \text{sum}(f x) I)$
by (*simp add: continuous_map_atin tendsto_sum*)

lemma $\text{continuous_map_prod}$ [*continuous_intros*]:
 $\llbracket \text{finite } I;$
 $\bigwedge i. i \in I \Longrightarrow \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x i) \rrbracket$
 $\Longrightarrow \text{continuous_map } X \text{ euclideanreal } (\lambda x. \text{prod}(f x) I)$
by (*simp add: continuous_map_atin tendsto_prod*)

```

lemma continuous_map_real_inverse [continuous_intros]:
  [[continuous_map X euclideanreal f;  $\bigwedge x. x \in \text{topspace } X \implies f x \neq 0$ ]]
     $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. \text{inverse}(f x))$ 
  by (simp add: continuous_map_atin tendsto_inverse)

```

```

lemma continuous_map_real_divide [continuous_intros]:
  [[continuous_map X euclideanreal f; continuous_map X euclideanreal g;  $\bigwedge x. x \in \text{topspace } X \implies g x \neq 0$ ]]
     $\implies \text{continuous\_map } X \text{ euclideanreal } (\lambda x. f x / g x)$ 
  by (simp add: continuous_map_atin tendsto_divide)

```

end

2.2 Non-Denumerability of the Continuum

```

theory Continuum_Not_Denumerable
imports
  Complex_Main
  HOL-Library.Countable_Set
begin

```

2.2.1 Abstract

The following document presents a proof that the Continuum is uncountable. It is formalised in the Isabelle/Isar theorem proving system.

Theorem: The Continuum \mathbb{R} is not denumerable. In other words, there does not exist a function $f: \mathbb{N} \Rightarrow \mathbb{R}$ such that f is surjective.

Outline: An elegant informal proof of this result uses Cantor's Diagonalisation argument. The proof presented here is not this one.

First we formalise some properties of closed intervals, then we prove the Nested Interval Property. This property relies on the completeness of the Real numbers and is the foundation for our argument. Informally it states that an intersection of countable closed intervals (where each successive interval is a subset of the last) is non-empty. We then assume a surjective function $f: \mathbb{N} \Rightarrow \mathbb{R}$ exists and find a real x such that x is not in the range of f by generating a sequence of closed intervals then using the Nested Interval Property.

```

theorem real_non_denum:  $\nexists f :: \text{nat} \Rightarrow \text{real. surj } f$ 
proof

```

```

  assume  $\exists f :: \text{nat} \Rightarrow \text{real. surj } f$ 
  then obtain  $f :: \text{nat} \Rightarrow \text{real}$  where  $\text{surj } f ..$ 

```

First we construct a sequence of nested intervals, ignoring *range* f .

```

  have  $a < b \implies \exists ka kb. ka < kb \wedge \{ka..kb\} \subseteq \{a..b\} \wedge c \notin \{ka..kb\}$  for  $a b c :: \text{real}$ 

```

```

  by (auto simp add: not_le cong: conj_cong)
    (metis dense le_less_linear less_linear less_trans order_refl)
then obtain  $i\ j$  where  $ij$ :
   $a < b \implies i\ a\ b\ c < j\ a\ b\ c$ 
   $a < b \implies \{i\ a\ b\ c .. j\ a\ b\ c\} \subseteq \{a .. b\}$ 
   $a < b \implies c \notin \{i\ a\ b\ c .. j\ a\ b\ c\}$ 
for  $a\ b\ c :: real$ 
by metis

define  $ivl$  where  $ivl =$ 
   $rec\_nat\ (f\ 0 + 1, f\ 0 + 2)\ (\lambda n\ x.\ (i\ (fst\ x)\ (snd\ x)\ (f\ n), j\ (fst\ x)\ (snd\ x)\ (f\ n)))$ 
define  $I$  where  $I\ n = \{fst\ (ivl\ n) .. snd\ (ivl\ n)\}$  for  $n$ 

have  $ivl$  [simp]:
   $ivl\ 0 = (f\ 0 + 1, f\ 0 + 2)$ 
   $\bigwedge n.\ ivl\ (Suc\ n) = (i\ (fst\ (ivl\ n))\ (snd\ (ivl\ n))\ (f\ n), j\ (fst\ (ivl\ n))\ (snd\ (ivl\ n))\ (f\ n))$ 
unfolding  $ivl\_def$  by  $simp\_all$ 

This is a decreasing sequence of non-empty intervals.

have  $less: fst\ (ivl\ n) < snd\ (ivl\ n)$  for  $n$ 
by ( $induct\ n$ ) ( $auto\ intro!: ij$ )

have  $decseq\ I$ 
unfolding  $I\_def\ decseq\_Suc\_iff\ ivl\_fst\_conv\ snd\_conv$ 
by ( $intro\ ij\ allI\ less$ )

Now we apply the finite intersection property of compact sets.

have  $I\ 0 \cap (\bigcap i.\ I\ i) \neq \{\}$ 
proof ( $rule\ compact\_imp\_fip\_image$ )
  fix  $S :: nat\ set$ 
  assume  $fin: finite\ S$ 
  have  $\{\} \subset I\ (Max\ (insert\ 0\ S))$ 
  unfolding  $I\_def$  using  $less$  [of  $Max\ (insert\ 0\ S)$ ] by  $auto$ 
  also have  $I\ (Max\ (insert\ 0\ S)) \subseteq (\bigcap i \in insert\ 0\ S.\ I\ i)$ 
  using  $fin\ decseqD$  [OF  $\langle decseq\ I \rangle$ , of  $Max\ (insert\ 0\ S)$ ]
  by ( $auto\ simp: Max\_ge\_iff$ )
  also have  $(\bigcap i \in insert\ 0\ S.\ I\ i) = I\ 0 \cap (\bigcap i \in S.\ I\ i)$ 
  by  $auto$ 
  finally show  $I\ 0 \cap (\bigcap i \in S.\ I\ i) \neq \{\}$ 
  by  $auto$ 
qed ( $auto\ simp: I\_def$ )
then obtain  $x$  where  $x \in I\ n$  for  $n$ 
by  $blast$ 
moreover from  $\langle surj\ f \rangle$  obtain  $j$  where  $x = f\ j$ 
by  $blast$ 
ultimately have  $f\ j \in I\ (Suc\ j)$ 
by  $blast$ 

```

```

with ij(3)[OF less] show False
  unfolding I_def ivl fst_conv snd_conv by auto
qed

```

```

lemma uncountable_UNIV_real: uncountable (UNIV :: real set)
  using real_non_denum unfolding uncountable_def by auto

```

```

corollary complex_non_denum:  $\nexists f :: \text{nat} \Rightarrow \text{complex. surj } f$ 
  by (metis (full_types) Re_complex_of_real comp_surj real_non_denum surj_def)

```

```

lemma uncountable_UNIV_complex: uncountable (UNIV :: complex set)
  using complex_non_denum unfolding uncountable_def by auto

```

```

lemma bij_betw_open_intervals:
  fixes a b c d :: real
  assumes a < b c < d
  shows  $\exists f. \text{bij\_betw } f \{a < .. < b\} \{c < .. < d\}$ 
proof -
  define f where f a b c d x = (d - c)/(b - a) * (x - a) + c for a b c d x :: real
  {
    fix a b c d x :: real
    assume *: a < b c < d a < x x < b
    moreover from * have (d - c) * (x - a) < (d - c) * (b - a)
      by (intro mult_strict_left_mono) simp_all
    moreover from * have 0 < (d - c) * (x - a) / (b - a)
      by simp
    ultimately have f a b c d x < d c < f a b c d x
      by (simp_all add: f_def field_simps)
  }
  with assms have bij_betw (f a b c d) {a < .. < b} {c < .. < d}
  by (intro bij_betw_byWitness[where f'=f c d a b]) (auto simp: f_def)
  then show ?thesis by auto
qed

```

```

lemma bij_betw_tan: bij_betw tan  $\{-\pi/2 < .. < \pi/2\}$  UNIV
  using arctan_ubound by (intro bij_betw_byWitness[where f'=arctan]) (auto
  simp: arctan arctan_tan)

```

```

lemma uncountable_open_interval: uncountable  $\{a < .. < b\} \longleftrightarrow a < b$  for a b ::
  real
proof
  show a < b if uncountable  $\{a < .. < b\}$ 
    using uncountable_def that by force
  show uncountable  $\{a < .. < b\}$  if a < b
  proof -
    obtain f where bij_betw f  $\{a < .. < b\} \{-\pi/2 < .. < \pi/2\}$ 
      using bij_betw_open_intervals[OF  $\langle a < b \rangle$ , of  $-\pi/2 \pi/2$ ] by auto
    then show ?thesis
      by (metis bij_betw_tan uncountable_bij_betw uncountable_UNIV_real)
  qed

```

qed
qed

lemma *uncountable_half_open_interval_1*: *uncountable* $\{a..<b\} \longleftrightarrow a < b$ **for**
a b :: real
 apply *auto*
 using *atLeastLessThan_empty_iff*
 apply *fastforce*
 using *uncountable_open_interval* [of *a b*]
 apply (*metis countable_Un_iff ivl_disj_un_singleton(3)*)
 done

lemma *uncountable_half_open_interval_2*: *uncountable* $\{a<..b\} \longleftrightarrow a < b$ **for**
a b :: real
 apply *auto*
 using *atLeastLessThan_empty_iff*
 apply *fastforce*
 using *uncountable_open_interval* [of *a b*]
 apply (*metis countable_Un_iff ivl_disj_un_singleton(4)*)
 done

lemma *real_interval_avoid_countable_set*:
 fixes *a b :: real and A :: real set*
 assumes *a < b and countable A*
 shows $\exists x \in \{a<..<b\}. x \notin A$
proof –
 from $\langle \text{countable } A \rangle$ **have** *: *countable* $(A \cap \{a<..<b\})$
 by *auto*
 with $\langle a < b \rangle$ **have** $\neg \text{countable } \{a<..<b\}$
 by (*simp add: uncountable_open_interval*)
 with * **have** $A \cap \{a<..<b\} \neq \{a<..<b\}$
 by *auto*
 then have $A \cap \{a<..<b\} \subset \{a<..<b\}$
 by (*intro psubsetI*) *auto*
 then have $\exists x. x \in \{a<..<b\} - A \cap \{a<..<b\}$
 by (*rule psubset_imp_ex_mem*)
 then show *?thesis*
 by *auto*
qed

lemma *uncountable_closed_interval*: *uncountable* $\{a..b\} \longleftrightarrow a < b$ **for** *a b :: real*
 using *infinite_Icc_iff* **by** (*fastforce dest: countable_finite real_interval_avoid_countable_set*)

lemma *open_minus_countable*:
 fixes *S A :: real set*
 assumes *countable A S ≠ {} open S*
 shows $\exists x \in S. x \notin A$
proof –
 obtain *x* **where** $x \in S$

```

    using ⟨S ≠ {}⟩ by auto
  then obtain e where 0 < e {y. dist y x < e} ⊆ S
    using ⟨open S⟩ by (auto simp: open_dist_subset_eq)
  moreover have {y. dist y x < e} = {x - e <..< x + e}
    by (auto simp: dist_real_def)
  ultimately have uncountable (S - A)
    using uncountable_open_interval[of x - e x + e] ⟨countable A⟩
    by (intro uncountable_minus_countable) (auto dest: countable_subset)
  then show ?thesis
    unfolding uncountable_def by auto
qed

end

```

2.3 Abstract Topology 2

```

theory Abstract_Topology_2
  imports
    Elementary_Topology Abstract_Topology Continuum_Not_Denumerable
    HOL-Library.Indicator_Function
    HOL-Library.Equipollence
begin

```

Combination of Elementary and Abstract Topology

```

lemma approachable_lt_le2:
  (∃(d::real) > 0. ∀x. Q x ⟶ f x < d ⟶ P x) ⟷ (∃ d > 0. ∀x. f x ≤ d ⟶ Q
x ⟶ P x)
  by (meson dense_less_eq_real_def order_le_less_trans)

```

```

lemma triangle_lemma:
  fixes x y z :: real
  assumes x: 0 ≤ x
    and y: 0 ≤ y
    and z: 0 ≤ z
    and xy: x2 ≤ y2 + z2
  shows x ≤ y + z
proof -
  have y2 + z2 ≤ y2 + 2 * y * z + z2
    using z y by simp
  with xy have th: x2 ≤ (y + z)2
    by (simp add: power2_eq_square field_simps)
  from y z have yz: y + z ≥ 0
    by arith
  from power2_le_imp_le[OF th yz] show ?thesis .
qed

```

```

lemma isCont_indicator:
  fixes x :: 'a::t2_space
  shows isCont (indicator A :: 'a ⇒ real) x = (x ∉ frontier A)

```



```

proof auto
  fix x
  assume cts_at: isCont (indicator A :: 'a  $\Rightarrow$  real) x and fr: x  $\in$  frontier A
  with continuous_at_open have 1:  $\forall V::\text{real set. open } V \wedge \text{indicator } A \ x \in V$ 
   $\longrightarrow$ 
    ( $\exists U::'a \text{ set. open } U \wedge x \in U \wedge (\forall y \in U. \text{indicator } A \ y \in V)$ ) by auto
  show False
  proof (cases x  $\in$  A)
    assume x: x  $\in$  A
    hence indicator A x  $\in$  ( $\{0 < .. < 2\}$  :: real set) by simp
    with 1 obtain U where U: open U x  $\in$  U  $\forall y \in U. \text{indicator } A \ y \in$  ( $\{0 < .. < 2\}$ 
  :: real set)
    using open_greaterThanLessThan by metis
    hence  $\forall y \in U. \text{indicator } A \ y >$  (0::real)
    unfolding greaterThanLessThan_def by auto
    hence U  $\subseteq$  A using indicator_eq_0_iff by force
    hence x  $\in$  interior A using U interiorI by auto
    thus ?thesis using fr unfolding frontier_def by simp
  next
    assume x: x  $\notin$  A
    hence indicator A x  $\in$  ( $\{-1 < .. < 1\}$  :: real set) by simp
    with 1 obtain U where U: open U x  $\in$  U  $\forall y \in U. \text{indicator } A \ y \in$  ( $\{-1 < .. < 1\}$ 
  :: real set)
    using 1 open_greaterThanLessThan by metis
    hence  $\forall y \in U. \text{indicator } A \ y <$  (1::real)
    unfolding greaterThanLessThan_def by auto
    hence U  $\subseteq$  -A by auto
    hence x  $\in$  interior (-A) using U interiorI by auto
    thus ?thesis using fr interior_complement unfolding frontier_def by auto
  qed
next
  assume nfr: x  $\notin$  frontier A
  hence x  $\in$  interior A  $\vee$  x  $\in$  interior (-A)
  by (auto simp: frontier_def closure_interior)
  thus isCont ((indicator A)::'a  $\Rightarrow$  real) x
  proof
    assume int: x  $\in$  interior A
    then obtain U where U: open U x  $\in$  U U  $\subseteq$  A unfolding interior_def by
  auto
    hence  $\forall y \in U. \text{indicator } A \ y =$  (1::real) unfolding indicator_def by auto
    hence continuous_on U (indicator A) by (simp add: indicator_eq_1_iff)
    thus ?thesis using U continuous_on_eq_continuous_at by auto
  next
    assume ext: x  $\in$  interior (-A)
    then obtain U where U: open U x  $\in$  U U  $\subseteq$  -A unfolding interior_def by
  auto
    then have continuous_on U (indicator A)
    using continuous_on_topological by (auto simp: subset_iff)
    thus ?thesis using U continuous_on_eq_continuous_at by auto

```

qed
qed

lemma *islimpt_closure*:

$\llbracket S \subseteq T; \bigwedge x. \llbracket x \text{ islimpt } S; x \in T \rrbracket \implies x \in S \rrbracket \implies S = T \cap \text{closure } S$
using *closure_def* **by** *fastforce*

lemma *closedin_limpt*:

$\text{closedin } (\text{top_of_set } T) S \longleftrightarrow S \subseteq T \wedge (\forall x. x \text{ islimpt } S \wedge x \in T \longrightarrow x \in S)$

proof –

have $\bigwedge U x. \llbracket \text{closed } U; S = T \cap U; x \text{ islimpt } S; x \in T \rrbracket \implies x \in S$

by (*meson IntI closed_limpt inf_le2 islimpt_subset*)

then show *?thesis*

by (*metis closed_closure closedin_closed closedin_imp_subset islimpt_closure*)

qed

lemma *closedin_closed_eq*: $\text{closed } S \implies \text{closedin } (\text{top_of_set } S) T \longleftrightarrow \text{closed } T \wedge T \subseteq S$

by (*meson closedin_limpt closed_subset closedin_closed_trans*)

lemma *connected_closed_set*:

$\text{closed } S$

$\implies \text{connected } S \longleftrightarrow (\nexists A B. \text{closed } A \wedge \text{closed } B \wedge A \neq \{\} \wedge B \neq \{\} \wedge A \cup B = S \wedge A \cap B = \{\})$

unfolding *connected_closedin_eq closedin_closed_eq connected_closedin_eq* **by** *blast*

If a connected set is written as the union of two nonempty closed sets, then these sets have to intersect.

lemma *connected_as_closed_union*:

assumes $\text{connected } C \ C = A \cup B \ \text{closed } A \ \text{closed } B \ A \neq \{\} \ B \neq \{\}$

shows $A \cap B \neq \{\}$

by (*metis assms closed_Un connected_closed_set*)

lemma *closedin_subset_trans*:

$\text{closedin } (\text{top_of_set } U) S \implies S \subseteq T \implies T \subseteq U \implies$

$\text{closedin } (\text{top_of_set } T) S$

by (*meson closedin_limpt subset_iff*)

lemma *openin_subset_trans*:

$\text{openin } (\text{top_of_set } U) S \implies S \subseteq T \implies T \subseteq U \implies$

$\text{openin } (\text{top_of_set } T) S$

by (*auto simp: openin_open*)

lemma *closedin_compact*:

$\llbracket \text{compact } S; \text{closedin } (\text{top_of_set } S) T \rrbracket \implies \text{compact } T$

by (*metis closedin_closed compact_Int_closed*)

lemma *closedin_compact_eq*:

```

fixes  $S :: 'a::t2\_space\ set$ 
shows  $compact\ S \implies (closedin\ (top\_of\_set\ S)\ T \longleftrightarrow compact\ T \wedge T \subseteq S)$ 
by  $(metis\ closedin\_imp\_subset\ closedin\_compact\ closed\_subset\ compact\_imp\_closed)$ 

```

2.3.1 Closure

```

lemma  $euclidean\_closure\_of\ [simp]:\ euclidean\_closure\_of\ S = closure\ S$ 
by  $(auto\ simp:\ closure\_of\_def\ closure\_def\ islimpt\_def)$ 

```

```

lemma  $closure\_openin\_Int\_closure:$ 

```

```

assumes  $ope:\ openin\ (top\_of\_set\ U)\ S$  and  $T \subseteq U$ 
shows  $closure(S \cap closure\ T) = closure(S \cap T)$ 

```

```

proof

```

```

obtain  $V$  where  $open\ V$  and  $S: S = U \cap V$ 

```

```

using  $ope$  using  $openin\_open$  by  $metis$ 

```

```

show  $closure(S \cap closure\ T) \subseteq closure(S \cap T)$ 

```

```

proof  $(clarsimp\ simp: S)$ 

```

```

fix  $x$ 

```

```

assume  $x \in closure(U \cap V \cap closure\ T)$ 

```

```

then have  $V \cap closure\ T \subseteq A \implies x \in closure\ A$  for  $A$ 

```

```

by  $(metis\ closure\_mono\ subsetD\ inf.coboundedI2\ inf\_assoc)$ 

```

```

then have  $x \in closure(T \cap V)$ 

```

```

by  $(metis\ \langle open\ V \rangle\ closure\_closure\ inf\_commute\ open\_Int\_closure\_subset)$ 

```

```

then show  $x \in closure(U \cap V \cap T)$ 

```

```

by  $(metis\ \langle T \subseteq U \rangle\ inf.absorb\_iff2\ inf\_assoc\ inf\_commute)$ 

```

```

qed

```

```

next

```

```

show  $closure(S \cap T) \subseteq closure(S \cap closure\ T)$ 

```

```

by  $(meson\ Int\_mono\ closure\_mono\ closure\_subset\ order\_refl)$ 

```

```

qed

```

```

corollary  $infinite\_openin:$ 

```

```

fixes  $S :: 'a :: t1\_space\ set$ 

```

```

shows  $\llbracket openin\ (top\_of\_set\ U)\ S; x \in S; x\ islimpt\ U \rrbracket \implies infinite\ S$ 

```

```

by  $(clarsimp\ simp\ add:\ openin\_open\ islimpt\_eq\_acc\_point\ inf\_commute)$ 

```

```

lemma  $closure\_Int\_ballI:$ 

```

```

assumes  $\bigwedge U. \llbracket openin\ (top\_of\_set\ S)\ U; U \neq \{\} \rrbracket \implies T \cap U \neq \{\}$ 

```

```

shows  $S \subseteq closure\ T$ 

```

```

proof  $(clarsimp\ simp:\ closure\_iff\_nhds\_not\_empty)$ 

```

```

fix  $x$  and  $A$  and  $V$ 

```

```

assume  $x \in S\ V \subseteq A\ open\ V\ x \in V\ T \cap A = \{\}$ 

```

```

then have  $openin\ (top\_of\_set\ S)\ (A \cap V \cap S)$ 

```

```

by  $(simp\ add:\ inf\_absorb2\ openin\_subtopology\_Int)$ 

```

```

moreover have  $A \cap V \cap S \neq \{\}$  using  $\langle x \in V \rangle\ \langle V \subseteq A \rangle\ \langle x \in S \rangle$ 

```

```

by  $auto$ 

```

```

ultimately show  $False$ 

```

```

using  $\langle T \cap A = \{\} \rangle$  assms by  $fastforce$ 

```

```

qed

```

2.3.2 Frontier

lemma *euclidean_interior_of* [*simp*]: *euclidean_interior_of* $S = \text{interior } S$
by (*auto simp: interior_of_def interior_def*)

lemma *euclidean_frontier_of* [*simp*]: *euclidean_frontier_of* $S = \text{frontier } S$
by (*auto simp: frontier_of_def frontier_def*)

lemma *connected_Int_frontier*:

$\llbracket \text{connected } S; S \cap T \neq \{\}; S - T \neq \{\} \rrbracket \implies S \cap \text{frontier } T \neq \{\}$
apply (*simp add: frontier_interiors connected_openin, safe*)
apply (*drule_tac x=S \cap interior T in spec, safe*)
apply (*drule_tac [2] x=S \cap interior (-T) in spec*)
apply (*auto simp: disjoint_eq_subset_Compl dest: interior_subset [THEN subsetD]*)
done

2.3.3 Compactness

lemma *openin_delete*:

fixes $a :: 'a :: t1_space$
shows *openin* (*top_of_set* u) $S \implies \text{openin } (\text{top_of_set } u) (S - \{a\})$
by (*metis Int_Diff open_delete openin_open*)

lemma *compact_eq_openin_cover*:

compact $S \iff$
 $(\forall C. (\forall c \in C. \text{openin } (\text{top_of_set } S) c) \wedge S \subseteq \bigcup C \longrightarrow$
 $(\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D))$

proof *safe*

fix C
assume *compact* S **and** $\forall c \in C. \text{openin } (\text{top_of_set } S) c$ **and** $S \subseteq \bigcup C$
then have $\forall c \in \{T. \text{open } T \wedge S \cap T \in C\}. \text{open } c$ **and** $S \subseteq \bigcup \{T. \text{open } T \wedge S \cap T \in C\}$
unfolding *openin_open* **by** *force+*
with $\langle \text{compact } S \rangle$ **obtain** D **where** $D \subseteq \{T. \text{open } T \wedge S \cap T \in C\}$ **and** *finite* D **and** $S \subseteq \bigcup D$
by (*meson compactE*)
then have *image* $(\lambda T. S \cap T) D \subseteq C \wedge \text{finite } (\text{image } (\lambda T. S \cap T) D) \wedge S \subseteq \bigcup (\text{image } (\lambda T. S \cap T) D)$
by *auto*
then show $\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D$ **..**

next

assume $1: \forall C. (\forall c \in C. \text{openin } (\text{top_of_set } S) c) \wedge S \subseteq \bigcup C \longrightarrow$
 $(\exists D \subseteq C. \text{finite } D \wedge S \subseteq \bigcup D)$

show *compact* S

proof (*rule compactI*)

fix C
let $?C = \text{image } (\lambda T. S \cap T) C$
assume $\forall t \in C. \text{open } t$ **and** $S \subseteq \bigcup C$
then have $(\forall c \in ?C. \text{openin } (\text{top_of_set } S) c) \wedge S \subseteq \bigcup ?C$

```

  unfolding openin_open by auto
with 1 obtain D where D ⊆ ?C and finite D and S ⊆ ⋃ D
  by metis
let ?D = inv_into C (λT. S ∩ T) ` D
have ?D ⊆ C ∧ finite ?D ∧ S ⊆ ⋃ ?D
proof (intro conjI)
  from ⟨D ⊆ ?C⟩ show ?D ⊆ C
  by (fast intro: inv_into_into)
  from ⟨finite D⟩ show finite ?D
  by (rule finite_imageI)
  from ⟨S ⊆ ⋃ D⟩ show S ⊆ ⋃ ?D
  by (metis ⟨D ⊆ (∩) S ` C⟩ image_inv_into_cancel inf_Sup le_infE)
qed
then show ∃ D ⊆ C. finite D ∧ S ⊆ ⋃ D ..
qed
qed

```

2.3.4 Continuity

```

lemma interior_image_subset:
  assumes inj f ∧ x. continuous (at x) f
  shows interior (f ` S) ⊆ f ` (interior S)
proof
  fix x assume x ∈ interior (f ` S)
  then obtain T where as: open T x ∈ T T ⊆ f ` S ..
  then have x ∈ f ` S by auto
  then obtain y where y: y ∈ S x = f y by auto
  have open (f - ` T)
    using assms ⟨open T⟩ by (simp add: continuous_at_imp_continuous_on
open_vimage)
  moreover have y ∈ vimage f T
    using ⟨x = f y⟩ ⟨x ∈ T⟩ by simp
  moreover have vimage f T ⊆ S
    using ⟨T ⊆ image f S⟩ ⟨inj f⟩ unfolding inj_on_def subset_eq by auto
  ultimately have y ∈ interior S ..
  with ⟨x = f y⟩ show x ∈ f ` interior S ..
qed

```

2.3.5 Equality of continuous functions on closure and related results

```

lemma continuous_closedin_preimage_constant:
  fixes f :: _ ⇒ 'b::t1_space
  shows continuous_on S f ⇒ closedin (top_of_set S) {x ∈ S. f x = a}
  using continuous_closedin_preimage[of S f {a}] by (simp add: vimage_def Collect_conj_eq)

```

```

lemma continuous_closed_preimage_constant:
  fixes f :: _ ⇒ 'b::t1_space

```

shows $\text{continuous_on } S f \implies \text{closed } S \implies \text{closed } \{x \in S. f x = a\}$
using $\text{continuous_closed_preimage}$ [of $S f \{a\}$] **by** ($\text{simp add: vimage_def Collect_conj_eq}$)

lemma $\text{continuous_constant_on_closure}$:

fixes $f :: _ \Rightarrow 'b::t1_space$
assumes $\text{continuous_on } (\text{closure } S) f$
and $\bigwedge x. x \in S \implies f x = a$
and $x \in \text{closure } S$
shows $f x = a$
using $\text{continuous_closed_preimage_constant}$ [of $\text{closure } S f a$]
 $\text{assms } \text{closure_minimal}$ [of $S \{x \in \text{closure } S. f x = a\}$] closure_subset
unfolding subset_eq
by *auto*

lemma $\text{image_closure_subset}$:

assumes $\text{contf: continuous_on } (\text{closure } S) f$
and $\text{closed } T$
and $f ' S \subseteq T$
shows $f ' (\text{closure } S) \subseteq T$

proof –

have $S \subseteq \{x \in \text{closure } S. f x \in T\}$
using $\text{assms}(?)$ closure_subset **by** *auto*
moreover **have** $\text{closed } (\text{closure } S \cap f - ' T)$
using $\text{continuous_closed_preimage}$ [OF contf] $\langle \text{closed } T \rangle$ **by** *auto*
ultimately **have** $\text{closure } S = (\text{closure } S \cap f - ' T)$
using closure_minimal [of $S (\text{closure } S \cap f - ' T)$] **by** *auto*
then **show** *?thesis* **by** *auto*

qed

lemma $\text{continuous_image_closure_subset}$:

assumes $\text{continuous_on } A f \text{ closure } B \subseteq A$
shows $f ' \text{closure } B \subseteq \text{closure } (f ' B)$
using assms **by** ($\text{meson } \text{closed_closure } \text{closure_subset } \text{continuous_on_subset } \text{image_closure_subset}$)

2.3.6 A function constant on a set

definition constant_on (**infixl** ($\text{constant}'_on$) 50)

where $f \text{ constant_on } A \equiv \exists y. \forall x \in A. f x = y$

lemma $\text{constant_on_subset}$: $\llbracket f \text{ constant_on } A; B \subseteq A \rrbracket \implies f \text{ constant_on } B$

unfolding constant_on_def **by** *blast*

lemma $\text{injective_not_constant}$:

fixes $S :: 'a::\{\text{perfect_space}\} \text{ set}$
shows $\llbracket \text{open } S; \text{inj_on } f S; f \text{ constant_on } S \rrbracket \implies S = \{\}$
unfolding constant_on_def
by ($\text{metis } \text{equals0I } \text{inj_on_contraD } \text{islimpt_UNIV } \text{islimpt_def}$)

lemma *constant_on_compose*:
assumes *f constant_on A*
shows $g \circ f$ *constant_on A*
using *assms* **by** (*auto simp: constant_on_def*)

lemma *not_constant_onI*:
 $f x \neq f y \implies x \in A \implies y \in A \implies \neg f$ *constant_on A*
unfolding *constant_on_def* **by** *metis*

lemma *constant_onE*:
assumes *f constant_on S* **and** $\bigwedge x. x \in S \implies f x = g x$
shows *g constant_on S*
using *assms* **unfolding** *constant_on_def* **by** *simp*

lemma *constant_on_closureI*:
fixes $f :: _ \Rightarrow 'b::t1_space$
assumes *cof: f constant_on S* **and** *contf: continuous_on (closure S) f*
shows *f constant_on (closure S)*
using *continuous_constant_on_closure [OF contf] cof* **unfolding** *constant_on_def*
by *metis*

2.3.7 Continuity relative to a union.

lemma *continuous_on_Un_local*:
 $\llbracket \text{closedin } (\text{top_of_set } (S \cup T)) S; \text{closedin } (\text{top_of_set } (S \cup T)) T; \text{continuous_on } S f; \text{continuous_on } T f \rrbracket$
 $\implies \text{continuous_on } (S \cup T) f$
unfolding *continuous_on_closedin_limpt*
by (*metis Lim_trivial_limit Lim_within_Un Un_iff trivial_limit_within*)

lemma *continuous_on_cases_local*:
 $\llbracket \text{closedin } (\text{top_of_set } (S \cup T)) S; \text{closedin } (\text{top_of_set } (S \cup T)) T; \text{continuous_on } S f; \text{continuous_on } T g; \bigwedge x. \llbracket x \in S \wedge \neg P x \vee x \in T \wedge P x \rrbracket \implies f x = g x \rrbracket$
 $\implies \text{continuous_on } (S \cup T) (\lambda x. \text{if } P x \text{ then } f x \text{ else } g x)$
by (*rule continuous_on_Un_local*) (*auto intro: continuous_on_eq*)

lemma *continuous_on_cases_le*:
fixes $h :: 'a :: \text{topological_space} \Rightarrow \text{real}$
assumes *continuous_on* $\{x \in S. h x \leq a\}$ *f*
and *continuous_on* $\{x \in S. a \leq h x\}$ *g*
and *h: continuous_on S h*
and $\bigwedge x. \llbracket x \in S; h x = a \rrbracket \implies f x = g x$
shows *continuous_on S* $(\lambda x. \text{if } h x \leq a \text{ then } f(x) \text{ else } g(x))$

proof –

have $S: S = (S \cap h -' \text{atMost } a) \cup (S \cap h -' \text{atLeast } a)$

by *force*

have $1: \text{closedin } (\text{top_of_set } S) (S \cap h -' \text{atMost } a)$

```

    by (rule continuous_closedin_preimage [OF h closed_atMost])
  have 2: closedin (top_of_set S) (S ∩ h -' atLeast a)
    by (rule continuous_closedin_preimage [OF h closed_atLeast])
  have [simp]: S ∩ h -' {..a} = {x ∈ S. h x ≤ a} S ∩ h -' {a..} = {x ∈ S. a ≤
h x}
    by auto
  have continuous_on (S ∩ h -' {..a} ∪ S ∩ h -' {a..}) (λx. if h x ≤ a then f x
else g x)
    by (intro continuous_on_cases_local) (use 1 2 S assms in auto)
  then show ?thesis
    using S by force
qed

```

```

lemma continuous_on_cases_1:
  fixes S :: real set
  assumes continuous_on {t ∈ S. t ≤ a} f
    and continuous_on {t ∈ S. a ≤ t} g
    and a ∈ S ⇒ f a = g a
  shows continuous_on S (λt. if t ≤ a then f(t) else g(t))
using assms
by (auto intro: continuous_on_cases_le [where h = id, simplified])

```

2.3.8 Inverse function property for open/closed maps

```

lemma continuous_on_inverse_open_map:
  assumes contf: continuous_on S f
    and imf: f ' S = T
    and injf: ∧x. x ∈ S ⇒ g (f x) = x
    and oo: ∧U. openin (top_of_set S) U ⇒ openin (top_of_set T) (f ' U)
  shows continuous_on T g
proof -
  from imf injf have gTS: g ' T = S
    by force
  from imf injf have fU: U ⊆ S ⇒ (f ' U) = T ∩ g -' U for U
    by force
  show ?thesis
    by (simp add: continuous_on_open [of T g] gTS) (metis openin_imp_subset
fU oo)
qed

```

```

lemma continuous_on_inverse_closed_map:
  assumes contf: continuous_on S f
    and imf: f ' S = T
    and injf: ∧x. x ∈ S ⇒ g(f x) = x
    and oo: ∧U. closedin (top_of_set S) U ⇒ closedin (top_of_set T) (f ' U)
  shows continuous_on T g
proof -
  from imf injf have gTS: g ' T = S
    by force

```


from $imf\ injf$ **have** $fU: U \subseteq S \implies (f \text{ ` } U) = T \cap g \text{ - ` } U$ **for** U
by *force*
show *?thesis*
by (*simp add: continuous_on_closed [of T g] gTS*) (*metis closedin_imp_subset fU oo*)
qed

lemma *homeomorphism_injective_open_map:*
assumes *contf: continuous_on S f*
and *imf: f ` S = T*
and *injf: inj_on f S*
and *oo: $\bigwedge U. openin (top_of_set S) U \implies openin (top_of_set T) (f \text{ ` } U)$*
obtains g **where** *homeomorphism S T f g*
proof
have *continuous_on T (inv_into S f)*
by (*metis contf continuous_on_inverse_open_map imf injf inv_into_f_f oo*)
with *imf injf contf* **show** *homeomorphism S T f (inv_into S f)*
by (*auto simp: homeomorphism_def*)
qed

lemma *homeomorphism_injective_closed_map:*
assumes *contf: continuous_on S f*
and *imf: f ` S = T*
and *injf: inj_on f S*
and *oo: $\bigwedge U. closedin (top_of_set S) U \implies closedin (top_of_set T) (f \text{ ` } U)$*
obtains g **where** *homeomorphism S T f g*
proof
have *continuous_on T (inv_into S f)*
by (*metis contf continuous_on_inverse_closed_map imf injf inv_into_f_f oo*)
with *imf injf contf* **show** *homeomorphism S T f (inv_into S f)*
by (*auto simp: homeomorphism_def*)
qed

lemma *homeomorphism_imp_open_map:*
assumes *hom: homeomorphism S T f g*
and *oo: openin (top_of_set S) U*
shows *openin (top_of_set T) (f ` U)*
proof –
from *hom oo* **have** [*simp*]: $f \text{ ` } U = T \cap g \text{ - ` } U$
using *openin_subset* **by** (*fastforce simp: homeomorphism_def rev_image_eqI*)
from *hom* **have** *continuous_on T g*
unfolding *homeomorphism_def* **by** *blast*
moreover **have** $g \text{ ` } T = S$
by (*metis hom homeomorphism_def*)
ultimately **show** *?thesis*
by (*simp add: continuous_on_open oo*)
qed

lemma *homeomorphism_imp_closed_map:*

```

assumes hom: homeomorphism S T f g
  and oo: closedin (top_of_set S) U
shows closedin (top_of_set T) (f ' U)
proof -
from hom oo have [simp]: f ' U = T ∩ g - ' U
  using closedin_subset by (fastforce simp: homeomorphism_def rev_image_eqI)
from hom have continuous_on T g
  unfolding homeomorphism_def by blast
moreover have g ' T = S
  by (metis hom homeomorphism_def)
ultimately show ?thesis
  by (simp add: continuous_on_closed oo)
qed

```

2.3.9 Seperability

lemma subset_second_countable:

```

obtains B :: 'a:: second_countable_topology set set
  where countable B
        {} ∉ B
        ∧ C. C ∈ B ⇒ openin(top_of_set S) C
        ∧ T. openin(top_of_set S) T ⇒ ∃U. U ⊆ B ∧ T = ∪U

```

```

proof -
obtain B :: 'a set set
  where countable B
    and opeB: ∧ C. C ∈ B ⇒ openin(top_of_set S) C
    and B: ∧ T. openin(top_of_set S) T ⇒ ∃U. U ⊆ B ∧ T = ∪U
proof -
obtain C :: 'a set set
  where countable C and ope: ∧ C. C ∈ C ⇒ open C
    and C: ∧ S. open S ⇒ ∃U. U ⊆ C ∧ S = ∪U
  by (metis univ_second_countable that)
show ?thesis
proof
show countable ((λC. S ∩ C) ' C)
  by (simp add: ‹countable C›)
show ∧ C. C ∈ (∩) S ' C ⇒ openin (top_of_set S) C
  using ope by auto
show ∧ T. openin (top_of_set S) T ⇒ ∃U ⊆ (∩) S ' C. T = ∪U
  by (metis C image_mono inf_Sup openin_open)
qed
qed
show ?thesis

```

```

proof
show countable (B - {{}})
  using ‹countable B› by blast
show ∧ C. [C ∈ B - {{}}] ⇒ openin (top_of_set S) C
  by (simp add: ‹∧ C. C ∈ B ⇒ openin (top_of_set S) C›)
show ∃U ⊆ B - {{}}. T = ∪U if openin (top_of_set S) T for T

```

```

    using  $\mathcal{B}$  [OF that]
    apply clarify
    by (rule_tac  $x=U - \{\{\}\}$  in exI, auto)
  qed auto
qed

```

lemma *Lindelof_openin*:

```

  fixes  $\mathcal{F} :: 'a::second\_countable\_topology\ set\ set$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies openin\ (top\_of\_set\ U)\ S$ 
  obtains  $\mathcal{F}'$  where  $\mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
proof -
  have  $\bigwedge S. S \in \mathcal{F} \implies \exists T. open\ T \wedge S = U \cap T$ 
    using assms by (simp add: openin_open)
  then obtain  $tf$  where  $tf: \bigwedge S. S \in \mathcal{F} \implies open\ (tf\ S) \wedge (S = U \cap tf\ S)$ 
    by metis
  have [simp]:  $\bigwedge \mathcal{F}'. \mathcal{F}' \subseteq \mathcal{F} \implies \bigcup \mathcal{F}' = U \cap \bigcup (tf\ ' \mathcal{F}')$ 
    using  $tf$  by fastforce
  obtain  $\mathcal{G}$  where countable  $\mathcal{G} \wedge \mathcal{G} \subseteq tf\ ' \mathcal{F} \cup \mathcal{G} = \bigcup (tf\ ' \mathcal{F})$ 
    using  $tf$  by (force intro: Lindelof [of  $tf\ ' \mathcal{F}$ ])
  then obtain  $\mathcal{F}'$  where  $\mathcal{F}': \mathcal{F}' \subseteq \mathcal{F}$  countable  $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F}$ 
    by (clarsimp simp add: countable_subset_image)
  then show ?thesis ..
qed

```

2.3.10 Closed Maps

lemma *continuous_imp_closed_map*:

```

  fixes  $f :: 'a::t2\_space \Rightarrow 'b::t2\_space$ 
  assumes closedin (top_of_set S) U
         continuous_on S f  $f\ ' S = T$  compact S
  shows closedin (top_of_set T) (f ' U)
  by (metis assms closedin_compact_eq compact_continuous_image continuous_on_subset
  subset_image_iff)

```

lemma *closed_map_restrict*:

```

  assumes cloU: closedin (top_of_set (S  $\cap$  f - ' T')) U
         and cc:  $\bigwedge U. closedin\ (top\_of\_set\ S)\ U \implies closedin\ (top\_of\_set\ T)\ (f\ ' U)$ 
         and  $T' \subseteq T$ 
  shows closedin (top_of_set T') (f ' U)
proof -
  obtain V where closed V  $U = S \cap f - ' T' \cap V$ 
    using cloU by (auto simp: closedin_closed)
  with cc [of S  $\cap$  V]  $\langle T' \subseteq T \rangle$  show ?thesis
    by (fastforce simp add: closedin_closed)
qed

```

2.3.11 Open Maps

lemma *open_map_restrict*:

```

  assumes opeU: openin (top_of_set (S  $\cap$  f - ' T')) U

```

and $oo: \bigwedge U. \text{openin } (\text{top_of_set } S) U \implies \text{openin } (\text{top_of_set } T) (f' U)$
and $T' \subseteq T$
shows $\text{openin } (\text{top_of_set } T') (f' U)$
proof –
obtain V **where** $\text{open } V U = S \cap f -' T' \cap V$
using $\text{ope}U$ **by** $(\text{auto simp: openin_open})$
with oo $[of\ S \cap V]$ $\langle T' \subseteq T \rangle$ **show** $?thesis$
by $(\text{fastforce simp add: openin_open})$
qed

2.3.12 Quotient maps

lemma *quotient_map_imp_continuous_open*:
assumes $T: f \in S \rightarrow T$
and $ope: \bigwedge U. U \subseteq T$
 $\implies (\text{openin } (\text{top_of_set } S) (S \cap f -' U) \longleftrightarrow \text{openin } (\text{top_of_set } T) U)$
shows $\text{continuous_on } S f$
proof –
have $[simp]: S \cap f -' f' S = S$ **by** *auto*
show $?thesis$
by $(\text{meson } T \text{ continuous_on_open_gen } \text{ope } \text{openin_imp_subset})$
qed

lemma *quotient_map_imp_continuous_closed*:
assumes $T: f \in S \rightarrow T$
and $ope: \bigwedge U. U \subseteq T$
 $\implies (\text{closedin } (\text{top_of_set } S) (S \cap f -' U) \longleftrightarrow \text{closedin } (\text{top_of_set } T) U)$
shows $\text{continuous_on } S f$
proof –
have $[simp]: S \cap f -' f' S = S$ **by** *auto*
show $?thesis$
by $(\text{meson } T \text{ closedin_imp_subset } \text{continuous_on_closed_gen } \text{ope})$
qed

lemma *open_map_imp_quotient_map*:
assumes $\text{contf: continuous_on } S f$
and $T: T \subseteq f' S$
and $ope: \bigwedge T. \text{openin } (\text{top_of_set } S) T$
 $\implies \text{openin } (\text{top_of_set } (f' S)) (f' T)$
shows $\text{openin } (\text{top_of_set } S) (S \cap f -' T) = \text{openin } (\text{top_of_set } (f' S)) T$
proof –
have $T = f' (S \cap f -' T)$
using T **by** *blast*
then show $?thesis$
using $\text{ope contf continuous_on_open}$ **by** *metis*
qed

lemma *closed_map_imp_quotient_map*:

assumes *contf*: *continuous_on S f*
and *T*: $T \subseteq f^{-1} S$
and *ope*: $\bigwedge T. \text{closedin } (\text{top_of_set } S) T$
 $\implies \text{closedin } (\text{top_of_set } (f^{-1} S)) (f^{-1} T)$
shows $\text{openin } (\text{top_of_set } S) (S \cap f^{-1} T) \longleftrightarrow$
 $\text{openin } (\text{top_of_set } (f^{-1} S)) T$
(is ?lhs = ?rhs)

proof

assume *?lhs*
then have ***: $\text{closedin } (\text{top_of_set } S) (S - (S \cap f^{-1} T))$
using *closedin_diff* **by** *fastforce*
have [*simp*]: $(f^{-1} S - f^{-1} (S - (S \cap f^{-1} T))) = T$
using *T* **by** *blast*
show *?rhs*
using *ope* [*OF* ***, *unfolded closedin_def*] **by** *auto*
next
assume *?rhs*
with *contf* **show** *?lhs*
by (*auto simp: continuous_on_open*)
qed

lemma *continuous_right_inverse_imp_quotient_map*:

assumes *contf*: *continuous_on S f* **and** *imf*: $f \in S \rightarrow T$
and *contg*: *continuous_on T g* **and** *img*: $g \in T \rightarrow S$
and *fg* [*simp*]: $\bigwedge y. y \in T \implies f(g y) = y$
and *U*: $U \subseteq T$
shows $\text{openin } (\text{top_of_set } S) (S \cap f^{-1} U) \longleftrightarrow$
 $\text{openin } (\text{top_of_set } T) U$
(is ?lhs = ?rhs)

proof –

have *f*: $\bigwedge Z. \text{openin } (\text{top_of_set } (f^{-1} S)) Z \implies$
 $\text{openin } (\text{top_of_set } S) (S \cap f^{-1} Z)$
and *g*: $\bigwedge Z. \text{openin } (\text{top_of_set } (g^{-1} T)) Z \implies$
 $\text{openin } (\text{top_of_set } T) (T \cap g^{-1} Z)$
using *contf contg* **by** (*auto simp: continuous_on_open*)
show *?thesis*

proof

have $T \cap g^{-1} (g^{-1} T \cap (S \cap f^{-1} U)) = \{x \in T. f(g x) \in U\}$
using *imf img* **by** *blast*
also have $\dots = U$
using *U* **by** *auto*
finally have *eq*: $T \cap g^{-1} (g^{-1} T \cap (S \cap f^{-1} U)) = U$.

assume *?lhs*

then have ***: $\text{openin } (\text{top_of_set } (g^{-1} T)) (g^{-1} T \cap (S \cap f^{-1} U))$

by (*meson img openin_Int openin_subtopology_Int_subset openin_subtopology_self image_subset_iff_funcset*)

show *?rhs*

```

    using g [OF *] eq by auto
  next
    assume rhs: ?rhs
    show ?lhs
      using assms continuous_openin_preimage rhs by blast
  qed
qed

```

```

lemma continuous_left_inverse_imp_quotient_map:
  assumes continuous_on S f
    and continuous_on (f ' S) g
    and  $\bigwedge x. x \in S \implies g(f x) = x$ 
    and  $U \subseteq f ' S$ 
  shows openin (top_of_set S) (S  $\cap$  f -' U)  $\longleftrightarrow$ 
    openin (top_of_set (f ' S)) U
  using assms
  by (intro continuous_right_inverse_imp_quotient_map) auto

```

```

lemma continuous_imp_quotient_map:
  fixes f :: 'a::t2_space  $\Rightarrow$  'b::t2_space
  assumes continuous_on S f f ' S = T compact S U  $\subseteq$  T
  shows openin (top_of_set S) (S  $\cap$  f -' U)  $\longleftrightarrow$ 
    openin (top_of_set T) U
  by (simp add: assms closed_map_imp_quotient_map continuous_imp_closed_map)

```

2.3.13 Pasting lemmas for functions, for of casewise definitions

on open sets

```

lemma pasting_lemma:
  assumes ope:  $\bigwedge i. i \in I \implies \text{openin } X (T i)$ 
    and cont:  $\bigwedge i. i \in I \implies \text{continuous\_map}(\text{subtopology } X (T i)) Y (f i)$ 
    and f:  $\bigwedge i j x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T i \cap T j \rrbracket \implies f i x = f j x$ 
    and g:  $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T j \wedge g x = f j x$ 
  shows continuous_map X Y g
  unfolding continuous_map_openin_preimage_eq
proof (intro conjI allI impI)
  show g  $\in$  topspace X  $\rightarrow$  topspace Y
    using g cont continuous_map_image_subset_topspace by fastforce
next
  fix U
  assume Y: openin Y U
  have T: T i  $\subseteq$  topspace X if i  $\in$  I for i
    using ope by (simp add: openin_subset that)
  have *: topspace X  $\cap$  g -' U = ( $\bigcup i \in I. T i \cap f i -' U$ )
    using f g T by fastforce
  have  $\bigwedge i. i \in I \implies \text{openin } X (T i \cap f i -' U)$ 
    using cont unfolding continuous_map_openin_preimage_eq
  by (metis Y T inf.commute inf_absorb1 ope topspace_subtopology openin_trans_full)

```

```

then show openin X (topspace X  $\cap$  g -' U)
  by (auto simp: *)
qed

```

lemma *pasting_lemma_exists*:

```

assumes X: topspace X  $\subseteq$  ( $\bigcup$  i  $\in$  I. T i)
  and ope:  $\bigwedge$  i. i  $\in$  I  $\implies$  openin X (T i)
  and cont:  $\bigwedge$  i. i  $\in$  I  $\implies$  continuous_map (subtopology X (T i)) Y (f i)
  and f:  $\bigwedge$  i j x. [i  $\in$  I; j  $\in$  I; x  $\in$  topspace X  $\cap$  T i  $\cap$  T j]  $\implies$  f i x = f j x
  obtains g where continuous_map X Y g  $\bigwedge$  x i. [i  $\in$  I; x  $\in$  topspace X  $\cap$  T i]
 $\implies$  g x = f i x

```

proof

```

let ?h =  $\lambda$  x. f (SOME i. i  $\in$  I  $\wedge$  x  $\in$  T i) x
show continuous_map X Y ?h
  apply (rule pasting_lemma [OF ope cont])
  apply (blast intro: f)+
  by (metis (no_types, lifting) UN_E X subsetD someI_ex)
show f (SOME i. i  $\in$  I  $\wedge$  x  $\in$  T i) x = f i x if i  $\in$  I x  $\in$  topspace X  $\cap$  T i for
i x
  by (metis (no_types, lifting) IntD2 IntI f someI_ex that)
qed

```

lemma *pasting_lemma_locally_finite*:

```

assumes fin:  $\bigwedge$  x  $\in$  topspace X  $\implies$   $\exists$  V. openin X V  $\wedge$  x  $\in$  V  $\wedge$  finite {i  $\in$ 
I. T i  $\cap$  V  $\neq$  {}}
  and clo:  $\bigwedge$  i. i  $\in$  I  $\implies$  closedin X (T i)
  and cont:  $\bigwedge$  i. i  $\in$  I  $\implies$  continuous_map(subtopology X (T i)) Y (f i)
  and f:  $\bigwedge$  i j x. [i  $\in$  I; j  $\in$  I; x  $\in$  topspace X  $\cap$  T i  $\cap$  T j]  $\implies$  f i x = f j x
  and g:  $\bigwedge$  x. x  $\in$  topspace X  $\implies$   $\exists$  j. j  $\in$  I  $\wedge$  x  $\in$  T j  $\wedge$  g x = f j x

```

```

shows continuous_map X Y g
unfolding continuous_map_closedin_preimage_eq

```

proof (*intro conjI allI impI*)

```

show g  $\in$  topspace X  $\rightarrow$  topspace Y
  using g cont continuous_map_image_subset_topspace by fastforce

```

next

fix U

assume Y: *closedin* Y U

have T: T *i* \subseteq *topspace* X **if** *i* \in I **for** *i*

using *clo* **by** (*simp add*: *closedin_subset* *that*)

have *: *topspace* X \cap *g* -' U = (\bigcup *i* \in I. T *i* \cap *f* *i* -' U)

using *f g T* **by** *fastforce*

have *cTf*: \bigwedge *i*. *i* \in I \implies *closedin* X (T *i* \cap *f* *i* -' U)

using *cont unfolding continuous_map_closedin_preimage_eq topspace_subtopology*
by (*simp add*: Int_absorb1 T Y *clo closedin_closed_subtopology*)

have *sub*: {Z \in (λ *i*. T *i* \cap *f* *i* -' U) -' I. Z \cap V \neq {}}
 \subseteq (λ *i*. T *i* \cap *f* *i* -' U) -' {*i* \in I. T *i* \cap V \neq {}} **for** V

by *auto*

have 1: (\bigcup *i* \in I. T *i* \cap *f* *i* -' U) \subseteq *topspace* X

using T **by** *blast*

then have *locally_finite_in* X $((\lambda i. T\ i \cap f\ i - ' U) ' I)$
unfolding *locally_finite_in_def*
using *finite_subset* [*OF sub*] *fin* **by force**
then show *closedin* X $(\text{topspace } X \cap g - ' U)$
by (*smt* (*verit*, *best*) * *cTf* *closedin_locally_finite_Union_image_iff*)
qed

Likewise on closed sets, with a finiteness assumption

lemma *pasting_lemma_closed*:
assumes *fin*: *finite* I
and *clo*: $\bigwedge i. i \in I \implies \text{closedin } X (T\ i)$
and *cont*: $\bigwedge i. i \in I \implies \text{continuous_map}(\text{subtopology } X (T\ i))\ Y (f\ i)$
and *f*: $\bigwedge i\ j\ x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T\ i \cap T\ j \rrbracket \implies f\ i\ x = f\ j\ x$
and *g*: $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
shows *continuous_map* $X\ Y\ g$
using *pasting_lemma_locally_finite* [*OF _ clo cont f g*] *fin* **by auto**

lemma *pasting_lemma_exists_locally_finite*:
assumes *fin*: $\bigwedge x. x \in \text{topspace } X \implies \exists V. \text{openin } X\ V \wedge x \in V \wedge \text{finite } \{i \in I. T\ i \cap V \neq \{\}\}$
and X : *topspace* $X \subseteq \bigcup (T\ ' I)$
and *clo*: $\bigwedge i. i \in I \implies \text{closedin } X (T\ i)$
and *cont*: $\bigwedge i. i \in I \implies \text{continuous_map}(\text{subtopology } X (T\ i))\ Y (f\ i)$
and *f*: $\bigwedge i\ j\ x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T\ i \cap T\ j \rrbracket \implies f\ i\ x = f\ j\ x$
and *g*: $\bigwedge x. x \in \text{topspace } X \implies \exists j. j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
obtains g **where** *continuous_map* $X\ Y\ g \wedge \bigwedge x\ i. \llbracket i \in I; x \in \text{topspace } X \cap T\ i \rrbracket \implies g\ x = f\ i\ x$

proof
show *continuous_map* $X\ Y$ $(\lambda x. f(@i. i \in I \wedge x \in T\ i)\ x)$
apply (*rule* *pasting_lemma_locally_finite* [*OF fin*])
apply (*blast* *intro*: *assms*)
by (*metis* (*no_types*, *lifting*) *UN_E* X *set_rev_mp* *someI_ex*)

next
fix $x\ i$
assume $i \in I$ **and** $x \in \text{topspace } X \cap T\ i$
then show f (*SOME* $i. i \in I \wedge x \in T\ i$) $x = f\ i\ x$
by (*metis* (*mono_tags*, *lifting*) *IntE* *IntI* f *someI2*)
qed

lemma *pasting_lemma_exists_closed*:
assumes *fin*: *finite* I
and X : *topspace* $X \subseteq \bigcup (T\ ' I)$
and *clo*: $\bigwedge i. i \in I \implies \text{closedin } X (T\ i)$
and *cont*: $\bigwedge i. i \in I \implies \text{continuous_map}(\text{subtopology } X (T\ i))\ Y (f\ i)$
and *f*: $\bigwedge i\ j\ x. \llbracket i \in I; j \in I; x \in \text{topspace } X \cap T\ i \cap T\ j \rrbracket \implies f\ i\ x = f\ j\ x$
obtains g **where** *continuous_map* $X\ Y\ g \wedge \bigwedge x\ i. \llbracket i \in I; x \in \text{topspace } X \cap T\ i \rrbracket \implies g\ x = f\ i\ x$
proof


```

show continuous_map X Y ( $\lambda x. f (SOME i. i \in I \wedge x \in T i) x$ )
  apply (rule pasting_lemma_closed [OF <finite I> clo cont])
  apply (blast intro: f)+
  by (metis (mono_tags, lifting) UN_iff X someI_ex subset_iff)
next
  fix x i
  assume  $i \in I \wedge x \in \text{topspace } X \cap T i$ 
  then show  $f (SOME i. i \in I \wedge x \in T i) x = f i x$ 
    by (metis (no_types, lifting) IntD2 IntI f someI_ex)
qed

lemma continuous_map_cases:
  assumes  $f: \text{continuous\_map } (\text{subtopology } X (X \text{ closure\_of } \{x. P x\})) Y f$ 
    and  $g: \text{continuous\_map } (\text{subtopology } X (X \text{ closure\_of } \{x. \neg P x\})) Y g$ 
    and  $fg: \bigwedge x. x \in X \text{ frontier\_of } \{x. P x\} \implies f x = g x$ 
  shows continuous_map X Y ( $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ )
proof (rule pasting_lemma_closed)
  let ?f =  $\lambda b. \text{if } b \text{ then } f \text{ else } g$ 
  let ?g =  $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ 
  let ?T =  $\lambda b. \text{if } b \text{ then } X \text{ closure\_of } \{x. P x\} \text{ else } X \text{ closure\_of } \{x. \neg P x\}$ 
  show finite {True, False} by auto
  have eq:  $\text{topspace } X - \text{Collect } P = \text{topspace } X \cap \{x. \neg P x\}$ 
    by blast
  show ?f i x = ?f j x
    if  $i \in \{True, False\} \wedge j \in \{True, False\}$  and  $x: x \in \text{topspace } X \cap ?T i \cap ?T j$  for
     $i j x$ 
  proof -
    have  $f x = g x$  if  $i \neg j$ 
    by (smt (verit, best) Diff_Diff_Int closure_of_interior_of_closure_of_restrict
    eq fg
      frontier_of_closures interior_of_complement that x)
    moreover
    have  $g x = f x$ 
    if  $x \in X \text{ closure\_of } \{x. \neg P x\} \wedge x \in X \text{ closure\_of } \text{Collect } P \neg i j$  for  $x$ 
    by (metis IntI closure_of_restrict eq fg frontier_of_closures that)
    ultimately show ?thesis
    using that by (auto simp flip: closure_of_restrict)
  qed
  show  $\exists j. j \in \{True, False\} \wedge x \in ?T j \wedge (\text{if } P x \text{ then } f x \text{ else } g x) = ?f j x$ 
    if  $x \in \text{topspace } X$  for  $x$ 
    by simp (metis in_closure_of_mem_Collect_eq that)
qed (auto simp: f g)

```

lemma continuous_map_cases_alt:

```

assumes  $f: \text{continuous\_map } (\text{subtopology } X (X \text{ closure\_of } \{x \in \text{topspace } X. P x\})) Y f$ 
  and  $g: \text{continuous\_map } (\text{subtopology } X (X \text{ closure\_of } \{x \in \text{topspace } X. \neg P x\})) Y g$ 
  and  $fg: \bigwedge x. x \in X \text{ frontier\_of } \{x \in \text{topspace } X. P x\} \implies f x = g x$ 

```

```

    shows continuous_map X Y ( $\lambda x$ . if P x then f x else g x)
  apply (rule continuous_map_cases)
  using assms
  apply (simp_all add: Collect_conj_eq closure_of_restrict [symmetric] frontier_of_restrict [symmetric])
  done

```

lemma *continuous_map_cases_function*:

```

  assumes contp: continuous_map X Z p
  and contf: continuous_map (subtopology X {x ∈ topspace X. p x ∈ Z closure_of U}) Y f
  and contg: continuous_map (subtopology X {x ∈ topspace X. p x ∈ Z closure_of (topspace Z - U)}) Y g
  and fg:  $\bigwedge x$ .  $\llbracket x \in \text{topspace } X; p \ x \in Z \text{ frontier\_of } U \rrbracket \implies f \ x = g \ x$ 
  shows continuous_map X Y ( $\lambda x$ . if p x ∈ U then f x else g x)
  proof (rule continuous_map_cases_alt)
    show continuous_map (subtopology X (X closure_of {x ∈ topspace X. p x ∈ U})) Y f
  proof (rule continuous_map_from_subtopology_mono)
    let ?T = {x ∈ topspace X. p x ∈ Z closure_of U}
    show continuous_map (subtopology X ?T) Y f
    by (simp add: contf)
    show X closure_of {x ∈ topspace X. p x ∈ U} ⊆ ?T
    by (rule continuous_map_closure_preimage_subset [OF contp])
  qed
  show continuous_map (subtopology X (X closure_of {x ∈ topspace X. p x ∉ U})) Y g
  proof (rule continuous_map_from_subtopology_mono)
    let ?T = {x ∈ topspace X. p x ∈ Z closure_of (topspace Z - U)}
    show continuous_map (subtopology X ?T) Y g
    by (simp add: contg)
    have X closure_of {x ∈ topspace X. p x ∉ U} ⊆ X closure_of {x ∈ topspace X. p x ∈ topspace Z - U}
    by (smt (verit) Collect_mono_iff DiffI closure_of_mono continuous_map contp image_subset_iff)
    then show X closure_of {x ∈ topspace X. p x ∉ U} ⊆ ?T
    by (rule order_trans [OF continuous_map_closure_preimage_subset [OF contp]])
  qed
  next
  show f x = g x if x ∈ X frontier_of {x ∈ topspace X. p x ∈ U} for x
  using that continuous_map_frontier_frontier_preimage_subset [OF contp, of U] fg by blast
  qed

```

2.3.14 Retractions

definition *retraction* :: ($'a::\text{topological_space}$) set \Rightarrow $'a$ set \Rightarrow ($'a \Rightarrow 'a$) \Rightarrow bool
 where *retraction* S T r \longleftrightarrow

$$T \subseteq S \wedge \text{continuous_on } S \ r \wedge r \in S \rightarrow T \wedge (\forall x \in T. r \ x = x)$$

definition *retract_of* (**infixl** *retract'_of* 50) **where**

$$T \text{ retract_of } S \iff (\exists r. \text{retraction } S \ T \ r)$$

lemma *retraction_idempotent*: $\text{retraction } S \ T \ r \implies x \in S \implies r \ (r \ x) = r \ x$
unfolding *retraction_def* **by** *auto*

Preservation of fixpoints under (more general notion of) retraction

lemma *invertible_fixpoint_property*:

fixes $S :: 'a::\text{topological_space set}$

and $T :: 'b::\text{topological_space set}$

assumes *contt*: $\text{continuous_on } T \ i$

and $i \in T \rightarrow S$

and *contr*: $\text{continuous_on } S \ r$

and $r \in S \rightarrow T$

and *ri*: $\bigwedge y. y \in T \implies r \ (i \ y) = y$

and *FP*: $\bigwedge f. [\text{continuous_on } S \ f; f \in S \rightarrow S] \implies \exists x \in S. f \ x = x$

and *contg*: $\text{continuous_on } T \ g$

and $g \in T \rightarrow T$

obtains y **where** $y \in T$ **and** $g \ y = y$

proof –

have $\exists x \in S. (i \circ g \circ r) \ x = x$

proof (*rule FP*)

show $\text{continuous_on } S \ (i \circ g \circ r)$

by (*metis* *assms(4)* *assms(8)* *contg* *continuous_on_compose* *continuous_on_subset* *contr* *contt* *funcset_image*)

show $(i \circ g \circ r) \in S \rightarrow S$

using *assms(2,4,8)* **by** *force*

qed

then obtain x **where** $x \in S \ (i \circ g \circ r) \ x = x \ ..$

then have $*$: $g \ (r \ x) \in T$

using *assms(4,8)* **by** *auto*

have $r \ ((i \circ g \circ r) \ x) = r \ x$

using x **by** *auto*

then show *?thesis*

using $*$ *ri* **that** **by** *auto*

qed

lemma *homeomorphic_fixpoint_property*:

fixes $S :: 'a::\text{topological_space set}$

and $T :: 'b::\text{topological_space set}$

assumes S *homeomorphic* T

shows $(\forall f. \text{continuous_on } S \ f \wedge f \in S \rightarrow S \longrightarrow (\exists x \in S. f \ x = x)) \iff$

$(\forall g. \text{continuous_on } T \ g \wedge g \in T \rightarrow T \longrightarrow (\exists y \in T. g \ y = y))$

(**is** *?lhs* = *?rhs*)

proof –

obtain $r \ i$ **where** r :

$\forall x \in S. i \ (r \ x) = x \ r \ 'S = T \ \text{continuous_on } S \ r$

```

     $\forall y \in T. r (i y) = y \text{ ' } T = S \text{ continuous\_on } T i$ 
    using assms unfolding homeomorphic_def homeomorphism_def by blast
  show ?thesis
  proof
    assume ?lhs
    with r show ?rhs
      by (smt (verit, del_insts) Pi_iff image_eqI invertible_fixpoint_property[of T
i S r])
    next
      assume ?rhs
      with r show ?lhs
        by (smt (verit, del_insts) Pi_iff image_eqI invertible_fixpoint_property[of S
r T i])
    qed
  qed

```

lemma *retract_fixpoint_property*:

```

  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
    and S :: 'a set
  assumes T retract_of S
    and FP:  $\bigwedge f. \llbracket \text{continuous\_on } S f; f \in S \rightarrow S \rrbracket \implies \exists x \in S. f x = x$ 
    and contg: continuous_on T g
    and g  $\in T \rightarrow T$ 
  obtains y where y  $\in T$  and g y = y
  proof -
    obtain h where retraction S T h
      using assms(1) unfolding retract_of_def ..
    then show ?thesis
      unfolding retraction_def
      using invertible_fixpoint_property[OF continuous_on_id FP]
      by (smt (verit, del_insts) Pi_iff assms(4) contg subsetD that)
  qed

```

lemma *retraction*:

```

  retraction S T r  $\longleftrightarrow T \subseteq S \wedge \text{continuous\_on } S r \wedge r \text{ ' } S = T \wedge (\forall x \in T. r x = x)$ 
  by (force simp: retract_def simp flip: image_subset_iff_funcset)

```

lemma *retractionE*: — yields properties normalized wrt. *simp* – less likely to loop

```

  assumes retraction S T r
  obtains  $T = r \text{ ' } S \text{ } r \in S \rightarrow S \text{ continuous\_on } S r \bigwedge x. x \in S \implies r (r x) = r x$ 
  proof (rule that)
    from retraction [of S T r] assms
    have  $T \subseteq S \text{ continuous\_on } S r \text{ } r \text{ ' } S = T$  and  $\forall x \in T. r x = x$ 
      by simp_all
    then show  $r \in S \rightarrow S \text{ continuous\_on } S r$ 
      by auto
    then show  $T = r \text{ ' } S$ 
      using  $\langle r \text{ ' } S = T \rangle$  by blast

```

from $\langle \forall x \in T. r x = x \rangle$ **have** $r x = x$ **if** $x \in T$ **for** x
using *that by simp*
with $\langle r ' S = T \rangle$ **show** $r (r x) = r x$ **if** $x \in S$ **for** x
using *that by auto*
qed

lemma *retract_ofE*: — yields properties normalized wrt. *simp* – less likely to loop
assumes T *retract_of* S
obtains r **where** $T = r ' S$ $r \in S \rightarrow S$ *continuous_on* S $r \wedge x. x \in S \implies r (r x) = r x$
proof –
from *assms* **obtain** r **where** *retraction* S T r
by (*auto simp add: retract_of_def*)
with *that* **show** *thesis*
by (*auto elim: retractionE*)
qed

lemma *retract_of_imp_extensible*:
assumes S *retract_of* T **and** *continuous_on* S f **and** $f \in S \rightarrow U$
obtains g **where** *continuous_on* T g $g \in T \rightarrow U \wedge x. x \in S \implies g x = f x$
proof –
from $\langle S$ *retract_of* $T \rangle$ **obtain** r **where** *retraction* T S r
by (*auto simp add: retract_of_def*)
then **have** *continuous_on* T $(f \circ r)$
by (*metis assms(2) continuous_on_compose retraction*)
then **show** *thesis*
by (*smt (verit, best) Pi_iff <retraction T S r> assms(3) comp_apply retraction_def that*)
qed

lemma *idempotent_imp_retraction*:
assumes *continuous_on* S f **and** $f \in S \rightarrow S$ **and** $\wedge x. x \in S \implies f(f x) = f x$
shows *retraction* S $(f ' S)$ f
by (*simp add: assms funcset_image retraction*)

lemma *retraction_subset*:
assumes *retraction* S T r **and** $T \subseteq S'$ **and** $S' \subseteq S$
shows *retraction* S' T r
unfolding *retraction_def*
by (*metis assms continuous_on_subset image_mono image_subset_iff_funcset retraction*)

lemma *retract_of_subset*:
assumes T *retract_of* S **and** $T \subseteq S'$ **and** $S' \subseteq S$
shows T *retract_of* S'
by (*meson assms retract_of_def retraction_subset*)

lemma *retraction_refl* [*simp*]: *retraction* S S $(\lambda x. x)$
by (*simp add: retraction*)

lemma *retract_of_refl* [*iff*]: S *retract_of* S
unfolding *retract_of_def* *retraction_def*
using *continuous_on_id* **by** *blast*

lemma *retract_of_imp_subset*:
 S *retract_of* $T \implies S \subseteq T$
by (*simp add: retract_of_def retraction_def*)

lemma *retract_of_empty* [*simp*]:
 $(\{\}$ *retract_of* $S) \iff S = \{\}$ (S *retract_of* $\{\}$) $\iff S = \{\}$
by (*auto simp: retract_of_def retraction_def*)

lemma *retract_of_singleton* [*iff*]: $(\{x\}$ *retract_of* $S) \iff x \in S$
unfolding *retract_of_def retraction_def* **by** *force*

lemma *retraction_comp*:
 \llbracket *retraction* S T f ; *retraction* T U g $\rrbracket \implies$ *retraction* S U $(g \circ f)$
by (*smt (verit, best) comp_apply continuous_on_compose image_comp retraction_subset_iff*)

lemma *retract_of_trans* [*trans*]:
assumes S *retract_of* T **and** T *retract_of* U
shows S *retract_of* U
using *assms* **by** (*auto simp: retract_of_def intro: retraction_comp*)

lemma *closedin_retract*:
fixes $S :: 'a :: t2_space$ *set*
assumes S *retract_of* T
shows *closedin* (*top_of_set* T) S
proof –
obtain r **where** $r: S \subseteq T$ *continuous_on* T r $r \in T \rightarrow S \wedge x. x \in S \implies r x = x$
using *assms* **by** (*auto simp: retract_of_def retraction_def*)
have $S = \{x \in T. x = r x\}$
using r **by** *force*
also have $\dots = T \cap ((\lambda x. (x, r x)) - \{ \{y. \exists x. y = (x, x)\} \})$
unfolding *vimage_def mem_Times_iff fst_conv snd_conv*
using r
by *auto*
also have *closedin* (*top_of_set* T) \dots
by (*rule continuous_closedin_preimage*) (*auto intro!: closed_diagonal continuous_on_Pair* r)
finally show *?thesis* .
qed

lemma *closedin_self* [*simp*]: *closedin* (*top_of_set* S) S
by *simp*

lemma *retract_of_closed*:

fixes $S :: 'a :: t2_space\ set$
shows $\llbracket closed\ T; S\ retract_of\ T \rrbracket \implies closed\ S$
by $(metis\ closedin_retract\ closedin_closed_eq)$

lemma *retract_of_compact*:

$\llbracket compact\ T; S\ retract_of\ T \rrbracket \implies compact\ S$
by $(metis\ compact_continuous_image\ retract_of_def\ retraction)$

lemma *retract_of_connected*:

$\llbracket connected\ T; S\ retract_of\ T \rrbracket \implies connected\ S$
by $(metis\ Topological_Spaces.connected_continuous_image\ retract_of_def\ retraction)$

lemma *retraction_openin_vimage_iff*:

assumes $r: retraction\ S\ T\ r$ **and** $U \subseteq T$
shows $openin\ (top_of_set\ S)\ (S \cap r^{-1}\ U) \longleftrightarrow openin\ (top_of_set\ T)\ U$ **(is**
 $?lhs = ?rhs)$

proof

show $?lhs \implies ?rhs$

using r

by $(smt\ (verit,\ del_insts)\ assms(2)\ continuous_right_inverse_imp_quotient_map\ image_subset_iff_funcset\ r\ retractionE\ retraction_def\ retraction_subset)$

show $?rhs \implies ?lhs$

by $(metis\ continuous_on_open\ r\ retraction)$

qed

lemma *retract_of_Times*:

$\llbracket S\ retract_of\ S'; T\ retract_of\ T' \rrbracket \implies (S \times T)\ retract_of\ (S' \times T')$
apply $(simp\ add: retract_of_def\ retraction_def\ Sigma_mono,\ clarify)$
apply $(rename_tac\ f\ g)$
apply $(rule_tac\ x=\lambda z. ((f \circ fst)\ z,\ (g \circ snd)\ z))\ in\ exI)$
apply $(rule\ conjI\ continuous_intros\ |\ erule\ continuous_on_subset\ | force)+$
done

2.3.15 Retractions on a topological space

definition *retract_of_space* $:: 'a\ set \Rightarrow 'a\ topology \Rightarrow bool$ **(infix** *retract'_of'_space* $50)$

where $S\ retract_of_space\ X$
 $\equiv S \subseteq topspace\ X \wedge (\exists r. continuous_map\ X\ (subtopology\ X\ S)\ r \wedge (\forall x \in S. r\ x = x))$

lemma *retract_of_space_retraction_maps*:

$S\ retract_of_space\ X \longleftrightarrow S \subseteq topspace\ X \wedge (\exists r. retraction_maps\ X\ (subtopology\ X\ S)\ r\ id)$

by $(auto\ simp: retract_of_space_def\ retraction_maps_def)$

lemma *retract_of_space_section_map*:

S retract_of_space $X \longleftrightarrow S \subseteq \text{topspace } X \wedge \text{section_map } (\text{subtopology } X S) X$
 id

unfolding retract_of_space_def retraction_maps_def section_map_def
by (auto simp: continuous_map_from_subtopology)

lemma retract_of_space_imp_subset:
 S retract_of_space $X \implies S \subseteq \text{topspace } X$
by (simp add: retract_of_space_def)

lemma retract_of_space_topspace:
 $\text{topspace } X$ retract_of_space X
using retract_of_space_def **by** force

lemma retract_of_space_empty [simp]:
 $\{\}$ retract_of_space $X \longleftrightarrow X = \text{trivial_topology}$
by (auto simp: retract_of_space_def)

lemma retract_of_space_singleton [simp]:
 $\{a\}$ retract_of_space $X \longleftrightarrow a \in \text{topspace } X$

proof –

have continuous_map X (subtopology $X \{a\}$) $(\lambda x. a) \wedge (\lambda x. a) a = a$ **if** $a \in \text{topspace } X$

using that **by** simp

then show ?thesis

by (force simp: retract_of_space_def)

qed

lemma retract_of_space_trans:
assumes S retract_of_space X T retract_of_space subtopology $X S$
shows T retract_of_space X
using assms
apply (simp add: retract_of_space_retraction_maps)
by (metis id_comp inf.absorb_iff2 retraction_maps_compose subtopology_subtopology)

lemma retract_of_space_subtopology:
assumes S retract_of_space X $S \subseteq U$
shows S retract_of_space subtopology $X U$
using assms
apply (clarsimp simp: retract_of_space_def)
by (metis continuous_map_from_subtopology inf.absorb2 subtopology_subtopology)

lemma retract_of_space_clopen:
assumes $\text{openin } X S$ $\text{closedin } X S$ $S = \{\}$ $\implies X = \text{trivial_topology}$
shows S retract_of_space X
proof (cases $S = \{\}$)
case False
then obtain a **where** $a \in S$
by blast
show ?thesis


```

unfolding retract_of_space_def
proof (intro exI conjI)
  show  $S \subseteq \text{topspace } X$ 
    by (simp add: assms closedin_subset)
  have continuous_map  $X X$  ( $\lambda x.$  if  $x \in S$  then  $x$  else  $a$ )
  proof (rule continuous_map_cases)
    show continuous_map (subtopology  $X$  ( $X$  closure_of  $\{x. x \in S\}$ ))  $X$  ( $\lambda x.$   $x$ )
      by (simp add: continuous_map_from_subtopology)
    show continuous_map (subtopology  $X$  ( $X$  closure_of  $\{x. x \notin S\}$ ))  $X$  ( $\lambda x.$   $a$ )
      using  $\langle S \subseteq \text{topspace } X \rangle \langle a \in S \rangle$  by force
    show  $x = a$  if  $x \in X$  frontier_of  $\{x. x \in S\}$  for  $x$ 
      using assms that clopenin_eq_frontier_of by fastforce
  qed
  then show continuous_map  $X$  (subtopology  $X S$ ) ( $\lambda x.$  if  $x \in S$  then  $x$  else  $a$ )
    using  $\langle S \subseteq \text{topspace } X \rangle \langle a \in S \rangle$  by (auto simp: continuous_map_in_subtopology)
  qed auto
qed (use assms in auto)

```

```

lemma retract_of_space_disjoint_union:
  assumes openin  $X S$  openin  $X T$  and  $ST: \text{disjnt } S T$   $S \cup T = \text{topspace } X$  and
   $S = \{\}$   $\implies X = \text{trivial\_topology}$ 
  shows  $S$  retract_of_space  $X$ 
proof (rule retract_of_space_clopen)
  have  $S \cap T = \{\}$ 
    by (meson  $ST$  disjnt_def)
  then have  $S = \text{topspace } X - T$ 
    using  $ST$  by auto
  then show closedin  $X S$ 
    using  $\langle \text{openin } X T \rangle$  by blast
qed (auto simp: assms)

```

```

lemma retraction_maps_section_image1:
  assumes retraction_maps  $X Y r s$ 
  shows  $s \text{ ' } (\text{topspace } Y)$  retract_of_space  $X$ 
  unfolding retract_of_space_section_map
proof
  show  $s \text{ ' } \text{topspace } Y \subseteq \text{topspace } X$ 
    using assms continuous_map_image_subset_topspace retraction_maps_def
by blast
  show section_map (subtopology  $X$  ( $s \text{ ' } \text{topspace } Y$ ))  $X$  id
    unfolding section_map_def
    using assms retraction_maps_to_retract_maps by blast
qed

```

```

lemma retraction_maps_section_image2:
  retraction_maps  $X Y r s$ 
   $\implies$  subtopology  $X$  ( $s \text{ ' } (\text{topspace } Y)$ ) homeomorphic_space  $Y$ 
  using embedding_map_imp_homeomorphic_space homeomorphic_space_sym
  section_imp_embedding_map

```

section_map_def by blast

lemma *hereditary_imp_retractive_property*:
assumes $\bigwedge X S. P X \implies P(\text{subtopology } X S)$
 $\bigwedge X X'. X \text{ homeomorphic_space } X' \implies (P X \longleftrightarrow Q X')$
assumes *retraction_map* $X X' r P X$
shows $Q X'$
by (*meson* *assms* *retraction_map_def* *retraction_maps_section_image2*)

2.3.16 Paths and path-connectedness

definition *pathin* :: $'a \text{ topology} \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{bool}$ **where**
 $\text{pathin } X g \equiv \text{continuous_map } (\text{subtopology } \text{euclideanreal } \{0..1\}) X g$

lemma *pathin_compose*:
 $\llbracket \text{pathin } X g; \text{continuous_map } X Y f \rrbracket \implies \text{pathin } Y (f \circ g)$
by (*simp* *add*: *continuous_map_compose* *pathin_def*)

lemma *pathin_subtopology*:
 $\text{pathin } (\text{subtopology } X S) g \longleftrightarrow \text{pathin } X g \wedge (\forall x \in \{0..1\}. g x \in S)$
by (*auto* *simp*: *pathin_def* *continuous_map_in_subtopology*)

lemma *pathin_const* [*simp*]: $\text{pathin } X (\lambda x. a) \longleftrightarrow a \in \text{topspace } X$
using *pathin_subtopology* **by** (*fastforce* *simp* *add*: *pathin_def*)

lemma *path_start_in_topspace*: $\text{pathin } X g \implies g 0 \in \text{topspace } X$
by (*force* *simp*: *pathin_def* *continuous_map*)

lemma *path_finish_in_topspace*: $\text{pathin } X g \implies g 1 \in \text{topspace } X$
by (*force* *simp*: *pathin_def* *continuous_map*)

lemma *path_image_subset_topspace*: $\text{pathin } X g \implies g \in (\{0..1\}) \rightarrow \text{topspace } X$
by (*force* *simp*: *pathin_def* *continuous_map*)

definition *path_connected_space* :: $'a \text{ topology} \Rightarrow \text{bool}$
where $\text{path_connected_space } X \equiv \forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists g. \text{pathin } X g \wedge g 0 = x \wedge g 1 = y$

definition *path_connectedin* :: $'a \text{ topology} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$
where $\text{path_connectedin } X S \equiv S \subseteq \text{topspace } X \wedge \text{path_connected_space}(\text{subtopology } X S)$

lemma *path_connectedin_absolute* [*simp*]:
 $\text{path_connectedin } (\text{subtopology } X S) S \longleftrightarrow \text{path_connectedin } X S$
by (*simp* *add*: *path_connectedin_def* *subtopology_subtopology*)

lemma *path_connectedin_subset_topspace*:
 $\text{path_connectedin } X S \implies S \subseteq \text{topspace } X$
by (*simp* *add*: *path_connectedin_def*)

lemma *path_connectedin_subtopology*:

$path_connectedin\ (subtopology\ X\ S)\ T \longleftrightarrow path_connectedin\ X\ T \wedge T \subseteq S$
by (*auto simp: path_connectedin_def subtopology_subtopology inf.absorb2*)

lemma *path_connectedin*:

$path_connectedin\ X\ S \longleftrightarrow$
 $S \subseteq topspace\ X \wedge$
 $(\forall x \in S. \forall y \in S. \exists g. pathin\ X\ g \wedge g \in \{0..1\} \rightarrow S \wedge g\ 0 = x \wedge g\ 1 = y)$

unfolding *path_connectedin_def path_connected_space_def pathin_def continuous_map_in_subtopology*

by (*intro conj_cong refl ball_cong*) (*simp_all add: inf.absorb_iff2 flip: image_subset_iff_funcset*)

lemma *path_connectedin_topspace*:

$path_connectedin\ X\ (topspace\ X) \longleftrightarrow path_connected_space\ X$
by (*simp add: path_connectedin_def*)

lemma *path_connected_imp_connected_space*:

assumes *path_connected_space X*

shows *connected_space X*

proof –

have *: $\exists S. connectedin\ X\ S \wedge g\ 0 \in S \wedge g\ 1 \in S$ **if** *pathin X g* **for** *g*

proof (*intro exI conjI*)

have *continuous_map* (*subtopology euclideanreal {0..1}*) *X g*

using *connectedin_absolute* **that** **by** (*simp add: pathin_def*)

then show *connectedin X* (*g ‘ {0..1}*)

by (*rule connectedin_continuous_map_image*) *auto*

qed *auto*

show *?thesis*

using *assms*

by (*auto intro: * simp add: path_connected_space_def connected_space_subconnected Ball_def*)

qed

lemma *path_connectedin_imp_connectedin*:

$path_connectedin\ X\ S \implies connectedin\ X\ S$

by (*simp add: connectedin_def path_connected_imp_connected_space path_connectedin_def*)

lemma *path_connected_space_topspace_empty*:

$path_connected_space\ trivial_topology$

by (*simp add: path_connected_space_def*)

lemma *path_connectedin_empty* [*simp*]: $path_connectedin\ X\ \{\}$

by (*simp add: path_connectedin*)

lemma *path_connectedin_singleton* [*simp*]: $path_connectedin\ X\ \{a\} \longleftrightarrow a \in topspace\ X$

proof

```

show path_connectedin X {a}  $\implies$  a  $\in$  topspace X
  by (simp add: path_connectedin)
show a  $\in$  topspace X  $\implies$  path_connectedin X {a}
  unfolding path_connectedin
  using pathin_const by fastforce
qed

```

```

lemma path_connectedin_continuous_map_image:
  assumes f: continuous_map X Y f and S: path_connectedin X S
  shows path_connectedin Y (f ' S)
proof -
  have fX: f  $\in$  (topspace X)  $\rightarrow$  topspace Y
    using continuous_map_def f by fastforce
  show ?thesis
    unfolding path_connectedin
  proof (intro conjI ballI; clarify?)
    fix x
    assume x  $\in$  S
    show f x  $\in$  topspace Y
      using S  $\langle$  x  $\in$  S  $\rangle$  fX path_connectedin_subset_topspace by fastforce
    next
    fix x y
    assume x  $\in$  S and y  $\in$  S
    then obtain g where g: pathin X g g  $\in$  {0..1}  $\rightarrow$  S g 0 = x g 1 = y
      using S by (force simp: path_connectedin)
    show  $\exists$  g. pathin Y g  $\wedge$  g  $\in$  {0..1}  $\rightarrow$  f ' S  $\wedge$  g 0 = f x  $\wedge$  g 1 = f y
      proof (intro exI conjI)
        show pathin Y (f  $\circ$  g)
          using  $\langle$  pathin X g  $\rangle$  f pathin_compose by auto
        qed (use g in auto)
      qed
    qed
  qed

```

```

lemma path_connectedin_discrete_topology:
  path_connectedin (discrete_topology U) S  $\longleftrightarrow$  S  $\subseteq$  U  $\wedge$  ( $\exists$  a. S  $\subseteq$  {a}) (is ?lhs
= ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (meson connectedin_discrete_topology path_connectedin_imp_connectedin)
  show ?rhs  $\implies$  ?lhs
    using subset_singletonD by fastforce
qed

```

```

lemma path_connected_space_discrete_topology:
  path_connected_space (discrete_topology U)  $\longleftrightarrow$  ( $\exists$  a. U  $\subseteq$  {a})
  by (metis path_connectedin_discrete_topology path_connectedin_topspace path_connected_space_topspace
subset_singletonD topspace_discrete_topology)

```

lemma *homeomorphic_path_connected_space_imp*:

$\llbracket \text{path_connected_space } X; X \text{ homeomorphic_space } Y \rrbracket \implies \text{path_connected_space } Y$

unfolding *homeomorphic_space_def homeomorphic_maps_def*

by (*smt (verit, ccfv_threshold) homeomorphic_imp_surjective_map homeomorphic_maps_def homeomorphic_maps_map path_connectedin_continuous_map_image path_connectedin_topspace*)

lemma *homeomorphic_path_connected_space*:

$X \text{ homeomorphic_space } Y \implies \text{path_connected_space } X \longleftrightarrow \text{path_connected_space } Y$

by (*meson homeomorphic_path_connected_space_imp homeomorphic_space_sym*)

lemma *homeomorphic_map_path_connectedness*:

assumes *homeomorphic_map* $X Y f U \subseteq \text{topspace } X$

shows $\text{path_connectedin } Y (f \text{ ' } U) \longleftrightarrow \text{path_connectedin } X U$

unfolding *path_connectedin_def*

proof (*intro conj_cong homeomorphic_path_connected_space*)

show $f \text{ ' } U \subseteq \text{topspace } Y \longleftrightarrow (U \subseteq \text{topspace } X)$

using *assms homeomorphic_imp_surjective_map* **by** *blast*

next

assume $U \subseteq \text{topspace } X$

show $\text{subtopology } Y (f \text{ ' } U) \text{ homeomorphic_space } \text{subtopology } X U$

using *assms unfolding homeomorphic_eq_everything_map*

by (*metis (no_types, opaque_lifting) assms homeomorphic_map_subtopologies homeomorphic_space homeomorphic_space_sym image_mono inf.absorb_iff2*)

qed

lemma *homeomorphic_map_path_connectedness_eq*:

$\text{homeomorphic_map } X Y f \implies \text{path_connectedin } X U \longleftrightarrow U \subseteq \text{topspace } X \wedge \text{path_connectedin } Y (f \text{ ' } U)$

by (*meson homeomorphic_map_path_connectedness path_connectedin_def*)

2.3.17 Connected components

definition *connected_component_of* :: $'a \text{ topology} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$

where $\text{connected_component_of } X x y \equiv$

$\exists T. \text{connectedin } X T \wedge x \in T \wedge y \in T$

abbreviation *connected_component_of_set*

where $\text{connected_component_of_set } X x \equiv \text{Collect } (\text{connected_component_of } X x)$

definition *connected_components_of* :: $'a \text{ topology} \Rightarrow ('a \text{ set}) \text{ set}$

where $\text{connected_components_of } X \equiv \text{connected_component_of_set } X \text{ ' } \text{topspace } X$

lemma *connected_component_in_topspace*:

$\text{connected_component_of } X x y \implies x \in \text{topspace } X \wedge y \in \text{topspace } X$

by (*meson connected_component_of_def connectedin_subset_topspace in_mono*)

lemma *connected_component_of_refl*:

connected_component_of X x x \longleftrightarrow *x* \in *topspace X*

by (*meson connected_component_in_topspace connected_component_of_def connectedin_sing insertII*)

lemma *connected_component_of_sym*:

connected_component_of X x y \longleftrightarrow *connected_component_of X y x*

by (*meson connected_component_of_def*)

lemma *connected_component_of_trans*:

\llbracket *connected_component_of X x y*; *connected_component_of X y z* \rrbracket

\implies *connected_component_of X x z*

unfolding *connected_component_of_def*

using *connectedin_Un* **by** *blast*

lemma *connected_component_of_mono*:

\llbracket *connected_component_of (subtopology X S) x y*; *S* \subseteq *T* \rrbracket

\implies *connected_component_of (subtopology X T) x y*

by (*metis connected_component_of_def connectedin_subtopology inf.absorb_iff2 subtopology_subtopology*)

lemma *connected_component_of_set*:

connected_component_of_set X x = $\{y. \exists T. \text{connectedin } X T \wedge x \in T \wedge y \in T\}$

by (*meson connected_component_of_def*)

lemma *connected_component_of_subset_topspace*:

connected_component_of_set X x \subseteq *topspace X*

using *connected_component_in_topspace* **by** *force*

lemma *connected_component_of_eq_empty*:

connected_component_of_set X x = $\{\}$ \longleftrightarrow (*x* \notin *topspace X*)

using *connected_component_in_topspace connected_component_of_refl* **by** *fastforce*

lemma *connected_space_iff_connected_component*:

connected_space X \longleftrightarrow ($\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \text{connected_component_of } X x y$)

by (*simp add: connected_component_of_def connected_space_subconnected*)

lemma *connected_space_imp_connected_component_of*:

\llbracket *connected_space X*; *a* \in *topspace X*; *b* \in *topspace X* \rrbracket

\implies *connected_component_of X a b*

by (*simp add: connected_space_iff_connected_component*)

lemma *connected_space_connected_component_set*:

connected_space X \longleftrightarrow ($\forall x \in \text{topspace } X. \text{connected_component_of_set } X x$)

= *topspace X*)

using *connected_component_of_subset_topspace connected_space_iff_connected_component*
by *fastforce*

lemma *connected_component_of_maximal*:

$\llbracket \text{connectedin } X \ S; \ x \in \ S \rrbracket \implies S \subseteq \text{connected_component_of_set } X \ x$
by (*meson Ball_Collect connected_component_of_def*)

lemma *connected_component_of_equiv*:

connected_component_of $X \ x \ y \longleftrightarrow$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{connected_component_of } X \ x = \text{connected_component_of } X \ y$
apply (*simp add: connected_component_in_topspace fun_eq_iff*)
by (*meson connected_component_of_refl connected_component_of_sym connected_component_of_trans*)

lemma *connected_component_of_disjoint*:

disjnt (*connected_component_of_set* $X \ x$) (*connected_component_of_set* $X \ y$)
 $\longleftrightarrow \sim(\text{connected_component_of } X \ x \ y)$
using *connected_component_of_equiv unfolding disjnt_iff* **by** *force*

lemma *connected_component_of_eq*:

connected_component_of $X \ x = \text{connected_component_of } X \ y \longleftrightarrow$
 $(x \notin \text{topspace } X) \wedge (y \notin \text{topspace } X) \vee$
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
connected_component_of $X \ x \ y$

by (*metis Collect_empty_eq_bot connected_component_of_eq_empty connected_component_of_equiv*)

lemma *connectedin_connected_component_of*:

connectedin $X \ (\text{connected_component_of_set } X \ x)$

proof –

have *connected_component_of_set* $X \ x = \bigcup \{T. \text{connectedin } X \ T \wedge x \in T\}$

by (*auto simp: connected_component_of_def*)

then show *?thesis*

by (*metis (no_types, lifting) InterI connectedin_Union emptyE mem_Collect_eq*)

qed

lemma *Union_connected_components_of*:

$\bigcup(\text{connected_components_of } X) = \text{topspace } X$

unfolding *connected_components_of_def*

using *connected_component_in_topspace connected_component_of_refl* **by** *fastforce*

lemma *connected_components_of_maximal*:

$\llbracket C \in \text{connected_components_of } X; \ \text{connectedin } X \ S; \ \sim \text{disjnt } C \ S \rrbracket \implies S \subseteq C$

unfolding *connected_components_of_def disjnt_def*

apply *clarify*

by (*metis Int_emptyI connected_component_of_def connected_component_of_trans*)

mem_Collect_eq)

lemma *pairwise_disjoint_connected_components_of*:

pairwise_disjnt (connected_components_of X)

unfolding *connected_components_of_def pairwise_def*

by (*smt (verit, best) connected_component_of_disjoint connected_component_of_eq imageE*)

lemma *complement_connected_components_of_Union*:

$C \in \text{connected_components_of } X$

$\implies \text{topspace } X - C = \bigcup (\text{connected_components_of } X - \{C\})$

by (*metis Union_connected_components_of bot.extremum ccpo_Sup_singleton diff_Union_pairwise_disjoint*

insert_subset pairwise_disjoint_connected_components_of)

lemma *nonempty_connected_components_of*:

$C \in \text{connected_components_of } X \implies C \neq \{\}$

unfolding *connected_components_of_def*

by (*metis (no_types, lifting) connected_component_of_eq_empty imageE*)

lemma *connected_components_of_subset*:

$C \in \text{connected_components_of } X \implies C \subseteq \text{topspace } X$

using *Union_connected_components_of* **by** *fastforce*

lemma *connectedin_connected_components_of*:

assumes $C \in \text{connected_components_of } X$

shows *connectedin X C*

proof –

have $C \in \text{connected_component_of_set } X \text{ ‘ } \text{topspace } X$

using *assms connected_components_of_def* **by** *blast*

then show *?thesis*

using *connectedin_connected_component_of* **by** *fastforce*

qed

lemma *connected_component_in_connected_components_of*:

connected_component_of_set X a \in connected_components_of X \longleftrightarrow a \in \text{topspace } X

by (*metis (no_types, lifting) connected_component_of_eq_empty connected_components_of_def image_iff*)

lemma *connected_space_iff_components_eq*:

connected_space X \longleftrightarrow (\forall C \in \text{connected_components_of } X. \forall C' \in \text{connected_components_of } X. C = C')

(**is** *?lhs = ?rhs*)

proof

show *?lhs \implies ?rhs*

by (*simp add: connected_components_of_def connected_space_connected_component_set*)

show *?rhs \implies ?lhs*

by (*metis Union_connected_components_of_Union_iff connected_space_subconnected*

connectedin_connected_components_of)
qed

lemma *connected_components_of_eq_empty*:
 $connected_components_of\ X = \{\}$ \longleftrightarrow $X = trivial_topology$
by (*simp add: connected_components_of_def*)

lemma *connected_components_of_empty_space*:
 $connected_components_of\ trivial_topology = \{\}$
by (*simp add: connected_components_of_eq_empty*)

lemma *connected_components_of_subset_sing*:
 $connected_components_of\ X \subseteq \{S\} \longleftrightarrow connected_space\ X \wedge (X = trivial_topology \vee topspace\ X = S)$

proof (*cases X = trivial_topology*)

case *True*

then show *?thesis*

by (*simp add: connected_components_of_empty_space*)

next

case *False*

then have $\llbracket connected_components_of\ X \subseteq \{S\} \rrbracket \implies topspace\ X = S$

using *Union_connected_components_of_connected_components_of_eq_empty*

by *fastforce*

with *False* **show** *?thesis*

unfolding *connected_components_of_def*

by (*metis connected_space_connected_component_set_empty_iff image_subset_iff insert_iff*)

qed

lemma *connected_space_iff_components_subset_singleton*:
 $connected_space\ X \longleftrightarrow (\exists a. connected_components_of\ X \subseteq \{a\})$
by (*simp add: connected_components_of_subset_sing*)

lemma *connected_components_of_eq_singleton*:
 $connected_components_of\ X = \{S\} \longleftrightarrow connected_space\ X \wedge X \neq trivial_topology \wedge S = topspace\ X$

by (*metis connected_components_of_eq_empty connected_components_of_subset_sing insert_not_empty_subset_singleton_iff*)

lemma *connected_components_of_connected_space*:
 $connected_space\ X \implies connected_components_of\ X = (if\ X = trivial_topology\ then\ \{\}\ else\ \{topspace\ X\})$
by (*simp add: connected_components_of_eq_empty connected_components_of_eq_singleton*)

lemma *exists_connected_component_of_superset*:
assumes *connectedin X S* **and** *ne: X \neq trivial_topology*
shows $\exists C. C \in connected_components_of\ X \wedge S \subseteq C$

proof (*cases S = \{\}*)

case *True*

```

then show ?thesis
  using ne_connected_components_of_eq_empty by fastforce
next
  case False
  then show ?thesis
  by (meson all_not_in_conv assms(1) connected_component_in_connected_components_of
connected_component_of_maximal_connectedin_subset_topspace in_mono)
qed

```

```

lemma closedin_connected_components_of:
  assumes  $C \in \text{connected\_components\_of } X$ 
  shows closedin  $X$   $C$ 

```

proof –

```

obtain  $x$  where  $x \in \text{topspace } X$  and  $x: C = \text{connected\_component\_of\_set } X$   $x$ 
  using assms by (auto simp: connected_components_of_def)
have connected_component_of_set  $X$   $x \subseteq \text{topspace } X$ 
  by (simp add: connected_component_of_subset_topspace)
moreover have  $X$  closure_of_connected_component_of_set  $X$   $x \subseteq \text{connected\_component\_of\_set}$ 
 $X$   $x$ 

```

proof (rule connected_component_of_maximal)

```

  show connectedin  $X$  ( $X$  closure_of_connected_component_of_set  $X$   $x$ )
  by (simp add: connectedin_closure_of_connectedin_connected_component_of)
  show  $x \in X$  closure_of_connected_component_of_set  $X$   $x$ 
  by (simp add:  $\langle x \in \text{topspace } X \rangle$  closure_of_connected_component_of_refl)

```

qed

ultimately

```

show ?thesis
  using closure_of_subset_eq  $x$  by auto

```

qed

```

lemma closedin_connected_component_of:

```

closedin X (connected_component_of_set X x)

```

by (metis closedin_connected_components_of_closedin_empty connected_component_in_connected_co
connected_component_of_eq_empty)

```

```

lemma connected_component_of_eq_overlap:

```

connected_component_of_set X $x = \text{connected_component_of_set } X$ $y \iff$

$(x \notin \text{topspace } X) \wedge (y \notin \text{topspace } X) \vee$

$\sim(\text{connected_component_of_set } X$ $x \cap \text{connected_component_of_set } X$ $y =$

$\{\})$

```

  using connected_component_of_equiv by fastforce

```

```

lemma connected_component_of_nonoverlap:

```

connected_component_of_set X $x \cap \text{connected_component_of_set } X$ $y = \{\}$

\iff

$(x \notin \text{topspace } X) \vee (y \notin \text{topspace } X) \vee$

$\sim(\text{connected_component_of_set } X$ $x = \text{connected_component_of_set } X$ $y)$

```

by (metis connected_component_of_eq_empty connected_component_of_eq_overlap
inf.idem)

```

lemma *connected_component_of_overlap*:
 $\sim(\text{connected_component_of_set } X \ x \cap \text{connected_component_of_set } X \ y = \{\})$
 \longleftrightarrow
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
 $\text{connected_component_of_set } X \ x = \text{connected_component_of_set } X \ y$
by (*meson connected_component_of_nonoverlap*)

2.3.18 Combining theorems for continuous functions into the reals

The homeomorphism between the real line and the open interval $(-1, 1)$

lemma *continuous_map_real_shrink*:
 $\text{continuous_map euclideanreal (top_of_set } \{-1 < .. < 1\}) (\lambda x. x / (1 + |x|))$
proof –
have *continuous_on UNIV* ($\lambda x::\text{real}. x / (1 + |x|)$)
by (*intro continuous_intros*) *auto*
then show *?thesis*
by (*auto simp: continuous_map_in_subtopology divide_simps*)
qed

lemma *continuous_on_real_grow*:
 $\text{continuous_on } \{-1 < .. < 1\} (\lambda x::\text{real}. x / (1 - |x|))$
by (*intro continuous_intros*) *auto*

lemma *real_grow_shrink*:
fixes $x::\text{real}$
shows $x / (1 + |x|) / (1 - |x / (1 + |x|)|) = x$
by (*force simp: divide_simps*)

lemma *homeomorphic_maps_real_shrink*:
 $\text{homeomorphic_maps euclideanreal (subtopology euclideanreal } \{-1 < .. < 1\})$
 $(\lambda x. x / (1 + |x|)) (\lambda y. y / (1 - |y|))$
by (*force simp: homeomorphic_maps_def continuous_map_real_shrink continuous_on_real_grow divide_simps*)

lemma *real_shrink_Galois*:
fixes $x::\text{real}$
shows $(x / (1 + |x|) = y) \longleftrightarrow (|y| < 1 \wedge y / (1 - |y|) = x)$
using *real_grow_shrink* **by** (*fastforce simp add: distrib_left*)

lemma *real_shrink_eq*:
fixes $x \ y::\text{real}$
shows $(x / (1 + |x|) = y / (1 + |y|)) \longleftrightarrow x = y$
by (*metis real_shrink_Galois*)

lemma *real_shrink_lt*:
fixes $x::\text{real}$
shows $x / (1 + |x|) < y / (1 + |y|) \longleftrightarrow x < y$

using *zero_less_mult_iff* [of $x\ y$] **by** (*auto simp: field_simps abs_if not_less*)

lemma *real_shrink_le*:

fixes $x::\text{real}$

shows $x / (1 + |x|) \leq y / (1 + |y|) \longleftrightarrow x \leq y$

by (*meson linorder_not_le real_shrink_lt*)

lemma *real_shrink_grow*:

fixes $x::\text{real}$

shows $|x| < 1 \implies x / (1 - |x|) / (1 + |x / (1 - |x|)|) = x$

using *real_shrink_Galois* **by** *blast*

lemma *continuous_shrink*:

continuous_on UNIV $(\lambda x::\text{real}. x / (1 + |x|))$

by (*intro continuous_intros*) *auto*

lemma *strict_mono_shrink*:

strict_mono $(\lambda x::\text{real}. x / (1 + |x|))$

by (*simp add: monotoneI real_shrink_lt*)

lemma *shrink_range*: $(\lambda x::\text{real}. x / (1 + |x|)) \text{ ' } S \subseteq \{-1 <..< 1\}$

by (*auto simp: divide_simps*)

Note: connected sets of real numbers are the same thing as intervals

lemma *connected_shrink*:

fixes $S :: \text{real set}$

shows *connected* $((\lambda x. x / (1 + |x|)) \text{ ' } S) \longleftrightarrow \text{connected } S$ (**is** *?lhs = ?rhs*)

proof

assume *?lhs*

then have *connected* $((\lambda x. x / (1 - |x|)) \text{ ' } (\lambda x. x / (1 + |x|)) \text{ ' } S)$

by (*metis continuous_on_real_grow shrink_range connected_continuous_image*

continuous_on_subset)

then show *?rhs*

using *real_grow_shrink* **by** (*force simp add: image_comp*)

next

assume *?rhs*

then show *?lhs*

using *connected_continuous_image*

by (*metis continuous_on_subset continuous_shrink subset_UNIV*)

qed

2.3.19 A few cardinality results

lemma *eqpoll_real_subset*:

fixes $a\ b::\text{real}$

assumes $a < b$ **and** $S: \bigwedge x. \llbracket a < x; x < b \rrbracket \implies x \in S$

shows $S \approx (\text{UNIV}::\text{real set})$

proof (*rule lepoll_antisym*)

```

show  $S \lesssim (UNIV::real\ set)$ 
  by (simp add: subset_imp_lepoll)
define f where  $f \equiv \lambda x. (a+b) / 2 + (b-a) / 2 * (x / (1 + |x|))$ 
show (UNIV::real set)  $\lesssim S$ 
  unfolding lepoll_def
proof (intro exI conjI)
  show inj f
    unfolding inj_on_def f_def
  by (smt (verit, ccfv_SIG) real_shrink_eq <a<b> divide_eq_0_iff mult_cancel_left
times_divide_eq_right)
  have pos:  $(b-a) / 2 > 0$ 
    using <a<b> by auto
  have *:  $a < (a + b) / 2 + (b - a) / 2 * x \longleftrightarrow (b - a) > (b - a) * -x$ 
     $(a + b) / 2 + (b - a) / 2 * x < b \longleftrightarrow (b - a) * x < (b - a)$  for x
    by (auto simp: field_simps)
  show range f  $\subseteq S$ 
    using shrink_range [of UNIV] <a < b>
  unfolding subset_iff f_def greaterThanLessThan_iff image_iff
  by (smt (verit, best) S * mult_less_cancel_left2 mult_minus_right)
qed
qed

lemma reals01_lepoll_nat_sets:  $\{0..<1::real\} \lesssim (UNIV::nat\ set\ set)$ 
proof -
  define nxt where  $nxt \equiv \lambda x::real. \text{if } x < 1/2 \text{ then } (True, 2*x) \text{ else } (False, 2*x - 1)$ 
  have nxt_fun:  $nxt \in \{0..<1\} \rightarrow UNIV \times \{0..<1\}$ 
    by (simp add: nxt_def Pi_iff)
  define  $\sigma$  where  $\sigma \equiv \lambda x. \text{rec\_nat } (True, x) (\lambda n (b,y). \text{nxt } y)$ 
  have  $\sigma$ Suc [simp]:  $\sigma\ x\ (Suc\ k) = \text{nxt}\ (\text{snd}\ (\sigma\ x\ k))$  for k
    by (simp add:  $\sigma$ _def case_prod_beta')
  have  $\sigma$ 01:  $x \in \{0..<1\} \implies \sigma\ x\ n \in UNIV \times \{0..<1\}$  for x n
  proof (induction n)
    case 0
    then show ?case
      by (simp add:  $\sigma$ _def)
    next
    case (Suc n)
    with nxt_fun show ?case
      by (force simp add: Pi_iff split: prod.split)
  qed
  define f where  $f \equiv \lambda x. \{n. \text{fst } (\sigma\ x\ (Suc\ n))\}$ 
  have snd_nxt:  $\text{snd}\ (\text{nxt}\ y) - \text{snd}\ (\text{nxt}\ x) = 2 * (y-x)$ 
    if  $\text{fst}\ (\text{nxt}\ x) = \text{fst}\ (\text{nxt}\ y)$  for x y
    using that by (simp add: nxt_def split: if_split_asm)
  have False if  $f\ x = f\ y\ x < y\ 0 \leq x\ x < 1\ 0 \leq y\ y < 1$  for x y :: real
  proof -
    have  $\bigwedge k. \text{fst}\ (\sigma\ x\ (Suc\ k)) = \text{fst}\ (\sigma\ y\ (Suc\ k))$ 
      using that by (force simp add: f_def)

```

```

then have eq:  $\bigwedge k. \text{fst} (\text{next} (\text{snd} (\sigma x k))) = \text{fst} (\text{next} (\text{snd} (\sigma y k)))$ 
  by (metis  $\sigma\_def$   $\text{case\_prod\_beta}'$   $\text{rec\_nat\_Suc\_imp}$ )
have *:  $\text{snd} (\sigma y k) - \text{snd} (\sigma x k) = 2^k * (y-x)$  for k
proof (induction k)
  case 0
  then show ?case
    by (simp add:  $\sigma\_def$ )
next
  case (Suc k)
  then show ?case
    by (simp add:  $\text{eq\_snd\_next}$ )
qed
define n where  $n \equiv \text{nat} (\lceil \log 2 (1 / (y - x)) \rceil)$ 
have  $2^n \geq 1 / (y - x)$ 
  by (simp add:  $n\_def$   $\text{power\_of\_nat\_log\_ge}$ )
then have  $2^n * (y-x) \geq 1$ 
  using  $\langle x < y \rangle$  by (simp add:  $n\_def$   $\text{field\_simps}$ )
with * have  $\text{snd} (\sigma y n) - \text{snd} (\sigma x n) \geq 1$ 
  by presburger
moreover have  $\text{snd} (\sigma x n) \in \{0..<1\}$   $\text{snd} (\sigma y n) \in \{0..<1\}$ 
  using that by (meson  $\sigma 01$   $\text{atLeastLessThan\_iff}$   $\text{mem\_Times\_iff}$ ) +
ultimately show False by simp
qed
then have  $\text{inj\_on } f \{0..<1\}$ 
  by (meson  $\text{atLeastLessThan\_iff}$   $\text{linorder\_inj\_onI}'$ )
then show ?thesis
  unfolding  $\text{lepoll\_def}$  by blast
qed

lemma  $\text{nat\_sets\_lepoll\_reals01}$ :  $(\text{UNIV}::\text{nat set set}) \lesssim \{0..<1::\text{real}\}$ 
proof -
  define F where  $F \equiv \lambda S i. \text{if } i \in S \text{ then } (\text{inverse } 3::\text{real})^i \text{ else } 0$ 
  have  $F_{ge0}$ :  $F S i \geq 0$  for S i
    by (simp add:  $F\_def$ )
  have F:  $\text{summable} (F S)$  for S
    unfolding  $F\_def$  by (force intro:  $\text{summable\_comparison\_test\_ev}$  [where  $g =$ 
 $\text{power} (\text{inverse } 3)$ ])
  have  $\text{sum} (F S) \{..<n\} \leq 3/2$  for n S
  proof (cases n)
    case (Suc n')
    have  $\text{sum} (F S) \{..<n\} \leq (\sum i < n. \text{inverse } 3^i)$ 
      by (simp add:  $F\_def$   $\text{sum\_mono}$ )
    also have  $\dots = (\sum i=0..n'. \text{inverse } 3^i)$ 
      using  $\text{Suc atLeast0AtMost lessThan\_Suc\_atMost}$  by presburger
    also have  $\dots = (3/2) * (1 - \text{inverse } 3^n)$ 
      using  $\text{sum\_gp\_multiplied}$  [of 0 n'  $\text{inverse } (3::\text{real})$ ] by (simp add:  $\text{Suc}$ 
 $\text{field\_simps}$ )
    also have  $\dots \leq 3/2$ 
      by (simp add:  $\text{field\_simps}$ )
  end

```

```

    finally show ?thesis .
qed auto
then have F32:  $\text{suminf } (F S) \leq 3/2$  for  $S$ 
  using  $F \text{ suminf\_le\_const}$  by blast
define  $f$  where  $f \equiv \lambda S. \text{suminf } (F S) / 2$ 
have  $\text{mono}F: F S n \leq F T n$  if  $S \subseteq T$  for  $S T n$ 
  using  $F\_def$  that by auto
then have  $\text{monof}: f S \leq f T$  if  $S \subseteq T$  for  $S T$ 
  using that  $F$  by ( $\text{simp add: } f\_def \text{ suminf\_le}$ )
have  $f S \in \{0..<1::\text{real}\}$  for  $S$ 
proof -
  have  $0 \leq \text{suminf } (F S)$ 
    using  $F$  by ( $\text{simp add: } F\_def \text{ suminf\_nonneg}$ )
  with  $F32[\text{of } S]$  show ?thesis
    by ( $\text{auto simp: } f\_def$ )
qed
moreover have  $\text{inj } f$ 
proof
  fix  $S T$ 
  assume  $f S = f T$ 
  show  $S = T$ 
  proof (rule ccontr)
    assume  $S \neq T$ 
    then have  $ST\_ne: (S-T) \cup (T-S) \neq \{\}$ 
      by blast
    define  $n$  where  $n \equiv \text{LEAST } n. n \in (S-T) \cup (T-S)$ 
    have  $\text{sum\_split}: \text{suminf } (F U) = \text{sum } (F U) \{..<Suc\ n\} + (\sum k. F U (k + Suc\ n))$  for  $U$ 
      by ( $\text{metis } F \text{ add.commute suminf\_split\_initial\_segment}$ )
    have  $\text{yes}: f U \geq (\text{sum } (F U) \{..<n\} + (\text{inverse } 3::\text{real}) ^ n) / 2$ 
      if  $n \in U$  for  $U$ 
    proof -
      have  $0 \leq (\sum k. F U (k + Suc\ n))$ 
        by ( $\text{metis } F \text{ Fge0 suminf\_nonneg summable\_iff\_shift}$ )
      moreover have  $F U n = (1/3) ^ n$ 
        by ( $\text{simp add: } F\_def$  that)
      ultimately show ?thesis
        by ( $\text{simp add: sum\_split } f\_def$ )
    qed
    have *:  $(\sum k. F UNIV (k + n)) = (\sum k. F UNIV k) * (\text{inverse } 3::\text{real}) ^ n$ 
    for  $n$ 
      by ( $\text{simp add: } F\_def \text{ power\_add suminf\_mult2}$ )
    have  $\text{no}: f U < (\text{sum } (F U) \{..<n\} + (\text{inverse } 3::\text{real}) ^ n) / 2$ 
      if  $n \notin U$  for  $U$ 
    proof -
      have [ $\text{simp}$ ]:  $F U n = 0$ 
        by ( $\text{simp add: } F\_def$  that)
      have  $(\sum k. F U (k + Suc\ n)) \leq (\sum k. F UNIV (k + Suc\ n))$ 
        by ( $\text{metis } F \text{ monoF subset\_UNIV suminf\_le summable\_ignore\_initial\_segment}$ )

```

```

then have  $\text{suminf } (F U) \leq (\sum k. F UNIV (k + Suc n)) + (\sum i < n. F U i)$ 
  by (simp add: sum_split)
also have  $\dots < (\text{inverse } 3 :: \text{real}) ^ n + (\sum i < n. F U i)$ 
  unfolding * using F32[of UNIV] by simp
finally have  $\text{suminf } (F U) < \text{inverse } 3 ^ n + \text{sum } (F U) \{..<n\}$  .
then show ?thesis
  by (simp add: f_def)
qed
have  $S \cap \{..<n\} = T \cap \{..<n\}$ 
  using not_less_Least by (fastforce simp add: n_def)
then have  $\text{sum } (F S) \{..<n\} = \text{sum } (F T) \{..<n\}$ 
  by (metis (no_types, lifting) F_def Int_iff sum.cong)
moreover consider  $n \in S - T \mid n \in T - S$ 
  by (metis LeastI_ex ST_ne UnE ex_in_conv n_def)
ultimately show False
  by (smt (verit, best) Diff_iff ‹f S = f T› yes no)
qed
qed
ultimately show ?thesis
  by (meson image_subsetI lepoll_def)
qed

lemma open_interval_eqpoll_reals:
  fixes  $a b :: \text{real}$ 
  shows  $\{a < .. < b\} \approx (\text{UNIV} :: \text{real set}) \longleftrightarrow a < b$ 
  using bij_betw_tan bij_betw_open_intervals eqpoll_def
  by (smt (verit, best) UNIV_I eqpoll_real_subset eqpoll_iff_bijections greaterThanLessThan_iff)

lemma closed_interval_eqpoll_reals:
  fixes  $a b :: \text{real}$ 
  shows  $\{a..b\} \approx (\text{UNIV} :: \text{real set}) \longleftrightarrow a < b$ 
proof
  show  $\{a..b\} \approx (\text{UNIV} :: \text{real set}) \implies a < b$ 
    using eqpoll_finite_iff infinite_Icc_iff infinite_UNIV_char_0 by blast
qed (auto simp: eqpoll_real_subset)

lemma reals_lepoll_reals01:  $(\text{UNIV} :: \text{real set}) \lesssim \{0..<1 :: \text{real}\}$ 
proof –
  have  $(\text{UNIV} :: \text{real set}) \approx \{0 < .. < 1 :: \text{real}\}$ 
    by (simp add: open_interval_eqpoll_reals eqpoll_sym)
  also have  $\dots \lesssim \{0..<1 :: \text{real}\}$ 
    by (simp add: greaterThanLessThan_subseteq_atLeastLessThan_iff subset_imp_lepoll)
  finally show ?thesis .
qed

lemma nat_sets_eqpoll_reals:  $(\text{UNIV} :: \text{nat set set}) \approx (\text{UNIV} :: \text{real set})$ 
  by (metis (mono_tags, opaque_lifting) reals_lepoll_reals01 lepoll_antisym lepoll_trans)

```



```
nat_sets_lepoll_reals01 reals01_lepoll_nat_sets subset_UNIV subset_imp_lepoll)
```

```
end
```

2.4 Connected Components

```
theory Connected
```

```
imports
```

```
  Abstract_Topology_2
```

```
begin
```

2.4.1 Connectedness

```
lemma connected_local:
```

```
connected S  $\longleftrightarrow$ 
```

```
   $\neg$  ( $\exists e1 e2.$ 
```

```
    openin (top_of_set S) e1  $\wedge$ 
```

```
    openin (top_of_set S) e2  $\wedge$ 
```

```
    S  $\subseteq$  e1  $\cup$  e2  $\wedge$ 
```

```
    e1  $\cap$  e2 = {}  $\wedge$ 
```

```
    e1  $\neq$  {}  $\wedge$ 
```

```
    e2  $\neq$  {})
```

```
using connected_openin by blast
```

```
lemma exists_diff:
```

```
fixes P :: 'a set  $\Rightarrow$  bool
```

```
shows ( $\exists S. P (- S)$ )  $\longleftrightarrow$  ( $\exists S. P S$ )
```

```
by (metis boolean_algebra_class.boolean_algebra.double_compl)
```

```
lemma connected_clopen: connected S  $\longleftrightarrow$ 
```

```
( $\forall T. openin (top_of_set S) T \wedge$ 
```

```
  closedin (top_of_set S) T  $\longrightarrow$  T = {}  $\vee$  T = S) (is ?lhs  $\longleftrightarrow$  ?rhs)
```

```
proof -
```

```
  have  $\neg$  connected S  $\longleftrightarrow$ 
```

```
    ( $\exists e1 e2. open e1 \wedge open (- e2) \wedge S \subseteq e1 \cup (- e2) \wedge e1 \cap (- e2) \cap S = \{\}$ 
```

```
 $\wedge e1 \cap S \neq \{\} \wedge (- e2) \cap S \neq \{\}$ )
```

```
  unfolding connected_def openin_open closedin_closed
```

```
  by (metis double_complement)
```

```
  then have th0: connected S  $\longleftrightarrow$ 
```

```
     $\neg$  ( $\exists e2 e1. closed e2 \wedge open e1 \wedge S \subseteq e1 \cup (- e2) \wedge e1 \cap (- e2) \cap S = \{\}$ 
```

```
 $\wedge e1 \cap S \neq \{\} \wedge (- e2) \cap S \neq \{\}$ )
```

```
    (is  $\_ \longleftrightarrow \neg$  ( $\exists e2 e1. ?P e2 e1$ ))
```

```
    by (simp add: closed_def) metis
```

```
  have th1: ?rhs  $\longleftrightarrow$   $\neg$  ( $\exists t' t. closed t' \wedge t = S \cap t' \wedge t \neq \{\} \wedge t \neq S \wedge (\exists t'. open t' \wedge t = S \cap t')$ )
```

```
    (is  $\_ \longleftrightarrow \neg$  ( $\exists t' t. ?Q t' t$ ))
```

```
  unfolding connected_def openin_open closedin_closed by auto
```

```
  have ( $\exists e1. ?P e2 e1$ )  $\longleftrightarrow$  ( $\exists t. ?Q e2 t$ ) for e2
```

```
proof -
```

```

have ?P e2 e1  $\longleftrightarrow$  ( $\exists t. \text{closed } e2 \wedge t = S \cap e2 \wedge \text{open } e1 \wedge t = S \cap e1 \wedge t \neq \{\}$ 
 $\wedge t \neq S$ ) for e1
  by auto
  then show ?thesis
    by metis
qed
then have  $\forall e2. (\exists e1. ?P e2 e1) \longleftrightarrow (\exists t. ?Q e2 t)$ 
  by blast
then show ?thesis
  by (simp add: th0 th1)
qed

```

2.4.2 Connected components, considered as a connectedness relation or a set

definition *connected_component* $S x y \equiv \exists T. \text{connected } T \wedge T \subseteq S \wedge x \in T \wedge y \in T$

abbreviation *connected_component_set* $S x \equiv \text{Collect } (\text{connected_component } S x)$

lemma *connected_componentI*:
 $\text{connected } T \implies T \subseteq S \implies x \in T \implies y \in T \implies \text{connected_component } S x y$
by (auto simp: connected_component_def)

lemma *connected_component_in*: $\text{connected_component } S x y \implies x \in S \wedge y \in S$
by (auto simp: connected_component_def)

lemma *connected_component_refl*: $x \in S \implies \text{connected_component } S x x$
using connected_component_def connected_sing **by** blast

lemma *connected_component_refl_eq* [simp]: $\text{connected_component } S x x \longleftrightarrow x \in S$
using connected_component_in connected_component_refl **by** blast

lemma *connected_component_sym*: $\text{connected_component } S x y \implies \text{connected_component } S y x$
by (auto simp: connected_component_def)

lemma *connected_component_trans*:
 $\text{connected_component } S x y \implies \text{connected_component } S y z \implies \text{connected_component } S x z$
unfolding connected_component_def
by (metis Int_iff Un_iff Un_subset_iff equals0D connected_Un)

lemma *connected_component_of_subset*:
 $\text{connected_component } S x y \implies S \subseteq T \implies \text{connected_component } T x y$
by (auto simp: connected_component_def)

lemma *connected_component_Union*: $\text{connected_component_set } S x = \bigcup \{T. \text{connected } T \wedge x \in T \wedge T \subseteq S\}$

by (*auto simp: connected_component_def*)

lemma *connected_connected_component* [*iff*]: $\text{connected } (\text{connected_component_set } S x)$

by (*auto simp: connected_component_Union intro: connected_Union*)

lemma *connected_iff_eq_connected_component_set*:

$\text{connected } S \longleftrightarrow (\forall x \in S. \text{connected_component_set } S x = S)$

by (*metis connected_component_def connected_component_in connected_connected_component mem_Collect_eq subsetI subset_antisym*)

lemma *connected_component_subset*: $\text{connected_component_set } S x \subseteq S$

using *connected_component_in* **by** *blast*

lemma *connected_component_eq_self*: $\text{connected } S \Longrightarrow x \in S \Longrightarrow \text{connected_component_set } S x = S$

by (*simp add: connected_iff_eq_connected_component_set*)

lemma *connected_iff_connected_component*:

$\text{connected } S \longleftrightarrow (\forall x \in S. \forall y \in S. \text{connected_component } S x y)$

using *connected_component_in* **by** (*auto simp: connected_iff_eq_connected_component_set*)

lemma *connected_component_maximal*:

$x \in T \Longrightarrow \text{connected } T \Longrightarrow T \subseteq S \Longrightarrow T \subseteq (\text{connected_component_set } S x)$

using *connected_component_eq_self connected_component_of_subset* **by** *blast*

lemma *connected_component_mono*:

$S \subseteq T \Longrightarrow \text{connected_component_set } S x \subseteq \text{connected_component_set } T x$

by (*simp add: Collect_mono connected_component_of_subset*)

lemma *connected_component_eq_empty* [*simp*]: $\text{connected_component_set } S x = \{\} \longleftrightarrow x \notin S$

using *connected_component_refl* **by** (*fastforce simp: connected_component_in*)

lemma *connected_component_set_empty* [*simp*]: $\text{connected_component_set } \{\} x = \{\}$

using *connected_component_eq_empty* **by** *blast*

lemma *connected_component_eq*:

$y \in \text{connected_component_set } S x \Longrightarrow (\text{connected_component_set } S y = \text{connected_component_set } S x)$

by (*metis (no_types, lifting)*)

Collect_cong connected_component_sym connected_component_trans mem_Collect_eq)

lemma *closed_connected_component*:

assumes *S: closed S*

```

shows closed (connected_component_set S x)
proof (cases x ∈ S)
  case False
  then show ?thesis
    by (metis connected_component_eq_empty closed_empty)
next
  case True
  show ?thesis
    unfolding closure_eq [symmetric]
  proof
    show closure (connected_component_set S x) ⊆ connected_component_set S
x
    proof (rule connected_component_maximal)
      show x ∈ closure (connected_component_set S x)
        by (simp add: closure_def True)
      show connected (closure (connected_component_set S x))
        by (simp add: connected_imp_connected_closure)
      show closure (connected_component_set S x) ⊆ S
        by (simp add: S_closure_minimal connected_component_subset)
    qed
  qed (simp add: closure_subset)
qed

```

```

lemma connected_component_disjoint:
  connected_component_set S a ∩ connected_component_set S b = {} ↔
  a ∉ connected_component_set S b
  by (smt (verit) connected_component_eq connected_component_eq_empty connected_component_refl_eq
    disjoint_iff_not_equal mem_Collect_eq)

```

```

lemma connected_component_nonoverlap:
  connected_component_set S a ∩ connected_component_set S b = {} ↔
  a ∉ S ∨ b ∉ S ∨ connected_component_set S a ≠ connected_component_set
S b
  by (metis connected_component_disjoint connected_component_eq connected_component_eq_empty
inf.idem)

```

```

lemma connected_component_overlap:
  connected_component_set S a ∩ connected_component_set S b ≠ {} ↔
  a ∈ S ∧ b ∈ S ∧ connected_component_set S a = connected_component_set
S b
  by (auto simp: connected_component_nonoverlap)

```

```

lemma connected_component_sym_eq: connected_component S x y ↔ connected_component
S y x
  using connected_component_sym by blast

```

```

lemma connected_component_eq_eq:
  connected_component_set S x = connected_component_set S y ↔

```

$x \notin S \wedge y \notin S \vee x \in S \wedge y \in S \wedge \text{connected_component } S \ x \ y$
by (*metis* *connected_component_eq* *connected_component_eq_empty* *connected_component_refl* *mem_Collect_eq*)

lemma *connected_iff_connected_component_eq*:

$\text{connected } S \longleftrightarrow (\forall x \in S. \forall y \in S. \text{connected_component_set } S \ x = \text{connected_component_set } S \ y)$

by (*simp* *add*: *connected_component_eq_eq* *connected_iff_connected_component*)

lemma *connected_component_idemp*:

$\text{connected_component_set } (\text{connected_component_set } S \ x) \ x = \text{connected_component_set } S \ x$

by (*metis* *Int_absorb* *connected_component_disjoint* *connected_component_eq_empty* *connected_component_eq_self* *connected_connected_component*)

lemma *connected_component_unique*:

$\llbracket x \in c; c \subseteq S; \text{connected } c; \wedge c'. \llbracket x \in c'; c' \subseteq S; \text{connected } c' \rrbracket \implies c' \subseteq c \rrbracket$
 $\implies \text{connected_component_set } S \ x = c$

by (*meson* *connected_component_maximal* *connected_component_subset* *connected_connected_component_subsetD* *subset_antisym*)

lemma *joinable_connected_component_eq*:

$\llbracket \text{connected } T; T \subseteq S; \text{connected_component_set } S \ x \cap T \neq \{\}; \text{connected_component_set } S \ y \cap T \neq \{\} \rrbracket$
 $\implies \text{connected_component_set } S \ x = \text{connected_component_set } S \ y$

by (*metis* (*full_types*) *subsetD* *connected_component_eq* *connected_component_maximal* *disjoint_iff_not_equal*)

lemma *Union_connected_component*: $\bigcup (\text{connected_component_set } S \ ` S) = S$

proof

show $\bigcup (\text{connected_component_set } S \ ` S) \subseteq S$

by (*simp* *add*: *SUP_least* *connected_component_subset*)

qed (*use* *connected_component_refl_eq* **in** *force*)

lemma *complement_connected_component_unions*:

$S - \text{connected_component_set } S \ x = \bigcup (\text{connected_component_set } S \ ` S - \{\text{connected_component_set } S \ x\})$
(is ?lhs = ?rhs)

proof

show *?lhs* \subseteq *?rhs*

by (*metis* *Diff_subset* *Diff_subset_conv* *Sup_insert* *Union_connected_component_insert_Diff_single*)

show *?rhs* \subseteq *?lhs*

by *clarsimp* (*metis* *connected_component_eq_eq* *connected_component_in*)

qed

lemma *connected_component_intermediate_subset*:
 $\llbracket \text{connected_component_set } U \ a \subseteq T; T \subseteq U \rrbracket$
 $\implies \text{connected_component_set } T \ a = \text{connected_component_set } U \ a$
by (*metis connected_component_idemp connected_component_mono subset_antisym*)

lemma *connected_component_homeomorphismI*:
assumes *homeomorphism* $A \ B \ f \ g$ *connected_component* $A \ x \ y$
shows *connected_component* $B \ (f \ x) \ (f \ y)$
proof –
from *assms* **obtain** T **where** $T: \text{connected } T \ T \subseteq A \ x \in T \ y \in T$
unfolding *connected_component_def* **by** *blast*
have *connected* $(f \ ' T) \ f \ ' T \subseteq B \ f \ x \in f \ ' T \ f \ y \in f \ ' T$
using *assms* T *continuous_on_subset*[*of* $A \ f \ T$]
by (*auto intro!*: *connected_continuous_image simp: homeomorphism_def*)
thus *?thesis*
unfolding *connected_component_def* **by** *blast*
qed

lemma *connected_component_homeomorphism_iff*:
assumes *homeomorphism* $A \ B \ f \ g$
shows *connected_component* $A \ x \ y \longleftrightarrow x \in A \ \wedge \ y \in A \ \wedge \ \text{connected_component}$
 $B \ (f \ x) \ (f \ y)$
by (*metis assms connected_component_homeomorphismI connected_component_in*
homeomorphism_apply1 homeomorphism_sym)

lemma *connected_component_set_homeomorphism*:
assumes *homeomorphism* $A \ B \ f \ g \ x \in A$
shows *connected_component_set* $B \ (f \ x) = f \ ' \text{connected_component_set } A \ x$
(is ?lhs = ?rhs)
proof –
have $y \in ?lhs \longleftrightarrow y \in ?rhs$ **for** y
by (*smt (verit, best) assms connected_component_homeomorphism_iff homeomorphism_def image_iff mem_Collect_eq*)
thus *?thesis*
by *blast*
qed

2.4.3 The set of connected components of a set

definition *components*:: $'a::\text{topological_space}$ *set* $\Rightarrow 'a$ *set set*
where *components* $S \equiv \text{connected_component_set } S \ ' S$

lemma *components_iff*: $S \in \text{components } U \longleftrightarrow (\exists x. x \in U \ \wedge \ S = \text{connected_component_set } U \ x)$
by (*auto simp: components_def*)

lemma *componentsI*: $x \in U \implies \text{connected_component_set } U \ x \in \text{components } U$
by (*auto simp: components_def*)

lemma *componentsE*:

assumes $S \in \text{components } U$

obtains x **where** $x \in U$ $S = \text{connected_component_set } U x$

using *assms* **by** (*auto simp: components_def*)

lemma *Union_components* [*simp*]: $\bigcup (\text{components } U) = U$

by (*simp add: Union_connected_component components_def*)

lemma *pairwise_disjoint_components*: *pairwise* $(\lambda X Y. X \cap Y = \{\})$ (*components* U)

unfolding *pairwise_def*

by (*metis (full_types) components_iff connected_component_nonoverlap*)

lemma *in_components_nonempty*: $C \in \text{components } S \implies C \neq \{\}$

by (*metis components_iff connected_component_eq_empty*)

lemma *in_components_subset*: $C \in \text{components } S \implies C \subseteq S$

using *Union_components* **by** *blast*

lemma *in_components_connected*: $C \in \text{components } S \implies \text{connected } C$

by (*metis components_iff connected_connected_component*)

lemma *in_components_maximal*:

$C \in \text{components } S \iff$

$C \neq \{\} \wedge C \subseteq S \wedge \text{connected } C \wedge (\forall d. d \neq \{\} \wedge C \subseteq d \wedge d \subseteq S \wedge \text{connected } d \implies d = C)$

(**is** *?lhs* \iff *?rhs*)

proof

assume *L*: *?lhs*

then have $C \subseteq S$ *connected* C

by (*simp_all add: in_components_subset in_components_connected*)

then show *?rhs*

by (*metis (full_types) L components_iff connected_component_maximal connected_component_refl empty_iff mem_Collect_eq subsetD subset_antisym*)

next

show *?rhs* \implies *?lhs*

by (*metis bot.extremum_uniqueI components_iff connected_component_eq_empty connected_component_maximal connected_component_subset connected_connected_component_subset_emptyI*)

qed

lemma *joinable_components_eq*:

$\text{connected } T \wedge T \subseteq S \wedge c1 \in \text{components } S \wedge c2 \in \text{components } S \wedge c1 \cap T \neq \{\} \wedge c2 \cap T \neq \{\} \implies c1 = c2$

by (*metis (full_types) components_iff joinable_connected_component_eq*)

lemma *closed_components*: $\llbracket \text{closed } S; C \in \text{components } S \rrbracket \implies \text{closed } C$

by (*metis closed_connected_component components_iff*)

lemma *components_nonoverlap*:

$\llbracket C \in \text{components } S; C' \in \text{components } S \rrbracket \implies (C \cap C' = \{\}) \longleftrightarrow (C \neq C')$
by (*metis (full_types) components_iff connected_component_nonoverlap*)

lemma *components_eq*: $\llbracket C \in \text{components } S; C' \in \text{components } S \rrbracket \implies (C = C' \longleftrightarrow C \cap C' \neq \{\})$

by (*metis components_nonoverlap*)

lemma *components_eq_empty* [*simp*]: $\text{components } S = \{\} \longleftrightarrow S = \{\}$

by (*simp add: components_def*)

lemma *components_empty* [*simp*]: $\text{components } \{\} = \{\}$

by *simp*

lemma *connected_eq_connected_components_eq*: $\text{connected } S \longleftrightarrow (\forall C \in \text{components } S. \forall C' \in \text{components } S. C = C')$

by (*metis (no_types, opaque_lifting) components_iff connected_component_eq_eq connected_iff_connected_component*)

lemma *components_eq_singleton_iff*: $\text{components } S = \{S\} \longleftrightarrow \text{connected } S \wedge S \neq \{\}$
(is ?lhs \longleftrightarrow ?rhs)

proof

show *?rhs \implies ?lhs*

by (*metis components_iff connected_component_eq_self equals0I insert_iff mk_disjoint_insert*)

qed (*use in_components_connected in fastforce*)

lemma *components_eq_singleton_exists*: $(\exists a. \text{components } S = \{a\}) \longleftrightarrow \text{connected } S \wedge S \neq \{\}$

by (*metis Union_components ccpo_Sup_singleton components_eq_singleton_iff*)

lemma *connected_eq_components_subset_singleton*: $\text{connected } S \longleftrightarrow \text{components } S \subseteq \{S\}$

by (*metis components_eq_empty components_eq_singleton_iff connected_empty_subset_singleton_iff*)

lemma *connected_eq_components_subset_singleton_exists*: $\text{connected } S \longleftrightarrow (\exists a. \text{components } S \subseteq \{a\})$

by (*metis components_eq_singleton_exists connected_eq_components_subset_singleton_subset_singleton_iff*)

lemma *in_components_self*: $S \in \text{components } S \longleftrightarrow \text{connected } S \wedge S \neq \{\}$

by (*metis components_empty components_eq_singleton_iff empty_iff in_components_connected insertI1*)

lemma *components_maximal*: $\llbracket C \in \text{components } S; \text{connected } T; T \subseteq S; C \cap T \neq \{\} \rrbracket \implies T \subseteq C$

by (*smt (verit, best) Int_Un_eq(4) Un_upper1 bot_eq_sup_iff connected_Un*)

in_components_maximal inf.orderE sup.mono sup.orderI)

lemma *exists_component_superset*: $\llbracket T \subseteq S; S \neq \{\}; \text{connected } T \rrbracket \implies \exists C. C \in \text{components } S \wedge T \subseteq C$
by (*meson componentsI connected_component_maximal equalsOI subset_eq*)

lemma *components_intermediate_subset*: $\llbracket S \in \text{components } U; S \subseteq T; T \subseteq U \rrbracket \implies S \in \text{components } T$
by (*smt (verit, best) dual_order.trans in_components_maximal*)

lemma *in_components_unions_complement*: $C \in \text{components } S \implies S - C = \bigcup (\text{components } S - \{C\})$
by (*metis complement_connected_component_unions components_def components_iff*)

lemma *connected_intermediate_closure*:
assumes *cs*: *connected* *S* **and** *st*: $S \subseteq T$ **and** *ts*: $T \subseteq \text{closure } S$
shows *connected* *T*
using *assms* **unfolding** *connected_def*
by (*smt (verit) Int_assoc inf.absorb_iff2 inf_bot_left open_Int_closure_eq_empty*)

lemma *closedin_connected_component*: *closedin* (*top_of_set* *S*) (*connected_component_set* *S* *x*)

proof (*cases connected_component_set S x = {}*)

case *True*

then show *?thesis*

by (*metis closedin_empty*)

next

case *False*

then obtain *y* **where** *y*: *connected_component* *S* *x* *y* **and** $x \in S$

using *connected_component_eq_empty* **by** *blast*

have ***: *connected_component_set* *S* $x \subseteq S \cap \text{closure} (\text{connected_component_set } S x)$

by (*auto simp: closure_def connected_component_in*)

have ****: $x \in \text{closure} (\text{connected_component_set } S x)$

by (*simp add: <x ∈ S> closure_def*)

have $S \cap \text{closure} (\text{connected_component_set } S x) \subseteq \text{connected_component_set } S x$ **if** *connected_component* *S* *x* *y*

proof (*rule connected_component_maximal*)

show *connected* ($S \cap \text{closure} (\text{connected_component_set } S x)$)

using *** *connected_intermediate_closure* **by** *blast*

qed (*use <x ∈ S> ** in auto*)

with *y* ***** **show** *?thesis*

by (*auto simp: closedin_closed*)

qed

lemma *closedin_component*:

$C \in \text{components } S \implies \text{closedin} (\text{top_of_set } S) C$

using *closedin_connected_component componentsE* **by** *blast*

2.4.4 Proving a function is constant on a connected set by proving that a level set is open

lemma *continuous_levelset_openin_cases*:
fixes $f :: _ \Rightarrow 'b::t1_space$
shows $connected\ S \Longrightarrow continuous_on\ S\ f \Longrightarrow$
 $openin\ (top_of_set\ S)\ \{x \in S. f\ x = a\}$
 $\Longrightarrow (\forall x \in S. f\ x \neq a) \vee (\forall x \in S. f\ x = a)$
unfolding *connected_clopen*
using *continuous_closedin_preimage_constant* **by** *auto*

lemma *continuous_levelset_openin*:
fixes $f :: _ \Rightarrow 'b::t1_space$
shows $connected\ S \Longrightarrow continuous_on\ S\ f \Longrightarrow$
 $openin\ (top_of_set\ S)\ \{x \in S. f\ x = a\} \Longrightarrow$
 $(\exists x \in S. f\ x = a) \Longrightarrow (\forall x \in S. f\ x = a)$
using *continuous_levelset_openin_cases*[*of S f*]
by *meson*

lemma *continuous_levelset_open*:
fixes $f :: _ \Rightarrow 'b::t1_space$
assumes $S: connected\ S\ continuous_on\ S\ f$
and $a: open\ \{x \in S. f\ x = a\}\ \exists x \in S. f\ x = a$
shows $\forall x \in S. f\ x = a$
using $a\ continuous_levelset_openin$ [*OF S, of a, unfolded openin_open*]
by *fast*

2.4.5 Preservation of Connectedness

lemma *homeomorphic_connectedness*:
assumes $S\ homeomorphic\ T$
shows $connected\ S \longleftrightarrow connected\ T$
using *assms* **unfolding** *homeomorphic_def* *homeomorphism_def* **by** (*metis connected_continuous_image*)

lemma *connected_monotone_quotient_preimage*:
assumes $connected\ T$
and $contf: continuous_on\ S\ f$ **and** $fm: f\ 'S = T$
and $opT: \bigwedge U. U \subseteq T$
 $\Longrightarrow openin\ (top_of_set\ S)\ (S \cap f\ -' U) \longleftrightarrow$
 $openin\ (top_of_set\ T)\ U$
and $connT: \bigwedge y. y \in T \Longrightarrow connected\ (S \cap f\ -' \{y\})$
shows $connected\ S$
proof (*rule connectedI*)
fix $U\ V$
assume $open\ U$ **and** $open\ V$ **and** $U \cap S \neq \{\}$ **and** $V \cap S \neq \{\}$
and $U \cap V \cap S = \{\}$ **and** $S \subseteq U \cup V$
moreover
have *disjoint*: $f\ '(S \cap U) \cap f\ '(S \cap V) = \{\}$
proof –

```

have False if  $y \in f^{-1}(S \cap U) \cap f^{-1}(S \cap V)$  for  $y$ 
proof -
  have  $y \in T$ 
  using fm that by blast
  show ?thesis
  using connectedD [OF connT [OF ⟨ $y \in T$ ⟩ ⟨open U⟩ ⟨open V⟩]
    ⟨ $S \subseteq U \cup V$ ⟩ ⟨ $U \cap V \cap S = \{\}$ ⟩ that by fastforce
qed
then show ?thesis by blast
qed
ultimately have  $UU: (S \cap f^{-1} f^{-1}(S \cap U)) = S \cap U$  and  $VV: (S \cap f^{-1} f^{-1}(S \cap V)) = S \cap V$ 
  by auto
have opeU: openin (top_of_set T) (f-1(S ∩ U))
  by (metis UU ⟨open U⟩ fm image_Int_subset le_inf_iff opT openin_open_Int)
have opeV: openin (top_of_set T) (f-1(S ∩ V))
  by (metis opT fm VV ⟨open V⟩ openin_open_Int image_Int_subset inf.bounded_iff)
have  $T \subseteq f^{-1}(S \cap U) \cup f^{-1}(S \cap V)$ 
  using  $\langle S \subseteq U \cup V \rangle$  fm by auto
then show False
  using  $\langle \text{connected } T \rangle$  disjoint opeU opeV ⟨ $U \cap S \neq \{\}$ ⟩ ⟨ $V \cap S \neq \{\}$ ⟩
  by (auto simp: connected_openin)
qed

lemma connected_open_monotone_preimage:
  assumes contf: continuous_on S f and fm:  $f^{-1} S = T$ 
  and ST:  $\bigwedge C. \text{openin} (\text{top\_of\_set } S) C \implies \text{openin} (\text{top\_of\_set } T) (f^{-1} C)$ 
  and connT:  $\bigwedge y. y \in T \implies \text{connected} (S \cap f^{-1} \{y\})$ 
  and connected C:  $C \subseteq T$ 
  shows connected (S ∩ f-1 C)
proof -
  have contf': continuous_on (S ∩ f-1 C) f
  by (meson contf continuous_on_subset inf_le1)
  have eqC:  $f^{-1}(S \cap f^{-1} C) = C$ 
  using  $\langle C \subseteq T \rangle$  fm by blast
  show ?thesis
  proof (rule connected_monotone_quotient_preimage [OF ⟨connected C⟩ contf' eqC])
    show connected (S ∩ f-1 C ∩ f-1 {y}) if  $y \in C$  for  $y$ 
    by (metis Int_assoc Int_empty_right Int_insert_right_if1 assms(6) connT in_mono that vimage_Int)
    have  $\bigwedge U. \text{openin} (\text{top\_of\_set } (S \cap f^{-1} C)) U \implies \text{openin} (\text{top\_of\_set } C) (f^{-1} U)$ 
    using open_map_restrict [OF ST ⟨ $C \subseteq T$ ⟩] by metis
    then show  $\bigwedge D. D \subseteq C \implies \text{openin} (\text{top\_of\_set } (S \cap f^{-1} C)) (S \cap f^{-1} C \cap f^{-1} D) = \text{openin} (\text{top\_of\_set } C) D$ 
    using open_map_imp_quotient_map [of (S ∩ f-1 C) f] contf' by (simp add: eqC)
  end
end

```

qed
qed

lemma *connected_closed_monotone_preimage*:
assumes *contf*: *continuous_on* S *f* **and** *fm*: $f \text{ ' } S = T$
and *ST*: $\bigwedge C. \text{closedin } (\text{top_of_set } S) \ C \implies \text{closedin } (\text{top_of_set } T) \ (f \text{ ' } C)$
and *connT*: $\bigwedge y. y \in T \implies \text{connected } (S \cap f \text{ ' } \{y\})$
and *connected* $C \subseteq T$
shows *connected* $(S \cap f \text{ ' } C)$
proof –
have *contf'*: *continuous_on* $(S \cap f \text{ ' } C)$ *f*
by (*meson* *contf* *continuous_on_subset* *inf_le1*)
have *eqC*: $f \text{ ' } (S \cap f \text{ ' } C) = C$
using $\langle C \subseteq T \rangle$ *fm* **by** *blast*
show *?thesis*
proof (*rule* *connected_monotone_quotient_preimage* [*OF* $\langle \text{connected } C \rangle$ *contf'* *eqC*])
show *connected* $(S \cap f \text{ ' } C \cap f \text{ ' } \{y\})$ **if** $y \in C$ **for** y
by (*metis* *Int_assoc* *Int_empty_right* *Int_insert_right_if1* $\langle C \subseteq T \rangle$ *connT* *subsetD* *that* *vimage_Int*)
have $\bigwedge U. \text{closedin } (\text{top_of_set } (S \cap f \text{ ' } C)) \ U \implies \text{closedin } (\text{top_of_set } C) \ (f \text{ ' } U)$
using *closed_map_restrict* [*OF* $_$ *ST* $\langle C \subseteq T \rangle$] **by** *metis*
then show $\bigwedge D. D \subseteq C \implies \text{openin } (\text{top_of_set } (S \cap f \text{ ' } C)) \ (S \cap f \text{ ' } C \cap f \text{ ' } D) = \text{openin } (\text{top_of_set } C) \ D$
using *closed_map_imp_quotient_map* [*of* $(S \cap f \text{ ' } C)$ *f*] *contf'* **by** (*simp* *add*: *eqC*)
qed
qed

2.4.6 Lemmas about components

See Newman IV, 3.3 and 3.4.

lemma *connected_Un_clopen_in_complement*:
fixes $S \ U :: 'a::\text{metric_space}$ *set*
assumes *connected* S *connected* U $S \subseteq U$
and *opeT*: *openin* $(\text{top_of_set } (U - S)) \ T$
and *cloT*: *closedin* $(\text{top_of_set } (U - S)) \ T$
shows *connected* $(S \cup T)$
proof –
have $*$: $\llbracket \bigwedge x \ y. P \ x \ y \longleftrightarrow P \ y \ x; \bigwedge x \ y. P \ x \ y \implies S \subseteq x \vee S \subseteq y; \bigwedge x \ y. \llbracket P \ x \ y; S \subseteq x \rrbracket \implies \text{False} \rrbracket \implies \neg(\exists x \ y. (P \ x \ y))$ **for** P
by *metis*
show *?thesis*
unfolding *connected_closedin_eq*
proof (*rule* $*$)
fix $H1 \ H2$

```

assume  $H$ :  $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ H1 \wedge$ 
            $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ H2 \wedge$ 
            $H1 \cup H2 = S \cup T \wedge H1 \cap H2 = \{\} \wedge H1 \neq \{\} \wedge H2 \neq \{\}$ 
then have  $\text{clo}$ :  $\text{closedin } (\text{top\_of\_set } S) (S \cap H1)$ 
            $\text{closedin } (\text{top\_of\_set } S) (S \cap H2)$ 
by ( $\text{metis } \text{Un\_upper1 } \text{closedin\_closed\_subset } \text{inf\_commute}$ )+
moreover have  $S \cap ((S \cup T) \cap H1) \cup S \cap ((S \cup T) \cap H2) = S$ 
using  $H$  by  $\text{blast}$ 
moreover have  $H1 \cap (S \cap ((S \cup T) \cap H2)) = \{\}$ 
using  $H$  by  $\text{blast}$ 
ultimately have  $S \cap H1 = \{\} \vee S \cap H2 = \{\}$ 
by ( $\text{smt } (\text{verit}) \ \text{Int\_assoc } \langle \text{connected } S \rangle \ \text{connected\_closedin\_eq } \text{inf\_commute}$ 
 $\text{inf\_sup\_absorb}$ )
then show  $S \subseteq H1 \vee S \subseteq H2$ 
using  $H$   $\langle \text{connected } S \rangle$  unfolding  $\text{connected\_closedin}$  by  $\text{blast}$ 
next
fix  $H1 \ H2$ 
assume  $H$ :  $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ H1 \wedge$ 
            $\text{closedin } (\text{top\_of\_set } (S \cup T)) \ H2 \wedge$ 
            $H1 \cup H2 = S \cup T \wedge H1 \cap H2 = \{\} \wedge H1 \neq \{\} \wedge H2 \neq \{\}$ 
and  $S \subseteq H1$ 
then have  $H2T$ :  $H2 \subseteq T$ 
by  $\text{auto}$ 
have  $T \subseteq U$ 
using  $\text{Diff\_iff } \text{opeT } \text{openin\_imp\_subset}$  by  $\text{auto}$ 
with  $\langle S \subseteq U \rangle$  have  $\text{Ueq}$ :  $U = (U - S) \cup (S \cup T)$ 
by  $\text{auto}$ 
have  $\text{openin } (\text{top\_of\_set } ((U - S) \cup (S \cup T))) \ H2$ 
proof ( $\text{rule } \text{openin\_subtopology\_Un}$ )
show  $\text{openin } (\text{top\_of\_set } (S \cup T)) \ H2$ 
by ( $\text{metis } \text{Diff\_cancel } H \ \text{Un\_Diff } \text{Un\_Diff\_Int } \text{closedin\_subset } \text{openin\_closedin\_eq}$ 
 $\text{topspace\_euclidean\_subtopology}$ )
then show  $\text{openin } (\text{top\_of\_set } (U - S)) \ H2$ 
by ( $\text{meson } H2T \ \text{Un\_upper2 } \text{opeT } \text{openin\_subset\_trans } \text{openin\_trans}$ )
qed
moreover have  $\text{closedin } (\text{top\_of\_set } ((U - S) \cup (S \cup T))) \ H2$ 
proof ( $\text{rule } \text{closedin\_subtopology\_Un}$ )
show  $\text{closedin } (\text{top\_of\_set } (U - S)) \ H2$ 
using  $H \ H2T \ \text{cloT } \text{closedin\_subset\_trans}$ 
by ( $\text{blast } \text{intro: } \text{closedin\_subtopology\_Un } \text{closedin\_trans}$ )
qed ( $\text{simp } \text{add: } H$ )
ultimately have  $H2$ :  $H2 = \{\} \vee H2 = U$ 
using  $\text{Ueq } \langle \text{connected } U \rangle$  unfolding  $\text{connected\_clopen}$  by  $\text{metis}$ 
then have  $H2 \subseteq S$ 
by ( $\text{metis } \text{Diff\_partition } H \ \text{Un\_Diff\_cancel } \text{Un\_subset\_iff } \langle H2 \subseteq T \rangle \ \text{assms}(3)$ 
 $\text{inf.orderE } \text{opeT } \text{openin\_imp\_subset}$ )
moreover have  $T \subseteq H2 - S$ 
by ( $\text{metis } (\text{no\_types}) \ H2 \ H \ \text{opeT } \text{openin\_closedin\_eq } \text{topspace\_euclidean\_subtopology}$ )
ultimately show  $\text{False}$ 

```

```

    using H ⟨S ⊆ H1⟩ by blast
  qed blast
qed

```

proposition *component_diff_connected*:

```

  fixes S :: 'a::metric_space set
  assumes connected S connected U S ⊆ U and C: C ∈ components (U - S)
  shows connected (U - C)
  using ⟨connected S⟩ unfolding connected_closedin_eq not_ex de_Morgan_conj
proof clarify
  fix H3 H4
  assume clo3: closedin (top_of_set (U - C)) H3
    and clo4: closedin (top_of_set (U - C)) H4
    and H34: H3 ∪ H4 = U - C H3 ∩ H4 = {} and H3 ≠ {} and H4 ≠ {}
    and * [rule_format]:
    ∀ H1 H2. ¬ closedin (top_of_set S) H1 ∨
      ¬ closedin (top_of_set S) H2 ∨
      H1 ∪ H2 ≠ S ∨ H1 ∩ H2 ≠ {} ∨ ¬ H1 ≠ {} ∨ ¬ H2 ≠ {}
  then have H3 ⊆ U - C and ope3: openin (top_of_set (U - C)) (U - C -
H3)
    and H4 ⊆ U - C and ope4: openin (top_of_set (U - C)) (U - C - H4)
  by (auto simp: closedin_def)
  have C ≠ {} C ⊆ U - S connected C
  using C in_components_nonempty in_components_subset in_components_maximal
by blast+
  have cCH3: connected (C ∪ H3)
  proof (rule connected_Un_clopen_in_complement [OF ⟨connected C⟩ ⟨con-
nected U⟩ _ _ clo3])
    show openin (top_of_set (U - C)) H3
    by (metis Diff_cancel Un_Diff Un_Diff_Int ⟨H3 ∩ H4 = {}⟩ ⟨H3 ∪ H4 =
U - C⟩ ope4)
  qed (use clo3 ⟨C ⊆ U - S⟩ in auto)
  have cCH4: connected (C ∪ H4)
  proof (rule connected_Un_clopen_in_complement [OF ⟨connected C⟩ ⟨con-
nected U⟩ _ _ clo4])
    show openin (top_of_set (U - C)) H4
    by (metis Diff_cancel Diff_triv Int_Un_eq(2) Un_Diff H34 inf_commute
ope3)
  qed (use clo4 ⟨C ⊆ U - S⟩ in auto)
  have closedin (top_of_set S) (S ∩ H3) closedin (top_of_set S) (S ∩ H4)
  using clo3 clo4 ⟨S ⊆ U⟩ ⟨C ⊆ U - S⟩ by (auto simp: closedin_closed)
  moreover have S ∩ H3 ≠ {}
  using components_maximal [OF C cCH3] ⟨C ≠ {}⟩ ⟨C ⊆ U - S⟩ ⟨H3 ≠ {}⟩
⟨H3 ⊆ U - C⟩ by auto
  moreover have S ∩ H4 ≠ {}
  using components_maximal [OF C cCH4] ⟨C ≠ {}⟩ ⟨C ⊆ U - S⟩ ⟨H4 ≠ {}⟩
⟨H4 ⊆ U - C⟩ by auto
  ultimately show False

```

```

    using * [of  $S \cap H3$   $S \cap H4$ ]  $\langle H3 \cap H4 = \{\} \rangle$   $\langle C \subseteq U - S \rangle$   $\langle H3 \cup H4 = U - C \rangle$   $\langle S \subseteq U \rangle$ 
    by auto
qed

```

2.4.7 Constancy of a function from a connected set into a finite, disconnected or discrete set

Still missing: versions for a set that is smaller than \mathbb{R} , or countable.

lemma *continuous_disconnected_range_constant*:

```

assumes  $S$ : connected  $S$ 
and  $conf$ : continuous_on  $S$   $f$ 
and  $fm$ :  $f \in S \rightarrow T$ 
and  $cct$ :  $\bigwedge y. y \in T \implies \text{connected\_component\_set } T \ y = \{y\}$ 
shows  $f$  constant_on  $S$ 
proof (cases  $S = \{\}$ )
case True then show ?thesis
  by (simp add: constant_on_def)
next
case False
then have  $f \text{ ' } S \subseteq \{f \ x\}$  if  $x \in S$  for  $x$ 
  by (metis PiE  $S$   $cct$  connected_component_maximal connected_continuous_image
  [OF  $conf$ ] fm image_eqI
  image_subset_iff that)
with False show ?thesis
  unfolding constant_on_def by blast
qed

```

This proof requires the existence of two separate values of the range type.

lemma *finite_range_constant_imp_connected*:

```

assumes  $\bigwedge f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra}_1.$ 
   $\llbracket \text{continuous\_on } S \ f; \text{finite}(f \text{ ' } S) \rrbracket \implies f$  constant_on  $S$ 
shows connected  $S$ 
proof -
  { fix  $T$   $U$ 
    assume  $clt$ : closedin (top_of_set  $S$ )  $T$ 
      and  $clu$ : closedin (top_of_set  $S$ )  $U$ 
      and  $tue$ :  $T \cap U = \{\}$  and  $tus$ :  $T \cup U = S$ 
    have continuous_on ( $T \cup U$ ) ( $\lambda x. \text{if } x \in T \text{ then } 0 \text{ else } 1$ )
      using  $clt$   $clu$   $tue$  by (intro continuous_on_cases_local) (auto simp: tus)
    then have  $conf$ : continuous_on  $S$  ( $\lambda x. \text{if } x \in T \text{ then } 0 \text{ else } 1$ )
      using  $tus$  by blast
    have  $fi$ : finite (( $\lambda x. \text{if } x \in T \text{ then } 0 \text{ else } 1$ ) '  $S$ )
      by (rule finite_subset [of _  $\{0,1\}$ ]) auto
    have  $T = \{\} \vee U = \{\}$ 
      using assms [OF  $conf$   $fi$ ]  $tus$  [symmetric]
      by (auto simp: Ball_def constant_on_def) (metis IntI empty_iff_one_neq_zero
      tue)
  }

```

```

}
then show ?thesis
  by (simp add: connected_closedin_eq)
qed

end

```

```

theory Function_Topology
imports
  Elementary_Topology
  Abstract_Limits
  Connected
begin

```

2.5 Function Topology

We want to define the general product topology.

The product topology on a product of topological spaces is generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. This is the coarsest topology for which the projection to each factor is continuous.

To form a product of objects in Isabelle/HOL, all these objects should be subsets of a common type 'a. The product is then $\prod_{i \in I} X_i$, the set of elements from 'a such that the i -th coordinate belongs to X_i for all $i \in I$.

Hence, to form a product of topological spaces, all these spaces should be subsets of a common type. This means that type classes can not be used to define such a product if one wants to take the product of different topological spaces (as the type 'a can only be given one structure of topological space using type classes). On the other hand, one can define different topologies (as introduced in *thy*) on one type, and these topologies do not need to share the same maximal open set. Hence, one can form a product of topologies in this sense, and this works well. The big caveat is that it does not interact well with the main body of topology in Isabelle/HOL defined in terms of type classes... For instance, continuity of maps is not defined in this setting.

As the product of different topological spaces is very important in several areas of mathematics (for instance adeles), I introduce below the product topology in terms of topologies, and reformulate afterwards the consequences in terms of type classes (which are of course very handy for applications).

Given this limitation, it looks to me that it would be very beneficial to revamp the theory of topological spaces in Isabelle/HOL in terms of topologies, and keep the statements involving type classes as consequences of more general statements in terms of topologies (but I am probably too naive here).

Here is an example of a reformulation using topologies. Let

$$\begin{aligned} \text{continuous_map } T1 \ T2 \ f = \\ ((\forall U. \text{openin } T2 \ U \longrightarrow \text{openin } T1 \ (f^{-1}U \cap \text{topspace}(T1))) \\ \wedge (f^{-1}(\text{topspace } T1) \subseteq (\text{topspace } T2))) \end{aligned}$$

be the natural continuity definition of a map from the topology $T1$ to the topology $T2$. Then the current *continuous_on* (with type classes) can be redefined as

$$\begin{aligned} \text{continuous_on } s \ f = \\ \text{continuous_map } (\text{top_of_set } s) \ (\text{topology euclidean}) \ f \end{aligned}$$

In fact, I need *continuous_map* to express the continuity of the projection on subfactors for the product topology, in Lemma *continuous_on_restrict_product_topology*, and I show the above equivalence in Lemma *continuous_map_iff_continuous*.

I only develop the basics of the product topology in this theory. The most important missing piece is Tychonov theorem, stating that a product of compact spaces is always compact for the product topology, even when the product is not finite (or even countable).

I realized afterwards that this theory has a lot in common with `~/src/HOL/Library/Finite_Map.thy`.

2.5.1 The product topology

We can now define the product topology, as generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. Equivalently, it is generated by sets which are one open set along one single coordinate, and the whole space along other coordinates. In fact, this is only equivalent for nonempty products, but for the empty product the first formulation is better (the second one gives an empty product space, while an empty product should have exactly one point, equal to *undefined* along all coordinates).

So, we use the first formulation, which moreover seems to give rise to more straightforward proofs.

definition *product_topology*: $('i \Rightarrow ('a \text{ topology})) \Rightarrow ('i \text{ set}) \Rightarrow (('i \Rightarrow 'a) \text{ topology})$
where *product_topology* $T \ I =$
topology_generated_by $\{(\Pi_E \ i \in I. X \ i) \mid X. (\forall i. \text{openin } (T \ i) \ (X \ i)) \wedge \text{finite } \{i. X \ i \neq \text{topspace } (T \ i)\}\}$

abbreviation *powertop_real* $:: 'a \text{ set} \Rightarrow ('a \Rightarrow \text{real}) \text{ topology}$
where *powertop_real* $\equiv \text{product_topology } (\lambda i. \text{euclideanreal})$

The total set of the product topology is the product of the total sets along each coordinate.

proposition *product_topology*:

product_topology $X I =$
topology
(arbitrary union_of
((finite intersection_of
 $(\lambda F. \exists i U. F = \{f. f i \in U\} \wedge i \in I \wedge \text{openin } (X i) U))$
relative_to $(\prod_{E} i \in I. \text{topspace } (X i)))$
(is $_ = \text{topology } (_ \text{ union_of } ((_ \text{ intersection_of } ?\Psi) \text{ relative_to } ?TOP))$
proof $-$

let $?\Omega = (\lambda F. \exists Y. F = \text{Pi}_E I Y \wedge (\forall i. \text{openin } (X i) (Y i)) \wedge \text{finite } \{i. Y i \neq \text{topspace } (X i)\})$

have $*$: *(finite' intersection_of* $?\Omega$) $A = (\text{finite intersection_of } ?\Psi \text{ relative_to } ?TOP) A$ **for** A

proof $-$

have 1 : $\exists U. (\exists \mathcal{U}. \text{finite } \mathcal{U} \wedge \mathcal{U} \subseteq \text{Collect } ?\Psi \wedge \bigcap \mathcal{U} = U) \wedge ?TOP \cap U = \bigcap \mathcal{U}$

if $\mathcal{U}: \mathcal{U} \subseteq \text{Collect } ?\Omega$ **and** *finite'* $\mathcal{U} A = \bigcap \mathcal{U} \neq \{\}$ **for** \mathcal{U}

proof $-$

have $\forall U \in \mathcal{U}. \exists Y. U = \text{Pi}_E I Y \wedge (\forall i. \text{openin } (X i) (Y i)) \wedge \text{finite } \{i. Y i \neq \text{topspace } (X i)\}$

using \mathcal{U} **by** *auto*

then obtain Y **where** $Y: \bigwedge U. U \in \mathcal{U} \implies U = \text{Pi}_E I (Y U) \wedge (\forall i. \text{openin } (X i) (Y U i)) \wedge \text{finite } \{i. (Y U) i \neq \text{topspace } (X i)\}$

by *metis*

obtain U **where** $U \in \mathcal{U}$

using $\langle \mathcal{U} \neq \{\} \rangle$ **by** *blast*

let $?F = \lambda U. (\lambda i. \{f. f i \in Y U i\})$ ‘ $\{i \in I. Y U i \neq \text{topspace } (X i)\}$

show *?thesis*

proof (*intro conjI exI*)

show *finite* $(\bigcup U \in \mathcal{U}. ?F U)$

using $Y \langle \text{finite}' \mathcal{U} \rangle$ **by** *auto*

show $?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F U) = \bigcap \mathcal{U}$

proof

have $*$: $f \in U$

if $U \in \mathcal{U}$ **and** $\forall V \in \mathcal{U}. \forall i. i \in I \wedge Y V i \neq \text{topspace } (X i) \longrightarrow f i \in Y V i$

and $\forall i \in I. f i \in \text{topspace } (X i)$ **and** $f \in \text{extensional } I$ **for** $f U$

by (*smt (verit) PiE_iff Y that*)

show $?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F U) \subseteq \bigcap \mathcal{U}$

by (*auto simp: PiE_iff **)

show $\bigcap \mathcal{U} \subseteq ?TOP \cap \bigcap (\bigcup U \in \mathcal{U}. ?F U)$

using $Y \text{ openin_subset } \langle \text{finite}' \mathcal{U} \rangle$ **by** *fastforce*

qed

qed (*use Y openin_subset in <blast+>*)

qed

have 2 : $\exists \mathcal{U}'. \text{finite}' \mathcal{U}' \wedge \mathcal{U}' \subseteq \text{Collect } ?\Omega \wedge \bigcap \mathcal{U}' = ?TOP \cap \bigcap \mathcal{U}$

if $\mathcal{U}: \mathcal{U} \subseteq \text{Collect } ?\Psi$ **and** *finite* \mathcal{U} **for** \mathcal{U}

proof (*cases* $\mathcal{U} = \{\}$)

case *True*

```

then show ?thesis
  apply (rule_tac x={?TOP} in exI, simp)
  apply (rule_tac x= $\lambda i. \text{topspace } (X i)$  in exI)
  apply force
  done
next
case False
then obtain U where  $U \in \mathcal{U}$ 
  by blast
have  $\forall U \in \mathcal{U}. \exists i Y. U = \{f. f i \in Y\} \wedge i \in I \wedge \text{openin } (X i) Y$ 
  using  $\mathcal{U}$  by auto
then obtain J Y where
   $Y: \bigwedge U. U \in \mathcal{U} \implies U = \{f. f (J U) \in Y U\} \wedge J U \in I \wedge \text{openin } (X (J U)) (Y U)$ 
  by metis
let ?G =  $\lambda U. \prod_E i \in I. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i)$ 
show ?thesis
proof (intro conjI exI)
  show finite (?G 'U) ?G 'U  $\neq \{\}$ 
  using  $\langle \text{finite } \mathcal{U} \rangle \langle U \in \mathcal{U} \rangle$  by blast+
  have *:  $\bigwedge U. U \in \mathcal{U} \implies \text{openin } (X (J U)) (Y U)$ 
  using Y by force
  show ?G 'U  $\subseteq \{PiE I Y \mid Y. (\forall i. \text{openin } (X i) (Y i)) \wedge \text{finite } \{i. Y i \neq \text{topspace } (X i)\}\}$ 
  apply clarsimp
  apply (rule_tac x= $\lambda i. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i)$  in exI)
  apply (auto simp: *)
  done
next
show  $(\bigcap U \in \mathcal{U}. ?G U) = ?TOP \cap \bigcap \mathcal{U}$ 
proof
  have  $(\prod_E i \in I. \text{if } i = J U \text{ then } Y U \text{ else } \text{topspace } (X i)) \subseteq (\prod_E i \in I. \text{topspace } (X i))$ 
  by (simp add: PiE_mono Y  $\langle U \in \mathcal{U} \rangle$  openin_subset)
  then have  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq ?TOP$ 
  using  $\langle U \in \mathcal{U} \rangle$  by fastforce
  moreover have  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq \bigcap \mathcal{U}$ 
  using PiE_mem Y by fastforce
  ultimately show  $(\bigcap U \in \mathcal{U}. ?G U) \subseteq ?TOP \cap \bigcap \mathcal{U}$ 
  by auto
  qed (use Y in fastforce)
qed
qed
show ?thesis
unfolding relative_to_def intersection_of_def
  by (safe; blast dest!: 1 2)
qed
show ?thesis
unfolding product_topology_def generate_topology_on_eq

```

```

apply (rule arg_cong [where f = topology])
apply (rule arg_cong [where f = (union_of)arbitrary])
apply (force simp: *)
done

```

qed

lemma *topspace_product_topology* [simp]:

$topspace (product_topology T I) = (\prod_E i \in I. topspace (T i))$

proof

show $topspace (product_topology T I) \subseteq (\prod_E i \in I. topspace (T i))$

unfolding *product_topology_def topology_generated_by_topspace*

unfolding *topspace_def* **by** *auto*

have $(\prod_E i \in I. topspace (T i)) \in \{(\prod_E i \in I. X i) \mid X. (\forall i. openin (T i) (X i)) \wedge finite \{i. X i \neq topspace (T i)\}\}$

using *openin_topspace not_finite_existsD* **by** *auto*

then show $(\prod_E i \in I. topspace (T i)) \subseteq topspace (product_topology T I)$

unfolding *product_topology_def* **using** *PiE_def* **by** (*auto*)

qed

lemma *product_topology_trivial_iff*:

$product_topology X I = trivial_topology \iff (\exists i \in I. X i = trivial_topology)$

by (*auto simp: PiE_eq_empty_iff simp flip: null_topspace_iff_trivial*)

lemma *topspace_product_topology_alt*:

$topspace (product_topology X I) = \{x \in extensional I. \forall i \in I. x i \in topspace (X i)\}$

by (*fastforce simp: PiE_iff*)

lemma *product_topology_basis*:

assumes $\bigwedge i. openin (T i) (X i) finite \{i. X i \neq topspace (T i)\}$

shows $openin (product_topology T I) (\prod_E i \in I. X i)$

unfolding *product_topology_def*

by (*rule topology_generated_by_Basis*) (*use assms in auto*)

proposition *product_topology_open_contains_basis*:

assumes $openin (product_topology T I) U x \in U$

shows $\exists X. x \in (\prod_E i \in I. X i) \wedge (\forall i. openin (T i) (X i)) \wedge finite \{i. X i \neq topspace (T i)\} \wedge (\prod_E i \in I. X i) \subseteq U$

proof –

define IT **where** $IT \equiv \lambda X. \{i. X i \neq topspace (T i)\}$

have $generate_topology_on \{(\prod_E i \in I. X i) \mid X. (\forall i. openin (T i) (X i)) \wedge finite (IT X)\} U$

using *assms* **unfolding** *product_topology_def IT_def* **by** (*intro openin_toplogy_generated_by*) *auto*

then have $\bigwedge x. x \in U \implies \exists X. x \in (\prod_E i \in I. X i) \wedge (\forall i. openin (T i) (X i)) \wedge finite (IT X) \wedge (\prod_E i \in I. X i) \subseteq U$

proof *induction*

case (*Int U V x*)

then obtain $XU XV$ **where** H :

```

    x ∈ Pi_E I XU ∧ i. openin (T i) (XU i) finite (IT XU) Pi_E I XU ⊆ U
    x ∈ Pi_E I XV ∧ i. openin (T i) (XV i) finite (IT XV) Pi_E I XV ⊆ V
  by (meson Int_iff)
define X where X = (λi. XU i ∩ XV i)
have Pi_E I X ⊆ Pi_E I XU ∩ Pi_E I XV
  by (auto simp add: Pi_E_iff X_def)
then have Pi_E I X ⊆ U ∩ V using H by auto
moreover have ∀ i. openin (T i) (X i)
  unfolding X_def using H by auto
moreover have finite (IT X)
  apply (rule rev_finite_subset[of IT XU ∪ IT XV])
  using H by (auto simp: X_def IT_def)
moreover have x ∈ Pi_E I X
  unfolding X_def using H by auto
ultimately show ?case
  by auto
next
  case (UN K x)
  then obtain k where k ∈ K x ∈ k by auto
  with ⟨k ∈ K⟩ UN show ?case
  by (meson Sup_upper2)
qed auto
then show ?thesis using ⟨x ∈ U⟩ IT_def by blast
qed

lemma product_topology_empty_discrete:
  product_topology T {} = discrete_topology {(λx. undefined)}
by (simp add: subtopology_eq_discrete_topology_sing)

lemma openin_product_topology:
  openin (product_topology X I) =
  arbitrary_union_of
  ((finite_intersection_of (λF. (∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin
  (X i) U)))
  relative_to topspace (product_topology X I))
by (simp add: istopology_subbase_product_topology)

lemma subtopology_product_topology:
  subtopology (product_topology X I) (Π_E i∈I. (S i)) = product_topology (λi.
  subtopology (X i) (S i)) I
proof –
  let ?P = λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (X i) U
  let ?X = Π_E i∈I. topspace (X i)
  have finite_intersection_of ?P relative_to ?X ∩ Pi_E I S =
  finite_intersection_of (?P relative_to ?X ∩ Pi_E I S) relative_to ?X ∩ Pi_E
  I S
  by (rule finite_intersection_of_relative_to)
  also have ... = finite_intersection_of
  ((λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ (openin (X i) relative_to

```

```

S i) U)
      relative_to ?X ∩ PiE I S)
      relative_to ?X ∩ PiE I S
apply (rule arg_cong2 [where f = (relative_to)])
apply (rule arg_cong [where f = (intersection_of)finite])
apply (rule ext)
apply (auto simp: relative_to_def intersection_of_def)
done
finally
have finite intersection_of ?P relative_to ?X ∩ PiE I S =
      finite intersection_of
      (λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ (openin (X i) relative_to S i) U)
      relative_to ?X ∩ PiE I S
      by (metis finite_intersection_of_relative_to)
then show ?thesis
unfolding topology_eq
apply clarify
apply (simp add: openin_product_topology flip: openin_relative_to)
apply (simp add: arbitrary_union_of_relative_to flip: PiE Int)
done
qed

```

```

lemma product_topology_base_alt:
  finite intersection_of (λF. (∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (X i) U))
  relative_to (ΠE i ∈ I. topspace (X i)) =
  (λF. (∃ U. F = ΠE i ∈ I. U i ∧ finite {i ∈ I. U i ≠ topspace(X i)} ∧ (∀ i ∈ I.
  openin (X i) (U i))))
  (is ?lhs = ?rhs)
proof –
  have (∀ F. ?lhs F → ?rhs F)
    unfolding all_relative_to_all_intersection_of_topspace_product_topology
  proof clarify
    fix F
    assume finite F and F ⊆ {f. f i ∈ U} | i U. i ∈ I ∧ openin (X i) U}
    then show ∃ U. (ΠE i ∈ I. topspace (X i)) ∩ ∩ F = ΠE i ∈ I. U i ∧
      finite {i ∈ I. U i ≠ topspace (X i)} ∧ (∀ i ∈ I. openin (X i) (U i))
    proof (induction)
      case (insert F F)
      then obtain U where eq: (ΠE i ∈ I. topspace (X i)) ∩ ∩ F = ΠE i ∈ I. U i
      and fin: finite {i ∈ I. U i ≠ topspace (X i)}
      and ope: ∧ i. i ∈ I ⇒ openin (X i) (U i)
      by auto
      obtain i V where F = {f. f i ∈ V} | i V. i ∈ I ∧ openin (X i) V
      using insert by auto
      let ?U = λ j. U j ∩ (if j = i then V else topspace(X j))
      show ?case
      proof (intro exI conjI)
      show (ΠE i ∈ I. topspace (X i)) ∩ ∩ (insert F F) = ΠE i ∈ I. ?U i
      using eq PiE mem <i ∈ I> by (auto simp: <F = {f. f i ∈ V}>) fastforce

```

```

next
  show finite {i ∈ I. ?U i ≠ topspace (X i)}
    by (rule rev_finite_subset [OF finite.insertI [OF fin]]) auto
next
  show ∀ i ∈ I. openin (X i) (?U i)
    by (simp add: ‹openin (X i) V› ope openin_Int)
qed
qed (auto intro: dest: not_finite_existsD)
qed
moreover have (∀ F. ?rhs F → ?lhs F)
proof clarify
  fix U :: 'a ⇒ 'b set
  assume fin: finite {i ∈ I. U i ≠ topspace (X i)} and ope: ∀ i ∈ I. openin (X i)
(U i)
  let ?U = ⋂ i ∈ {i ∈ I. U i ≠ topspace (X i)}. {x. x i ∈ U i}
  show ?lhs (PiE I U)
    unfolding relative_to_def topspace_product_topology
  proof (intro exI conjI)
    show (finite intersection_of (λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin
(X i) U)) ?U
      using fin ope by (intro finite_intersection_of_Inter finite_intersection_of_inc)
  auto
    show (ΠE i ∈ I. topspace (X i)) ∩ ?U = PiE I U
      using ope openin_subset by fastforce
  qed
qed
ultimately show ?thesis
  by meson
qed

corollary openin_product_topology_alt:
  openin (product_topology X I) S ↔
  (∀ x ∈ S. ∃ U. finite {i ∈ I. U i ≠ topspace (X i)} ∧
  (∀ i ∈ I. openin (X i) (U i)) ∧ x ∈ PiE I U ∧ PiE I U ⊆ S)
  unfolding openin_product_topology_arbitrary_union_of_alt product_topology_base_alt
  topspace_product_topology
  by (smt (verit, best))

lemma closure_of_product_topology:
  (product_topology X I) closure_of (PiE I S) = PiE I (λi. (X i) closure_of (S
i))
proof -
  have *: (∀ T. f ∈ T ∧ openin (product_topology X I) T → (∃ y ∈ PiE I S. y ∈
T))
    ↔ (∀ i ∈ I. ∀ T. f i ∈ T ∧ openin (X i) T → S i ∩ T ≠ {})
  (is ?lhs = ?rhs)
  if top: ∧ i. i ∈ I ⇒ f i ∈ topspace (X i) and ext: f ∈ extensional I for f
proof
  assume L[rule_format]: ?lhs

```

```

show ?rhs
proof clarify
  fix i T
  assume  $i \in I \wedge f i \in T \wedge \text{openin } (X \ i) \ T \ S \ i \cap T = \{\}$ 
  then have  $\text{openin } (\text{product\_topology } X \ I) \ ((\prod_{E \ i \in I} \ \text{topspace } (X \ i)) \cap \{x. \ x \ i \in T\})$ 
  by (force simp: openin_product_topology intro: arbitrary_union_of_inc
  relative_to_inc finite_intersection_of_inc)
  then show False
  using L [of  $\text{topspace } (\text{product\_topology } X \ I) \cap \{f. \ f \ i \in T\} \ S \ i \cap T = \{\}$ ]
   $\langle f \ i \in T \rangle \langle i \in I \rangle$ 
  by (auto simp: top_ext PiE_iff)
qed
next
assume R [rule_format]: ?rhs
show ?lhs
proof (clarsimp simp: openin_product_topology union_of_def arbitrary_def)
  fix  $\mathcal{U} \ U$ 
  assume
     $\mathcal{U}: \mathcal{U} \subseteq \text{Collect}$ 
    (finite_intersection_of  $(\lambda F. \ \exists i \ U. \ F = \{x. \ x \ i \in U\} \wedge i \in I \wedge \text{openin } (X \ i) \ U)$ 
    relative_to
       $(\prod_{E \ i \in I} \ \text{topspace } (X \ i))$ ) and
     $f \in U \ U \in \mathcal{U}$ 
  then have (finite_intersection_of  $(\lambda F. \ \exists i \ U. \ F = \{x. \ x \ i \in U\} \wedge i \in I \wedge$ 
   $\text{openin } (X \ i) \ U)$ 
    relative_to  $(\prod_{E \ i \in I} \ \text{topspace } (X \ i))$ )  $U$ 
  by blast
  with  $\langle f \in U \rangle \langle U \in \mathcal{U} \rangle$ 
  obtain  $\mathcal{T}$  where finite  $\mathcal{T}$ 
  and  $\mathcal{T}: \bigwedge C. \ C \in \mathcal{T} \implies \exists i \in I. \ \exists V. \ \text{openin } (X \ i) \ V \wedge C = \{x. \ x \ i \in V\}$ 
  and  $\text{topspace } (\text{product\_topology } X \ I) \cap \bigcap \mathcal{T} \subseteq U \wedge f \in \text{topspace } (\text{product\_topology } X \ I) \cap \bigcap \mathcal{T}$ 
  apply (clarsimp simp add: relative_to_def intersection_of_def)
  apply (rule that, auto dest!: subsetD)
  done
  then have  $f \in \text{PiE } I \ (\text{topspace } \circ X) \ f \in \bigcap \mathcal{T}$  and  $\text{subU}: \text{PiE } I \ (\text{topspace } \circ X) \cap \bigcap \mathcal{T} \subseteq U$ 
  by (auto simp: PiE_iff)
  have *:  $f \ i \in \text{topspace } (X \ i) \cap \bigcap \{U. \ \text{openin } (X \ i) \ U \wedge \{x. \ x \ i \in U\} \in \mathcal{T}\} \wedge \text{openin } (X \ i) \ (\text{topspace } (X \ i) \cap \bigcap \{U. \ \text{openin } (X \ i) \ U \wedge \{x. \ x \ i \in U\} \in \mathcal{T}\})$ 
  if  $i \in I$  for  $i$ 
  proof -
  have finite  $(\lambda U. \ \{x. \ x \ i \in U\}) - \mathcal{T}$ 
  proof (rule finite_vimageI [OF  $\langle \text{finite } \mathcal{T} \rangle$ ])
  show inj  $(\lambda U. \ \{x. \ x \ i \in U\})$ 
  by (auto simp: inj_on_def)
  qed

```



```

    then have fin: finite {U. openin (X i) U ∧ {x. x i ∈ U} ∈ T}
      by (rule rev_finite_subset) auto
    have openin (X i) (∩ (insert (topspace (X i)) {U. openin (X i) U ∧ {x. x
i ∈ U} ∈ T}))
      by (rule openin_Inter) (auto simp: fin)
    then show ?thesis
      using ⟨f ∈ ∩ T⟩ by (fastforce simp: that top)
  qed
  define Φ where Φ ≡ λi. topspace (X i) ∩ ∩ {U. openin (X i) U ∧ {f. f i ∈
U} ∈ T}
  have ∀i ∈ I. ∃x. x ∈ S i ∩ Φ i
    using R [OF _ *] unfolding Φ_def by blast
  then obtain ∅ where ∅ [rule_format]: ∀i ∈ I. ∅ i ∈ S i ∩ Φ i
    by metis
  show ∃y ∈ PiE I S. ∃x ∈ U. y ∈ x
  proof
    show ∃U ∈ U. (λi ∈ I. ∅ i) ∈ U
  proof
    have restrict ∅ I ∈ PiE I (topspace ∘ X) ∩ ∩ T
      using T by (fastforce simp: Φ_def PiE_def dest: ∅)
    then show restrict ∅ I ∈ U
      using subU by blast
    qed (rule ⟨U ∈ U⟩)
  next
    show (λi ∈ I. ∅ i) ∈ PiE I S
      using ∅ by simp
    qed
  qed
  qed
  show ?thesis
    apply (simp add: * closure_of_def PiE_iff set_eq_iff cong: conj_cong)
    by metis
  qed

```

corollary *closedin_product_topology:*

$closedin (product_topology X I) (PiE I S) \longleftrightarrow PiE I S = \{\} \vee (\forall i \in I. closedin (X i) (S i))$

by (smt (verit, best) PiE_eq closedin_empty closure_of_eq closure_of_product_topology)

corollary *closedin_product_topology_singleton:*

$f \in extensional I \implies closedin (product_topology X I) \{f\} \longleftrightarrow (\forall i \in I. closedin (X i) \{f i\})$

using PiE_singleton closedin_product_topology [of X I]

by (metis (no_types, lifting) all_not_in_conv insertI1)

lemma *product_topology_empty:*

$product_topology X \{\} = topology (\lambda S. S \in \{\{\}, \{\lambda k. undefined\}\})$

unfolding product_topology_union_of_def intersection_of_def arbitrary_def relative_to_def

by (auto intro: arg_cong [where f=topology])

lemma *openin_product_topology_empty*: $openin (product_topology X \{\}) S \longleftrightarrow S \in \{\{\}, \{\lambda k. undefined\}\}$
unfolding *union_of_def intersection_of_def arbitrary_def relative_to_def openin_product_topology*
by *auto*

The basic property of the product topology is the continuity of projections:

lemma *continuous_map_product_coordinates* [*simp*]:
assumes $i \in I$
shows *continuous_map* (product_topology T I) (T i) ($\lambda x. x i$)
proof –
{
fix U **assume** *openin* (T i) U
define X **where** $X = (\lambda j. \text{if } j = i \text{ then } U \text{ else } \text{topspace } (T j))$
then have *: ($\lambda x. x i$) –‘ $U \cap (\prod_{E \ i \in I. \text{topspace } (T i)} = (\prod_{E \ j \in I. X j})$
unfolding X_def **using** *assms openin_subset[OF <openin (T i) U>*
by (auto *simp add: PiE_iff, auto, metis subsetCE*)
have **: ($\forall i. \text{openin } (T i) (X i) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\}$)
unfolding X_def **using** <*openin (T i) U*> **by** *auto*
have *openin* (product_topology T I) (($\lambda x. x i$) –‘ $U \cap (\prod_{E \ i \in I. \text{topspace } (T i)}$)))
unfolding *product_topology_def*
apply (*rule topology_generated_by_Basis*)
apply (*subst **)
using ** **by** *auto*
}
then show ?*thesis* **unfolding** *continuous_map_alt*
by (auto *simp add: assms PiE_iff*)
qed

lemma *continuous_map_coordinatewise_then_product* [*intro*]:
assumes $\bigwedge i. i \in I \implies \text{continuous_map } T1 (T i) (\lambda x. f x i)$
 $\bigwedge i x. i \notin I \implies x \in \text{topspace } T1 \implies f x i = \text{undefined}$
shows *continuous_map* T1 (product_topology T I) f
unfolding *product_topology_def*
proof (*rule continuous_on_generated_topo*)
fix U **assume** $U \in \{PiE \ I \ X \mid X. (\forall i. \text{openin } (T i) (X i) \wedge \text{finite } \{i. X i \neq \text{topspace } (T i)\})\}$
then obtain X **where** $H: U = PiE \ I \ X \wedge \bigwedge i. \text{openin } (T i) (X i) \text{ finite } \{i. X i \neq \text{topspace } (T i)\}$
by *blast*
define J **where** $J = \{i \in I. X i \neq \text{topspace } (T i)\}$
have *finite* $J \subseteq I$ **unfolding** J_def **using** $H(3)$ **by** *auto*
have ($\lambda x. f x i$) –‘ ($\text{topspace } (T i) \cap \text{topspace } T1 = \text{topspace } T1$ **if** $i \in I$ **for** i
using *that assms(1)* **by** (*simp add: continuous_map_preimage_topspace*)
then have *: ($\lambda x. f x i$) –‘ ($X i \cap \text{topspace } T1 = \text{topspace } T1$ **if** $i \in I - J$ **for** i

```

    using that unfolding J_def by auto
    have f- $'U \cap \text{topspace } T1 = (\bigcap i \in I. (\lambda x. f x i) - '(X i) \cap \text{topspace } T1) \cap$ 
    (topspace T1)
    by (subst H(1), auto simp add: PiE_iff assms)
    also have ... =  $(\bigcap i \in J. (\lambda x. f x i) - '(X i) \cap \text{topspace } T1) \cap (\text{topspace } T1)$ 
    using *  $\langle J \subseteq I \rangle$  by auto
    also have openin T1 (...)
    using H(2)  $\langle J \subseteq I \rangle$   $\langle \text{finite } J \rangle$  assms(1) by blast
    ultimately show openin T1  $(f - 'U \cap \text{topspace } T1)$  by simp
next
    have  $f \in \text{topspace } T1 \rightarrow \text{topspace } (\text{product\_topology } T I)$ 
    using assms continuous_map_funspace by (force simp: Pi_iff)
    then show  $f \text{ 'topspace } T1 \subseteq \bigcup \{PiE I X \mid X. (\forall i. \text{openin } (T i) (X i)) \wedge \text{finite}$ 
     $\{i. X i \neq \text{topspace } (T i)\}\}$ 
    by (fastforce simp add: product_topology_def Pi_iff)
qed

```

lemma *continuous_map_product_then_coordinatewise* [intro]:

```

    assumes continuous_map T1 (product_topology T I) f
    shows  $\bigwedge i. i \in I \implies \text{continuous\_map } T1 (T i) (\lambda x. f x i)$ 
     $\bigwedge i x. i \notin I \implies x \in \text{topspace } T1 \implies f x i = \text{undefined}$ 
proof -
    fix i assume  $i \in I$ 
    have  $(\lambda x. f x i) = (\lambda y. y i) \circ f$  by auto
    also have continuous_map T1 (T i) (...)
    by (metis  $\langle i \in I \rangle$  assms continuous_map_compose continuous_map_product_coordinates)
    ultimately show continuous_map T1 (T i)  $(\lambda x. f x i)$ 
    by simp
next
    fix i x assume  $i \notin I$   $x \in \text{topspace } T1$ 
    have  $f x \in \text{topspace } (\text{product\_topology } T I)$ 
    using assms  $\langle x \in \text{topspace } T1 \rangle$  unfolding continuous_map_def by auto
    then have  $f x \in (\Pi_E i \in I. \text{topspace } (T i))$ 
    using topspace_product_topology by metis
    then show  $f x i = \text{undefined}$ 
    using  $\langle i \notin I \rangle$  by (auto simp add: PiE_iff extensional_def)
qed

```

lemma *continuous_on_restrict*:

```

    assumes  $J \subseteq I$ 
    shows continuous_map (product_topology T I) (product_topology T J)  $(\lambda x.$ 
    restrict x J)
proof (rule continuous_map_coordinatewise_then_product)
    fix i assume  $i \in J$ 
    then have  $(\lambda x. \text{restrict } x J i) = (\lambda x. x i)$  unfolding restrict_def by auto
    then show continuous_map (product_topology T I) (T i)  $(\lambda x. \text{restrict } x J i)$ 
    using  $\langle i \in J \rangle$   $\langle J \subseteq I \rangle$  by auto
next
    fix i assume  $i \notin J$ 

```

```

then show restrict x J i = undefined for x::'a ⇒ 'b
  unfolding restrict_def by auto
qed

```

Powers of a single topological space as a topological space, using type classes

```

instantiation fun :: (type, topological_space) topological_space
begin

```

```

definition open_fun_def:
  open U = openin (product_topology (λi. euclidean) UNIV) U

```

instance proof

```

  have topspace (product_topology (λ(i::'a). euclidean::('b topology)) UNIV) =
    UNIV
  unfolding topspace_product_topology topspace_euclidean by auto
  then show open (UNIV::('a ⇒ 'b) set)
  unfolding open_fun_def by (metis openin_tospace)
qed (auto simp add: open_fun_def)

```

end

```

lemma open_PiE [intro?]:
  fixes X::'i ⇒ ('b::topological_space) set
  assumes ∧i. open (X i) finite {i. X i ≠ UNIV}
  shows open (Pi_E UNIV X)
  by (simp add: assms open_fun_def product_topology_basis)

```

lemma euclidean_product_topology:

```

  product_topology (λi. euclidean::('b::topological_space) topology) UNIV = euclidean
by (metis open_openin topology_eq open_fun_def)

```

proposition product_topology_basis':

```

  fixes x::'i ⇒ 'a and U::'i ⇒ ('b::topological_space) set
  assumes finite I ∧i. i ∈ I ⇒ open (U i)
  shows open {f. ∀ i∈I. f (x i) ∈ U i}

```

proof –

```

define V where V ≡ (λy. if y ∈ x'I then ⋂ {U i | i. i ∈ I ∧ x i = y} else UNIV)
define X where X ≡ (λy. if y ∈ x'I then V y else UNIV)
have *: open (X i) for i
  unfolding X_def V_def using assms by auto
then have open (Pi_E UNIV X)
  by (simp add: X_def assms(1) open_PiE)
moreover have Pi_E UNIV X = {f. ∀ i∈I. f (x i) ∈ U i}
  by (fastforce simp add: Pi_E_iff X_def V_def split: if_split_asm)
ultimately show ?thesis by simp
qed

```

The results proved in the general situation of products of possibly different spaces have their counterparts in this simpler setting.

lemma *continuous_on_product_coordinates* [simp]:
continuous_on UNIV ($\lambda x. x \text{ i}::('b::\text{topological_space})$)
using *continuous_map_product_coordinates* [of *UNIV* $\lambda i. \text{euclidean}$]
by (*metis* (*no_types*) *continuous_map_iff_continuous euclidean_product_topology iso_tuple_UNIV_I subtopology_UNIV*)

lemma *continuous_on_coordinatewise_then_product* [*continuous_intros*]:
fixes $f :: 'a::\text{topological_space} \Rightarrow 'b \Rightarrow 'c::\text{topological_space}$
assumes $\bigwedge i. \text{continuous_on } S (\lambda x. f \ x \ i)$
shows *continuous_on* $S \ f$
by (*metis* *UNIV_I assms continuous_map_iff_continuous euclidean_product_topology continuous_map_coordinatewise_then_product*)

lemma *continuous_on_product_then_coordinatewise*:
assumes *continuous_on* $S \ f$
shows *continuous_on* $S (\lambda x. f \ x \ i)$
by (*metis* *UNIV_I assms continuous_map_iff_continuous continuous_map_product_then_coordinatewise(1) euclidean_product_topology*)

lemma *continuous_on_coordinatewise_iff*:
fixes $f :: ('a \Rightarrow \text{real}) \Rightarrow 'b \Rightarrow \text{real}$
shows *continuous_on* $(A \cap S) \ f \longleftrightarrow (\forall i. \text{continuous_on } (A \cap S) (\lambda x. f \ x \ i))$
by (*auto simp: continuous_on_product_then_coordinatewise continuous_on_coordinatewise_then_product*)

lemma *continuous_map_span_sum*:
fixes $B :: 'a::\text{real_normed_vector_set}$
assumes $biB: \bigwedge i. i \in I \implies b \ i \in B$
shows *continuous_map euclidean* (*top_of_set* (*span* B)) ($\lambda x. \sum_{i \in I}. x \ i \ *_R \ b \ i$)
proof (*rule continuous_map_euclidean_top_of_set*)
show $(\lambda x. \sum_{i \in I}. x \ i \ *_R \ b \ i) - 'span \ B = UNIV$
by *auto* (*meson biB lessThan_iff span_base span_scale span_sum*)
show *continuous_on* *UNIV* ($\lambda x. \sum_{i \in I}. x \ i \ *_R \ b \ i$)
by (*intro continuous_intros*) *auto*
qed

Topological countability for product spaces

The next two lemmas are useful to prove first or second countability of product spaces, but they have more to do with countability and could be put in the corresponding theory.

lemma *countable_nat_product_event_const*:
fixes $F::'a \text{ set}$ **and** $a::'a$
assumes $a \in F$ *countable* F
shows *countable* $\{x::(\text{nat} \Rightarrow 'a). (\forall i. x \ i \in F) \wedge \text{finite } \{i. x \ i \neq a\}\}$
proof –

```

have *: {x::(nat ⇒ 'a). (∀ i. x i ∈ F) ∧ finite {i. x i ≠ a}}
      ⊆ (⋃ N. {x. (∀ i. x i ∈ F) ∧ (∀ i ≥ N. x i = a)})
  using infinite_nat_iff_unbounded_le by fastforce
have countable {x. (∀ i. x i ∈ F) ∧ (∀ i ≥ N. x i = a)} for N::nat
proof (induction N)
  case 0
  have {x. (∀ i. x i ∈ F) ∧ (∀ i ≥ (0::nat). x i = a)} = {(λ i. a)}
    using ⟨a ∈ F⟩ by auto
  then show ?case by auto
next
  case (Suc N)
  define f::((nat ⇒ 'a) × 'a) ⇒ (nat ⇒ 'a)
    where f = (λ(x, b). x(N:=b))
  have {x. (∀ i. x i ∈ F) ∧ (∀ i ≥ Suc N. x i = a)} ⊆ f'({x. (∀ i. x i ∈ F) ∧
(∀ i ≥ N. x i = a)}) × F
  proof (auto)
    fix x assume H: ∀ i::nat. x i ∈ F ∀ i ≥ Suc N. x i = a
    have f (x(N:=a), x N) = x
      unfolding f_def by auto
    moreover have (x(N:=a), x N) ∈ {x. (∀ i. x i ∈ F) ∧ (∀ i ≥ N. x i = a)} ×
F
      using H ⟨a ∈ F⟩ by auto
    ultimately show x ∈ f'({x. (∀ i. x i ∈ F) ∧ (∀ i ≥ N. x i = a)}) × F
      by (metis (no_types, lifting) image_eqI)
  qed
  moreover have countable ({x. (∀ i. x i ∈ F) ∧ (∀ i ≥ N. x i = a)}) × F
    using Suc.IH assms(2) by auto
  ultimately show ?case
    by (meson countable_image countable_subset)
  qed
then show ?thesis using countable_subset[OF *] by auto
qed

```

lemma countable_product_event_const:

fixes F::('a::countable) ⇒ 'b set and b::'b

assumes $\bigwedge i. \text{countable } (F i)$

shows countable {f::('a ⇒ 'b). (∀ i. f i ∈ F i) ∧ (finite {i. f i ≠ b})}

proof –

define G where $G = (\bigcup i. F i) \cup \{b\}$

have countable G unfolding G_def using assms by auto

have b ∈ G unfolding G_def by auto

define pi where $pi = (\lambda(x::(nat ⇒ 'b)). (\lambda i::'a. x ((to_nat::('a ⇒ nat)) i)))$

have {f::('a ⇒ 'b). (∀ i. f i ∈ F i) ∧ (finite {i. f i ≠ b})}

⊆ pi' {g::(nat ⇒ 'b). (∀ j. g j ∈ G) ∧ (finite {j. g j ≠ b})}

proof (auto)

fix f assume H: ∀ i. f i ∈ F i finite {i. f i ≠ b}

define I where $I = \{i. f i ≠ b\}$

define g where $g = (\lambda j. \text{if } j \in \text{to_nat } I \text{ then } f \text{ (from_nat } j) \text{ else } b)$

have {j. g j ≠ b} ⊆ to_nat' I unfolding g_def by auto

```

then have finite {j. g j ≠ b}
  unfolding I_def using H(2) using finite_surj by blast
moreover have g j ∈ G for j
  unfolding g_def G_def using H by auto
ultimately have g ∈ {g::(nat ⇒ 'b). (∀j. g j ∈ G) ∧ (finite {j. g j ≠ b})}
  by auto
moreover have f = pi g
  unfolding pi_def g_def I_def using H by fastforce
ultimately show f ∈ pi {g. (∀j. g j ∈ G) ∧ finite {j. g j ≠ b}}
  by auto
qed
then show ?thesis
  using countable_nat_product_event_const[OF ‹b ∈ G› ‹countable G›]
  by (meson countable_image countable_subset)
qed

instance fun :: (countable, first_countable_topology) first_countable_topology
proof
  fix x::'a ⇒ 'b
  have ∃ A::('b ⇒ nat ⇒ 'b set). ∀x. (∀i. x ∈ A x i ∧ open (A x i)) ∧ (∀S. open
S ∧ x ∈ S ⟶ (∃i. A x i ⊆ S))
  apply (rule choice) using first_countable_basis by auto
  then obtain A::('b ⇒ nat ⇒ 'b set) where A: ⋀x i. x ∈ A x i
    ⋀x i. open (A x i)
    ⋀x S. open S ⟹ x ∈ S ⟹ (∃i. A x i ⊆ S)
  by metis

  B i is a countable basis of neighborhoods of xi.

  define B where B = (λi. (A (x i))'UNIV ∪ {UNIV})
  have countB: countable (B i) for i unfolding B_def by auto
  have open_B: ⋀X i. X ∈ B i ⟹ open X
  by (auto simp: B_def A)
  define K where K = {Pi_E UNIV X | X. (∀i. X i ∈ B i) ∧ finite {i. X i ≠
UNIV}}
  have Pi_E UNIV (λi. UNIV) ∈ K
  unfolding K_def B_def by auto
  then have K ≠ {} by auto
  have countable {X. (∀i. X i ∈ B i) ∧ finite {i. X i ≠ UNIV}}
  by (simp add: countB countable_product_event_const)
  moreover have K = (λX. Pi_E UNIV X)'{X. (∀i. X i ∈ B i) ∧ finite {i. X i
≠ UNIV}}
  unfolding K_def by auto
  ultimately have countable K by auto
  have I: x ∈ k if k ∈ K for k
  using that unfolding K_def B_def apply auto using A(1) by auto
  have II: open k if k ∈ K for k
  using that unfolding K_def by (blast intro: open_B open_PiE)
  have Inc: ∃k∈K. k ⊆ U if open U ∧ x ∈ U for U
  proof -

```

```

have openin (product_topology (λi. euclidean) UNIV) U x ∈ U
  using ⟨open U ∧ x ∈ U⟩ unfolding open_fun_def by auto
with product_topology_open_contains_basis[OF this]
have ∃X. x ∈ (ΠE i ∈ UNIV. X i) ∧ (∀i. open (X i)) ∧ finite {i. X i ≠ UNIV}
∧ (ΠE i ∈ UNIV. X i) ⊆ U
  by simp
then obtain X where H: x ∈ (ΠE i ∈ UNIV. X i)
  ∧ i. open (X i)
  ∧ finite {i. X i ≠ UNIV}
  (ΠE i ∈ UNIV. X i) ⊆ U

  by auto
define I where I = {i. X i ≠ UNIV}
define Y where Y = (λi. if i ∈ I then (SOME y. y ∈ B i ∧ y ⊆ X i) else
UNIV)
have *: ∃y. y ∈ B i ∧ y ⊆ X i for i
  unfolding B_def using A(3)[OF H(2)] H(1) by (metis PiE_E UNIV_1
UnCI image_iff)
have **: Y i ∈ B i ∧ Y i ⊆ X i for i
proof (cases i ∈ I)
  case True
    then show ?thesis
      by (metis (mono_tags, lifting) * Nitpick.Eps_psimp Y_def)
  next
    case False
      then show ?thesis by (simp add: B_def I_def Y_def)
qed
have {i. Y i ≠ UNIV} ⊆ I
  unfolding Y_def by auto
with ** have (∀i. Y i ∈ B i) ∧ finite {i. Y i ≠ UNIV}
  using H(3) I_def finite_subset by blast
then have PiE UNIV Y ∈ K
  unfolding K_def by auto
have Y i ⊆ X i for i
  using ** by auto
then have PiE UNIV Y ⊆ U
  by (metis H(4) PiE_mono subset_trans)
then show ?thesis using ⟨PiE UNIV Y ∈ K⟩ by auto
qed
show ∃L. (∀(i::nat). x ∈ L i ∧ open (L i)) ∧ (∀U. open U ∧ x ∈ U → (∃i.
L i ⊆ U))
  using ⟨countable K⟩ I II Inc by (simp add: first_countableI)
qed

proposition product_topology_countable_basis:
shows ∃K::('a::countable ⇒ 'b::second_countable_topology) set set.
  topological_basis K ∧ countable K ∧
  (∀k ∈ K. ∃X. (k = PiE UNIV X) ∧ (∀i. open (X i)) ∧ finite {i. X i ≠
UNIV})
proof –

```



```

obtain B::'b set set where B: countable B  $\wedge$  topological_basis B
  using ex_countable_basis by auto
then have B  $\neq$  {} by (meson UNIV_I empty_iff open_UNIV topological_basisE)
define B2 where B2 = B  $\cup$  {UNIV}
have countable B2
  unfolding B2_def using B by auto
have open U if U  $\in$  B2 for U
  using that unfolding B2_def using B topological_basis_open by auto

define K where K = {Pi_E UNIV X | X. ( $\forall i::'a$ . X i  $\in$  B2)  $\wedge$  finite {i. X i  $\neq$  UNIV}}
have i:  $\forall k \in K$ .  $\exists X$ . (k = Pi_E UNIV X)  $\wedge$  ( $\forall i$ . open (X i))  $\wedge$  finite {i. X i  $\neq$  UNIV}
  unfolding K_def using  $\langle \bigwedge U$ . U  $\in$  B2  $\implies$  open U  $\rangle$  by auto

have countable {X. ( $\forall i::'a$ . X i  $\in$  B2)  $\wedge$  finite {i. X i  $\neq$  UNIV}}
  using  $\langle$ countable B2 $\rangle$  by (intro countable_product_event_const) auto
moreover have K = ( $\lambda X$ . Pi_E UNIV X)  $\langle$ {X. ( $\forall i$ . X i  $\in$  B2)  $\wedge$  finite {i. X i  $\neq$  UNIV}} $\rangle$ 
  unfolding K_def by auto
ultimately have ii: countable K by auto

have iii: topological_basis K
proof (rule topological_basisI)
  fix U and x::'a $\implies$ 'b assume open U x  $\in$  U
  then have openin (product_topology ( $\lambda i$ . euclidean) UNIV) U
    unfolding open_fun_def by auto
  with product_topology_open_contains_basis[OF this  $\langle$ x  $\in$  U $\rangle$ ]
  obtain X where H: x  $\in$  ( $\Pi_E i \in UNIV$ . X i)
     $\bigwedge i$ . open (X i)
    finite {i. X i  $\neq$  UNIV}
    ( $\Pi_E i \in UNIV$ . X i)  $\subseteq$  U

    by auto
  then have x i  $\in$  X i for i by auto
  define I where I = {i. X i  $\neq$  UNIV}
  define Y where Y = ( $\lambda i$ . if i  $\in$  I then (SOME y. y  $\in$  B2  $\wedge$  y  $\subseteq$  X i  $\wedge$  x i  $\in$  y) else UNIV)
  have *:  $\exists y$ . y  $\in$  B2  $\wedge$  y  $\subseteq$  X i  $\wedge$  x i  $\in$  y for i
    unfolding B2_def using B  $\langle$ open (X i) $\rangle$   $\langle$ x i  $\in$  X i $\rangle$  by (meson UnCI topological_basisE)
  have **: Y i  $\in$  B2  $\wedge$  Y i  $\subseteq$  X i  $\wedge$  x i  $\in$  Y i for i
    using someI_ex[OF *] by (simp add: B2_def I_def Y_def)
  have {i. Y i  $\neq$  UNIV}  $\subseteq$  I
    unfolding Y_def by auto
  then have ( $\forall i$ . Y i  $\in$  B2)  $\wedge$  finite {i. Y i  $\neq$  UNIV}
    using ** H(3) I_def finite_subset by blast
  then have Pi_E UNIV Y  $\in$  K
    unfolding K_def by auto
  then show  $\exists V \in K$ . x  $\in$  V  $\wedge$  V  $\subseteq$  U

```

```

    by (meson ** H(4) PiE_I PiE_mono UNIV_I order.trans)
  next
  fix U assume U ∈ K
  show open U
    using ⟨U ∈ K⟩ unfolding open_fun_def K_def by clarify (metis ⟨U ∈ K⟩
i open_PiE open_fun_def)
  qed

  show ?thesis using i ii iii by auto
qed

```

```

instance fun :: (countable, second_countable_topology) second_countable_topology
proof
  show ∃ B::('a ⇒ 'b) set set. countable B ∧ open = generate_topology B
    using product_topology_countable_basis topological_basis_imp_subbasis
    by auto
  qed

```

2.5.2 The Alexander subbase theorem

```

theorem Alexander_subbase:
  assumes X: topology (arbitrary union_of (finite intersection_of (λx. x ∈ B)
relative_to ∪ B)) = X
  and fin: ∧ C. [C ⊆ B; ∪ C = topspace X] ⇒ ∃ C'. finite C' ∧ C' ⊆ C ∧
∪ C' = topspace X
  shows compact_space X
proof -
  have UB: ∪ B = topspace X
    by (simp flip: X)
  have False if U: ∀ U ∈ U. openin X U and sub: topspace X ⊆ ∪ U
    and neg: ∧ F. [F ⊆ U; finite F] ⇒ ¬ topspace X ⊆ ∪ F for U
  proof -
    define A where A ≡ {C. (∀ U ∈ C. openin X U) ∧ topspace X ⊆ ∪ C ∧ (∀ F.
finite F → F ⊆ C → ∼(topspace X ⊆ ∪ F))}
    have 1: A ≠ {}
      unfolding A_def using sub U neg by force
    have 2: ∪ C ∈ A if C ≠ {} and C: subset.chain A C for C
      unfolding A_def
    proof (intro CollectI conjI ballI allI impI notI)
      show openin X U if U: U ∈ ∪ C for U
        using U C unfolding A_def subset_chain_def by force
      have C ⊆ A
        using subset_chain_def C by blast
      with that A_def show UUC: topspace X ⊆ ∪ (∪ C)
        by blast
      show False if finite F and F ⊆ ∪ C and topspace X ⊆ ∪ F for F
    proof -
      obtain B where B ∈ C F ⊆ B
        by (metis Sup_empty C ⟨F ⊆ ∪ C⟩ ⟨finite F⟩ UUC empty_subsetI

```

```

finite.emptyI finite_subset_Union_chain neg)
  then show False
    using  $\mathcal{A\_def}$   $\langle C \subseteq \mathcal{A} \rangle$   $\langle \text{finite } \mathcal{F} \rangle$   $\langle \text{topspace } X \subseteq \bigcup \mathcal{F} \rangle$  by blast
  qed
  qed
  obtain  $\mathcal{K}$  where  $\mathcal{K} \in \mathcal{A}$  and  $\bigwedge X. \llbracket X \in \mathcal{A}; \mathcal{K} \subseteq X \rrbracket \implies X = \mathcal{K}$ 
    using subset_Zorn_nonempty [OF 1 2] by metis
  then have *:  $\bigwedge \mathcal{W}. \llbracket \bigwedge W. W \in \mathcal{W} \implies \text{openin } X W; \text{topspace } X \subseteq \bigcup \mathcal{W}; \mathcal{K} \subseteq \mathcal{W};$ 
     $\bigwedge \mathcal{F}. \llbracket \text{finite } \mathcal{F}; \mathcal{F} \subseteq \mathcal{W}; \text{topspace } X \subseteq \bigcup \mathcal{F} \rrbracket \implies \text{False} \rrbracket$ 
     $\implies \mathcal{W} = \mathcal{K}$ 
    and ope:  $\forall U \in \mathcal{K}. \text{openin } X U$  and top:  $\text{topspace } X \subseteq \bigcup \mathcal{K}$ 
    and non:  $\bigwedge \mathcal{F}. \llbracket \text{finite } \mathcal{F}; \mathcal{F} \subseteq \mathcal{K}; \text{topspace } X \subseteq \bigcup \mathcal{F} \rrbracket \implies \text{False}$ 
    unfolding  $\mathcal{A\_def}$  by simp_all metis+
  then obtain  $x$  where  $x \in \text{topspace } X$   $x \notin \bigcup (\mathcal{B} \cap \mathcal{K})$ 
  proof -
    have  $\bigcup (\mathcal{B} \cap \mathcal{K}) \neq \bigcup \mathcal{B}$ 
      by (metis  $\langle \bigcup \mathcal{B} = \text{topspace } X \rangle$  fin_inf.bounded_iff_non_order_refl)
    then have  $\exists a. a \notin \bigcup (\mathcal{B} \cap \mathcal{K}) \wedge a \in \bigcup \mathcal{B}$ 
      by blast
    then show ?thesis
      using that by (metis UB)
  qed
  obtain  $C$  where  $C: \text{openin } X C$   $C \in \mathcal{K}$   $x \in C$ 
    using  $\langle x \in \text{topspace } X \rangle$  ope top by auto
  then have  $C \subseteq \text{topspace } X$ 
    by (metis openin_subset)
  then have (arbitrary union_of (finite intersection_of  $(\lambda x. x \in \mathcal{B})$  relative_to  $\bigcup \mathcal{B}$ ))  $C$ 
    using openin_subbase C unfolding X [symmetric] by blast
  moreover have  $C \neq \text{topspace } X$ 
    using  $\langle \mathcal{K} \in \mathcal{A} \rangle$   $\langle C \in \mathcal{K} \rangle$  unfolding  $\mathcal{A\_def}$  by blast
  ultimately obtain  $\mathcal{V}$   $W$  where  $W: (\text{finite intersection_of } (\lambda x. x \in \mathcal{B}) \text{ relative\_to } \text{topspace } X)$   $W$ 
    and  $x \in W$   $W \in \mathcal{V}$   $\bigcup \mathcal{V} \neq \text{topspace } X$   $C = \bigcup \mathcal{V}$ 
    using  $C$  by (auto simp: union_of_def UB)
  then have  $\bigcup \mathcal{V} \subseteq \text{topspace } X$ 
    by (metis  $\langle C \subseteq \text{topspace } X \rangle$ )
  then have  $\text{topspace } X \notin \mathcal{V}$ 
    using  $\langle \bigcup \mathcal{V} \neq \text{topspace } X \rangle$  by blast
  then obtain  $\mathcal{B}'$  where  $\mathcal{B}': \text{finite } \mathcal{B}'$   $\mathcal{B}' \subseteq \mathcal{B}$   $x \in \bigcap \mathcal{B}'$   $W = \text{topspace } X \cap \bigcap \mathcal{B}'$ 
    using  $W$   $\langle x \in W \rangle$  unfolding relative_to_def intersection_of_def by auto
  then have  $\bigcap \mathcal{B}' \subseteq \bigcup \mathcal{B}$ 
    using  $\langle W \in \mathcal{V} \rangle$   $\langle \bigcup \mathcal{V} \neq \text{topspace } X \rangle$   $\langle \bigcup \mathcal{V} \subseteq \text{topspace } X \rangle$  by blast
  then have  $\bigcap \mathcal{B}' \subseteq C$ 
    using UB  $\langle C = \bigcup \mathcal{V} \rangle$   $\langle W = \text{topspace } X \cap \bigcap \mathcal{B}' \rangle$   $\langle W \in \mathcal{V} \rangle$  by auto
  have  $\forall b \in \mathcal{B}'. \exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b C')$ 
  proof
    fix  $b$ 

```

```

assume  $b \in \mathcal{B}'$ 
have  $\text{insert } b \ \mathcal{K} = \mathcal{K}$  if  $\text{neg: } \neg (\exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq$ 
 $\bigcup (\text{insert } b \ C'))$ 
proof (rule *)
  show  $\text{openin } X \ W$  if  $W \in \text{insert } b \ \mathcal{K}$  for  $W$ 
    using that
    proof
      have  $b \in \mathcal{B}$ 
        using  $\langle b \in \mathcal{B}' \rangle \langle \mathcal{B}' \subseteq \mathcal{B} \rangle$  by blast
      then have  $\exists \mathcal{U}. \text{finite } \mathcal{U} \wedge \mathcal{U} \subseteq \mathcal{B} \wedge \bigcap \mathcal{U} = b$ 
        by (rule_tac  $x = \{b\}$  in exI) auto
      moreover have  $\bigcup \mathcal{B} \cap b = b$ 
        using  $\mathcal{B}'(2) \langle b \in \mathcal{B}' \rangle$  by auto
      ultimately show  $\text{openin } X \ W$  if  $W = b$ 
        using that  $\langle b \in \mathcal{B}' \rangle$ 
      apply (simp add: openin_subbase_flip: X)
      apply (auto simp: arbitrary_def intersection_of_def relative_to_def
intro!: union_of_inc)
    done
    show  $\text{openin } X \ W$  if  $W \in \mathcal{K}$ 
      by (simp add:  $\langle W \in \mathcal{K} \rangle$  ope)
    qed
  next
    show  $\text{topspace } X \subseteq \bigcup (\text{insert } b \ \mathcal{K})$ 
      using top by auto
  next
    show False if  $\text{finite } \mathcal{F}$  and  $\mathcal{F} \subseteq \text{insert } b \ \mathcal{K}$   $\text{topspace } X \subseteq \bigcup \mathcal{F}$  for  $\mathcal{F}$ 
    proof –
      have  $\text{insert } b \ (\mathcal{F} \cap \mathcal{K}) = \mathcal{F}$ 
        using non that by blast
      then show False
        by (metis Int_lower2 finite_insert neg that(1) that(3))
    qed
  qed auto
  then show  $\exists C'. \text{finite } C' \wedge C' \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq \bigcup (\text{insert } b \ C')$ 
    using  $\langle b \in \mathcal{B}' \rangle \langle x \notin \bigcup (\mathcal{B} \cap \mathcal{K}) \rangle \mathcal{B}'$ 
    by (metis IntI InterE Union_iff subsetD insertI1)
  qed
  then obtain  $F$  where  $F: \forall b \in \mathcal{B}'. \text{finite } (F \ b) \wedge F \ b \subseteq \mathcal{K} \wedge \text{topspace } X \subseteq$ 
 $\bigcup (\text{insert } b \ (F \ b))$ 
    by metis
  let  $?D = \text{insert } C \ (\bigcup (F \ \mathcal{B}'))$ 
  show False
  proof (rule non)
    have  $\text{topspace } X \subseteq (\bigcap b \in \mathcal{B}'. \bigcup (\text{insert } b \ (F \ b)))$ 
      using  $F$  by (simp add: INT_greatest)
    also have  $\dots \subseteq \bigcup ?D$ 
      using  $\langle \bigcap \mathcal{B}' \subseteq C \rangle$  by force
    finally show  $\text{topspace } X \subseteq \bigcup ?D$  .

```

```

    show ? $\mathcal{D} \subseteq \mathcal{K}$ 
      using  $\langle C \in \mathcal{K} \rangle F$  by auto
    show finite ? $\mathcal{D}$ 
      using  $\langle \text{finite } \mathcal{B}' \rangle F$  by auto
  qed
qed
then show ?thesis
  by (force simp: compact_space_def compactin_def)
qed

```

corollary Alexander_subbase_alt:

```

  assumes  $U \subseteq \bigcup \mathcal{B}$ 
  and fin:  $\bigwedge C. [\![C \subseteq \mathcal{B}; U \subseteq \bigcup C]\!] \implies \exists C'. \text{finite } C' \wedge C' \subseteq C \wedge U \subseteq \bigcup C'$ 
  and X: topology
    (arbitrary union_of
     (finite intersection_of  $(\lambda x. x \in \mathcal{B})$  relative_to U)) = X
  shows compact_space X
proof -
  have topology X = U
    using X topology_subbase by fastforce
  have eq:  $\bigcup (\text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } U)) = U$ 
    unfolding relative_to_def
    using  $\langle U \subseteq \bigcup \mathcal{B} \rangle$  by blast
  have *:  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{C} \wedge \bigcup \mathcal{F} = \text{topspace } X$ 
    if  $C \subseteq \text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } \text{topspace } X)$  and UC:  $\bigcup C = \text{topspace } X$ 
  for C
proof -
  have  $C \subseteq (\lambda U. \text{topspace } X \cap U) \text{ ` } \mathcal{B}$ 
    using that by (auto simp: relative_to_def)
  then obtain  $\mathcal{B}'$  where  $\mathcal{B}' \subseteq \mathcal{B}$  and  $\mathcal{B}': C = (\cap) (\text{topspace } X) \text{ ` } \mathcal{B}'$ 
    by (auto simp: subset_image_iff)
  moreover have  $U \subseteq \bigcup \mathcal{B}'$ 
    using  $\mathcal{B}' \langle \text{topspace } X = U \rangle UC$  by auto
  ultimately obtain  $C'$  where finite  $C' \ C' \subseteq \mathcal{B}' \ U \subseteq \bigcup C'$ 
    using fin [of  $\mathcal{B}'$ ]  $\langle \text{topspace } X = U \rangle \langle U \subseteq \bigcup \mathcal{B}' \rangle$  by blast
  then show ?thesis
    unfolding  $\mathcal{B}' \text{ ex\_finite\_subset\_image } \langle \text{topspace } X = U \rangle$  by auto
qed
show ?thesis
  apply (rule Alexander_subbase [where  $\mathcal{B} = \text{Collect } ((\lambda x. x \in \mathcal{B}) \text{ relative\_to } (\text{topspace } X))$ ])
  apply (simp flip: X)
  apply (metis finite_intersection_of_relative_to eq)
  apply (blast intro: *)
done
qed

```

proposition continuous_map_componentwise:

```

continuous_map X (product_topology Y I) f  $\longleftrightarrow$ 
  f ' (topspace X)  $\subseteq$  extensional I  $\wedge$  ( $\forall k \in I$ . continuous_map X (Y k) ( $\lambda x$ . f x
k))
  (is ?lhs  $\longleftrightarrow$  _  $\wedge$  ?rhs)
proof (cases  $\forall x \in$  topspace X. f x  $\in$  extensional I)
  case True
  then have f ' (topspace X)  $\subseteq$  extensional I
  by force
  moreover have ?rhs if L: ?lhs
  proof -
  have openin X {x  $\in$  topspace X. f x k  $\in$  U} if k  $\in$  I and openin (Y k) U for
k U
  proof -
  have openin (product_topology Y I) ({Y. Y k  $\in$  U}  $\cap$  ( $\prod_{E \ i \in I}$ . topspace (Y
i)))
  apply (simp add: openin_product_topology flip: arbitrary_union_of_relative_to)
  apply (simp add: relative_to_def)
  using that apply (blast intro: arbitrary_union_of_inc finite_intersection_of_inc)
  done
  with that have openin X {x  $\in$  topspace X. f x  $\in$  ({Y. Y k  $\in$  U}  $\cap$  ( $\prod_{E \ i \in I}$ .
topspace (Y i)))}
  using L unfolding continuous_map_def by blast
  moreover have {x  $\in$  topspace X. f x  $\in$  ({Y. Y k  $\in$  U}  $\cap$  ( $\prod_{E \ i \in I}$ . topspace
(Y i)))} = {x  $\in$  topspace X. f x k  $\in$  U}
  using L by (auto simp: continuous_map_def)
  ultimately show ?thesis
  by metis
  qed
  with that
  show ?thesis
  by (auto simp: continuous_map_def)
  qed
  moreover have ?lhs if ?rhs
  proof -
  have 1:  $\bigwedge x$ . x  $\in$  topspace X  $\implies$  f x  $\in$  ( $\prod_{E \ i \in I}$ . topspace (Y i))
  using that True by (auto simp: continuous_map_def PiE_iff)
  have 2: {x  $\in$  S.  $\exists T \in \mathcal{T}$ . f x  $\in$  T} = ( $\bigcup T \in \mathcal{T}$ . {x  $\in$  S. f x  $\in$  T}) for S  $\mathcal{T}$ 
  by blast
  have 3: {x  $\in$  S.  $\forall U \in \mathcal{U}$ . f x  $\in$  U} = ( $\bigcap$  (insert S (( $\lambda U$ . {x  $\in$  S. f x  $\in$  U}) '
 $\mathcal{U}$ ))) for S  $\mathcal{U}$ 
  by blast
  show ?thesis
  unfolding continuous_map_def openin_product_topology arbitrary_def
  proof (clarsimp simp: all_union_of 1 2)
  fix  $\mathcal{T}$ 
  assume  $\mathcal{T}$ :  $\mathcal{T} \subseteq$  Collect (finite_intersection_of ( $\lambda F$ .  $\exists i \ U$ . F = {f. f i  $\in$  U}
 $\wedge i \in I \wedge$  openin (Y i) U)
  relative_to ( $\prod_{E \ i \in I}$ . topspace (Y i)))
  show openin X ( $\bigcup T \in \mathcal{T}$ . {x  $\in$  topspace X. f x  $\in$  T})

```

```

proof (rule openin_Union; clarify)
  fix  $S T$ 
  assume  $T \in \mathcal{T}$ 
  obtain  $\mathcal{U}$  where  $T = (\prod_{E \ i \in I. \text{topspace } (Y \ i)}) \cap \bigcap \mathcal{U}$  and finite  $\mathcal{U}$ 
     $\mathcal{U} \subseteq \{\{f. f \ i \in U\} \mid i \ U. \ i \in I \wedge \text{openin } (Y \ i) \ U\}$ 
    using subsetD [OF  $\mathcal{T} \ \langle T \in \mathcal{T} \rangle$ ] by (auto simp: intersection_of_def
relative_to_def)
  with that show openin  $X \ \{x \in \text{topspace } X. f \ x \in T\}$ 
    apply (simp add: continuous_map_def 1 cong: conj_cong)
    unfolding 3
    apply (rule openin_Inter; auto)
    done
  qed
qed
qed
ultimately show ?thesis
  by metis
qed (auto simp: continuous_map_def PiE_def)

```

```

lemma continuous_map_componentwise_UNIV:
  continuous_map  $X$  (product_topology  $Y$  UNIV)  $f \longleftrightarrow (\forall k. \text{continuous\_map } X$ 
 $(Y \ k) (\lambda x. f \ x \ k))$ 
  by (simp add: continuous_map_componentwise)

```

```

lemma continuous_map_product_projection [continuous_intros]:
   $k \in I \implies \text{continuous\_map } (\text{product\_topology } X \ I) (X \ k) (\lambda x. x \ k)$ 
  using continuous_map_componentwise [of product_topology  $X \ I \ X \ I \ \text{id}$ ] by simp

```

```

declare continuous_map_from_subtopology [OF continuous_map_product_projection,
continuous_intros]

```

```

proposition open_map_product_projection:
  assumes  $i \in I$ 
  shows open_map (product_topology  $Y \ I) (Y \ i) (\lambda f. f \ i)$ 
  unfolding openin_product_topology all_union_of_arbitrary_def open_map_def
image_Union
proof clarify
  fix  $\mathcal{V}$ 
  assume  $\mathcal{V}: \mathcal{V} \subseteq \text{Collect}$ 
    (finite intersection_of
     $(\lambda F. \exists i \ U. F = \{f. f \ i \in U\} \wedge i \in I \wedge \text{openin } (Y \ i) \ U)$  relative_to
     $\text{topspace } (\text{product\_topology } Y \ I))$ 
  show openin  $(Y \ i) (\bigcup_{x \in \mathcal{V}. (\lambda f. f \ i) \ 'x}$ )
  proof (rule openin_Union, clarify)
    fix  $S \ V$ 
    assume  $V \in \mathcal{V}$ 
    obtain  $\mathcal{F}$  where finite  $\mathcal{F}$ 
      and  $V: V = (\prod_{E \ i \in I. \text{topspace } (Y \ i)}) \cap \bigcap \mathcal{F}$ 

```

```

and  $\mathcal{F}: \mathcal{F} \subseteq \{\{f. f i \in U\} \mid i U. i \in I \wedge \text{openin } (Y i) U\}$ 
using subsetD [OF  $\mathcal{V} \langle V \in \mathcal{V} \rangle$ ]
by (auto simp: intersection_of_def relative_to_def)
show openin (Y i) (( $\lambda f. f i$ ) ‘ V)
proof (subst openin_subopen; clarify)
  fix x f
  assume f  $\in V$ 
  let ?T = {a  $\in \text{topspace}(Y i).$ 
    ( $\lambda j \in I. f j$ )(i:=a)  $\in (\prod_E i \in I. \text{topspace } (Y i)) \cap \bigcap \mathcal{F}$ }
  show  $\exists T. \text{openin } (Y i) T \wedge f i \in T \wedge T \subseteq (\lambda f. f i) ‘ V$ 
  proof (intro exI conjI)
    show openin (Y i) ?T
    proof (rule openin_continuous_map_preimage)
      have continuous_map (Y i) (Y k) ( $\lambda x. \text{if } k = i \text{ then } x \text{ else } f k$ ) if  $k \in I$ 
for k
      proof (cases k=i)
        case True
          then show ?thesis
            by (metis (mono_tags) continuous_map_id eq_id_iff)
        next
          case False
            then show ?thesis
              by simp (metis IntD1 PiE_iff V  $\langle f \in V \rangle$  that)
      qed
      then show continuous_map (Y i) (product_topology Y I)
        ( $\lambda x. (\lambda j \in I. f j)(i:=x)$ )
      by (auto simp: continuous_map_componentwise assms extensional_def
restrict_def)
      next
        have openin (product_topology Y I) ( $\prod_E i \in I. \text{topspace } (Y i)$ )
          by (metis openin_topspace topspace_product_topology)
        moreover have openin (product_topology Y I) ( $\bigcap B \in \mathcal{F}. (\prod_E i \in I. \text{topspace } (Y i)) \cap B$ )
          if  $\mathcal{F} \neq \{\}$ 
        proof –
          show ?thesis
          proof (rule openin_Inter)
            show  $\bigwedge X. X \in (\bigcap) (\prod_E i \in I. \text{topspace } (Y i)) ‘ \mathcal{F} \implies \text{openin } (product\_topology Y I) X$ 
            unfolding openin_product_topology relative_to_def
            apply (clarify intro!: arbitrary_union_of_inc)
            using subsetD [OF  $\mathcal{F}$ ]
            by (metis (mono_tags, lifting) finite_intersection_of_inc mem_Collect_eq
topspace_product_topology)
            qed (use  $\langle \text{finite } \mathcal{F} \rangle \langle \mathcal{F} \neq \{\} \rangle$  in auto)
          qed
        ultimately show openin (product_topology Y I) ( $(\prod_E i \in I. \text{topspace } (Y i)) \cap \bigcap \mathcal{F}$ )
        by (auto simp only: Int_Inter_eq split: if_split)

```



```

    qed
  next
  have eqf:  $(\lambda j \in I. f j)(i := f i) = f$ 
    using  $PiE\_arb$   $V \langle f \in V \rangle$  by force
  show  $f i \in ?T$ 
    using  $V$   $assms \langle f \in V \rangle$  by (auto simp:  $PiE\_iff$  eqf)
  next
  show  $?T \subseteq (\lambda f. f i) ' V$ 
    unfolding  $V$  by (auto simp: intro!:  $rev\_image\_eqI$ )
  qed
qed
qed
qed
qed

lemma retraction_map_product_projection:
  assumes  $i \in I$ 
  shows  $(retraction\_map (product\_topology X I) (X i) (\lambda x. x i) \longleftrightarrow$ 
     $((product\_topology X I) = trivial\_topology) \longrightarrow (X i) = trivial\_topology)$ 
    (is  $?lhs = ?rhs$ )
  proof
    assume  $?lhs$ 
    then show  $?rhs$ 
      using  $retraction\_imp\_surjective\_map$ 
      by ( $metis$   $image\_empty$   $subtopology\_eq\_discrete\_topology\_empty$ )
  next
    assume  $R: ?rhs$ 
    show  $?lhs$ 
      proof (cases  $(product\_topology X I) = trivial\_topology$ )
        case True
          then show  $?thesis$ 
            using  $R$  by (auto simp:  $retraction\_map\_def$   $retraction\_maps\_def$ )
        case False
          have *:  $\exists g. continuous\_map (X i) (product\_topology X I) g \wedge (\forall x \in topspace$ 
             $(X i). g x i = x)$ 
            if  $z: z \in (\prod_{E} i \in I. topspace (X i))$  for  $z$ 
            proof -
              have  $cm: continuous\_map (X i) (X j) (\lambda x. if j = i then x else z j)$  if  $j \in I$ 
                for  $j$ 
                using  $\langle j \in I \rangle z$  by (case_tac  $j = i$ ) auto
              show  $?thesis$ 
                using  $\langle i \in I \rangle$  that
                by ( $rule\_tac$   $x = \lambda x j. if j = i then x else z j$  in  $exI$ ) (auto simp:  $continuous\_map\_componentwise$   $PiE\_iff$   $extensional\_def$   $cm$ )
            qed
          with  $\langle i \in I \rangle$   $False$   $assms$  show  $?thesis$ 
            by (auto simp:  $retraction\_map\_def$   $retraction\_maps\_def$   $simp$   $flip: null\_topspace\_iff\_trivial$ )
        qed
      qed
  qed

```

2.5.3 Open Pi-sets in the product topology

proposition *openin_PiE_gen*:

$openin (product_topology X I) (PiE I S) \longleftrightarrow$
 $PiE I S = \{\} \vee$
 $finite \{i \in I. S i \neq topspace (X i)\} \wedge (\forall i \in I. openin (X i) (S i))$
(is ?lhs \longleftrightarrow _ \vee ?rhs)

proof (*cases* $PiE I S = \{\}$)

case *False*

moreover have *?lhs = ?rhs*

proof

assume *L: ?lhs*

moreover

obtain *z where z: z \in PiE I S*

using *False by blast*

ultimately obtain *U where fin: finite {i \in I. U i \neq topspace (X i)}*

and *PiE I U \neq {}*

and *sub: PiE I U \subseteq PiE I S*

by (*fastforce simp add: openin_product_topology_alt*)

then have **: $\bigwedge i. i \in I \implies U i \subseteq S i$*

by (*simp add: subset_PiE*)

show *?rhs*

proof (*intro conjI ballI*)

show *finite {i \in I. S i \neq topspace (X i)}*

apply (*rule finite_subset [OF _ fin], clarify*)

using ***

by (*metis False L openin_subset topspace_product_topology subset_PiE*

subset_antisym)

next

fix *i :: 'a*

assume *i \in I*

then show *openin (X i) (S i)*

using *open_map_product_projection [of i I X] L*

apply (*simp add: open_map_def*)

apply (*drule_tac x=PiE I S in spec*)

apply (*simp add: False image_projection_PiE split: if_split_asm*)

done

qed

next

assume *?rhs*

then show *?lhs*

unfolding *openin_product_topology*

by (*intro arbitrary_union_of_inc*) (*auto simp: product_topology_base_alt*)

qed

ultimately show *?thesis*

by *simp*

qed *simp*

corollary *openin_PiE*:

```

  finite I  $\implies$  openin (product_topology X I) (PiE I S)  $\longleftrightarrow$  PiE I S = {}  $\vee$  ( $\forall i$ 
 $\in I$ . openin (X i) (S i))
  by (simp add: openin_PiE_gen)

```

proposition compact_space_product_topology:

```

compact_space(product_topology X I)  $\longleftrightarrow$ 
(product_topology X I) = trivial_topology  $\vee$  ( $\forall i \in I$ . compact_space(X i))
(is ?lhs = ?rhs)

```

proof (cases (product_topology X I) = trivial_topology)

case False

then obtain z where z: z \in ($\prod_E i \in I$. topspace(X i))

by (auto simp flip: null_topspace_iff_trivial)

show ?thesis

proof

assume L: ?lhs

show ?rhs

proof (clarsimp simp add: False compact_space_def)

fix i

assume i $\in I$

with L have continuous_map (product_topology X I) (X i) (λf . f i)

by (simp add: continuous_map_product_projection)

moreover

have $\bigwedge x$. x \in topspace (X i) \implies x \in (λf . f i) ' ($\prod_E i \in I$. topspace (X i))

using $\langle i \in I \rangle$ z by (rule_tac x=z(i:=x) in image_eqI) auto

then have (λf . f i) ' ($\prod_E i \in I$. topspace (X i)) = topspace (X i)

using $\langle i \in I \rangle$ z by auto

ultimately show compactin (X i) (topspace (X i))

by (metis L compact_space_def image_compactin topspace_product_topology)

qed

next

assume R: ?rhs

show ?lhs

proof (cases I = {})

case True

with R show ?thesis

by (simp add: compact_space_def)

next

case False

then obtain i where i $\in I$

by blast

show ?thesis

using R

proof

assume com [rule_format]: $\forall i \in I$. compact_space (X i)

let ?C = { {f. f i $\in U$ } | i U. i $\in I$ \wedge openin (X i) U }

show compact_space (product_topology X I)

proof (rule Alexander_subbase_alt)

show topspace (product_topology X I) $\subseteq \bigcup ?C$

unfolding topspace_product_topology using $\langle i \in I \rangle$ by blast

```

next
  fix C
  assume Csub:  $C \subseteq ?\mathcal{C}$  and UC:  $\text{topspace}(\text{product\_topology } X I) \subseteq \bigcup C$ 
  define  $\mathcal{D}$  where  $\mathcal{D} \equiv \lambda i. \{U. \text{openin } (X i) U \wedge \{f. f i \in U\} \in C\}$ 
  show  $\exists C'. \text{finite } C' \wedge C' \subseteq C \wedge \text{topspace}(\text{product\_topology } X I) \subseteq \bigcup C'$ 
  proof (cases  $\exists i. i \in I \wedge \text{topspace} (X i) \subseteq \bigcup (\mathcal{D} i)$ )
    case True
      then obtain i where  $i \in I$ 
        and i:  $\text{topspace} (X i) \subseteq \bigcup (\mathcal{D} i)$ 
        unfolding  $\mathcal{D\_def}$  by blast
      then have *:  $\bigwedge \mathcal{U}. [\text{Ball } \mathcal{U} (\text{openin } (X i)); \text{topspace} (X i) \subseteq \bigcup \mathcal{U}] \implies$ 
         $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \text{topspace} (X i) \subseteq \bigcup \mathcal{F}$ 
        using com [OF  $\langle i \in I \rangle$ ] by (auto simp: compact_space_def compactin_def)
      have  $\text{topspace} (X i) \subseteq \bigcup (\mathcal{D} i)$ 
        using i by auto
      with * obtain  $\mathcal{F}$  where  $\text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq (\mathcal{D} i) \wedge \text{topspace} (X i) \subseteq \bigcup \mathcal{F}$ 
        unfolding  $\mathcal{D\_def}$  by fastforce
      with  $\langle i \in I \rangle$  show ?thesis
        unfolding  $\mathcal{D\_def}$ 
        by (rule_tac  $x = (\lambda U. \{x. x i \in U\})$  '  $\mathcal{F}$  in exI) auto
    next
      case False
      then have  $\forall i \in I. \exists y. y \in \text{topspace} (X i) \wedge y \notin \bigcup (\mathcal{D} i)$ 
        by force
      then obtain g where  $g: \bigwedge i. i \in I \implies g i \in \text{topspace} (X i) \wedge g i \notin \bigcup (\mathcal{D} i)$ 
        by metis
      then have  $(\lambda i. \text{if } i \in I \text{ then } g i \text{ else undefined}) \in \text{topspace}(\text{product\_topology } X I)$ 
        by (simp add: PiE_I)
      moreover have  $(\lambda i. \text{if } i \in I \text{ then } g i \text{ else undefined}) \notin \bigcup C$ 
        using Csub g unfolding  $\mathcal{D\_def}$  by force
      ultimately show ?thesis
        using UC by blast
    qed
  qed (simp add: product_topology)
  qed simp
  qed
  qed auto

corollary compactin_PiE:
  compactin (product_topology X I) (PiE I S)  $\longleftrightarrow$ 
  PiE I S =  $\{\} \vee (\forall i \in I. \text{compactin } (X i) (S i))$ 
  by (fastforce simp add: compactin_subspace subtopology_product_topology compact_space_product_topology
  subset_PiE product_topology_trivial_iff subtopology_trivial_iff)

```

```

lemma in_product_topology_closure_of:
   $z \in (\text{product\_topology } X \ I) \ \text{closure\_of } S$ 
     $\implies i \in I \implies z \ i \in ((X \ i) \ \text{closure\_of } ((\lambda x. x \ i) \ ` \ S))$ 
using continuous_map_product_projection
by (force simp: continuous_map_eq_image_closure_subset_image_subset_iff)

lemma homeomorphic_space_singleton_product:
   $\text{product\_topology } X \ \{k\} \ \text{homeomorphic\_space } (X \ k)$ 
unfolding homeomorphic_space
apply (rule_tac x=\lambda x. x \ k in exI)
apply (rule bijective_open_imp_homeomorphic_map)
  apply (simp_all add: continuous_map_product_projection open_map_product_projection)
unfolding PiE_over_singleton_iff
  apply (auto simp: image_iff inj_on_def)
done

```

2.5.4 Relationship with connected spaces, paths, etc.

```

proposition connected_space_product_topology:
   $\text{connected\_space}(\text{product\_topology } X \ I) \longleftrightarrow$ 
   $(\exists i \in I. X \ i = \text{trivial\_topology}) \vee (\forall i \in I. \text{connected\_space}(X \ i))$ 
(is ?lhs  $\longleftrightarrow$  ?eq  $\vee$  ?rhs)
proof (cases ?eq)
case False
moreover have ?lhs = ?rhs
proof
  assume ?lhs
  moreover
  have connectedin(X i) (topspace(X i))
  if i \in I and ci: connectedin(product_topology X I) (topspace(product_topology X I)) for i
  proof –
  have cm: continuous_map (product_topology X I) (X i) (\lambda f. f i)
  by (simp add: \langle i \in I \rangle continuous_map_product_projection)
  show ?thesis
  using connectedin_continuous_map_image [OF cm ci] \langle i \in I \rangle
  by (simp add: False image_projection_PiE PiE_eq_empty_iff)
  qed
  ultimately show ?rhs
  by (meson connectedin_topspace)
next
assume cs [rule_format]: ?rhs
have False
  if disj: U \cap V = \{\} and subUV: (\Pi_E i \in I. topspace (X i)) \subseteq U \cup V
  and U: openin (product_topology X I) U
  and V: openin (product_topology X I) V
  and U \neq \{\} V \neq \{\}
  for U V
proof –

```

```

obtain  $f$  where  $f \in U$ 
  using  $\langle U \neq \{\} \rangle$  by blast
then have  $f: f \in (\prod_{E \ i \in I}. \text{topspace } (X \ i))$ 
  using  $U \ \text{openin\_subset}$  by fastforce
have  $U \subseteq \text{topspace}(\text{product\_topology } X \ I) \ V \subseteq \text{topspace}(\text{product\_topology } X$ 
 $I)$ 
  using  $U \ V \ \text{openin\_subset}$  by blast+
moreover have  $(\prod_{E \ i \in I}. \text{topspace } (X \ i)) \subseteq U$ 
proof –
  obtain  $C$  where (finite intersection_of  $(\lambda F. \exists i \ U. F = \{x. x \ i \in U\} \wedge i$ 
 $\in I \wedge \text{openin } (X \ i) \ U)$  relative_to
 $(\prod_{E \ i \in I}. \text{topspace } (X \ i)))$   $C \ C \subseteq U \ f \in C$ 
  using  $U \ \langle f \in U \rangle$  unfolding openin_product_topology_union_of_def by
auto
then obtain  $\mathcal{T}$  where finite  $\mathcal{T}$ 
and  $t: \bigwedge C. C \in \mathcal{T} \implies \exists i \ u. (i \in I \wedge \text{openin } (X \ i) \ u) \wedge C = \{x. x \ i \in u\}$ 
and  $\text{sub}U: \text{topspace } (\text{product\_topology } X \ I) \cap \bigcap \mathcal{T} \subseteq U$ 
and  $\text{ftop}: f \in \text{topspace } (\text{product\_topology } X \ I)$ 
and  $\text{fint}: f \in \bigcap \mathcal{T}$ 
by (fastforce simp: relative_to_def intersection_of_def subset_iff)
let  $?L = \bigcup C \in \mathcal{T}. \{i. (\lambda x. x \ i) \text{ ' } C \subset \text{topspace } (X \ i)\}$ 
obtain  $L$  where finite  $L$ 
and  $L: \bigwedge i \ U. \llbracket i \in I; \text{openin } (X \ i) \ U; U \subset \text{topspace}(X \ i); \{x. x \ i \in U\} \in$ 
 $\mathcal{T} \rrbracket \implies i \in L$ 
proof
  show finite  $?L$ 
proof (rule finite_Union)
    fix  $M$ 
    assume  $M \in (\lambda C. \{i. (\lambda x. x \ i) \text{ ' } C \subset \text{topspace } (X \ i)\}) \text{ ' } \mathcal{T}$ 
    then obtain  $C$  where  $C \in \mathcal{T}$  and  $C: M = \{i. (\lambda x. x \ i) \text{ ' } C \subset \text{topspace}$ 
 $(X \ i)\}$ 
      by blast
    then obtain  $j \ V$  where  $j \in I$  and  $\text{ope}: \text{openin } (X \ j) \ V$  and  $\text{Ceq}: C =$ 
 $\{x. x \ j \in V\}$ 
      using  $t$  by meson
    then have  $f \ j \in V$ 
      using  $\langle C \in \mathcal{T} \rangle$   $\text{fint}$  by force
    then have  $(\lambda x. x \ k) \text{ ' } \{x. x \ j \in V\} = \text{UNIV}$  if  $k \neq j$  for  $k$ 
      using that
    apply (clarsimp simp add: set_eq_iff)
    apply (rule_tac x=f(k:=x) in image_eqI, auto)
    done
    then have  $\{i. (\lambda x. x \ i) \text{ ' } C \subset \text{topspace } (X \ i)\} \subseteq \{j\}$ 
      using  $\text{Ceq}$  by auto
    then show finite  $M$ 
      using  $C$  finite_subset by fastforce
  qed (use  $\langle \text{finite } \mathcal{T} \rangle$  in blast)
next
  fix  $i \ U$ 

```

```

    assume  $i \in I$  and  $ope: \text{openin } (X \ i) \ U$  and  $psub: U \subset \text{topspace } (X \ i)$ 
and  $int: \{x. x \ i \in U\} \in \mathcal{T}$ 
    then show  $i \in ?L$ 
      by (rule_tac  $a = \{x. x \ i \in U\}$  in  $UN\_I$ ) (force+)
qed
show ?thesis
proof
  fix  $h$ 
  assume  $h: h \in (\prod_{E \ i \in I. \text{topspace } (X \ i)})$ 
  define  $g$  where  $g \equiv \lambda i. \text{if } i \in L \text{ then } f \ i \ \text{else } h \ i$ 
  have  $gin: g \in (\prod_{E \ i \in I. \text{topspace } (X \ i)})$ 
    unfolding  $g\_def$  using  $f \ h$  by auto
  moreover have  $g \in X$  if  $X \in \mathcal{T}$  for  $X$ 
    using  $fin \ \text{openin\_subset } t \ [OF \ that] \ L \ g\_def \ h$  that by fastforce
  ultimately have  $g \in U$ 
    using  $subU$  by auto
  have  $h \in U$  if finite  $M \ h \in \text{PiE } I \ (\text{topspace} \circ X) \ \{i \in I. h \ i \neq g \ i\} \subseteq M$ 
for  $M \ h$ 
    using that
  proof (induction arbitrary:  $h$ )
    case empty
      then show ?case
        using  $\text{PiE\_ext } \langle g \in U \rangle \ gin$  by force
    next
      case (insert  $i \ M$ )
        define  $f$  where  $f \equiv h(i := g \ i)$ 
        have  $fin: f \in \text{PiE } I \ (\text{topspace} \circ X)$ 
          unfolding  $f\_def$  using  $gin \ insert.premis(1)$  by auto
        have  $subM: \{j \in I. f \ j \neq g \ j\} \subseteq M$ 
          unfolding  $f\_def$  using  $insert.premis(2)$  by auto
        have  $f \in U$ 
          using  $insert.IH \ [OF \ fin \ subM]$  .
        show ?case
          proof (cases  $h \in V$ )
            case True
              show ?thesis
            proof (cases  $i \in I$ )
              case True
                let  $?U = \{x \in \text{topspace}(X \ i). h(i := x) \in U\}$ 
                let  $?V = \{x \in \text{topspace}(X \ i). h(i := x) \in V\}$ 
                have False
                  proof (rule connected_spaceD [OF cs [OF  $\langle i \in I \rangle$ ]])
                    have  $\bigwedge k. k \in I \implies \text{continuous\_map } (X \ i) \ (X \ k) \ (\lambda x. \text{if } k = i \ \text{then } x \ \text{else } h \ k)$ 
                      using  $\text{continuous\_map\_eq\_topcontinuous\_at } insert.premis(1)$ 
 $\text{topcontinuous\_at\_def}$  by fastforce
                    then have  $cm: \text{continuous\_map } (X \ i) \ (\text{product\_topology } X \ I) \ (\lambda x. h(i := x))$ 
                      using  $\langle i \in I \rangle \ insert.premis(1)$ 

```

```

    by (auto simp: continuous_map_componentwise_extensional_def)
  show openin (X i) ?U
    by (rule openin_continuous_map_preimage [OF cm U])
  show openin (X i) ?V
    by (rule openin_continuous_map_preimage [OF cm V])
  show topspace (X i)  $\subseteq$  ?U  $\cup$  ?V
  proof clarsimp
    fix x
    assume x  $\in$  topspace (X i) and h(i:=x)  $\notin$  V
    with True subUV  $\langle h \in PiE I (topspace \circ X) \rangle$ 
    show h(i:=x)  $\in$  U
      by (force dest: subsetD [where c=h(i:=x)])
    qed
  show ?U  $\cap$  ?V = {}
    using disj by blast
  show ?U  $\neq$  {}
    using True  $\langle f \in U \rangle$  f_def gin by auto
  show ?V  $\neq$  {}
    using True  $\langle h \in V \rangle$  V openin_subset by fastforce
  qed
  then show ?thesis ..
next
case False
show ?thesis
  using insert.premis(1) by (metis False gin PiE_E  $\langle f \in U \rangle$  f_def
fun_upd_triv)
  qed
next
case False
then show ?thesis
  using subUV insert.premis(1) by auto
  qed
qed
then show h  $\in$  U
  unfolding g_def using PiE_iff  $\langle finite L \rangle$  h by fastforce
  qed
qed
ultimately show ?thesis
  using disj inf_absorb2  $\langle V \neq \{ \} \rangle$  by fastforce
  qed
then show ?lhs
  unfolding connected_space_def
  by auto
  qed
ultimately show ?thesis
  by simp
qed (metis connected_space_trivial_topology_product_topology_trivial_iff)

```


lemma *connectedin_PiE*:

connectedin (product_topology X I) (PiE I S) \longleftrightarrow
 PiE I S = {} \vee ($\forall i \in I$. *connectedin* (X i) (S i))

by (auto simp: *connectedin_def* *subtopology_product_topology* *connected_space_product_topology* *subset_PiE*)

PiE_eq_empty_iff *subtopology_trivial_iff*)

lemma *path_connected_space_product_topology*:

path_connected_space(product_topology X I) \longleftrightarrow
topspace(product_topology X I) = {} \vee ($\forall i \in I$. *path_connected_space*(X i))

(is ?lhs \longleftrightarrow ?eq \vee ?rhs)

proof (cases ?eq)

case False

moreover have ?lhs = ?rhs

proof

assume L: ?lhs

show ?rhs

proof (clarsimp simp flip: *path_connectedin_tospace*)

fix i :: 'a

assume i \in I

have cm: *continuous_map* (product_topology X I) (X i) (λf . f i)

by (simp add: $\langle i \in I \rangle$ *continuous_map_product_projection*)

show *path_connectedin* (X i) (*topspace* (X i))

using *path_connectedin_continuous_map_image* [OF cm L [unfolded
path_connectedin_tospace [symmetric]]]

by (metis $\langle i \in I \rangle$ False *retraction_imp_surjective_map* *retraction_map_product_projection* *topspace_discrete_topology*)

qed

next

assume R [rule_format]: ?rhs

show ?lhs

unfolding *path_connected_space_def* *topspace_product_topology*

proof clarify

fix x y

assume x: x \in ($\prod_{E \ i \in I}$. *topspace* (X i)) and y: y \in ($\prod_{E \ i \in I}$. *topspace* (X i))

have $\forall i$. $\exists g$. i \in I \longrightarrow *pathin* (X i) g \wedge g 0 = x i \wedge g 1 = y i

using *PiE_mem* R *path_connected_space_def* x y **by** force

then obtain g **where** g: $\bigwedge i$. i \in I \implies *pathin* (X i) (g i) \wedge g i 0 = x i \wedge g i 1 = y i

by metis

with x y **show** $\exists g$. *pathin* (product_topology X I) g \wedge g 0 = x \wedge g 1 = y

apply (rule_tac x= λa . $\lambda i \in I$. g i a **in** exI)

apply (force simp: *pathin_def* *continuous_map_componentwise*)

done

qed

qed

ultimately show ?thesis

by simp

next

qed (*force simp: path_connected_space_topspace_empty_iff: null_topspace_iff_trivial*)

lemma *path_connectedin_PiE*:

path_connectedin (*product_topology X I*) (*PiE I S*) \longleftrightarrow

PiE I S = $\{\}$ \vee ($\forall i \in I. \text{path_connectedin } (X\ i) (S\ i)$)

by (*fastforce simp add: path_connectedin_def subtopology_product_topology path_connected_space_product_topology PiE PiE_eq_empty_iff topspace_subtopology_subset*)

2.5.5 Projections from a function topology to a component

lemma *quotient_map_product_projection*:

assumes $i \in I$

shows *quotient_map*(*product_topology X I*) (*X i*) ($\lambda x. x\ i$) \longleftrightarrow

$((\text{product_topology } X\ I) = \text{trivial_topology} \longrightarrow (X\ i) = \text{trivial_topology})$

by (*metis (no_types) assms image_is_empty null_topspace_iff_trivial quotient_imp_surjective_map*)

retraction_imp_quotient_map retraction_map_product_projection)

lemma *product_topology_homeomorphic_component*:

assumes $i \in I \wedge j. [j \in I; j \neq i] \implies \exists a. \text{topspace}(X\ j) = \{a\}$

shows *product_topology X I homeomorphic_space* (*X i*)

proof –

have *quotient_map* (*product_topology X I*) (*X i*) ($\lambda x. x\ i$)

using *assms* **by** (*metis (full_types) discrete_topology_unique empty_not_insert*)

product_topology_trivial_iff quotient_map_product_projection)

moreover

have *inj_on* ($\lambda x. x\ i$) ($\Pi_E i \in I. \text{topspace}(X\ i)$)

using *assms* **by** (*auto simp: inj_on_def PiE_iff*) (*metis extensionalityI singletonD*)

ultimately show *?thesis*

unfolding *homeomorphic_space_def*

by (*rule_tac x = $\lambda x. x\ i$ in exI*) (*simp add: homeomorphic_map_def flip: homeomorphic_map_maps*)

qed

lemma *topological_property_of_product_component*:

assumes *major*: $P(\text{product_topology } X\ I)$

and *minor*: $\bigwedge z\ i. [z \in (\Pi_E i \in I. \text{topspace}(X\ i)); P(\text{product_topology } X\ I); i \in I]$

$\implies P(\text{subtopology } (\text{product_topology } X\ I) (\text{PiE } I (\lambda j. \text{if } j = i \text{ then } \text{topspace}(X\ i) \text{ else } \{z\ j\})))$

(**is** $\bigwedge z\ i. [z \in _; _ \implies P(?SX\ z\ i)$)

and *PQ*: $\bigwedge X\ X'. X \text{ homeomorphic_space } X' \implies (P\ X \longleftrightarrow Q\ X')$

shows ($\exists i \in I. (X\ i) = \text{trivial_topology}$) \vee ($\forall i \in I. Q(X\ i)$)

proof –

have $Q(X\ i)$ **if** $\forall i \in I. (X\ i) \neq \text{trivial_topology}$ **for** $i \in I$ **for** i

proof –

```

from that obtain  $f$  where  $f: f \in (\prod_{E \ i \in I}. \text{topspace } (X \ i))$ 
  by (meson null_topspace_iff_trivial PiE_eq_empty_iff_ex_in_conv)
have  $?SX \ f \ i$  homeomorphic_space  $X \ i$ 
  using  $f$  product_topology_homeomorphic_component [OF  $\langle i \in I \rangle$ , of  $\lambda j.$ 
subtopology  $(X \ j)$  (if  $j = i$  then topspace  $(X \ i)$  else  $\{f \ j\}$ )]
  by (force simp add: subtopology_product_topology)
then show  $?thesis$ 
  using minor [OF  $f$  major  $\langle i \in I \rangle$ ] PQ by auto
qed
then show  $?thesis$  by metis
qed

```

2.5.6 Limits

The original HOL Light proof was a mess, yuk

lemma *limitin_componentwise*:

```

limitin (product_topology  $X \ I$ )  $f \ l \ F \longleftrightarrow$ 
   $l \in \text{extensional } I \wedge$ 
  eventually  $(\lambda a. f \ a \in \text{topspace}(\text{product\_topology } X \ I)) \ F \wedge$ 
   $(\forall i \in I. \text{limitin } (X \ i) (\lambda c. f \ c \ i) (l \ i) \ F)$ 
  (is  $?L \longleftrightarrow \_ \wedge ?R1 \wedge ?R2$ )

```

proof (*cases* $l \in \text{extensional } I$)

case l : *True*

show $?thesis$

proof (*cases* $\forall i \in I. l \ i \in \text{topspace } (X \ i)$)

case *True*

have $?R1$ **if** $?L$

by (*metis limitin_subtopology subtopology_topspace that*)

moreover

have $?R2$ **if** $?L$

unfolding *limitin_def*

proof (*intro conjI strip*)

fix $i \ U$

assume $i \in I$ **and** U : *openin* $(X \ i) \ U \wedge l \ i \in U$

then have *openin* (*product_topology* $X \ I$) $(\{y. y \ i \in U\} \cap \text{topspace } (\text{product_topology } X \ I))$

unfolding *openin_product_topology arbitrary_union_of_relative_to* [*symmetric*]

apply (*simp add: relative_to_def topspace_product_topology_alt*)

by (*smt* (*verit, del_insts*) *Collect_cong arbitrary_union_of_inc finite_intersection_of_inc inf_commute*)

moreover have $l \in \{y. y \ i \in U\} \cap \text{topspace } (\text{product_topology } X \ I)$

using $U \ \text{True} \ l$ **by** (*auto simp: extensional_def*)

ultimately have *eventually* $(\lambda x. f \ x \in \{y. y \ i \in U\} \cap \text{topspace } (\text{product_topology } X \ I)) \ F$

by (*metis limitin_def that*)

then show $\forall_F \ x \ \text{in } F. f \ x \ i \in U$

by (*simp add: eventually_conj_iff*)

qed (*use True in auto*)

moreover

```

have ?L if R1: ?R1 and R2: ?R2
  unfolding limitin_def openin_product_topology all_union_of_imp_conjL
  arbitrary_def
  proof (intro conjI strip)
    show l: l ∈ topspace (product_topology X I)
      by (simp add: PiE_iff True l)
    fix V
    assume V ⊆ Collect (finite_intersection_of (λF. ∃ i U. F = {f. f i ∈ U} ∧
i ∈ I ∧ openin (X i) U))
      relative_to topspace (product_topology X I))
    and l ∈ ⋃ V
    then obtain W where finite W and WX: ∀ X ∈ W. l ∈ X
      and W: ∧ C. C ∈ W ⇒ C ∈ {{x. x i ∈ U} | i U. i ∈ I ∧ openin (X
i) U}
      and WV: topspace (product_topology X I) ∩ ⋂ W ∈ V
      by (fastforce simp: intersection_of_def relative_to_def subset_eq)
    have ∀ F x in F. f x ∈ topspace (product_topology X I) ∩ ⋂ W
    proof -
      have ∧ W. W ∈ {{x. x i ∈ U} | i U. i ∈ I ∧ openin (X i) U} ⇒ W ∈
W ⇒ ∀ F x in F. f x ∈ W
      using WX R2 by (auto simp: limitin_def)
    with W have ∀ F x in F. ∀ W ∈ W. f x ∈ W
      by (simp add: ⟨finite W⟩ eventually_ball_finite)
    with R1 show ?thesis
      by (simp add: eventually_conj_iff)
    qed
    then show ∀ F x in F. f x ∈ ⋃ V
      by (smt (verit, ccfv_threshold) WV UnionI eventually_mono)
    qed
  ultimately show ?thesis
    using l by blast
  next
  case False
  then show ?thesis
    by (metis PiE_iff limitin_def topspace_product_topology)
  qed
next
  case False
  then show ?thesis
    by (simp add: limitin_def PiE_iff)
  qed
end

```

2.6 The binary product topology

```

theory Product_Topology
  imports Function_Topology
begin

```

2.7 Product Topology

2.7.1 Definition

definition *prod_topology* :: 'a topology \Rightarrow 'b topology \Rightarrow ('a \times 'b) topology **where**
prod_topology X Y \equiv topology (arbitrary_union_of ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X \ S \wedge \text{openin } Y \ T\}$))

lemma *open_product_open*:

assumes *open* A

shows $\exists \mathcal{U}. \mathcal{U} \subseteq \{S \times T \mid S \text{ T. } \text{open } S \wedge \text{open } T\} \wedge \bigcup \mathcal{U} = A$

proof –

obtain *f g* **where** *: $\bigwedge u. u \in A \Longrightarrow \text{open } (f \ u) \wedge \text{open } (g \ u) \wedge u \in (f \ u) \times (g \ u) \wedge (f \ u) \times (g \ u) \subseteq A$

using *open_prod_def* [of A] **assms** **by** *metis*

let ? \mathcal{U} = ($\lambda u. f \ u \times g \ u$) ' A

show ?thesis

by (*rule_tac* x=? \mathcal{U} **in** *exI*) (*auto simp: dest: **)

qed

lemma *open_product_open_eq*: (arbitrary_union_of ($\lambda U. \exists S \ T. U = S \times T \wedge \text{open } S \wedge \text{open } T$)) = *open*

by (*force simp: union_of_def arbitrary_def intro: open_product_open open_Times*)

lemma *openin_prod_topology*:

openin (*prod_topology* X Y) = arbitrary_union_of ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X \ S \wedge \text{openin } Y \ T\}$)

unfolding *prod_topology_def*

proof (*rule topology_inverse'*)

show *istopology* (arbitrary_union_of ($\lambda U. U \in \{S \times T \mid S \text{ T. } \text{openin } X \ S \wedge \text{openin } Y \ T\}$))

apply (*rule istopology_base, simp*)

by (*metis openin_Int Times_Int_Times*)

qed

lemma *topspace_prod_topology* [*simp*]:

topspace (*prod_topology* X Y) = *topspace* X \times *topspace* Y

proof –

have *topspace*(*prod_topology* X Y) = \bigcup (*Collect* (*openin* (*prod_topology* X Y)))
(*is* _ = ?Z)

unfolding *topspace_def* ..

also have ... = *topspace* X \times *topspace* Y

proof

show ?Z \subseteq *topspace* X \times *topspace* Y

apply (*auto simp: openin_prod_topology union_of_def arbitrary_def*)

using *openin_subset* **by** *force+*

next

have *: $\exists A \ B. \text{topspace } X \times \text{topspace } Y = A \times B \wedge \text{openin } X \ A \wedge \text{openin } Y \ B$

B

by *blast*

```

show topspace X × topspace Y ⊆ ?Z
apply (rule Union_upper)
using * by (simp add: openin_prod_topology arbitrary_union_of_inc)
qed
finally show ?thesis .
qed

```

```

lemma prod_topology_trivial_iff [simp]:
  prod_topology X Y = trivial_topology ↔ X = trivial_topology ∨ Y = trivial_topology
by (metis (full_types) Sigma_empty1 null_topspace_iff_trivial subset_empty times_subset_iff topspace_prod_topology)

```

```

lemma subtopology_Times:
  shows subtopology (prod_topology X Y) (S × T) = prod_topology (subtopology X S) (subtopology Y T)
proof –
  have ((λU. ∃ S T. U = S × T ∧ openin X S ∧ openin Y T) relative_to S × T) =
    (λU. ∃ S' T'. U = S' × T' ∧ (openin X relative_to S) S' ∧ (openin Y relative_to T) T')
  by (auto simp: relative_to_def Times_Int_Times fun_eq_iff) metis
  then show ?thesis
  by (simp add: topology_eq openin_prod_topology arbitrary_union_of_relative_to flip: openin_relative_to)
qed

```

```

lemma prod_topology_subtopology:
  prod_topology (subtopology X S) Y = subtopology (prod_topology X Y) (S × topspace Y)
  prod_topology X (subtopology Y T) = subtopology (prod_topology X Y) (topspace X × T)
by (auto simp: subtopology_Times)

```

```

lemma prod_topology_discrete_topology:
  discrete_topology (S × T) = prod_topology (discrete_topology S) (discrete_topology T)
by (auto simp: discrete_topology_unique openin_prod_topology intro: arbitrary_union_of_inc)

```

```

lemma prod_topology_euclidean [simp]: prod_topology euclidean euclidean = euclidean
by (simp add: prod_topology_def open_product_open_eq)

```

```

lemma prod_topology_subtopology_eu [simp]:
  prod_topology (subtopology euclidean S) (subtopology euclidean T) = subtopology euclidean (S × T)
by (simp add: prod_topology_subtopology subtopology_subtopology Times_Int_Times)

```

```

lemma openin_prod_topology_alt:

```

$openin (prod_topology X Y) S \longleftrightarrow$
 $(\forall x y. (x,y) \in S \longrightarrow (\exists U V. openin X U \wedge openin Y V \wedge x \in U \wedge y \in V$
 $\wedge U \times V \subseteq S))$
apply (auto simp: openin_prod_topology arbitrary_union_of_alt, fastforce)
by (metis mem_Sigma_iff)

lemma open_map_fst: open_map (prod_topology X Y) X fst
unfolding open_map_def openin_prod_topology_alt
by (force simp: openin_subopen [of X fst ' _] intro: subset_fst_imageI)

lemma open_map_snd: open_map (prod_topology X Y) Y snd
unfolding open_map_def openin_prod_topology_alt
by (force simp: openin_subopen [of Y snd ' _] intro: subset_snd_imageI)

lemma openin_prod_Times_iff:
 $openin (prod_topology X Y) (S \times T) \longleftrightarrow S = \{\} \vee T = \{\} \vee openin X S \wedge$
 $openin Y T$
proof (cases $S = \{\} \vee T = \{\}$)
case False
then show ?thesis
apply (simp add: openin_prod_topology_alt openin_subopen [of X S] openin_subopen
[of Y T] times_subset_iff, safe)
apply (meson|force)+
done
qed force

lemma closure_of_Times:
 $(prod_topology X Y) closure_of (S \times T) = (X closure_of S) \times (Y closure_of$
 $T)$ (is ?lhs = ?rhs)
proof
show ?lhs \subseteq ?rhs
by (clarsimp simp: closure_of_def openin_prod_topology_alt) blast
show ?rhs \subseteq ?lhs
by (clarsimp simp: closure_of_def openin_prod_topology_alt) (meson SigmaI
subsetD)
qed

lemma closedin_prod_Times_iff:
 $closedin (prod_topology X Y) (S \times T) \longleftrightarrow S = \{\} \vee T = \{\} \vee closedin X S$
 $\wedge closedin Y T$
by (auto simp: closure_of_Times times_eq_iff simp flip: closure_of_eq)

lemma interior_of_Times: (prod_topology X Y) interior_of (S × T) = (X in-
terior_of S) × (Y interior_of T)
proof (rule interior_of_unique)
show (X interior_of S) × Y interior_of T \subseteq S × T
by (simp add: Sigma_mono interior_of_subset)
show openin (prod_topology X Y) ((X interior_of S) × Y interior_of T)

```

    by (simp add: openin_prod_Times_iff)
next
  show  $T' \subseteq (X \text{ interior\_of } S) \times Y \text{ interior\_of } T$  if  $T' \subseteq S \times T$  openin
(prod_topology X Y) T' for T'
  proof (clarsimp; intro conjI)
    fix a :: 'a and b :: 'b
    assume (a, b) ∈ T'
    with that obtain U V where UV: openin X U openin Y V a ∈ U b ∈ V U ×
V ⊆ T'
    by (metis openin_prod_topology_alt)
    then show a ∈ X interior_of S
    using interior_of_maximal_eq that(1) by fastforce
    show b ∈ Y interior_of T
    using UV interior_of_maximal_eq that(1)
    by (metis SigmaI mem_Sigma_iff subset_eq)
  qed
qed

```

Missing the opposite direction. Does it hold? A converse is proved for proper maps, a stronger condition

```

lemma closed_map_prod:
  assumes closed_map (prod_topology X Y) (prod_topology X' Y') ((λ(x,y). (f x,
g y)))
  shows (prod_topology X Y) = trivial_topology ∨ closed_map X X' f ∧ closed_map
Y Y' g
proof (cases (prod_topology X Y) = trivial_topology)
  case False
  then have ne: topspace X ≠ {} topspace Y ≠ {}
    by (auto simp flip: null_topospace_iff_trivial)
  have closed_map X X' f
    unfolding closed_map_def
  proof (intro strip)
    fix C
    assume closedin X C
    show closedin X' (f ` C)
    proof (cases C={})
      case False
      with assms have closedin (prod_topology X' Y') ((λ(x,y). (f x, g y))) ` (C ×
topospace Y)
        by (simp add: ⟨closedin X C⟩ closed_map_def closedin_prod_Times_iff)
      with False ne show ?thesis
        by (simp add: image_paired_Times closedin_Times closedin_prod_Times_iff)
    qed auto
  qed
moreover
  have closed_map Y Y' g
    unfolding closed_map_def
  proof (intro strip)
    fix C

```



```

assume closedin Y C
show closedin Y' (g ' C)
proof (cases C={})
  case False
    with assms have closedin (prod_topology X' Y') (( $\lambda(x,y). (f\ x, g\ y)$ ) '
(topspace X  $\times$  C))
    by (simp add:  $\langle$ closedin Y C $\rangle$  closed_map_def closedin_prod_Times_iff)
    with False ne show ?thesis
    by (simp add: image_paired_Times closedin_Times closedin_prod_Times_iff)
  qed auto
qed
ultimately show ?thesis
  by (auto simp: False)
qed auto

```

2.7.2 Continuity

lemma *continuous_map_pairwise*:

```

continuous_map Z (prod_topology X Y) f  $\longleftrightarrow$  continuous_map Z X (fst  $\circ$  f)
 $\wedge$  continuous_map Z Y (snd  $\circ$  f)
(is ?lhs = ?rhs)

```

proof –

```

let ?g = fst  $\circ$  f and ?h = snd  $\circ$  f
have f: f x = (?g x, ?h x) for x
  by auto
show ?thesis
proof (cases ?g  $\in$  topspace Z  $\rightarrow$  topspace X  $\wedge$  ?h  $\in$  topspace Z  $\rightarrow$  topspace Y)
  case True
    show ?thesis
    proof safe
      assume continuous_map Z (prod_topology X Y) f
      then have openin Z {x  $\in$  topspace Z. fst (f x)  $\in$  U} if openin X U for U
        unfolding continuous_map_def using True that
        apply clarify
        apply (drule_tac x=U  $\times$  topspace Y in spec)
        by (auto simp: openin_prod_Times_iff mem_Times_iff Pi_iff cong:
conj_cong)
      with True show continuous_map Z X (fst  $\circ$  f)
        by (auto simp: continuous_map_def)
    next
      assume continuous_map Z (prod_topology X Y) f
      then have openin Z {x  $\in$  topspace Z. snd (f x)  $\in$  V} if openin Y V for V
        unfolding continuous_map_def using True that
        apply clarify
        apply (drule_tac x=topspace X  $\times$  V in spec)
      by (simp add: openin_prod_Times_iff mem_Times_iff Pi_iff cong: conj_cong)
      with True show continuous_map Z Y (snd  $\circ$  f)
        by (auto simp: continuous_map_def)
    next

```

```

assume Z: continuous_map Z X (fst ∘ f) continuous_map Z Y (snd ∘ f)
have *: openin Z {x ∈ topspace Z. f x ∈ W}
  if  $\bigwedge w. w \in W \implies \exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge w \in U \times V \wedge U$ 
  × V ⊆ W for W
proof (subst openin_subopen, clarify)
  fix x :: 'a
  assume x ∈ topspace Z and f x ∈ W
  with that [OF ⟨f x ∈ W⟩]
  obtain U V where UV: openin X U openin Y V f x ∈ U × V U × V ⊆ W
  by auto
  with Z UV show  $\exists T. \text{openin } Z T \wedge x \in T \wedge T \subseteq \{x \in \text{topspace } Z. f x$ 
  ∈ W}
  apply (rule_tac x={x ∈ topspace Z. ?g x ∈ U} ∩ {x ∈ topspace Z. ?h x
  ∈ V} in exI)
  apply (auto simp: ⟨x ∈ topspace Z⟩ continuous_map_def)
  done
qed
show continuous_map Z (prod_topology X Y) f
  using True by (force simp: continuous_map_def openin_prod_topology_alt
  mem_Times_iff *)
qed
qed (force simp: continuous_map_def)
qed

```

lemma *continuous_map_paired*:

```

continuous_map Z (prod_topology X Y) (λx. (f x, g x)) ↔ continuous_map Z
X f ∧ continuous_map Z Y g
by (simp add: continuous_map_pairwise o_def)

```

lemma *continuous_map_pairedI* [*continuous_intros*]:

```

[[continuous_map Z X f; continuous_map Z Y g]] ⇒ continuous_map Z (prod_topology
X Y) (λx. (f x, g x))
by (simp add: continuous_map_pairwise o_def)

```

lemma *continuous_map_fst* [*continuous_intros*]: *continuous_map* (*prod_topology*
X Y) X *fst*

```

using continuous_map_pairwise [of prod_topology X Y X Y id]
by (simp add: continuous_map_pairwise)

```

lemma *continuous_map_snd* [*continuous_intros*]: *continuous_map* (*prod_topology*
X Y) Y *snd*

```

using continuous_map_pairwise [of prod_topology X Y X Y id]
by (simp add: continuous_map_pairwise)

```

lemma *continuous_map_fst_of* [*continuous_intros*]:

```

continuous_map Z (prod_topology X Y) f ⇒ continuous_map Z X (fst ∘ f)
by (simp add: continuous_map_pairwise)

```

lemma *continuous_map_snd_of* [*continuous_intros*]:

$continuous_map\ Z\ (prod_topology\ X\ Y)\ f \implies continuous_map\ Z\ Y\ (snd \circ f)$
by (*simp add: continuous_map_pairwise*)

lemma *continuous_map_prod_fst*:

$i \in I \implies continuous_map\ (prod_topology\ (product_topology\ (\lambda i. Y)\ I)\ X)\ Y$
 $(\lambda x. fst\ x\ i)$
using *continuous_map_componentwise_UNIV continuous_map_fst* **by** *fastforce*

lemma *continuous_map_prod_snd*:

$i \in I \implies continuous_map\ (prod_topology\ X\ (product_topology\ (\lambda i. Y)\ I))\ Y$
 $(\lambda x. snd\ x\ i)$
using *continuous_map_componentwise_UNIV continuous_map_snd* **by** *fastforce*

lemma *continuous_map_if_iff* [*simp*]: $continuous_map\ X\ Y\ (\lambda x. if\ P\ then\ f\ x\ else\ g\ x) \longleftrightarrow continuous_map\ X\ Y\ (if\ P\ then\ f\ else\ g)$
by *simp*

lemma *continuous_map_if* [*continuous_intros*]: $\llbracket P \implies continuous_map\ X\ Y\ f; \sim P \implies continuous_map\ X\ Y\ g \rrbracket$
 $\implies continuous_map\ X\ Y\ (\lambda x. if\ P\ then\ f\ x\ else\ g\ x)$
by *simp*

lemma *prod_topology_trivial1* [*simp*]: $prod_topology\ trivial_topology\ Y = trivial_topology$
using *continuous_map_fst continuous_map_on_empty2* **by** *blast*

lemma *prod_topology_trivial2* [*simp*]: $prod_topology\ X\ trivial_topology = trivial_topology$
using *continuous_map_snd continuous_map_on_empty2* **by** *blast*

lemma *continuous_map_subtopology_fst* [*continuous_intros*]: $continuous_map\ (subtopology\ (prod_topology\ X\ Y)\ Z)\ X\ fst$
using *continuous_map_from_subtopology continuous_map_fst* **by** *force*

lemma *continuous_map_subtopology_snd* [*continuous_intros*]: $continuous_map\ (subtopology\ (prod_topology\ X\ Y)\ Z)\ Y\ snd$
using *continuous_map_from_subtopology continuous_map_snd* **by** *force*

lemma *quotient_map_fst* [*simp*]:

$quotient_map\ (prod_topology\ X\ Y)\ X\ fst \longleftrightarrow (Y = trivial_topology \longrightarrow X = trivial_topology)$
apply (*simp add: continuous_open_quotient_map open_map_fst continuous_map_fst*)
by (*metis null_topospace_iff_trivial*)

lemma *quotient_map_snd* [*simp*]:

$quotient_map\ (prod_topology\ X\ Y)\ Y\ snd \longleftrightarrow (X = trivial_topology \longrightarrow Y = trivial_topology)$
apply (*simp add: continuous_open_quotient_map open_map_snd continuous_map_snd*)

by (*metis null_topospace_iff_trivial*)

lemma *retraction_map_fst*:

retraction_map (prod_topology X Y) X fst \longleftrightarrow (*Y = trivial_topology* \longrightarrow *X = trivial_topology*)

proof (*cases Y = trivial_topology*)

case *True*

then show *?thesis*

using *continuous_map_image_subset_topospace*

by (*auto simp: retraction_map_def retraction_maps_def continuous_map_pairwise*)

next

case *False*

have $\exists g. \text{continuous_map } X \text{ (prod_topology } X \text{ Y) } g \wedge (\forall x \in \text{topospace } X. \text{fst } (g \ x) = x)$

if *y: y* \in *topospace Y* for *y*

by (*rule_tac x = $\lambda x. (x, y)$ in exI*) (*auto simp: y continuous_map_paired*)

with *False* have *retraction_map (prod_topology X Y) X fst*

by (*fastforce simp: retraction_map_def retraction_maps_def continuous_map_fst*)

with *False* show *?thesis*

by *simp*

qed

lemma *retraction_map_snd*:

retraction_map (prod_topology X Y) Y snd \longleftrightarrow (*X = trivial_topology* \longrightarrow *Y = trivial_topology*)

proof (*cases X = trivial_topology*)

case *True*

then show *?thesis*

using *continuous_map_image_subset_topospace*

by (*fastforce simp: retraction_map_def retraction_maps_def continuous_map_snd*)

next

case *False*

have $\exists g. \text{continuous_map } Y \text{ (prod_topology } X \text{ Y) } g \wedge (\forall y \in \text{topospace } Y. \text{snd } (g \ y) = y)$

if *x: x* \in *topospace X* for *x*

by (*rule_tac x = $\lambda y. (x, y)$ in exI*) (*auto simp: x continuous_map_paired*)

with *False* have *retraction_map (prod_topology X Y) Y snd*

by (*fastforce simp: retraction_map_def retraction_maps_def continuous_map_snd*)

with *False* show *?thesis*

by *simp*

qed

lemma *continuous_map_of_fst*:

continuous_map (prod_topology X Y) Z (f \circ fst) \longleftrightarrow *Y = trivial_topology* \vee *continuous_map X Z f*

proof (*cases Y = trivial_topology*)

case *True*

then show *?thesis*

```

    by (simp add: continuous_map_on_empty)
next
  case False
  then show ?thesis
    by (simp add: continuous_compose_quotient_map_eq)
qed

```

```

lemma continuous_map_of_snd:
  continuous_map (prod_topology X Y) Z (f ∘ snd) ↔ X = trivial_topology ∨
  continuous_map Y Z f
proof (cases X = trivial_topology)
  case True
  then show ?thesis
    by (simp add: continuous_map_on_empty)
next
  case False
  then show ?thesis
    by (simp add: continuous_compose_quotient_map_eq)
qed

```

```

lemma continuous_map_prod_top:
  continuous_map (prod_topology X Y) (prod_topology X' Y') (λ(x,y). (f x, g y))
  ↔
  (prod_topology X Y) = trivial_topology ∨ continuous_map X X' f ∧ continu-
  ous_map Y Y' g
proof (cases (prod_topology X Y) = trivial_topology)
  case False
  then show ?thesis
    by (auto simp: continuous_map_paired case_prod_unfold
      continuous_map_of_fst [unfolded o_def] continuous_map_of_snd
      [unfolded o_def])
qed auto

```

```

lemma in_prod_topology_closure_of:
  assumes z ∈ (prod_topology X Y) closure_of S
  shows fst z ∈ X closure_of (fst ` S) snd z ∈ Y closure_of (snd ` S)
  using assms continuous_map_eq_image_closure_subset continuous_map_fst
  apply fastforce
  using assms continuous_map_eq_image_closure_subset continuous_map_snd
  apply fastforce
  done

```

```

proposition compact_space_prod_topology:
  compact_space (prod_topology X Y) ↔ (prod_topology X Y) = trivial_topology
  ∨ compact_space X ∧ compact_space Y
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis

```

```

    by fastforce
next
case False
then have non_mt: topspace X ≠ {} topspace Y ≠ {}
  by auto
have compact_space X compact_space Y if compact_space (prod_topology X Y)
proof -
  have compactin X (fst ' (topspace X × topspace Y))
    by (metis compact_space_def continuous_map_fst image_compactin that
topspace_prod_topology)
  moreover
  have compactin Y (snd ' (topspace X × topspace Y))
    by (metis compact_space_def continuous_map_snd image_compactin that
topspace_prod_topology)
  ultimately show compact_space X compact_space Y
    using non_mt by (auto simp: compact_space_def)
qed
moreover
define  $\mathcal{X}$  where  $\mathcal{X} \equiv (\lambda V. \text{topspace } X \times V) \text{ ' Collect (openin } Y)$ 
define  $\mathcal{Y}$  where  $\mathcal{Y} \equiv (\lambda U. U \times \text{topspace } Y) \text{ ' Collect (openin } X)$ 
have compact_space (prod_topology X Y) if compact_space X compact_space Y
proof (rule Alexander_subbase_alt)
  show toptop: topspace X × topspace Y ⊆ ⋃ (X ∪ Y)
    unfolding X_def Y_def by auto
  fix C :: ('a × 'b) set set
  assume C: C ⊆ X ∪ Y topspace X × topspace Y ⊆ ⋃ C
  then obtain X' Y' where XY: X' ⊆ X Y' ⊆ Y and Ceq: C = X' ∪ Y'
    using subset_UnE by metis
  then have sub: topspace X × topspace Y ⊆ ⋃ (X' ∪ Y')
    using C by simp
  obtain U V where U: ⋀ U. U ∈ U ⇒ openin X U Y' = (λ U. U × topspace
Y) ' U
    and V: ⋀ V. V ∈ V ⇒ openin Y V X' = (λ V. topspace X × V) ' V
    using XY by (clarsimp simp add: X_def Y_def subset_image_iff) (force
simp: subset_iff)
  have ∃ D. finite D ∧ D ⊆ X' ∪ Y' ∧ topspace X × topspace Y ⊆ ⋃ D
  proof -
    have topspace X ⊆ ⋃ U ∨ topspace Y ⊆ ⋃ V
      using U V C Ceq by auto
    then have *: ∃ D. finite D ∧
      (∀ x ∈ D. x ∈ (λ V. topspace X × V) ' V ∨ x ∈ (λ U. U × topspace
Y) ' U) ∧
      (topspace X × topspace Y ⊆ ⋃ D)
    proof
      assume topspace X ⊆ ⋃ U
      with ⟨compact_space X⟩ U obtain F where finite F F ⊆ U topspace X
⊆ ⋃ F
        by (meson compact_space_alt)
      with that show ?thesis

```

```

    by (rule_tac x=( $\lambda D. D \times \text{topspace } Y$ ) '  $\mathcal{F}$  in exI) auto
  next
    assume  $\text{topspace } Y \subseteq \bigcup \mathcal{V}$ 
    with  $\langle \text{compact\_space } Y \rangle \mathcal{V}$  obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq \mathcal{V}$   $\text{topspace } Y$ 
 $\subseteq \bigcup \mathcal{F}$ 
    by (meson compact_space_alt)
    with that show ?thesis
    by (rule_tac x=( $\lambda C. \text{topspace } X \times C$ ) '  $\mathcal{F}$  in exI) auto
  qed
  then show ?thesis
    using that  $U \mathcal{V}$  by blast
  qed
  then show  $\exists \mathcal{D}. \text{finite } \mathcal{D} \wedge \mathcal{D} \subseteq \mathcal{C} \wedge \text{topspace } X \times \text{topspace } Y \subseteq \bigcup \mathcal{D}$ 
    using  $\mathcal{C} \text{ Ceq}$  by blast
  next
    have (finite intersection_of ( $\lambda x. x \in \mathcal{X} \vee x \in \mathcal{Y}$ ) relative_to  $\text{topspace } X \times$ 
 $\text{topspace } Y$ )
      = ( $\lambda U. \exists S T. U = S \times T \wedge \text{openin } X S \wedge \text{openin } Y T$ )
      (is ?lhs = ?rhs)
    proof -
      have ?rhs  $U$  if ?lhs  $U$  for  $U$ 
    proof -
      have  $\text{topspace } X \times \text{topspace } Y \cap \bigcap T \in \{A \times B \mid A B. A \in \text{Collect } (\text{openin}$ 
 $X) \wedge B \in \text{Collect } (\text{openin } Y)\}$ 
        if finite  $T$   $T \subseteq \mathcal{X} \cup \mathcal{Y}$  for  $T$ 
        using that
      proof induction
        case (insert  $B \mathcal{B}$ )
        then show ?case
          unfolding  $\mathcal{X}$ _def  $\mathcal{Y}$ _def
            apply (simp add: Int_ac subset_eq image_def)
            apply (metis (no_types) openin_Int openin_topspace Times_Int_Times)
          done
      qed auto
      then show ?thesis
        using that
        by (auto simp: subset_eq elim!: relative_toE intersection_ofE)
    qed
  moreover
  have ?lhs  $Z$  if  $Z: ?rhs Z$  for  $Z$ 
  proof -
    obtain  $U V$  where  $Z = U \times V$   $\text{openin } X U$   $\text{openin } Y V$ 
      using  $Z$  by blast
    then have  $UV: U \times V = (\text{topspace } X \times \text{topspace } Y) \cap (U \times V)$ 
      by (simp add: Sigma_mono inf_absorb2 openin_subset)
    moreover
    have ?lhs  $((\text{topspace } X \times \text{topspace } Y) \cap (U \times V))$ 
  proof (rule relative_to_inc)
    show (finite intersection_of ( $\lambda x. x \in \mathcal{X} \vee x \in \mathcal{Y}$ ))  $(U \times V)$ 

```

```

apply (simp add: intersection_of_def  $\mathcal{X}$ _def  $\mathcal{Y}$ _def)
apply (rule_tac x={( $U \times \text{topspace } Y$ ),( $\text{topspace } X \times V$ )} in exI)
using <openin  $X \ U$ > <openin  $Y \ V$ > openin_subset  $UV$  apply (fastforce
simp:)
done
qed
ultimately show ?thesis
using < $Z = U \times V$ > by auto
qed
ultimately show ?thesis
by meson
qed
then show topology (arbitrary_union_of (finite_intersection_of ( $\lambda x. x \in \mathcal{X} \cup$ 
 $\mathcal{Y}$ )
relative_to ( $\text{topspace } X \times \text{topspace } Y$ ))) =
prod_topology  $X \ Y$ 
by (simp add: prod_topology_def)
qed
ultimately show ?thesis
using False by blast
qed

```

lemma compactin_Times:

$\text{compactin } (\text{prod_topology } X \ Y) (S \times T) \iff S = \{\} \vee T = \{\} \vee \text{compactin } X$
 $S \wedge \text{compactin } Y \ T$

by (auto simp: compactin_subspace subtopology_Times compact_space_prod_topology
subtopology_trivial_iff)

2.7.3 Homeomorphic maps

lemma homeomorphic_maps_prod:

$\text{homeomorphic_maps } (\text{prod_topology } X \ Y) (\text{prod_topology } X' \ Y') (\lambda(x,y). (f \ x,$
 $g \ y)) (\lambda(x,y). (f' \ x, g' \ y)) \iff$
 $(\text{prod_topology } X \ Y) = \text{trivial_topology} \wedge (\text{prod_topology } X' \ Y') = \text{triv-$
 ial_topology

$\vee \text{homeomorphic_maps } X \ X' \ f \ f' \wedge \text{homeomorphic_maps } Y \ Y' \ g \ g' \ (\text{is } ?lhs$
 $= ?rhs)$

proof

show ?lhs \implies ?rhs

by (fastforce simp: homeomorphic_maps_def continuous_map_prod_top ball_conj_distrib)

next

show ?rhs \implies ?lhs

by (auto simp: homeomorphic_maps_def continuous_map_prod_top)

qed

lemma homeomorphic_maps_swap:

$\text{homeomorphic_maps } (\text{prod_topology } X \ Y) (\text{prod_topology } Y \ X) (\lambda(x,y). (y,x))$
 $(\lambda(y,x). (x,y))$

by (*auto simp: homeomorphic_maps_def case_prod_unfold continuous_map_fst continuous_map_pairedI continuous_map_snd*)

lemma *homeomorphic_map_swap*:

homeomorphic_map (*prod_topology* *X Y*) (*prod_topology* *Y X*) ($\lambda(x,y). (y,x)$)
using *homeomorphic_map_maps homeomorphic_maps_swap* **by** *metis*

lemma *homeomorphic_space_prod_topology_swap*:

(*prod_topology* *X Y*) *homeomorphic_space* (*prod_topology* *Y X*)
using *homeomorphic_map_swap homeomorphic_space* **by** *blast*

lemma *embedding_map_graph*:

embedding_map *X* (*prod_topology* *X Y*) ($\lambda x. (x, f x)$) \longleftrightarrow *continuous_map* *X*
Y f
(is ?lhs = ?rhs)

proof

assume *L*: *?lhs*

have *snd* $\circ (\lambda x. (x, f x)) = f$

by *force*

moreover have *continuous_map* *X Y* (*snd* $\circ (\lambda x. (x, f x))$)

using *L unfolding embedding_map_def*

by (*meson continuous_map_in_subtopology continuous_map_snd_of homeomorphic_imp_continuous_map*)

ultimately show *?rhs*

by *simp*

next

assume *R*: *?rhs*

then show *?lhs*

unfolding *homeomorphic_map_maps embedding_map_def homeomorphic_maps_def*

by (*rule_tac x=fst in exI*)

(*auto simp: continuous_map_in_subtopology continuous_map_paired continuous_map_from_subtopology continuous_map_fst*)

qed

lemma *homeomorphic_space_prod_topology*:

$\llbracket X \text{ homeomorphic_space } X''; Y \text{ homeomorphic_space } Y' \rrbracket$

$\implies \text{prod_topology } X Y \text{ homeomorphic_space prod_topology } X'' Y'$

using *homeomorphic_maps_prod unfolding homeomorphic_space_def* **by** *blast*

lemma *prod_topology_homeomorphic_space_left*:

$Y = \text{discrete_topology } \{b\} \implies \text{prod_topology } X Y \text{ homeomorphic_space } X$

unfolding *homeomorphic_space_def*

apply (*rule_tac x=fst in exI*)

apply (*simp add: homeomorphic_map_def inj_on_def discrete_topology_unique flip: homeomorphic_map_maps*)

done

lemma *prod_topology_homeomorphic_space_right*:

$X = \text{discrete_topology } \{a\} \implies \text{prod_topology } X \ Y \ \text{homeomorphic_space } Y$
unfolding *homeomorphic_space_def*
by (*meson homeomorphic_space_def homeomorphic_space_prod_topology_swap*
homeomorphic_space_trans prod_topology_homeomorphic_space_left)

lemma *homeomorphic_space_prod_topology_sing1*:

$b \in \text{topspace } Y \implies X \ \text{homeomorphic_space} \ (\text{prod_topology } X \ (\text{subtopology } Y \ \{b\}))$

by (*metis empty_subsetI homeomorphic_space_sym insert_subset prod_topology_homeomorphic_space_subtopology_eq_discrete_topology_sing topspace_subtopology_subset*)

lemma *homeomorphic_space_prod_topology_sing2*:

$a \in \text{topspace } X \implies Y \ \text{homeomorphic_space} \ (\text{prod_topology} \ (\text{subtopology } X \ \{a\}) \ Y)$

by (*metis empty_subsetI homeomorphic_space_sym insert_subset prod_topology_homeomorphic_space_subtopology_eq_discrete_topology_sing topspace_subtopology_subset*)

lemma *topological_property_of_prod_component*:

assumes *major*: $P(\text{prod_topology } X \ Y)$

and $X: \bigwedge x. \llbracket x \in \text{topspace } X; P(\text{prod_topology } X \ Y) \rrbracket \implies P(\text{subtopology} \ (\text{prod_topology } X \ Y) \ (\{x\} \times \text{topspace } Y))$

and $Y: \bigwedge y. \llbracket y \in \text{topspace } Y; P(\text{prod_topology } X \ Y) \rrbracket \implies P(\text{subtopology} \ (\text{prod_topology } X \ Y) \ (\text{topspace } X \times \{y\}))$

and $PQ: \bigwedge X \ X'. X \ \text{homeomorphic_space } X' \implies (P \ X \ \longleftrightarrow \ Q \ X')$

and $PR: \bigwedge X \ X'. X \ \text{homeomorphic_space } X' \implies (P \ X \ \longleftrightarrow \ R \ X')$

shows $(\text{prod_topology } X \ Y) = \text{trivial_topology} \vee Q \ X \ \wedge \ R \ Y$

proof –

have $Q \ X \ \wedge \ R \ Y$ **if** $\text{topspace}(\text{prod_topology } X \ Y) \neq \{\}$

proof –

from that obtain $a \ b$ **where** $a: a \in \text{topspace } X$ **and** $b: b \in \text{topspace } Y$

by force

show *?thesis*

using X [*OF a major*] **and** Y [*OF b major*] *homeomorphic_space_prod_topology_sing1*
[*OF b, of X*] *homeomorphic_space_prod_topology_sing2* [*OF a, of Y*]

by (*simp add: subtopology_Times*) (*meson PQ PR homeomorphic_space_prod_topology_sing2*
homeomorphic_space_sym)

qed

then show *?thesis* **by force**

qed

lemma *limitin_pairwise*:

$\text{limitin} \ (\text{prod_topology } X \ Y) \ f \ l \ F \ \longleftrightarrow \ \text{limitin } X \ (fst \circ f) \ (fst \ l) \ F \ \wedge \ \text{limitin } Y \ (snd \circ f) \ (snd \ l) \ F$

(**is** *?lhs* = *?rhs*)

proof

assume *?lhs*

then obtain $a \ b$ **where** $ev: \bigwedge U. \llbracket (a,b) \in U; \text{openin} \ (\text{prod_topology } X \ Y) \ U \rrbracket$
 $\implies \forall_F \ x \ \text{in } F. f \ x \in U$

```

      and a: a ∈ topspace X and b: b ∈ topspace Y and l: l = (a,b)
    by (auto simp: limitin_def)
  moreover have ∀F x in F. fst (f x) ∈ U if openin X U a ∈ U for U
  proof -
    have ∀F c in F. f c ∈ U × topspace Y
      using b that ev [of U × topspace Y] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
      by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  moreover have ∀F x in F. snd (f x) ∈ U if openin Y U b ∈ U for U
  proof -
    have ∀F c in F. f c ∈ topspace X × U
      using a that ev [of topspace X × U] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
      by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  ultimately show ?rhs
    by (simp add: limitin_def)
next
  have limitin (prod_topology X Y) f (a,b) F
    if limitin X (fst ∘ f) a F limitin Y (snd ∘ f) b F for a b
    using that
  proof (clarsimp simp: limitin_def)
    fix Z :: ('a × 'b) set
    assume a: a ∈ topspace X ∀ U. openin X U ∧ a ∈ U ⟶ (∀F x in F. fst (f
x) ∈ U)
    and b: b ∈ topspace Y ∀ U. openin Y U ∧ b ∈ U ⟶ (∀F x in F. snd (f x)
∈ U)
    and Z: openin (prod_topology X Y) Z (a, b) ∈ Z
    then obtain U V where openin X U openin Y V a ∈ U b ∈ V U × V ⊆ Z
      using Z by (force simp: openin_prod_topology_alt)
    then have ∀F x in F. fst (f x) ∈ U ∀F x in F. snd (f x) ∈ V
      by (simp_all add: a b)
    then show ∀F x in F. f x ∈ Z
      by (rule eventually_elim2) (use ⟨U × V ⊆ Z⟩ subsetD in auto)
  qed
  then show ?rhs ⟹ ?lhs
    by (metis prod.collapse)
qed

proposition connected_space_prod_topology:
  connected_space(prod_topology X Y) ⟷
  (prod_topology X Y) = trivial_topology ∨ connected_space X ∧ connected_space
Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    by auto
next

```

```

case False
then have nonempty: topspace X ≠ {} topspace Y ≠ {}
  by force+
show ?thesis
proof
  assume ?lhs
  then show ?rhs
  by (metis connected_space_quotient_map_image nonempty quotient_map_fst
quotient_map_snd
subtopology_eq_discrete_topology_empty)
next
  assume ?rhs
  then have conX: connected_space X and conY: connected_space Y
  using False by blast+
  have False
  if openin (prod_topology X Y) U and openin (prod_topology X Y) V
  and UV: topspace X × topspace Y ⊆ U ∪ V U ∩ V = {}
  and U ≠ {} and V ≠ {}
  for U V
  proof -
  have Usub: U ⊆ topspace X × topspace Y and Vsub: V ⊆ topspace X ×
topspace Y
  using that by (metis openin_subset topspace_prod_topology)+
  obtain a b where ab: (a,b) ∈ U and a: a ∈ topspace X and b: b ∈ topspace
Y
  using ⟨U ≠ {}⟩ Usub by auto
  have ¬ topspace X × topspace Y ⊆ U
  using Usub Vsub ⟨U ∩ V = {}⟩ ⟨V ≠ {}⟩ by auto
  then obtain x y where x: x ∈ topspace X and y: y ∈ topspace Y and (x,y)
∉ U
  by blast
  have oX: openin X {x ∈ topspace X. (x,y) ∈ U} openin X {x ∈ topspace X.
(x,y) ∈ V}
  and oY: openin Y {y ∈ topspace Y. (a,y) ∈ U} openin Y {y ∈ topspace Y.
(a,y) ∈ V}
  by (force intro: openin_continuous_map_preimage [where Y = prod_topology
X Y]
simp: that continuous_map_pairwise o_def x y a)+
  have 1: topspace Y ⊆ {y ∈ topspace Y. (a,y) ∈ U} ∪ {y ∈ topspace Y. (a,y)
∈ V}
  using a that(3) by auto
  have 2: {y ∈ topspace Y. (a,y) ∈ U} ∩ {y ∈ topspace Y. (a,y) ∈ V} = {}
  using that(4) by auto
  have 3: {y ∈ topspace Y. (a,y) ∈ U} ≠ {}
  using ab b by auto
  have 4: {y ∈ topspace Y. (a,y) ∈ V} ≠ {}
  proof -
  show ?thesis
  using connected_spaceD [OF conX oX] UV ⟨(x,y) ∉ U⟩ a x y

```

```

      disjoint_iff_not_equal by blast
    qed
    show ?thesis
      using connected_spaceD [OF conY oY 1 2 3 4] by auto
    qed
    then show ?lhs
      unfolding connected_space_def topspace_prod_topology by blast
    qed
  qed

lemma connectedin_Times:
  connectedin (prod_topology X Y) (S × T) ↔
    S = {} ∨ T = {} ∨ connectedin X S ∧ connectedin Y T
  by (auto simp: connectedin_def subtopology_Times connected_space_prod_topology
    subtopology_trivial_iff)

end

```

2.8 T1 and Hausdorff spaces

```

theory T1_Spaces
imports Product_Topology
begin

```

2.9 T1 spaces with equivalences to many naturally "nice" properties.

definition *t1_space* where
 $t1_space\ X \equiv \forall x \in topspace\ X. \forall y \in topspace\ X. x \neq y \longrightarrow (\exists U. openin\ X\ U \wedge x \in U \wedge y \notin U)$

lemma *t1_space_expansive*:
 $\llbracket topspace\ Y = topspace\ X; \bigwedge U. openin\ X\ U \implies openin\ Y\ U \rrbracket \implies t1_space\ X \implies t1_space\ Y$
 by (metis t1_space_def)

lemma *t1_space_alt*:
 $t1_space\ X \longleftrightarrow (\forall x \in topspace\ X. \forall y \in topspace\ X. x \neq y \longrightarrow (\exists U. closedin\ X\ U \wedge x \in U \wedge y \notin U))$
 by (metis DiffE DiffI closedin_def openin_closedin_eq t1_space_def)

lemma *t1_space_empty* [iff]: *t1_space trivial_topology*
 by (simp add: t1_space_def)

lemma *t1_space_derived_set_of_singleton*:
 $t1_space\ X \longleftrightarrow (\forall x \in topspace\ X. X\ derived_set_of\ \{x\} = \{x\})$
 apply (simp add: t1_space_def derived_set_of_def, safe)
 apply (metis openin_topspace)

by force

lemma *t1_space_derived_set_of_finite*:

$t1_space\ X \longleftrightarrow (\forall S. finite\ S \longrightarrow X\ derived_set_of\ S = \{\})$

proof (intro iffI allI impI)

fix $S :: 'a\ set$

assume *finite S*

then have *fin*: $finite\ ((\lambda x. \{x\})\ ' (topspace\ X \cap S))$

by blast

assume *t1_space X*

then have $X\ derived_set_of\ (\bigcup x \in\ topspace\ X \cap S. \{x\}) = \{\}$

unfolding *derived_set_of_Union [OF fin]*

by (*auto simp: t1_space_derived_set_of_singleton*)

then have $X\ derived_set_of\ (topspace\ X \cap S) = \{\}$

by *simp*

then show $X\ derived_set_of\ S = \{\}$

by *simp*

qed (*auto simp: t1_space_derived_set_of_singleton*)

lemma *t1_space_closedin_singleton*:

$t1_space\ X \longleftrightarrow (\forall x \in\ topspace\ X. closedin\ X\ \{x\})$

apply (*rule iffI*)

apply (*simp add: closedin_contains_derived_set t1_space_derived_set_of_singleton*)

using *t1_space_alt* **by** *auto*

lemma *continuous_closed_imp_proper_map*:

$\llbracket compact_space\ X; t1_space\ Y; continuous_map\ X\ Y\ f; closed_map\ X\ Y\ f \rrbracket$

$\implies proper_map\ X\ Y\ f$

unfolding *proper_map_def*

by (*smt (verit) closedin_compact_space closedin_continuous_map_preimage*

Collect_cong singleton_iff t1_space_closedin_singleton)

lemma *t1_space_euclidean*: $t1_space\ (euclidean\ ::\ 'a::metric_space\ topology)$

by (*simp add: t1_space_closedin_singleton*)

lemma *closedin_t1_singleton*:

$\llbracket t1_space\ X; a \in\ topspace\ X \rrbracket \implies closedin\ X\ \{a\}$

by (*simp add: t1_space_closedin_singleton*)

lemma *t1_space_closedin_finite*:

$t1_space\ X \longleftrightarrow (\forall S. finite\ S \wedge S \subseteq\ topspace\ X \longrightarrow closedin\ X\ S)$

apply (*rule iffI*)

apply (*simp add: closedin_contains_derived_set t1_space_derived_set_of_finite*)

by (*simp add: t1_space_closedin_singleton*)

lemma *closure_of_singleton*:

$t1_space\ X \implies X\ closure_of\ \{a\} = (if\ a \in\ topspace\ X\ then\ \{a\}\ else\ \{\})$

by (*simp add: closure_of_eq t1_space_closedin_singleton closure_of_eq_empty_gen*)

lemma *separated_in_singleton*:
assumes *t1_space X*
shows *separatedin X {a} S \longleftrightarrow a \in topspace X \wedge S \subseteq topspace X \wedge (a \notin X closure_of S)*
separatedin X S {a} \longleftrightarrow a \in topspace X \wedge S \subseteq topspace X \wedge (a \notin X closure_of S)
unfolding *separatedin_def*
using *assms closure_of_closure_of_singleton by fastforce+*

lemma *t1_space_openin_delete*:
t1_space X \longleftrightarrow ($\forall U x. \text{openin } X \ U \wedge x \in U \longrightarrow \text{openin } X \ (U - \{x\})$)
apply (*rule iffI*)
apply (*meson closedin_t1_singleton_in_mono openin_diff openin_subset*)
by (*simp add: closedin_def t1_space_closedin_singleton*)

lemma *t1_space_openin_delete_alt*:
t1_space X \longleftrightarrow ($\forall U x. \text{openin } X \ U \longrightarrow \text{openin } X \ (U - \{x\})$)
by (*metis Diff_empty Diff_insert0 t1_space_openin_delete*)

lemma *t1_space_singleton_Inter_open*:
t1_space X \longleftrightarrow ($\forall x \in \text{topspace } X. \bigcap \{U. \text{openin } X \ U \wedge x \in U\} = \{x\}$) (**is** *?P=?Q*)
and *t1_space_Inter_open_supersets*:
t1_space X \longleftrightarrow ($\forall S. S \subseteq \text{topspace } X \longrightarrow \bigcap \{U. \text{openin } X \ U \wedge S \subseteq U\} = S$) (**is** *?P=?R*)
proof –
have *?R \implies ?Q*
apply *clarify*
apply (*drule_tac x={x} in spec, simp*)
done
moreover have *?Q \implies ?P*
apply (*clarsimp simp add: t1_space_def*)
apply (*drule_tac x=x in bspec*)
apply (*simp_all add: set_eq_iff*)
by (*metis (no_types, lifting)*)
moreover have *?P \implies ?R*
proof (*clarsimp simp add: t1_space_closedin_singleton, rule subset_antisym*)
fix *S*
assume *S: $\forall x \in \text{topspace } X. \text{closedin } X \ \{x\} \ S \subseteq \text{topspace } X$*
then show $\bigcap \{U. \text{openin } X \ U \wedge S \subseteq U\} \subseteq S$
apply *clarsimp*
by (*metis Diff_insert_absorb Set.set_insert closedin_def openin_topspace subset_insert*)
qed force
ultimately show *?P=?Q ?P=?R*
by auto
qed

lemma *t1_space_derived_set_of_infinite_openin*:

$t1_space\ X \longleftrightarrow$
 $(\forall S. X\ derived_set_of\ S =$
 $\{x \in\ topspace\ X. \forall U. x \in U \wedge openin\ X\ U \longrightarrow infinite(S \cap U)\})$
 $(is\ _ =\ ?rhs)$

proof

assume *t1_space X*

show *?rhs*

proof safe

fix *S x U*

assume $x \in X\ derived_set_of\ S\ x \in U\ openin\ X\ U\ finite\ (S \cap U)$

with $\langle t1_space\ X \rangle$ **show** *False*

apply (*simp add: t1_space_derived_set_of_finite*)

by (*metis IntI empty_iff empty_subsetI inf_commute openin_Int derived_set_of_subset subset_antisym*)

next

fix *S x*

have $eq: (\exists y. (y \neq x) \wedge y \in S \wedge y \in T) \longleftrightarrow \sim((S \cap T) \subseteq \{x\})$ **for** $x\ S\ T$

by *blast*

assume $x \in topspace\ X\ \forall U. x \in U \wedge openin\ X\ U \longrightarrow infinite\ (S \cap U)$

then show $x \in X\ derived_set_of\ S$

apply (*clarsimp simp add: derived_set_of_def eq*)

by (*meson finite.emptyI finite.insertI finite_subset*)

qed (*auto simp: in_derived_set_of*)

qed (*auto simp: t1_space_derived_set_of_singleton*)

lemma *finite_t1_space_imp_discrete_topology*:

$\llbracket topspace\ X = U; finite\ U; t1_space\ X \rrbracket \Longrightarrow X = discrete_topology\ U$

by (*metis discrete_topology_unique_derived_set t1_space_derived_set_of_finite*)

lemma *t1_space_subtopology*: $t1_space\ X \Longrightarrow t1_space(subtopology\ X\ U)$

by (*simp add: derived_set_of_subtopology t1_space_derived_set_of_finite*)

lemma *closedin_derived_set_of_gen*:

$t1_space\ X \Longrightarrow closedin\ X\ (X\ derived_set_of\ S)$

apply (*clarsimp simp add: in_derived_set_of closedin_contains_derived_set derived_set_of_subset_topspace*)

by (*metis DiffD2 insert_Diff insert_iff t1_space_openin_delete*)

lemma *derived_set_of_derived_set_subset_gen*:

$t1_space\ X \Longrightarrow X\ derived_set_of\ (X\ derived_set_of\ S) \subseteq X\ derived_set_of\ S$

by (*meson closedin_contains_derived_set closedin_derived_set_of_gen*)

lemma *subtopology_eq_discrete_topology_gen_finite*:

$\llbracket t1_space\ X; finite\ S \rrbracket \Longrightarrow subtopology\ X\ S = discrete_topology(tospace\ X \cap S)$

by (*simp add: subtopology_eq_discrete_topology_gen t1_space_derived_set_of_finite*)

lemma *subtopology_eq_discrete_topology_finite*:


```

[[t1_space X; S ⊆ topspace X; finite S]]
  ⇒ subtopyology X S = discrete_topoylogy S
by (simp add: subtopyology_eq_discrete_topoylogy_eq t1_space_derived_set_of_finite)

lemma t1_space_closed_map_image:
  [[closed_map X Y f; f '(topspace X) = topspace Y; t1_space X]] ⇒ t1_space
  Y
  by (metis closed_map_def finite_subset_image t1_space_closedin_finite)

lemma homeomorphic_t1_space: X homeomorphic_space Y ⇒ (t1_space X ↔
t1_space Y)
  apply (clarsimp simp add: homeomorphic_space_def)
  by (meson homeomorphic_eq_everything_map homeomorphic_maps_map t1_space_closed_map_image)

proposition t1_space_product_topoylogy:
  t1_space (product_topoylogy X I)
  ↔ (product_topoylogy X I) = trivial_topoylogy ∨ (∀ i ∈ I. t1_space (X i))
proof (cases (product_topoylogy X I) = trivial_topoylogy)
  case True
  then show ?thesis
    using True t1_space_empty by force
next
  case False
  then obtain f where f: f ∈ (ΠE i∈I. topspace(X i))
    using discrete_topoylogy_unique by (fastforce iff: null_topspace_iff_trivial)
  have t1_space (product_topoylogy X I) ↔ (∀ i∈I. t1_space (X i))
  proof (intro iffI ballI)
    show t1_space (X i) if t1_space (product_topoylogy X I) and i ∈ I for i
    proof -
      have clo: ∧h. h ∈ (ΠE i∈I. topspace (X i)) ⇒ closedin (product_topoylogy
X I) {h}
      using that by (simp add: t1_space_closedin_singleton)
      show ?thesis
      unfolding t1_space_closedin_singleton
      proof clarify
        show closedin (X i) {xi} if xi ∈ topspace (X i) for xi
          using clo [of λj ∈ I. if i=j then xi else f j] f that ⟨i ∈ I⟩
          by (fastforce simp add: closedin_product_topoylogy_singleton)
      qed
    qed
  next
  next
  show t1_space (product_topoylogy X I) if ∀ i∈I. t1_space (X i)
    using that
    by (simp add: t1_space_closedin_singleton Ball_def PiE_iff closedin_product_topoylogy_singleton)
  qed
  then show ?thesis
    using False by blast
qed

```

```

lemma t1_space_prod_topology:
  t1_space(prod_topology X Y)  $\longleftrightarrow$  (prod_topology X Y) = trivial_topology  $\vee$ 
t1_space X  $\wedge$  t1_space Y
proof (cases (prod_topology X Y) = trivial_topology)
  case True then show ?thesis
  by auto
next
  case False
  have eq:  $\{(x,y)\} = \{x\} \times \{y\}$  for x::'a and y::'b
  by simp
  have t1_space (prod_topology X Y)  $\longleftrightarrow$  (t1_space X  $\wedge$  t1_space Y)
  using False
  apply(simp add: t1_space_closedin_singleton closedin_prod_Times_iff eq
    del: insert_Times_insert flip: null_topspace_iff_trivial ex_in_conv)
  by blast
  with False show ?thesis
  by simp
qed

```

2.9.1 Hausdorff Spaces

definition *Hausdorff_space*

where

Hausdorff_space *X* \equiv

$\forall x y. x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge (x \neq y)$

$\longrightarrow (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U V)$

lemma *Hausdorff_space_expansive*:

$\llbracket \text{Housdorff_space } X; \text{topspace } X = \text{topspace } Y; \bigwedge U. \text{openin } X U \implies \text{openin } Y U \rrbracket \implies \text{Housdorff_space } Y$

by (*metis* *Hausdorff_space_def*)

lemma *Hausdorff_space_topspace_empty* [*iff*]: *Hausdorff_space* *trivial_topology*

by (*simp* *add*: *Hausdorff_space_def*)

lemma *Hausdorff_imp_t1_space*:

Hausdorff_space *X* \implies *t1_space* *X*

by (*metis* *Hausdorff_space_def* *disjnt_iff* *t1_space_def*)

lemma *closedin_derived_set_of*:

Hausdorff_space *X* \implies *closedin* *X* (*X* *derived_set_of* *S*)

by (*simp* *add*: *Hausdorff_imp_t1_space* *closedin_derived_set_of_gen*)

lemma *t1_or_Hausdorff_space*:

t1_space *X* \vee *Hausdorff_space* *X* \longleftrightarrow *t1_space* *X*

using *Hausdorff_imp_t1_space* **by** *blast*

lemma *Hausdorff_space_sing_Inter_opens*:

$\llbracket \text{Hausdorff_space } X; a \in \text{topspace } X \rrbracket \implies \bigcap \{u. \text{openin } X \ u \wedge a \in u\} = \{a\}$
using *Hausdorff_imp_t1_space t1_space_singleton_Inter_open* **by force**

lemma *Hausdorff_space_subtopology*:

assumes *Hausdorff_space* *X* **shows** *Hausdorff_space*(*subtopology* *X* *S*)

proof –

have $*$: *disjnt* *U* *V* \implies *disjnt* (*S* \cap *U*) (*S* \cap *V*) **for** *U* *V*

by (*simp* *add*: *disjnt_iff*)

from *assms* **show** *?thesis*

apply (*simp* *add*: *Hausdorff_space_def* *openin_subtopology_alt*)

apply (*fast* *intro*: $*$ *elim*!: *all_forward*)

done

qed

lemma *Hausdorff_space_compact_separation*:

assumes *X*: *Hausdorff_space* *X* **and** *S*: *compactin* *X* *S* **and** *T*: *compactin* *X* *T*

and *disjnt* *S* *T*

obtains *U* *V* **where** *openin* *X* *U* *openin* *X* *V* $S \subseteq U$ $T \subseteq V$ *disjnt* *U* *V*

proof (*cases* *S* = $\{\}$)

case *True*

then show *thesis*

by (*metis* \langle *compactin* *X* *T* \rangle *compactin_subset_topspace* *disjnt_empty1* *empty_subsetI* *openin_empty* *openin_topspace* *that*)

next

case *False*

have $\forall x \in S. \exists U V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge x \in U \wedge T \subseteq V \wedge \text{disjnt } U \ V$

proof

fix *a*

assume $a \in S$

then have $a \notin T$

by (*meson* *assms*(\mathcal{A}) *disjnt_iff*)

have $a: a \in \text{topspace } X$

using $S \langle a \in S \rangle$ *compactin_subset_topspace* **by** *blast*

show $\exists U V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge a \in U \wedge T \subseteq V \wedge \text{disjnt } U \ V$

proof (*cases* *T* = $\{\}$)

case *True*

then show *?thesis*

using a *disjnt_empty2* *openin_empty* **by** *blast*

next

case *False*

have $\forall x \in \text{topspace } X - \{a\}. \exists U V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge x \in U \wedge a \in V \wedge \text{disjnt } U \ V$

using $X \ a$ **by** (*simp* *add*: *Hausdorff_space_def*)

then obtain *U* *V* **where** $UV: \forall x \in \text{topspace } X - \{a\}. \text{openin } X \ (U \ x) \wedge \text{openin } X \ (V \ x) \wedge x \in U \ x \wedge a \in V \ x \wedge \text{disjnt } (U \ x) \ (V \ x)$

by *metis*

with $\langle a \notin T \rangle$ *compactin_subset_topspace* [*OF* *T*]

```

have Topen:  $\forall W \in U' T. \text{openin } X W$  and Tsub:  $T \subseteq \bigcup (U' T)$ 
  by auto
then obtain  $\mathcal{F}$  where  $\mathcal{F}$ : finite  $\mathcal{F}$   $\mathcal{F} \subseteq U' T$  and  $T \subseteq \bigcup \mathcal{F}$ 
  using T unfolding compactin_def by meson
then obtain  $F$  where  $F$ : finite  $F$   $F \subseteq T$   $\mathcal{F} = U' F$  and  $SUF$ :  $T \subseteq \bigcup (U' F)$ 
and  $a \notin F$ 
  using finite_subset_image [OF  $\mathcal{F}$ ]  $\langle a \notin T \rangle$  by (metis subsetD)
have  $U$ :  $\bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{openin } X (U x)$ 
  and  $V$ :  $\bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{openin } X (V x)$ 
  and disj:  $\bigwedge x. \llbracket x \in \text{topspace } X; x \neq a \rrbracket \implies \text{disjnt } (U x) (V x)$ 
  using UV by blast+
show ?thesis
proof (intro exI conjI)
  have  $F \neq \{\}$ 
    using False  $SUF$  by blast
  with  $\langle a \notin F \rangle$  show  $\text{openin } X (\bigcap (V' F))$ 
    using F compactin_subset_topspace [OF T] by (force intro: V)
  show  $\text{openin } X (\bigcup (U' F))$ 
    using F Topen Tsub by (force intro: U)
  show  $\text{disjnt } (\bigcap (V' F)) (\bigcup (U' F))$ 
    using disj
    apply (auto simp: disjnt_def)
    using  $\langle F \subseteq T \rangle \langle a \notin F \rangle$  compactin_subset_topspace [OF T] by blast
  show  $a \in (\bigcap (V' F))$ 
    using  $\langle F \subseteq T \rangle T UV \langle a \notin T \rangle$  compactin_subset_topspace by blast
  qed (auto simp:  $SUF$ )
qed
qed
then obtain  $U V$  where  $UV$ :  $\forall x \in S. \text{openin } X (U x) \wedge \text{openin } X (V x) \wedge x \in U x \wedge T \subseteq V x \wedge \text{disjnt } (U x) (V x)$ 
  by metis
then have  $S \subseteq \bigcup (U' S)$ 
  by auto
moreover have  $\forall W \in U' S. \text{openin } X W$ 
  using UV by blast
ultimately obtain  $I$  where  $I$ :  $S \subseteq \bigcup (U' I)$   $I \subseteq S$  finite  $I$ 
  by (metis S compactin_def finite_subset_image)
show thesis
proof
  show  $\text{openin } X (\bigcup (U' I))$ 
    using  $\langle I \subseteq S \rangle UV$  by blast
  show  $\text{openin } X (\bigcap (V' I))$ 
    using False UV  $\langle I \subseteq S \rangle \langle S \subseteq \bigcup (U' I) \rangle \langle \text{finite } I \rangle$  by blast
  show  $\text{disjnt } (\bigcup (U' I)) (\bigcap (V' I))$ 
    by simp (meson UV  $\langle I \subseteq S \rangle$  disjnt_subset2 in_mono le_INF_iff order_refl)
  qed (use UV I in auto)
qed

```

lemma *Hausdorff_space_compact_sets*:

Hausdorff_space $X \longleftrightarrow$
 $(\forall S T. \text{compactin } X S \wedge \text{compactin } X T \wedge \text{disjnt } S T$
 $\longrightarrow (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$
 $V))$
(is *?lhs* = *?rhs*)

proof

assume *?lhs*

then show *?rhs*

by (*meson Hausdorff_space_compact_separation*)

next

assume R [*rule_format*]: *?rhs*

show *?lhs*

proof (*clarsimp simp add: Hausdorff_space_def*)

fix $x y$

assume $x \in \text{topspace } X \ y \in \text{topspace } X \ x \neq y$

then show $\exists U. \text{openin } X U \wedge (\exists V. \text{openin } X V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U V)$

using R [*of* $\{x\}$ $\{y\}$] **by** *auto*

qed

qed

lemma *compactin_imp_closedin*:

assumes X : *Hausdorff_space* X **and** S : *compactin* $X S$ **shows** *closedin* $X S$

proof –

have $S \subseteq \text{topspace } X$

by (*simp add: assms compactin_subset_topspace*)

moreover

have $\exists T. \text{openin } X T \wedge x \in T \wedge T \subseteq \text{topspace } X - S$ **if** $x \in \text{topspace } X \ x \notin S$

for x

using *Hausdorff_space_compact_separation* [*OF* $X _ S$, *of* $\{x\}$] **that**

apply (*simp add: disjnt_def*)

by (*metis Diff_mono Diff_triv openin_subset*)

ultimately show *?thesis*

using *closedin_def openin_subopen* **by** *force*

qed

lemma *closedin_Hausdorff_singleton*:

$\llbracket \text{Hausdorff_space } X; x \in \text{topspace } X \rrbracket \Longrightarrow \text{closedin } X \{x\}$

by (*simp add: Hausdorff_imp_t1_space closedin_t1_singleton*)

lemma *closedin_Hausdorff_sing_eq*:

Hausdorff_space $X \Longrightarrow \text{closedin } X \{x\} \longleftrightarrow x \in \text{topspace } X$

by (*meson closedin_Hausdorff_singleton closedin_subset insert_subset*)

lemma *Hausdorff_space_discrete_topology* [*simp*]:

Hausdorff_space (*discrete_topology* U)

unfolding *Hausdorff_space_def*

by (*metis Hausdorff_space_compact_sets Hausdorff_space_def compactin_discrete_topology*)

equality *E* *openin_discrete_topology*)

lemma *compactin_Int*:

$\llbracket \text{Hausdorff_space } X; \text{compactin } X S; \text{compactin } X T \rrbracket \implies \text{compactin } X (S \cap T)$
by (*simp add: closed_Int_compactin compactin_imp_closedin*)

lemma *finite_topospace_imp_discrete_topology*:

$\llbracket \text{topospace } X = U; \text{finite } U; \text{Hausdorff_space } X \rrbracket \implies X = \text{discrete_topology } U$
using *Hausdorff_imp_t1_space finite_t1_space_imp_discrete_topology* **by** *blast*

lemma *derived_set_of_finite*:

$\llbracket \text{Hausdorff_space } X; \text{finite } S \rrbracket \implies X \text{ derived_set_of } S = \{\}$
using *Hausdorff_imp_t1_space t1_space_derived_set_of_finite* **by** *auto*

lemma *infinite_perfect_set*:

$\llbracket \text{Hausdorff_space } X; S \subseteq X \text{ derived_set_of } S; S \neq \{\} \rrbracket \implies \text{infinite } S$
using *derived_set_of_finite* **by** *blast*

lemma *derived_set_of_singleton*:

$\llbracket \text{Hausdorff_space } X \rrbracket \implies X \text{ derived_set_of } \{x\} = \{\}$
by (*simp add: derived_set_of_finite*)

lemma *closedin_Hausdorff_finite*:

$\llbracket \text{Hausdorff_space } X; S \subseteq \text{topospace } X; \text{finite } S \rrbracket \implies \text{closedin } X S$
by (*simp add: compactin_imp_closedin finite_imp_compactin_eq*)

lemma *open_in_Hausdorff_delete*:

$\llbracket \text{Hausdorff_space } X; \text{openin } X S \rrbracket \implies \text{openin } X (S - \{x\})$
using *Hausdorff_imp_t1_space t1_space_openin_delete_alt* **by** *auto*

lemma *closedin_Hausdorff_finite_eq*:

$\llbracket \text{Hausdorff_space } X; \text{finite } S \rrbracket \implies \text{closedin } X S \iff S \subseteq \text{topospace } X$
by (*meson closedin_Hausdorff_finite closedin_def*)

lemma *derived_set_of_infinite_openin*:

$\text{Hausdorff_space } X$
 $\implies X \text{ derived_set_of } S =$
 $\{x \in \text{topospace } X. \forall U. x \in U \wedge \text{openin } X U \longrightarrow \text{infinite}(S \cap U)\}$
using *Hausdorff_imp_t1_space t1_space_derived_set_of_infinite_openin* **by** *fastforce*

lemma *Hausdorff_space_discrete_compactin*:

$\text{Hausdorff_space } X$
 $\implies S \cap X \text{ derived_set_of } S = \{\} \wedge \text{compactin } X S \iff S \subseteq \text{topospace } X \wedge$
 $\text{finite } S$
using *derived_set_of_finite discrete_compactin_eq_finite* **by** *fastforce*

lemma *Hausdorff_space_finite_topospace*:

$\text{Hausdorff_space } X \implies X \text{ derived_set_of } (\text{topospace } X) = \{\} \wedge \text{compact_space}$

$X \longleftrightarrow \text{finite}(\text{topspace } X)$
using *derived_set_of_finite discrete_compact_space_eq_finite* **by** *auto*

lemma *derived_set_of_derived_set_subset*:
 $\text{Hausdorff_space } X \implies X \text{ derived_set_of } (X \text{ derived_set_of } S) \subseteq X \text{ derived_set_of } S$
by (*simp add: Hausdorff_imp_t1_space derived_set_of_derived_set_subset_gen*)

lemma *Hausdorff_space_injective_preimage*:
assumes *Hausdorff_space Y* **and** *cmf: continuous_map X Y f* **and** *inj_on f (topspace X)*
shows *Hausdorff_space X*
unfolding *Hausdorff_space_def*
proof *clarify*
fix $x\ y$
assume $x: x \in \text{topspace } X$ **and** $y: y \in \text{topspace } X$ **and** $x \neq y$
then obtain $U\ V$ **where** *openin Y U openin Y V f x \in U f y \in V disjoint U V*
using *assms*
by (*smt (verit, ccfv_threshold) Hausdorff_space_def continuous_map_image_subset_iff inj_on_def*)
show $\exists U\ V. \text{openin } X\ U \wedge \text{openin } X\ V \wedge x \in U \wedge y \in V \wedge \text{disjoint } U\ V$
proof (*intro exI conjI*)
show *openin X {x \in topspace X. f x \in U}*
using $\langle \text{openin } Y\ U \rangle$ *cmf continuous_map* **by** *fastforce*
show *openin X {x \in topspace X. f x \in V}*
using $\langle \text{openin } Y\ V \rangle$ *cmf openin_continuous_map_preimage* **by** *blast*
show *disjnt {x \in topspace X. f x \in U} {x \in topspace X. f x \in V}*
using $\langle \text{disjnt } U\ V \rangle$ **by** (*auto simp add: disjnt_def*)
qed (*use x \langle f x \in U \rangle y \langle f y \in V \rangle in auto*)
qed

lemma *homeomorphic_Hausdorff_space*:
 $X \text{ homeomorphic_space } Y \implies \text{Hausdorff_space } X \longleftrightarrow \text{Hausdorff_space } Y$
unfolding *homeomorphic_space_def homeomorphic_maps_map*
by (*auto simp: homeomorphic_eq_everything_map Hausdorff_space_injective_preimage*)

lemma *Hausdorff_space_retraction_map_image*:
 $\llbracket \text{retraction_map } X\ Y\ r; \text{Hausdorff_space } X \rrbracket \implies \text{Hausdorff_space } Y$
unfolding *retraction_map_def*
using *Hausdorff_space_subtopology homeomorphic_Hausdorff_space retraction_maps_section_image2*
by *blast*

lemma *compact_Hausdorff_space_optimal*:
assumes *eq: topspace Y = topspace X* **and** $XY: \bigwedge U. \text{openin } X\ U \implies \text{openin } Y\ U$
and *Hausdorff_space X compact_space Y*
shows $Y = X$
proof –

```

have  $\bigwedge U. \text{closedin } X U \implies \text{closedin } Y U$ 
  using XY using topology_finer_closedin [OF eq]
  by metis
have openin Y S = openin X S for S
  by (metis XY assms(3) assms(4) closedin_compact_space compactin_contractive
compactin_imp_closedin eq openin_closedin_eq)
  then show ?thesis
    by (simp add: topology_eq)
qed

```

```

lemma continuous_map_imp_closed_graph:
  assumes f: continuous_map X Y f and Y: Hausdorff_space Y
  shows closedin (prod_topology X Y) (( $\lambda x. (x, f x)$ ) 'topspace X)
  unfolding closedin_def
proof
  show ( $\lambda x. (x, f x)$ ) 'topspace X  $\subseteq$  topspace (prod_topology X Y)
    using continuous_map_def f by fastforce
  show openin (prod_topology X Y) (topspace (prod_topology X Y) - ( $\lambda x. (x, f x)$ ) 'topspace X)
    unfolding openin_prod_topology_alt
  proof (intro allI impI)
    show  $\exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge x \in U \wedge y \in V \wedge U \times V \subseteq \text{topspace}$ 
      (prod_topology X Y) - ( $\lambda x. (x, f x)$ ) 'topspace X
      if ( $x, y \in \text{topspace (prod\_topology X Y) - (\lambda x. (x, f x)) 'topspace X}$ )
      for x y
    proof -
      have  $x \in \text{topspace } X$  and  $y: y \in \text{topspace } Y \ y \neq f x$ 
        using that by auto
      then have  $f x \in \text{topspace } Y$ 
        using continuous_map_image_subset_topspace f by blast
      then obtain U V where UV: openin Y U openin Y V  $f x \in U \ y \in V \ \text{disjnt}$ 
        U V
        using Y y Hausdorff_space_def by metis
      show ?thesis
        proof (intro exI conjI)
          show openin X { $x \in \text{topspace } X. f x \in U$ }
            using  $\langle \text{openin } Y U \rangle f$  openin_continuous_map_preimage by blast
          show { $x \in \text{topspace } X. f x \in U$ }  $\times V \subseteq \text{topspace (prod\_topology X Y) -}$ 
            ( $\lambda x. (x, f x)$ ) 'topspace X
            using UV by (auto simp: disjnt_iff dest: openin_subset)
          qed (use UV  $\langle x \in \text{topspace } X \rangle$  in auto)
        qed
    qed
  qed

```

```

lemma continuous_imp_closed_map:
   $\llbracket \text{continuous\_map } X Y f; \text{compact\_space } X; \text{Hausdorff\_space } Y \rrbracket \implies \text{closed\_map}$ 
  X Y f
  by (meson closed_map_def closedin_compact_space compactin_imp_closedin

```


image_compactin)

lemma *continuous_imp_quotient_map*:

$\llbracket \text{continuous_map } X \ Y \ f; \text{ compact_space } X; \text{ Hausdorff_space } Y; f' \ (\text{topspace } X) = \text{topspace } Y \rrbracket$

$\implies \text{quotient_map } X \ Y \ f$

by (*simp add: continuous_imp_closed_map continuous_closed_imp_quotient_map*)

lemma *continuous_imp_homeomorphic_map*:

$\llbracket \text{continuous_map } X \ Y \ f; \text{ compact_space } X; \text{ Hausdorff_space } Y;$

$f' \ (\text{topspace } X) = \text{topspace } Y; \text{ inj_on } f \ (\text{topspace } X) \rrbracket$

$\implies \text{homeomorphic_map } X \ Y \ f$

by (*simp add: continuous_imp_closed_map bijective_closed_imp_homeomorphic_map*)

lemma *continuous_imp_embedding_map*:

$\llbracket \text{continuous_map } X \ Y \ f; \text{ compact_space } X; \text{ Hausdorff_space } Y; \text{ inj_on } f \ (\text{topspace } X) \rrbracket$

$\implies \text{embedding_map } X \ Y \ f$

by (*simp add: continuous_imp_closed_map injective_closed_imp_embedding_map*)

lemma *continuous_inverse_map*:

assumes *compact_space X Hausdorff_space Y*

and *cmf: continuous_map X Y f and gf: $\bigwedge x. x \in \text{topspace } X \implies g(f\ x) = x$*

and *Sf: $S \subseteq f' \ (\text{topspace } X)$*

shows *continuous_map (subtopology Y S) X g*

proof (*rule continuous_map_from_subtopology_mono [OF _ $\langle S \subseteq f' \ (\text{topspace } X) \rangle$]*)

show *continuous_map (subtopology Y (f' (topspace X))) X g*

unfolding *continuous_map_closedin*

proof (*intro conjI ballI allI impI*)

show $g \in \text{topspace (subtopology Y (f' topspace X))} \rightarrow \text{topspace } X$

using *gf by auto*

next

fix *C*

assume *C: closedin X C*

show *closedin (subtopology Y (f' topspace X))*

$\{x \in \text{topspace (subtopology Y (f' topspace X))}. g\ x \in C\}$

proof (*rule compactin_imp_closedin*)

show *Hausdorff_space (subtopology Y (f' topspace X))*

using *Hausdorff_space_subtopology [OF $\langle \text{Hausdorff_space } Y \rangle$]* **by** *blast*

have *compactin Y (f' C)*

using *C cmf image_compactin closedin_compact_space [OF $\langle \text{compact_space } X \rangle$]* **by** *blast*

moreover **have** $\{x \in \text{topspace } Y. x \in f' \ \text{topspace } X \wedge g\ x \in C\} = f' \ C$

using *closedin_subset [OF C] cmf* **by** (*auto simp: gf continuous_map_def*)

ultimately **have** *compactin Y $\{x \in \text{topspace } Y. x \in f' \ \text{topspace } X \wedge g\ x \in C\}$*

C

by *simp*

then **show** *compactin (subtopology Y (f' topspace X))*

$\{x \in \text{topspace } (\text{subtopology } Y (f \text{ ' } \text{topspace } X)). g \ x \in C\}$
by (*auto simp add: compactin_subtopology*)
qed
qed
qed

lemma *closed_map_paired_continuous_map_right*:
 $\llbracket \text{continuous_map } X \ Y \ f; \text{Hausdorff_space } Y \rrbracket \implies \text{closed_map } X \ (\text{prod_topology } X \ Y) \ (\lambda x. (x, f \ x))$
by (*simp add: continuous_map_imp_closed_graph embedding_map_graph embedding_imp_closed_map*)

lemma *closed_map_paired_continuous_map_left*:
assumes *f: continuous_map X Y f and Y: Hausdorff_space Y*
shows $\text{closed_map } X \ (\text{prod_topology } Y \ X) \ (\lambda x. (f \ x, x))$
proof –
have *eq: $(\lambda x. (f \ x, x)) = (\lambda (a, b). (b, a)) \circ (\lambda x. (x, f \ x))$*
by *auto*
show *?thesis*
unfolding *eq*
proof (*rule closed_map_compose*)
show $\text{closed_map } X \ (\text{prod_topology } X \ Y) \ (\lambda x. (x, f \ x))$
using *Y closed_map_paired_continuous_map_right f by blast*
show $\text{closed_map } (\text{prod_topology } X \ Y) \ (\text{prod_topology } Y \ X) \ (\lambda (a, b). (b, a))$
by (*metis homeomorphic_map_swap homeomorphic_imp_closed_map*)
qed
qed

lemma *proper_map_paired_continuous_map_right*:
 $\llbracket \text{continuous_map } X \ Y \ f; \text{Hausdorff_space } Y \rrbracket$
 $\implies \text{proper_map } X \ (\text{prod_topology } X \ Y) \ (\lambda x. (x, f \ x))$
using *closed_injective_imp_proper_map closed_map_paired_continuous_map_right*
by (*metis (mono_tags, lifting) Pair_inject inj_onI*)

lemma *proper_map_paired_continuous_map_left*:
 $\llbracket \text{continuous_map } X \ Y \ f; \text{Hausdorff_space } Y \rrbracket$
 $\implies \text{proper_map } X \ (\text{prod_topology } Y \ X) \ (\lambda x. (f \ x, x))$
using *closed_injective_imp_proper_map closed_map_paired_continuous_map_left*
by (*metis (mono_tags, lifting) Pair_inject inj_onI*)

lemma *Hausdorff_space_prod_topology*:
 $\text{Hausdorff_space}(\text{prod_topology } X \ Y) \iff (\text{prod_topology } X \ Y) = \text{trivial_topology}$
 $\vee \text{Hausdorff_space } X \wedge \text{Hausdorff_space } Y$
(is ?lhs = ?rhs)
proof
assume *?lhs*
then show *?rhs*
by (*rule topological_property_of_prod_component*) (*auto simp: Hausdorff_space_subtopology homeomorphic_Hausdorff_space*)

```

next
  assume R: ?rhs
  show ?lhs
  proof (cases (topspace X × topspace Y) = {})
    case False
      with R have ne: topspace X ≠ {} topspace Y ≠ {} and X: Hausdorff_space
X and Y: Hausdorff_space Y
      by auto
      show ?thesis
        unfolding Hausdorff_space_def
      proof clarify
        fix x y x' y'
        assume xy: (x, y) ∈ topspace (prod_topology X Y)
          and xy': (x', y') ∈ topspace (prod_topology X Y)
          and *: ∃ U V. openin (prod_topology X Y) U ∧ openin (prod_topology X
Y) V
          ∧ (x, y) ∈ U ∧ (x', y') ∈ V ∧ disjnt U V
          have False if x ≠ x' ∨ y ≠ y'
            using that
          proof
            assume x ≠ x'
            then obtain U V where openin X U openin X V x ∈ U x' ∈ V disjnt U V
            by (metis Hausdorff_space_def X mem_Sigma_iff topspace_prod_topology
xy xy')
            let ?U = U × topspace Y
            let ?V = V × topspace Y
            have openin (prod_topology X Y) ?U openin (prod_topology X Y) ?V
              by (simp_all add: openin_prod_Times_iff ⟨openin X U⟩ ⟨openin X V⟩)
            moreover have disjnt ?U ?V
              by (simp add: ⟨disjnt U V⟩)
            ultimately show False
          using * ⟨x ∈ U⟩ ⟨x' ∈ V⟩ xy xy' by (metis SigmaD2 SigmaI topspace_prod_topology)
          next
            assume y ≠ y'
            then obtain U V where openin Y U openin Y V y ∈ U y' ∈ V disjnt U V
            by (metis Hausdorff_space_def Y mem_Sigma_iff topspace_prod_topology
xy xy')
            let ?U = topspace X × U
            let ?V = topspace X × V
            have openin (prod_topology X Y) ?U openin (prod_topology X Y) ?V
              by (simp_all add: openin_prod_Times_iff ⟨openin Y U⟩ ⟨openin Y V⟩)
            moreover have disjnt ?U ?V
              by (simp add: ⟨disjnt U V⟩)
            ultimately show False
          using * ⟨y ∈ U⟩ ⟨y' ∈ V⟩ xy xy' by (metis SigmaD1 SigmaI topspace_prod_topology)
          qed
        then show x = x' ∧ y = y'
          by blast
      qed
  qed

```

qed *force*
qed

lemma *Hausdorff_space_product_topology:*

$Hausdorff_space\ (product_topology\ X\ I) \longleftrightarrow (\prod_{E\ i \in I}.\ topspace(X\ i)) = \{\}$ \vee
 $(\forall\ i \in I.\ Hausdorff_space\ (X\ i))$
(is ?lhs = ?rhs)

proof

assume *?lhs*

then show *?rhs*

by (*simp add: Hausdorff_space_subtopology PiE_eq_empty_iff homeomorphic_Hausdorff_space*
topological_property_of_product_component)

next

assume *R: ?rhs*

show *?lhs*

proof (*cases* $(\prod_{E\ i \in I}. topspace(X\ i)) = \{\}$)

case *True*

then show *?thesis*

by (*simp add: Hausdorff_space_def*)

next

case *False*

have $\exists\ U\ V.\ openin\ (product_topology\ X\ I)\ U \wedge openin\ (product_topology\ X\ I)\ V \wedge f \in U \wedge g \in V \wedge disjnt\ U\ V$

if *f: f ∈ (∏_{E i ∈ I}. topspace (X i))* **and** *g: g ∈ (∏_{E i ∈ I}. topspace (X i))* **and**
f ≠ g

for *f g :: 'a ⇒ 'b*

proof *–*

obtain *m* **where** *f m ≠ g m*

using $\langle f \neq g \rangle$ **by** *blast*

then have *m ∈ I*

using *f g* **by** *fastforce*

then have *Hausdorff_space (X m)*

using *False that R* **by** *blast*

then obtain *U V* **where** *U: openin (X m) U* **and** *V: openin (X m) V* **and**
f m ∈ U g m ∈ V disjnt U V

by (*metis Hausdorff_space_def PiE_mem <f m ≠ g m> <m ∈ I> f g*)

show *?thesis*

proof (*intro exI conjI*)

let *?U = (∏_{E i ∈ I}. topspace(X i)) ∩ {x. x m ∈ U}*

let *?V = (∏_{E i ∈ I}. topspace(X i)) ∩ {x. x m ∈ V}*

show *openin (product_topology X I) ?U* *openin (product_topology X I) ?V*

using $\langle m \in I \rangle$ *U V*

by (*force simp add: openin_product_topology intro: arbitrary_union_of_inc*
relative_to_inc finite_intersection_of_inc)**+**

show *f ∈ ?U*

using $\langle f m \in U \rangle$ *f* **by** *blast*

show *g ∈ ?V*

```

    using ⟨g m ∈ V⟩ g by blast
  show disjnt ?U ?V
    using ⟨disjnt U V⟩ by (auto simp: PiE_def Pi_def disjnt_def)
  qed
qed
then show ?thesis
  by (simp add: Hausdorff_space_def)
qed
qed

lemma Hausdorff_space_closed_neighbourhood:
  Hausdorff_space X ⟷
  (∀ x ∈ topspace X. ∃ U C. openin X U ∧ closedin X C ∧
    Hausdorff_space(subtopology X C) ∧ x ∈ U ∧ U ⊆ C) (is _ =
  ?rhs)
proof
  assume R: ?rhs
  show Hausdorff_space X
    unfolding Hausdorff_space_def
  proof clarify
    fix x y
    assume x: x ∈ topspace X and y: y ∈ topspace X and x ≠ y
    obtain T C where *: openin X T closedin X C x ∈ T T ⊆ C
      and C: Hausdorff_space (subtopology X C)
      by (meson R ⟨x ∈ topspace X⟩)
    show ∃ U V. openin X U ∧ openin X V ∧ x ∈ U ∧ y ∈ V ∧ disjnt U V
    proof (cases y ∈ C)
    case True
      with * C obtain U V where U: openin (subtopology X C) U
        and V: openin (subtopology X C) V
        and x ∈ U y ∈ V disjnt U V
      unfolding Hausdorff_space_def
    by (smt (verit, best) ⟨x ≠ y⟩ closedin_subset subsetD topspace_subtopology_subset)
    then obtain U' V' where UV': U = U' ∩ C openin X U' V = V' ∩ C
      openin X V'
      by (meson openin_subtopology)
    have disjnt (T ∩ U') V'
      using ⟨disjnt U V⟩ UV' ⟨T ⊆ C⟩ by (force simp: disjnt_iff)
    with ⟨T ⊆ C⟩ have disjnt (T ∩ U') (V' ∪ (topspace X - C))
      unfolding disjnt_def by blast
    moreover
    have openin X (T ∩ U')
      by (simp add: ⟨openin X T⟩ ⟨openin X U'⟩ openin_Int)
    moreover have openin X (V' ∪ (topspace X - C))
      using ⟨closedin X C⟩ ⟨openin X V'⟩ by auto
    ultimately show ?thesis
      using UV' ⟨x ∈ T⟩ ⟨x ∈ U⟩ ⟨y ∈ V⟩ by blast
  next
    case False

```

```

    with * y show ?thesis
    by (force simp: closedin_def disjnt_def)
  qed
  qed
  qed fastforce
end

```

2.10 Lindelöf spaces

```

theory Lindelof_Spaces
imports T1_Spaces
begin

```

definition *Lindelof_space* where

$$\begin{aligned} \text{Lindelof_space } X &\equiv \\ &\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X \ U) \wedge \bigcup \mathcal{U} = \text{topspace } X \\ &\longrightarrow (\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } X) \end{aligned}$$

lemma *Lindelof_spaceD*:

$$\begin{aligned} &[[\text{Lindelof_space } X; \bigwedge U. U \in \mathcal{U} \implies \text{openin } X \ U; \bigcup \mathcal{U} = \text{topspace } X]] \\ &\implies \exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } X \\ &\text{by (auto simp: Lindelof_space_def)} \end{aligned}$$

lemma *Lindelof_space_alt*:

$$\begin{aligned} \text{Lindelof_space } X &\longleftrightarrow \\ &(\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X \ U) \wedge \text{topspace } X \subseteq \bigcup \mathcal{U} \\ &\longrightarrow (\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{V})) \end{aligned}$$

unfolding *Lindelof_space_def*

using *openin_subset* **by** *fastforce*

lemma *compact_imp_Lindelof_space*:

$$\text{compact_space } X \implies \text{Lindelof_space } X$$

unfolding *Lindelof_space_def compact_space*

by (*meson uncountable_infinite*)

lemma *Lindelof_space_topspace_empty*:

$$\text{topspace } X = \{\} \implies \text{Lindelof_space } X$$

using *compact_imp_Lindelof_space compact_space_trivial_topology* **by** *force*

lemma *Lindelof_space_Union*:

assumes \mathcal{U} : *countable* \mathcal{U} **and** *lin*: $\bigwedge U. U \in \mathcal{U} \implies \text{Lindelof_space (subtopology } X \ U)$

shows *Lindelof_space (subtopology } X \ ($\bigcup \mathcal{U}$))*

proof –

have $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{F} \wedge \bigcup \mathcal{U} \cap \bigcup \mathcal{V} = \text{topspace } X \cap \bigcup \mathcal{U}$

if \mathcal{F} : $\mathcal{F} \subseteq \text{Collect (openin } X)$ **and** *UF*: $\bigcup \mathcal{U} \cap \bigcup \mathcal{F} = \text{topspace } X \cap \bigcup \mathcal{U}$

for \mathcal{F}

proof –

```

have  $\bigwedge U. \llbracket U \in \mathcal{U}; U \cap \bigcup \mathcal{F} = \text{topspace } X \cap U \rrbracket$ 
   $\implies \exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{F} \wedge U \cap \bigcup \mathcal{V} = \text{topspace } X \cap U$ 
using lin  $\mathcal{F}$ 
unfolding Lindelof_space_def openin_subtopology_alt Ball_def subset_iff
[symmetric]
by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
then obtain  $g$  where  $g: \bigwedge U. \llbracket U \in \mathcal{U}; U \cap \bigcup \mathcal{F} = \text{topspace } X \cap U \rrbracket$ 
   $\implies \text{countable } (g \ U) \wedge (g \ U) \subseteq \mathcal{F} \wedge U \cap \bigcup (g \ U) =$ 
topspace  $X \cap U$ 
by metis
show ?thesis
proof (intro exI conjI)
show countable  $(\bigcup (g \ \mathcal{U}))$ 
using Int_commute UF g by (fastforce intro: countable_UN [OF  $\mathcal{U}$ ])
show  $\bigcup (g \ \mathcal{U}) \subseteq \mathcal{F}$ 
using  $g \ UF$  by blast
show  $\bigcup \mathcal{U} \cap \bigcup (\bigcup (g \ \mathcal{U})) = \text{topspace } X \cap \bigcup \mathcal{U}$ 
proof
show  $\bigcup \mathcal{U} \cap \bigcup (\bigcup (g \ \mathcal{U})) \subseteq \text{topspace } X \cap \bigcup \mathcal{U}$ 
using  $g \ UF$  by blast
show  $\text{topspace } X \cap \bigcup \mathcal{U} \subseteq \bigcup \mathcal{U} \cap \bigcup (\bigcup (g \ \mathcal{U}))$ 
proof clarsimp
show  $\exists y \in \mathcal{U}. \exists W \in g \ y. x \in W$ 
if  $x \in \text{topspace } X \ x \in V \ V \in \mathcal{U}$  for  $x \ V$ 
proof –
have  $V \cap \bigcup \mathcal{F} = \text{topspace } X \cap V$ 
using  $UF \ \langle V \in \mathcal{U} \rangle$  by blast
with that  $g \ [OF \ \langle V \in \mathcal{U} \rangle]$  show ?thesis by blast
qed
qed
qed
qed
qed
then show ?thesis
unfolding Lindelof_space_def openin_subtopology_alt Ball_def subset_iff
[symmetric]
by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
qed

```

lemma *countable_imp_Lindelof_space:*

assumes *countable*(*topspace* X)

shows *Lindelof_space* X

proof –

have *Lindelof_space* (*subtopology* $X \ (\bigcup x \in \text{topspace } X. \{x\})$)

proof (*rule Lindelof_space_Union*)

show *countable* $((\lambda x. \{x\}) \ \text{topspace } X)$

using *assms* **by** *blast*

show *Lindelof_space* (*subtopology* $X \ U$)

if $U \in (\lambda x. \{x\}) \ \text{topspace } X$ **for** U

```

proof –
  have compactin X U
    using that by force
  then show ?thesis
    by (meson compact_imp_Lindelof_space compact_space_subtopology)
  qed
qed
then show ?thesis
  by simp
qed
lemma Lindelof_space_subtopology:
  Lindelof_space(subtopology X S) ↔
    ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge \text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$ 
     $\rightarrow (\exists V. \text{countable } V \wedge V \subseteq \mathcal{U} \wedge \text{topspace } X \cap S \subseteq \bigcup V)$ )
proof –
  have  $*$ : ( $S \cap \bigcup \mathcal{U} = \text{topspace } X \cap S$ ) = ( $\text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$ )
    if  $\bigwedge x. x \in \mathcal{U} \implies \text{openin } X x$  for  $\mathcal{U}$ 
    by (blast dest: openin_subset [OF that])
  moreover have ( $\mathcal{V} \subseteq \mathcal{U} \wedge S \cap \bigcup \mathcal{V} = \text{topspace } X \cap S$ ) = ( $\mathcal{V} \subseteq \mathcal{U} \wedge \text{topspace } X$ 
 $\cap S \subseteq \bigcup \mathcal{V}$ )
    if  $\forall x. x \in \mathcal{U} \rightarrow \text{openin } X x$   $\text{topspace } X \cap S \subseteq \bigcup \mathcal{U}$  countable  $\mathcal{V}$  for  $\mathcal{U} \mathcal{V}$ 
    using that * by blast
  ultimately show ?thesis
    unfolding Lindelof_space_def openin_subtopology_alt Ball_def
    apply (simp add: all_subset_image imp_conjL ex_countable_subset_image
flip: subset_iff)
    apply (intro all_cong1 imp_cong ex_cong, auto)
    done
qed

lemma Lindelof_space_subtopology_subset:
   $S \subseteq \text{topspace } X$ 
   $\implies (\text{Lindelof\_space}(\text{subtopology } X S) \leftrightarrow$ 
    ( $\forall \mathcal{U}. (\forall U \in \mathcal{U}. \text{openin } X U) \wedge S \subseteq \bigcup \mathcal{U}$ 
     $\rightarrow (\exists V. \text{countable } V \wedge V \subseteq \mathcal{U} \wedge S \subseteq \bigcup V))$ )
  by (metis Lindelof_space_subtopology_topospace_subtopology_topospace_subtopology_subset)

lemma Lindelof_space_closedin_subtopology:
assumes  $X$ : Lindelof_space X and  $\text{clo}$ : closedin X S
shows Lindelof_space (subtopology X S)
proof –
  have  $S \subseteq \text{topspace } X$ 
    by (simp add: clo_closedin_subset)
  then show ?thesis
proof (clarisimp simp add: Lindelof_space_subtopology_subset)
  show  $\exists V. \text{countable } V \wedge V \subseteq \mathcal{F} \wedge S \subseteq \bigcup V$ 
    if  $\forall U \in \mathcal{F}. \text{openin } X U$  and  $S \subseteq \bigcup \mathcal{F}$  for  $\mathcal{F}$ 
proof –
  have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \text{insert}(\text{topspace } X - S) \mathcal{F} \wedge \bigcup \mathcal{V} = \text{topspace } X$ 

```



```

proof (rule Lindelof_spaceD [OF X, of insert (topspace X - S)  $\mathcal{F}$ ])
  show openin X U
    if  $U \in \text{insert } (\text{topspace } X - S) \mathcal{F}$  for U
      using that  $\langle \forall U \in \mathcal{F}. \text{openin } X \ U \rangle$  clo by blast
    show  $\bigcup (\text{insert } (\text{topspace } X - S) \mathcal{F}) = \text{topspace } X$ 
      apply auto
      apply (meson in_mono openin_closedin_eq that(1))
      using UnionE  $\langle S \subseteq \bigcup \mathcal{F} \rangle$  by auto
  qed
  then obtain  $\mathcal{V}$  where countable  $\mathcal{V}$   $\mathcal{V} \subseteq \text{insert } (\text{topspace } X - S) \mathcal{F} \bigcup \mathcal{V} =$ 
  topspace X
    by metis
  with  $\langle S \subseteq \text{topspace } X \rangle$ 
  show ?thesis
    by (rule_tac  $x = (\mathcal{V} - \{\text{topspace } X - S\})$  in exI) auto
  qed
qed
qed

```

lemma Lindelof_space_continuous_map_image:

```

assumes X: Lindelof_space X and f: continuous_map X Y f and fim: f ‘
(topspace X) = topspace Y
shows Lindelof_space Y

```

```

proof –
  have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } Y$ 
    if  $\mathcal{U}: \bigwedge U. U \in \mathcal{U} \implies \text{openin } Y \ U$  and  $UU: \bigcup \mathcal{U} = \text{topspace } Y$  for  $\mathcal{U}$ 
  proof –
    define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\lambda U. \{x \in \text{topspace } X. f \ x \in U\}) \text{ ‘ } \mathcal{U}$ 
    have  $\bigwedge V. V \in \mathcal{V} \implies \text{openin } X \ V$ 
      unfolding  $\mathcal{V}$ _def using  $\mathcal{U}$  continuous_map f by fastforce
    moreover have  $\bigcup \mathcal{V} = \text{topspace } X$ 
      unfolding  $\mathcal{V}$ _def using UU fim by fastforce
    ultimately have  $\exists \mathcal{W}. \text{countable } \mathcal{W} \wedge \mathcal{W} \subseteq \mathcal{V} \wedge \bigcup \mathcal{W} = \text{topspace } X$ 
      using X by (simp add: Lindelof_space_def)
    then obtain  $\mathcal{C}$  where countable  $\mathcal{C}$   $\mathcal{C} \subseteq \mathcal{U}$  and  $\mathcal{C}: (\bigcup U \in \mathcal{C}. \{x \in \text{topspace } X. f$ 
   $x \in U\}) = \text{topspace } X$ 
      by (metis (no_types, lifting)  $\mathcal{V}$ _def countable_subset_image)
    moreover have  $\bigcup \mathcal{C} = \text{topspace } Y$ 
  proof
    show  $\bigcup \mathcal{C} \subseteq \text{topspace } Y$ 
      using UU  $\mathcal{C} \langle \mathcal{C} \subseteq \mathcal{U} \rangle$  by fastforce
    have  $y \in \bigcup \mathcal{C}$  if  $y \in \text{topspace } Y$  for y
  proof –
    obtain x where  $x \in \text{topspace } X$   $y = f \ x$ 
      using that fim by (metis  $\langle y \in \text{topspace } Y \rangle$  imageE)
    with  $\mathcal{C}$  show ?thesis by auto
  qed
  then show topspace Y  $\subseteq \bigcup \mathcal{C}$  by blast
qed

```

```

ultimately show ?thesis
  by blast
qed
then show ?thesis
  unfolding Lindelof_space_def
  by auto
qed

```

lemma *Lindelof_space_quotient_map_image*:
 $\llbracket \text{quotient_map } X \ Y \ q; \text{Lindelof_space } X \rrbracket \implies \text{Lindelof_space } Y$
by (*meson Lindelof_space_continuous_map_image quotient_imp_continuous_map quotient_imp_surjective_map*)

lemma *Lindelof_space_retraction_map_image*:
 $\llbracket \text{retraction_map } X \ Y \ r; \text{Lindelof_space } X \rrbracket \implies \text{Lindelof_space } Y$
using *Abstract_Topology.retraction_imp_quotient_map Lindelof_space_quotient_map_image*
by *blast*

lemma *locally_finite_cover_of_Lindelof_space*:
assumes *X: Lindelof_space X and UU: topspace X $\subseteq \bigcup \mathcal{U}$ and fin: locally_finite_in X \mathcal{U}*

shows *countable \mathcal{U}*

proof –

have *UU_eq: $\bigcup \mathcal{U} = \text{topspace } X$*

by (*meson UU fin locally_finite_in_def subset_antisym*)

obtain *T where T: $\bigwedge x. x \in \text{topspace } X \implies \text{openin } X \ (T \ x) \wedge x \in T \ x \wedge \text{finite } \{U \in \mathcal{U}. U \cap T \ x \neq \{\}\}$*

using *fin unfolding locally_finite_in_def by meson*

then obtain *I where countable I I $\subseteq \text{topspace } X$ and I: $\text{topspace } X \subseteq \bigcup (T \ ` \ I)$*

using *X unfolding Lindelof_space_alt*

by (*drule_tac x=image T (topspace X) in spec*) (*auto simp: ex_countable_subset_image*)

show *?thesis*

proof (*rule countable_subset*)

have $\bigwedge i. i \in I \implies \text{countable } \{U \in \mathcal{U}. U \cap T \ i \neq \{\}\}$

using *T*

by (*meson $\langle I \subseteq \text{topspace } X \rangle \text{in_mono uncountable_infinite}$*)

then show *countable (insert $\{\}$ ($\bigcup i \in I. \{U \in \mathcal{U}. U \cap T \ i \neq \{\}\}$))*

by (*simp add: $\langle \text{countable } I \rangle$*)

qed (*use UU_eq I in auto*)

qed

lemma *Lindelof_space_proper_map_preimage*:
assumes *f: proper_map X Y f and Y: Lindelof_space Y*
shows *Lindelof_space X*

proof (*clarsimp simp: Lindelof_space_alt*)

show $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \text{topspace } X \subseteq \bigcup \mathcal{V}$

if *$\mathcal{U}: \forall U \in \mathcal{U}. \text{openin } X \ U$ and sub_UU: $\text{topspace } X \subseteq \bigcup \mathcal{U}$ for \mathcal{U}*

```

proof –
  have  $\exists \mathcal{V}. \text{finite } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \{x \in \text{topspace } X. f x = y\} \subseteq \bigcup \mathcal{V}$  if  $y \in \text{topspace } Y$ 
for  $y$ 
  proof (rule compactinD)
    show  $\text{compactin } X \{x \in \text{topspace } X. f x = y\}$ 
    using  $f \text{ proper\_map\_def}$  that by fastforce
  qed (use sub_UU U in auto)
  then obtain  $\mathcal{V}$  where  $\mathcal{V}: \bigwedge y. y \in \text{topspace } Y \implies \text{finite } (\mathcal{V} y) \wedge \mathcal{V} y \subseteq \mathcal{U} \wedge$ 
 $\{x \in \text{topspace } X. f x = y\} \subseteq \bigcup (\mathcal{V} y)$ 
  by meson
  define  $\mathcal{W}$  where  $\mathcal{W} \equiv (\lambda y. \text{topspace } Y - \text{image } f (\text{topspace } X - \bigcup (\mathcal{V} y)))$  ‘
 $\text{topspace } Y$ 
  have  $\forall U \in \mathcal{W}. \text{openin } Y U$ 
  using  $f \mathcal{U} \mathcal{V}$  unfolding  $\mathcal{W\_def}$   $\text{proper\_map\_def}$   $\text{closed\_map\_def}$ 
  by (simp add: closedin_diff openin_Union openin_diff subset_iff)
  moreover have  $\text{topspace } Y \subseteq \bigcup \mathcal{W}$ 
  using  $\mathcal{V}$  unfolding  $\mathcal{W\_def}$  by clarsimp fastforce
  ultimately have  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{W} \wedge \text{topspace } Y \subseteq \bigcup \mathcal{V}$ 
  using  $Y$  by (simp add: Lindelof_space_alt)
  then obtain  $I$  where  $\text{countable } I \wedge I \subseteq \text{topspace } Y$ 
  and  $I: \text{topspace } Y \subseteq (\bigcup i \in I. \text{topspace } Y - f^{-1} (\text{topspace } X - \bigcup (\mathcal{V} i)))$ 
  unfolding  $\mathcal{W\_def}$   $\text{ex\_countable\_subset\_image}$  by metis
  show ?thesis
  proof (intro exI conjI)
    have  $\bigwedge i. i \in I \implies \text{countable } (\mathcal{V} i)$ 
    by (meson  $\mathcal{V} \langle I \subseteq \text{topspace } Y \rangle \text{in\_mono uncountable\_infinite}$ )
    with  $\langle \text{countable } I \rangle$  show  $\text{countable } (\bigcup (\mathcal{V} \text{ ` } I))$ 
    by auto
    show  $\bigcup (\mathcal{V} \text{ ` } I) \subseteq \mathcal{U}$ 
    using  $\mathcal{V} \langle I \subseteq \text{topspace } Y \rangle$  by fastforce
    show  $\text{topspace } X \subseteq \bigcup (\bigcup (\mathcal{V} \text{ ` } I))$ 
  proof
    show  $x \in \bigcup (\bigcup (\mathcal{V} \text{ ` } I))$  if  $x \in \text{topspace } X$  for  $x$ 
    proof –
      have  $f x \in \text{topspace } Y$ 
      using  $f \text{ proper\_map\_imp\_subset\_topspace}$  that by fastforce
      then show ?thesis
      using that I by auto
    qed
  qed
  qed
  qed
  qed

```

lemma *Lindelof_space_perfect_map_image:*

$\llbracket \text{Lindelof_space } X; \text{perfect_map } X Y f \rrbracket \implies \text{Lindelof_space } Y$

using *Lindelof_space_quotient_map_image perfect_imp_quotient_map* **by** *blast*

lemma *Lindelof_space_perfect_map_image_eq:*

584

perfect_map $X Y f \implies \text{Lindelof_space } X \longleftrightarrow \text{Lindelof_space } Y$
using *Lindelof_space_perfect_map_image* *Lindelof_space_proper_map_preimage*
perfect_map_def **by** *blast*

end

Chapter 3

Functional Analysis

3.1 A decision procedure for metric spaces

```
theory Metric_Arith
  imports HOL.Real_Vector_Spaces
begin
```

A port of the decision procedure “Formalization of metric spaces in HOL Light” [6] for the type class *metric_space*, with the *Argo* solver as backend.

```
named_theorems metric_prenex
named_theorems metric_nnf
named_theorems metric_unfold
named_theorems metric_pre_arith
```

```
lemmas pre_arith_simps =
  max.bounded_iff max_less_iff_conj
  le_max_iff_disj less_max_iff_disj
  simp_thms HOL.eq_commute
declare pre_arith_simps [metric_pre_arith]
```

```
lemmas unfold_simps =
  Un_iff subset_iff disjoint_iff_not_equal
  Ball_def Bex_def
declare unfold_simps [metric_unfold]
```

```
declare HOL.nnf_simps(4) [metric_prenex]
```

```
lemma imp_prenex [metric_prenex]:

$$\bigwedge P Q. (\exists x. P x) \longrightarrow Q \equiv \forall x. (P x \longrightarrow Q)$$


$$\bigwedge P Q. P \longrightarrow (\exists x. Q x) \equiv \exists x. (P \longrightarrow Q x)$$


$$\bigwedge P Q. (\forall x. P x) \longrightarrow Q \equiv \exists x. (P x \longrightarrow Q)$$


$$\bigwedge P Q. P \longrightarrow (\forall x. Q x) \equiv \forall x. (P \longrightarrow Q x)$$

  by simp_all
```

```
lemma ex_prenex [metric_prenex]:
```

$\bigwedge P Q. (\exists x. P x) \wedge Q \equiv \exists x. (P x \wedge Q)$
 $\bigwedge P Q. P \wedge (\exists x. Q x) \equiv \exists x. (P \wedge Q x)$
 $\bigwedge P Q. (\exists x. P x) \vee Q \equiv \exists x. (P x \vee Q)$
 $\bigwedge P Q. P \vee (\exists x. Q x) \equiv \exists x. (P \vee Q x)$
 $\bigwedge P. \neg(\exists x. P x) \equiv \forall x. \neg P x$
 by *simp_all*

lemma *all_prenex* [*metric_prenex*]:

$\bigwedge P Q. (\forall x. P x) \wedge Q \equiv \forall x. (P x \wedge Q)$
 $\bigwedge P Q. P \wedge (\forall x. Q x) \equiv \forall x. (P \wedge Q x)$
 $\bigwedge P Q. (\forall x. P x) \vee Q \equiv \forall x. (P x \vee Q)$
 $\bigwedge P Q. P \vee (\forall x. Q x) \equiv \forall x. (P \vee Q x)$
 $\bigwedge P. \neg(\forall x. P x) \equiv \exists x. \neg P x$
 by *simp_all*

lemma *nnf_thms* [*metric_nnf*]:

$(\neg (P \wedge Q)) = (\neg P \vee \neg Q)$
 $(\neg (P \vee Q)) = (\neg P \wedge \neg Q)$
 $(P \longrightarrow Q) = (\neg P \vee Q)$
 $(P = Q) \longleftrightarrow (P \vee \neg Q) \wedge (\neg P \vee Q)$
 $(\neg (P = Q)) \longleftrightarrow (\neg P \vee \neg Q) \wedge (P \vee Q)$
 $(\neg \neg P) = P$
 by *blast+*

lemmas *nnf_simps* = *nnf_thms* *linorder_not_less* *linorder_not_le*
declare *nnf_simps*[*metric_nnf*]

lemma *ball_insert*: $(\forall x \in \text{insert } a \ B. P x) = (P a \wedge (\forall x \in B. P x))$
 by *blast*

lemma *Sup_insert_insert*:

fixes *a::real*
shows *Sup* (*insert* *a* (*insert* *b* *s*)) = *Sup* (*insert* (*max* *a* *b*) *s*)
 by (*simp* *add*: *Sup_real_def*)

lemma *real_abs_dist*: $|\text{dist } x \ y| = \text{dist } x \ y$
 by *simp*

lemma *maxdist_thm* [*THEN HOL.eq_reflection*]:

assumes *finite* *s* $x \in s$ $y \in s$
defines $\bigwedge a. f \ a \equiv |\text{dist } x \ a - \text{dist } a \ y|$
shows $\text{dist } x \ y = \text{Sup } (f \ ' \ s)$

proof –

have $\text{dist } x \ y \leq \text{Sup } (f \ ' \ s)$

proof –

have *finite* (*f* ' *s*)

by (*simp* *add*: $\langle \text{finite } s \rangle$)

moreover have $|\text{dist } x \ y - \text{dist } y \ y| \in f \ ' \ s$

by (*metis* $\langle y \in s \rangle$ *f_def* *imageI*)

```

    ultimately show ?thesis
      using le_cSup_finite by simp
  qed
  also have  $Sup (f ` s) \leq dist x y$ 
    using  $\langle x \in s \rangle$  cSUP_least[of s f] abs_dist_diff_le
    unfolding f_def
    by blast
  finally show ?thesis .
  qed

theorem metric_eq_thm [THEN HOL.eq_reflection]:
   $x \in s \implies y \in s \implies x = y \iff (\forall a \in s. dist x a = dist y a)$ 
  by auto

ML_file <metric_arith.ML>

method_setup metric = <
  Scan.succeed (SIMPLE_METHOD' o Metric_Arith.metric_arith_tac)
> prove simple linear statements in metric spaces ( $\forall \exists_p$  fragment)

end

```


Chapter 4

Elementary Metric Spaces

```
theory Elementary_Metric_Spaces
  imports
    Abstract_Topology_2
    Metric_Arith
begin
```

4.1 Open and closed balls

```
definition ball :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where ball x e = {y. dist x y < e}
```

```
definition cball :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where cball x e = {y. dist x y  $\leq$  e}
```

```
definition sphere :: 'a::metric_space  $\Rightarrow$  real  $\Rightarrow$  'a set
  where sphere x e = {y. dist x y = e}
```

```
lemma mem_ball [simp, metric_unfold]:  $y \in \text{ball } x \ e \iff \text{dist } x \ y < e$ 
  by (simp add: ball_def)
```

```
lemma mem_cball [simp, metric_unfold]:  $y \in \text{cball } x \ e \iff \text{dist } x \ y \leq e$ 
  by (simp add: cball_def)
```

```
lemma mem_sphere [simp]:  $y \in \text{sphere } x \ e \iff \text{dist } x \ y = e$ 
  by (simp add: sphere_def)
```

```
lemma ball_trivial [simp]:  $\text{ball } x \ 0 = \{\}$ 
  by auto
```

```
lemma cball_trivial [simp]:  $\text{cball } x \ 0 = \{x\}$ 
  by auto
```

```
lemma sphere_trivial [simp]:  $\text{sphere } x \ 0 = \{x\}$ 
  by auto
```

lemma disjoint_ballI: $\text{dist } x \ y \geq r+s \implies \text{ball } x \ r \cap \text{ball } y \ s = \{\}$
using *dist_triangle_less_add not_le* **by** *fastforce*

lemma disjoint_cballI: $\text{dist } x \ y > r + s \implies \text{cball } x \ r \cap \text{cball } y \ s = \{\}$
by (*metis add_mono disjoint_iff_not_equal dist_triangle2 dual_order.trans leD mem_cball*)

lemma sphere_empty [simp]: $r < 0 \implies \text{sphere } a \ r = \{\}$
for $a :: 'a::\text{metric_space}$
by *auto*

lemma centre_in_ball [simp]: $x \in \text{ball } x \ e \longleftrightarrow 0 < e$
by *simp*

lemma centre_in_cball [simp]: $x \in \text{cball } x \ e \longleftrightarrow 0 \leq e$
by *simp*

lemma ball_subset_cball [simp, intro]: $\text{ball } x \ e \subseteq \text{cball } x \ e$
by (*simp add: subset_eq*)

lemma mem_ball_imp_mem_cball: $x \in \text{ball } y \ e \implies x \in \text{cball } y \ e$
by *auto*

lemma sphere_cball [simp,intro]: $\text{sphere } z \ r \subseteq \text{cball } z \ r$
by *force*

lemma cball_diff_sphere: $\text{cball } a \ r - \text{sphere } a \ r = \text{ball } a \ r$
by *auto*

lemma subset_ball[intro]: $d \leq e \implies \text{ball } x \ d \subseteq \text{ball } x \ e$
by *auto*

lemma subset_cball[intro]: $d \leq e \implies \text{cball } x \ d \subseteq \text{cball } x \ e$
by *auto*

lemma mem_ball_leI: $x \in \text{ball } y \ e \implies e \leq f \implies x \in \text{ball } y \ f$
by *auto*

lemma mem_cball_leI: $x \in \text{cball } y \ e \implies e \leq f \implies x \in \text{cball } y \ f$
by *auto*

lemma cball_trans: $y \in \text{cball } z \ b \implies x \in \text{cball } y \ a \implies x \in \text{cball } z \ (b + a)$
by *metric*

lemma ball_max_Un: $\text{ball } a \ (\max \ r \ s) = \text{ball } a \ r \cup \text{ball } a \ s$
by *auto*

lemma ball_min_Int: $\text{ball } a \ (\min \ r \ s) = \text{ball } a \ r \cap \text{ball } a \ s$

by auto

lemma *cball_max_Un*: $cball\ a\ (max\ r\ s) = cball\ a\ r \cup cball\ a\ s$
by auto

lemma *cball_min_Int*: $cball\ a\ (min\ r\ s) = cball\ a\ r \cap cball\ a\ s$
by auto

lemma *cball_diff_eq_sphere*: $cball\ a\ r - ball\ a\ r = sphere\ a\ r$
by auto

lemma *open_ball* [intro, simp]: $open\ (ball\ x\ e)$

proof –

have $open\ (dist\ x\ -\ \{..\ < e\})$

by (intro open_vimage open_lessThan continuous_intros)

also have $dist\ x\ -\ \{..\ < e\} = ball\ x\ e$

by auto

finally show ?thesis .

qed

lemma *open_contains_ball*: $open\ S \longleftrightarrow (\forall x \in S. \exists e > 0. ball\ x\ e \subseteq S)$
by (simp add: open_dist_subset_eq Ball_def dist_commute)

lemma *openI* [intro?]: $(\bigwedge x. x \in S \implies \exists e > 0. ball\ x\ e \subseteq S) \implies open\ S$
by (auto simp: open_contains_ball)

lemma *openE* [elim?]:

assumes $open\ S\ x \in S$

obtains e where $e > 0\ ball\ x\ e \subseteq S$

using assms unfolding open_contains_ball by auto

lemma *open_contains_ball_eq*: $open\ S \implies x \in S \longleftrightarrow (\exists e > 0. ball\ x\ e \subseteq S)$
by (metis open_contains_ball_subset_eq centre_in_ball)

lemma *ball_eq_empty* [simp]: $ball\ x\ e = \{\} \longleftrightarrow e \leq 0$
unfolding mem_ball set_eq_iff
by (simp add: not_less) metric

lemma *ball_empty*: $e \leq 0 \implies ball\ x\ e = \{\}$
by simp

lemma *closed_cball* [iff]: $closed\ (cball\ x\ e)$

proof –

have $closed\ (dist\ x\ -\ \{..\ e\})$

by (intro closed_vimage closed_atMost continuous_intros)

also have $dist\ x\ -\ \{..\ e\} = cball\ x\ e$

by auto

finally show ?thesis .

qed

lemma *open_contains_cball*: $open\ S \longleftrightarrow (\forall x \in S. \exists e > 0. cball\ x\ e \subseteq S)$

proof –

```

{
  fix x and e::real
  assume x∈S e>0 ball x e ⊆ S
  then have ∃ d>0. cball x d ⊆ S
    unfolding subset_eq by (rule_tac x=e/2 in exI, auto)
}
moreover
{
  fix x and e::real
  assume x∈S e>0 cball x e ⊆ S
  then have ∃ d>0. ball x d ⊆ S
    using mem_ball_imp_mem_cball by blast
}
ultimately show ?thesis
  unfolding open_contains_ball by auto

```

qed

lemma *open_contains_cball_eq*: $open\ S \implies (\forall x. x \in S \longleftrightarrow (\exists e > 0. cball\ x\ e \subseteq S))$

by (*metis open_contains_cball subset_eq order_less_imp_le centre_in_cball*)

lemma *eventually_nhds_ball*: $d > 0 \implies eventually\ (\lambda x. x \in ball\ z\ d)\ (nhds\ z)$

by (*rule eventually_nhds_in_open simp_all*)

lemma *eventually_at_ball*: $d > 0 \implies eventually\ (\lambda t. t \in ball\ z\ d \wedge t \in A)\ (at\ z\ within\ A)$

unfolding *eventually_at* **by** (*intro exI[of _ d] (simp_all add: dist_commute)*)

lemma *eventually_at_ball'*: $d > 0 \implies eventually\ (\lambda t. t \in ball\ z\ d \wedge t \neq z \wedge t \in A)\ (at\ z\ within\ A)$

unfolding *eventually_at* **by** (*intro exI[of _ d] (simp_all add: dist_commute)*)

lemma *at_within_ball*: $e > 0 \implies dist\ x\ y < e \implies at\ y\ within\ ball\ x\ e = at\ y$

by (*subst at_within_open auto*)

lemma *atLeastAtMost_eq_cball*:

fixes $a\ b::real$

shows $\{a .. b\} = cball\ ((a + b)/2)\ ((b - a)/2)$

by (*auto simp: dist_real_def field_simps*)

lemma *cball_eq_atLeastAtMost*:

fixes $a\ b::real$

shows $cball\ a\ b = \{a - b .. a + b\}$

by (*auto simp: dist_real_def*)

lemma *greaterThanLessThan_eq_ball*:

```

fixes  $a\ b::\text{real}$ 
shows  $\{a <..< < b\} = \text{ball } ((a + b)/2) ((b - a)/2)$ 
by (auto simp: dist_real_def field_simps)

```

```

lemma ball_eq_greaterThanLessThan:
fixes  $a\ b::\text{real}$ 
shows  $\text{ball } a\ b = \{a - b <..< < a + b\}$ 
by (auto simp: dist_real_def)

```

```

lemma interior_ball [simp]:  $\text{interior } (\text{ball } x\ e) = \text{ball } x\ e$ 
by (simp add: interior_open)

```

```

lemma cball_eq_empty [simp]:  $\text{cball } x\ e = \{\} \longleftrightarrow e < 0$ 
by (smt (verit, best) Diff_empty ball_eq_empty cball_diff_sphere centre_in_ball centre_in_cball sphere_empty)

```

```

lemma cball_empty [simp]:  $e < 0 \implies \text{cball } x\ e = \{\}$ 
by simp

```

```

lemma cball_singleton:
fixes  $x :: 'a::\text{metric\_space}$ 
shows  $e = 0 \implies \text{cball } x\ e = \{x\}$ 
by simp

```

```

lemma ball_divide_subset:  $d \geq 1 \implies \text{ball } x\ (e/d) \subseteq \text{ball } x\ e$ 
by (metis ball_eq_empty div_by_1 frac_le linear_subset_ball zero_less_one)

```

```

lemma ball_divide_subset_numeral:  $\text{ball } x\ (e / \text{numeral } w) \subseteq \text{ball } x\ e$ 
using ball_divide_subset one_le_numeral by blast

```

```

lemma cball_divide_subset:  $d \geq 1 \implies \text{cball } x\ (e/d) \subseteq \text{cball } x\ e$ 
by (smt (verit, best) cball_empty_div_by_1 frac_le subset_cball zero_le_divide_iff)

```

```

lemma cball_divide_subset_numeral:  $\text{cball } x\ (e / \text{numeral } w) \subseteq \text{cball } x\ e$ 
using cball_divide_subset one_le_numeral by blast

```

```

lemma cball_scale:
assumes  $a \neq 0$ 
shows  $(\lambda x. a *_R x) ` \text{cball } c\ r = \text{cball } (a *_R c :: 'a :: \text{real\_normed\_vector}) (|a| *_R r)$ 

```

proof –

```

have 1:  $(\lambda x. a *_R x) ` \text{cball } c\ r \subseteq \text{cball } (a *_R c) (|a| *_R r)$  if  $a \neq 0$  for  $a\ r$  and  $c :: 'a$ 

```

proof *safe*

fix x

assume $x: x \in \text{cball } c\ r$

have $\text{dist } (a *_R c) (a *_R x) = \text{norm } (a *_R c - a *_R x)$

by (*auto simp: dist_norm*)

also have $a *_R c - a *_R x = a *_R (c - x)$

by (*simp add: algebra_simps*)
finally show $a *_R x \in cball (a *_R c) (|a| * r)$
using that x by (*auto simp: dist_norm*)
qed

have $cball (a *_R c) (|a| * r) = (\lambda x. a *_R x) \text{ ' } (\lambda x. inverse a *_R x) \text{ ' } cball (a *_R c) (|a| * r)$
unfolding image_image using assms by simp
also have $\dots \subseteq (\lambda x. a *_R x) \text{ ' } cball (inverse a *_R (a *_R c)) (|inverse a| * (|a| * r))$
using assms by (*intro image_mono 1*) *auto*
also have $\dots = (\lambda x. a *_R x) \text{ ' } cball c r$
using assms by (*simp add: algebra_simps*)
finally have $cball (a *_R c) (|a| * r) \subseteq (\lambda x. a *_R x) \text{ ' } cball c r$.
moreover from assms have $(\lambda x. a *_R x) \text{ ' } cball c r \subseteq cball (a *_R c) (|a| * r)$
by (*intro 1*) *auto*
ultimately show ?thesis by blast
qed

lemma ball_scale:

assumes $a \neq 0$

shows $(\lambda x. a *_R x) \text{ ' } ball c r = ball (a *_R c :: 'a :: real_normed_vector) (|a| * r)$

proof –

have 1: $(\lambda x. a *_R x) \text{ ' } ball c r \subseteq ball (a *_R c) (|a| * r)$ **if** $a \neq 0$ **for** a r **and** c $:: 'a$

proof safe

fix x

assume $x: x \in ball c r$

have $dist (a *_R c) (a *_R x) = norm (a *_R c - a *_R x)$

by (*auto simp: dist_norm*)

also have $a *_R c - a *_R x = a *_R (c - x)$

by (*simp add: algebra_simps*)

finally show $a *_R x \in ball (a *_R c) (|a| * r)$

using that x by (*auto simp: dist_norm*)

qed

have $ball (a *_R c) (|a| * r) = (\lambda x. a *_R x) \text{ ' } (\lambda x. inverse a *_R x) \text{ ' } ball (a *_R c) (|a| * r)$

unfolding image_image using assms by simp

also have $\dots \subseteq (\lambda x. a *_R x) \text{ ' } ball (inverse a *_R (a *_R c)) (|inverse a| * (|a| * r))$

using assms by (*intro image_mono 1*) *auto*

also have $\dots = (\lambda x. a *_R x) \text{ ' } ball c r$

using assms by (*simp add: algebra_simps*)

finally have $ball (a *_R c) (|a| * r) \subseteq (\lambda x. a *_R x) \text{ ' } ball c r$.

moreover from assms have $(\lambda x. a *_R x) \text{ ' } ball c r \subseteq ball (a *_R c) (|a| * r)$

by (*intro 1*) *auto*

ultimately show ?thesis by blast

qed

lemma frequently_atE:

```

  fixes x :: 'a :: metric_space
  assumes frequently P (at x within s)
  shows  $\exists f. \text{filterlim } f \text{ (at } x \text{ within } s) \text{ sequentially} \wedge (\forall n. P (f n))$ 
proof -
  have  $\exists y. y \in s \cap (\text{ball } x (1 / \text{real } (\text{Suc } n)) - \{x\}) \wedge P y$  for n
  proof -
    have  $\exists z \in s. z \neq x \wedge \text{dist } z x < (1 / \text{real } (\text{Suc } n)) \wedge P z$ 
    by (metis assms divide_pos_pos frequently_at of_nat_0_less_iff zero_less_Suc
zero_less_one)
    then show ?thesis
    by (auto simp: dist_commute conj_commute)
  qed
  then obtain f where f:  $\bigwedge n. f n \in s \cap (\text{ball } x (1 / \text{real } (\text{Suc } n)) - \{x\}) \wedge P (f n)$ 
  by metis
  have filterlim f (nhds x) sequentially
  unfolding tendsto_iff
proof clarify
  fix e :: real
  assume e:  $e > 0$ 
  then obtain n where n:  $\text{Suc } n > 1 / e$ 
  by (meson le_nat_floor lessI not_le)
  have  $\text{dist } (f k) x < e$  if  $k \geq n$  for k
  proof -
    have  $\text{dist } (f k) x < 1 / \text{real } (\text{Suc } k)$ 
    using f[of k] by (auto simp: dist_commute)
    also have  $\dots \leq 1 / \text{real } (\text{Suc } n)$ 
    using that by (intro divide_left_mono) auto
    also have  $\dots < e$ 
    using n e by (simp add: field_simps)
    finally show ?thesis .
  qed
  thus  $\forall_F k \text{ in sequentially. } \text{dist } (f k) x < e$ 
  unfolding eventually_at_top_linorder by blast
qed
moreover have  $f n \neq x$  for n
  using f[of n] by auto
ultimately have filterlim f (at x within s) sequentially
  using f by (auto simp: filterlim_at)
with f show ?thesis
  by blast
qed
```

4.2 Limit Points

lemma islimpt_approachable:

fixes $x :: 'a::\text{metric_space}$
shows $x \text{ islimpt } S \longleftrightarrow (\forall e>0. \exists x'\in S. x' \neq x \wedge \text{dist } x' x < e)$
unfolding $\text{islimpt_iff_eventually_eventually_at}$ **by** fast

lemma $\text{islimpt_approachable_le}: x \text{ islimpt } S \longleftrightarrow (\forall e>0. \exists x'\in S. x' \neq x \wedge \text{dist } x' x \leq e)$

for $x :: 'a::\text{metric_space}$
unfolding $\text{islimpt_approachable}$
using $\text{approachable_lt_le2}$ **[where** $f=\lambda y. \text{dist } y x$ **and** $P=\lambda y. y \notin S \vee y = x$
and $Q=\lambda x. \text{True}]$
by auto

lemma $\text{limpt_of_limpts}: x \text{ islimpt } \{y. y \text{ islimpt } S\} \implies x \text{ islimpt } S$

for $x :: 'a::\text{metric_space}$
by $(\text{metis } \text{islimpt_def } \text{islimpt_eq_acc_point } \text{mem_Collect_eq})$

lemma $\text{closed_limpts}: \text{closed } \{x::'a::\text{metric_space}. x \text{ islimpt } S\}$

using $\text{closed_limpt } \text{limpt_of_limpts}$ **by** blast

lemma $\text{limpt_of_closure}: x \text{ islimpt } \text{closure } S \longleftrightarrow x \text{ islimpt } S$

for $x :: 'a::\text{metric_space}$
by $(\text{auto } \text{simp: } \text{closure_def } \text{islimpt_Un } \text{dest: } \text{limpt_of_limpts})$

lemma $\text{islimpt_eq_infinite_ball}: x \text{ islimpt } S \longleftrightarrow (\forall e>0. \text{infinite}(S \cap \text{ball } x e))$

unfolding $\text{islimpt_eq_acc_point}$
by $(\text{metis } \text{open_ball } \text{Int_commute } \text{Int_mono } \text{finite_subset } \text{open_contains_ball_eq } \text{subset_eq})$

lemma $\text{islimpt_eq_infinite_cball}: x \text{ islimpt } S \longleftrightarrow (\forall e>0. \text{infinite}(S \cap \text{cball } x e))$

unfolding $\text{islimpt_eq_infinite_ball}$
by $(\text{metis } \text{open_ball } \text{ball_subset_cball } \text{centre_in_ball } \text{finite_Int } \text{inf.} \text{absorb_iff2 } \text{inf_assoc } \text{open_contains_cball_eq})$

4.3 Perfect Metric Spaces

lemma $\text{perfect_choose_dist}: 0 < r \implies \exists a. a \neq x \wedge \text{dist } a x < r$

for $x :: 'a::\{\text{perfect_space}, \text{metric_space}\}$
using islimpt_UNIV **[of** $x]$ **by** $(\text{simp } \text{add: } \text{islimpt_approachable})$

lemma cball_eq_sing :

fixes $x :: 'a::\{\text{metric_space}, \text{perfect_space}\}$

shows $\text{cball } x e = \{x\} \longleftrightarrow e = 0$

by $(\text{smt } (\text{verit}, \text{best}) \text{open_ball } \text{ball_eq_empty } \text{ball_subset_cball } \text{cball_empty } \text{cball_trivial})$

$\text{not_open_singleton } \text{subset_singleton_iff})$

4.4 Finite and discrete

lemma *finite_ball_include*:

fixes $a :: 'a::metric_space$

assumes *finite S*

shows $\exists e > 0. S \subseteq \text{ball } a \ e$

using *assms*

proof *induction*

case (*insert x S*)

then obtain $e0 > 0$ where $e0 > 0$ and $e0 : S \subseteq \text{ball } a \ e0$ by *auto*

define e where $e = \max e0 \ (2 * \text{dist } a \ x)$

have $e > 0$ unfolding *e_def* using $\langle e0 > 0 \rangle$ by *auto*

moreover have *insert x S* $\subseteq \text{ball } a \ e$

using $e0 \ \langle e > 0 \rangle$ unfolding *e_def* by *auto*

ultimately show *?case* by *auto*

qed (*auto intro: zero_less_one*)

lemma *finite_set_avoid*:

fixes $a :: 'a::metric_space$

assumes *finite S*

shows $\exists d > 0. \forall x \in S. x \neq a \longrightarrow d \leq \text{dist } a \ x$

using *assms*

proof *induction*

case (*insert x S*)

then obtain $d > 0$ where $d > 0$ and $d : \forall x \in S. x \neq a \longrightarrow d \leq \text{dist } a \ x$

by *blast*

show *?case*

by (*smt (verit, del_insts) dist_pos_lt insert.IH insert_iff*)

qed (*auto intro: zero_less_one*)

lemma *discrete_imp_closed*:

fixes $S :: 'a::metric_space \text{ set}$

assumes $e : 0 < e$

and $d : \forall x \in S. \forall y \in S. \text{dist } y \ x < e \longrightarrow y = x$

shows *closed S*

proof –

have *False* if $C : \bigwedge e. e > 0 \implies \exists x' \in S. x' \neq x \wedge \text{dist } x' \ x < e$ for x

proof –

from e have $e2 : e/2 > 0$ by *arith*

from $C[OF \ e2]$ obtain y where $y : y \in S \ y \neq x \ \text{dist } y \ x < e/2$

by *blast*

from $e2 \ y(2)$ have $mp : \min (e/2) (\text{dist } x \ y) > 0$

by *simp*

from $d \ y \ C[OF \ mp]$ show *?thesis*

by *metric*

qed

then show *?thesis*

by (*metis islimpt_approachable closed_limpt [where 'a='a]*)

qed

lemma *discrete_imp_not_islimpt*:
assumes $e: 0 < e$
and $d: \bigwedge x y. x \in S \implies y \in S \implies \text{dist } y \ x < e \implies y = x$
shows $\neg x \text{ islimpt } S$
proof
assume $x \text{ islimpt } S$
hence $x \text{ islimpt } S - \{x\}$
by (*meson islimpt_punctured*)
moreover from *assms* **have** $\text{closed } (S - \{x\})$
by (*intro discrete_imp_closed*) *auto*
ultimately show *False*
unfolding *closed_limpt* **by** *blast*
qed

4.5 Interior

lemma *mem_interior*: $x \in \text{interior } S \iff (\exists e > 0. \text{ball } x \ e \subseteq S)$
using *open_contains_ball_eq* [**where** $S = \text{interior } S$]
by (*simp add: open_subset_interior*)

lemma *mem_interior_cball*: $x \in \text{interior } S \iff (\exists e > 0. \text{cball } x \ e \subseteq S)$
by (*meson ball_subset_cball interior_subset mem_interior open_contains_cball open_interior subset_trans*)

lemma *ball_iff_cball*: $(\exists r > 0. \text{ball } x \ r \subseteq U) \iff (\exists r > 0. \text{cball } x \ r \subseteq U)$
by (*meson mem_interior mem_interior_cball*)

4.6 Frontier

lemma *frontier_straddle*:
fixes $a :: 'a :: \text{metric_space}$
shows $a \in \text{frontier } S \iff (\forall e > 0. (\exists x \in S. \text{dist } a \ x < e) \wedge (\exists x. x \notin S \wedge \text{dist } a \ x < e))$
unfolding *frontier_def closure_interior*
by (*auto simp: mem_interior subset_eq ball_def*)

4.7 Limits

proposition *Lim*: $(f \longrightarrow l) \text{ net} \iff \text{trivial_limit } \text{net} \vee (\forall e > 0. \text{eventually } (\lambda x. \text{dist } (f \ x) \ l < e) \text{ net})$
by (*auto simp: tendsto_iff trivial_limit_eq*)

Show that they yield usual definitions in the various cases.

proposition *Lim_within_le*: $(f \longrightarrow l)(\text{at } a \ \text{within } S) \iff (\forall e > 0. \exists d > 0. \forall x \in S. 0 < \text{dist } x \ a \wedge \text{dist } x \ a \leq d \longrightarrow \text{dist } (f \ x) \ l < e)$

by (auto simp: tendsto_iff eventually_at_le)

proposition *Lim_within*: $(f \longrightarrow l)$ (at a within S) \longleftrightarrow
 $(\forall e > 0. \exists d > 0. \forall x \in S. 0 < \text{dist } x \ a \wedge \text{dist } x \ a < d \longrightarrow \text{dist } (f \ x) \ l < e)$
 by (auto simp: tendsto_iff eventually_at)

corollary *Lim_withinI* [intro?]:
 assumes $\bigwedge e. e > 0 \implies \exists d > 0. \forall x \in S. 0 < \text{dist } x \ a \wedge \text{dist } x \ a < d \longrightarrow \text{dist } (f \ x) \ l \leq e$
 shows $(f \longrightarrow l)$ (at a within S)
 unfolding *Lim_within* by (smt (verit, ccfv_SIG) assms zero_less_dist_iff)

proposition *Lim_at*: $(f \longrightarrow l)$ (at a) \longleftrightarrow
 $(\forall e > 0. \exists d > 0. \forall x. 0 < \text{dist } x \ a \wedge \text{dist } x \ a < d \longrightarrow \text{dist } (f \ x) \ l < e)$
 by (auto simp: tendsto_iff eventually_at)

lemma *Lim_transform_within_set*:
 fixes $a :: 'a::\text{metric_space}$ and $l :: 'b::\text{metric_space}$
 shows $\llbracket (f \longrightarrow l)$ (at a within S); eventually $(\lambda x. x \in S \longleftrightarrow x \in T)$ (at a) \rrbracket
 $\implies (f \longrightarrow l)$ (at a within T)
 by (simp add: eventually_at_Lim_within) (smt (verit, best))

Another limit point characterization.

lemma *limpt_sequential_inj*:
 fixes $x :: 'a::\text{metric_space}$
 shows $x \text{ islimpt } S \longleftrightarrow$
 $(\exists f. (\forall n::\text{nat}. f \ n \in S - \{x\}) \wedge \text{inj } f \wedge (f \longrightarrow x) \text{ sequentially})$
 (is ?lhs = ?rhs)

proof

assume ?lhs
 then have $\forall e > 0. \exists x' \in S. x' \neq x \wedge \text{dist } x' \ x < e$
 by (force simp: islimpt_approachable)
 then obtain y where $y: \bigwedge e. e > 0 \implies y \in S \wedge y \neq x \wedge \text{dist } (y \ e) \ x < e$
 by metis
 define f where $f \equiv \text{rec_nat } (y \ 1) (\lambda n \ \text{fn}. y \ (\text{min } (\text{inverse}(2 \wedge (\text{Suc } n))) \ (\text{dist } \text{fn } \ x)))$
 have [simp]: $f \ 0 = y \ 1$
 and $f \ \text{Suc } n = y \ (\text{min } (\text{inverse}(2 \wedge (\text{Suc } n))) \ (\text{dist } (f \ n) \ x))$ for n
 by (simp_all add: f_def)
 have $f: f \ n \in S \wedge (f \ n \neq x) \wedge \text{dist } (f \ n) \ x < \text{inverse}(2 \wedge n)$ for n
 proof (induction n)
 case 0 show ?case
 by (simp add: y)
 next
 case (Suc n) then show ?case
 by (smt (verit, best) fSuc dist_pos_lt inverse_positive_iff_positive y zero_less_power)
 qed
 show ?rhs
 proof (intro exI conjI allI)

```

show  $\bigwedge n. f\ n \in S - \{x\}$ 
  using f by blast
have  $\text{dist } (f\ n)\ x < \text{dist } (f\ m)\ x$  if  $m < n$  for  $m\ n$ 
  using that
proof (induction n)
  case 0 then show ?case by simp
next
  case (Suc n)
  then consider  $m < n \mid m = n$  using less_Suc_eq by blast
  then show ?case
  proof cases
    assume  $m < n$ 
    have  $\text{dist } (f\ (\text{Suc } n))\ x = \text{dist } (y\ (\text{min } (\text{inverse } (2 \wedge (\text{Suc } n)))\ (\text{dist } (f\ n)\ x)))$ 
      x
      by (simp add: fSuc)
    also have  $\dots < \text{dist } (f\ n)\ x$ 
      by (metis dist_pos_lt f min.strict_order_iff min_less_iff_conj y)
    also have  $\dots < \text{dist } (f\ m)\ x$ 
      using Suc.IH  $\langle m < n \rangle$  by blast
    finally show ?thesis .
  next
    assume  $m = n$  then show ?case
      by (smt (verit, best) dist_pos_lt f fSuc y)
  qed
qed
then show inj f
  by (metis less_irrefl linorder_injI)
have  $\bigwedge e\ n. \llbracket 0 < e; \text{nat } \lceil 1 / e \rceil \leq n \rrbracket \implies \text{dist } (f\ n)\ x < e$ 
  apply (rule less_trans [OF f [THEN conjunct2, THEN conjunct2]])
  by (simp add: divide_simps order_le_less_trans)
then show  $f \longrightarrow x$ 
  using lim_sequentially by blast
qed
next
  assume ?rhs
  then show ?lhs
    by (fastforce simp add: islimpt_approachable lim_sequentially)
qed

lemma Lim_dist_ubound:
  assumes  $\neg(\text{trivial\_limit } \text{net})$ 
    and  $(f \longrightarrow l)$  net
    and eventually  $(\lambda x. \text{dist } a\ (f\ x) \leq e)$  net
  shows  $\text{dist } a\ l \leq e$ 
  using assms by (fast intro: tendsto_le tendsto_intros)

```

4.8 Continuity

Derive the epsilon-delta forms, which we often use as "definitions"

proposition *continuous_within_eps_delta*:

continuous (at x within s) f \longleftrightarrow $(\forall e > 0. \exists d > 0. \forall x' \in s. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) < e)$

unfolding *continuous_within* and *Lim_within* by *fastforce*

corollary *continuous_at_eps_delta*:

continuous (at x) f \longleftrightarrow $(\forall e > 0. \exists d > 0. \forall x'. \text{dist } x' x < d \longrightarrow \text{dist } (f x') (f x) < e)$

using *continuous_within_eps_delta* [of *x UNIV f*] by *simp*

lemma *continuous_at_right_real_increasing*:

fixes *f* :: *real* \Rightarrow *real*

assumes *nondecF*: $\bigwedge x y. x \leq y \Longrightarrow f x \leq f y$

shows *continuous (at_right a) f* \longleftrightarrow $(\forall e > 0. \exists d > 0. f (a + d) - f a < e)$

apply (*simp add: greaterThan_def dist_real_def continuous_within Lim_within_le*)

apply (*intro all_cong ex_cong*)

by (*smt (verit, best) nondecF*)

lemma *continuous_at_left_real_increasing*:

assumes *nondecF*: $\bigwedge x y. x \leq y \Longrightarrow f x \leq ((f y) :: \text{real})$

shows (*continuous (at_left (a :: real)) f*) \longleftrightarrow $(\forall e > 0. \exists \text{delta} > 0. f a - f (a - \text{delta}) < e)$

apply (*simp add: lessThan_def dist_real_def continuous_within Lim_within_le*)

apply (*intro all_cong ex_cong*)

by (*smt (verit) nondecF*)

Versions in terms of open balls.

lemma *continuous_within_ball*:

continuous (at x within S) f \longleftrightarrow

$(\forall e > 0. \exists d > 0. f' (\text{ball } x d \cap S) \subseteq \text{ball } (f x) e)$

(*is ?lhs = ?rhs*)

proof

assume *?lhs*

{

fix *e* :: *real*

assume *e* > 0

then obtain *d* **where** *d* > 0 **and** *d*: $\forall y \in S. 0 < \text{dist } y x \wedge \text{dist } y x < d \longrightarrow \text{dist } (f y) (f x) < e$

using $\langle ?lhs \rangle$ [*unfolded continuous_within Lim_within*] **by** *auto*

{ **fix** *y*

assume $y \in f' (\text{ball } x d \cap S)$ **then have** $y \in \text{ball } (f x) e$

using $\langle e > 0 \rangle$ **by** (*auto simp: dist_commute*)

}

then have $\exists d > 0. f' (\text{ball } x d \cap S) \subseteq \text{ball } (f x) e$

using $\langle d > 0 \rangle$ **by** *blast*

}

```

then show ?rhs by auto
next
  assume ?rhs
  then show ?lhs
    apply (simp add: continuous_within Lim_within ball_def subset_eq)
    by (metis (mono_tags, lifting) Int_iff dist_commute mem_Collect_eq)
qed

```

```

lemma continuous_at_ball:
  continuous (at x) f  $\longleftrightarrow$   $(\forall e > 0. \exists d > 0. f \text{ ` } (ball\ x\ d) \subseteq ball\ (f\ x)\ e)$ 
  apply (simp add: continuous_at Lim_at subset_eq Ball_def Bex_def image_iff)
  by (smt (verit, ccfv_threshold) dist_commute dist_self)

```

Define setwise continuity in terms of limits within the set.

```

lemma continuous_on_iff:
  continuous_on s f  $\longleftrightarrow$ 
   $(\forall x \in s. \forall e > 0. \exists d > 0. \forall x' \in s. dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e)$ 
  unfolding continuous_on_def Lim_within
  by (metis dist_pos_lt dist_self)

```

```

lemma continuous_within_E:
  assumes continuous (at x within S) f e > 0
  obtains d where d > 0  $\wedge x'. \llbracket x' \in S; dist\ x'\ x \leq d \rrbracket \implies dist\ (f\ x')\ (f\ x) < e$ 
  using assms unfolding continuous_within_eps_delta
  by (metis dense_order_le_less_trans)

```

```

lemma continuous_onI [intro?]:
  assumes  $\wedge x e. \llbracket e > 0; x \in S \rrbracket \implies \exists d > 0. \forall x' \in S. dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) \leq e$ 
  shows continuous_on S f
  apply (simp add: continuous_on_iff, clarify)
  apply (rule ex_forward [OF assms [OF half_gt_zero]], auto)
  done

```

Some simple consequential lemmas.

```

lemma continuous_onE:
  assumes continuous_on s f x ∈ s e > 0
  obtains d where d > 0  $\wedge x'. \llbracket x' \in s; dist\ x'\ x \leq d \rrbracket \implies dist\ (f\ x')\ (f\ x) < e$ 
  using assms
  unfolding continuous_on_iff by (metis dense_order_le_less_trans)

```

The usual transformation theorems.

```

lemma continuous_transform_within:
  fixes f g :: 'a::metric_space  $\Rightarrow$  'b::topological_space
  assumes continuous (at x within s) f
  and 0 < d
  and x ∈ s
  and  $\wedge x'. \llbracket x' \in s; dist\ x'\ x < d \rrbracket \implies f\ x' = g\ x'$ 
  shows continuous (at x within s) g

```

using *assms*
 unfolding *continuous_within* by (force intro: *Lim_transform_within*)

4.9 Closure and Limit Characterization

lemma *closure_approachable*:
 fixes $S :: 'a::metric_space\ set$
 shows $x \in closure\ S \longleftrightarrow (\forall e>0. \exists y \in S. dist\ y\ x < e)$
 using *dist_self* by (force simp: *closure_def islimpt_approachable*)

lemma *closure_approachable_le*:
 fixes $S :: 'a::metric_space\ set$
 shows $x \in closure\ S \longleftrightarrow (\forall e>0. \exists y \in S. dist\ y\ x \leq e)$
 unfolding *closure_approachable*
 using *dense* by force

lemma *closure_approachableD*:
 assumes $x \in closure\ S\ e>0$
 shows $\exists y \in S. dist\ x\ y < e$
 using *assms* unfolding *closure_approachable* by (auto simp: *dist_commute*)

lemma *closed_approachable*:
 fixes $S :: 'a::metric_space\ set$
 shows $closed\ S \implies (\forall e>0. \exists y \in S. dist\ y\ x < e) \longleftrightarrow x \in S$
 by (metis *closure_closed closure_approachable*)

lemma *closure_contains_Inf*:
 fixes $S :: real\ set$
 assumes $S \neq \{\}$ *bdd_below* S
 shows $Inf\ S \in closure\ S$
 proof –
 have *: $\forall x \in S. Inf\ S \leq x$
 using *cInf_lower*[of _ S] *assms* by *metis*
 { fix $e :: real$
 assume $e > 0$
 then have $Inf\ S < Inf\ S + e$ by *simp*
 with *assms* obtain x where $x \in S\ x < Inf\ S + e$
 using *cInf_lessD* by *blast*
 with * have $\exists x \in S. dist\ x\ (Inf\ S) < e$
 using *dist_real_def* by force
 }
 then show *?thesis* unfolding *closure_approachable* by *auto*
 qed

lemma *closure_contains_Sup*:
 fixes $S :: real\ set$
 assumes $S \neq \{\}$ *bdd_above* S
 shows $Sup\ S \in closure\ S$
 proof –

```

have *:  $\forall x \in S. x \leq \text{Sup } S$ 
  using cSup_upper[of _ S] assms by metis
{
  fix e :: real
  assume e > 0
  then have  $\text{Sup } S - e < \text{Sup } S$  by simp
  with assms obtain x where  $x \in S$   $\text{Sup } S - e < x$ 
    using less_cSupE by blast
  with * have  $\exists x \in S. \text{dist } x (\text{Sup } S) < e$ 
    using dist_real_def by force
}
then show ?thesis unfolding closure_approachable by auto
qed

lemma not_trivial_limit_within_ball:
   $\neg \text{trivial\_limit (at } x \text{ within } S) \longleftrightarrow (\forall e > 0. S \cap \text{ball } x e - \{x\} \neq \{\})$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?rhs if ?lhs
  proof -
    { fix e :: real
      assume e > 0
      then obtain y where  $y \in S - \{x\}$  and  $\text{dist } y x < e$ 
        using  $\langle ?lhs \rangle$  not_trivial_limit_within[of x S] closure_approachable[of x S
        -  $\{x\}$ ]
        by auto
      then have  $y \in S \cap \text{ball } x e - \{x\}$ 
        unfolding ball_def by (simp add: dist_commute)
      then have  $S \cap \text{ball } x e - \{x\} \neq \{\}$  by blast
    }
    then show ?thesis by auto
  qed
  show ?lhs if ?rhs
  proof -
    { fix e :: real
      assume e > 0
      then obtain y where  $y \in S \cap \text{ball } x e - \{x\}$ 
        using  $\langle ?rhs \rangle$  by blast
      then have  $y \in S - \{x\}$  and  $\text{dist } y x < e$ 
        unfolding ball_def by (simp_all add: dist_commute)
      then have  $\exists y \in S - \{x\}. \text{dist } y x < e$ 
        by auto
    }
    then show ?thesis
      using not_trivial_limit_within[of x S] closure_approachable[of x S -  $\{x\}$ ]
      by auto
  qed
qed

```


4.10 Boundedness

definition (in *metric_space*) *bounded* :: 'a set \Rightarrow bool
 where *bounded* $S \longleftrightarrow (\exists x e. \forall y \in S. \text{dist } x \ y \leq e)$

lemma *bounded_subset_cball*: *bounded* $S \longleftrightarrow (\exists e x. S \subseteq \text{cball } x \ e \wedge 0 \leq e)$
unfolding *bounded_def subset_eq* **by** *auto* (*meson order_trans zero_le_dist*)

lemma *bounded_any_center*: *bounded* $S \longleftrightarrow (\exists e. \forall y \in S. \text{dist } a \ y \leq e)$
unfolding *bounded_def*
by *auto* (*metis add.commute add_le_cancel_right dist_commute dist_triangle_le*)

lemma *bounded_iff*: *bounded* $S \longleftrightarrow (\exists a. \forall x \in S. \text{norm } x \leq a)$
unfolding *bounded_any_center* [**where** $a=0$]
by (*simp add: dist_norm*)

lemma *bdd_above_norm*: *bdd_above* (*norm* ' X) \longleftrightarrow *bounded* X
by (*simp add: bounded_iff bdd_above_def*)

lemma *bounded_norm_comp*: *bounded* $((\lambda x. \text{norm } (f \ x)) \ ' S) = \text{bounded } (f \ ' S)$
by (*simp add: bounded_iff*)

lemma *boundedI*:
assumes $\bigwedge x. x \in S \Longrightarrow \text{norm } x \leq B$
shows *bounded* S
using *assms bounded_iff* **by** *blast*

lemma *bounded_empty* [*simp*]: *bounded* $\{\}$
by (*simp add: bounded_def*)

lemma *bounded_subset*: *bounded* $T \Longrightarrow S \subseteq T \Longrightarrow \text{bounded } S$
by (*metis bounded_def subset_eq*)

lemma *bounded_interior*[*intro*]: *bounded* $S \Longrightarrow \text{bounded}(\text{interior } S)$
by (*metis bounded_subset interior_subset*)

lemma *bounded_closure*[*intro*]:

assumes *bounded* S
shows *bounded* (*closure* S)

proof –

from *assms* **obtain** x **and** a **where** $a: \forall y \in S. \text{dist } x \ y \leq a$
unfolding *bounded_def* **by** *auto*

{ **fix** y

assume $y \in \text{closure } S$

then obtain f **where** $f: \forall n. f \ n \in S \ (f \longrightarrow y)$ *sequentially*

unfolding *closure_sequential* **by** *auto*

have $\forall n. f \ n \in S \longrightarrow \text{dist } x \ (f \ n) \leq a$ **using** a **by** *simp*

then have *eventually* $(\lambda n. \text{dist } x \ (f \ n) \leq a)$ *sequentially*

by (*simp add: f(1)*)

```

    then have  $\text{dist } x \ y \leq a$ 
      using  $\text{Lim\_dist\_ubound } f(2) \ \text{trivial\_limit\_sequentially}$  by blast
    }
    then show ?thesis
      unfolding  $\text{bounded\_def}$  by auto
  qed

```

```

lemma  $\text{bounded\_closure\_image}$ :  $\text{bounded } (f \text{ ' closure } S) \implies \text{bounded } (f \text{ ' } S)$ 
  by ( $\text{simp add: bounded\_subset closure\_subset image\_mono}$ )

```

```

lemma  $\text{bounded\_cball}$ [ $\text{simp,intro}$ ]:  $\text{bounded } (\text{cball } x \ e)$ 
  unfolding  $\text{bounded\_def}$  using  $\text{mem\_cball}$  by blast

```

```

lemma  $\text{bounded\_ball}$ [ $\text{simp,intro}$ ]:  $\text{bounded } (\text{ball } x \ e)$ 
  by ( $\text{metis ball\_subset\_cball bounded\_cball bounded\_subset}$ )

```

```

lemma  $\text{bounded\_Un}$ [ $\text{simp}$ ]:  $\text{bounded } (S \cup T) \iff \text{bounded } S \wedge \text{bounded } T$ 
  by ( $\text{auto simp: bounded\_def}$ ) ( $\text{metis Un\_iff bounded\_any\_center le\_max\_iff\_disj}$ )

```

```

lemma  $\text{bounded\_Union}$ [ $\text{intro}$ ]:  $\text{finite } F \implies \forall S \in F. \text{bounded } S \implies \text{bounded } (\bigcup F)$ 
  by ( $\text{induct rule: finite\_induct[of } F]$ ) auto

```

```

lemma  $\text{bounded\_UN}$  [ $\text{intro}$ ]:  $\text{finite } A \implies \forall x \in A. \text{bounded } (B \ x) \implies \text{bounded } (\bigcup_{x \in A} B \ x)$ 
  by auto

```

```

lemma  $\text{bounded\_insert}$  [ $\text{simp}$ ]:  $\text{bounded } (\text{insert } x \ S) \iff \text{bounded } S$ 

```

```

proof –
  have  $\forall y \in \{x\}. \text{dist } x \ y \leq 0$ 
    by  $\text{simp}$ 
  then have  $\text{bounded } \{x\}$ 
    unfolding  $\text{bounded\_def}$  by fast
  then show ?thesis
    by ( $\text{metis insert\_is\_Un bounded\_Un}$ )
qed

```

```

lemma  $\text{bounded\_subset\_ballI}$ :  $S \subseteq \text{ball } x \ r \implies \text{bounded } S$ 
  by ( $\text{meson bounded\_ball bounded\_subset}$ )

```

```

lemma  $\text{bounded\_subset\_ballD}$ :
  assumes  $\text{bounded } S$  shows  $\exists r. 0 < r \wedge S \subseteq \text{ball } x \ r$ 

```

```

proof –
  obtain  $e::\text{real}$  and  $y$  where  $S \subseteq \text{cball } y \ e$  and  $0 \leq e$ 
    using  $\text{assms}$  by ( $\text{auto simp: bounded\_subset\_cball}$ )
  then show ?thesis
    by ( $\text{intro exI[where } x = \text{dist } x \ y + e + 1]$ )  $\text{metric}$ 
qed

```

```

lemma  $\text{finite\_imp\_bounded}$  [ $\text{intro}$ ]:  $\text{finite } S \implies \text{bounded } S$ 

```

by (induct set: finite) simp_all

lemma *bounded_Int[intro]*: $\text{bounded } S \vee \text{bounded } T \implies \text{bounded } (S \cap T)$
 by (metis Int_lower1 Int_lower2 bounded_subset)

lemma *bounded_diff[intro]*: $\text{bounded } S \implies \text{bounded } (S - T)$
 by (metis Diff_subset bounded_subset)

lemma *bounded_dist_comp*:

assumes $\text{bounded } (f \text{ ' } S) \text{ bounded } (g \text{ ' } S)$

shows $\text{bounded } ((\lambda x. \text{dist } (f x) (g x)) \text{ ' } S)$

proof –

from *assms* obtain $M1 M2$ where $*: \text{dist } (f x) \text{ undefined} \leq M1 \text{ dist } \text{undefined}$
 $(g x) \leq M2$ if $x \in S$ for x

by (auto simp: bounded_any_center[of _ undefined] dist_commute)

have $\text{dist } (f x) (g x) \leq M1 + M2$ if $x \in S$ for x

using $*[OF \text{ that}]$

by *metric*

then show *?thesis*

by (auto intro!: boundedI)

qed

lemma *bounded_Times*:

assumes $\text{bounded } s \text{ bounded } t$

shows $\text{bounded } (s \times t)$

proof –

obtain $x y a b$ where $\forall z \in s. \text{dist } x z \leq a \ \forall z \in t. \text{dist } y z \leq b$

using *assms* [unfolded bounded_def] by *auto*

then have $\forall z \in s \times t. \text{dist } (x, y) z \leq \text{sqrt } (a^2 + b^2)$

by (auto simp: dist_Pair_Pair real_sqrt_le_mono add_mono power_mono)

then show *?thesis* unfolding bounded_any_center [where $a=(x, y)$] by *auto*

qed

4.11 Compactness

lemma *compact_imp_bounded*:

assumes *compact* U

shows *bounded* U

proof –

have *compact* $U \ \forall x \in U. \text{open } (\text{ball } x \ 1) \ U \subseteq (\bigcup x \in U. \text{ball } x \ 1)$

using *assms* by *auto*

then obtain D where $D: D \subseteq U \ \text{finite } D \ U \subseteq (\bigcup x \in D. \text{ball } x \ 1)$

by (*metis compactE_image*)

from $\langle \text{finite } D \rangle$ have *bounded* $(\bigcup x \in D. \text{ball } x \ 1)$

by (*simp add: bounded_UN*)

then show *bounded* U using $\langle U \subseteq (\bigcup x \in D. \text{ball } x \ 1) \rangle$

by (*rule bounded_subset*)

qed

lemma *continuous_on_compact_bound*:
assumes *compact A continuous_on A f*
obtains *B* **where** $B \geq 0 \wedge x. x \in A \implies \text{norm } (f x) \leq B$
proof –
have *compact (f ' A)* **by** (*metis assms compact_continuous_image*)
then obtain *B* **where** $\forall x \in A. \text{norm } (f x) \leq B$
by (*auto dest!: compact_imp_bounded simp: bounded_iff*)
hence $\max B 0 \geq 0$ **and** $\forall x \in A. \text{norm } (f x) \leq \max B 0$ **by** *auto*
thus *?thesis* **using** *that* **by** *blast*
qed

lemma *closure_Int_ball_not_empty*:
assumes $S \subseteq \text{closure } T \ x \in S \ r > 0$
shows $T \cap \text{ball } x \ r \neq \{\}$
using *assms centre_in_ball closure_iff_nhds_not_empty* **by** *blast*

lemma *compact_sup_maxdistance*:
fixes $S :: 'a::\text{metric_space}$ *set*
assumes *compact S*
and $S \neq \{\}$
shows $\exists x \in S. \exists y \in S. \forall u \in S. \forall v \in S. \text{dist } u \ v \leq \text{dist } x \ y$
proof –
have *compact (S × S)*
using $\langle \text{compact } S \rangle$ **by** (*intro compact_Times*)
moreover **have** $S \times S \neq \{\}$
using $\langle S \neq \{\} \rangle$ **by** *auto*
moreover **have** *continuous_on (S × S) ($\lambda x. \text{dist } (\text{fst } x) (\text{snd } x)$)*
by (*intro continuous_at_imp_continuous_on ballI continuous_intros*)
ultimately show *?thesis*
using *continuous_attains_sup[of S × S $\lambda x. \text{dist } (\text{fst } x) (\text{snd } x)$]* **by** *auto*
qed

If A is a compact subset of an open set B in a metric space, then there exists an $\varepsilon > 0$ such that the Minkowski sum of A with an open ball of radius ε is also a subset of B .

lemma *compact_subset_open_imp_ball_epsilon_subset*:
assumes *compact A open B A ⊆ B*
obtains *e* **where** $e > 0 \ (\bigcup_{x \in A} \text{ball } x \ e) \subseteq B$
proof –
have $\forall x \in A. \exists e. e > 0 \wedge \text{ball } x \ e \subseteq B$
using *assms unfolding open_contains_ball* **by** *blast*
then obtain *e* **where** $e: \bigwedge x. x \in A \implies e \ x > 0 \ \bigwedge x. x \in A \implies \text{ball } x \ (e \ x) \subseteq B$
by *metis*
define *C* **where** $C = e \ ' \ A$
obtain *X* **where** $X: X \subseteq A \ \text{finite } X \ A \subseteq (\bigcup_{c \in X} \text{ball } c \ (e \ c / 2))$
using *assms(1)*
proof (*rule compactE_image*)
show *open (ball x (e x / 2))* **if** $x \in A$ **for** *x*

```

    by simp
  show  $A \subseteq (\bigcup c \in A. \text{ball } c (e / 2))$ 
    using  $e$  by auto
qed auto

define  $e'$  where  $e' = \text{Min } (\text{insert } 1 ((\lambda x. e / 2) ` X))$ 
have  $e' > 0$ 
  unfolding  $e'_\text{def}$  using  $e \in X$  by (subst  $\text{Min\_gr\_iff}$ ) auto
have  $e': e' \leq e / 2$  if  $x \in X$  for  $x$ 
  using  $\text{that } X$  unfolding  $e'_\text{def}$  by (intro  $\text{Min.coboundedI}$ ) auto

show ?thesis
proof
  show  $e' > 0$ 
    by fact
next
  show  $(\bigcup x \in A. \text{ball } x e') \subseteq B$ 
  proof clarify
    fix  $x y$  assume  $xy: x \in A \ y \in \text{ball } x e'$ 
    from  $xy(1) X$  obtain  $z$  where  $z: z \in X \ x \in \text{ball } z (e / 2)$ 
      by auto
    have  $\text{dist } y z \leq \text{dist } x y + \text{dist } z x$ 
      by (metis  $\text{dist\_commute dist\_triangle}$ )
    also have  $\text{dist } z x < e / 2$ 
      using  $xy z$  by auto
    also have  $\text{dist } x y < e'$ 
      using  $xy$  by auto
    also have  $\dots \leq e / 2$ 
      using  $z$  by (intro  $e'$ ) auto
    finally have  $y \in \text{ball } z (e / 2)$ 
      by (simp add:  $\text{dist\_commute}$ )
    also have  $\dots \subseteq B$ 
      using  $z X$  by (intro  $e$ ) auto
    finally show  $y \in B$  .
  qed
qed
qed

lemma compact_subset_open_imp_cball_epsilon_subset:
  assumes compact  $A$  open  $B$   $A \subseteq B$ 
  obtains  $e > 0$   $(\bigcup x \in A. \text{cball } x e) \subseteq B$ 
proof -
  obtain  $e$  where  $e > 0$  and  $e: (\bigcup x \in A. \text{ball } x e) \subseteq B$ 
    using compact_subset_open_imp_ball_epsilon_subset [ $OF$   $\text{assms}$ ] by blast
  then have  $(\bigcup x \in A. \text{cball } x (e / 2)) \subseteq (\bigcup x \in A. \text{ball } x e)$ 
    by auto
  with  $\langle 0 < e \rangle$  that show ?thesis
    by (metis  $e$  half_gt_zero_iff order_trans)
qed

```

Totally bounded

proposition *seq_compact_imp_totally_bounded*:

assumes *seq_compact S*

shows $\forall e > 0. \exists k. \text{finite } k \wedge k \subseteq S \wedge S \subseteq (\bigcup_{x \in k}. \text{ball } x \ e)$

proof –

{ **fix** $e :: \text{real}$ **assume** $e > 0$ **assume** *: $\bigwedge k. \text{finite } k \implies k \subseteq S \implies \neg S \subseteq (\bigcup_{x \in k}. \text{ball } x \ e)$

let $?Q = \lambda x \ n \ r. r \in S \wedge (\forall m < (n :: \text{nat}). \neg (\text{dist } (x \ m) \ r < e))$

have $\exists x. \forall n :: \text{nat}. ?Q \ x \ n \ (x \ n)$

proof (*rule dependent_wellorder_choice*)

fix $n \ x$ **assume** $\bigwedge y. y < n \implies ?Q \ x \ y \ (x \ y)$

then have $\neg S \subseteq (\bigcup_{x \in x' \ \{0..<n\}}. \text{ball } x \ e)$

using **[of x ' {0 ..<n}]* **by** (*auto simp: subset_eq*)

then obtain z **where** $z :: z \in S \ z \notin (\bigcup_{x \in x' \ \{0..<n\}}. \text{ball } x \ e)$

unfolding *subset_eq* **by** *auto*

show $\exists r. ?Q \ x \ n \ r$

using z **by** *auto*

qed *simp*

then obtain x **where** $\forall n :: \text{nat}. x \ n \in S$ **and** $x : \bigwedge n \ m. m < n \implies \neg (\text{dist } (x \ m) \ (x \ n) < e)$

by *blast*

then obtain $l \ r$ **where** $l \in S$ **and** $r : \text{strict_mono } r$ **and** $((x \circ r) \longrightarrow l)$ *sequentially*

using *assms* **by** (*metis seq_compact_def*)

then have *Cauchy* $(x \circ r)$

using *LIMSEQ_imp_Cauchy* **by** *auto*

then obtain $N :: \text{nat}$ **where** $\bigwedge m \ n. N \leq m \implies N \leq n \implies \text{dist } ((x \circ r) \ m) \ ((x \circ r) \ n) < e$

unfolding *Cauchy_def* **using** $\langle e > 0 \rangle$ **by** *blast*

then have *False*

using x *[of r N r (N+1)]* r **by** (*auto simp: strict_mono_def*) }

then show *?thesis*

by *metis*

qed

Heine-Borel theorem

proposition *seq_compact_imp_Heine_Borel*:

fixes $S :: 'a :: \text{metric_space}$ *set*

assumes *seq_compact S*

shows *compact S*

proof –

from *seq_compact_imp_totally_bounded* *[OF <seq_compact S>]*

obtain f **where** $f : \forall e > 0. \text{finite } (f \ e) \wedge f \ e \subseteq S \wedge S \subseteq (\bigcup_{x \in f \ e}. \text{ball } x \ e)$

unfolding *choice_iff'* ..

define K **where** $K = (\lambda(x, r). \text{ball } x \ r) \ \langle (\bigcup_{e \in \mathbb{Q} \cap \{0 <.. \}}. f \ e) \times \mathbb{Q} \rangle$

have *countably_compact S*

using $\langle \text{seq_compact } S \rangle$ **by** (*rule seq_compact_imp_countably_compact*)

then show *compact S*

```

proof (rule countably_compact_imp_compact)
  show countable K
    unfolding K_def using f
    by (auto intro: countable_finite countable_subset countable_rat
      intro!: countable_image countable_SIGMA countable_UN)
  show  $\forall b \in K. \text{open } b$  by (auto simp: K_def)
next
  fix T x
  assume T: open T  $x \in T$  and x:  $x \in S$ 
  from openE[OF T] obtain e where  $0 < e$  ball x e  $\subseteq T$ 
    by auto
  then have  $0 < e/2$  ball x (e/2)  $\subseteq T$ 
    by auto
  from Rats_dense_in_real[OF  $\langle 0 < e/2 \rangle$ ] obtain r where  $r \in \mathbb{Q}$   $0 < r < e/2$ 
    by auto
  from f[rule_format, of r]  $\langle 0 < r \rangle \langle x \in S \rangle$  obtain k where  $k \in f r$   $x \in \text{ball } k r$ 
    by auto
  from  $\langle r \in \mathbb{Q} \rangle \langle 0 < r \rangle \langle k \in f r \rangle$  have ball k r  $\in K$ 
    by (auto simp: K_def)
  then show  $\exists b \in K. x \in b \wedge b \cap S \subseteq T$ 
proof (rule bexI[rotated], safe)
  fix y
  assume  $y \in \text{ball } k r$ 
  with  $\langle r < e/2 \rangle \langle x \in \text{ball } k r \rangle$  have dist x y  $< e$ 
    by (intro dist_triangle_half_r [of k _ e]) (auto simp: dist_commute)
  with  $\langle \text{ball } x e \subseteq T \rangle$  show  $y \in T$ 
    by auto
next
  show  $x \in \text{ball } k r$  by fact
qed
qed
qed

```

proposition compact_eq_seq_compact_metric:
 compact (S :: 'a::metric_space set) \longleftrightarrow seq_compact S
using compact_imp_seq_compact seq_compact_imp_Heine_Borel **by** blast

proposition compact_def: — this is the definition of compactness in HOL Light
 compact (S :: 'a::metric_space set) \longleftrightarrow
 ($\forall f. (\forall n. f n \in S) \longrightarrow (\exists l \in S. \exists r::\text{nat} \Rightarrow \text{nat. strict_mono } r \wedge (f \circ r) \longrightarrow l)$)
unfolding compact_eq_seq_compact_metric seq_compact_def **by** auto

Complete the chain of compactness variants

proposition compact_eq_Bolzano_Weierstrass:
 fixes S :: 'a::metric_space set

shows $compact\ S \longleftrightarrow (\forall T. infinite\ T \wedge T \subseteq S \longrightarrow (\exists x \in S. x\ islimpt\ T))$
 by (meson Bolzano_Weierstrass_imp_seq_compact Heine_Borel_imp_Bolzano_Weierstrass
 seq_compact_imp_Heine_Borel)

proposition Bolzano_Weierstrass_imp_bounded:

$(\bigwedge T. \llbracket infinite\ T; T \subseteq S \rrbracket \implies (\exists x \in S. x\ islimpt\ T)) \implies bounded\ S$
 using compact_imp_bounded unfolding compact_eq_Bolzano_Weierstrass by
 metis

4.12 Banach fixed point theorem

theorem banach_fix:— TODO: rename to Banach_fix

assumes $s: complete\ s\ s \neq \{\}$

and $c: 0 \leq c < 1$

and $f: f' s \subseteq s$

and lipschitz: $\forall x \in s. \forall y \in s. dist\ (f\ x)\ (f\ y) \leq c * dist\ x\ y$

shows $\exists! x \in s. f\ x = x$

proof —

from c have $1 - c > 0$ by simp

from $s(2)$ obtain $z0$ where $z0: z0 \in s$ by blast

define z where $z\ n = (f \hat{\ } n)\ z0$ for n

with $f\ z0$ have $z_in_s: z\ n \in s$ for $n :: nat$

by (induct n) auto

define d where $d = dist\ (z\ 0)\ (z\ 1)$

have $fzn: f\ (z\ n) = z\ (Suc\ n)$ for n

by (simp add: z_def)

have $cf_z: dist\ (z\ n)\ (z\ (Suc\ n)) \leq (c \hat{\ } n) * d$ for $n :: nat$

proof (induct n)

case 0

then show ?case

by (simp add: d_def)

next

case (Suc m)

with $\langle 0 \leq c \rangle$ have $c * dist\ (z\ m)\ (z\ (Suc\ m)) \leq c \hat{\ } Suc\ m * d$

using mult_left_mono[of $dist\ (z\ m)\ (z\ (Suc\ m))\ c \hat{\ } m * d\ c$] by simp

then show ?case

using lipschitz[THEN bspec[where $x=z\ m$], OF z_in_s , THEN bspec[where
 $x=z\ (Suc\ m)$], OF z_in_s]

by (simp add: $fzn\ mult_le_cancel_left$)

qed

have $cf_z2: (1 - c) * dist\ (z\ m)\ (z\ (m + n)) \leq (c \hat{\ } m) * d * (1 - c \hat{\ } n)$ for
 $n\ m :: nat$

proof (induct n)

case 0

show ?case by simp

next


```

    case (Suc k)
    from c have  $(1 - c) * \text{dist } (z \ m) \ (z \ (m + \text{Suc } k)) \leq$ 
       $(1 - c) * (\text{dist } (z \ m) \ (z \ (m + k)) + \text{dist } (z \ (m + k)) \ (z \ (\text{Suc } (m + k))))$ 
      by (simp add: dist_triangle)
    also from c cf_z[of m + k] have  $\dots \leq (1 - c) * (\text{dist } (z \ m) \ (z \ (m + k)) +$ 
 $c \wedge^{m+k} * d)$ 
      by simp
    also from Suc have  $\dots \leq c \wedge^m * d * (1 - c \wedge^k) + (1 - c) * c \wedge^{m+k}$ 
 $* d$ 
      by (simp add: field_simps)
    also have  $\dots = (c \wedge^m) * (d * (1 - c \wedge^k) + (1 - c) * c \wedge^k * d)$ 
      by (simp add: power_add field_simps)
    also from c have  $\dots \leq (c \wedge^m) * d * (1 - c \wedge^{\text{Suc } k})$ 
      by (simp add: field_simps)
    finally show ?case by simp
qed

have  $\exists N. \forall m \ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (z \ m) \ (z \ n) < e$  if  $e > 0$  for e
proof (cases d = 0)
  case True
  from  $\langle 1 - c > 0 \rangle$  have  $(1 - c) * x \leq 0 \iff x \leq 0$  for x
    by (simp add: mult_le_0_iff)
  with c cf_z2[of 0] True have  $z \ n = z \ 0$  for n
    by (simp add: z_def)
  with  $\langle e > 0 \rangle$  show ?thesis by simp
next
  case False
  with zero_le_dist[of z 0 z 1] have  $d > 0$ 
    by (metis d_def less_le)
  with  $\langle 1 - c > 0 \rangle \langle e > 0 \rangle$  have  $0 < e * (1 - c) / d$ 
    by simp
  with c obtain N where  $N: c \wedge^N < e * (1 - c) / d$ 
    using real_arch_pow_inv[of e * (1 - c) / d c] by auto
  have  $*$ :  $\text{dist } (z \ m) \ (z \ n) < e$  if  $m > n$  and as:  $m \geq N \ n \geq N$  for m n :: nat
  proof -
    from c  $\langle n \geq N \rangle$  have  $*$ :  $c \wedge^n \leq c \wedge^N$ 
      using power_decreasing[OF  $\langle n \geq N \rangle$ , of c] by simp
    from c  $\langle m > n \rangle$  have  $1 - c \wedge^{m-n} > 0$ 
      using power_strict_mono[of c 1 m - n] by simp
    with  $\langle d > 0 \rangle \langle 0 < 1 - c \rangle$  have  $**$ :  $d * (1 - c \wedge^{m-n}) / (1 - c) > 0$ 
      by simp
    from cf_z2[of n m - n]  $\langle m > n \rangle$ 
    have  $\text{dist } (z \ m) \ (z \ n) \leq c \wedge^n * d * (1 - c \wedge^{m-n}) / (1 - c)$ 
    by (simp add: pos_le_divide_eq[OF  $\langle 1 - c > 0 \rangle$ ] mult_commute dist_commute)
    also have  $\dots \leq c \wedge^N * d * (1 - c \wedge^{m-n}) / (1 - c)$ 
      using mult_right_mono[OF  $*$  order_less_imp_le[OF  $**$ ]]
      by (simp add: mult.assoc)
    also have  $\dots < (e * (1 - c) / d) * d * (1 - c \wedge^{m-n}) / (1 - c)$ 
      using mult_strict_right_mono[OF N  $**$ ] by (auto simp: mult.assoc)
  
```

```

also from  $c \langle d \rangle 0 \rangle \langle 1 - c \rangle 0 \rangle$  have  $\dots = e * (1 - c \wedge (m - n))$ 
  by simp
also from  $c \langle 1 - c \wedge (m - n) \rangle 0 \rangle \langle e \rangle 0 \rangle$  have  $\dots \leq e$ 
  using mult_right_le_one_le[of  $e \ 1 - c \wedge (m - n)$ ] by auto
finally show ?thesis by simp
qed
have  $\text{dist } (z \ n) \ (z \ m) < e$  if  $N \leq m \ N \leq n$  for  $m \ n :: \text{nat}$ 
proof (cases  $n = m$ )
  case True
    with  $\langle e \rangle 0 \rangle$  show ?thesis by simp
  next
    case False
      with  $*[of \ n \ m] \ *[of \ m \ n]$  and that show ?thesis
      by (auto simp: dist_commute nat_neq_iff)
    qed
  then show ?thesis by auto
qed
then have Cauchy  $z$ 
  by (metis metric_CauchyI)
then obtain  $x$  where  $x \in s$  and  $x : (z \ \longrightarrow \ x)$  sequentially
  using  $s(1)[\text{unfolded compact\_def complete\_def}, \text{ THEN spec}[\text{where } x=z]]$  and
 $z\_in\_s$  by auto

define  $e$  where  $e = \text{dist } (f \ x) \ x$ 
have  $e = 0$ 
proof (rule ccontr)
  assume  $e \neq 0$ 
  then have  $e > 0$ 
    unfolding  $e\_def$  using zero_le_dist[of  $f \ x \ x$ ]
    by (metis dist_eq_0_iff dist_nz e_def)
  then obtain  $N$  where  $N : \forall n \geq N. \text{dist } (z \ n) \ x < e/2$ 
    using  $x[\text{unfolded lim\_sequentially}, \text{ THEN spec}[\text{where } x=e/2]]$  by auto
  then have  $N' : \text{dist } (z \ N) \ x < e/2$  by auto
  have  $*$ :  $c * \text{dist } (z \ N) \ x \leq \text{dist } (z \ N) \ x$ 
    unfolding mult_le_cancel_right2
    using zero_le_dist[of  $z \ N \ x$ ] and  $c$ 
    by (metis dist_eq_0_iff dist_nz order_less_asym less_le)
  have  $\text{dist } (f \ (z \ N)) \ (f \ x) \leq c * \text{dist } (z \ N) \ x$ 
    using lipschitz[THEN bspec[where  $x=z \ N$ ], THEN bspec[where  $x=x$ ]]
    using  $z\_in\_s$ [of  $N$ ]  $\langle x \in s \rangle$ 
    using  $c$ 
    by auto
  also have  $\dots < e/2$ 
    using  $N'$  and  $c$  using  $*$  by auto
  finally show False
    unfolding  $fzn$ 
    using  $N$ [THEN spec[where  $x=\text{Suc } N$ ]] and dist_triangle_half_r[of  $(\text{Suc } N) \ f \ x \ e \ x$ ]
    unfolding  $e\_def$ 

```

```

    by auto
  qed
  then have  $f x = x$  by (auto simp: e_def)
  moreover have  $y = x$  if  $f y = y$   $y \in s$  for  $y$ 
  proof -
    from  $\langle x \in s \rangle \langle f x = x \rangle$  that have  $\text{dist } x y \leq c * \text{dist } x y$ 
      using lipschitz[THEN bspec[where  $x=x$ ], THEN bspec[where  $x=y$ ]] by simp
    with  $c$  and zero_le_dist[of  $x y$ ] have  $\text{dist } x y = 0$ 
      by (simp add: mult_le_cancel_right1)
    then show ?thesis by simp
  qed
  ultimately show ?thesis
    using  $\langle x \in s \rangle$  by blast
qed

```

4.13 Edelstein fixed point theorem

```

theorem Edelstein_fix:
  fixes  $S :: 'a::metric\_space$  set
  assumes  $S$ : compact  $S$   $S \neq \{\}$ 
    and  $gs$ :  $(g \text{ ' } S) \subseteq S$ 
    and  $dist$ :  $\forall x \in S. \forall y \in S. x \neq y \longrightarrow \text{dist } (g x) (g y) < \text{dist } x y$ 
  shows  $\exists ! x \in S. g x = x$ 
proof -
  let  $?D = (\lambda x. (x, x)) \text{ ' } S$ 
  have  $D$ : compact  $?D$   $?D \neq \{\}$ 
    by (rule compact_continuous_image)
    (auto intro!: S continuous_Pair continuous_ident simp: continuous_on_eq_continuous_within)

  have  $\bigwedge x y e. x \in S \implies y \in S \implies 0 < e \implies \text{dist } y x < e \implies \text{dist } (g y) (g x) < e$ 
    using dist by fastforce
  then have continuous_on  $S$   $g$ 
    by (auto simp: continuous_on_iff)
  then have cont: continuous_on  $?D$   $(\lambda x. \text{dist } ((g \circ fst) x) (snd x))$ 
    unfolding continuous_on_eq_continuous_within
    by (intro continuous_dist ballI continuous_within_compose)
    (auto intro!: continuous_fst continuous_snd continuous_ident simp: image_image)

  obtain  $a$  where  $a \in S$  and  $le$ :  $\bigwedge x. x \in S \implies \text{dist } (g a) a \leq \text{dist } (g x) x$ 
    using continuous_attains_inf[OF  $D$  cont] by auto

  have  $g a = a$ 
  proof (rule ccontr)
    assume  $g a \neq a$ 
    with  $\langle a \in S \rangle$   $gs$  have  $\text{dist } (g (g a)) (g a) < \text{dist } (g a) a$ 
      by (intro dist[rule_format]) auto
    moreover have  $\text{dist } (g a) a \leq \text{dist } (g (g a)) (g a)$ 

```

```

    using ⟨a ∈ S⟩ gs by (intro le) auto
    ultimately show False by auto
  qed
  moreover have  $\bigwedge x. x \in S \implies g x = x \implies x = a$ 
    using dist[THEN bspec[where x=a]] ⟨g a = a⟩ and ⟨a ∈ S⟩ by auto
  ultimately show  $\exists ! x \in S. g x = x$ 
    using ⟨a ∈ S⟩ by blast
  qed

```

4.14 The diameter of a set

definition *diameter* :: 'a::metric_space set \Rightarrow real **where**
diameter S = (if S = {} then 0 else SUP (x,y) ∈ S × S. dist x y)

lemma *diameter_empty* [simp]: *diameter*{ } = 0
 by (auto simp: *diameter_def*)

lemma *diameter_singleton* [simp]: *diameter*{x} = 0
 by (auto simp: *diameter_def*)

lemma *diameter_le*:
 assumes $S \neq \{\}$ $\vee 0 \leq d$
 and no: $\bigwedge x y. \llbracket x \in S; y \in S \rrbracket \implies \text{norm}(x - y) \leq d$
 shows *diameter* S $\leq d$
 using *assms*
 by (auto simp: *dist_norm diameter_def intro: cSUP_least*)

lemma *diameter_bounded_bound*:
 fixes S :: 'a :: metric_space set
 assumes S: *bounded* S $x \in S y \in S$
 shows $\text{dist } x y \leq \text{diameter } S$

proof –
 from S obtain z d **where** $z: \bigwedge x. x \in S \implies \text{dist } z x \leq d$
 unfolding *bounded_def* by auto
 have *bdd_above* (case_prod dist ' (S × S))
proof (intro *bdd_aboveI*, safe)
 fix a b
 assume $a \in S b \in S$
 with $z[\text{of } a] z[\text{of } b]$ *dist_triangle*[of a b z]
 show $\text{dist } a b \leq 2 * d$
 by (simp add: *dist_commute*)
 qed
 moreover have $(x,y) \in S \times S$ using S by auto
 ultimately have $\text{dist } x y \leq (\text{SUP } (x,y) \in S \times S. \text{dist } x y)$
 by (rule *cSUP_upper2*) simp
 with ⟨x ∈ S⟩ show ?thesis
 by (auto simp: *diameter_def*)
 qed

```

lemma diameter_lower_bounded:
  fixes  $S :: 'a :: \text{metric\_space}$  set
  assumes  $S$ : bounded  $S$ 
    and  $d$ :  $0 < d$   $d < \text{diameter } S$ 
  shows  $\exists x \in S. \exists y \in S. d < \text{dist } x \ y$ 
proof (rule ccontr)
  assume contr:  $\neg$  ?thesis
  moreover have  $S \neq \{\}$ 
    using  $d$  by (auto simp: diameter_def)
  ultimately have  $\text{diameter } S \leq d$ 
    by (auto simp: not_less diameter_def intro!: cSUP_least)
  with  $\langle d < \text{diameter } S \rangle$  show False by auto
qed

lemma diameter_bounded:
  assumes bounded  $S$ 
  shows  $\forall x \in S. \forall y \in S. \text{dist } x \ y \leq \text{diameter } S$ 
    and  $\forall d > 0. d < \text{diameter } S \longrightarrow (\exists x \in S. \exists y \in S. \text{dist } x \ y > d)$ 
  using diameter_bounded_bound[of  $S$ ] diameter_lower_bounded[of  $S$ ] assms
  by auto

lemma bounded_two_points: bounded  $S \longleftrightarrow (\exists e. \forall x \in S. \forall y \in S. \text{dist } x \ y \leq e)$ 
  by (meson bounded_def diameter_bounded(1))

lemma diameter_compact_attained:
  assumes compact  $S$ 
    and  $S \neq \{\}$ 
  shows  $\exists x \in S. \exists y \in S. \text{dist } x \ y = \text{diameter } S$ 
proof –
  have  $b$ : bounded  $S$  using assms(1)
    by (rule compact_imp_bounded)
  then obtain  $x \ y$  where  $xys$ :  $x \in S \ y \in S$ 
    and  $xy$ :  $\forall u \in S. \forall v \in S. \text{dist } u \ v \leq \text{dist } x \ y$ 
    using compact_sup_maxdistance[OF assms] by auto
  then have  $\text{diameter } S \leq \text{dist } x \ y$ 
    unfolding diameter_def by (force intro!: cSUP_least)
  then show ?thesis
    by (metis b diameter_bounded_bound order_antisym xys)
qed

lemma diameter_ge_0:
  assumes bounded  $S$  shows  $0 \leq \text{diameter } S$ 
  by (metis all_not_in_conv assms diameter_bounded_bound diameter_empty
dist_self order_refl)

lemma diameter_subset:
  assumes  $S \subseteq T$  bounded  $T$ 
  shows  $\text{diameter } S \leq \text{diameter } T$ 
proof (cases  $S = \{\}$   $\vee$   $T = \{\}$ )

```

```

    case True
    with assms show ?thesis
      by (force simp: diameter_ge_0)
  next
  case False
  then have bdd_above (( $\lambda x$ . case  $x$  of ( $x, xa$ )  $\Rightarrow$   $\text{dist } x \ xa$ ) ' ( $T \times T$ ))
    using <bounded T> diameter_bounded_bound by (force simp: bdd_above_def)
  with False < $S \subseteq T$ > show ?thesis
    apply (simp add: diameter_def)
    apply (rule cSUP_subset_mono, auto)
  done
qed

lemma diameter_closure:
  assumes bounded S
  shows diameter(closure S) = diameter S
proof (rule order_antisym)
  have False if d_less_d: diameter S < diameter (closure S)
  proof -
    define d where  $d = \text{diameter}(\text{closure } S) - \text{diameter}(S)$ 
    have  $d > 0$ 
      using that by (simp add: d_def)
    then have dle:  $\text{diameter}(\text{closure}(S)) - d / 2 < \text{diameter}(\text{closure}(S))$ 
      by simp
    have dd:  $\text{diameter}(\text{closure } S) - d / 2 = (\text{diameter}(\text{closure}(S)) + \text{diameter}(S)) / 2$ 
      by (simp add: d_def field_split_simps)
    have bocl: bounded (closure S)
      using assms by blast
    moreover have  $0 \leq \text{diameter } S$ 
      using assms diameter_ge_0 by blast
    ultimately obtain  $x \ y$  where  $x \in \text{closure } S \ y \in \text{closure } S$  and  $xy$ :  $\text{diameter}(\text{closure}(S)) - d / 2 < \text{dist } x \ y$ 
      by (smt (verit) dle d_less_d d_def dd diameter_lower_bounded)
    then obtain  $x' \ y'$  where  $x'y'$ :  $x' \in S \ \text{dist } x' \ x < d/4 \ y' \in S \ \text{dist } y' \ y < d/4$ 
      by (metis < $0 < d$ > zero_less_divide_iff zero_less_numeral closure_approachable)
    then have  $\text{dist } x' \ y' \leq \text{diameter } S$ 
      using assms diameter_bounded_bound by blast
    with  $x'y'$  have  $\text{dist } x \ y \leq d / 4 + \text{diameter } S + d / 4$ 
      by (meson add_mono dist_triangle dist_triangle3 less_eq_real_def order_trans)
    then show ?thesis
      using  $xy \ d\_def$  by linarith
  qed
  then show diameter (closure S)  $\leq$  diameter S
    by fastforce
  next
  show diameter S  $\leq$  diameter (closure S)
    by (simp add: assms bounded_closure closure_subset diameter_subset)

```

qed

proposition *Lebesgue_number_lemma:*

assumes *compact* $S \subseteq \bigcup C \neq \{\}$ **and** *ope*: $\bigwedge B. B \in C \implies \text{open } B$

obtains δ **where** $0 < \delta \wedge T. \llbracket T \subseteq S; \text{diameter } T < \delta \rrbracket \implies \exists B \in C. T \subseteq B$

proof (*cases* $S = \{\}$)

case *True*

then show *?thesis*

by (*metis* $\langle C \neq \{\} \rangle$ *zero_less_one empty_subsetI equals0I subset_trans that*)

next

case *False*

{ fix x **assume** $x \in S$

then obtain C **where** $C: x \in C \wedge C \in \mathcal{C}$

using $\langle S \subseteq \bigcup C \rangle$ **by** *blast*

then obtain r **where** $r: r > 0 \wedge \text{ball } x (2*r) \subseteq C$

by (*metis* *mult.commute mult_2_right not_le ope openE field_sum_of_halves zero_le_numeral zero_less_mult_iff*)

then have $\exists r C. r > 0 \wedge \text{ball } x (2*r) \subseteq C \wedge C \in \mathcal{C}$

using C **by** *blast*

}

then obtain r **where** $r: \bigwedge x. x \in S \implies r x > 0 \wedge (\exists C \in \mathcal{C}. \text{ball } x (2*r x) \subseteq C)$

by *metis*

then have $S \subseteq (\bigcup x \in S. \text{ball } x (r x))$

by *auto*

then obtain \mathcal{T} **where** *finite* $\mathcal{T} \wedge S \subseteq \bigcup \mathcal{T}$ **and** $\mathcal{T}: \mathcal{T} \subseteq (\lambda x. \text{ball } x (r x)) \text{ ' } S$

by (*rule compactE [OF <compact S>]*) *auto*

then obtain S_0 **where** $S_0 \subseteq S$ *finite* S_0 **and** $S_0: \mathcal{T} = (\lambda x. \text{ball } x (r x)) \text{ ' } S_0$

by (*meson finite_subset_image*)

then have $S_0 \neq \{\}$

using *False* $\langle S \subseteq \bigcup \mathcal{T} \rangle$ **by** *auto*

define δ **where** $\delta = \text{Inf } (r \text{ ' } S_0)$

have $\delta > 0$

using $\langle \text{finite } S_0 \rangle \langle S_0 \subseteq S \rangle \langle S_0 \neq \{\} \rangle$ **by** (*auto simp: delta_def finite_less_Inf_iff*)

show *?thesis*

proof

show $0 < \delta$

by (*simp add: <0 < delta>*)

show $\exists B \in \mathcal{C}. T \subseteq B$ **if** $T \subseteq S$ **and** *dia*: *diameter* $T < \delta$ **for** T

proof (*cases* $T = \{\}$)

case *True*

then show *?thesis*

using $\langle C \neq \{\} \rangle$ **by** *blast*

next

case *False*

then obtain y **where** $y \in T$ **by** *blast*

then have $y \in S$

using $\langle T \subseteq S \rangle$ **by** *auto*

then obtain x **where** $x \in S_0$ **and** $x: y \in \text{ball } x (r x)$

```

    using ⟨S ⊆ ⋃ T⟩ S0 that by blast
  have ball y δ ⊆ ball y (r x)
    by (metis δ_def ⟨S0 ≠ {}⟩ ⟨finite S0⟩ ⟨x ∈ S0⟩ empty_is_image fi-
nite_imageI finite_less_Inf_iff imageI less_irrefl not_le subset_ball)
  also have ... ⊆ ball x (2*r x)
    using x by metric
  finally obtain C where C ∈ C ball y δ ⊆ C
    by (meson r ⟨S0 ⊆ S⟩ ⟨x ∈ S0⟩ dual_order.trans subsetCE)
  have bounded T
    using ⟨compact S⟩ bounded_subset compact_imp_bounded ⟨T ⊆ S⟩ by blast
  then have T ⊆ ball y δ
    using ⟨y ∈ T⟩ dia diameter_bounded_bound by fastforce
  then show ?thesis
    by (meson ⟨C ∈ C⟩ ⟨ball y δ ⊆ C⟩ subset_eq)
qed
qed
qed

```

4.15 Metric spaces with the Heine-Borel property

A metric space (or topological vector space) is said to have the Heine-Borel property if every closed and bounded subset is compact.

```

class heine_borel = metric_space +
  assumes bounded_imp_convergent_subsequence:
    bounded (range f) ⟹ ∃ l r. strict_mono (r::nat⇒nat) ∧ ((f ∘ r) ⟶ l)
    sequentially

```

proposition *bounded_closed_imp_seq_compact*:

```

  fixes S::'a::heine_borel set
  assumes bounded S
    and closed S
  shows seq_compact S
proof (unfold seq_compact_def, clarify)
  fix f :: nat ⇒ 'a
  assume f: ∀ n. f n ∈ S
  with ⟨bounded S⟩ have bounded (range f)
    by (auto intro: bounded_subset)
  obtain l r where r: strict_mono (r :: nat ⇒ nat) and l: ((f ∘ r) ⟶ l)
    sequentially
  using bounded_imp_convergent_subsequence [OF ⟨bounded (range f)⟩] by auto
  from f have fr: ∀ n. (f ∘ r) n ∈ S
    by simp
  show ∃ l ∈ S. ∃ r. strict_mono r ∧ (f ∘ r) ⟶ l
    using assms(2) closed_sequentially fr l r by blast
qed

```

lemma *compact_eq_bounded_closed*:

```

  fixes S :: 'a::heine_borel set

```



```

shows compact S  $\longleftrightarrow$  bounded S  $\wedge$  closed S
using bounded_closed_imp_seq_compact compact_eq_seq_compact_metric compact_imp_bounded compact_imp_closed
by auto

```

```

lemma bounded_infinite_imp_islimpt:
  fixes S :: 'a::heine_borel set
  assumes T  $\subseteq$  S bounded S infinite T
  obtains x where x islimpt S
by (meson assms closed_limpt compact_eq_Bolzano_Weierstrass compact_eq_bounded_closed islimpt_subset)

```

```

lemma compact_Inter:
  fixes  $\mathcal{F}$  :: 'a :: heine_borel set set
  assumes com:  $\bigwedge S. S \in \mathcal{F} \implies \text{compact } S$  and  $\mathcal{F} \neq \{\}$ 
  shows compact( $\bigcap \mathcal{F}$ )
  using assms
by (meson Inf_lower all_not_in_conv bounded_subset closed_Inter compact_eq_bounded_closed)

```

```

lemma compact_closure [simp]:
  fixes S :: 'a::heine_borel set
  shows compact(closure S)  $\longleftrightarrow$  bounded S
by (meson bounded_closure bounded_subset closed_closure closure_subset compact_eq_bounded_closed)

```

```

instance real :: heine_borel

```

```

proof

```

```

  fix f :: nat  $\Rightarrow$  real
  assume f: bounded (range f)
  obtain r :: nat  $\Rightarrow$  nat where r: strict_mono r monoseq (f  $\circ$  r)
  unfolding comp_def by (metis seq_monosub)
  then have Bseq (f  $\circ$  r)
  unfolding Bseq_eq_bounded by (metis f BseqI' bounded_iff comp_apply rangeI)
  with r show  $\exists l r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  using Bseq_monoseq_convergent[of f  $\circ$  r] by (auto simp: convergent_def)
qed

```

```

lemma compact_lemma_general:

```

```

  fixes f :: nat  $\Rightarrow$  'a
  fixes proj::'a  $\Rightarrow$  'b  $\Rightarrow$  'c::heine_borel (infixl proj 60)
  fixes unproj::('b  $\Rightarrow$  'c)  $\Rightarrow$  'a
  assumes finite_basis: finite basis
  assumes bounded_proj:  $\bigwedge k. k \in \text{basis} \implies \text{bounded } ((\lambda x. x \text{ proj } k) \text{ ` range } f)$ 
  assumes proj_unproj:  $\bigwedge e k. k \in \text{basis} \implies (\text{unproj } e) \text{ proj } k = e k$ 
  assumes unproj_proj:  $\bigwedge x. \text{unproj } (\lambda k. x \text{ proj } k) = x$ 
  shows  $\forall d \subseteq \text{basis}. \exists l::'a. \exists r::\text{nat} \Rightarrow \text{nat}.
    \text{strict\_mono } r \wedge (\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f (r n) \text{ proj } i) (l \text{ proj } i) < e) \text{ sequentially})$ 

```

```

proof safe
  fix d :: 'b set
  assume d: d  $\subseteq$  basis
  with finite_basis have finite d
    by (blast intro: finite_subset)
  from this d show  $\exists l::'a. \exists r::nat \Rightarrow nat. strict\_mono\ r \wedge$ 
    ( $\forall e > 0. eventually\ (\lambda n. \forall i \in d. dist\ (f\ (r\ n)\ proj\ i)\ (l\ proj\ i) < e)$  sequentially)
  proof (induct d)
    case empty
    then show ?case
      unfolding strict_mono_def by auto
  next
    case (insert k d)
    have k[intro]: k  $\in$  basis
      using insert by auto
    have s': bounded (( $\lambda x. x\ proj\ k$ ) ' range f)
      using k
      by (rule bounded_proj)
    obtain l1::'a and r1 where r1: strict_mono r1
      and lr1:  $\forall e > 0. \forall_F\ n\ in\ sequentially. \forall i \in d. dist\ (f\ (r1\ n)\ proj\ i)\ (l1\ proj$ 
i) < e
      using insert by auto
    have f':  $\forall n. f\ (r1\ n)\ proj\ k \in (\lambda x. x\ proj\ k)$  ' range f
      by simp
    have bounded (range ( $\lambda i. f\ (r1\ i)\ proj\ k$ ))
      by (metis (lifting) bounded_subset f' image_subsetI s')
    then obtain l2 r2 where r2: strict_mono r2 and lr2: ( $\lambda i. f\ (r1\ (r2\ i))\ proj$ 
k)  $\longrightarrow$  l2
      using bounded_imp_convergent_subsequence[of  $\lambda i. f\ (r1\ i)\ proj\ k$ ]
      by (auto simp: o_def)
    define r where r = r1  $\circ$  r2
    have r: strict_mono r
      using r1 and r2 unfolding r_def o_def strict_mono_def by auto
    moreover
    define l where l = unproj ( $\lambda i. if\ i = k\ then\ l2\ else\ l1\ proj\ i$ )
    { fix e::real
      assume e > 0
      from lr1  $\langle e > 0 \rangle$  have N1:  $\forall_F\ n\ in\ sequentially. \forall i \in d. dist\ (f\ (r1\ n)\ proj$ 
i) (l1 proj i) < e
        by blast
      from lr2  $\langle e > 0 \rangle$  have N2:  $\forall_F\ n\ in\ sequentially. dist\ (f\ (r1\ (r2\ n))\ proj\ k)$ 
l2 < e
        by (rule tendstoD)
      from r2 N1 have N1':  $\forall_F\ n\ in\ sequentially. \forall i \in d. dist\ (f\ (r1\ (r2\ n))\ proj$ 
i) (l1 proj i) < e
        by (rule eventually_subseq)
      have  $\forall_F\ n\ in\ sequentially. \forall i \in insert\ k\ d. dist\ (f\ (r\ n)\ proj\ i)\ (l\ proj\ i) < e$ 
using N1' N2
        by eventually_elim (use insert.premis in  $\langle auto\ simp: l\_def\ r\_def\ o\_def$ 

```

```

proj_unproj)
}
ultimately show ?case by auto
qed
qed

```

```

lemma bounded_fst: bounded s  $\implies$  bounded (fst ' s)
  unfolding bounded_def
  by (metis (erased, opaque_lifting) dist_fst_le image_iff order_trans)

```

```

lemma bounded_snd: bounded s  $\implies$  bounded (snd ' s)
  unfolding bounded_def
  by (metis (no_types, opaque_lifting) dist_snd_le image_iff order.trans)

```

```

instance prod :: (heine_borel, heine_borel) heine_borel

```

```

proof

```

```

  fix f :: nat  $\implies$  'a  $\times$  'b
  assume f: bounded (range f)
  then have bounded (fst ' range f)
    by (rule bounded_fst)
  then have s1: bounded (range (fst  $\circ$  f))
    by (simp add: image_comp)
  obtain l1 r1 where r1: strict_mono r1 and l1: ( $\lambda n. \text{fst } (f (r1 n))$ )  $\longrightarrow$  l1
    using bounded_imp_convergent_subsequence [OF s1] unfolding o_def by fast
  from f have s2: bounded (range (snd  $\circ$  f  $\circ$  r1))
    by (auto simp: image_comp intro: bounded_snd bounded_subset)
  obtain l2 r2 where r2: strict_mono r2 and l2: ( $\lambda n. \text{snd } (f (r1 (r2 n)))$ )  $\longrightarrow$ 
    l2
    using bounded_imp_convergent_subsequence [OF s2]
    unfolding o_def by fast
  have l1': (( $\lambda n. \text{fst } (f (r1 (r2 n)))$ ))  $\longrightarrow$  l1 sequentially
    using LIMSEQ_subseq_LIMSEQ [OF l1 r2] unfolding o_def .
  have l: ((f  $\circ$  (r1  $\circ$  r2))  $\longrightarrow$  (l1, l2)) sequentially
    using tendsto_Pair [OF l1' l2] unfolding o_def by simp
  have r: strict_mono (r1  $\circ$  r2)
    using r1 r2 unfolding strict_mono_def by simp
  show  $\exists l r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l)$  sequentially
    using l r by fast
qed

```

4.16 Completeness

```

proposition (in metric_space) completeI:

```

```

  assumes  $\bigwedge f. \forall n. f n \in s \implies \text{Cauchy } f \implies \exists l \in s. f \longrightarrow l$ 
  shows complete s
  using assms unfolding complete_def by fast

```

```

proposition (in metric_space) completeE:

```

```

  assumes complete s and  $\forall n. f n \in s$  and Cauchy f

```

obtains l where $l \in s$ and $f \longrightarrow l$
using *assms* **unfolding** *complete_def* **by** *fast*

lemma *compact_imp_complete*:

fixes $s :: 'a::metric_space$ *set*

assumes *compact s*

shows *complete s*

proof –

{

fix f

assume $as: (\forall n::nat. f\ n \in s)$ *Cauchy f*

from $as(1)$ **obtain** $l\ r$ where $lr: l \in s$ *strict_mono r (f o r) \longrightarrow l*

using *assms* **unfolding** *compact_def* **by** *blast*

note $lr' = seq_suble$ [*OF lr(2)*]

{

fix $e :: real$

assume $e > 0$

from $as(2)$ **obtain** N where $N:\forall m\ n. N \leq m \wedge N \leq n \longrightarrow dist\ (f\ m)\ (f\ n) < e/2$

unfolding *Cauchy_def* **using** $\langle e > 0 \rangle$ **by** (*meson half_gt_zero*)

then obtain M where $M:\forall n \geq M. dist\ ((f\ o\ r)\ n)\ l < e/2$

by (*metis dist_self_lim_sequentially lr(3)*)

{

fix $n :: nat$

assume $n: n \geq \max\ N\ M$

have $dist\ ((f\ o\ r)\ n)\ l < e/2$

using $n\ M$ **by** *auto*

moreover have $r\ n \geq N$

using $lr'[of\ n]\ n$ **by** *auto*

then have $dist\ (f\ n)\ ((f\ o\ r)\ n) < e/2$

using N and n **by** *auto*

ultimately have $dist\ (f\ n)\ l < e$ **using** $n\ M$

by *metric*

}

then have $\exists N. \forall n \geq N. dist\ (f\ n)\ l < e$ **by** *blast*

}

then have $\exists l \in s. (f \longrightarrow l)$ *sequentially* **using** $\langle l \in s \rangle$

unfolding *lim_sequentially* **by** *auto*

}

then show *?thesis* **unfolding** *complete_def* **by** *auto*

qed

proposition *compact_eq_totally_bounded*:

$compact\ s \longleftrightarrow complete\ s \wedge (\forall e > 0. \exists k. finite\ k \wedge s \subseteq (\bigcup x \in k. ball\ x\ e))$

(*is* $_ \longleftrightarrow$ *?rhs*)

proof

assume *assms: ?rhs*

then obtain k **where** $k: \bigwedge e. 0 < e \implies \text{finite } (k\ e) \wedge e. 0 < e \implies s \subseteq (\bigcup_{x \in k\ e} \text{ball } x\ e)$

by (*auto simp: choice_iff*)

show *compact s*

proof *cases*

assume $s = \{\}$

then show *compact s* **by** (*simp add: compact_def*)

next

assume $s \neq \{\}$

show *?thesis*

unfolding *compact_def*

proof *safe*

fix $f :: \text{nat} \Rightarrow 'a$

assume $f: \forall n. f\ n \in s$

define e **where** $e\ n = 1 / (2 * \text{Suc } n)$ **for** n

then have [*simp*]: $\bigwedge n. 0 < e\ n$ **by** *auto*

define B **where** $B\ n\ U = (\text{SOME } b. \text{infinite } \{n. f\ n \in b\} \wedge (\exists x. b \subseteq \text{ball } x\ (e\ n) \cap U))$ **for** $n\ U$

{

fix $n\ U$

assume *infinite* $\{n. f\ n \in U\}$

then have $\exists b \in k\ (e\ n). \text{infinite } \{i \in \{n. f\ n \in U\}. f\ i \in \text{ball } b\ (e\ n)\}$

using $k\ f$ **by** (*intro pigeonhole_infinite_rel*) (*auto simp: subset_eq*)

then obtain a **where**

$a \in k\ (e\ n)$

infinite $\{i \in \{n. f\ n \in U\}. f\ i \in \text{ball } a\ (e\ n)\}$ **..**

then have $\exists b. \text{infinite } \{i. f\ i \in b\} \wedge (\exists x. b \subseteq \text{ball } x\ (e\ n) \cap U)$

by (*intro exI[of _ ball a (e n) \cap U] exI[of _ a]*) (*auto simp: ac_simps*)

from *someI_ex[OF this]*

have *infinite* $\{i. f\ i \in B\ n\ U\} \exists x. B\ n\ U \subseteq \text{ball } x\ (e\ n) \cap U$

unfolding B_def **by** *auto*

}

note $B = \text{this}$

define F **where** $F = \text{rec_nat } (B\ 0\ \text{UNIV})\ B$

{

fix n

have *infinite* $\{i. f\ i \in F\ n\}$

by (*induct n*) (*auto simp: F_def B*)

}

then have $F: \bigwedge n. \exists x. F\ (\text{Suc } n) \subseteq \text{ball } x\ (e\ n) \cap F\ n$

using B **by** (*simp add: F_def*)

then have $F_dec: \bigwedge m\ n. m \leq n \implies F\ n \subseteq F\ m$

using *decseq_SucI*[of F] **by** (*auto simp: decseq_def*)

obtain sel **where** $sel: \bigwedge k\ i. i < sel\ k\ i \wedge k\ i. f\ (sel\ k\ i) \in F\ k$

proof (*atomize_elim, unfold all_conj_distrib[symmetric], intro choice allI*)

```

fix  $k\ i$ 
have  $\text{infinite } (\{n. f\ n \in F\ k\} - \{.. i\})$ 
  using  $\langle \text{infinite } \{n. f\ n \in F\ k\} \rangle$  by  $\text{auto}$ 
from  $\text{infinite\_imp\_nonempty}$  [OF this]
show  $\exists x > i. f\ x \in F\ k$ 
  by  $(\text{simp add: set\_eq\_iff not\_le conj\_commute})$ 
qed

define  $t$  where  $t = \text{rec\_nat } (\text{sel } 0\ 0) (\lambda n\ i. \text{sel } (\text{Suc } n)\ i)$ 
have  $\text{strict\_mono } t$ 
  unfolding  $\text{strict\_mono\_Suc\_iff}$  by  $(\text{simp add: } t\_def\ \text{sel})$ 
moreover have  $\forall i. (f \circ t)\ i \in s$ 
  using  $f$  by  $\text{auto}$ 
moreover
have  $t: (f \circ t)\ n \in F\ n$  for  $n$ 
  by  $(\text{cases } n) (\text{simp\_all add: } t\_def\ \text{sel})$ 

have  $\text{Cauchy } (f \circ t)$ 
proof  $(\text{safe intro!; metric\_CauchyI exI elim!; nat\_approx\_posE})$ 
  fix  $r :: \text{real}$  and  $N\ n\ m$ 
  assume  $1 / \text{Suc } N < r$   $\text{Suc } N \leq n$   $\text{Suc } N \leq m$ 
  then have  $(f \circ t)\ n \in F\ (\text{Suc } N)$   $(f \circ t)\ m \in F\ (\text{Suc } N)$   $2 * e\ N < r$ 
    using  $F\_dec\ t$  by  $(\text{auto simp: } e\_def\ \text{field\_simps})$ 
  with  $F$  [of  $N$ ] obtain  $x$  where  $\text{dist } x ((f \circ t)\ n) < e\ N$   $\text{dist } x ((f \circ t)\ m)$ 
     $< e\ N$ 
    by  $(\text{auto simp: subset\_eq})$ 
  with  $\langle 2 * e\ N < r \rangle$  show  $\text{dist } ((f \circ t)\ m) ((f \circ t)\ n) < r$ 
    by  $\text{metric}$ 
qed

ultimately show  $\exists l \in s. \exists r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  using  $\text{assms}$  unfolding  $\text{complete\_def}$  by  $\text{blast}$ 
qed
qed
qed  $(\text{metis compact\_imp\_complete compact\_imp\_seq\_compact seq\_compact\_imp\_totally\_bounded})$ 

lemma  $\text{cauchy\_imp\_bounded}$ :
  assumes  $\text{Cauchy } s$ 
  shows  $\text{bounded } (\text{range } s)$ 
proof –
  from  $\text{assms}$  obtain  $N :: \text{nat}$  where  $\forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (s\ m)\ (s\ n) < 1$ 
    by  $(\text{meson } \text{Cauchy\_def zero\_less\_one})$ 
  then have  $N: \forall n. N \leq n \longrightarrow \text{dist } (s\ N)\ (s\ n) < 1$  by  $\text{auto}$ 
  moreover
  have  $\text{bounded } (s\ \{0..N\})$ 
    using  $\text{finite\_imp\_bounded}$  [of  $s\ \{1..N\}$ ] by  $\text{auto}$ 
  then obtain  $a$  where  $a: \forall x \in s\ \{0..N\}. \text{dist } (s\ N)\ x \leq a$ 
    unfolding  $\text{bounded\_any\_center}$  [where  $a = s\ N$ ] by  $\text{auto}$ 

```

```

ultimately show ?thesis
  unfolding bounded_any_center [where a=s N]
  apply (rule_tac x=max a 1 in exI, auto)
  apply (erule_tac x=y in allE)
  apply (erule_tac x=y in ballE, auto)
  done
qed

instance heine_borel < complete_space
proof
  fix f :: nat  $\Rightarrow$  'a assume Cauchy f
  then show convergent f
    unfolding convergent_def
    using Cauchy_converges_subseq cauchy_imp_bounded bounded_imp_convergent_subsequence
  by blast
qed

lemma complete_UNIV: complete (UNIV :: ('a::complete_space) set)
proof (rule completeI)
  fix f :: nat  $\Rightarrow$  'a assume Cauchy f
  then show  $\exists l \in UNIV. f \longrightarrow l$ 
    using Cauchy_convergent convergent_def by blast
qed

lemma complete_imp_closed:
  fixes S :: 'a::metric_space set
  shows complete S  $\implies$  closed S
  by (metis LIMSEQ_imp_Cauchy LIMSEQ_unique closed_sequential_limits completeE)

lemma complete_Int_closed:
  fixes S :: 'a::metric_space set
  assumes complete S and closed t
  shows complete (S  $\cap$  t)
  by (metis Int_iff assms closed_sequentially completeE completeI)

lemma complete_closed_subset:
  fixes S :: 'a::metric_space set
  assumes closed S and S  $\subseteq$  t and complete t
  shows complete S
  using assms complete_Int_closed [of t S] by (simp add: Int_absorb1)

lemma complete_eq_closed:
  fixes S :: ('a::complete_space) set
  shows complete S  $\iff$  closed S
proof
  assume closed S then show complete S
    using subset_UNIV complete_UNIV by (rule complete_closed_subset)
next

```

```

assume complete S then show closed S
  by (rule complete_imp_closed)
qed

```

```

lemma convergent_eq_Cauchy:
  fixes S :: nat  $\Rightarrow$  'a::complete_space
  shows ( $\exists l. (S \longrightarrow l)$  sequentially)  $\longleftrightarrow$  Cauchy S
  unfolding Cauchy_convergent_iff convergent_def ..

```

```

lemma convergent_imp_bounded:
  fixes S :: nat  $\Rightarrow$  'a::metric_space
  shows ( $S \longrightarrow l$ ) sequentially  $\implies$  bounded (range S)
  by (intro cauchy_imp_bounded LIMSEQ_imp_Cauchy)

```

```

lemma frontier_subset_compact:
  fixes S :: 'a::heine_borel set
  shows compact S  $\implies$  frontier S  $\subseteq$  S
  using frontier_subset_closed compact_eq_bounded_closed
  by blast

```

```

lemma banach_fix_type:
  fixes f::'a::complete_space $\Rightarrow$ 'a
  assumes c: 0  $\leq$  c < 1
    and lipschitz: $\forall x. \forall y. \text{dist } (f\ x) (f\ y) \leq c * \text{dist } x\ y$ 
  shows  $\exists!x. (f\ x = x)$ 
  using assms banach_fix[OF complete_UNIV UNIV_not_empty assms(1,2) subset_UNIV, of f]
  by auto

```

4.17 Cauchy continuity

```

definition Cauchy_continuous_on where
  Cauchy_continuous_on  $\equiv \lambda S f. \forall \sigma. \text{Cauchy } \sigma \longrightarrow \text{range } \sigma \subseteq S \longrightarrow \text{Cauchy } (f \circ \sigma)$ 

```

```

lemma continuous_closed_imp_Cauchy_continuous:
  fixes S :: ('a::complete_space) set
  shows  $\llbracket \text{continuous\_on } S\ f; \text{closed } S \rrbracket \implies \text{Cauchy\_continuous\_on } S\ f$ 
  unfolding Cauchy_continuous_on_def
  by (metis LIMSEQ_imp_Cauchy completeE complete_eq_closed continuous_on_sequentially range_subsetD)

```

```

lemma uniformly_continuous_imp_Cauchy_continuous:
  fixes f :: 'a::metric_space  $\Rightarrow$  'b::metric_space
  shows uniformly_continuous_on S f  $\implies$  Cauchy_continuous_on S f
  by (simp add: uniformly_continuous_on_def Cauchy_continuous_on_def Cauchy_def image_subset_iff) metis

```

```

lemma Cauchy_continuous_on_imp_continuous:

```



```

fixes f :: 'a::metric_space  $\Rightarrow$  'b::metric_space
assumes Cauchy_continuous_on S f
shows continuous_on S f
proof -
  have False if x:  $\forall n. \exists x' \in S. \text{dist } x' x < \text{inverse}(\text{Suc } n) \wedge \neg \text{dist } (f x') (f x) < \varepsilon$ 
 $\varepsilon > 0 \ x \in S$  for x and  $\varepsilon :: \text{real}$ 
  proof -
    obtain  $\varrho$  where  $\varrho: \forall n. \varrho n \in S$  and  $dx: \forall n. \text{dist } (\varrho n) x < \text{inverse}(\text{Suc } n)$ 
  and  $dfx: \forall n. \neg \text{dist } (f (\varrho n)) (f x) < \varepsilon$ 
    using x by metis
    define  $\sigma$  where  $\sigma \equiv \lambda n. \text{if even } n \text{ then } \varrho n \text{ else } x$ 
    with  $\varrho \langle x \in S \rangle$  have  $\text{range } \sigma \subseteq S$ 
      by auto
    have  $\sigma \longrightarrow x$ 
      unfolding tendsto_iff
    proof (intro strip)
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain N where  $\text{inverse } (\text{Suc } N) < e$ 
        using reals_Archimedean by blast
      then have  $\forall n. N \leq n \longrightarrow \text{dist } (\varrho n) x < e$ 
      by (smt (verit, ccfv_SIG) dx inverse_Suc inverse_less_iff_less inverse_positive_iff_positive
of_nat_Suc of_nat_mono)
      with  $\langle e > 0 \rangle$  show  $\forall_F n$  in sequentially.  $\text{dist } (\sigma n) x < e$ 
        by (auto simp add: eventually_sequentially  $\sigma\_def$ )
    qed
    then have Cauchy  $\sigma$ 
      by (intro LIMSEQ_imp_Cauchy)
    then have Cf: Cauchy  $(f \circ \sigma)$ 
      by (meson Cauchy_continuous_on_def  $\langle \text{range } \sigma \subseteq S \rangle$  assms)
    have  $(f \circ \sigma) \longrightarrow f x$ 
      unfolding tendsto_iff
    proof (intro strip)
      fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then obtain N where  $N: \forall m \geq N. \forall n \geq N. \text{dist } ((f \circ \sigma) m) ((f \circ \sigma) n) < e$ 
        using Cf unfolding Cauchy_def by presburger
      moreover have  $(f \circ \sigma) (\text{Suc}(N+N)) = f x$ 
        by (simp add:  $\sigma\_def$ )
      ultimately have  $\forall n \geq N. \text{dist } ((f \circ \sigma) n) (f x) < e$ 
        by (metis add_Suc le_add2)
      then show  $\forall_F n$  in sequentially.  $\text{dist } ((f \circ \sigma) n) (f x) < e$ 
        using eventually_sequentially by blast
    qed
    moreover have  $\bigwedge n. \neg \text{dist } (f (\sigma (2*n))) (f x) < \varepsilon$ 
      using dfx by (simp add:  $\sigma\_def$ )
    ultimately show False
      using  $\langle \varepsilon > 0 \rangle$  by (fastforce simp: mult_2 nat_le_iff_add tendsto_iff eventually_sequentially)
  
```

```

qed
then show ?thesis
  unfolding continuous_on_iff by (meson inverse_Suc)
qed

```

4.18 Finite intersection property

Also developed in HOL's topological spaces theory, but the Heine-Borel type class isn't available there.

```

lemma closed_imp_fip:
  fixes S :: 'a::heine_borel set
  assumes closed S
    and T: T ∈ F bounded T
    and clof: ∧T. T ∈ F ⇒ closed T
    and ∧F'. [finite F'; F' ⊆ F] ⇒ S ∩ ∩F' ≠ {}
  shows S ∩ ∩F ≠ {}
proof -
  have compact (S ∩ T)
    using ⟨closed S⟩ clof compact_eq_bounded_closed T by blast
  then have (S ∩ T) ∩ ∩F ≠ {}
    by (smt (verit, best) Inf_insert Int_assoc assms compact_imp_fip finite_insert
insert_subset)
  then show ?thesis by blast
qed

```

```

lemma closed_imp_fip_compact:
  fixes S :: 'a::heine_borel set
  shows
    [closed S; ∧T. T ∈ F ⇒ compact T;
    ∧F'. [finite F'; F' ⊆ F] ⇒ S ∩ ∩F' ≠ {}]
    ⇒ S ∩ ∩F ≠ {}
  by (metis closed_imp_fip compact_eq_bounded_closed equalsOI finite.emptyI or-
der.refl)

```

```

lemma closed_fip_Heine_Borel:
  fixes F :: 'a::heine_borel set set
  assumes T ∈ F bounded T
    and ∧T. T ∈ F ⇒ closed T
    and ∧F'. [finite F'; F' ⊆ F] ⇒ ∩F' ≠ {}
  shows ∩F ≠ {}
  using closed_imp_fip [OF closed_UNIV] assms by auto

```

```

lemma compact_fip_Heine_Borel:
  fixes F :: 'a::heine_borel set set
  assumes clof: ∧T. T ∈ F ⇒ compact T
    and none: ∧F'. [finite F'; F' ⊆ F] ⇒ ∩F' ≠ {}
  shows ∩F ≠ {}
  by (metis InterI clof closed_fip_Heine_Borel compact_eq_bounded_closed equalsOD)

```

none)

lemma *compact_sequence_with_limit*:
fixes $f :: \text{nat} \Rightarrow 'a::\text{heine_borel}$
shows $f \longrightarrow l \implies \text{compact } (\text{insert } l \ (\text{range } f))$
by (*simp add: closed_limpt compact_eq_bounded_closed convergent_imp_bounded islimpt_insert sequence_unique_limpt*)

4.19 Properties of Balls and Spheres

lemma *compact_cball[simp]*:
fixes $x :: 'a::\text{heine_borel}$
shows $\text{compact } (\text{cball } x \ e)$
using *compact_eq_bounded_closed bounded_cball closed_cball* **by** *blast*

lemma *compact_frontier_bounded[intro]*:
fixes $S :: 'a::\text{heine_borel}$ *set*
shows $\text{bounded } S \implies \text{compact } (\text{frontier } S)$
unfolding *frontier_def*
using *compact_eq_bounded_closed* **by** *blast*

lemma *compact_frontier[intro]*:
fixes $S :: 'a::\text{heine_borel}$ *set*
shows $\text{compact } S \implies \text{compact } (\text{frontier } S)$
using *compact_eq_bounded_closed compact_frontier_bounded* **by** *blast*

lemma *no_limpt_imp_countable*:
assumes $\bigwedge z. \neg \text{islimpt } (X :: 'a :: \{\text{real_normed_vector}, \text{heine_borel}\} \text{ set})$
shows *countable* X

proof –

have *countable* $(\bigcup n. \text{cball } 0 \ (\text{real } n) \cap X)$
proof (*intro countable_UN[OF _ countable_finite]*)
fix $n :: \text{nat}$
show *finite* $(\text{cball } 0 \ (\text{real } n) \cap X)$
using *assms* **by** (*intro finite_not_islimpt_in_compact*) *auto*
qed *auto*
also **have** $(\bigcup n. \text{cball } 0 \ (\text{real } n)) = (\text{UNIV} :: 'a \text{ set})$
by (*force simp: real_arch_simple*)
hence $(\bigcup n. \text{cball } 0 \ (\text{real } n) \cap X) = X$
by *blast*
finally **show** *countable* X .

qed

4.20 Distance from a Set

lemma *distance_attains_sup*:
assumes $\text{compact } s \ s \neq \{\}$
shows $\exists x \in s. \forall y \in s. \text{dist } a \ y \leq \text{dist } a \ x$

```

proof (rule continuous_attains_sup [OF assms])
  {
    fix x
    assume x∈s
    have (dist a  $\longrightarrow$  dist a x) (at x within s)
      by (intro tendsto_dist tendsto_const tendsto_ident_at)
  }
  then show continuous_on s (dist a)
    unfolding continuous_on ..
qed

```

For *minimal* distance, we only need closure, not compactness.

```

lemma distance_attains_inf:
  fixes a :: 'a::heine_borel
  assumes closed s and s ≠ {}
  obtains x where x∈s  $\wedge$  y. y ∈ s  $\implies$  dist a x  $\leq$  dist a y
proof -
  from assms obtain b where b ∈ s by auto
  let ?B = s  $\cap$  cball a (dist b a)
  have ?B ≠ {} using ⟨b ∈ s⟩
    by (auto simp: dist_commute)
  moreover have continuous_on ?B (dist a)
    by (auto intro!: continuous_at_imp_continuous_on continuous_dist continuous_ident continuous_const)
  moreover have compact ?B
    by (intro closed_Int_compact ⟨closed s⟩ compact_cball)
  ultimately obtain x where x ∈ ?B  $\forall$  y∈?B. dist a x  $\leq$  dist a y
    by (metis continuous_attains_inf)
  with that show ?thesis by fastforce
qed

```

4.21 Infimum Distance

definition infdist x A = (if A = {} then 0 else INF a∈A. dist x a)

```

lemma bdd_below_image_dist[intro, simp]: bdd_below (dist x ‘ A)
  by (auto intro!: zero_le_dist)

```

```

lemma infdist_notempty: A ≠ {}  $\implies$  infdist x A = (INF a∈A. dist x a)
  by (simp add: infdist_def)

```

```

lemma infdist_nonneg: 0  $\leq$  infdist x A
  by (auto simp: infdist_def intro: cINF_greatest)

```

```

lemma infdist_le: a ∈ A  $\implies$  infdist x A  $\leq$  dist x a
  by (auto intro: cINF_lower simp add: infdist_def)

```

```

lemma infdist_le2: a ∈ A  $\implies$  dist x a  $\leq$  d  $\implies$  infdist x A  $\leq$  d
  by (auto intro!: cINF_lower2 simp add: infdist_def)

```

```

lemma infdist_zero[simp]:  $a \in A \implies \text{infdist } a \ A = 0$ 
  by (auto intro!: antisym infdist_nonneg infdist_le2)

lemma infdist_Un_min:
  assumes  $A \neq \{\}$   $B \neq \{\}$ 
  shows  $\text{infdist } x \ (A \cup B) = \min (\text{infdist } x \ A) (\text{infdist } x \ B)$ 
using assms by (simp add: infdist_def cINF_union inf_real_def)

lemma infdist_triangle:  $\text{infdist } x \ A \leq \text{infdist } y \ A + \text{dist } x \ y$ 
proof (cases  $A = \{\}$ )
  case True
  then show ?thesis by (simp add: infdist_def)
next
  case False
  then obtain  $a$  where  $a \in A$  by auto
  have  $\text{infdist } x \ A \leq \text{Inf } \{\text{dist } x \ y + \text{dist } y \ a \mid a. a \in A\}$ 
  proof (rule cInf_greatest)
    from  $\langle A \neq \{\} \rangle$  show  $\{\text{dist } x \ y + \text{dist } y \ a \mid a. a \in A\} \neq \{\}$ 
    by simp
  fix  $d$ 
  assume  $d \in \{\text{dist } x \ y + \text{dist } y \ a \mid a. a \in A\}$ 
  then obtain  $a$  where  $d = \text{dist } x \ y + \text{dist } y \ a$   $a \in A$ 
    by auto
  show  $\text{infdist } x \ A \leq d$ 
  unfolding infdist_notempty[OF  $\langle A \neq \{\} \rangle$ ]
  proof (rule cINF_lower2)
    show  $a \in A$  by fact
    show  $\text{dist } x \ a \leq d$ 
    unfolding  $d$  by (rule dist_triangle)
  qed simp
qed
also have  $\dots = \text{dist } x \ y + \text{infdist } y \ A$ 
proof (rule cInf_eq, safe)
  fix  $a$ 
  assume  $a \in A$ 
  then show  $\text{dist } x \ y + \text{infdist } y \ A \leq \text{dist } x \ y + \text{dist } y \ a$ 
    by (auto intro: infdist_le)
next
  fix  $i$ 
  assume  $\text{inf}: \bigwedge d. d \in \{\text{dist } x \ y + \text{dist } y \ a \mid a. a \in A\} \implies i \leq d$ 
  then have  $i - \text{dist } x \ y \leq \text{infdist } y \ A$ 
  unfolding infdist_notempty[OF  $\langle A \neq \{\} \rangle$ ] using  $\langle a \in A \rangle$ 
  by (intro cINF_greatest) (auto simp: field_simps)
  then show  $i \leq \text{dist } x \ y + \text{infdist } y \ A$ 
  by simp
qed
finally show ?thesis by simp
qed

```

lemma *infdist_triangle_abs*: $|infdist\ x\ A - infdist\ y\ A| \leq dist\ x\ y$
by (*metis* (*full_types*) *abs_diff_le_iff_diff_le_eq* *dist_commute* *infdist_triangle*)

lemma *in_closure_iff_infdist_zero*:
assumes $A \neq \{\}$
shows $x \in closure\ A \longleftrightarrow infdist\ x\ A = 0$

proof

assume $x \in closure\ A$

show $infdist\ x\ A = 0$

proof (*rule* *ccontr*)

assume $infdist\ x\ A \neq 0$

with *infdist_nonneg*[of $x\ A$] **have** $infdist\ x\ A > 0$

by *auto*

then have $ball\ x\ (infdist\ x\ A) \cap closure\ A = \{\}$

by (*smt* (*verit*, *best*) $\langle x \in closure\ A \rangle$ *closure_approachableD* *infdist_le*)

then have $x \notin closure\ A$

by (*metis* $\langle 0 < infdist\ x\ A \rangle$ *centre_in_ball_disjoint_iff_not_equal*)

then show *False* **using** $\langle x \in closure\ A \rangle$ **by** *simp*

qed

next

assume $x: infdist\ x\ A = 0$

then obtain a **where** $a \in A$

by *atomize_elim* (*metis* *all_not_in_conv* *assms*)

have *False* **if** $e > 0 \rightarrow (\exists y \in A. dist\ y\ x < e)$ **for** e

proof –

have $infdist\ x\ A \geq e$ **using** $\langle a \in A \rangle$

unfolding *infdist_def* **using** *that*

by (*force* *simp*: *dist_commute* *intro*: *cINF_greatest*)

with $x \langle e > 0 \rangle$ **show** *False* **by** *auto*

qed

then show $x \in closure\ A$

using *closure_approachable* **by** *blast*

qed

lemma *in_closed_iff_infdist_zero*:

assumes $closed\ A\ A \neq \{\}$

shows $x \in A \longleftrightarrow infdist\ x\ A = 0$

proof –

have $x \in closure\ A \longleftrightarrow infdist\ x\ A = 0$

by (*simp* *add*: $\langle A \neq \{\} \rangle$ *in_closure_iff_infdist_zero*)

with *assms* **show** *?thesis* **by** *simp*

qed

lemma *infdist_pos_not_in_closed*:

assumes $closed\ S\ S \neq \{\}\ x \notin S$

shows $infdist\ x\ S > 0$

by (*smt* (*verit*, *best*) *assms* *in_closed_iff_infdist_zero* *infdist_nonneg*)

```

lemma
  infdist_attains_inf:
  fixes X::'a::heine_borel set
  assumes closed X
  assumes X ≠ {}
  obtains x where x ∈ X infdist y X = dist y x
proof -
  have bdd_below (dist y ` X)
  by auto
  from distance_attains_inf[OF assms, of y]
  obtain x where x ∈ X ∧ z. z ∈ X ⇒ dist y x ≤ dist y z by auto
  then have infdist y X = dist y x
  by (metis antisym assms(2) cINF_greatest infdist_def infdist_le)
  with ⟨x ∈ X⟩ show ?thesis ..
qed

```

Every metric space is a T4 space:

```

instance metric_space ⊆ t4_space
proof
  fix S T::'a set assume H: closed S closed T S ∩ T = {}
  consider S = {} | T = {} | S ≠ {} ∧ T ≠ {} by auto
  then show ∃ U V. open U ∧ open V ∧ S ⊆ U ∧ T ⊆ V ∧ U ∩ V = {}
  proof (cases)
    case 1 then show ?thesis by blast
  next
    case 2 then show ?thesis by blast
  next
    case 3
    define U where U = (⋃ x∈S. ball x ((infdist x T)/2))
    have A: open U unfolding U_def by auto
    have infdist x T > 0 if x ∈ S for x
      using H that 3 by (auto intro!: infdist_pos_not_in_closed)
    then have B: S ⊆ U unfolding U_def by auto
    define V where V = (⋃ x∈T. ball x ((infdist x S)/2))
    have C: open V unfolding V_def by auto
    have infdist x S > 0 if x ∈ T for x
      using H that 3 by (auto intro!: infdist_pos_not_in_closed)
    then have D: T ⊆ V unfolding V_def by auto

    have (ball x ((infdist x T)/2)) ∩ (ball y ((infdist y S)/2)) = {} if x ∈ S y ∈
    T for x y
  proof auto
    fix z assume H: dist x z * 2 < infdist x T dist y z * 2 < infdist y S
    have 2 * dist x y ≤ 2 * dist x z + 2 * dist y z
    by metric
    also have ... < infdist x T + infdist y S
    using H by auto
    finally have dist x y < infdist x T ∨ dist x y < infdist y S
    by auto
  end
end

```

```

then show False
  using infdist_le[OF  $\langle x \in S \rangle$ , of y] infdist_le[OF  $\langle y \in T \rangle$ , of x] by (auto
simp add: dist_commute)
qed
then have E:  $U \cap V = \{\}$ 
  unfolding U_def V_def by auto
show ?thesis
  using A B C D E by blast
qed
qed

```

```

lemma tendsto_infdist [tendsto_intros]:
  assumes f:  $(f \longrightarrow l)$  F
  shows  $((\lambda x. \text{infdist } (f x) A) \longrightarrow \text{infdist } l A)$  F
proof (rule tendstoI)
  fix e :: real
  assume  $e > 0$ 
  from tendstoD[OF f this]
  show eventually  $(\lambda x. \text{dist } (\text{infdist } (f x) A) (\text{infdist } l A) < e)$  F
  proof (eventually_elim)
    fix x
    from infdist_triangle[of l A f x] infdist_triangle[of f x A l]
    have  $\text{dist } (\text{infdist } (f x) A) (\text{infdist } l A) \leq \text{dist } (f x) l$ 
      by (simp add: dist_commute dist_real_def)
    also assume  $\text{dist } (f x) l < e$ 
    finally show  $\text{dist } (\text{infdist } (f x) A) (\text{infdist } l A) < e$  .
  qed
qed

```

```

lemma continuous_infdist[continuous_intros]:
  assumes continuous F f
  shows continuous F  $(\lambda x. \text{infdist } (f x) A)$ 
  using assms unfolding continuous_def by (rule tendsto_infdist)

```

```

lemma continuous_on_infdist [continuous_intros]:
  assumes continuous_on S f
  shows continuous_on S  $(\lambda x. \text{infdist } (f x) A)$ 
using assms unfolding continuous_on by (auto intro: tendsto_infdist)

```

```

lemma compact_infdist_le:
  fixes A::'a::heine_borel set
  assumes  $A \neq \{\}$ 
  assumes compact A
  assumes  $e > 0$ 
  shows compact  $\{x. \text{infdist } x A \leq e\}$ 
proof –
  from continuous_closed_vimage[of  $\{0..e\}$   $\lambda x. \text{infdist } x A$ ]
    continuous_infdist[OF continuous_ident, of UNIV A]
  have closed  $\{x. \text{infdist } x A \leq e\}$  by (auto simp: vimage_def infdist_nonneg)

```



```

moreover
from assms obtain  $x0$   $b$  where  $b: \bigwedge x. x \in A \implies \text{dist } x0 \ x \leq b$  closed A
by (auto simp: compact_eq_bounded_closed bounded_def)
{
  fix  $y$ 
  assume  $\text{infdist } y \ A \leq e$ 
  moreover
  from infdist_attains_inf[OF <closed A> <A ≠ {}>, of y]
  obtain  $z$  where  $z \in A$   $\text{infdist } y \ A = \text{dist } y \ z$  by blast
  ultimately
  have  $\text{dist } x0 \ y \leq b + e$  using  $b$  by metric
} then
have bounded  $\{x. \text{infdist } x \ A \leq e\}$ 
by (auto simp: bounded_any_center[where a=x0] intro!: exI[where x=b +
e])
ultimately show compact  $\{x. \text{infdist } x \ A \leq e\}$ 
by (simp add: compact_eq_bounded_closed)
qed

```

4.22 Separation between Points and Sets

proposition *separate_point_closed*:

```

fixes  $S :: 'a::\text{heine\_borel set}$ 
assumes closed S and a ∉ S
shows  $\exists d > 0. \forall x \in S. d \leq \text{dist } a \ x$ 
by (metis assms distance_attains_inf empty_iff gt_ex zero_less_dist_iff)

```

proposition *separate_compact_closed*:

```

fixes  $S \ T :: 'a::\text{heine\_borel set}$ 
assumes compact S
and  $T: \text{closed } T \ S \cap T = \{\}$ 
shows  $\exists d > 0. \forall x \in S. \forall y \in T. d \leq \text{dist } x \ y$ 
proof cases
assume  $S \neq \{\} \wedge T \neq \{\}$ 
then have  $S \neq \{\} \ T \neq \{\}$  by auto
let  $?inf = \lambda x. \text{infdist } x \ T$ 
have continuous_on S ?inf
by (auto intro!: continuous_at_imp_continuous_on continuous_infdist continuous_ident)
then obtain  $x$  where  $x: x \in S \ \forall y \in S. ?inf \ x \leq ?inf \ y$ 
using continuous_attains_inf[OF <compact S> <S ≠ {}>] by auto
then have  $0 < ?inf \ x$ 
using  $T \neq \{\}$  in_closed_iff_infdist_zero by (auto simp: less_le infdist_nonneg)
moreover have  $\forall x' \in S. \forall y \in T. ?inf \ x \leq \text{dist } x' \ y$ 
using  $x$  by (auto intro: order_trans infdist_le)
ultimately show ?thesis by auto
qed (auto intro!: exI[of _ 1])

```

proposition *separate_closed_compact*:

```

fixes  $S T :: 'a::\text{heine\_borel\_set}$ 
assumes  $S: \text{closed } S$ 
and  $T: \text{compact } T$ 
and  $\text{dis}: S \cap T = \{\}$ 
shows  $\exists d > 0. \forall x \in S. \forall y \in T. d \leq \text{dist } x y$ 
by (metis separate_compact_closed[OF T S] dis dist_commute inf_commute)

```

proposition *compact_in_open_separated:*

```

fixes  $A::'a::\text{heine\_borel\_set}$ 
assumes  $A: A \neq \{\}$  compact A
assumes open B
assumes  $A \subseteq B$ 
obtains  $e \text{ where } e > 0 \{x. \text{infdist } x A \leq e\} \subseteq B$ 
proof atomize_elim
have closed (- B) compact A - B  $\cap$  A =  $\{\}$ 
using assms by (auto simp: open_Diff compact_eq_bounded_closed)
from separate_closed_compact[OF this]
obtain  $d'::\text{real}$  where  $d': d' > 0 \wedge x y. x \notin B \implies y \in A \implies d' \leq \text{dist } x y$ 
by auto
define  $d$  where  $d = d' / 2$ 
hence  $d > 0 \ d < d'$  using  $d'$  by auto
with  $d'$  have  $d: \wedge x y. x \notin B \implies y \in A \implies d < \text{dist } x y$ 
by force
show  $\exists e > 0. \{x. \text{infdist } x A \leq e\} \subseteq B$ 
proof (rule ccontr)
assume  $\nexists e. 0 < e \wedge \{x. \text{infdist } x A \leq e\} \subseteq B$ 
with  $\langle d > 0 \rangle$  obtain  $x$  where  $x: \text{infdist } x A \leq d \wedge x \notin B$ 
by auto
then show False
by (metis A compact_eq_bounded_closed infdist_attains_inf x d linorder_not_less)
qed
qed

```

4.23 Uniform Continuity

lemma *uniformly_continuous_onE:*

```

assumes uniformly_continuous_on s f 0 < e
obtains  $d$  where  $d > 0 \wedge x x'. \llbracket x \in s; x' \in s; \text{dist } x' x < d \rrbracket \implies \text{dist } (f x') (f x) < e$ 
using assms
by (auto simp: uniformly_continuous_on_def)

```

lemma *uniformly_continuous_on_sequentially:*

```

uniformly_continuous_on s f  $\longleftrightarrow$  ( $\forall x y. (\forall n. x n \in s) \wedge (\forall n. y n \in s) \wedge$ 
( $\lambda n. \text{dist } (x n) (y n)$ )  $\longrightarrow 0 \longrightarrow (\lambda n. \text{dist } (f(x n)) (f(y n))) \longrightarrow 0$ ) (is
?lhs = ?rhs)
proof
assume ?lhs
{
fix  $x y$ 

```

```

assume  $x: \forall n. x\ n \in s$ 
and  $y: \forall n. y\ n \in s$ 
and  $xy: ((\lambda n. \text{dist } (x\ n) (y\ n)) \longrightarrow 0)$  sequentially
{
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  then obtain  $d$  where  $d > 0$  and  $d: \forall x \in s. \forall x' \in s. \text{dist } x' x < d \longrightarrow \text{dist } (f$ 
 $x') (f\ x) < e$ 
    by (metis  $\langle ?lhs \rangle$  uniformly_continuous_onE)
  obtain  $N$  where  $N: \forall n \geq N. \text{dist } (x\ n) (y\ n) < d$ 
    using  $xy[\text{unfolded } \text{lim\_sequentially\_dist\_norm}]$  and  $\langle d > 0 \rangle$  by auto
  then have  $\exists N. \forall n \geq N. \text{dist } (f\ (x\ n)) (f\ (y\ n)) < e$ 
    using  $d\ x\ y$  by blast
}
then have  $((\lambda n. \text{dist } (f\ (x\ n)) (f\ (y\ n))) \longrightarrow 0)$  sequentially
  unfolding lim_sequentially and dist_real_def by auto
}
then show  $?rhs$  by auto
next
assume  $?rhs$ 
{
  assume  $\neg ?lhs$ 
  then obtain  $e$  where  $e > 0 \ \forall d > 0. \exists x \in s. \exists x' \in s. \text{dist } x' x < d \wedge \neg \text{dist } (f$ 
 $x') (f\ x) < e$ 
    unfolding uniformly_continuous_on_def by auto
  then obtain  $fa$  where fa:
     $\forall x. 0 < x \longrightarrow \text{fst } (fa\ x) \in s \wedge \text{snd } (fa\ x) \in s \wedge \text{dist } (\text{fst } (fa\ x)) (\text{snd } (fa\ x))$ 
 $< x \wedge \neg \text{dist } (f\ (\text{fst } (fa\ x))) (f\ (\text{snd } (fa\ x))) < e$ 
    using choice[of  $\lambda d\ x. d > 0 \longrightarrow \text{fst } x \in s \wedge \text{snd } x \in s \wedge \text{dist } (\text{snd } x) (\text{fst } x)$ 
 $< d \wedge \neg \text{dist } (f\ (\text{snd } x)) (f\ (\text{fst } x)) < e]$ 
    by (auto simp: Bex_def dist_commute)
  define  $x$  where  $x\ n = \text{fst } (fa\ (\text{inverse } (\text{real } n + 1)))$  for  $n$ 
  define  $y$  where  $y\ n = \text{snd } (fa\ (\text{inverse } (\text{real } n + 1)))$  for  $n$ 
  have  $xy_n: \forall n. x\ n \in s \wedge y\ n \in s$ 
    and  $xy_0: \forall n. \text{dist } (x\ n) (y\ n) < \text{inverse } (\text{real } n + 1)$ 
    and  $fx_y: \forall n. \neg \text{dist } (f\ (x\ n)) (f\ (y\ n)) < e$ 
  unfolding  $x\_def$  and  $y\_def$  using  $fa$ 
  by auto
}
fix  $e :: \text{real}$ 
assume  $e > 0$ 
then obtain  $N :: \text{nat}$  where  $N \neq 0$  and  $N: 0 < \text{inverse } (\text{real } N) \wedge \text{inverse}$ 
 $(\text{real } N) < e$ 
  unfolding real_arch_inverse[of  $e$ ] by auto
  then have  $\exists N. \forall n \geq N. \text{dist } (x\ n) (y\ n) < e$ 
  by (smt (verit, ccfv_SIG) inverse_le_imp_le nat_le_real_less of_nat_le_0_iff
 $xy_0$ )
}
then have  $\forall e > 0. \exists N. \forall n \geq N. \text{dist } (f\ (x\ n)) (f\ (y\ n)) < e$ 

```

```

    using ‹?rhs›[THEN spec[where x=x], THEN spec[where x=y]] and xyn
    unfolding lim_sequentially dist_real_def by auto
    then have False using fxy and ‹e>0› by auto
  }
  then show ?lhs
    unfolding uniformly_continuous_on_def by blast
qed

```

4.24 Continuity on a Compact Domain Implies Uniform Continuity

From the proof of the Heine-Borel theorem: Lemma 2 in section 3.7, page 69 of J. C. Burkill and H. Burkill. A Second Course in Mathematical Analysis (CUP, 2002)

lemma *Heine_Borel_lemma*:

```

  assumes compact S and Ssub:  $S \subseteq \bigcup \mathcal{G}$  and opn:  $\bigwedge G. G \in \mathcal{G} \implies \text{open } G$ 
  obtains e where  $0 < e \wedge x \in S \implies \exists G \in \mathcal{G}. \text{ball } x e \subseteq G$ 
proof –
  have False if neg:  $\bigwedge e. 0 < e \implies \exists x \in S. \forall G \in \mathcal{G}. \neg \text{ball } x e \subseteq G$ 
  proof –
    have  $\exists x \in S. \forall G \in \mathcal{G}. \neg \text{ball } x (1 / \text{Suc } n) \subseteq G$  for n
      using neg by simp
    then obtain f where  $\bigwedge n. f n \in S$  and fG:  $\bigwedge G n. G \in \mathcal{G} \implies \neg \text{ball } (f n) (1 / \text{Suc } n) \subseteq G$ 
      by metis
    then obtain l r where  $l \in S$  strict_mono r and to_l:  $(f \circ r) \longrightarrow l$ 
      using ‹compact S› compact_def that by metis
    then obtain G where  $l \in G$   $G \in \mathcal{G}$ 
      using Ssub by auto
    then obtain e where  $0 < e$  and e:  $\bigwedge z. \text{dist } z l < e \implies z \in G$ 
      using opn open_dist by blast
    obtain N1 where N1:  $\bigwedge n. n \geq N1 \implies \text{dist } (f (r n)) l < e/2$ 
      using to_l apply (simp add: lim_sequentially)
      using ‹0 < e› half_gt_zero that by blast
    obtain N2 where N2: of_nat N2 > 2/e
      using reals_Archimedean2 by blast
    obtain x where  $x \in \text{ball } (f (r (\max N1 N2))) (1 / \text{real } (\text{Suc } (r (\max N1 N2))))$  and  $x \notin G$ 
      using fG [OF ‹G ∈ G›, of r (max N1 N2)] by blast
    then have  $\text{dist } (f (r (\max N1 N2))) x < 1 / \text{real } (\text{Suc } (r (\max N1 N2)))$ 
      by simp
    also have  $\dots \leq 1 / \text{real } (\text{Suc } (\max N1 N2))$ 
      by (meson Suc_le_mono ‹strict_mono r› inverse_of_nat_le nat.discI seq_suble)
    also have  $\dots \leq 1 / \text{real } (\text{Suc } N2)$ 
      by (simp add: field_simps)
    also have  $\dots < e/2$ 
      using N2 ‹0 < e› by (simp add: field_simps)

```

finally have $\text{dist } (f \text{ (} r \text{ (} \max N1 N2 \text{))) } x < e/2$.
moreover have $\text{dist } (f \text{ (} r \text{ (} \max N1 N2 \text{))) } l < e/2$
using $N1 \text{ max.cobounded1}$ **by** blast
ultimately have $\text{dist } x \ l < e$
by metric
then show $?thesis$
using $e \langle x \notin G \rangle$ **by** blast
qed
then show $?thesis$
by (meson that)
qed

lemma compact_uniformly_equicontinuous:
assumes $\text{compact } S$
and cont: $\bigwedge x e. \llbracket x \in S; 0 < e \rrbracket$
 $\implies \exists d. 0 < d \wedge$
 $(\forall f \in \mathcal{F}. \forall x' \in S. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) < e)$
and $0 < e$
obtains d **where** $0 < d$
 $\bigwedge f \ x \ x'. \llbracket f \in \mathcal{F}; x \in S; x' \in S; \text{dist } x' \ x < d \rrbracket \implies \text{dist } (f \ x') \ (f \ x)$
 $< e$
proof –
obtain d **where** $d_pos: \bigwedge x e. \llbracket x \in S; 0 < e \rrbracket \implies 0 < d \ x \ e$
and $d_dist : \bigwedge x \ x' e f. \llbracket \text{dist } x' \ x < d \ x \ e; x \in S; x' \in S; 0 < e; f \in \mathcal{F} \rrbracket \implies$
 $\text{dist } (f \ x') \ (f \ x) < e$
using cont **by** metis
let $?G = ((\lambda x. \text{ball } x \ (d \ x \ (e/2)))) \ ' S)$
have $S_{\text{sub}}: S \subseteq \bigcup ?G$
using $\langle 0 < e \rangle$ d_pos **by** auto
then obtain k **where** $0 < k$ **and** $k: \bigwedge x. x \in S \implies \exists G \in ?G. \text{ball } x \ k \subseteq G$
by $(\text{rule Heine_Borel_lemma } [OF \ \langle \text{compact } S \rangle]) \ \text{auto}$
moreover have $\text{dist } (f \ v) \ (f \ u) < e$ **if** $f \in \mathcal{F} \ u \in S \ v \in S \ \text{dist } v \ u < k$ **for** $f \ u \ v$
proof –
obtain G **where** $G \in ?G \ u \in G \ v \in G$
using k **that**
by $(\text{metis } \langle \text{dist } v \ u < k \rangle \ \langle u \in S \rangle \ \langle 0 < k \rangle \ \text{centre_in_ball subsetD dist_commute mem_ball})$
then obtain w **where** $w: \text{dist } w \ u < d \ w \ (e/2) \ \text{dist } w \ v < d \ w \ (e/2) \ w \in S$
by auto
with that d_dist **have** $\text{dist } (f \ w) \ (f \ v) < e/2$
by $(\text{metis } \langle 0 < e \rangle \ \text{dist_commute half_gt_zero})$
moreover
have $\text{dist } (f \ w) \ (f \ u) < e/2$
using that $d_dist \ w$ **by** $(\text{metis } \langle 0 < e \rangle \ \text{dist_commute divide_pos_pos zero_less_numeral})$
ultimately show $?thesis$
using $\text{dist_triangle_half_r}$ **by** blast
qed
ultimately show $?thesis$ **using that** **by** blast

qed

corollary *compact_uniformly_continuous*:

fixes $f :: 'a :: \text{metric_space} \Rightarrow 'b :: \text{metric_space}$

assumes f : *continuous_on S f* and S : *compact S*

shows *uniformly_continuous_on S f*

using f

unfolding *continuous_on_iff_uniformly_continuous_on_def*

by (*force intro: compact_uniformly_equicontinuous [OF S, of {f}]*)

4.25 Theorems relating continuity and uniform continuity to closures

lemma *continuous_on_closure*:

continuous_on (closure S) f \longleftrightarrow

$(\forall x e. x \in \text{closure } S \wedge 0 < e$

$\longrightarrow (\exists d. 0 < d \wedge (\forall y. y \in S \wedge \text{dist } y \ x < d \longrightarrow \text{dist } (f \ y) \ (f \ x) < e))$)

(is *?lhs = ?rhs*)

proof

assume *?lhs* then show *?rhs*

unfolding *continuous_on_iff* by (*metis Un_iff closure_def*)

next

assume R [*rule_format*]: *?rhs*

show *?lhs*

proof

fix x and $e::\text{real}$

assume $0 < e$ and x : $x \in \text{closure } S$

obtain $\delta::\text{real}$ where $\delta > 0$

and δ : $\bigwedge y. \llbracket y \in S; \text{dist } y \ x < \delta \rrbracket \Longrightarrow \text{dist } (f \ y) \ (f \ x) < e/2$

using R [*of x e/2*] $\langle 0 < e \rangle$ x by *auto*

have $\text{dist } (f \ y) \ (f \ x) \leq e$ if y : $y \in \text{closure } S$ and dyx : $\text{dist } y \ x < \delta/2$ for y

proof –

obtain $\delta'::\text{real}$ where $\delta' > 0$

and δ' : $\bigwedge z. \llbracket z \in S; \text{dist } z \ y < \delta' \rrbracket \Longrightarrow \text{dist } (f \ z) \ (f \ y) < e/2$

using R [*of y e/2*] $\langle 0 < e \rangle$ y by *auto*

obtain z where $z \in S$ and z : $\text{dist } z \ y < \min \delta' \delta / 2$

using *closure_approachable y*

by (*metis* $\langle 0 < \delta' \rangle \langle 0 < \delta \rangle$ *divide_pos_pos min_less_iff conj zero_less_numeral*)

have $\text{dist } (f \ z) \ (f \ y) < e/2$

using δ' [*OF* $\langle z \in S \rangle$] $z \langle 0 < \delta' \rangle$ by *metric*

moreover have $\text{dist } (f \ z) \ (f \ x) < e/2$

using δ [*OF* $\langle z \in S \rangle$] $z \ dyx$ by *metric*

ultimately show *?thesis*

by *metric*

qed

then show $\exists d > 0. \forall x' \in \text{closure } S. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) \leq e$

by (*rule_tac x = $\delta/2$ in exI*) (*simp add: $\langle \delta > 0 \rangle$*)

qed

qed

lemma *continuous_on_closure_sequentially*:

fixes $f :: 'a::metric_space \Rightarrow 'b::metric_space$

shows

$continuous_on (closure\ S)\ f \longleftrightarrow$

$(\forall x\ a.\ a \in closure\ S \wedge (\forall n.\ x\ n \in S) \wedge x \longrightarrow a \longrightarrow (f \circ x) \longrightarrow f\ a)$

(is $?lhs = ?rhs$)

proof -

have $continuous_on (closure\ S)\ f \longleftrightarrow$

$(\forall x \in closure\ S.\ continuous\ (at\ x\ within\ S)\ f)$

by (force simp: *continuous_on_closure_continuous_within_eps_delta*)

also have $\dots = ?rhs$

by (force simp: *continuous_within_sequentially*)

finally show $?thesis$.

qed

lemma *uniformly_continuous_on_closure*:

fixes $f :: 'a::metric_space \Rightarrow 'b::metric_space$

assumes $ucont: uniformly_continuous_on\ S\ f$

and $cont: continuous_on (closure\ S)\ f$

shows $uniformly_continuous_on (closure\ S)\ f$

unfolding *uniformly_continuous_on_def*

proof (intro allI impI)

fix $e::real$

assume $0 < e$

then obtain $d::real$

where $d > 0$

and $d: \bigwedge x\ x'. \llbracket x \in S; x' \in S; dist\ x'\ x < d \rrbracket \implies dist\ (f\ x')\ (f\ x) < e/3$

using $ucont$ [*unfolded_uniformly_continuous_on_def*, *rule_format*, of $e/3$] by

auto

show $\exists d > 0. \forall x \in closure\ S. \forall x' \in closure\ S. dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e$

proof (rule exI [where $x=d/3$], clarsimp simp: $\langle d > 0 \rangle$)

fix $x\ y$

assume $x: x \in closure\ S$ and $y: y \in closure\ S$ and $dyx: dist\ y\ x * 3 < d$

obtain $d1::real$ where $d1 > 0$

and $d1: \bigwedge w. \llbracket w \in closure\ S; dist\ w\ x < d1 \rrbracket \implies dist\ (f\ w)\ (f\ x) < e/3$

using $cont$ [*unfolded_continuous_on_iff*, *rule_format*, of $x\ e/3$] $\langle 0 < e \rangle x$

by auto

obtain x' where $x' \in S$ and $x': dist\ x'\ x < min\ d1\ (d / 3)$

using *closure_approachable* [of $x\ S$]

by (metis $\langle 0 < d1 \rangle \langle 0 < d \rangle divide_pos_pos\ min_less_iff_conj\ x\ zero_less_numeral$)

obtain $d2::real$ where $d2 > 0$

and $d2: \forall w \in closure\ S. dist\ w\ y < d2 \longrightarrow dist\ (f\ w)\ (f\ y) < e/3$

using $cont$ [*unfolded_continuous_on_iff*, *rule_format*, of $y\ e/3$] $\langle 0 < e \rangle y$

by auto

obtain y' where $y' \in S$ and $y': dist\ y'\ y < min\ d2\ (d / 3)$

using *closure_approachable* [of $y\ S$]

```

by (metis <0 < d2> <0 < d> divide_pos_pos min_less_iff_conj y zero_less_numeral)
have dist x' x < d/3 using x' by auto
then have dist x' y' < d
  using dya y' by metric
then have dist (f x') (f y') < e/3
  by (rule d [OF <y' ∈ S> <x' ∈ S>])
moreover have dist (f x') (f x) < e/3 using <x' ∈ S> closure_subset x' d1
  by (simp add: closure_def)
moreover have dist (f y') (f y) < e/3 using <y' ∈ S> closure_subset y' d2
  by (simp add: closure_def)
ultimately show dist (f y) (f x) < e by metric
qed
qed

```

lemma *uniformly_continuous_on_extension_at_closure*:

```

fixes f::'a::metric_space ⇒ 'b::complete_space
assumes uc: uniformly_continuous_on X f
assumes x ∈ closure X
obtains l where (f ⟶ l) (at x within X)
proof -
  from assms obtain xs where xs: xs ⟶ x ∧ n. xs n ∈ X
    by (auto simp: closure_sequential)

  from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF xs]
  obtain l where l: (λn. f (xs n)) ⟶ l
    by atomize_elim (simp only: convergent_eq_Cauchy)

  have (f ⟶ l) (at x within X)
  proof (safe intro!: Lim_within_LIMSEQ)
    fix xs'
    assume ∀n. xs' n ≠ x ∧ xs' n ∈ X
    and xs': xs' ⟶ x
    then have xs' n ≠ x xs' n ∈ X for n by auto

    from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF
  <xs' ⟶ x> <xs' _ ∈ X>]
  obtain l' where l': (λn. f (xs' n)) ⟶ l'
    by atomize_elim (simp only: convergent_eq_Cauchy)

  show (λn. f (xs' n)) ⟶ l
  proof (rule tendstoI)
    fix e::real assume e > 0
    define e' where e' ≡ e/2
    have e' > 0 using <e > 0> by (simp add: e'_def)

    have ∀F n in sequentially. dist (f (xs n)) l < e'
      by (simp add: <0 < e'> l tendstoD)
    moreover
    from uc[unfolded uniformly_continuous_on_def, rule_format, OF <e' > 0>]

```



```

obtain  $d$  where  $d: d > 0 \wedge x x'. x \in X \implies x' \in X \implies \text{dist } x x' < d \implies$ 
 $\text{dist } (f x) (f x') < e'$ 
by auto
have  $\forall_F n$  in sequentially.  $\text{dist } (x s n) (x s' n) < d$ 
by (auto intro!:  $\langle 0 < d \rangle$  order_tendstoD tendsto_eq_intros  $x s x s'$ )
ultimately
show  $\forall_F n$  in sequentially.  $\text{dist } (f (x s' n)) l < e$ 
proof eventually_elim
case (elim  $n$ )
have  $\text{dist } (f (x s' n)) l \leq \text{dist } (f (x s n)) (f (x s' n)) + \text{dist } (f (x s n)) l$ 
by metric
also have  $\text{dist } (f (x s n)) (f (x s' n)) < e'$ 
by (auto intro!:  $d x s \langle x s' \_ \in \_ \rangle$  elim)
also note  $\langle \text{dist } (f (x s n)) l < e' \rangle$ 
also have  $e' + e' = e$  by (simp add: e'_def)
finally show ?case by simp
qed
qed
qed
thus ?thesis ..
qed

```

lemma *uniformly_continuous_on_extension_on_closure:*

```

fixes  $f::'a::\text{metric\_space} \Rightarrow 'b::\text{complete\_space}$ 
assumes  $uc: \text{uniformly\_continuous\_on } X f$ 
obtains  $g$  where  $\text{uniformly\_continuous\_on } (\text{closure } X) g \wedge x. x \in X \implies f x =$ 
 $g x$ 
 $\wedge Y h x. X \subseteq Y \implies Y \subseteq \text{closure } X \implies \text{continuous\_on } Y h \implies (\wedge x. x \in X$ 
 $\implies f x = h x) \implies x \in Y \implies h x = g x$ 
proof -
from  $uc$  have  $\text{cont\_f}: \text{continuous\_on } X f$ 
by (simp add: uniformly_continuous_imp_continuous)
obtain  $y$  where  $y: (f \longrightarrow y x)$  (at  $x$  within  $X$ ) if  $x \in \text{closure } X$  for  $x$ 
using  $\text{uniformly\_continuous\_on\_extension\_at\_closure}$  [OF assms] by meson
let  $?g = \lambda x. \text{if } x \in X \text{ then } f x \text{ else } y x$ 

have  $\text{uniformly\_continuous\_on } (\text{closure } X) ?g$ 
unfolding  $\text{uniformly\_continuous\_on\_def}$ 
proof safe
fix  $e::\text{real}$  assume  $e > 0$ 
define  $e'$  where  $e' \equiv e / 3$ 
have  $e' > 0$  using  $\langle e > 0 \rangle$  by (simp add: e'_def)
from  $uc$  [unfolded uniformly_continuous_on_def, rule_format, OF  $\langle 0 < e' \rangle$ ]
obtain  $d$  where  $d > 0$  and  $d: \wedge x x'. x \in X \implies x' \in X \implies \text{dist } x' x < d$ 
 $\implies \text{dist } (f x') (f x) < e'$ 
by auto
define  $d'$  where  $d' = d / 3$ 
have  $d' > 0$  using  $\langle d > 0 \rangle$  by (simp add: d'_def)
show  $\exists d > 0. \forall x \in \text{closure } X. \forall x' \in \text{closure } X. \text{dist } x' x < d \longrightarrow \text{dist } (?g x') (?g$ 

```

```

x) < e
  proof (safe intro!: exI[where x=d'] ⟨d' > 0⟩)
    fix x x' assume x: x ∈ closure X and x': x' ∈ closure X and dist: dist x' x
    < d'
    then obtain xs xs' where xs: xs ⟶ x ∧ n. xs n ∈ X
      and xs': xs' ⟶ x' ∧ n. xs' n ∈ X
      by (auto simp: closure_sequential)
    have ∀_F n in sequentially. dist (xs' n) x' < d'
      and ∀_F n in sequentially. dist (xs n) x < d'
      by (auto intro!: ⟨0 < d'⟩ order_tendstoD tendsto_eq_intros xs xs')
    moreover
    have (λx. f (xs x)) ⟶ y x if x ∈ closure X x ∉ X xs ⟶ x ∧ n. xs n
    ∈ X for xs x
      using that not_eventuallyD
      by (force intro!: filterlim_compose[OF y[OF ⟨x ∈ closure X⟩]] simp: filter-
lim_at)
    then have (λx. f (xs' x)) ⟶ ?g x' (λx. f (xs x)) ⟶ ?g x
      using x x'
      by (auto intro!: continuous_on_tendsto_compose[OF cont_f] simp: xs' xs)
    then have ∀_F n in sequentially. dist (f (xs' n)) (?g x') < e'
      ∀_F n in sequentially. dist (f (xs n)) (?g x) < e'
      by (auto intro!: ⟨0 < e'⟩ order_tendstoD tendsto_eq_intros)
    ultimately
    have ∀_F n in sequentially. dist (?g x') (?g x) < e
    proof eventually_elim
      case (elim n)
      have dist (?g x') (?g x) ≤
        dist (f (xs' n)) (?g x') + dist (f (xs' n)) (f (xs n)) + dist (f (xs n)) (?g x)
        by (metis add.commute add_le_cancel_left dist_commute dist_triangle
dist_triangle_le)
      also
      from ⟨dist (xs' n) x' < d'⟩ ⟨dist x' x < d'⟩ ⟨dist (xs n) x < d'⟩
      have dist (xs' n) (xs n) < d unfolding d'_def by metric
      with ⟨xs _ ∈ X⟩ ⟨xs' _ ∈ X⟩ have dist (f (xs' n)) (f (xs n)) < e'
        by (rule d)
      also note ⟨dist (f (xs' n)) (?g x') < e'⟩
      also note ⟨dist (f (xs n)) (?g x) < e'⟩
      finally show ?case by (simp add: e'_def)
    qed
    then show dist (?g x') (?g x) < e by simp
  qed
  moreover have f x = ?g x if x ∈ X for x using that by simp
  moreover
  {
    fix Y h x
    assume Y: x ∈ Y X ⊆ Y Y ⊆ closure X and cont_h: continuous_on Y h
      and extension: (∧x. x ∈ X ⟹ f x = h x)
  }

```

```

    assume  $x \notin X$ 
    have  $x \in \text{closure } X$  using  $Y$  by auto
    then obtain  $xs$  where  $xs: xs \longrightarrow x \wedge n. xs\ n \in X$ 
      by (auto simp: closure_sequential)
    from continuous_on_tendsto_compose[OF cont_h xs(1)] xs(2)  $Y$ 
    have  $hx: (\lambda x. f (xs\ x)) \longrightarrow h\ x$ 
      by (auto simp: subsetD extension)
    then have  $(\lambda x. f (xs\ x)) \longrightarrow y\ x$ 
      using  $\langle x \notin X \rangle$  not_eventuallyD xs(2)
      by (force intro: filterlim_compose[OF y[OF  $\langle x \in \text{closure } X \rangle$ ]] simp: filter-
lim_at xs)
    with  $hx$  have  $h\ x = y\ x$  by (rule LIMSEQ_unique)
  } then
  have  $h\ x = ?g\ x$ 
    using extension by auto
  }
  ultimately show ?thesis ..
qed

```

lemma *bounded_uniformly_continuous_image:*

fixes $f :: 'a :: \text{heine_borel} \Rightarrow 'b :: \text{heine_borel}$

assumes *uniformly_continuous_on* S f *bounded* S

shows *bounded*($f\ 'S$)

by (*metis* (*no_types*, *lifting*) *assms* *bounded_closure_image* *compact_closure* *compact_continuous_image* *compact_eq_bounded_closed_image_cong* *uniformly_continuous_imp_continuous* *uniformly_continuous_on_extension_on_closure*)

4.26 With Abstract Topology (TODO: move and remove dependency?)

lemma *openin_contains_ball:*

openin (*top_of_set* T) $S \longleftrightarrow$

$S \subseteq T \wedge (\forall x \in S. \exists e. 0 < e \wedge \text{ball } x\ e \cap T \subseteq S)$

(**is** ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

by (*metis* *IntD2* *Int_commute* *Int_lower1* *Int_mono* *inf.idem* *openE* *openin_open*)

next

assume ?rhs

then show ?lhs

by (*smt* (*verit*) *open_ball* *Int_commute* *Int_iff* *centre_in_ball* *in_mono* *openin_open_Int* *openin_subopen*)

qed

lemma *openin_contains_cball:*

openin (*top_of_set* T) $S \longleftrightarrow$

$S \subseteq T \wedge (\forall x \in S. \exists e. 0 < e \wedge \text{cball } x\ e \cap T \subseteq S)$

```

    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (force simp add: openin_contains_ball intro: exI [where x=_/2])
next
  assume ?rhs
  then show ?lhs
    by (force simp add: openin_contains_ball)
qed

```

4.27 Closed Nest

Bounded closed nest property (proof does not use Heine-Borel)

```

lemma bounded_closed_nest:
  fixes  $S :: nat \Rightarrow ('a::heine_borel) set$ 
  assumes  $\bigwedge n. closed (S n)$ 
    and  $\bigwedge n. S n \neq \{\}$ 
    and  $\bigwedge m n. m \leq n \implies S n \subseteq S m$ 
    and  $bounded (S 0)$ 
  obtains  $a$  where  $\bigwedge n. a \in S n$ 
proof -
  from  $assms(2)$  obtain  $x$  where  $x: \forall n. x n \in S n$ 
    using  $choice[of \lambda n x. x \in S n]$  by auto
  from  $assms(4,1)$  have  $seq\_compact (S 0)$ 
    by ( $simp$  add:  $bounded\_closed\_imp\_seq\_compact$ )
  then obtain  $l r$  where  $lr: l \in S 0$   $strict\_mono r (x \circ r) \longrightarrow l$ 
    using  $x$  and  $assms(3)$  unfolding  $seq\_compact\_def$  by blast
  have  $\forall n. l \in S n$ 
proof
  fix  $n :: nat$ 
  have  $closed (S n)$ 
    using  $assms(1)$  by simp
  moreover have  $\forall i. (x \circ r) i \in S i$ 
    using  $x$  and  $assms(3)$  and  $lr(2)$  [THEN seq_suble] by auto
  then have  $\forall i. (x \circ r) (i + n) \in S n$ 
    using  $assms(3)$  by ( $fast$  intro!:  $le\_add2$ )
  moreover have  $(\lambda i. (x \circ r) (i + n)) \longrightarrow l$ 
    using  $lr(3)$  by (rule  $LIMSEQ\_ignore\_initial\_segment$ )
  ultimately show  $l \in S n$ 
    by (metis  $closed\_sequentially$ )
qed
  then show ?thesis
    using that by blast
qed

```

Decreasing case does not even need compactness, just completeness.

```

lemma decreasing_closed_nest:

```

```

fixes  $S :: \text{nat} \Rightarrow ('a::\text{complete\_space}) \text{ set}$ 
assumes  $\bigwedge n. \text{closed } (S\ n)$ 
 $\bigwedge n. S\ n \neq \{\}$ 
 $\bigwedge m\ n. m \leq n \implies S\ n \subseteq S\ m$ 
 $\bigwedge e. e > 0 \implies \exists n. \forall x \in S\ n. \forall y \in S\ n. \text{dist } x\ y < e$ 
obtains  $a$  where  $\bigwedge n. a \in S\ n$ 
proof -
obtain  $t$  where  $t: \forall n. t\ n \in S\ n$ 
by (meson assms(2) equals0I)
{
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  then obtain  $N$  where  $N: \forall x \in S\ N. \forall y \in S\ N. \text{dist } x\ y < e$ 
  using assms(4) by blast
  {
    fix  $m\ n :: \text{nat}$ 
    assume  $N \leq m \wedge N \leq n$ 
    then have  $t\ m \in S\ N\ t\ n \in S\ N$ 
    using assms(3) t unfolding subset_eq by blast+
    then have  $\text{dist } (t\ m)\ (t\ n) < e$ 
    using  $N$  by auto
  }
  then have  $\exists N. \forall m\ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (t\ m)\ (t\ n) < e$ 
  by auto
}
then have Cauchy  $t$ 
by (metis metric_CauchyI)
then obtain  $l$  where  $l: (t \longrightarrow l)$  sequentially
using complete_UNIV unfolding complete_def by auto
{ fix  $n :: \text{nat}$ 
  { fix  $e :: \text{real}$ 
    assume  $e > 0$ 
    then obtain  $N :: \text{nat}$  where  $N: \forall n \geq N. \text{dist } (t\ n)\ l < e$ 
    using  $l[\text{unfolded } \text{lim\_sequentially}]$  by auto
    have  $t\ (\text{max } n\ N) \in S\ n$ 
    by (meson assms(3) contra_subsetD max.cobounded1  $t$ )
    then have  $\exists y \in S\ n. \text{dist } y\ l < e$ 
    using  $N$  max.cobounded2 by blast
  }
  then have  $l \in S\ n$ 
  using closed_approachable[of S n l] assms(1) by auto
}
then show ?thesis
using that by blast
qed

```

Strengthen it to the intersection actually being a singleton.

```

lemma decreasing_closed_nest_sing:
fixes  $S :: \text{nat} \Rightarrow 'a::\text{complete\_space} \text{ set}$ 

```

```

assumes  $\bigwedge n. \text{closed}(S\ n)$ 
           $\bigwedge n. S\ n \neq \{\}$ 
           $\bigwedge m\ n. m \leq n \implies S\ n \subseteq S\ m$ 
           $\bigwedge e. e > 0 \implies \exists n. \forall x \in (S\ n). \forall y \in (S\ n). \text{dist}\ x\ y < e$ 
shows  $\exists a. \bigcap(\text{range}\ S) = \{a\}$ 
proof -
obtain  $a$  where  $a: \forall n. a \in S\ n$ 
  using decreasing_closed_nest[of  $S$ ] using assms by auto
  { fix  $b$ 
    assume  $b: b \in \bigcap(\text{range}\ S)$ 
    { fix  $e :: \text{real}$ 
      assume  $e > 0$ 
      then have  $\text{dist}\ a\ b < e$ 
        using assms(4) and  $b$  and  $a$  by blast
      }
    }
  then have  $\text{dist}\ a\ b = 0$ 
    by (metis dist_eq_0_iff dist_nz less_le)
  }
with  $a$  have  $\bigcap(\text{range}\ S) = \{a\}$ 
  unfolding image_def by auto
then show ?thesis ..
qed

```

4.28 Making a continuous function avoid some value in a neighbourhood

```

lemma continuous_within_avoid:
  fixes  $f :: 'a::\text{metric\_space} \Rightarrow 'b::t1\_space$ 
  assumes continuous (at  $x$  within  $s$ )  $f$ 
  and  $f\ x \neq a$ 
  shows  $\exists e > 0. \forall y \in s. \text{dist}\ x\ y < e \implies f\ y \neq a$ 
proof -
obtain  $U$  where open  $U$  and  $f\ x \in U$  and  $a \notin U$ 
  using t1_space [OF  $\langle f\ x \neq a \rangle$ ] by fast
  have ( $f \longrightarrow f\ x$ ) (at  $x$  within  $s$ )
  using assms(1) by (simp add: continuous_within)
  then have eventually ( $\lambda y. f\ y \in U$ ) (at  $x$  within  $s$ )
  using  $\langle \text{open}\ U \rangle$  and  $\langle f\ x \in U \rangle$ 
  unfolding tendsto_def by fast
  then have eventually ( $\lambda y. f\ y \neq a$ ) (at  $x$  within  $s$ )
  using  $\langle a \notin U \rangle$  by (fast elim: eventually_mono)
  then show ?thesis
  using  $\langle f\ x \neq a \rangle$  by (auto simp: dist_commute eventually_at)
qed

```

```

lemma continuous_at_avoid:
  fixes  $f :: 'a::\text{metric\_space} \Rightarrow 'b::t1\_space$ 
  assumes continuous (at  $x$ )  $f$ 

```

```

  and  $f x \neq a$ 
  shows  $\exists e > 0. \forall y. \text{dist } x y < e \longrightarrow f y \neq a$ 
  using assms continuous_within_avoid[of  $x UNIV f a$ ] by simp

```

```

lemma continuous_on_avoid:
  fixes  $f :: 'a::\text{metric\_space} \Rightarrow 'b::t1\_space$ 
  assumes continuous_on  $s f$ 
  and  $x \in s$ 
  and  $f x \neq a$ 
  shows  $\exists e > 0. \forall y \in s. \text{dist } x y < e \longrightarrow f y \neq a$ 
  using assms(1)[unfolded continuous_on_eq_continuous_within, THEN bspec][where
 $x=x$ ],
  OF assms(2)] continuous_within_avoid[of  $x s f a$ ]
  using assms(3)
  by auto

```

```

lemma continuous_on_open_avoid:
  fixes  $f :: 'a::\text{metric\_space} \Rightarrow 'b::t1\_space$ 
  assumes continuous_on  $s f$ 
  and open  $s$ 
  and  $x \in s$ 
  and  $f x \neq a$ 
  shows  $\exists e > 0. \forall y. \text{dist } x y < e \longrightarrow f y \neq a$ 
  using assms(1)[unfolded continuous_on_eq_continuous_at][OF assms(2)], THEN
bspec [where  $x=x$ ], OF assms(3)]
  using continuous_at_avoid[of  $x f a$ ] assms(4)
  by auto

```

4.29 Consequences for Real Numbers

```

lemma closed_contains_Inf:
  fixes  $S :: \text{real set}$ 
  shows  $S \neq \{\} \Longrightarrow \text{bdd\_below } S \Longrightarrow \text{closed } S \Longrightarrow \text{Inf } S \in S$ 
  by (metis closure_contains_Inf closure_closed)

```

```

lemma closed_subset_contains_Inf:
  fixes  $A C :: \text{real set}$ 
  shows  $\text{closed } C \Longrightarrow A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow \text{bdd\_below } A \Longrightarrow \text{Inf } A \in C$ 
  by (metis closure_contains_Inf closure_minimal_subset_eq)

```

```

lemma closed_contains_Sup:
  fixes  $S :: \text{real set}$ 
  shows  $S \neq \{\} \Longrightarrow \text{bdd\_above } S \Longrightarrow \text{closed } S \Longrightarrow \text{Sup } S \in S$ 
  by (subst closure_closed[symmetric], assumption, rule closure_contains_Sup)

```

```

lemma closed_subset_contains_Sup:
  fixes  $A C :: \text{real set}$ 
  shows  $\text{closed } C \Longrightarrow A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow \text{bdd\_above } A \Longrightarrow \text{Sup } A \in C$ 
  by (metis closure_contains_Sup closure_minimal_subset_eq)

```

lemma *atLeastAtMost_subset_contains_Inf*:

fixes $A :: \text{real set}$ **and** $a\ b :: \text{real}$
shows $A \neq \{\}$ $\implies a \leq b \implies A \subseteq \{a..b\} \implies \text{Inf } A \in \{a..b\}$
by (*rule closed_subset_contains_Inf*)
(auto intro: closed_real_atLeastAtMost intro!: bdd_belowI[of A a])

lemma *bounded_real*: $\text{bounded } (S :: \text{real set}) \longleftrightarrow (\exists a. \forall x \in S. |x| \leq a)$
by (*simp add: bounded_iff*)

lemma *bounded_imp_bdd_above*: $\text{bounded } S \implies \text{bdd_above } (S :: \text{real set})$
by (*auto simp: bounded_def bdd_above_def dist_real_def*)
(metis abs_le_D1 abs_minus_commute diff_le_eq)

lemma *bounded_imp_bdd_below*: $\text{bounded } S \implies \text{bdd_below } (S :: \text{real set})$
by (*auto simp: bounded_def bdd_below_def dist_real_def*)
(metis abs_le_D1 add.commute diff_le_eq)

lemma *bounded_norm_le_SUP_norm*:
 $\text{bounded } (\text{range } f) \implies \text{norm } (f\ x) \leq (\text{SUP } x. \text{norm } (f\ x))$
by (*auto intro!: cSUP_upper bounded_imp_bdd_above simp: bounded_norm_comp*)

lemma *bounded_has_Sup*:
fixes $S :: \text{real set}$
assumes *bounded S*
and $S \neq \{\}$
shows $\forall x \in S. x \leq \text{Sup } S$
and $\forall b. (\forall x \in S. x \leq b) \longrightarrow \text{Sup } S \leq b$
proof
show $\forall b. (\forall x \in S. x \leq b) \longrightarrow \text{Sup } S \leq b$
using *assms* **by** (*metis cSup_least*)
qed (*metis cSup_upper assms(1) bounded_imp_bdd_above*)

lemma *Sup_insert*:
fixes $S :: \text{real set}$
shows $\text{bounded } S \implies \text{Sup } (\text{insert } x\ S) = (\text{if } S = \{\} \text{ then } x \text{ else } \max\ x\ (\text{Sup } S))$
by (*auto simp: bounded_imp_bdd_above sup_max cSup_insert_If*)

lemma *bounded_has_Inf*:
fixes $S :: \text{real set}$
assumes *bounded S*
and $S \neq \{\}$
shows $\forall x \in S. x \geq \text{Inf } S$
and $\forall b. (\forall x \in S. x \geq b) \longrightarrow \text{Inf } S \geq b$
proof
show $\forall b. (\forall x \in S. x \geq b) \longrightarrow \text{Inf } S \geq b$
using *assms* **by** (*metis cInf_greatest*)
qed (*metis cInf_lower assms(1) bounded_imp_bdd_below*)

lemma *Inf_insert*:
fixes $S :: \text{real set}$
shows $\text{bounded } S \implies \text{Inf } (\text{insert } x S) = (\text{if } S = \{\} \text{ then } x \text{ else } \min x (\text{Inf } S))$
by (*auto simp: bounded_imp_bdd_below inf_min cInf_insert_If*)

lemma *open_real*:
fixes $s :: \text{real set}$
shows $\text{open } s \iff (\forall x \in s. \exists e > 0. \forall x'. |x' - x| < e \implies x' \in s)$
unfolding *open_dist dist_norm* **by** *simp*

lemma *islimpt_approachable_real*:
fixes $s :: \text{real set}$
shows $x \text{ islimpt } s \iff (\forall e > 0. \exists x' \in s. x' \neq x \wedge |x' - x| < e)$
unfolding *islimpt_approachable dist_norm* **by** *simp*

lemma *closed_real*:
fixes $s :: \text{real set}$
shows $\text{closed } s \iff (\forall x. (\forall e > 0. \exists x' \in s. x' \neq x \wedge |x' - x| < e) \implies x \in s)$
unfolding *closed_limpt islimpt_approachable dist_norm* **by** *simp*

lemma *continuous_at_real_range*:
fixes $f :: 'a::\text{real_normed_vector} \Rightarrow \text{real}$
shows $\text{continuous } (at\ x) f \iff (\forall e > 0. \exists d > 0. \forall x'. \text{norm}(x' - x) < d \implies |f\ x' - f\ x| < e)$
by (*metis (mono_tags, opaque_lifting) LIM_eq continuous_within norm_eq_zero real_norm_def right_minus_eq*)

lemma *continuous_on_real_range*:
fixes $f :: 'a::\text{real_normed_vector} \Rightarrow \text{real}$
shows $\text{continuous_on } s f \iff$
 $(\forall x \in s. \forall e > 0. \exists d > 0. (\forall x' \in s. \text{norm}(x' - x) < d \implies |f\ x' - f\ x| < e))$
unfolding *continuous_on_iff dist_norm* **by** *simp*

lemma *continuous_on_closed_Collect_le*:
fixes $f\ g :: 'a::\text{topological_space} \Rightarrow \text{real}$
assumes $f: \text{continuous_on } s f$ **and** $g: \text{continuous_on } s g$ **and** $s: \text{closed } s$
shows $\text{closed } \{x \in s. f\ x \leq g\ x\}$
proof –
have $\text{closed } ((\lambda x. g\ x - f\ x) - \{0..\} \cap s)$
using *closed_real_atLeast continuous_on_diff [OF g f]*
by (*simp add: continuous_on_closed_vimage [OF s]*)
also have $((\lambda x. g\ x - f\ x) - \{0..\} \cap s) = \{x \in s. f\ x \leq g\ x\}$
by *auto*
finally show *?thesis* .
qed

lemma *continuous_le_on_closure*:
fixes $a::\text{real}$
assumes $f: \text{continuous_on } (\text{closure } s) f$

```

    and x: x ∈ closure(s)
    and xlo:  $\bigwedge x. x \in s \implies f(x) \leq a$ 
    shows f(x) ≤ a
    using image_closure_subset [OF f, where T = {x. x ≤ a}] assms
    continuous_on_closed_Collect_le[of UNIV  $\lambda x. x \lambda x. a$ ]
    by auto

```

```

lemma continuous_ge_on_closure:
  fixes a::real
  assumes f: continuous_on (closure s) f
    and x: x ∈ closure(s)
    and xlo:  $\bigwedge x. x \in s \implies f(x) \geq a$ 
  shows f(x) ≥ a
  using image_closure_subset [OF f, where T = {x. a ≤ x}] assms
  continuous_on_closed_Collect_le[of UNIV  $\lambda x. a \lambda x. x$ ]
  by auto

```

4.30 The infimum of the distance between two sets

definition *setdist* :: 'a::metric_space set ⇒ 'a set ⇒ real **where**

```

  setdist s t ≡
    (if s = {} ∨ t = {} then 0
     else Inf {dist x y | x y. x ∈ s ∧ y ∈ t})

```

```

lemma setdist_empty1 [simp]: setdist {} t = 0
  by (simp add: setdist_def)

```

```

lemma setdist_empty2 [simp]: setdist t {} = 0
  by (simp add: setdist_def)

```

```

lemma setdist_pos_le [simp]: 0 ≤ setdist s t
  by (auto simp: setdist_def ex_in_conv [symmetric] intro: cInf_greatest)

```

```

lemma le_setdistI:
  assumes s ≠ {} t ≠ {}  $\bigwedge x y. [x \in s; y \in t] \implies d \leq \text{dist } x y$ 
  shows d ≤ setdist s t
  using assms
  by (auto simp: setdist_def Set.ex_in_conv [symmetric] intro: cInf_greatest)

```

```

lemma setdist_le_dist:  $[x \in s; y \in t] \implies \text{setdist } s t \leq \text{dist } x y$ 
  unfolding setdist_def
  by (auto intro!: bdd_belowI [where m=0] cInf_lower)

```

```

lemma le_setdist_iff:
  d ≤ setdist S T  $\longleftrightarrow$ 
  ( $\forall x \in S. \forall y \in T. d \leq \text{dist } x y$ )  $\wedge$  (S = {} ∨ T = {}  $\longrightarrow$  d ≤ 0)
  by (smt (verit) le_setdistI setdist_def setdist_le_dist)

```

```

lemma setdist_ltE:

```

assumes $\text{setdist } S \ T < b$ $S \neq \{\}$ $T \neq \{\}$
obtains $x \ y$ **where** $x \in S \ y \in T$ $\text{dist } x \ y < b$
using *assms*
by (*auto simp: not_le [symmetric] le_setdist_iff*)

lemma *setdist_refl*: $\text{setdist } S \ S = 0$
by (*metis dist_eq_0_iff ex_in_conv order_antisym setdist_def setdist_le_dist setdist_pos_le*)

lemma *setdist_sym*: $\text{setdist } S \ T = \text{setdist } T \ S$
by (*force simp: setdist_def dist_commute intro!: arg_cong [where f=Inf]*)

lemma *setdist_triangle*: $\text{setdist } S \ T \leq \text{setdist } S \ \{a\} + \text{setdist } \{a\} \ T$
proof (*cases* $S = \{\} \vee T = \{\}$)
case *True* **then show** *?thesis*
using *setdist_pos_le* **by** *fastforce*
next
case *False*
then have $\bigwedge x. x \in S \implies \text{setdist } S \ T - \text{dist } x \ a \leq \text{setdist } \{a\} \ T$
using *dist_self dist_triangle3 empty_not_insert le_setdist_iff setdist_le_dist singleton_iff*
by (*smt (verit, best) dist_self dist_triangle3 empty_not_insert le_setdist_iff setdist_le_dist singleton_iff*)
then have $\text{setdist } S \ T - \text{setdist } \{a\} \ T \leq \text{setdist } S \ \{a\}$
using *False* **by** (*fastforce intro: le_setdistI*)
then show *?thesis*
by (*simp add: algebra_simps*)
qed

lemma *setdist_singletons* [*simp*]: $\text{setdist } \{x\} \ \{y\} = \text{dist } x \ y$
by (*simp add: setdist_def*)

lemma *setdist_Lipschitz*: $|\text{setdist } \{x\} \ S - \text{setdist } \{y\} \ S| \leq \text{dist } x \ y$
apply (*subst setdist_singletons [symmetric]*)
by (*metis abs_diff_le_iff diff_le_eq setdist_triangle setdist_sym*)

lemma *continuous_at_setdist* [*continuous_intros*]: $\text{continuous } (at \ x) \ (\lambda y. (\text{setdist } \{y\} \ S))$
by (*force simp: continuous_at_eps_delta dist_real_def intro: le_less_trans [OF setdist_Lipschitz]*)

lemma *continuous_on_setdist* [*continuous_intros*]: $\text{continuous_on } T \ (\lambda y. (\text{setdist } \{y\} \ S))$
by (*metis continuous_at_setdist continuous_at_imp_continuous_on*)

lemma *uniformly_continuous_on_setdist*: $\text{uniformly_continuous_on } T \ (\lambda y. (\text{setdist } \{y\} \ S))$
by (*force simp: uniformly_continuous_on_def dist_real_def intro: le_less_trans [OF setdist_Lipschitz]*)

lemma *setdist_subset_right*: $\llbracket T \neq \{\}; T \subseteq u \rrbracket \implies \text{setdist } S \ u \leq \text{setdist } S \ T$
by (*smt (verit, best) in_mono le_setdist_iff*)

lemma *setdist_subset_left*: $\llbracket S \neq \{\}; S \subseteq T \rrbracket \implies \text{setdist } T \ u \leq \text{setdist } S \ u$
by (*metis setdist_subset_right setdist_sym*)

lemma *setdist_closure_1* [*simp*]: $\text{setdist } (\text{closure } S) \ T = \text{setdist } S \ T$

proof (*cases* $S = \{\} \vee T = \{\}$)

case *True* **then show** *?thesis* **by force**

next

case *False*

{ **fix** *y*

assume $y \in T$

have *continuous_on* (*closure* *S*) ($\lambda a. \text{dist } a \ y$)

by (*auto simp: continuous_intros dist_norm*)

then have $*$: $\bigwedge x. x \in \text{closure } S \implies \text{setdist } S \ T \leq \text{dist } x \ y$

by (*fast intro: setdist_le_dist <y ∈ T> continuous_ge_on_closure*)

} **then**

show *?thesis*

by (*metis False antisym closure_eq_empty closure_subset le_setdist_iff setdist_subset_left*)

qed

lemma *setdist_closure_2* [*simp*]: $\text{setdist } T \ (\text{closure } S) = \text{setdist } T \ S$

by (*metis setdist_closure_1 setdist_sym*)

lemma *setdist_eq_0I*: $\llbracket x \in S; x \in T \rrbracket \implies \text{setdist } S \ T = 0$

by (*metis antisym dist_self setdist_le_dist setdist_pos_le*)

lemma *setdist_unique*:

$\llbracket a \in S; b \in T; \bigwedge x y. x \in S \wedge y \in T \implies \text{dist } a \ b \leq \text{dist } x \ y \rrbracket$

$\implies \text{setdist } S \ T = \text{dist } a \ b$

by (*force simp add: setdist_le_dist le_setdist_iff intro: antisym*)

lemma *setdist_le_sing*: $x \in S \implies \text{setdist } S \ T \leq \text{setdist } \{x\} \ T$

using *setdist_subset_left* **by auto**

lemma *infdist_eq_setdist*: $\text{infdist } x \ A = \text{setdist } \{x\} \ A$

by (*simp add: infdist_def setdist_def Setcompr_eq_image*)

lemma *setdist_eq_infdist*: $\text{setdist } A \ B = (\text{if } A = \{\} \text{ then } 0 \text{ else } \text{INF } a \in A. \text{ infdist } a \ B)$

proof –

have $\text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} = (\text{INF } x \in A. \text{Inf } (\text{dist } x \ ' B))$

if $b \in B$ **for** $a \ b$

proof (*rule order_antisym*)

have $\text{Inf } \{\text{dist } x \ y \mid x y. x \in A \wedge y \in B\} \leq \text{Inf } (\text{dist } x \ ' B)$

if $b \in B$ **for** $a \ x \in A$ **for** x

```

proof -
  have  $\bigwedge b'. b' \in B \implies \text{Inf } \{ \text{dist } x y \mid x y. x \in A \wedge y \in B \} \leq \text{dist } x b'$ 
    by (metis (mono_tags, lifting) ex_in_conv setdist_def setdist_le_dist
that(3))
  then show ?thesis
    by (smt (verit) cINF_greatest ex_in_conv  $\langle b \in B \rangle$ )
  qed
  then show  $\text{Inf } \{ \text{dist } x y \mid x y. x \in A \wedge y \in B \} \leq (\text{INF } x \in A. \text{Inf } (\text{dist } x ' B))$ 
    by (metis (mono_tags, lifting) cINF_greatest emptyE that)
  next
  have *:  $\bigwedge x y. [\![ b \in B; a \in A; x \in A; y \in B ]\!] \implies \exists a \in A. \text{Inf } (\text{dist } a ' B) \leq$ 
 $\text{dist } x y$ 
    by (meson bdd_below_image_dist cINF_lower)
  show  $(\text{INF } x \in A. \text{Inf } (\text{dist } x ' B)) \leq \text{Inf } \{ \text{dist } x y \mid x y. x \in A \wedge y \in B \}$ 
  proof (rule conditionally_complete_lattice_class.cInf_mono)
    show bdd_below  $((\lambda x. \text{Inf } (\text{dist } x ' B)) ' A)$ 
    by (metis (no_types, lifting) bdd_belowI2 ex_in_conv infdist_def infdist_nonneg
that(1))
  qed (use that in  $\langle \text{auto simp: *} \rangle$ )
  qed
  then show ?thesis
    by (auto simp: setdist_def infdist_def)
qed

```

```

lemma infdist_mono:
  assumes  $A \subseteq B$   $A \neq \{\}$ 
  shows  $\text{infdist } x B \leq \text{infdist } x A$ 
  by (simp add: assms infdist_eq_setdist setdist_subset_right)

```

```

lemma infdist_singleton [simp]:  $\text{infdist } x \{y\} = \text{dist } x y$ 
  by (simp add: infdist_eq_setdist)

```

```

proposition setdist_attains_inf:
  assumes compact B  $B \neq \{\}$ 
  obtains y where  $y \in B$   $\text{setdist } A B = \text{infdist } y A$ 
proof (cases  $A = \{\}$ )
  case True
    then show thesis
      by (metis assms diameter_compact_attained infdist_def setdist_def that)
  next
  case False
    obtain y where  $y \in B$  and min:  $\bigwedge y'. y' \in B \implies \text{infdist } y A \leq \text{infdist } y' A$ 
      by (metis continuous_attains_inf [OF assms continuous_on_infdist] continuous_on_id)
    show thesis
  proof
    have  $\text{setdist } A B = (\text{INF } y \in B. \text{infdist } y A)$ 
      by (metis  $\langle B \neq \{\} \rangle$  setdist_eq_infdist setdist_sym)
    also have  $\dots = \text{infdist } y A$ 

```

```

proof (rule order_antisym)
  show (INF y∈B. infdist y A) ≤ infdist y A
    by (meson ⟨y ∈ B⟩ bdd_belowI2 cInf_lower image_eqI infdist_nonneg)
next
  show infdist y A ≤ (INF y∈B. infdist y A)
    by (simp add: ⟨B ≠ {}⟩ cINF_greatest min)
qed
finally show setdist A B = infdist y A .
qed (fact ⟨y ∈ B⟩)
qed

```

4.31 Diameter Lemma

taken from the AFP entry Martingales, by Ata Keskin, TU München

```

lemma diameter_comp_strict_mono:
  fixes s :: nat ⇒ 'a :: metric_space
  assumes strict_mono r bounded {s i | i. r n ≤ i}
  shows diameter {s (r i) | i. n ≤ i} ≤ diameter {s i | i. r n ≤ i}
proof (rule diameter_subset)
  show {s (r i) | i. n ≤ i} ⊆ {s i | i. r n ≤ i} using assms(1) monotoneD
  strict_mono_mono by fastforce
qed (intro assms(2))

```

```

lemma diameter_bounded_bound':
  fixes S :: 'a :: metric_space set
  assumes S: bdd_above (case_prod dist ' (S×S)) and x ∈ S y ∈ S
  shows dist x y ≤ diameter S
proof –
  have (x,y) ∈ S×S using assms by auto
  then have dist x y ≤ (SUP (x,y)∈S×S. dist x y)
    by (metis S cSUP_upper case_prod_conv)
  with ⟨x ∈ S⟩ show ?thesis by (auto simp: diameter_def)
qed

```

```

lemma bounded_imp_dist_bounded:
  assumes bounded (range s)
  shows bounded ((λ(i, j). dist (s i) (s j)) ' ({n..} × {n..}))
  using bounded_dist_comp[OF boundedfst bounded_snd, OF bounded_Times(1,1)[OF
  assms(1,1)]] by (rule bounded_subset, force)

```

A sequence is Cauchy, if and only if it is bounded and its diameter tends to zero. The diameter is well-defined only if the sequence is bounded.

```

lemma cauchy_iff_diameter_tends_to_zero_and_bounded:
  fixes s :: nat ⇒ 'a :: metric_space
  shows Cauchy s ↔ ((λn. diameter {s i | i. i ≥ n}) → 0 ∧ bounded (range
  s))
proof –
  have {s i | i. N ≤ i} ≠ {} for N by blast

```

```

hence diameter_SUP: diameter {s i | i. N ≤ i} = (SUP (i, j) ∈ {N..} × {N..}.
dist (s i) (s j)) for N unfolding diameter_def by (auto intro!: arg_cong[of _ _
Sup])
show ?thesis
proof (intro iffI)
  assume asm: Cauchy s
  have ∃N. ∀n≥N. norm (diameter {s i | i. n ≤ i}) < e if e_pos: e > 0 for e
  proof -
    obtain N where dist_less: dist (s n) (s m) < (1/2) * e if n ≥ N m ≥ N for
n m using asm e_pos by (metis Cauchy_def mult_pos_pos zero_less_divide_iff
zero_less_numeral zero_less_one)
    {
      fix r assume r ≥ N
      hence dist (s n) (s m) < (1/2) * e if n ≥ r m ≥ r for n m using dist_less
      that by simp
      hence (SUP (i, j) ∈ {r..} × {r..}. dist (s i) (s j)) ≤ (1/2) * e by (intro
cSup_least) fastforce+
      also have ... < e using e_pos by simp
      finally have diameter {s i | i. r ≤ i} < e using diameter_SUP by presburger
    }
    moreover have diameter {s i | i. r ≤ i} ≥ 0 for r unfolding diam-
eter_SUP using bounded_imp_dist_bounded[OF cauchy_imp_bounded, THEN
bounded_imp_bdd_above, OF asm] by (intro cSup_upper2, auto)
    ultimately show ?thesis by auto
  qed
  thus (λn. diameter {s i | i. n ≤ i}) → 0 ∧ bounded (range s) using
cauchy_imp_bounded[OF asm] by (simp add: LIMSEQ_iff)
  next
  assume asm: (λn. diameter {s i | i. n ≤ i}) → 0 ∧ bounded (range s)
  have ∃N. ∀n≥N. ∀m≥N. dist (s n) (s m) < e if e_pos: e > 0 for e
  proof -
    obtain N where diam_less: diameter {s i | i. r ≤ i} < e if r ≥ N for r
using LIMSEQ_D asm(1) e_pos by fastforce
    {
      fix n m assume n ≥ N m ≥ N
      hence dist (s n) (s m) < e using cSUP_lessD[OF bounded_imp_dist_bounded[THEN
bounded_imp_bdd_above], OF diam_less[unfolded diameter_SUP]] asm by auto
    }
    thus ?thesis by blast
  qed
  then show Cauchy s by (simp add: Cauchy_def)
qed
qed
end

```

4.32 Elementary Normed Vector Spaces

theory *Elementary_Normed_Spaces*

```

imports
  HOL-Library.FuncSet
  Elementary_Metric_Spaces Cartesian_Space
  Connected
begin

```

4.32.1 Orthogonal Transformation of Balls

4.32.2 Various Lemmas Combining Imports

```

lemma open_sums:
  fixes T :: ('b::real_normed_vector) set
  assumes open S  $\vee$  open T
  shows open ( $\bigcup_{x \in S} \bigcup_{y \in T} \{x + y\}$ )
  using assms
proof
  assume S: open S
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x  $\in$  S y  $\in$  T
    with S obtain e where e > 0 and e:  $\bigwedge x'. \text{dist } x' x < e \implies x' \in S$ 
    by (auto simp: open_dist)
    then have  $\bigwedge z. \text{dist } z (x + y) < e \implies \exists x \in S. \exists y \in T. z = x + y$ 
    by (metis  $\langle y \in T \rangle$  diff_add_cancel dist_add_cancel2)
    then show  $\exists e > 0. \forall z. \text{dist } z (x + y) < e \longrightarrow (\exists x \in S. \exists y \in T. z = x + y)$ 
    using  $\langle 0 < e \rangle \langle x \in S \rangle$  by blast
  qed
next
  assume T: open T
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x  $\in$  S y  $\in$  T
    with T obtain e where e > 0 and e:  $\bigwedge x'. \text{dist } x' y < e \implies x' \in T$ 
    by (auto simp: open_dist)
    then have  $\bigwedge z. \text{dist } z (x + y) < e \implies \exists x \in S. \exists y \in T. z = x + y$ 
    by (metis  $\langle x \in S \rangle$  add_diff_cancel_left' add_diff_eq diff_diff_add dist_norm)
    then show  $\exists e > 0. \forall z. \text{dist } z (x + y) < e \longrightarrow (\exists x \in S. \exists y \in T. z = x + y)$ 
    using  $\langle 0 < e \rangle \langle y \in T \rangle$  by blast
  qed
qed

lemma image_orthogonal_transformation_ball:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a
  assumes orthogonal_transformation f
  shows f ' ball x r = ball (f x) r
proof (intro equalityI subsetI)
  fix y assume y  $\in$  f ' ball x r
  with assms show y  $\in$  ball (f x) r

```



```

  by (auto simp: orthogonal_transformation_isometry)
next
  fix y assume y: y ∈ ball (f x) r
  then obtain z where z: y = f z
  using assms orthogonal_transformation_surj by blast
  with y assms show y ∈ f ` ball x r
  by (auto simp: orthogonal_transformation_isometry)
qed

```

```

lemma image_orthogonal_transformation_cball:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes orthogonal_transformation f
  shows f ` cball x r = cball (f x) r
proof (intro equalityI subsetI)
  fix y assume y ∈ f ` cball x r
  with assms show y ∈ cball (f x) r
  by (auto simp: orthogonal_transformation_isometry)
next
  fix y assume y: y ∈ cball (f x) r
  then obtain z where z: y = f z
  using assms orthogonal_transformation_surj by blast
  with y assms show y ∈ f ` cball x r
  by (auto simp: orthogonal_transformation_isometry)
qed

```

4.32.3 Support

definition (in monoid_add) support_on :: 'b set ⇒ ('b ⇒ 'a) ⇒ 'b set
 where support_on S f = {x ∈ S. f x ≠ 0}

lemma in_support_on: $x \in \text{support_on } S f \iff x \in S \wedge f x \neq 0$
 by (simp add: support_on_def)

lemma support_on_simps[simp]:
 support_on {} f = {}
 support_on (insert x S) f =
 (if f x = 0 then support_on S f else insert x (support_on S f))
 support_on (S ∪ T) f = support_on S f ∪ support_on T f
 support_on (S ∩ T) f = support_on S f ∩ support_on T f
 support_on (S - T) f = support_on S f - support_on T f
 support_on (f ` S) g = f ` (support_on S (g ∘ f))
 unfolding support_on_def by auto

lemma support_on_cong:
 $(\bigwedge x. x \in S \implies f x = 0 \iff g x = 0) \implies \text{support_on } S f = \text{support_on } S g$
 by (auto simp: support_on_def)

lemma support_on_if: $a \neq 0 \implies \text{support_on } A (\lambda x. \text{if } P x \text{ then } a \text{ else } 0) = \{x \in A. P x\}$

by (auto simp: support_on_def)

lemma support_on_if_subset: support_on A ($\lambda x. \text{if } P x \text{ then } a \text{ else } 0$) \subseteq { $x \in A. P x$ }

by (auto simp: support_on_def)

lemma finite_support[intro]: finite S \implies finite (support_on S f)
 unfolding support_on_def by auto

definition (in comm_monoid_add) supp_sum :: ('b \Rightarrow 'a) \Rightarrow 'b set \Rightarrow 'a
 where supp_sum f S = ($\sum_{x \in \text{support_on } S f. f x$)

lemma supp_sum_empty[simp]: supp_sum f {} = 0
 unfolding supp_sum_def by auto

lemma supp_sum_insert[simp]:
 finite (support_on S f) \implies
 supp_sum f (insert x S) = (if $x \in S$ then supp_sum f S else $f x + \text{supp_sum } f S$)
 by (simp add: supp_sum_def in_support_on insert_absorb)

lemma supp_sum_divide_distrib: supp_sum f A / (r::'a::field) = supp_sum ($\lambda n. f n / r$) A
 by (cases r = 0)
 (auto simp: supp_sum_def sum_divide_distrib intro!: sum.cong support_on_cong)

4.32.4 Intervals

lemma image_affinity_interval:
 fixes c :: 'a::ordered_real_vector
 shows (($\lambda x. m *_R x + c$) ' {a..b}) =
 (if {a..b}={} then {}
 else if $0 \leq m$ then { $m *_R a + c .. m *_R b + c$ }
 else { $m *_R b + c .. m *_R a + c$ })
 (is ?lhs = ?rhs)

proof (cases m=0)

case True

then show ?thesis

by force

next

case False

show ?thesis

proof

show ?lhs \subseteq ?rhs

by (auto simp: scaleR_left_mono scaleR_left_mono_neg)

show ?rhs \subseteq ?lhs

proof (clarsimp, intro conjI impI subsetI)

show [$0 \leq m; a \leq b; x \in \{m *_R a + c .. m *_R b + c\}$]

```

     $\implies x \in (\lambda x. m *_{\mathbb{R}} x + c) \text{ ' } \{a..b\} \text{ for } x$ 
  using False
  by (rule_tac x=inverse m *R (x-c) in image_eqI)
    (auto simp: pos_le_divideR_eq pos_divideR_le_eq le_diff_eq diff_le_eq)
  show  $\llbracket \neg 0 \leq m; a \leq b; x \in \{m *_{\mathbb{R}} b + c..m *_{\mathbb{R}} a + c\} \rrbracket$ 
     $\implies x \in (\lambda x. m *_{\mathbb{R}} x + c) \text{ ' } \{a..b\} \text{ for } x$ 
  by (rule_tac x=inverse m *R (x-c) in image_eqI)
    (auto simp add: neg_le_divideR_eq neg_divideR_le_eq le_diff_eq
diff_le_eq)
  qed
  qed
  qed

```

4.32.5 Limit Points

lemma *islimpt_ball*:

```

  fixes  $x y :: 'a :: \{real\_normed\_vector, perfect\_space\}$ 
  shows  $y \text{ islimpt ball } x e \iff 0 < e \wedge y \in \text{cball } x e$ 
  (is ?lhs  $\iff$  ?rhs)

```

proof

show ?rhs if ?lhs

proof

```

  {
    assume  $e \leq 0$ 
    then have *:  $\text{ball } x e = \{\}$ 
      using ball_eq_empty[of x e] by auto
    have False using  $\langle ?lhs \rangle$ 
      unfolding * using islimpt_EMPTY[of y] by auto
  }

```

then show $e > 0$ by (metis not_less)

show $y \in \text{cball } x e$

```

  using closed_cball[of x e] islimpt_subset[of y ball x e cball x e]
    ball_subset_cball[of x e]  $\langle ?lhs \rangle$ 
  unfolding closed_limpt by auto

```

qed

show ?lhs if ?rhs

proof -

from that have $e > 0$ by auto

```

  {
    fix  $d :: real$ 
    assume  $d > 0$ 
    have  $\exists x' \in \text{ball } x e. x' \neq y \wedge \text{dist } x' y < d$ 
    proof (cases  $d \leq \text{dist } x y$ )
    case True
    then show ?thesis
    proof (cases  $x = y$ )
    case True
    then have False
      using  $\langle d \leq \text{dist } x y \rangle \langle d > 0 \rangle$  by auto

```

```

    then show ?thesis
      by auto
  next
    case False
    have  $\text{dist } x (y - (d / (2 * \text{dist } y x)) *_{\mathbb{R}} (y - x)) =$ 
       $\text{norm } (x - y + (d / (2 * \text{norm } (y - x))) *_{\mathbb{R}} (y - x))$ 
    unfolding mem_cball mem_ball dist_norm diff_diff_eq2 diff_add_eq[symmetric]
      by auto
    also have  $\dots = |- 1 + d / (2 * \text{norm } (x - y))| * \text{norm } (x - y)$ 
      using scaleR_left_distrib[of  $- 1 d / (2 * \text{norm } (y - x))$ ], symmetric, of
 $y - x]$ 
      unfolding scaleR_minus_left scaleR_one
      by (auto simp: norm_minus_commute)
    also have  $\dots = |- \text{norm } (x - y) + d / 2|$ 
      unfolding abs_mult_pos[of  $\text{norm } (x - y)$ ], OF norm_ge_zero[of  $x - y$ ]
      unfolding distrib_right using  $\langle x \neq y \rangle$  by auto
    also have  $\dots \leq e - d/2$  using  $\langle d \leq \text{dist } x y \rangle$  and  $\langle d > 0 \rangle$  and  $\langle ?rhs \rangle$ 
      by (auto simp: dist_norm)
    finally have  $y - (d / (2 * \text{dist } y x)) *_{\mathbb{R}} (y - x) \in \text{ball } x e$  using  $\langle d > 0 \rangle$ 
      by auto
    moreover
    have  $(d / (2 * \text{dist } y x)) *_{\mathbb{R}} (y - x) \neq 0$ 
      using  $\langle x \neq y \rangle$ [unfolded dist_nz]  $\langle d > 0 \rangle$  unfolding scaleR_eq_0_iff
      by (auto simp: dist_commute)
    moreover
    have  $\text{dist } (y - (d / (2 * \text{dist } y x)) *_{\mathbb{R}} (y - x)) y < d$ 
      using  $\langle 0 < d \rangle$  by (fastforce simp: dist_norm)
    ultimately show ?thesis
      by (rule_tac  $x = y - (d / (2 * \text{dist } y x)) *_{\mathbb{R}} (y - x)$  in ballI) auto
  qed
next
  case False
  then have  $d > \text{dist } x y$  by auto
  show  $\exists x' \in \text{ball } x e. x' \neq y \wedge \text{dist } x' y < d$ 
  proof (cases  $x = y$ )
    case True
    obtain  $z$  where  $z: z \neq y \text{ dist } z y < \min e d$ 
      using perfect_choose_dist[of  $\min e d y$ ]
      using  $\langle d > 0 \rangle \langle e > 0 \rangle$  by auto
    show ?thesis
      by (metis True  $z \text{ dist\_commute mem\_ball min\_less\_iff\_conj}$ )
  next
    case False
    then show ?thesis
      using  $\langle d > 0 \rangle \langle d > \text{dist } x y \rangle \langle ?rhs \rangle$  by force
  qed
qed
}
then show ?thesis

```

```

    unfolding mem_cball islimpt_approachable mem_ball by auto
  qed
qed

lemma closure_ball_lemma:
  fixes x y :: 'a::real_normed_vector
  assumes x ≠ y
  shows y islimpt ball x (dist x y)
proof (rule islimptI)
  fix T
  assume y ∈ T open T
  then obtain r where 0 < r ∀ z. dist z y < r ⟶ z ∈ T
    unfolding open_dist by fast
  — choose point between x and y, within distance r of y.
  define k where k = min 1 (r / (2 * dist x y))
  define z where z = y + scaleR k (x - y)
  have z_def2: z = x + scaleR (1 - k) (y - x)
    unfolding z_def by (simp add: algebra_simps)
  have dist z y < r
    unfolding z_def k_def using ‹0 < r›
    by (simp add: dist_norm min_def)
  then have z ∈ T
    using ‹∀ z. dist z y < r ⟶ z ∈ T› by simp
  have dist x z < dist x y
    using ‹0 < r› assms by (simp add: z_def2 k_def dist_norm norm_minus_commute)

  then have z ∈ ball x (dist x y)
    by simp
  have z ≠ y
    unfolding z_def k_def using ‹x ≠ y› ‹0 < r›
    by (simp add: min_def)
  show ∃ z ∈ ball x (dist x y). z ∈ T ∧ z ≠ y
    using ‹z ∈ ball x (dist x y)› ‹z ∈ T› ‹z ≠ y›
    by fast
qed

```

4.32.6 Balls and Spheres in Normed Spaces

```

lemma mem_ball_0 [simp]: x ∈ ball 0 e ⟷ norm x < e
  for x :: 'a::real_normed_vector
  by simp

```

```

lemma mem_cball_0 [simp]: x ∈ cball 0 e ⟷ norm x ≤ e
  for x :: 'a::real_normed_vector
  by simp

```

```

lemma closure_ball [simp]:
  fixes x :: 'a::real_normed_vector
  assumes 0 < e

```

```

  shows closure (ball x e) = cball x e
proof
  show closure (ball x e)  $\subseteq$  cball x e
    using closed_cball closure_minimal by blast
  have  $\bigwedge y. \text{dist } x \ y < e \vee \text{dist } x \ y = e \implies y \in \text{closure } (\text{ball } x \ e)$ 
    by (metis Un_iff assms closure_ball_lemma closure_def dist_eq_0_iff mem_Collect_eq
mem_ball)
  then show cball x e  $\subseteq$  closure (ball x e)
    by force
qed

```

```

lemma mem_sphere_0 [simp]:  $x \in \text{sphere } 0 \ e \longleftrightarrow \text{norm } x = e$ 
  for  $x :: 'a::\text{real\_normed\_vector}$ 
  by simp

```

```

lemma interior_cball [simp]:
  fixes  $x :: 'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}$ 
  shows interior (cball x e) = ball x e
proof (cases  $e \geq 0$ )
  case False note cs = this
  from cs have null:  $\text{ball } x \ e = \{\}$ 
    using ball_empty[of e x] by auto
  moreover
  have  $\text{cball } x \ e = \{\}$ 
  proof (rule equalsOI)
    fix  $y$ 
    assume  $y \in \text{cball } x \ e$ 
    then show False
      by (metis ball_eq_empty null cs dist_eq_0_iff dist_le_zero_iff empty_subsetI
mem_cball
subset_antisym subset_ball)
  qed
  then have interior (cball x e) =  $\{\}$ 
    using interior_empty by auto
  ultimately show ?thesis by blast
next
  case True note cs = this
  have  $\text{ball } x \ e \subseteq \text{cball } x \ e$ 
    using ball_subset_cball by auto
  moreover
  {
    fix  $S \ y$ 
    assume  $as: S \subseteq \text{cball } x \ e \ \text{open } S \ y \in S$ 
    then obtain  $d > 0$  and  $d: \forall x'. \text{dist } x' \ y < d \longrightarrow x' \in S$ 
      unfolding open_dist by blast
    then obtain  $xa$  where  $xa\_y: xa \neq y$  and  $xa: \text{dist } xa \ y < d$ 
      using perfect_choose_dist [of d] by auto
    have  $xa \in S$ 

```

```

    using d[THEN spec[where x = xa]]
    using xa by (auto simp: dist_commute)
  then have xa_cball: xa ∈ cball x e
    using as(1) by auto
  then have y ∈ ball x e
  proof (cases x = y)
    case True
    then have e > 0 using cs order.order_iff_strict xa_cball xa_y by fastforce
    then show y ∈ ball x e
      using ⟨x = y⟩ by simp
    next
    case False
    have dist (y + (d / 2 / dist y x) *R (y - x)) y < d
      unfolding dist_norm
      using ⟨d>0⟩ norm_ge_zero[of y - x] ⟨x ≠ y⟩ by auto
    then have *: y + (d / 2 / dist y x) *R (y - x) ∈ cball x e
      using d as(1)[unfolded subset_eq] by blast
    have y - x ≠ 0 using ⟨x ≠ y⟩ by auto
    hence **: d / (2 * norm (y - x)) > 0
      unfolding zero_less_norm_iff[symmetric] using ⟨d>0⟩ by auto
    have dist (y + (d / 2 / dist y x) *R (y - x)) x =
      norm (y + (d / (2 * norm (y - x))) *R y - (d / (2 * norm (y - x))) *R
x - x)
      by (auto simp: dist_norm algebra_simps)
    also have ... = norm ((1 + d / (2 * norm (y - x))) *R (y - x))
      by (auto simp: algebra_simps)
    also have ... = |1 + d / (2 * norm (y - x))| * norm (y - x)
      using ** by auto
    also have ... = (dist y x) + d/2
      using ** by (auto simp: distrib_right dist_norm)
    finally have e ≥ dist x y + d/2
      using *[unfolded mem_cball] by (auto simp: dist_commute)
    then show y ∈ ball x e
      unfolding mem_ball using ⟨d>0⟩ by auto
  qed
}
then have ∀ S ⊆ cball x e. open S ⟶ S ⊆ ball x e
  by auto
ultimately show ?thesis
  using interior_unique[of ball x e cball x e]
  using open_ball[of x e]
  by auto
qed

lemma frontier_ball [simp]:
  fixes a :: 'a::real_normed_vector
  shows 0 < e ⟹ frontier (ball a e) = sphere a e
  by (force simp: frontier_def)

```

```

lemma frontier_cball [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space}
  shows frontier (cball a e) = sphere a e
  by (force simp: frontier_def)

corollary compact_sphere [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space, heine_borel}
  shows compact (sphere a r)
using compact_frontier [of cball a r] by simp

corollary bounded_sphere [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space, heine_borel}
  shows bounded (sphere a r)
by (simp add: compact_imp_bounded)

corollary closed_sphere [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space, heine_borel}
  shows closed (sphere a r)
by (simp add: compact_imp_closed)

lemma image_add_ball [simp]:
  fixes a :: 'a::real_normed_vector
  shows (+) b ' ball a r = ball (a+b) r
proof -
  { fix x :: 'a
    assume dist (a + b) x < r
    moreover
    have b + (x - b) = x
      by simp
    ultimately have x ∈ (+) b ' ball a r
      by (metis add_commute dist_add_cancel image_eqI mem_ball) }
  then show ?thesis
    by (auto simp: add_commute)
qed

lemma image_add_cball [simp]:
  fixes a :: 'a::real_normed_vector
  shows (+) b ' cball a r = cball (a+b) r
proof -
  have  $\bigwedge x. \text{dist } (a + b) x \leq r \implies \exists y \in \text{cball } a r. x = b + y$ 
  by (metis (no_types) add_commute diff_add_cancel dist_add_cancel2 mem_cball)
  then show ?thesis
    by (force simp: add_commute)
qed

```

4.32.7 Various Lemmas on Normed Algebras

```

lemma closed_of_nat_image: closed (of_nat ' A :: 'a::real_normed_algebra_1
set)

```


by (rule discrete_imp_closed[of 1]) (auto simp: dist_of_nat)

lemma *closed_of_int_image*: closed (of_int 'A :: 'a::real_normed_algebra_1 set)

by (rule discrete_imp_closed[of 1]) (auto simp: dist_of_int)

lemma *closed_Nats* [simp]: closed (\mathbb{N} :: 'a :: real_normed_algebra_1 set)

unfolding *Nats_def* by (rule closed_of_nat_image)

lemma *closed_Ints* [simp]: closed (\mathbb{Z} :: 'a :: real_normed_algebra_1 set)

unfolding *Ints_def* by (rule closed_of_int_image)

lemma *closed_subset_Ints*:

fixes *A* :: 'a :: real_normed_algebra_1 set

assumes $A \subseteq \mathbb{Z}$

shows closed *A*

proof (intro discrete_imp_closed[OF zero_less_one] ballI impI, goal_cases)

case (1 *x y*)

with *assms* have $x \in \mathbb{Z}$ and $y \in \mathbb{Z}$ by auto

with $\langle \text{dist } y \ x < 1 \rangle$ show $y = x$

by (auto elim!: *Ints_cases* simp: dist_of_int)

qed

4.32.8 Filters

definition *indirection* :: 'a::real_normed_vector \Rightarrow 'a \Rightarrow 'a filter (infixr *indirection* 70)

where *a indirection* $v = \text{at } a \text{ within } \{b. \exists c \geq 0. b - a = \text{scaleR } c \ v\}$

4.32.9 Trivial Limits

lemma *trivial_limit_at_infinity*:

\neg trivial_limit (at_infinity :: ('a::{real_normed_vector,perfect_space}) filter)

proof –

obtain *x*::'a where $x \neq 0$

by (meson perfect_choose_dist zero_less_one)

then have $b \leq \text{norm } ((b / \text{norm } x) *_{\mathbb{R}} x)$ for *b*

by simp

then show *?thesis*

unfolding *trivial_limit_def* eventually_at_infinity

by blast

qed

lemma *at_within_ball_bot_iff*:

fixes *x y* :: 'a::{real_normed_vector,perfect_space}

shows *at* *x* within ball *y* *r* = bot $\longleftrightarrow (r=0 \vee x \notin \text{cball } y \ r)$

unfolding *trivial_limit_within*

by (metis (no_types) cball_empty_equals0D islimpt_ball_less_linear)

4.32.10 Limits

proposition *Lim_at_infinity*: $(f \longrightarrow l) \text{ at_infinity} \longleftrightarrow (\forall e > 0. \exists b. \forall x. \text{norm } x \geq b \longrightarrow \text{dist } (f x) l < e)$

by (*auto simp: tendsto_iff eventually_at_infinity*)

corollary *Lim_at_infinityI* [*intro?*]:

assumes $\bigwedge e. e > 0 \implies \exists B. \forall x. \text{norm } x \geq B \longrightarrow \text{dist } (f x) l \leq e$

shows $(f \longrightarrow l) \text{ at_infinity}$

proof –

have $\bigwedge e. e > 0 \implies \exists B. \forall x. \text{norm } x \geq B \longrightarrow \text{dist } (f x) l < e$

by (*meson assms dense le_less_trans*)

then show *?thesis*

using *Lim_at_infinity* by *blast*

qed

lemma *Lim_transform_within_set_eq*:

fixes $a :: 'a::\text{metric_space}$ and $l :: 'b::\text{metric_space}$

shows *eventually* $(\lambda x. x \in S \longleftrightarrow x \in T) \text{ (at } a)$

$\implies ((f \longrightarrow l) \text{ (at } a \text{ within } S) \longleftrightarrow (f \longrightarrow l) \text{ (at } a \text{ within } T))$

by (*force intro: Lim_transform_within_set elim: eventually_mono*)

lemma *Lim_null*:

fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$

shows $(f \longrightarrow l) \text{ net} \longleftrightarrow ((\lambda x. f(x) - l) \longrightarrow 0) \text{ net}$

by (*simp add: Lim_dist_norm*)

lemma *Lim_null_comparison*:

fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$

assumes *eventually* $(\lambda x. \text{norm } (f x) \leq g x) \text{ net } (g \longrightarrow 0) \text{ net}$

shows $(f \longrightarrow 0) \text{ net}$

using *assms(2)*

proof (*rule metric_tendsto_imp_tendsto*)

show *eventually* $(\lambda x. \text{dist } (f x) 0 \leq \text{dist } (g x) 0) \text{ net}$

using *assms(1)* by (*rule eventually_mono*) (*simp add: dist_norm*)

qed

lemma *Lim_transform_bound*:

fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$

and $g :: 'a \Rightarrow 'c::\text{real_normed_vector}$

assumes *eventually* $(\lambda n. \text{norm } (f n) \leq \text{norm } (g n)) \text{ net}$

and $(g \longrightarrow 0) \text{ net}$

shows $(f \longrightarrow 0) \text{ net}$

using *assms(1)* *tendsto_norm_zero* [*OF assms(2)*]

by (*rule Lim_null_comparison*)

lemma *lim_null_mult_right_bounded*:

fixes $f :: 'a \Rightarrow 'b::\text{real_normed_div_algebra}$

assumes $f: (f \longrightarrow 0) F$ and $g: \text{eventually } (\lambda x. \text{norm}(g x) \leq B) F$

shows $((\lambda z. f z * g z) \longrightarrow 0) F$

```

proof -
  have (( $\lambda x. \text{norm } (f x) * \text{norm } (g x) \longrightarrow 0$ )  $F$ )
  proof (rule Lim_null_comparison)
    show  $\forall_F x \text{ in } F. \text{norm } (\text{norm } (f x) * \text{norm } (g x)) \leq \text{norm } (f x) * B$ 
      by (simp add: eventually_mono [OF g] mult_left_mono)
    show (( $\lambda x. \text{norm } (f x) * B \longrightarrow 0$ )  $F$ )
      by (simp add: f tendsto_mult_left_zero tendsto_norm_zero)
  qed
then show ?thesis
  by (subst tendsto_norm_zero_iff [symmetric]) (simp add: norm_mult)
qed

```

lemma *lim_null_mult_left_bounded*:

```

fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_div\_algebra}$ 
assumes  $g: \text{eventually } (\lambda x. \text{norm}(g x) \leq B) F$  and  $f: (f \longrightarrow 0) F$ 
shows (( $\lambda z. g z * f z \longrightarrow 0$ )  $F$ )

```

```

proof -
  have (( $\lambda x. \text{norm } (g x) * \text{norm } (f x) \longrightarrow 0$ )  $F$ )
  proof (rule Lim_null_comparison)
    show  $\forall_F x \text{ in } F. \text{norm } (\text{norm } (g x) * \text{norm } (f x)) \leq B * \text{norm } (f x)$ 
      by (simp add: eventually_mono [OF g] mult_right_mono)
    show (( $\lambda x. B * \text{norm } (f x) \longrightarrow 0$ )  $F$ )
      by (simp add: f tendsto_mult_right_zero tendsto_norm_zero)
  qed
then show ?thesis
  by (subst tendsto_norm_zero_iff [symmetric]) (simp add: norm_mult)
qed

```

lemma *lim_null_scaleR_bounded*:

```

assumes  $f: (f \longrightarrow 0) \text{ net}$  and  $gB: \text{eventually } (\lambda a. f a = 0 \vee \text{norm}(g a) \leq B)$ 
net
shows (( $\lambda n. f n *_{\mathbb{R}} g n \longrightarrow 0$ )  $\text{net}$ )

```

```

proof
  fix  $\varepsilon::\text{real}$ 
  assume  $0 < \varepsilon$ 
  then have  $B: 0 < \varepsilon / (\text{abs } B + 1)$  by simp
  have  $*$ :  $|f x| * \text{norm } (g x) < \varepsilon$  if  $f: |f x| * (|B| + 1) < \varepsilon$  and  $g: \text{norm } (g x) \leq B$ 
for  $x$ 
  proof -
    have  $|f x| * \text{norm } (g x) \leq |f x| * B$ 
      by (simp add: mult_left_mono g)
    also have  $\dots \leq |f x| * (|B| + 1)$ 
      by (simp add: mult_left_mono)
    also have  $\dots < \varepsilon$ 
      by (rule f)
    finally show ?thesis .
  qed
have  $\bigwedge x. [|f x| < \varepsilon / (|B| + 1); \text{norm } (g x) \leq B] \implies |f x| * \text{norm } (g x) < \varepsilon$ 
  by (simp add: * pos_less_divide_eq)

```

then show $\forall_F x \text{ in } \text{net}. \text{dist } (f x *_{\mathbb{R}} g x) 0 < \varepsilon$
using $\langle 0 < \varepsilon \rangle$ **by** (auto intro: eventually_mono [OF eventually_conj [OF
tendstoD [OF f B] gB]])
qed

lemma *Lim_norm_ubound*:
fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$
assumes $\neg(\text{trivial_limit } \text{net})$ $(f \longrightarrow l)$ *net eventually* $(\lambda x. \text{norm}(f x) \leq e)$ *net*
shows $\text{norm}(l) \leq e$
using *assms* **by** (fast intro: tendsto_le tendsto_intros)

lemma *Lim_norm_lbound*:
fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$
assumes $\neg \text{trivial_limit } \text{net}$
and $(f \longrightarrow l)$ *net*
and *eventually* $(\lambda x. e \leq \text{norm } (f x))$ *net*
shows $e \leq \text{norm } l$
using *assms* **by** (fast intro: tendsto_le tendsto_intros)

Limit under bilinear function

lemma *Lim_bilinear*:
assumes $(f \longrightarrow l)$ *net*
and $(g \longrightarrow m)$ *net*
and *bounded_bilinear* h
shows $((\lambda x. h (f x) (g x)) \longrightarrow (h l m))$ *net*
using $\langle \text{bounded_bilinear } h \rangle$ $\langle f \longrightarrow l \rangle$ *net* $\langle g \longrightarrow m \rangle$ *net*
by (rule bounded_bilinear.tendsto)

lemma *Lim_at_zero*:
fixes $a :: 'a::\text{real_normed_vector}$
and $l :: 'b::\text{topological_space}$
shows $(f \longrightarrow l) \text{ (at } a) \longleftrightarrow ((\lambda x. f(a + x)) \longrightarrow l) \text{ (at } 0)$
using *LIM_offset_zero* *LIM_offset_zero_cancel* ..

4.32.11 Limit Point of Filter

lemma *netlimit_at_vector*:
fixes $a :: 'a::\text{real_normed_vector}$
shows $\text{netlimit } (\text{at } a) = a$
proof (cases $\exists x. x \neq a$)
case *True* **then obtain** x **where** $x \neq a$..
have $\bigwedge d. 0 < d \implies \exists x. x \neq a \wedge \text{norm } (x - a) < d$
by (rule_tac $x = a + \text{scaleR } (d / 2) (\text{sgn } (x - a))$ **in** *exI*) (simp add: norm_sgn
sgn_zero_iff x)
then have $\neg \text{trivial_limit } (\text{at } a)$
by (auto simp: trivial_limit_def eventually_at dist_norm)
then show *?thesis*
by (rule *Lim_ident_at* [of a *UNIV*])
qed *simp*

4.32.12 Boundedness

lemma *continuous_on_closure_norm_le*:

fixes $f :: 'a::metric_space \Rightarrow 'b::real_normed_vector$

assumes *continuous_on* (closure s) f

and $\forall y \in s. \text{norm}(f\ y) \leq b$

and $x \in (\text{closure } s)$

shows $\text{norm}(f\ x) \leq b$

proof –

have $*$: $f\ ` s \subseteq \text{cball } 0\ b$

using *assms*(2)[*unfolded mem_cball_0[symmetric]*] **by** *auto*

show *?thesis*

by (*meson* $*$ *assms*(1) *assms*(3) *closed_cball_image_closure_subset_image_subset_iff mem_cball_0*)

qed

lemma *bounded_pos*: $\text{bounded } S \longleftrightarrow (\exists b > 0. \forall x \in S. \text{norm } x \leq b)$

unfolding *bounded_iff*

by (*meson less_imp_le not_le order_trans zero_less_one*)

lemma *bounded_pos_less*: $\text{bounded } S \longleftrightarrow (\exists b > 0. \forall x \in S. \text{norm } x < b)$

by (*metis bounded_pos le_less_trans less_imp_le linordered_field_no_ub*)

lemma *bounded_normE*:

assumes *bounded* A

obtains B **where** $B > 0 \wedge z. z \in A \implies \text{norm } z \leq B$

by (*meson assms bounded_pos*)

lemma *bounded_normE_less*:

assumes *bounded* A

obtains B **where** $B > 0 \wedge z. z \in A \implies \text{norm } z < B$

by (*meson assms bounded_pos_less*)

lemma *Bseq_eq_bounded*:

fixes $f :: \text{nat} \Rightarrow 'a::real_normed_vector$

shows $B\text{seq } f \longleftrightarrow \text{bounded } (\text{range } f)$

unfolding *Bseq_def bounded_pos* **by** *auto*

lemma *bounded_linear_image*:

assumes *bounded* S

and *bounded_linear* f

shows *bounded* ($f\ ` S$)

proof –

from *assms*(1) **obtain** b **where** $b > 0$ **and** b : $\forall x \in S. \text{norm } x \leq b$

unfolding *bounded_pos* **by** *auto*

from *assms*(2) **obtain** B **where** B : $B > 0 \forall x. \text{norm}(f\ x) \leq B * \text{norm } x$

using *bounded_linear.pos_bounded* **by** (*auto simp: ac_simps*)

show *?thesis*

unfolding *bounded_pos*

proof (*intro exI, safe*)

```

  show  $\text{norm } (f x) \leq B * b$  if  $x \in S$  for  $x$ 
  by (meson  $B b$  less_imp_le mult_left_mono order_trans that)
  qed (use <b > 0> <B > 0> in auto)
qed

```

```

lemma bounded_scaling:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  shows  $\text{bounded } S \implies \text{bounded } ((\lambda x. c *_{\mathbb{R}} x) ' S)$ 
  by (simp add: bounded_linear_image bounded_linear_scaleR_right)

```

```

lemma bounded_scaleR_comp:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes  $\text{bounded } (f ' S)$ 
  shows  $\text{bounded } ((\lambda x. r *_{\mathbb{R}} f x) ' S)$ 
  using bounded_scaling[of  $f ' S r$ ] assms
  by (auto simp: image_image)

```

```

lemma bounded_translation:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes  $\text{bounded } S$ 
  shows  $\text{bounded } ((\lambda x. a + x) ' S)$ 
proof -
  from assms obtain  $b$  where  $b: b > 0 \ \forall x \in S. \text{norm } x \leq b$ 
  unfolding bounded_pos by auto
  {
    fix  $x$ 
    assume  $x \in S$ 
    then have  $\text{norm } (a + x) \leq b + \text{norm } a$ 
      using norm_triangle_ineq[of  $a x$ ]  $b$  by auto
  }
  then show ?thesis
  unfolding bounded_pos
  using norm_ge_zero[of  $a$ ]  $b(1)$  and add_strict_increasing[of  $b 0 \text{norm } a$ ]
  by (auto intro!: exI[of _  $b + \text{norm } a$ ])
qed

```

```

lemma bounded_translation_minus:
  fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
  shows  $\text{bounded } S \implies \text{bounded } ((\lambda x. x - a) ' S)$ 
  using bounded_translation [of  $S -a$ ] by simp

```

```

lemma bounded_uminus [simp]:
  fixes  $X :: 'a::\text{real\_normed\_vector\_set}$ 
  shows  $\text{bounded } (\text{uminus } ' X) \longleftrightarrow \text{bounded } X$ 
  by (auto simp: bounded_def dist_norm; rule_tac  $x = -x$  in exI; force simp: add_commute
  norm_minus_commute)

```

```

lemma uminus_bounded_comp [simp]:
  fixes  $f :: 'a \Rightarrow 'b::\text{real\_normed\_vector}$ 

```

```

shows bounded (( $\lambda x. - f x$ ) '  $S$ )  $\longleftrightarrow$  bounded ( $f$  '  $S$ )
using bounded_uminus[of  $f$  '  $S$ ]
by (auto simp: image_image)

```

lemma bounded_plus_comp:

```

fixes  $f g :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$ 
assumes bounded ( $f$  '  $S$ )
assumes bounded ( $g$  '  $S$ )
shows bounded (( $\lambda x. f x + g x$ ) '  $S$ )

```

proof –

```

{
  fix  $B C$ 
  assume  $\bigwedge x. x \in S \implies \text{norm } (f x) \leq B \wedge x. x \in S \implies \text{norm } (g x) \leq C$ 
  then have  $\bigwedge x. x \in S \implies \text{norm } (f x + g x) \leq B + C$ 
    by (auto intro!: norm_triangle_le add_mono)
} then show ?thesis
using assms by (fastforce simp: bounded_iff)

```

qed

lemma bounded_plus:

```

fixes  $S :: 'a :: \text{real\_normed\_vector set}$ 
assumes bounded  $S$  bounded  $T$ 
shows bounded (( $\lambda(x,y). x + y$ ) ' ( $S \times T$ ))
using bounded_plus_comp [of  $\text{fst } S \times T \text{ snd}$ ] assms
by (auto simp: split_def split: if_split_asm)

```

lemma bounded_minus_comp:

```

bounded ( $f$  '  $S$ )  $\implies$  bounded ( $g$  '  $S$ )  $\implies$  bounded (( $\lambda x. f x - g x$ ) '  $S$ )
for  $f g :: 'a \Rightarrow 'b :: \text{real\_normed\_vector}$ 
using bounded_plus_comp [of  $f S \lambda x. - g x$ ]
by auto

```

lemma bounded_minus:

```

fixes  $S :: 'a :: \text{real\_normed\_vector set}$ 
assumes bounded  $S$  bounded  $T$ 
shows bounded (( $\lambda(x,y). x - y$ ) ' ( $S \times T$ ))
using bounded_minus_comp [of  $\text{fst } S \times T \text{ snd}$ ] assms
by (auto simp: split_def split: if_split_asm)

```

lemma bounded_sums:

```

fixes  $S :: 'a :: \text{real\_normed\_vector set}$ 
assumes bounded  $S$  and bounded  $T$ 
shows bounded ( $\bigcup x \in S. \bigcup y \in T. \{x + y\}$ )
using assms by (simp add: bounded_iff) (meson norm_triangle_mono)

```

lemma bounded_differences:

```

fixes  $S :: 'a :: \text{real\_normed\_vector set}$ 
assumes bounded  $S$  and bounded  $T$ 
shows bounded ( $\bigcup x \in S. \bigcup y \in T. \{x - y\}$ )

```

using *assms* **by** (*simp* *add: bounded_iff*) (*meson* *add_mono norm_triangle_le_diff*)

lemma *not_bounded_UNIV*[*simp*]:
 $\neg \text{bounded } (\text{UNIV} :: 'a::\{\text{real_normed_vector}, \text{perfect_space}\} \text{ set})$
proof (*auto simp: bounded_pos not_le*)
obtain $x :: 'a$ **where** $x \neq 0$
using *perfect_choose_dist* [*OF zero_less_one*] **by** *fast*
fix $b :: \text{real}$
assume $b > 0$
have $b1: b + 1 \geq 0$
using b **by** *simp*
with $\langle x \neq 0 \rangle$ **have** $b < \text{norm } (\text{scaleR } (b + 1) (\text{sgn } x))$
by (*simp add: norm_sgn*)
then show $\exists x::'a. b < \text{norm } x ..$
qed

corollary *cobounded_imp_unbounded*:
fixes $S :: 'a::\{\text{real_normed_vector}, \text{perfect_space}\} \text{ set}$
shows $\text{bounded } (- S) \implies \neg \text{bounded } S$
using *bounded_Un* [*of S -S*] **by** (*simp*)

4.32.13 Relations among convergence and absolute convergence for power series

lemma *summable_imp_bounded*:
fixes $f :: \text{nat} \Rightarrow 'a::\text{real_normed_vector}$
shows $\text{summable } f \implies \text{bounded } (\text{range } f)$
by (*frule summable_LIMSEQ_zero*) (*simp add: convergent_imp_bounded*)

lemma *summable_imp_sums_bounded*:
 $\text{summable } f \implies \text{bounded } (\text{range } (\lambda n. \text{sum } f \{..<n\}))$
by (*auto simp: summable_def sums_def dest: convergent_imp_bounded*)

lemma *power_series_conv_imp_absconv_weak*:
fixes $a:: \text{nat} \Rightarrow 'a::\{\text{real_normed_div_algebra}, \text{banach}\}$ **and** $w :: 'a$
assumes $\text{sum: summable } (\lambda n. a\ n * z^{\wedge} n)$ **and** $\text{no: norm } w < \text{norm } z$
shows $\text{summable } (\lambda n. \text{of_real}(\text{norm}(a\ n)) * w^{\wedge} n)$
proof –
obtain M **where** $M: \bigwedge x. \text{norm } (a\ x * z^{\wedge} x) \leq M$
using *summable_imp_bounded* [*OF sum*] **by** (*force simp: bounded_iff*)
show *?thesis*
proof (*rule series_comparison_complex*)
have $\bigwedge n. \text{norm } (a\ n) * \text{norm } z^{\wedge} n \leq M$
by (*metis (no_types) M norm_mult norm_power*)
then show $\text{summable } (\lambda n. \text{complex_of_real } (\text{norm } (a\ n)) * \text{norm } w^{\wedge} n)$
using *Abel_lemma no_norm_ge_zero summable_of_real* **by** *blast*
qed (*auto simp: norm_mult norm_power*)
qed

4.32.14 Normed spaces with the Heine-Borel property

lemma *not_compact_UNIV*[*simp*]:
fixes $s :: 'a::\{\text{real_normed_vector}, \text{perfect_space}, \text{heine_borel}\}$ *set*
shows $\neg \text{compact } (\text{UNIV}::'a \text{ set})$
by (*simp add: compact_eq_bounded_closed*)

lemma *not_compact_space_euclideanreal* [*simp*]: $\neg \text{compact_space euclideanreal}$
by (*simp add: compact_space_def*)

Representing sets as the union of a chain of compact sets.

lemma *closed_Union_compact_subsets*:
fixes $S :: 'a::\{\text{heine_borel}, \text{real_normed_vector}\}$ *set*
assumes *closed S*
obtains F **where** $\bigwedge n. \text{compact}(F n) \wedge n. F n \subseteq S \wedge n. F n \subseteq F(\text{Suc } n)$
 $(\bigcup n. F n) = S \wedge K. [\text{compact } K; K \subseteq S] \implies \exists N. \forall n \geq N. K \subseteq$

$F n$

proof

show $\text{compact } (S \cap \text{cball } 0 \text{ (of_nat } n))$ **for** n
using *assms compact_eq_bounded_closed* **by** *auto*

next

show $(\bigcup n. S \cap \text{cball } 0 \text{ (real } n)) = S$
by (*auto simp: real_arch_simple*)

next

fix $K :: 'a \text{ set}$

assume $\text{compact } K \wedge K \subseteq S$

then obtain N **where** $K \subseteq \text{cball } 0 N$

by (*meson bounded_pos mem_cball_0 compact_imp_bounded_subsetI*)

then show $\exists N. \forall n \geq N. K \subseteq S \cap \text{cball } 0 \text{ (real } n)$

by (*metis of_nat_le_iff Int_subset_iff <K ⊆ S> real_arch_simple subset_cball subset_trans*)

qed *auto*

4.32.15 Intersecting chains of compact sets and the Baire property

proposition *bounded_closed_chain*:

fixes $\mathcal{F} :: 'a::\text{heine_borel}$ *set set*

assumes $B \in \mathcal{F}$ *bounded B* **and** $\mathcal{F}: \bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ **and** $\{\} \notin \mathcal{F}$

and chain: $\bigwedge S T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$

shows $\bigcap \mathcal{F} \neq \{\}$

proof –

have $B \cap \bigcap \mathcal{F} \neq \{\}$

proof (*rule compact_imp_fip*)

show $\text{compact } B \wedge T. T \in \mathcal{F} \implies \text{closed } T$

by (*simp_all add: assms compact_eq_bounded_closed*)

show $[\text{finite } \mathcal{G}; \mathcal{G} \subseteq \mathcal{F}] \implies B \cap \bigcap \mathcal{G} \neq \{\}$ **for** \mathcal{G}

proof (*induction* \mathcal{G} *rule: finite_induct*)

case *empty*

with *assms* **show** *?case* **by** *force*

```

next
  case (insert U G)
  then have U ∈ F and ne: B ∩ ⋂ G ≠ {} by auto
  then consider B ⊆ U | U ⊆ B
    using ⟨B ∈ F⟩ chain by blast
  then show ?case
  proof cases
    case 1
    then show ?thesis
      using Int_left_commute ne by auto
  next
    case 2
    have U ≠ {}
      using ⟨U ∈ F⟩ ⟨{} ∉ F⟩ by blast
    moreover
    have False if ⋀x. x ∈ U ⇒ ∃ Y ∈ G. x ∉ Y
    proof -
      have ⋀x. x ∈ U ⇒ ∃ Y ∈ G. Y ⊆ U
        by (metis chain contra_subsetD insert.prem insert_subset that)
      then obtain Y where Y ∈ G Y ⊆ U
        by (metis all_not_in_conv ⟨U ≠ {}⟩)
      moreover obtain x where x ∈ ⋂ G
        by (metis Int_emptyI ne)
      ultimately show ?thesis
        by (metis Inf_lower subset_eq that)
    qed
  with 2 show ?thesis
    by blast
  qed
qed
qed
qed
then show ?thesis by blast
qed

corollary compact_chain:
  fixes F :: 'a::heine_borel set set
  assumes ⋀S. S ∈ F ⇒ compact S {} ∉ F
    ⋀S T. S ∈ F ∧ T ∈ F ⇒ S ⊆ T ∨ T ⊆ S
  shows ⋂ F ≠ {}
proof (cases F = {})
  case True
  then show ?thesis by auto
next
  case False
  show ?thesis
    by (metis False all_not_in_conv assms compact_imp_bounded compact_imp_closed
      bounded_closed_chain)
qed

```

```

lemma compact_nest:
  fixes F :: 'a::linorder  $\Rightarrow$  'b::heine_borel set
  assumes F:  $\bigwedge n. \text{compact}(F\ n) \wedge n. F\ n \neq \{\}$  and mono:  $\bigwedge m\ n. m \leq n \implies F\ m \subseteq F\ n$ 
  shows  $\bigcap (\text{range } F) \neq \{\}$ 
proof -
  have *:  $\bigwedge S\ T. S \in \text{range } F \wedge T \in \text{range } F \implies S \subseteq T \vee T \subseteq S$ 
    by (metis mono image_iff le_cases)
  show ?thesis
    using F by (intro compact_chain [OF _ _ *]; blast dest: *)
qed

```

The Baire property of dense sets

```

theorem Baire:
  fixes S::'a::{real_normed_vector,heine_borel} set
  assumes closed S countable  $\mathcal{G}$ 
    and ope:  $\bigwedge T. T \in \mathcal{G} \implies \text{openin } (\text{top\_of\_set } S) T \wedge S \subseteq \text{closure } T$ 
  shows  $S \subseteq \text{closure}(\bigcap \mathcal{G})$ 
proof (cases  $\mathcal{G} = \{\}$ )
  case True
  then show ?thesis
    using closure_subset by auto
next
  let ?g = from_nat_into  $\mathcal{G}$ 
  case False
  then have gin:  $\exists g\ n \in \mathcal{G}$  for n
    by (simp add: from_nat_into)
  show ?thesis
  proof (clarsimp simp: closure_approachable)
    fix x and e::real
    assume  $x \in S$   $0 < e$ 
    obtain TF where opeF:  $\bigwedge n. \text{openin } (\text{top\_of\_set } S) (TF\ n)$ 
      and ne:  $\bigwedge n. TF\ n \neq \{\}$ 
      and subg:  $\bigwedge n. S \cap \text{closure}(TF\ n) \subseteq ?g\ n$ 
      and subball:  $\bigwedge n. \text{closure}(TF\ n) \subseteq \text{ball } x\ e$ 
      and decr:  $\bigwedge n. TF(Suc\ n) \subseteq TF\ n$ 
    proof -
      have *:  $\exists Y. (\text{openin } (\text{top\_of\_set } S) Y \wedge Y \neq \{\}) \wedge S \cap \text{closure } Y \subseteq ?g\ n \wedge \text{closure } Y \subseteq \text{ball } x\ e \wedge Y \subseteq U$ 
        if opeU:  $\text{openin } (\text{top\_of\_set } S) U$  and  $U \neq \{\}$  and cloU:  $\text{closure } U \subseteq \text{ball } x\ e$  for U n
      proof -
        obtain T where T:  $\text{open } T\ U = T \cap S$ 
          using  $\langle \text{openin } (\text{top\_of\_set } S) U \rangle$  by (auto simp: openin_subtopology)
        with  $\langle U \neq \{\} \rangle$  have  $T \cap \text{closure } (?g\ n) \neq \{\}$ 
          using gin ope by fastforce
        then have  $T \cap ?g\ n \neq \{\}$ 
          using  $\langle \text{open } T \rangle$  open_Int_closure_eq_empty by blast
        then obtain y where  $y \in U$   $y \in ?g\ n$ 

```

```

    using T ope [of ?g n, OF gin] by (blast dest: openin_imp_subset)
  moreover have openin (top_of_set S) (U ∩ ?g n)
    using gin ope opeU by blast
  ultimately obtain d where U: U ∩ ?g n ⊆ S and d > 0 and d: ball y d
∩ S ⊆ U ∩ ?g n
  by (force simp: openin_contains_ball)
show ?thesis
proof (intro exI conjI)
  show openin (top_of_set S) (S ∩ ball y (d/2))
    by (simp add: openin_open_Int)
  show S ∩ ball y (d/2) ≠ {}
    using ‹0 < d› ‹y ∈ U› opeU openin_imp_subset by fastforce
  have S ∩ closure (S ∩ ball y (d/2)) ⊆ S ∩ closure (ball y (d/2))
    using closure_mono by blast
  also have ... ⊆ ?g n
    using ‹d > 0› d by force
  finally show S ∩ closure (S ∩ ball y (d/2)) ⊆ ?g n .
  have closure (S ∩ ball y (d/2)) ⊆ S ∩ ball y d
  proof -
    have closure (ball y (d/2)) ⊆ ball y d
      using ‹d > 0› by auto
    then have closure (S ∩ ball y (d/2)) ⊆ ball y d
      by (meson closure_mono inf.cobounded2 subset_trans)
    then show ?thesis
      by (simp add: ‹closed S› closure_minimal)
  qed
  also have ... ⊆ ball x e
    using cloU closure_subset d by blast
  finally show closure (S ∩ ball y (d/2)) ⊆ ball x e .
  show S ∩ ball y (d/2) ⊆ U
    using ball_divide_subset_numeral d by blast
  qed
  qed
let ?Φ = λn X. openin (top_of_set S) X ∧ X ≠ {} ∧
  S ∩ closure X ⊆ ?g n ∧ closure X ⊆ ball x e
have closure (S ∩ ball x (e/2)) ⊆ closure(ball x (e/2))
  by (simp add: closure_mono)
also have ... ⊆ ball x e
  using ‹e > 0› by auto
finally have closure (S ∩ ball x (e/2)) ⊆ ball x e .
moreover have openin (top_of_set S) (S ∩ ball x (e/2)) S ∩ ball x (e/2) ≠
{}
  using ‹0 < e› ‹x ∈ S› by auto
ultimately obtain Y where Y: ?Φ 0 Y ∧ Y ⊆ S ∩ ball x (e/2)
  using * [of S ∩ ball x (e/2) 0] by metis
show thesis
proof (rule exE [OF dependent_nat_choice])
  show ∃x. ?Φ 0 x
    using Y by auto

```

```

    show  $\exists Y. ?\Phi (Suc\ n)\ Y \wedge Y \subseteq X$  if  $?\Phi\ n\ X$  for  $X\ n$ 
      using that by (blast intro: *)
    qed (use that in metis)
  qed
  have  $(\bigcap n. S \cap \text{closure}\ (TF\ n)) \neq \{\}$ 
  proof (rule compact_nest)
    show  $\bigwedge n. \text{compact}\ (S \cap \text{closure}\ (TF\ n))$ 
    by (metis closed_closure subball bounded_subset_ballI compact_eq_bounded_closed
closed_Int_compact [OF <closed S>])
    show  $\bigwedge n. S \cap \text{closure}\ (TF\ n) \neq \{\}$ 
    by (metis Int_absorb1 opeF <closed S> closure_eq_empty closure_minimal
ne openin_imp_subset)
    show  $\bigwedge m\ n. m \leq n \implies S \cap \text{closure}\ (TF\ n) \subseteq S \cap \text{closure}\ (TF\ m)$ 
    by (meson closure_mono decr dual_order.refl inf_mono lift_Suc_antimono_le)
  qed
  moreover have  $(\bigcap n. S \cap \text{closure}\ (TF\ n)) \subseteq \{y \in \bigcap \mathcal{G}. \text{dist}\ y\ x < e\}$ 
  proof (clarsimp, intro conjI)
    fix y
    assume  $y \in S$  and  $y: \forall n. y \in \text{closure}\ (TF\ n)$ 
    then show  $\forall T \in \mathcal{G}. y \in T$ 
    by (metis Int_iff from_nat_into_surj [OF <countable G>] subsetD subg)
    show  $\text{dist}\ y\ x < e$ 
    by (metis y dist_commute mem_ball subball subsetCE)
  qed
  ultimately show  $\exists y \in \bigcap \mathcal{G}. \text{dist}\ y\ x < e$ 
  by auto
  qed
qed

```

4.32.16 Continuity

Structural rules for uniform continuity

```

lemma (in bounded_linear) uniformly_continuous_on[continuous_intros]:
  fixes  $g :: \_::\text{metric\_space} \Rightarrow \_$ 
  assumes uniformly_continuous_on  $s\ g$ 
  shows uniformly_continuous_on  $s\ (\lambda x. f\ (g\ x))$ 
  using assms unfolding uniformly_continuous_on_sequentially
  unfolding dist_norm tendsto_norm_zero_iff diff[symmetric]
  by (auto intro: tendsto_zero)

```

```

lemma uniformly_continuous_on_dist[continuous_intros]:
  fixes  $f\ g :: 'a::\text{metric\_space} \Rightarrow 'b::\text{metric\_space}$ 
  assumes uniformly_continuous_on  $s\ f$ 
    and uniformly_continuous_on  $s\ g$ 
  shows uniformly_continuous_on  $s\ (\lambda x. \text{dist}\ (f\ x)\ (g\ x))$ 
proof -
  {
    fix  $a\ b\ c\ d :: 'b$ 
    have  $|\text{dist}\ a\ b - \text{dist}\ c\ d| \leq \text{dist}\ a\ c + \text{dist}\ b\ d$ 

```

```

    using dist_triangle2 [of a b c] dist_triangle2 [of b c d]
    using dist_triangle3 [of c d a] dist_triangle [of a d b]
    by arith
  } note le = this
  {
    fix x y
    assume f: (λn. dist (f (x n)) (f (y n))) → 0
    assume g: (λn. dist (g (x n)) (g (y n))) → 0
    have (λn. |dist (f (x n)) (g (x n)) - dist (f (y n)) (g (y n))|) → 0
      by (rule Lim_transform_bound [OF _ tendsto_add_zero [OF f g]],
        simp add: le)
  }
  then show ?thesis
    using assms unfolding uniformly_continuous_on_sequentially
    unfolding dist_real_def by simp
qed

lemma uniformly_continuous_on_cmul_right [continuous_intros]:
  fixes f :: 'a::real_normed_vector ⇒ 'b::real_normed_algebra
  shows uniformly_continuous_on s f ⇒ uniformly_continuous_on s (λx. f x *
c)
  using bounded_linear.uniformly_continuous_on[OF bounded_linear_mult_left]
  .

lemma uniformly_continuous_on_cmul_left [continuous_intros]:
  fixes f :: 'a::real_normed_vector ⇒ 'b::real_normed_algebra
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s (λx. c * f x)
by (metis assms bounded_linear.uniformly_continuous_on bounded_linear_mult_right)

lemma uniformly_continuous_on_norm [continuous_intros]:
  fixes f :: 'a :: metric_space ⇒ 'b :: real_normed_vector
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s (λx. norm (f x))
  unfolding norm_conv_dist using assms
  by (intro uniformly_continuous_on_dist uniformly_continuous_on_const)

lemma uniformly_continuous_on_cmul [continuous_intros]:
  fixes f :: 'a::metric_space ⇒ 'b::real_normed_vector
  assumes uniformly_continuous_on s f
  shows uniformly_continuous_on s (λx. c *R f(x))
  using bounded_linear_scaleR_right assms
  by (rule bounded_linear.uniformly_continuous_on)

lemma dist_minus:
  fixes x y :: 'a::real_normed_vector
  shows dist (- x) (- y) = dist x y
  unfolding dist_norm minus_diff_minus norm_minus_cancel ..

```

```

lemma uniformly_continuous_on_minus[continuous_intros]:
  fixes  $f :: 'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
  shows uniformly_continuous_on  $s\ f \implies$  uniformly_continuous_on  $s\ (\lambda x. - f\ x)$ 
  unfolding uniformly_continuous_on_def dist_minus .

```

```

lemma uniformly_continuous_on_add[continuous_intros]:
  fixes  $f\ g :: 'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
  assumes uniformly_continuous_on  $s\ f$ 
  and uniformly_continuous_on  $s\ g$ 
  shows uniformly_continuous_on  $s\ (\lambda x. f\ x + g\ x)$ 
  using assms
  unfolding uniformly_continuous_on_sequentially
  unfolding dist_norm_tendsto_norm_zero_iff_add_diff_add
  by (auto intro: tendsto_add_zero)

```

```

lemma uniformly_continuous_on_diff[continuous_intros]:
  fixes  $f :: 'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
  assumes uniformly_continuous_on  $s\ f$ 
  and uniformly_continuous_on  $s\ g$ 
  shows uniformly_continuous_on  $s\ (\lambda x. f\ x - g\ x)$ 
  using assms uniformly_continuous_on_add [of  $s\ f - g$ ]
  by (simp add: fun_Cmpl_def uniformly_continuous_on_minus)

```

```

lemma uniformly_continuous_on_sum [continuous_intros]:
  fixes  $f :: 'a \Rightarrow 'b::metric\_space \Rightarrow 'c::real\_normed\_vector$ 
  shows  $(\bigwedge i. i \in I \implies$  uniformly_continuous_on  $S\ (f\ i)) \implies$  uniformly_continuous_on
   $S\ (\lambda x. \sum_{i \in I. f\ i\ x)$ 
  by (induction I rule: infinite_finite_induct)
  (auto simp: uniformly_continuous_on_add uniformly_continuous_on_const)

```

4.32.17 Arithmetic Preserves Topological Properties

```

lemma open_scaling[intro]:
  fixes  $s :: 'a::real\_normed\_vector\ set$ 
  assumes  $c \neq 0$ 
  and open  $s$ 
  shows open $((\lambda x. c *_{\mathbb{R}} x) \ ` s)$ 
proof -
  {
  fix  $x$ 
  assume  $x \in s$ 
  then obtain  $e$  where  $e > 0$ 
  and  $e : \forall x'. dist\ x'\ x < e \longrightarrow x' \in s$  using assms(2)[unfolded open_dist,
  THEN bspec [where  $x=x$ ]]
  by auto
  have  $e * |c| > 0$ 
  using assms(1)[unfolded zero_less_abs_iff [symmetric]]  $\langle e > 0 \rangle$  by auto
  moreover

```

```

{
  fix y
  assume  $\text{dist } y (c *_{R} x) < e * |c|$ 
  then have  $\text{norm } (c *_{R} ((1 / c) *_{R} y - x)) < e * \text{norm } c$ 
    by (simp add:  $\langle c \neq 0 \rangle \text{dist\_norm\_scale\_right\_diff\_distrib}$ )
  then have  $\text{norm } ((1 / c) *_{R} y - x) < e$ 
    by (simp add:  $\langle c \neq 0 \rangle$ )
  then have  $y \in (*_{R}) c \text{ ' } s$ 
    using  $\text{rev\_image\_eqI [of } (1 / c) *_{R} y \text{ } s \text{ } y (*_{R}) c]$ 
    by (simp add:  $\langle c \neq 0 \rangle \text{dist\_norm } e$ )
}
ultimately have  $\exists e > 0. \forall x'. \text{dist } x' (c *_{R} x) < e \longrightarrow x' \in (*_{R}) c \text{ ' } s$ 
  by (rule_tac  $x = e * |c|$  in  $\text{exI, auto}$ )
}
then show ?thesis unfolding  $\text{open\_dist}$  by auto
qed

```

```

lemma minus_image_eq_vimage:
  fixes  $A :: 'a::\text{ab\_group\_add set}$ 
  shows  $(\lambda x. - x) \text{ ' } A = (\lambda x. - x) \text{ - ' } A$ 
  by (auto intro!:  $\text{image\_eqI [where } f = \lambda x. - x]$ )

```

```

lemma open_negations:
  fixes  $S :: 'a::\text{real\_normed\_vector set}$ 
  shows  $\text{open } S \implies \text{open } ((\lambda x. - x) \text{ ' } S)$ 
  using  $\text{open\_scaling [of } - 1 \text{ } S]$  by simp

```

```

lemma open_translation:
  fixes  $S :: 'a::\text{real\_normed\_vector set}$ 
  assumes  $\text{open } S$ 
  shows  $\text{open } ((\lambda x. a + x) \text{ ' } S)$ 

```

```

proof -
{
  fix x
  have  $\text{continuous (at } x) (\lambda x. x - a)$ 
    by (intro  $\text{continuous\_diff continuous\_ident continuous\_const}$ )
}
moreover have  $\{x. x - a \in S\} = (+) a \text{ ' } S$ 
  by force
ultimately show ?thesis
  by (metis  $\text{assms continuous\_open\_vimage vimage\_def}$ )
qed

```

```

lemma open_translation_subtract:
  fixes  $S :: 'a::\text{real\_normed\_vector set}$ 
  assumes  $\text{open } S$ 
  shows  $\text{open } ((\lambda x. x - a) \text{ ' } S)$ 
  using  $\text{assms open\_translation [of } S - a]$  by (simp cong:  $\text{image\_cong\_simp}$ )

```



```

lemma open_neg_translation:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open(( $\lambda x. a - x$ ) ' S)
  using open_translation[OF open_negations[OF assms], of a]
  by (auto simp: image_image)

lemma open_affinity:
  fixes S :: 'a::real_normed_vector set
  assumes open S c  $\neq 0$ 
  shows open (( $\lambda x. a + c *_R x$ ) ' S)
proof -
  have *: ( $\lambda x. a + c *_R x$ ) = ( $\lambda x. a + x$ )  $\circ$  ( $\lambda x. c *_R x$ )
    unfolding o_def ..
  have (+) a ' ( $*_R$ ) c ' S = ((+) a  $\circ$  ( $*_R$ ) c) ' S
    by auto
  then show ?thesis
    using assms open_translation[of ( $*_R$ ) c ' S a]
    unfolding *
    by auto
qed

lemma interior_translation:
  interior ((+) a ' S) = (+) a ' (interior S) for S :: 'a::real_normed_vector set
proof (rule set_eqI, rule)
  fix x
  assume x  $\in$  interior ((+) a ' S)
  then obtain e where e > 0 and e: ball x e  $\subseteq$  (+) a ' S
    unfolding mem_interior by auto
  then have ball (x - a) e  $\subseteq$  S
    unfolding subset_eq Ball_def mem_ball dist_norm
    by (auto simp: diff_diff_eq)
  then show x  $\in$  (+) a ' interior S
    unfolding image_iff
    by (metis <0 < e> add commute diff_add_cancel mem_interior)
next
  fix x
  assume x  $\in$  (+) a ' interior S
  then obtain y e where e > 0 and e: ball y e  $\subseteq$  S and y: x = a + y
    unfolding image_iff Bex_def mem_interior by auto
  {
    fix z
    have *: a + y - z = y + a - z by auto
    assume z  $\in$  ball y e
    then have z - a  $\in$  S
      using e[unfolded subset_eq, THEN bspec[where x=z - a]]
      unfolding mem_ball dist_norm y_group_add_class.diff_diff_eq2 *
      by auto
    then have z  $\in$  (+) a ' S
  }

```

```

      unfolding image_iff by (auto intro!: be_xI[where x=z - a])
    }
  then have ball_x_e ⊆ (+) a ' S
    unfolding subset_eq by auto
  then show x ∈ interior ((+) a ' S)
    unfolding mem_interior using ⟨e > 0⟩ by auto
qed

```

```

lemma interior_translation_subtract:
  interior ((λx. x - a) ' S) = (λx. x - a) ' interior S for S :: 'a::real_normed_vector
  set
  using interior_translation [of - a] by (simp cong: image_cong_simp)

```

```

lemma compact_scaling:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact ((λx. c *R x) ' s)
proof -
  let ?f = λx. scaleR c x
  have *: bounded_linear ?f by (rule bounded_linear_scaleR_right)
  show ?thesis
    using compact_continuous_image[of s ?f] continuous_at_imp_continuous_on[of
  s ?f]
    using linear_continuous_at[OF *] assms
    by auto
qed

```

```

lemma compact_negations:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact ((λx. - x) ' s)
  using compact_scaling [OF assms, of - 1] by auto

```

```

lemma compact_sums:
  fixes s t :: 'a::real_normed_vector set
  assumes compact s
    and compact t
  shows compact {x + y | x y. x ∈ s ∧ y ∈ t}
proof -
  have *: {x + y | x y. x ∈ s ∧ y ∈ t} = (λz. fst z + snd z) ' (s × t)
    by (fastforce simp: image_iff)
  have continuous_on (s × t) (λz. fst z + snd z)
    unfolding continuous_on by (rule ballI) (intro tendsto_intros)
  then show ?thesis
    unfolding * using compact_continuous_image compact_Times [OF assms]
  by auto
qed

```

```

lemma compact_differences:
  fixes s t :: 'a::real_normed_vector set
  assumes compact s
    and compact t
  shows compact {x - y | x y. x ∈ s ∧ y ∈ t}
proof -
  have {x - y | x y. x ∈ s ∧ y ∈ t} = {x + y | x y. x ∈ s ∧ y ∈ (uminus ` t)}
    using diff_conv_add_uminus by force
  then show ?thesis
    using compact_sums[OF assms(1) compact_negations[OF assms(2)]] by auto
qed

```

```

lemma compact_sums':
  fixes S :: 'a::real_normed_vector set
  assumes compact S and compact T
  shows compact (⋃ x ∈ S. ⋃ y ∈ T. {x + y})
proof -
  have (⋃ x ∈ S. ⋃ y ∈ T. {x + y}) = {x + y | x y. x ∈ S ∧ y ∈ T}
    by blast
  then show ?thesis
    using compact_sums [OF assms] by simp
qed

```

```

lemma compact_differences':
  fixes S :: 'a::real_normed_vector set
  assumes compact S and compact T
  shows compact (⋃ x ∈ S. ⋃ y ∈ T. {x - y})
proof -
  have (⋃ x ∈ S. ⋃ y ∈ T. {x - y}) = {x - y | x y. x ∈ S ∧ y ∈ T}
    by blast
  then show ?thesis
    using compact_differences [OF assms] by simp
qed

```

```

lemma compact_translation:
  compact ((+) a ` s) if compact s for s :: 'a::real_normed_vector set
proof -
  have {x + y | x y. x ∈ s ∧ y ∈ {a}} = (λx. a + x) ` s
    by auto
  then show ?thesis
    using compact_sums [OF that compact_sing [of a]] by auto
qed

```

```

lemma compact_translation_subtract:
  compact ((λx. x - a) ` s) if compact s for s :: 'a::real_normed_vector set
  using that compact_translation [of s - a] by (simp cong: image_cong_simp)

```

```

lemma compact_affinity:
  fixes s :: 'a::real_normed_vector set

```

```

  assumes compact s
  shows compact (( $\lambda x. a + c *_R x$ ) ' s)
proof -
  have (+) a ' (*R) c ' s = ( $\lambda x. a + c *_R x$ ) ' s
  by auto
  then show ?thesis
  using compact_translation[OF compact_scaling[OF assms], of a c] by auto
qed

```

```

lemma closed_scaling:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows closed (( $\lambda x. c *_R x$ ) ' S)
proof (cases c = 0)
  case True then show ?thesis
  by (auto simp: image_constant_conv)
next
  case False
  from assms have closed (( $\lambda x. inverse\ c *_R x$ ) -' S)
  by (simp add: continuous_closed_vimage)
  also have ( $\lambda x. inverse\ c *_R x$ ) -' S = ( $\lambda x. c *_R x$ ) ' S
  using <c ≠ 0> by (auto elim: image_eqI [rotated])
  finally show ?thesis .
qed

```

```

lemma closed_negations:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows closed (( $\lambda x. -x$ ) ' S)
  using closed_scaling[OF assms, of - 1] by simp

```

```

lemma compact_closed_sums:
  fixes S :: 'a::real_normed_vector set
  assumes compact S and closed T
  shows closed ( $\bigcup x \in S. \bigcup y \in T. \{x + y\}$ )
proof -
  let ?S = {x + y | x y. x ∈ S ∧ y ∈ T}
  {
    fix x l
    assume as:  $\forall n. x\ n \in ?S$  (x  $\longrightarrow$  l) sequentially
    from as(1) obtain f where f:  $\forall n. x\ n = fst\ (f\ n) + snd\ (f\ n)$   $\forall n. fst\ (f\ n) \in S$   $\forall n. snd\ (f\ n) \in T$ 
    using choice[of  $\lambda n\ y. x\ n = (fst\ y) + (snd\ y) \wedge fst\ y \in S \wedge snd\ y \in T$ ] by auto
    obtain l' r where l' ∈ S and r: strict_mono r and lr: ((( $\lambda n. fst\ (f\ n)$ )  $\circ$  r)  $\longrightarrow$  l') sequentially
    using assms(1)[unfolded compact_def, THEN spec[where x= $\lambda n. fst\ (f\ n)$ ]]
  }
  using f(2) by auto
  have (( $\lambda n. snd\ (f\ (r\ n))$ )  $\longrightarrow$  l - l') sequentially

```

```

    using tendsto_diff[OF LIMSEQ_subseq_LIMSEQ[OF as(2) r] lr] and f(1)
    unfolding o_def
    by auto
  then have  $l - l' \in T$ 
    using assms(2)[unfolded closed_sequential_limits,
      THEN spec[where  $x=\lambda n. \text{snd } (f (r n))$ ],
      THEN spec[where  $x=l - l'$ ]]
    using f(3)
    by auto
  then have  $l \in ?S$ 
    using  $\langle l' \in S \rangle$  by force
}
moreover have  $?S = (\bigcup x \in S. \bigcup y \in T. \{x + y\})$ 
  by force
ultimately show ?thesis
  unfolding closed_sequential_limits
  by (metis (no_types, lifting))
qed

```

```

lemma closed_compact_sums:
  fixes  $S T :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes closed  $S$  compact  $T$ 
  shows closed  $(\bigcup x \in S. \bigcup y \in T. \{x + y\})$ 
proof -
  have  $(\bigcup x \in T. \bigcup y \in S. \{x + y\}) = (\bigcup x \in S. \bigcup y \in T. \{x + y\})$ 
    by auto
  then show ?thesis
    using compact_closed_sums[OF assms(2,1)] by simp
qed

```

```

lemma compact_closed_differences:
  fixes  $S T :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes compact  $S$  closed  $T$ 
  shows closed  $(\bigcup x \in S. \bigcup y \in T. \{x - y\})$ 
proof -
  have  $(\bigcup x \in S. \bigcup y \in \text{uminus } ' T. \{x + y\}) = (\bigcup x \in S. \bigcup y \in T. \{x - y\})$ 
    by force
  then show ?thesis
    by (metis assms closed_negations compact_closed_sums)
qed

```

```

lemma closed_compact_differences:
  fixes  $S T :: 'a::\text{real\_normed\_vector\_set}$ 
  assumes closed  $S$  compact  $T$ 
  shows closed  $(\bigcup x \in S. \bigcup y \in T. \{x - y\})$ 
proof -
  have  $(\bigcup x \in S. \bigcup y \in \text{uminus } ' T. \{x + y\}) = \{x - y \mid x y. x \in S \wedge y \in T\}$ 
    by auto
  then show ?thesis

```

using *closed_compact_sums*[*OF assms(1)*] *compact_negations*[*OF assms(2)*] by *simp*
qed

lemma *closed_translation*:

closed $((+) a \text{ ' } S)$ if *closed* *S* for *a* :: 'a::real_normed_vector

proof –

have $(\bigcup x \in \{a\}. \bigcup y \in S. \{x + y\}) = ((+) a \text{ ' } S)$ by *auto*

then show ?thesis

using *compact_closed_sums* [*OF compact_sing* [*of a*] *that*] by *auto*

qed

lemma *closed_translation_subtract*:

closed $((\lambda x. x - a) \text{ ' } S)$ if *closed* *S* for *a* :: 'a::real_normed_vector

using *that* *closed_translation* [*of S - a*] by (*simp cong: image_cong_simp*)

lemma *closure_translation*:

closure $((+) a \text{ ' } s) = (+) a \text{ ' } \textit{closure } s$ for *a* :: 'a::real_normed_vector

proof –

have *: $(+) a \text{ ' } (- s) = - (+) a \text{ ' } s$

by (*auto intro!: image_eqI* [**where** $x = x - a$ for *x*])

show ?thesis

using *interior_translation* [*of a - s, symmetric*]

by (*simp add: closure_interior_translation_Comp* *)

qed

lemma *closure_translation_subtract*:

closure $((\lambda x. x - a) \text{ ' } s) = (\lambda x. x - a) \text{ ' } \textit{closure } s$ for *a* :: 'a::real_normed_vector

using *closure_translation* [*of - a s*] by (*simp cong: image_cong_simp*)

lemma *frontier_translation*:

frontier $((+) a \text{ ' } s) = (+) a \text{ ' } \textit{frontier } s$ for *a* :: 'a::real_normed_vector

by (*auto simp add: frontier_def translation_diff interior_translation closure_translation*)

lemma *frontier_translation_subtract*:

frontier $((+) a \text{ ' } s) = (+) a \text{ ' } \textit{frontier } s$ for *a* :: 'a::real_normed_vector

by (*auto simp add: frontier_def translation_diff interior_translation closure_translation*)

lemma *sphere_translation*:

sphere $(a + c) r = (+) a \text{ ' } \textit{sphere } c r$ for *a* :: 'n::real_normed_vector

by (*auto simp: dist_norm algebra_simps intro!: image_eqI* [**where** $x = x - a$ for *x*])

lemma *sphere_translation_subtract*:

sphere $(c - a) r = (\lambda x. x - a) \text{ ' } \textit{sphere } c r$ for *a* :: 'n::real_normed_vector

using *sphere_translation* [*of - a c*] by (*simp cong: image_cong_simp*)

lemma *cball_translation*:

cball $(a + c) r = (+) a \text{ ' } \textit{cball } c r$ for *a* :: 'n::real_normed_vector

by (auto simp: dist_norm algebra_simps intro!: image_eqI [where $x = x - a$ for x])

lemma *cball_translation_subtract*:

cball $(c - a) r = (\lambda x. x - a) \text{ ` } \textit{cball } c r$ **for** $a :: 'n::\textit{real_normed_vector}$
using *cball_translation* [of $- a c$] **by** (*simp cong: image_cong_simp*)

lemma *ball_translation*:

ball $(a + c) r = (+) a \text{ ` } \textit{ball } c r$ **for** $a :: 'n::\textit{real_normed_vector}$
by (auto simp: dist_norm algebra_simps intro!: image_eqI [where $x = x - a$ for x])

lemma *ball_translation_subtract*:

ball $(c - a) r = (\lambda x. x - a) \text{ ` } \textit{ball } c r$ **for** $a :: 'n::\textit{real_normed_vector}$
using *ball_translation* [of $- a c$] **by** (*simp cong: image_cong_simp*)

4.32.18 Homeomorphisms

lemma *homeomorphic_scaling*:

fixes $S :: 'a::\textit{real_normed_vector } \textit{set}$
assumes $c \neq 0$
shows S *homeomorphic* $((\lambda x. c *_{\mathbb{R}} x) \text{ ` } S)$
unfolding *homeomorphic_minimal*
apply (*rule_tac* $x = \lambda x. c *_{\mathbb{R}} x$ **in** *exI*)
apply (*rule_tac* $x = \lambda x. (1 / c) *_{\mathbb{R}} x$ **in** *exI*)
using *assms* **by** (*auto simp: continuous_intros*)

lemma *homeomorphic_translation*:

fixes $S :: 'a::\textit{real_normed_vector } \textit{set}$
shows S *homeomorphic* $((\lambda x. a + x) \text{ ` } S)$
unfolding *homeomorphic_minimal*
apply (*rule_tac* $x = \lambda x. a + x$ **in** *exI*)
apply (*rule_tac* $x = \lambda x. -a + x$ **in** *exI*)
by (*auto simp: continuous_intros*)

lemma *homeomorphic_affinity*:

fixes $S :: 'a::\textit{real_normed_vector } \textit{set}$
assumes $c \neq 0$
shows S *homeomorphic* $((\lambda x. a + c *_{\mathbb{R}} x) \text{ ` } S)$

proof –

have $*$: $(+) a \text{ ` } (*_{\mathbb{R}}) c \text{ ` } S = (\lambda x. a + c *_{\mathbb{R}} x) \text{ ` } S$ **by** *auto*
show *?thesis*

by (*metis * assms homeomorphic_scaling homeomorphic_trans homeomorphic_translation*)

qed

lemma *homeomorphic_balls*:

fixes $a b :: 'a::\textit{real_normed_vector}$
assumes $0 < d \ 0 < e$

```

shows (ball a d) homeomorphic (ball b e) (is ?th)
and (cball a d) homeomorphic (cball b e) (is ?cth)
proof -
show ?th unfolding homeomorphic_minimal
  apply(rule_tac x= $\lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
  apply(rule_tac x= $\lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
  using assms
  by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_less_eq)
show ?cth unfolding homeomorphic_minimal
  apply(rule_tac x= $\lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
  apply(rule_tac x= $\lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
  using assms
  by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_le_eq)
qed

```

```

lemma homeomorphic_spheres:
  fixes a b :: 'a::real_normed_vector
  assumes 0 < d 0 < e
  shows (sphere a d) homeomorphic (sphere b e)
unfolding homeomorphic_minimal
  apply(rule_tac x= $\lambda x. b + (e/d) *_{\mathbb{R}} (x - a)$  in exI)
  apply(rule_tac x= $\lambda x. a + (d/e) *_{\mathbb{R}} (x - b)$  in exI)
  using assms
  by (auto intro!: continuous_intros simp: dist_commute dist_norm pos_divide_less_eq)

```

```

lemma homeomorphic_ball01_UNIV:
  ball (0::'a::real_normed_vector) 1 homeomorphic (UNIV::'a set)
  (is ?B homeomorphic ?U)
proof
  have  $x \in (\lambda z. z /_{\mathbb{R}} (1 - \text{norm } z))$  ' ball 0 1 for  $x::'a$ 
  apply (rule_tac  $x=x /_{\mathbb{R}} (1 + \text{norm } x)$  in image_eqI)
  apply (auto simp: field_split_simps)
  using norm_ge_zero [of x] apply linarith+
  done
  then show  $(\lambda z::'a. z /_{\mathbb{R}} (1 - \text{norm } z))$  ' ?B = ?U
  by blast
  have  $x \in \text{range } (\lambda z. (1 / (1 + \text{norm } z)) *_{\mathbb{R}} z)$  if  $\text{norm } x < 1$  for  $x::'a$ 
  using that
  by (rule_tac  $x=x /_{\mathbb{R}} (1 - \text{norm } x)$  in image_eqI) (auto simp: field_split_simps)
  then show  $(\lambda z::'a. z /_{\mathbb{R}} (1 + \text{norm } z))$  ' ?U = ?B
  by (force simp: field_split_simps dest: add_less_zeroD)
  show continuous_on (ball 0 1)  $(\lambda z. z /_{\mathbb{R}} (1 - \text{norm } z))$ 
  by (rule continuous_intros | force)+
  have 0:  $\bigwedge z. 1 + \text{norm } z \neq 0$ 
  by (metis (no_types) le_add_same_cancel1 norm_ge_zero not_one_le_zero)
  then show continuous_on UNIV  $(\lambda z. z /_{\mathbb{R}} (1 + \text{norm } z))$ 
  by (auto intro!: continuous_intros)
  show  $\bigwedge x. x \in \text{ball } 0 \ 1 \implies$ 
     $x /_{\mathbb{R}} (1 - \text{norm } x) /_{\mathbb{R}} (1 + \text{norm } (x /_{\mathbb{R}} (1 - \text{norm } x))) = x$ 

```



```

  by (auto simp: field_split_simps)
  show  $\bigwedge y. y /_R (1 + \text{norm } y) /_R (1 - \text{norm } (y /_R (1 + \text{norm } y))) = y$ 
    using 0 by (auto simp: field_split_simps)
qed

```

proposition *homeomorphic_ball_UNIV*:

```

  fixes a :: 'a::real_normed_vector
  assumes 0 < r shows ball a r homeomorphic (UNIV:: 'a set)
  using assms homeomorphic_ball01_UNIV homeomorphic_balls(1) homeomor-
  phic_trans zero_less_one by blast

```

4.32.19 Discrete

lemma *finite_implies_discrete*:

```

  fixes S :: 'a::topological_space set
  assumes finite (f ' S)
  shows  $(\forall x \in S. \exists e > 0. \forall y. y \in S \wedge f y \neq f x \longrightarrow e \leq \text{norm } (f y - f x))$ 
  proof -
    have  $\exists e > 0. \forall y. y \in S \wedge f y \neq f x \longrightarrow e \leq \text{norm } (f y - f x)$  if  $x \in S$  for  $x$ 
    proof (cases f ' S - {f x} = {})
      case True
        with zero_less_numeral show ?thesis
          by (fastforce simp add: Set.image_subset_iff cong: conj_cong)
      case False
        then obtain z where  $z \in S \wedge f z \neq f x$ 
          by blast
        moreover have finn:  $\text{finite } \{\text{norm } (z - f x) \mid z. z \in f ' S - \{f x\}\}$ 
          using assms by simp
        ultimately have *:  $0 < \text{Inf}\{\text{norm}(z - f x) \mid z. z \in f ' S - \{f x\}\}$ 
          by (force intro: finite_imp_less_Inf)
        show ?thesis
          by (force intro!: * cInf_le_finite [OF finn])
    qed
  with assms show ?thesis
    by blast
qed

```

4.32.20 Completeness of "Isometry" (up to constant bounds)

lemma *cauchy_isometric*:— TODO: rename lemma to *Cauchy_isometric*

```

  assumes e:  $e > 0$ 
    and s: subspace s
    and f: bounded_linear f
    and normf:  $\forall x \in s. \text{norm } (f x) \geq e * \text{norm } x$ 
    and xs:  $\forall n. x n \in s$ 
    and cf: Cauchy (f o x)
  shows Cauchy x
  proof -
    interpret f: bounded_linear f by fact

```

```

have  $\exists N. \forall n \geq N. \text{norm } (x\ n - x\ N) < d$  if  $d > 0$  for  $d :: \text{real}$ 
proof -
  from that obtain  $N$  where  $N: \forall n \geq N. \text{norm } (f\ (x\ n) - f\ (x\ N)) < e * d$ 
  using cf[unfolded Cauchy_def o_def dist_norm, THEN spec[where  $x=e*d$ ]]
e
  by auto
  have  $\text{norm } (x\ n - x\ N) < d$  if  $n \geq N$  for  $n$ 
  proof -
    have  $e * \text{norm } (x\ n - x\ N) \leq \text{norm } (f\ (x\ n) - f\ (x\ N))$ 
      using subspace_diff[OF  $s$ , of  $x\ n\ x\ N$ ]
      using xs[THEN spec[where  $x=N$ ]] and xs[THEN spec[where  $x=n$ ]]
      using normf[THEN bspec[where  $x=x\ n - x\ N$ ]]
    by auto
    also have  $\text{norm } (f\ (x\ n) - f\ (x\ N)) < e * d$ 
      using  $\langle N \leq n \rangle N$  unfolding f.diff[symmetric] by auto
    finally show ?thesis
      using  $\langle e > 0 \rangle$  by simp
  qed
  then show ?thesis by auto
qed
then show ?thesis
  by (simp add: Cauchy_altdef2 dist_norm)
qed

```

lemma complete_isometric_image:

```

assumes  $0 < e$ 
  and  $s$ : subspace  $s$ 
  and  $f$ : bounded_linear  $f$ 
  and normf:  $\forall x \in s. \text{norm}(f\ x) \geq e * \text{norm}(x)$ 
  and  $cs$ : complete  $s$ 
shows complete  $(f\ 's)$ 
proof -
  have  $\exists l \in f\ 's. (g \longrightarrow l)$  sequentially
  if  $as: \forall n::\text{nat}. g\ n \in f\ 's$  and  $cfg: \text{Cauchy } g$  for  $g$ 
  proof -
    from that obtain  $x$  where  $\forall n. x\ n \in s \wedge g\ n = f\ (x\ n)$ 
      using choice[of  $\lambda n\ xa. xa \in s \wedge g\ n = f\ xa$ ] by auto
    then have  $x: \forall n. x\ n \in s \wedge g\ n = f\ (x\ n)$  by auto
    then have  $f \circ x = g$  by (simp add: fun_eq_iff)
    then obtain  $l$  where  $l \in s$  and  $l: (x \longrightarrow l)$  sequentially
      using cs[unfolded complete_def, THEN spec[where  $x=x$ ]]
      using cauchy_isometric[OF  $\langle 0 < e \rangle s\ f\ \text{normf}$ ] and  $cfg$  and  $x(1)$ 
    by auto
    then show ?thesis
      using linear_continuous_at[OF  $f$ , unfolded continuous_at_sequentially,
      THEN spec[where  $x=x$ ], of  $l$ ]
      by (auto simp:  $\langle f \circ x = g \rangle$ )
  qed
  then show ?thesis

```

unfolding complete_def by auto
qed

4.32.21 Connected Normed Spaces

lemma compact_components:

fixes $s :: 'a::heine_borel\ set$

shows $\llbracket compact\ s; c \in components\ s \rrbracket \implies compact\ c$

by (meson bounded_subset closed_components_in_components_subset compact_eq_bounded_closed)

lemma discrete_subset_disconnected:

fixes $S :: 'a::topological_space\ set$

fixes $t :: 'b::real_normed_vector\ set$

assumes conf: continuous_on S f

and no: $\bigwedge x. x \in S \implies \exists e > 0. \forall y. y \in S \wedge f\ y \neq f\ x \longrightarrow e \leq norm\ (f\ y - f\ x)$

shows $f^{-1}\ S \subseteq \{y. connected_component_set\ (f^{-1}\ S)\ y = \{y\}\}$

proof -

{ fix x assume x: $x \in S$

then obtain e where $e > 0$ and ele: $\bigwedge y. \llbracket y \in S; f\ y \neq f\ x \rrbracket \implies e \leq norm\ (f\ y - f\ x)$

using conf no [OF x] by auto

then have e2: $0 \leq e/2$

by simp

define F where $F \equiv connected_component_set\ (f^{-1}\ S)\ (f\ x)$

have False if $y \in S$ and ccs: $f\ y \in F$ and not: $f\ y \neq f\ x$ for y

proof -

define C where $C \equiv cball\ (f\ x)\ (e/2)$

define D where $D \equiv -\ ball\ (f\ x)\ e$

have disj: $C \cap D = \{\}$

unfolding C_def D_def using $\langle 0 < e \rangle$ by fastforce

moreover have FCD: $F \subseteq C \cup D$

proof -

have $t \in C \vee t \in D$ if $t \in F$ for t

proof -

obtain y where $y \in S\ t = f\ y$

using F_def $\langle t \in F \rangle$ connected_component_in by blast

then show ?thesis

by (metis C_def ComplI D_def centre_in_cball dist_norm e2 ele mem_ball norm_minus_commute not_le)

qed

then show ?thesis

by auto

qed

ultimately have $C \cap F = \{\} \vee D \cap F = \{\}$

using connected_closed [of F] $\langle e > 0 \rangle$ not

unfolding C_def D_def

by (metis Elementary_Metric_Spaces.open_ball F_def closed_cball connected_connected_component inf_bot_left open_closed)

```

moreover have  $C \cap F \neq \{\}$ 
  unfolding disjoint_iff
    by (metis FCD ComplD image_eqI mem_Collect_eq subsetD x D_def
F_def Un_iff  $\langle 0 < e \rangle$  centre_in_ball connected_component_refl_eq)
  moreover have  $D \cap F \neq \{\}$ 
    unfolding disjoint_iff
    by (metis ComplI D_def ccs dist_norm ele mem_ball norm_minus_commute
not_not_le that(1))
  ultimately show ?thesis by metis
qed
moreover have connected_component_set  $(f \text{ ` } S) (f x) \subseteq f \text{ ` } S$ 
  by (auto simp: connected_component_in)
ultimately have connected_component_set  $(f \text{ ` } S) (f x) = \{f x\}$ 
  by (auto simp: x F_def)
}
with assms show ?thesis
  by blast
qed

lemma continuous_disconnected_range_constant_eq:
  (connected  $S \longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra}_1.$ 
       $\forall t. \text{continuous\_on } S f \wedge f \text{ ` } S \subseteq t \wedge (\forall y \in t. \text{connected\_component\_set}$ 
t y = \{y\})
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis1)
and continuous_discrete_range_constant_eq:
  (connected  $S \longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra}_1.$ 
      continuous_on  $S f \wedge$ 
      ( $\forall x \in S. \exists e. 0 < e \wedge (\forall y. y \in S \wedge (f y \neq f x) \longrightarrow e \leq \text{norm}(f y - f x))$ 
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis2)
and continuous_finite_range_constant_eq:
  (connected  $S \longleftrightarrow$ 
    ( $\forall f::'a::\text{topological\_space} \Rightarrow 'b::\text{real\_normed\_algebra}_1.$ 
      continuous_on  $S f \wedge \text{finite } (f \text{ ` } S)$ 
       $\longrightarrow f \text{ constant\_on } S$ )) (is ?thesis3)

proof -
have *:  $\bigwedge s t u v. [s \Longrightarrow t; t \Longrightarrow u; u \Longrightarrow v; v \Longrightarrow s]$ 
   $\Longrightarrow (s \longleftrightarrow t) \wedge (s \longleftrightarrow u) \wedge (s \longleftrightarrow v)$ 
  by blast
have ?thesis1  $\wedge$  ?thesis2  $\wedge$  ?thesis3
apply (rule *)
using continuous_disconnected_range_constant
apply (metis image_subset_iff_funcset)
apply (smt (verit, best) discrete_subset_disconnected mem_Collect_eq subsetD
subsetI)
apply (blast dest: finite_implies_discrete)
apply (blast intro!: finite_range_constant_imp_connected)
done

```

```

then show ?thesis1 ?thesis2 ?thesis3
  by blast+
qed

```

```

lemma continuous_discrete_range_constant:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_algebra_1
  assumes S: connected S
    and continuous_on S f
    and  $\bigwedge x. x \in S \implies \exists e > 0. \forall y. y \in S \wedge f y \neq f x \longrightarrow e \leq \text{norm } (f y - f x)$ 
  shows f_constant_on S
  using continuous_discrete_range_constant_eq [THEN iffD1, OF S] assms by
  blast

```

```

lemma continuous_finite_range_constant:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::real_normed_algebra_1
  assumes connected S
    and continuous_on S f
    and finite (f ` S)
  shows f_constant_on S
  using assms continuous_finite_range_constant_eq by blast

```

```

end

```

4.33 Linear Decision Procedure for Normed Spaces

```

theory Norm_Arith
imports HOL-Library.Sum_of_Squares
begin

```

```

lemma sum_sqs_eq:
  fixes x::'a::idom shows  $x * x + y * y = x * (y * 2) \implies y = x$ 
  by algebra

```

```

lemma norm_cmul_rule_thm:
  fixes x :: 'a::real_normed_vector
  shows  $b \geq \text{norm } x \implies |c| * b \geq \text{norm } (\text{scaleR } c x)$ 
  unfolding norm_scaleR
  apply (erule mult_left_mono)
  apply simp
  done

```

```

lemma norm_add_rule_thm:
  fixes x1 x2 :: 'a::real_normed_vector
  shows  $\text{norm } x1 \leq b1 \implies \text{norm } x2 \leq b2 \implies \text{norm } (x1 + x2) \leq b1 + b2$ 
  by (rule order_trans [OF norm_triangle_ineq add_mono])

```

```

lemma ge_iff_diff_ge_0:

```

```

fixes a :: 'a::linordered_ring
shows a ≥ b ≡ a - b ≥ 0
by (simp add: field_simps)

```

```

lemma pth_1:
fixes x :: 'a::real_normed_vector
shows x ≡ scaleR 1 x by simp

```

```

lemma pth_2:
fixes x :: 'a::real_normed_vector
shows x - y ≡ x + -y
by (atomize (full)) simp

```

```

lemma pth_3:
fixes x :: 'a::real_normed_vector
shows - x ≡ scaleR (-1) x
by simp

```

```

lemma pth_4:
fixes x :: 'a::real_normed_vector
shows scaleR 0 x ≡ 0
and scaleR c 0 = (0::'a)
by simp_all

```

```

lemma pth_5:
fixes x :: 'a::real_normed_vector
shows scaleR c (scaleR d x) ≡ scaleR (c * d) x
by simp

```

```

lemma pth_6:
fixes x :: 'a::real_normed_vector
shows scaleR c (x + y) ≡ scaleR c x + scaleR c y
by (simp add: scaleR_right_distrib)

```

```

lemma pth_7:
fixes x :: 'a::real_normed_vector
shows 0 + x ≡ x
and x + 0 ≡ x
by simp_all

```

```

lemma pth_8:
fixes x :: 'a::real_normed_vector
shows scaleR c x + scaleR d x ≡ scaleR (c + d) x
by (simp add: scaleR_left_distrib)

```

```

lemma pth_9:
fixes x :: 'a::real_normed_vector
shows (scaleR c x + z) + scaleR d x ≡ scaleR (c + d) x + z
and scaleR c x + (scaleR d x + z) ≡ scaleR (c + d) x + z

```

and $(\text{scaleR } c \ x + w) + (\text{scaleR } d \ x + z) \equiv \text{scaleR } (c + d) \ x + (w + z)$
by $(\text{simp_all add: algebra_simps})$

lemma *pth_a*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $\text{scaleR } 0 \ x + y \equiv y$
by *simp*

lemma *pth_b*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $\text{scaleR } c \ x + \text{scaleR } d \ y \equiv \text{scaleR } c \ x + \text{scaleR } d \ y$
and $(\text{scaleR } c \ x + z) + \text{scaleR } d \ y \equiv \text{scaleR } c \ x + (z + \text{scaleR } d \ y)$
and $\text{scaleR } c \ x + (\text{scaleR } d \ y + z) \equiv \text{scaleR } c \ x + (\text{scaleR } d \ y + z)$
and $(\text{scaleR } c \ x + w) + (\text{scaleR } d \ y + z) \equiv \text{scaleR } c \ x + (w + (\text{scaleR } d \ y + z))$
by $(\text{simp_all add: algebra_simps})$

lemma *pth_c*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $\text{scaleR } c \ x + \text{scaleR } d \ y \equiv \text{scaleR } d \ y + \text{scaleR } c \ x$
and $(\text{scaleR } c \ x + z) + \text{scaleR } d \ y \equiv \text{scaleR } d \ y + (\text{scaleR } c \ x + z)$
and $\text{scaleR } c \ x + (\text{scaleR } d \ y + z) \equiv \text{scaleR } d \ y + (\text{scaleR } c \ x + z)$
and $(\text{scaleR } c \ x + w) + (\text{scaleR } d \ y + z) \equiv \text{scaleR } d \ y + ((\text{scaleR } c \ x + w) + z)$
by $(\text{simp_all add: algebra_simps})$

lemma *pth_d*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $x + 0 \equiv x$
by *simp*

lemma *norm_imp_pos_and_ge*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $\text{norm } x \equiv n \implies \text{norm } x \geq 0 \wedge n \geq \text{norm } x$
by *atomize auto*

lemma *real_eq_0_iff_le_ge_0*:
fixes $x :: \text{real}$
shows $x = 0 \equiv x \geq 0 \wedge -x \geq 0$
by *arith*

lemma *norm_pths*:
fixes $x :: 'a::\text{real_normed_vector}$
shows $x = y \longleftrightarrow \text{norm } (x - y) \leq 0$
and $x \neq y \longleftrightarrow \neg (\text{norm } (x - y) \leq 0)$
using *norm_ge_zero*[of $x - y$] **by** *auto*

lemmas *arithmetic_simps* =
arith_simps

700

```
add_numeral_special  
add_neg_numeral_special  
mult_1_left  
mult_1_right
```

ML_file \langle normarith.ML \rangle

```
method_setup norm =  $\langle$   
  Scan.succeed (SIMPLE_METHOD' o NormArith.norm_arith_tac)  
 $\rangle$  prove simple linear statements about vector norms
```

Hence more metric properties.

proposition *dist_triangle_add*:

```
fixes x y x' y' :: 'a::real_normed_vector  
shows dist (x + y) (x' + y')  $\leq$  dist x x' + dist y y'  
by norm
```

lemma *dist_triangle_add_half*:

```
fixes x x' y y' :: 'a::real_normed_vector  
shows dist x x'  $< e / 2 \implies$  dist y y'  $< e / 2 \implies$  dist(x + y) (x' + y')  $< e$   
by norm
```

end

Chapter 5

Vector Analysis

```
theory Topology_Euclidean_Space
imports
  Elementary_Normed_Spaces
  Linear_Algebra
  Norm_Arith
begin
```

5.1 Elementary Topology in Euclidean Space

```
lemma euclidean_dist_l2:
  fixes  $x\ y :: 'a :: euclidean\_space$ 
  shows  $dist\ x\ y = L2\_set\ (\lambda i. dist\ (x \cdot i)\ (y \cdot i))\ Basis$ 
  unfolding  $dist\_norm\ norm\_eq\_sqrt\_inner\ L2\_set\_def$ 
  by (subst euclidean_inner) (simp add: power2_eq_square inner_diff_left)
```

```
lemma norm_nth_le:  $norm\ (x \cdot i) \leq norm\ x$  if  $i \in Basis$ 
proof -
  have  $(x \cdot i)^2 = (\sum_{i \in \{i\}} (x \cdot i)^2)$ 
  by simp
  also have  $\dots \leq (\sum_{i \in Basis} (x \cdot i)^2)$ 
  by (intro sum_mono2) (auto simp: that)
  finally show ?thesis
  unfolding  $norm\_conv\_dist\ euclidean\_dist\_l2[of\ x]\ L2\_set\_def$ 
  by (auto intro!: real_le_sqrt)
qed
```

5.1.1 Continuity of the representation WRT an orthogonal basis

```
lemma orthogonal_Basis: pairwise orthogonal Basis
  by (simp add: inner_not_same_Basis orthogonal_def pairwise_def)
```

```
lemma representation_bound:
  fixes  $B :: 'N :: real\_inner\ set$ 
```

assumes *finite B independent B b ∈ B and orth: pairwise orthogonal B*
obtains *m where m > 0 ∧ x. x ∈ span B ⇒ |representation B x b| ≤ m * norm x*
proof
fix *x*
assume *x: x ∈ span B*
have *b ≠ 0*
using *⟨independent B⟩ ⟨b ∈ B⟩ dependent_zero by blast*
have [*simp*]: *b · b' = (if b' = b then (norm b)² else 0)*
if *b ∈ B b' ∈ B for b b'*
using *orth by (simp add: orthogonal_def pairwise_def norm_eq_sqrt_inner that)*
have *norm x = norm (∑ b∈B. representation B x b *_R b)*
using *real_vector.sum_representation_eq [OF ⟨independent B⟩ x ⟨finite B⟩ order_refl]*
by *simp*
also *have ... = sqrt ((∑ b∈B. representation B x b *_R b) · (∑ b∈B. representation B x b *_R b))*
by *(simp add: norm_eq_sqrt_inner)*
also *have ... = sqrt (∑ b∈B. (representation B x b *_R b) · (representation B x b *_R b))*
using *⟨finite B⟩*
by *(simp add: inner_sum_left inner_sum_right if_distrib [of λx. _ * x] cong: if_cong sum.cong_simp)*
also *have ... = sqrt (∑ b∈B. (norm (representation B x b *_R b))²)*
by *(simp add: mult.commute mult.left_commute power2_eq_square)*
also *have ... = sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
by *(simp add: norm_mult_power_mult_distrib)*
finally *have norm x = sqrt (∑ b∈B. (representation B x b)² * (norm b)²) .*
moreover
have *sqrt ((representation B x b)² * (norm b)²) ≤ sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
using *⟨b ∈ B⟩ ⟨finite B⟩ by (auto intro: member_le_sum)*
then *have |representation B x b| ≤ (1 / norm b) * sqrt (∑ b∈B. (representation B x b)² * (norm b)²)*
using *⟨b ≠ 0⟩ by (simp add: field_split_simps real_sqrt_mult del: real_sqrt_le_iff)*
ultimately *show |representation B x b| ≤ (1 / norm b) * norm x*
by *simp*
next
show *0 < 1 / norm b*
using *⟨independent B⟩ ⟨b ∈ B⟩ dependent_zero by auto*
qed

lemma *continuous_on_representation:*

fixes *B :: 'N::euclidean_space set*

assumes *finite B independent B b ∈ B pairwise orthogonal B*

shows *continuous_on (span B) (λx. representation B x b)*

proof

show $\exists d > 0. \forall x' \in \text{span } B. \text{dist } x' x < d \longrightarrow \text{dist } (\text{representation } B x' b) (\text{representation } B x b)$

```

B x b) ≤ e
  if e > 0 x ∈ span B for x e
  proof -
    obtain m where m > 0 and m:  $\bigwedge x. x \in \text{span } B \implies |\text{representation } B x b| \leq m * \text{norm } x$ 
    using assms representation_bound by blast
    show ?thesis
      unfolding dist_norm
    proof (intro exI conjI ballI impI)
      show e/m > 0
        by (simp add: <e > 0> <m > 0>)
      show norm (representation B x' b - representation B x b) ≤ e
        if x': x' ∈ span B and less: norm (x'-x) < e/m for x'
      proof -
        have |representation B (x'-x) b| ≤ m * norm (x'-x)
          using m [of x'-x] <x ∈ span B> span_diff x' by blast
        also have ... < e
          by (metis <m > 0> less mult.commute pos_less_divide_eq)
        finally have |representation B (x'-x) b| ≤ e by simp
      then show ?thesis
        by (simp add: <x ∈ span B> <independent B> representation_diff x')
    qed
  qed
qed
qed
qed

```

5.1.2 Balls in Euclidean Space

```

lemma cball_subset_cball_iff:
  fixes a :: 'a :: euclidean_space
  shows cball a r ⊆ cball a' r'  $\iff$  dist a a' + r ≤ r'  $\vee$  r < 0
    (is ?lhs  $\iff$  ?rhs)
  proof
    assume ?lhs
    then show ?rhs
  proof (cases r < 0)
    case True
      then show ?rhs by simp
    next
      case False
        then have [simp]: r ≥ 0 by simp
        have norm (a - a') + r ≤ r'
        proof (cases a = a')
          case True
            then show ?thesis
              using subsetD [where c = a + r *R (SOME i. i ∈ Basis), OF <?lhs>]
            subsetD [where c = a, OF <?lhs>]
            by (force simp: SOME_Basis dist_norm)
          case False
            next

```

```

    case False
    have norm (a' - (a + (r / norm (a - a')) *R (a - a'))) = norm ((-1 -
(r / norm (a - a'))) *R (a - a'))
    by (simp add: algebra_simps)
    also from ⟨a ≠ a'⟩ have ... = |- norm (a - a') - r|
    by (simp add: divide_simps)
    finally have [simp]: norm (a' - (a + (r / norm (a - a')) *R (a - a'))) =
|norm (a - a') + r|
    by linarith
    from ⟨a ≠ a'⟩ show ?thesis
    using subsetD [where c = a' + (1 + r / norm(a - a')) *R (a - a'), OF
⟨?lhs⟩]
    by (simp add: dist_norm scaleR_add_left)
  qed
  then show ?rhs
  by (simp add: dist_norm)
  qed
qed metric

```

lemma *cball_subset_ball_iff*: $cball\ a\ r \subseteq ball\ a'\ r' \iff dist\ a\ a' + r < r' \vee r < 0$

(is ?lhs \iff ?rhs)

for $a :: 'a::euclidean_space$

proof

assume ?lhs

then show ?rhs

proof (cases $r < 0$)

case True then

show ?rhs by simp

next

case False

then have [simp]: $r \geq 0$ by simp

have $norm\ (a - a') + r < r'$

proof (cases $a = a'$)

case True

then show ?thesis

using subsetD [where $c = a + r *_{R}\ (SOME\ i.\ i \in Basis)$, OF ⟨?lhs⟩]
subsetD [where $c = a$, OF ⟨?lhs⟩]

by (force simp: SOME_Basis dist_norm)

next

case False

have False if $norm\ (a - a') + r \geq r'$

proof -

from that have $|r' - norm\ (a - a')| \leq r$

by (smt (verit, best) ⟨ $0 \leq r$ ⟩ ⟨?lhs⟩ ball_subset_cball cball_subset_cball_iff
dist_norm order_trans)

then show ?thesis

using subsetD [where $c = a + (r' / norm(a - a') - 1) *_{R}\ (a - a')$, OF
⟨?lhs⟩] ⟨ $a \neq a'$ ⟩

```

    apply (simp add: dist_norm)
    apply (simp add: scaleR_left_diff_distrib)
    apply (simp add: field_simps)
  done
qed
then show ?thesis by force
qed
then show ?rhs by (simp add: dist_norm)
qed
next
  assume ?rhs
  then show ?lhs
    by metric
qed

lemma ball_subset_cball_iff: ball a r  $\subseteq$  cball a' r'  $\longleftrightarrow$  dist a a' + r  $\leq$  r'  $\vee$  r  $\leq$ 
0
  (is ?lhs = ?rhs)
  for a :: 'a::euclidean_space
proof (cases r  $\leq$  0)
  case True
  then show ?thesis
    by metric
  next
  case False
  show ?thesis
  proof
    assume ?lhs
    then have (cball a r  $\subseteq$  cball a' r')
      by (metis False closed_cball_closure_ball_closure_closed closure_mono not_less)
    with False show ?rhs
      by (fastforce iff: cball_subset_cball_iff)
  next
    assume ?rhs
    with False show ?lhs
      by metric
  qed
qed
qed

lemma ball_subset_ball_iff:
  fixes a :: 'a :: euclidean_space
  shows ball a r  $\subseteq$  ball a' r'  $\longleftrightarrow$  dist a a' + r  $\leq$  r'  $\vee$  r  $\leq$  0
  (is ?lhs = ?rhs)
proof (cases r  $\leq$  0)
  case True then show ?thesis
    by metric
  next
  case False show ?thesis
  proof

```

```

assume ?lhs
then have  $0 < r'$ 
  using False by metric
then have  $\text{cball } a \ r \subseteq \text{cball } a' \ r'$ 
  by (metis False <?lhs> closure_ball closure_mono not_less)
then show ?rhs
  using False cball_subset_cball_iff by fastforce
qed metric
qed

```

```

lemma ball_eq_ball_iff:
  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  shows  $\text{ball } x \ d = \text{ball } y \ e \longleftrightarrow d \leq 0 \wedge e \leq 0 \vee x=y \wedge d=e$ 
  by (smt (verit, del_insts) ball_empty_ball_subset_cball_iff dist_norm norm_pths(2))

```

```

lemma cball_eq_cball_iff:
  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  shows  $\text{cball } x \ d = \text{cball } y \ e \longleftrightarrow d < 0 \wedge e < 0 \vee x=y \wedge d=e$ 
  by (smt (verit, ccfv_SIG) cball_empty_cball_subset_cball_iff dist_norm norm_pths(2)
  zero_le_dist)

```

```

lemma ball_eq_cball_iff:
  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  shows  $\text{ball } x \ d = \text{cball } y \ e \longleftrightarrow d \leq 0 \wedge e < 0$  (is ?lhs = ?rhs)
  by (smt (verit) ball_eq_empty_ball_subset_cball_iff cball_eq_empty_cball_subset_ball_iff
  order.refl)

```

```

lemma cball_eq_ball_iff:
  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  shows  $\text{cball } x \ d = \text{ball } y \ e \longleftrightarrow d < 0 \wedge e \leq 0$ 
  using ball_eq_cball_iff by blast

```

```

lemma finite_ball_avoid:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes open S finite X  $p \in S$ 
  shows  $\exists e > 0. \forall w \in \text{ball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
proof -
  obtain  $e1$  where  $0 < e1$  and  $e1\_b: \text{ball } p \ e1 \subseteq S$ 
    using open_contains_ball_eq[OF <open S>] assms by auto
  obtain  $e2$  where  $0 < e2$  and  $\forall x \in X. x \neq p \longrightarrow e2 \leq \text{dist } p \ x$ 
    using finite_set_avoid[OF <finite X>, of p] by auto
  hence  $\forall w \in \text{ball } p \ (\min \ e1 \ e2). w \in S \wedge (w \neq p \longrightarrow w \notin X)$  using  $e1\_b$  by auto
  thus  $\exists e > 0. \forall w \in \text{ball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
    using <e2 > 0> <e1 > 0> by (rule_tac  $x = \min \ e1 \ e2$  in  $e1\_b$ ) auto
qed

```

```

lemma finite_cball_avoid:
  fixes  $S :: 'a :: \text{euclidean\_space}$  set

```

```

assumes open S finite X p ∈ S
shows  $\exists e > 0. \forall w \in \text{cball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
proof –
  obtain e1 where  $e1 > 0$  and  $e1: \forall w \in \text{ball } p \ e1. w \in S \wedge (w \neq p \longrightarrow w \notin X)$ 
    using finite_ball_avoid[OF assms] by auto
  define e2 where  $e2 \equiv e1/2$ 
  have  $e2 > 0$  and  $e2 < e1$  unfolding e2_def using  $\langle e1 > 0 \rangle$  by auto
  then have  $\text{cball } p \ e2 \subseteq \text{ball } p \ e1$  by (subst cball_subset_ball_iff, auto)
  then show  $\exists e > 0. \forall w \in \text{cball } p \ e. w \in S \wedge (w \neq p \longrightarrow w \notin X)$  using  $\langle e2 > 0 \rangle$ 
e1 by auto
qed

```

```

lemma dim_cball:
  assumes  $e > 0$ 
  shows  $\text{dim } (\text{cball } (0 :: 'n::\text{euclidean\_space}) \ e) = \text{DIM}('n)$ 
proof –
  {
    fix x ::  $'n::\text{euclidean\_space}$ 
    define y where  $y = (e / \text{norm } x) *_{\mathbb{R}} x$ 
    then have  $y \in \text{cball } 0 \ e$ 
      using assms by auto
    moreover have  $*: x = (\text{norm } x / e) *_{\mathbb{R}} y$ 
      using y_def assms by simp
    moreover from  $*$  have  $x = (\text{norm } x / e) *_{\mathbb{R}} y$ 
      by auto
    ultimately have  $x \in \text{span } (\text{cball } 0 \ e)$ 
      using span_scale[of y cball 0 e norm x/e]
        span_superset[of cball 0 e]
      by (simp add: span_base)
  }
  then have  $\text{span } (\text{cball } 0 \ e) = (\text{UNIV} :: 'n::\text{euclidean\_space} \ \text{set})$ 
    by auto
  then show ?thesis
    using dim_span[of cball (0 :: 'n::euclidean_space) e] by (auto)
qed

```

5.1.3 Boxes

```

abbreviation One ::  $'a::\text{euclidean\_space}$  where
One  $\equiv \sum \text{Basis}$ 

```

```

lemma One_non_0: assumes  $\text{One} = (0 :: 'a::\text{euclidean\_space})$  shows False
proof –
  have dependent (Basis ::  $'a \ \text{set}$ )
    apply (simp add: dependent_finite)
    apply (rule_tac x = \lambda i. 1 in exI)
    using SOME_Basis apply (auto simp: assms)
    done
  with independent_Basis show False by force

```

qed

corollary *One_neq_0*[*iff*]: $One \neq 0$
 by (*metis One_non_0*)

corollary *Zero_neq_One*[*iff*]: $0 \neq One$
 by (*metis One_non_0*)

definition (in *euclidean_space*) *eucl_less* (**infix** $<e$ 50) **where**
 $eucl_less\ a\ b \longleftrightarrow (\forall i \in Basis. a \cdot i < b \cdot i)$

definition *box_eucl_less*: $box\ a\ b = \{x. a <e\ x \wedge x <e\ b\}$

definition *cbox* $a\ b = \{x. \forall i \in Basis. a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i\}$

lemma *box_def*: $box\ a\ b = \{x. \forall i \in Basis. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i\}$
and *in_box_eucl_less*: $x \in box\ a\ b \longleftrightarrow a <e\ x \wedge x <e\ b$
and *mem_box*: $x \in box\ a\ b \longleftrightarrow (\forall i \in Basis. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i)$
 $x \in cbox\ a\ b \longleftrightarrow (\forall i \in Basis. a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$
 by (*auto simp: box_eucl_less eucl_less_def cbox_def*)

lemma *cbox_Pair_eq*: $cbox\ (a, c)\ (b, d) = cbox\ a\ b \times cbox\ c\ d$
 by (*force simp: cbox_def Basis_prod_def*)

lemma *cbox_Pair_iff* [*iff*]: $(x, y) \in cbox\ (a, c)\ (b, d) \longleftrightarrow x \in cbox\ a\ b \wedge y \in cbox\ c\ d$
 by (*force simp: cbox_Pair_eq*)

lemma *cbox_Complex_eq*: $cbox\ (Complex\ a\ c)\ (Complex\ b\ d) = (\lambda(x,y). Complex\ x\ y) \text{ ` } (cbox\ a\ b \times cbox\ c\ d)$
 by (*force simp: cbox_def Basis_complex_def*)

lemma *cbox_Pair_eq_0*: $cbox\ (a, c)\ (b, d) = \{\} \longleftrightarrow cbox\ a\ b = \{\} \vee cbox\ c\ d = \{\}$
 by (*force simp: cbox_Pair_eq*)

lemma *swap_cbox_Pair* [*simp*]: $prod.swap \text{ ` } cbox\ (c, a)\ (d, b) = cbox\ (a,c)\ (b,d)$
 by *auto*

lemma *mem_box_real*[*simp*]:
 $(x::real) \in box\ a\ b \longleftrightarrow a < x \wedge x < b$
 $(x::real) \in cbox\ a\ b \longleftrightarrow a \leq x \wedge x \leq b$
 by (*auto simp: mem_box*)

lemma *box_real*[*simp*]:
fixes $a\ b::real$
shows $box\ a\ b = \{a <..< b\}$ $cbox\ a\ b = \{a .. b\}$
 by *auto*

lemma *box_Int_box*:


```

fixes  $a :: 'a::euclidean\_space$ 
shows  $\text{box } a \ b \cap \text{box } c \ d =$ 
 $\text{box } (\sum_{i \in \text{Basis}. \max (a \cdot i) (c \cdot i) *_{\mathbb{R}} i) (\sum_{i \in \text{Basis}. \min (b \cdot i) (d \cdot i) *_{\mathbb{R}} i)$ 
unfolding  $\text{set\_eq\_iff}$  and  $\text{Int\_iff}$  and  $\text{mem\_box}$  by  $\text{auto}$ 

```

lemma *rational_boxes*:

```

fixes  $x :: 'a::euclidean\_space$ 
assumes  $e > 0$ 
shows  $\exists a \ b. (\forall i \in \text{Basis}. a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}) \wedge x \in \text{box } a \ b \wedge \text{box } a \ b \subseteq \text{ball } x$ 
 $e$ 

```

proof –

```

define  $e'$  where  $e' = e / (2 * \text{sqrt } (\text{real } (\text{DIM } ('a))))$ 
then have  $e: e' > 0$ 
using  $\text{assms}$  by  $(\text{auto})$ 
have  $\exists y. y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$  for  $i$ 
using  $\text{Rats\_dense\_in\_real}$ [ $\text{of } x \cdot i - e' x \cdot i$ ]  $e$  by  $\text{force}$ 
then obtain  $a$  where
 $a: \bigwedge u. a \cdot u \in \mathbb{Q} \wedge a \cdot u < x \cdot u \wedge x \cdot u - a \cdot u < e'$  by  $\text{metis}$ 
have  $\exists y. y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$  for  $i$ 
using  $\text{Rats\_dense\_in\_real}$ [ $\text{of } x \cdot i \ x \cdot i + e'$ ]  $e$  by  $\text{force}$ 
then obtain  $b$  where
 $b: \bigwedge u. b \cdot u \in \mathbb{Q} \wedge x \cdot u < b \cdot u \wedge b \cdot u - x \cdot u < e'$  by  $\text{metis}$ 
let  $?a = \sum_{i \in \text{Basis}. a \cdot i *_{\mathbb{R}} i$  and  $?b = \sum_{i \in \text{Basis}. b \cdot i *_{\mathbb{R}} i$ 
show  $?thesis$ 
proof ( $\text{rule } \text{exI}$ [ $\text{of } ?a$ ],  $\text{rule } \text{exI}$ [ $\text{of } ?b$ ],  $\text{safe}$ )
fix  $y :: 'a$ 
assume  $*$ :  $y \in \text{box } ?a \ ?b$ 
have  $\text{dist } x \ y = \text{sqrt } (\sum_{i \in \text{Basis}. (\text{dist } (x \cdot i) (y \cdot i))^2)$ 
unfolding  $\text{L2\_set\_def}$ [ $\text{symmetric}$ ] by ( $\text{rule } \text{euclidean\_dist\_l2}$ )
also have  $\dots < \text{sqrt } (\sum_{(i::'a) \in \text{Basis}. e'^2 / \text{real } (\text{DIM } ('a)))$ 
proof ( $\text{rule } \text{real\_sqrt\_less\_mono}$ ,  $\text{rule } \text{sum\_strict\_mono}$ )
fix  $i :: 'a$ 
assume  $i: i \in \text{Basis}$ 
have  $a \cdot i < y \cdot i \wedge y \cdot i < b \cdot i$ 
using  $*$   $i$  by ( $\text{auto simp: box\_def}$ )
moreover have  $a \cdot i < x \cdot i \ x \cdot i - a \cdot i < e' \ x \cdot i < b \cdot i \ b \cdot i - x \cdot i < e'$ 
using  $a \ b$  by  $\text{auto}$ 
ultimately have  $|x \cdot i - y \cdot i| < 2 * e'$ 
by  $\text{auto}$ 
then have  $\text{dist } (x \cdot i) (y \cdot i) < e / \text{sqrt } (\text{real } (\text{DIM } ('a)))$ 
unfolding  $e'\_def$  by ( $\text{auto simp: dist\_real\_def}$ )
then have  $(\text{dist } (x \cdot i) (y \cdot i))^2 < (e / \text{sqrt } (\text{real } (\text{DIM } ('a))))^2$ 
by ( $\text{rule } \text{power\_strict\_mono}$ )  $\text{auto}$ 
then show  $(\text{dist } (x \cdot i) (y \cdot i))^2 < e^2 / \text{real } \text{DIM } ('a)$ 
by ( $\text{simp add: power\_divide}$ )
qed  $\text{auto}$ 
also have  $\dots = e$ 
using  $\langle 0 < e \rangle$  by  $\text{simp}$ 
finally show  $y \in \text{ball } x \ e$ 

```

```

    by (auto simp: ball_def)
  qed (use a b in ⟨auto simp: box_def⟩)
qed

```

lemma *open_UNION_box*:

```

  fixes M :: 'a::euclidean_space set
  assumes open M
  defines a' ≡ λf :: 'a ⇒ real × real. (∑ (i::'a)∈Basis. fst (f i) *R i)
  defines b' ≡ λf :: 'a ⇒ real × real. (∑ (i::'a)∈Basis. snd (f i) *R i)
  defines I ≡ {f∈Basis →E Q × Q. box (a' f) (b' f) ⊆ M}
  shows M = (⋃ f∈I. box (a' f) (b' f))
proof -
  have x ∈ (⋃ f∈I. box (a' f) (b' f)) if x ∈ M for x
  proof -
    obtain e where e: e > 0 ball x e ⊆ M
    using openE[OF ⟨open M⟩ ⟨x ∈ M⟩] by auto
    moreover obtain a b where ab:
      x ∈ box a b
      ∀ i ∈ Basis. a · i ∈ Q
      ∀ i ∈ Basis. b · i ∈ Q
      box a b ⊆ ball x e
    using rational_boxes[OF e(1)] by metis
    ultimately show ?thesis
      by (intro UN_I[of λi∈Basis. (a · i, b · i)])
        (auto simp: euclidean_representation I_def a'_def b'_def)
  qed
  then show ?thesis by (auto simp: I_def)
qed

```

corollary *open_countable_Union_open_box*:

```

  fixes S :: 'a :: euclidean_space set
  assumes open S
  obtains D where countable D D ⊆ Pow S ∧ X. X ∈ D ⇒ ∃ a b. X = box a b
  ⋃ D = S
proof -
  let ?a = λf. (∑ (i::'a)∈Basis. fst (f i) *R i)
  let ?b = λf. (∑ (i::'a)∈Basis. snd (f i) *R i)
  let ?I = {f∈Basis →E Q × Q. box (?a f) (?b f) ⊆ S}
  let ?D = (λf. box (?a f) (?b f)) ' ?I
  show ?thesis
  proof
    have countable ?I
      by (simp add: countable_PiE countable_rat)
    then show countable ?D
      by blast
    show ⋃ ?D = S
      using open_UNION_box [OF assms] by metis
  qed auto
qed

```

lemma *rational_cboxes*:

fixes $x :: 'a::\text{euclidean_space}$

assumes $e > 0$

shows $\exists a b. (\forall i \in \text{Basis}. a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}) \wedge x \in \text{cbox } a b \wedge \text{cbox } a b \subseteq \text{ball}$

$x e$

proof –

define e' where $e' = e / (2 * \text{sqrt } (\text{real } (\text{DIM } ('a))))$

then have $e: e' > 0$

using *assms* by *auto*

have $\exists y. y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$ for i

using *Rats_dense_in_real*[of $x \cdot i - e' x \cdot i$] e by *force*

then obtain a where

$a: \forall u. a u \in \mathbb{Q} \wedge a u < x \cdot u \wedge x \cdot u - a u < e'$ by *metis*

have $\exists y. y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$ for i

using *Rats_dense_in_real*[of $x \cdot i x \cdot i + e'$] e by *force*

then obtain b where

$b: \forall u. b u \in \mathbb{Q} \wedge x \cdot u < b u \wedge b u - x \cdot u < e'$ by *metis*

let $?a = \sum i \in \text{Basis}. a i *_R i$ and $?b = \sum i \in \text{Basis}. b i *_R i$

show *?thesis*

proof (rule *exI*[of $?a$], rule *exI*[of $?b$], *safe*)

fix $y :: 'a$

assume $*$: $y \in \text{cbox } ?a ?b$

have $\text{dist } x y = \text{sqrt } (\sum i \in \text{Basis}. (\text{dist } (x \cdot i) (y \cdot i))^2)$

unfolding *L2_set_def*[*symmetric*] by (rule *euclidean_dist_l2*)

also have $\dots < \text{sqrt } (\sum (i::'a) \in \text{Basis}. e'^2 / \text{real } (\text{DIM } ('a)))$

proof (rule *real_sqrt_less_mono*, rule *sum_strict_mono*)

fix $i :: 'a$

assume $i: i \in \text{Basis}$

have $a i \leq y \cdot i \wedge y \cdot i \leq b i$

using $*$ i by (*auto simp: cbox_def*)

moreover have $a i < x \cdot i x \cdot i - a i < e' x \cdot i < b i b i - x \cdot i < e'$

using $a b$ by *auto*

ultimately have $|x \cdot i - y \cdot i| < 2 * e'$

by *auto*

then have $\text{dist } (x \cdot i) (y \cdot i) < e / \text{sqrt } (\text{real } (\text{DIM } ('a)))$

unfolding *e'_def* by (*auto simp: dist_real_def*)

then have $(\text{dist } (x \cdot i) (y \cdot i))^2 < (e / \text{sqrt } (\text{real } (\text{DIM } ('a))))^2$

by (*rule power_strict_mono*) *auto*

then show $(\text{dist } (x \cdot i) (y \cdot i))^2 < e^2 / \text{real } \text{DIM } ('a)$

by (*simp add: power_divide*)

qed *auto*

also have $\dots = e$

using $\langle 0 < e \rangle$ by *simp*

finally show $y \in \text{ball } x e$

by (*auto simp: ball_def*)

next

show $x \in \text{cbox } (\sum i \in \text{Basis}. a i *_R i) (\sum i \in \text{Basis}. b i *_R i)$

using $a b$ *less_imp_le* by (*auto simp: cbox_def*)

qed (use a b cbox_def in auto)
qed

lemma open_UNION_cbox:

fixes $M :: 'a :: euclidean_space\ set$

assumes open M

defines $a' \equiv \lambda f. (\sum (i::'a) \in Basis. fst (f i) *_{\mathbb{R}} i)$

defines $b' \equiv \lambda f. (\sum (i::'a) \in Basis. snd (f i) *_{\mathbb{R}} i)$

defines $I \equiv \{f \in Basis \rightarrow_E \mathbb{Q} \times \mathbb{Q}. cbox (a' f) (b' f) \subseteq M\}$

shows $M = (\bigcup f \in I. cbox (a' f) (b' f))$

proof –

have $x \in (\bigcup f \in I. cbox (a' f) (b' f))$ **if** $x \in M$ **for** x

proof –

obtain e **where** $e: e > 0$ ball $x e \subseteq M$

using openE[OF ‹open M › ‹ $x \in M$ ›] **by** auto

moreover obtain $a\ b$ **where** $ab: x \in cbox\ a\ b \ \forall i \in Basis. a \cdot i \in \mathbb{Q}$
 $\forall i \in Basis. b \cdot i \in \mathbb{Q}$ cbox $a\ b \subseteq$ ball $x\ e$

using rational_cboxes[OF $e(1)$] **by** metis

ultimately show ?thesis

by (intro UN_I[of $\lambda i \in Basis. (a \cdot i, b \cdot i)$])

(auto simp: euclidean_representation I_def a'_def b'_def)

qed

then show ?thesis **by** (auto simp: I_def)

qed

corollary open_countable_Union_open_cbox:

fixes $S :: 'a :: euclidean_space\ set$

assumes open S

obtains \mathcal{D} **where** countable \mathcal{D} $\mathcal{D} \subseteq Pow\ S \ \bigwedge X. X \in \mathcal{D} \implies \exists a\ b. X = cbox\ a\ b \ \bigcup \mathcal{D} = S$

proof –

let ?a = $\lambda f. (\sum (i::'a) \in Basis. fst (f i) *_{\mathbb{R}} i)$

let ?b = $\lambda f. (\sum (i::'a) \in Basis. snd (f i) *_{\mathbb{R}} i)$

let ?I = $\{f \in Basis \rightarrow_E \mathbb{Q} \times \mathbb{Q}. cbox (?a\ f) (?b\ f) \subseteq S\}$

let ?D = $(\lambda f. cbox (?a\ f) (?b\ f)) \text{ ` } ?I$

show ?thesis

proof

have countable ?I

by (simp add: countable_PiE countable_rat)

then show countable ?D

by blast

show $\bigcup ?D = S$

using open_UNION_cbox [OF assms] **by** metis

qed auto

qed

lemma box_eq_empty:

fixes $a :: 'a :: euclidean_space$

shows (box $a\ b = \{\}$) $\longleftrightarrow (\exists i \in Basis. b \cdot i \leq a \cdot i)$ (is ?th1)

```

    and (cbox a b = {})  $\longleftrightarrow$  ( $\exists i \in \text{Basis}. b \cdot i < a \cdot i$ ) (is ?th2)
  proof -
    have False if  $i \in \text{Basis}$  and  $b \cdot i \leq a \cdot i$  and  $x \in \text{cbox } a \ b$  for  $i \ x$ 
      by (smt (verit, ccfv_SIG) mem_box(1) that)
    moreover
    { assume as:  $\forall i \in \text{Basis}. \neg (b \cdot i \leq a \cdot i)$ 
      let ?x =  $(1/2) *_{\mathbb{R}} (a + b)$ 
        { fix i :: 'a
          assume i:  $i \in \text{Basis}$ 
          have  $a \cdot i < b \cdot i$ 
            using as i by fastforce
          then have  $a \cdot i < ((1/2) *_{\mathbb{R}} (a+b)) \cdot i ((1/2) *_{\mathbb{R}} (a+b)) \cdot i < b \cdot i$ 
            by (auto simp: inner_add_left)
        }
      then have  $\text{cbox } a \ b \neq \{\}$ 
        by (metis (no_types, opaque_lifting) emptyE mem_box(1))
    }
    ultimately show ?th1 by blast

    have False if  $i \in \text{Basis}$  and  $b \cdot i < a \cdot i$  and  $x \in \text{cbox } a \ b$  for  $i \ x$ 
      using mem_box(2) that by force
    moreover
    have  $\text{cbox } a \ b \neq \{\}$  if  $\forall i \in \text{Basis}. \neg (b \cdot i < a \cdot i)$ 
      by (metis emptyE linorder_linear mem_box(2) order.strict_iff_not that)
    ultimately show ?th2 by blast
  qed

```

```

lemma box_ne_empty:
  fixes a :: 'a::euclidean_space
  shows  $\text{cbox } a \ b \neq \{\} \longleftrightarrow (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$ 
  and  $\text{box } a \ b \neq \{\} \longleftrightarrow (\forall i \in \text{Basis}. a \cdot i < b \cdot i)$ 
  unfolding box_eq_empty[of a b] by fastforce+

```

```

lemma
  fixes a :: 'a::euclidean_space
  shows cbox_idem [simp]:  $\text{cbox } a \ a = \{a\}$ 
  and box_idem [simp]:  $\text{box } a \ a = \{\}$ 
  unfolding set_eq_iff mem_box eq_iff [symmetric] using euclidean_eq_iff by
  fastforce+

```

```

lemma subset_box_imp:
  fixes a :: 'a::euclidean_space
  shows ( $\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ )  $\implies \text{cbox } c \ d \subseteq \text{cbox } a \ b$ 
  and ( $\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge d \cdot i < b \cdot i$ )  $\implies \text{cbox } c \ d \subseteq \text{box } a \ b$ 
  and ( $\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ )  $\implies \text{box } c \ d \subseteq \text{cbox } a \ b$ 
  and ( $\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge d \cdot i < b \cdot i$ )  $\implies \text{box } c \ d \subseteq \text{box } a \ b$ 
  unfolding subset_eq[unfolded Ball_def] unfolding mem_box
  by (best intro: order_trans less_le_trans le_less_trans less_imp_le)+

```

lemma *box_subset_cbox*:
fixes $a :: 'a::euclidean_space$
shows $\text{box } a \ b \subseteq \text{cbox } a \ b$
unfolding *subset_eq* [*unfolded Ball_def*] *mem_box*
by (*fast intro: less_imp_le*)

lemma *subset_box*:
fixes $a :: 'a::euclidean_space$
shows $\text{cbox } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i \leq d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i)$ (**is** *?th1*)
and $\text{cbox } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i \leq d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i < c \cdot i \wedge d \cdot i < b \cdot i)$ (**is** *?th2*)
and $\text{box } c \ d \subseteq \text{cbox } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i < d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i)$ (**is** *?th3*)
and $\text{box } c \ d \subseteq \text{box } a \ b \longleftrightarrow (\forall i \in \text{Basis}. c \cdot i < d \cdot i) \longrightarrow (\forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i)$ (**is** *?th4*)
proof –
let $?lesscd = \forall i \in \text{Basis}. c \cdot i < d \cdot i$
let $?lerhs = \forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$
show *?th1 ?th2*
by (*fastforce simp: mem_box*)
have *acdb*: $a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$
if $i \in \text{Basis}$ **and** *box*: $\text{box } c \ d \subseteq \text{cbox } a \ b$ **and** *cd*: $\bigwedge i. i \in \text{Basis} \implies c \cdot i < d \cdot i$ **for** i
proof –
have $\text{box } c \ d \neq \{\}$
using *that*
unfolding *box_eq_empty* **by** *force*
{ **let** $?x = (\sum j \in \text{Basis}. (\text{if } j=i \text{ then } ((\min (a \cdot j) (d \cdot j)) + c \cdot j) / 2 \text{ else } (c \cdot j + d \cdot j) / 2))$
 $*_R \ j :: 'a$
assume $*$: $a \cdot i > c \cdot i$
then **have** $c \cdot j < ?x \cdot j \wedge ?x \cdot j < d \cdot j$ **if** $j \in \text{Basis}$ **for** j
using *cd that by (fastforce simp add: i *)*
then **have** $?x \in \text{box } c \ d$
unfolding *mem_box* **by** *auto*
moreover **have** $?x \notin \text{cbox } a \ b$
using *i cd * by (force simp: mem_box)*
ultimately **have** *False* **using** *box by auto*
}
then **have** $a \cdot i \leq c \cdot i$ **by** *force*
moreover
{ **let** $?x = (\sum j \in \text{Basis}. (\text{if } j=i \text{ then } ((\max (b \cdot j) (c \cdot j)) + d \cdot j) / 2 \text{ else } (c \cdot j + d \cdot j) / 2))$
 $*_R \ j :: 'a$
assume $*$: $b \cdot i < d \cdot i$
then **have** $d \cdot j > ?x \cdot j \wedge ?x \cdot j > c \cdot j$ **if** $j \in \text{Basis}$ **for** j
using *cd that by (fastforce simp add: i *)*
then **have** $?x \in \text{box } c \ d$
unfolding *mem_box* **by** *auto*
moreover **have** $?x \notin \text{cbox } a \ b$

```

    using  $i \cdot c \leq d \cdot i$  by (force simp: mem_box)
    ultimately have  $\text{False}$  using  $\text{box}$  by auto
  }
  then have  $b \cdot i \geq d \cdot i$  by (rule ccontr) auto
  ultimately show ?thesis by auto
qed
show ?th3
  using  $\text{acdb}$  by (fastforce simp add: mem_box)
  have  $\text{acdb}'$ :  $a \cdot i \leq c \cdot i \wedge d \cdot i \leq b \cdot i$ 
  if  $i \in \text{Basis}$   $\text{box } c \ d \subseteq \text{box } a \ b \wedge i. i \in \text{Basis} \implies c \cdot i < d \cdot i$  for  $i$ 
    using  $\text{box\_subset\_cbox}$ [of  $a \ b$ ] that  $\text{acdb}$  by auto
  show ?th4
    using  $\text{acdb}'$  by (fastforce simp add: mem_box)
qed

lemma eq_cbox:  $\text{cbox } a \ b = \text{cbox } c \ d \iff \text{cbox } a \ b = \{\} \wedge \text{cbox } c \ d = \{\} \vee a = c \wedge b = d$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have  $\text{cbox } a \ b \subseteq \text{cbox } c \ d \ \text{cbox } c \ d \subseteq \text{cbox } a \ b$ 
    by auto
  then show ?rhs
    by (force simp: subset_box box_eq_empty intro: antisym euclidean_eqI)
qed auto

lemma eq_box_box [simp]:  $\text{box } a \ b = \text{box } c \ d \iff \text{cbox } a \ b = \{\} \wedge \text{box } c \ d = \{\}$ 
  (is ?lhs  $\iff$  ?rhs)
proof
  assume  $L$ : ?lhs
  then have  $\text{cbox } a \ b \subseteq \text{box } c \ d \ \text{box } c \ d \subseteq \text{cbox } a \ b$ 
    by auto
  with  $L$  subset_box show ?rhs
    by (smt (verit) SOME_Basis box_ne_empty(1))
qed force

lemma eq_box_cbox [simp]:  $\text{box } a \ b = \text{cbox } c \ d \iff \text{box } a \ b = \{\} \wedge \text{cbox } c \ d = \{\}$ 
  by (metis eq_box_box)

lemma eq_box:  $\text{box } a \ b = \text{box } c \ d \iff \text{box } a \ b = \{\} \wedge \text{box } c \ d = \{\} \vee a = c \wedge b = d$ 
  (is ?lhs  $\iff$  ?rhs)
proof
  assume  $L$ : ?lhs
  then have  $\text{box } a \ b \subseteq \text{box } c \ d \ \text{box } c \ d \subseteq \text{box } a \ b$ 
    by auto
  then show ?rhs
    by auto
qed

```

unfolding subset_box by (smt (verit) box_ne_empty(2) euclidean_eq_iff)+
qed force

lemma subset_box_complex:

$cbox\ a\ b \subseteq cbox\ c\ d \iff$
 $(Re\ a \leq Re\ b \wedge Im\ a \leq Im\ b) \longrightarrow Re\ a \geq Re\ c \wedge Im\ a \geq Im\ c \wedge Re\ b \leq Re\ d \wedge Im\ b \leq Im\ d$
 $cbox\ a\ b \subseteq box\ c\ d \iff$
 $(Re\ a \leq Re\ b \wedge Im\ a \leq Im\ b) \longrightarrow Re\ a > Re\ c \wedge Im\ a > Im\ c \wedge Re\ b < Re\ d \wedge Im\ b < Im\ d$
 $box\ a\ b \subseteq cbox\ c\ d \iff$
 $(Re\ a < Re\ b \wedge Im\ a < Im\ b) \longrightarrow Re\ a \geq Re\ c \wedge Im\ a \geq Im\ c \wedge Re\ b \leq Re\ d \wedge Im\ b \leq Im\ d$
 $box\ a\ b \subseteq box\ c\ d \iff$
 $(Re\ a < Re\ b \wedge Im\ a < Im\ b) \longrightarrow Re\ a \geq Re\ c \wedge Im\ a \geq Im\ c \wedge Re\ b \leq Re\ d \wedge Im\ b \leq Im\ d$
by (subst subset_box; force simp: Basis_complex_def)+

lemma in_cbox_complex_iff:

$x \in cbox\ a\ b \iff Re\ x \in \{Re\ a..Re\ b\} \wedge Im\ x \in \{Im\ a..Im\ b\}$
by (cases x; cases a; cases b) (auto simp: cbox_Complex_eq)

lemma cbox_complex_of_real: $cbox\ (complex_of_real\ x)\ (complex_of_real\ y) = complex_of_real\ \{x..y\}$

proof –

have $(x \leq Re\ z \wedge Re\ z \leq y \wedge Im\ z = 0) = (z \in complex_of_real\ \{x..y\})$ **for** z
by (cases z) (simp add: complex_eq_cancel_iff2 image_iff)
then show ?thesis
by (auto simp: in_cbox_complex_iff)

qed

lemma box_Complex_eq:

$box\ (Complex\ a\ c)\ (Complex\ b\ d) = (\lambda(x,y). Complex\ x\ y)\ \{box\ a\ b \times box\ c\ d\}$
by (auto simp: box_def Basis_complex_def image_iff complex_eq_iff)

lemma in_box_complex_iff:

$x \in box\ a\ b \iff Re\ x \in \{Re\ a<..
by (cases x; cases a; cases b) (auto simp: box_Complex_eq)$

lemma box_complex_of_real [simp]: $box\ (complex_of_real\ x)\ (complex_of_real\ y) = \{ \}$

by (auto simp: in_box_complex_iff)

lemma Int_interval:

fixes $a :: 'a::euclidean_space$

shows $cbox\ a\ b \cap cbox\ c\ d =$

$cbox\ (\sum_{i \in Basis}. \max\ (a \cdot i)\ (c \cdot i) *_R i)\ (\sum_{i \in Basis}. \min\ (b \cdot i)\ (d \cdot i) *_R i)$

unfolding set_eq_iff and Int_iff and mem_box

by auto

lemma disjoint_interval:

fixes $a::'a::euclidean_space$
shows $cbox\ a\ b \cap cbox\ c\ d = \{\}$ $\longleftrightarrow (\exists i \in Basis. (b \cdot i < a \cdot i \vee d \cdot i < c \cdot i \vee b \cdot i < c \cdot i \vee d \cdot i < a \cdot i))$ **(is ?th1)**
and $cbox\ a\ b \cap box\ c\ d = \{\}$ $\longleftrightarrow (\exists i \in Basis. (b \cdot i < a \cdot i \vee d \cdot i \leq c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ **(is ?th2)**
and $box\ a\ b \cap cbox\ c\ d = \{\}$ $\longleftrightarrow (\exists i \in Basis. (b \cdot i \leq a \cdot i \vee d \cdot i < c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ **(is ?th3)**
and $box\ a\ b \cap box\ c\ d = \{\}$ $\longleftrightarrow (\exists i \in Basis. (b \cdot i \leq a \cdot i \vee d \cdot i \leq c \cdot i \vee b \cdot i \leq c \cdot i \vee d \cdot i \leq a \cdot i))$ **(is ?th4)**
proof –
let $?z = (\sum i \in Basis. (((max\ (a \cdot i)\ (c \cdot i)) + (min\ (b \cdot i)\ (d \cdot i))) / 2) *_{\mathbb{R}} i)::'a$
have $**:$ $\bigwedge P\ Q. (\bigwedge i :: 'a. i \in Basis \implies Q\ ?z\ i \implies P\ i) \implies$
 $(\bigwedge i\ x :: 'a. i \in Basis \implies P\ i \implies Q\ x\ i) \implies (\forall x. \exists i \in Basis. Q\ x\ i) \longleftrightarrow$
 $(\exists i \in Basis. P\ i)$
by *blast*
note $*$ = *set_eq_iff Int_iff empty_iff mem_box ball_conj_distrib[symmetric] eq_False ball_simps(10)*
show *?th1 unfolding * by (intro **) auto*
show *?th2 unfolding * by (intro **) auto*
show *?th3 unfolding * by (intro **) auto*
show *?th4 unfolding * by (intro **) auto*
qed

lemma UN_box_eq_UNIV: $(\bigcup i::nat. box\ (-\ (real\ i\ *_{\mathbb{R}}\ One))\ (real\ i\ *_{\mathbb{R}}\ One)) = UNIV$

proof –
have $|x \cdot b| < real_of_int\ (\lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil + 1)$
if *[simp]:* $b \in Basis$ **for** $x\ b :: 'a$
proof –
have $|x \cdot b| \leq real_of_int\ \lceil |x \cdot b| \rceil$
by *(rule le_of_int_ceiling)*
also **have** $\dots \leq real_of_int\ \lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil$
by *(auto intro!: ceiling_mono)*
also **have** $\dots < real_of_int\ (\lceil Max\ ((\lambda b. |x \cdot b|)\ 'Basis) \rceil + 1)$
by *simp*
finally **show** *?thesis .*
qed
then **have** $\exists n::nat. \forall b \in Basis. |x \cdot b| < real\ n$ **for** $x :: 'a$
by *(metis order.strict_trans reals_Archimedean2)*
moreover **have** $\bigwedge x\ b::'a. \bigwedge n::nat. |x \cdot b| < real\ n \longleftrightarrow -\ real\ n < x \cdot b \wedge x \cdot b < real\ n$
by *auto*
ultimately **show** *?thesis*
by *(auto simp: box_def inner_sum_left inner_Basis sum.If_cases)*
qed

lemma image_affinity_cbox: **fixes** $m::real$

```

fixes  $a\ b\ c :: 'a::euclidean\_space$ 
shows  $(\lambda x. m *_R x + c) \text{ ' } cbox\ a\ b =$ 
   $(if\ cbox\ a\ b = \{\} \text{ then } \{\}$ 
     $else\ (if\ 0 \leq m \text{ then } cbox\ (m *_R a + c)\ (m *_R b + c)$ 
     $else\ cbox\ (m *_R b + c)\ (m *_R a + c)))$ 
proof  $(cases\ m = 0)$ 
case True
{
  fix  $x$ 
  assume  $\forall i \in Basis. x \cdot i \leq c \cdot i \ \forall i \in Basis. c \cdot i \leq x \cdot i$ 
  then have  $x = c$ 
    by  $(simp\ add: dual\_order.antisym\ euclidean\_eqI)$ 
}
moreover have  $c \in cbox\ (m *_R a + c)\ (m *_R b + c)$ 
  unfolding True by auto
ultimately show ?thesis using True by  $(auto\ simp: cbox\_def)$ 
next
case False
{
  fix  $y$ 
  assume  $\forall i \in Basis. a \cdot i \leq y \cdot i \ \forall i \in Basis. y \cdot i \leq b \cdot i\ m > 0$ 
  then have  $\forall i \in Basis. (m *_R a + c) \cdot i \leq (m *_R y + c) \cdot i$ 
    and  $\forall i \in Basis. (m *_R y + c) \cdot i \leq (m *_R b + c) \cdot i$ 
    by  $(auto\ simp: inner\_distrib)$ 
}
moreover
{
  fix  $y$ 
  assume  $\forall i \in Basis. a \cdot i \leq y \cdot i \ \forall i \in Basis. y \cdot i \leq b \cdot i\ m < 0$ 
  then have  $\forall i \in Basis. (m *_R b + c) \cdot i \leq (m *_R y + c) \cdot i$ 
    and  $\forall i \in Basis. (m *_R y + c) \cdot i \leq (m *_R a + c) \cdot i$ 
    by  $(auto\ simp: mult\_left\_mono\_neg\ inner\_distrib)$ 
}
moreover
{
  fix  $y$ 
  assume  $m > 0$  and  $\forall i \in Basis. (m *_R a + c) \cdot i \leq y \cdot i$ 
    and  $\forall i \in Basis. y \cdot i \leq (m *_R b + c) \cdot i$ 
  then have  $y \in (\lambda x. m *_R x + c) \text{ ' } cbox\ a\ b$ 
    unfolding image_iff Bex_def mem_box
    apply  $(intro\ exI[\text{where } x=(1 / m) *_R (y - c)])$ 
    apply  $(auto\ simp: pos\_le\_divide\_eq\ pos\_divide\_le\_eq\ mult.commute\ inner\_distrib\ inner\_diff\_left)$ 
  done
}
moreover
{
  fix  $y$ 
  assume  $\forall i \in Basis. (m *_R b + c) \cdot i \leq y \cdot i \ \forall i \in Basis. y \cdot i \leq (m *_R a + c) \cdot$ 

```

```

i m < 0
  then have  $y \in (\lambda x. m *_{\mathbb{R}} x + c)$  ' cbox a b
    unfolding image_iff Bex_def mem_box
    apply (intro exI[where  $x=(1 / m) *_{\mathbb{R}} (y - c)$ ])
    apply (auto simp: neg_le_divide_eq neg_divide_le_eq mult.commute inner_distrib inner_diff_left)
    done
  }
  ultimately show ?thesis using False by (auto simp: cbox_def)
qed

```

```

lemma image_smult_cbox:  $(\lambda x. m *_{\mathbb{R}} (x :: \text{euclidean\_space}))$  ' cbox a b =
  (if cbox a b = {} then {} else if  $0 \leq m$  then cbox  $(m *_{\mathbb{R}} a)$   $(m *_{\mathbb{R}} b)$  else cbox
 $(m *_{\mathbb{R}} b)$   $(m *_{\mathbb{R}} a)$ )
  using image_affinity_cbox[of m 0 a b] by auto

```

```

lemma swap_continuous:
  assumes continuous_on (cbox (a,c) (b,d))  $(\lambda(x,y). f x y)$ 
  shows continuous_on (cbox (c,a) (d,b))  $(\lambda(x,y). f y x)$ 
proof -
  have  $(\lambda(x,y). f y x) = (\lambda(x,y). f x y) \circ \text{prod.swap}$ 
  by auto
  then show ?thesis
  by (metis assms continuous_on_compose continuous_on_swap swap_cbox_Pair)
qed

```

```

lemma open_contains_cbox:
  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  assumes open A  $x \in A$ 
  obtains a b where cbox a b  $\subseteq A$   $x \in \text{box a b}$   $\forall i \in \text{Basis}. a \cdot i < b \cdot i$ 
proof -
  from assms obtain R where  $R: R > 0$  ball x R  $\subseteq A$ 
  by (auto simp: open_contains_ball)
  define r :: real where  $r = R / (2 * \text{sqrt DIM}('a))$ 
  from  $\langle R > 0 \rangle$  have [simp]:  $r > 0$  by (auto simp: r_def)
  define d :: 'a where  $d = r *_{\mathbb{R}} \text{Topology\_Euclidean\_Space.One}$ 
  have cbox  $(x - d)$   $(x + d) \subseteq A$ 
  proof safe
    fix y assume  $y \in \text{cbox } (x - d)$   $(x + d)$ 
    have  $\text{dist } x y = \text{sqrt } (\sum i \in \text{Basis}. (\text{dist } (x \cdot i) (y \cdot i))^2)$ 
    by (subst euclidean_dist_l2) (auto simp: L2_set_def)
    also from y have  $\text{sqrt } (\sum i \in \text{Basis}. (\text{dist } (x \cdot i) (y \cdot i))^2) \leq \text{sqrt } (\sum i \in (\text{Basis}::'a \text{ set}). r^2)$ 
    by (intro real_sqrt_le_mono sum_mono power_mono)
    (auto simp: dist_norm d_def cbox_def algebra_simps)
    also have ... =  $\text{sqrt } (\text{DIM}('a) * r^2)$  by simp
    also have  $\text{DIM}('a) * r^2 = (R / 2) ^ 2$ 
    by (simp add: r_def power_divide)
    also have  $\text{sqrt } \dots = R / 2$ 

```

```

    using ⟨R > 0⟩ by simp
    also from ⟨R > 0⟩ have ... < R by simp
    finally have  $y \in \text{ball } x \ R$  by simp
    with R show  $y \in A$  by blast
  qed
  thus ?thesis
    using that[of  $x - d \ x + d$ ] by (auto simp: algebra_simps d_def box_def)
  qed

```

lemma *open_contains_box*:

```

  fixes  $x :: 'a :: \text{euclidean\_space}$ 
  assumes open A  $x \in A$ 
  obtains  $a \ b$  where  $\text{box } a \ b \subseteq A$   $x \in \text{box } a \ b$   $\forall i \in \text{Basis}. a \cdot i < b \cdot i$ 
  by (meson assms box_subset_cbox dual_order.trans open_contains_cbox)

```

lemma *inner_image_box*:

```

  assumes ( $i :: 'a :: \text{euclidean\_space}$ )  $\in \text{Basis}$ 
  assumes  $\forall i \in \text{Basis}. a \cdot i < b \cdot i$ 
  shows  $(\lambda x. x \cdot i) \text{ ` } \text{box } a \ b = \{a \cdot i .. b \cdot i\}$ 
  proof safe
    fix  $x$  assume  $x \in \{a \cdot i .. b \cdot i\}$ 
    let  $?y = (\sum j \in \text{Basis}. (\text{if } i = j \text{ then } x \text{ else } (a + b) \cdot j / 2)) *_{\mathbb{R}} j$ 
    from  $x$  assms have  $?y \cdot i \in (\lambda x. x \cdot i) \text{ ` } \text{box } a \ b$ 
      by (intro imageI) (auto simp: box_def algebra_simps)
    also have  $?y \cdot i = (\sum j \in \text{Basis}. (\text{if } i = j \text{ then } x \text{ else } (a + b) \cdot j / 2)) * (j \cdot i)$ 
      by (simp add: inner_sum_left)
    also have ... =  $(\sum j \in \text{Basis}. \text{if } i = j \text{ then } x \text{ else } 0)$ 
      by (intro sum.cong) (auto simp: inner_not_same_Basis assms)
    also have ... =  $x$  using assms by simp
    finally show  $x \in (\lambda x. x \cdot i) \text{ ` } \text{box } a \ b$  .
  qed (insert assms, auto simp: box_def)

```

lemma *inner_image_cbox*:

```

  assumes ( $i :: 'a :: \text{euclidean\_space}$ )  $\in \text{Basis}$ 
  assumes  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$ 
  shows  $(\lambda x. x \cdot i) \text{ ` } \text{cbox } a \ b = \{a \cdot i .. b \cdot i\}$ 
  proof safe
    fix  $x$  assume  $x \in \{a \cdot i .. b \cdot i\}$ 
    let  $?y = (\sum j \in \text{Basis}. (\text{if } i = j \text{ then } x \text{ else } a \cdot j)) *_{\mathbb{R}} j$ 
    from  $x$  assms have  $?y \cdot i \in (\lambda x. x \cdot i) \text{ ` } \text{cbox } a \ b$ 
      by (intro imageI) (auto simp: cbox_def)
    also have  $?y \cdot i = (\sum j \in \text{Basis}. (\text{if } i = j \text{ then } x \text{ else } a \cdot j)) * (j \cdot i)$ 
      by (simp add: inner_sum_left)
    also have ... =  $(\sum j \in \text{Basis}. \text{if } i = j \text{ then } x \text{ else } 0)$ 
      by (intro sum.cong) (auto simp: inner_not_same_Basis assms)
    also have ... =  $x$  using assms by simp
    finally show  $x \in (\lambda x. x \cdot i) \text{ ` } \text{cbox } a \ b$  .
  qed (insert assms, auto simp: cbox_def)

```

5.1.4 General Intervals

definition *is_interval* ($s::('a::euclidean_space) set$) \longleftrightarrow
 $(\forall a \in s. \forall b \in s. \forall x. (\forall i \in \text{Basis}. ((a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i) \vee (b \cdot i \leq x \cdot i \wedge x \cdot i \leq a \cdot i)))$
 $\longrightarrow x \in s)$

lemma *is_interval_1*:
 $is_interval (s::real\ set) \longleftrightarrow (\forall a \in s. \forall b \in s. \forall x. a \leq x \wedge x \leq b \longrightarrow x \in s)$
unfolding *is_interval_def* **by** *auto*

lemma *is_interval_Int*: $is_interval\ X \Longrightarrow is_interval\ Y \Longrightarrow is_interval\ (X \cap Y)$
unfolding *is_interval_def*
by *blast*

lemma *is_interval_cbox* [*simp*]: $is_interval\ (cbox\ a\ (b::'a::euclidean_space))$ (**is** *?th1*)
and *is_interval_box* [*simp*]: $is_interval\ (box\ a\ b)$ (**is** *?th2*)
unfolding *is_interval_def mem_box Ball_def atLeastAtMost_iff*
by (*meson order_trans le_less_trans less_le_trans less_trans*) $+$

lemma *is_interval_empty* [*iff*]: $is_interval\ \{\}$
unfolding *is_interval_def* **by** *simp*

lemma *is_interval_univ* [*iff*]: $is_interval\ UNIV$
unfolding *is_interval_def* **by** *simp*

lemma *mem_is_intervalI*:
assumes $is_interval\ S$
and $a \in S\ b \in S$
and $\bigwedge i. i \in \text{Basis} \Longrightarrow a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i \vee b \cdot i \leq x \cdot i \wedge x \cdot i \leq a \cdot i$
shows $x \in S$
using *assms is_interval_def* **by** *force*

lemma *interval_subst*:
fixes $S::'a::euclidean_space\ set$
assumes $is_interval\ S$
and $x \in S\ y\ j \in S$
and $j \in \text{Basis}$
shows $(\sum i \in \text{Basis}. (if\ i = j\ then\ y\ i \cdot i\ else\ x \cdot i) *_{\mathbb{R}} i) \in S$
by (*rule mem_is_intervalI[OF assms(1,2)]*) (*auto simp: assms*)

lemma *mem_box_componentwiseI*:
fixes $S::'a::euclidean_space\ set$
assumes $is_interval\ S$
assumes $\bigwedge i. i \in \text{Basis} \Longrightarrow x \cdot i \in ((\lambda x. x \cdot i) ' S)$
shows $x \in S$
proof $-$
from *assms* **have** $\forall i \in \text{Basis}. \exists s \in S. x \cdot i = s \cdot i$

```

    by auto
  with finite_Basis obtain s and bs::'a list
    where s:  $\bigwedge i. i \in \text{Basis} \implies x \cdot i = s \ i \cdot i$   $\bigwedge i. i \in \text{Basis} \implies s \ i \in S$ 
      and bs: set bs = Basis distinct bs
    by (metis finite_distinct_list)
  from nonempty_Basis s obtain j where j:  $j \in \text{Basis} \ s \ j \in S$ 
    by blast
  define y where
    y = rec_list (s j) ( $\lambda j \_ Y. (\sum i \in \text{Basis}. (\text{if } i = j \text{ then } s \ i \cdot i \text{ else } Y \cdot i) *_{\mathbb{R}} i)$ )
  have x =  $(\sum i \in \text{Basis}. (\text{if } i \in \text{set } bs \text{ then } s \ i \cdot i \text{ else } s \ j \cdot i) *_{\mathbb{R}} i)$ 
    using bs by (auto simp: s(1)[symmetric] euclidean_representation)
  also have [symmetric]:  $y \ bs = \dots$ 
    using bs(2) bs(1)[THEN equalityD1]
  by (induct bs) (auto simp: y_def euclidean_representation intro!: euclidean_eqI[where
'a='a])
  also have  $y \ bs \in S$ 
    using bs(1)[THEN equalityD1]
  proof (induction bs)
    case Nil
      then show ?case
        by (simp add: j y_def)
    next
      case (Cons a bs)
        then show ?case
          using interval_subst[OF assms(1)] s by (simp add: y_def)
  qed
  finally show ?thesis .
qed

```

```

lemma cbox01_nonempty [simp]:  $\text{cbox } 0 \ \text{One} \neq \{\}$ 
  by (simp add: box_ne_empty inner_Basis inner_sum_left sum_nonneg)

```

```

lemma box01_nonempty [simp]:  $\text{box } 0 \ \text{One} \neq \{\}$ 
  by (simp add: box_ne_empty inner_Basis inner_sum_left)

```

```

lemma empty_as_interval:  $\{\} = \text{cbox } \text{One} \ (0::'a::\text{euclidean\_space})$ 
  using nonempty_Basis box01_nonempty box_eq_empty(1) box_ne_empty(1)
  by blast

```

```

lemma interval_subset_is_interval:
  assumes is_interval S
  shows  $\text{cbox } a \ b \subseteq S \iff \text{cbox } a \ b = \{\} \vee a \in S \wedge b \in S$  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs using box_ne_empty(1) mem_box(2) by fastforce
next
  assume ?rhs
  have  $\text{cbox } a \ b \subseteq S$  if  $a \in S \ b \in S$ 
    using assms that

```

```

  by (force simp: mem_box intro: mem_is_intervalI)
  with ‹?rhs› show ?lhs
  by blast
qed

```

```

lemma is_real_interval_union:
  is_interval (X ∪ Y)
  if X: is_interval X and Y: is_interval Y and I: (X ≠ {} ⇒ Y ≠ {} ⇒ X
  ∩ Y ≠ {})
  for X Y::real set
proof -
  consider X ≠ {} Y ≠ {} | X = {} | Y = {} by blast
  then show ?thesis
  proof cases
    case 1
    then obtain r where r ∈ X ∨ X ∩ Y = {} r ∈ Y ∨ X ∩ Y = {}
    by blast
    then show ?thesis
    using I 1 X Y unfolding is_interval_1
    by (metis (full_types) Un_iff le_cases)
  qed (use that in auto)
qed

```

```

lemma is_interval_translationI:
  assumes is_interval X
  shows is_interval ((+) x ` X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume b ∈ X d ∈ X
  ∀ i ∈ Basis. (x + b) · i ≤ e · i ∧ e · i ≤ (x + d) · i ∨
  (x + d) · i ≤ e · i ∧ e · i ≤ (x + b) · i
  hence e - x ∈ X
  by (intro mem_is_intervalI[OF assms ‹b ∈ X› ‹d ∈ X›, of e - x])
  (auto simp: algebra_simps)
  thus e ∈ (+) x ` X by force
qed

```

```

lemma is_interval_uinverseI:
  assumes is_interval X
  shows is_interval (uminus ` X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume b ∈ X d ∈ X
  ∀ i ∈ Basis. (- b) · i ≤ e · i ∧ e · i ≤ (- d) · i ∨
  (- d) · i ≤ e · i ∧ e · i ≤ (- b) · i
  hence - e ∈ X
  by (smt (verit, ccfv_threshold) assms inner_minus_left mem_is_intervalI)

```

thus $e \in \text{uminus } ' X$ **by force**
qed

lemma *is_interval_uminus*[simp]: $\text{is_interval } (\text{uminus } ' x) = \text{is_interval } x$
using *is_interval_uminusI*[of x] *is_interval_uminusI*[of $\text{uminus } ' x$]
by (*auto simp: image_image*)

lemma *is_interval_neg_translationI*:
assumes *is_interval* X
shows $\text{is_interval } ((-) x ' X)$
proof –
have $(-) x ' X = (+) x ' \text{uminus } ' X$
by (*force simp: algebra_simps*)
also have $\text{is_interval } \dots$
by (*metis is_interval_uminusI is_interval_translationI assms*)
finally show ?thesis .
qed

lemma *is_interval_translation*[simp]:
 $\text{is_interval } ((+) x ' X) = \text{is_interval } X$
using *is_interval_neg_translationI*[of $(+) x ' X x$]
by (*auto intro!: is_interval_translationI simp: image_image*)

lemma *is_interval_minus_translation*[simp]:
shows $\text{is_interval } ((-) x ' X) = \text{is_interval } X$
proof –
have $(-) x ' X = (+) x ' \text{uminus } ' X$
by (*force simp: algebra_simps*)
also have $\text{is_interval } \dots = \text{is_interval } X$
by *simp*
finally show ?thesis .
qed

lemma *is_interval_minus_translation'*[simp]:
shows $\text{is_interval } ((\lambda x. x - c) ' X) = \text{is_interval } X$
using *is_interval_translation*[of $-c X$]
by (*metis image_cong uminus_add_conv_diff*)

lemma *is_interval_cball_1*[intro, simp]: $\text{is_interval } (\text{cball } a b)$ **for** $a b :: \text{real}$
by (*simp add: cball_eq_atLeastAtMost is_interval_def*)

lemma *is_interval_ball_real*: $\text{is_interval } (\text{ball } a b)$ **for** $a b :: \text{real}$
by (*simp add: ball_eq_greaterThanLessThan is_interval_def*)

5.1.5 Bounded Projections

lemma *bounded_inner_imp_bdd_above*:
assumes *bounded* s
shows *bdd_above* $((\lambda x. x \cdot a) ' s)$

by (simp add: assms bounded_imp_bdd_above bounded_linear_image bounded_linear_inner_left)

lemma bounded_inner_imp_bdd_below:

assumes bounded s

shows bdd_below (($\lambda x. x \cdot a$) ' s)

by (simp add: assms bounded_imp_bdd_below bounded_linear_image bounded_linear_inner_left)

5.1.6 Structural rules for pointwise continuity

lemma continuous_infnorm[continuous_intros]:

continuous F f \implies continuous F ($\lambda x. \text{infnorm } (f x)$)

unfolding continuous_def by (rule tendsto_infnorm)

lemma continuous_inner[continuous_intros]:

assumes continuous F f

and continuous F g

shows continuous F ($\lambda x. \text{inner } (f x) (g x)$)

using assms unfolding continuous_def by (rule tendsto_inner)

5.1.7 Structural rules for setwise continuity

lemma continuous_on_infnorm[continuous_intros]:

continuous_on s f \implies continuous_on s ($\lambda x. \text{infnorm } (f x)$)

unfolding continuous_on by (fast intro: tendsto_infnorm)

lemma continuous_on_inner[continuous_intros]:

fixes g :: 'a::topological_space \Rightarrow 'b::real_inner

assumes continuous_on s f

and continuous_on s g

shows continuous_on s ($\lambda x. \text{inner } (f x) (g x)$)

using bounded_bilinear_inner assms

by (rule bounded_bilinear.continuous_on)

5.1.8 Openness of halfspaces.

lemma open_halfspace_lt: open {x. inner a x < b}

by (simp add: open_Collect_less continuous_on_inner)

lemma open_halfspace_gt: open {x. inner a x > b}

by (simp add: open_Collect_less continuous_on_inner)

lemma open_halfspace_component_lt: open {x::'a::euclidean_space. x*i* < a}

by (simp add: open_Collect_less continuous_on_inner)

lemma open_halfspace_component_gt: open {x::'a::euclidean_space. x*i* > a}

by (simp add: open_Collect_less continuous_on_inner)

lemma eucl_less_eq_halfspaces:

fixes a :: 'a::euclidean_space

shows {x. x < e a} = ($\bigcap_{i \in \text{Basis}} \{x. x \cdot i < a \cdot i\}$)

$\{x. a < e x\} = (\bigcap i \in \text{Basis}. \{x. a \cdot i < x \cdot i\})$
by (*auto simp: eucl_less_def*)

lemma *open_Collect_eucl_less*[*simp, intro*]:
fixes $a :: 'a::\text{euclidean_space}$
shows *open* $\{x. x < e a\}$ *open* $\{x. a < e x\}$
by (*auto simp: eucl_less_eq_halfspaces open_halfspace_component_lt open_halfspace_component_gt*)

5.1.9 Closure and Interior of halfspaces and hyperplanes

lemma *continuous_at_inner*: *continuous* (*at* x) (*inner* a)
unfolding *continuous_at* **by** (*intro tendsto_intros*)

lemma *closed_halfspace_le*: *closed* $\{x. \text{inner } a \ x \leq b\}$
by (*simp add: closed_Collect_le continuous_on_inner*)

lemma *closed_halfspace_ge*: *closed* $\{x. \text{inner } a \ x \geq b\}$
by (*simp add: closed_Collect_le continuous_on_inner*)

lemma *closed_hyperplane*: *closed* $\{x. \text{inner } a \ x = b\}$
by (*simp add: closed_Collect_eq continuous_on_inner*)

lemma *closed_halfspace_component_le*: *closed* $\{x::'a::\text{euclidean_space}. x \cdot i \leq a\}$
by (*simp add: closed_Collect_le continuous_on_inner*)

lemma *closed_halfspace_component_ge*: *closed* $\{x::'a::\text{euclidean_space}. x \cdot i \geq a\}$
by (*simp add: closed_Collect_le continuous_on_inner*)

lemma *closed_interval_left*:
fixes $b :: 'a::\text{euclidean_space}$
shows *closed* $\{x::'a. \forall i \in \text{Basis}. x \cdot i \leq b \cdot i\}$
by (*simp add: Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner*)

lemma *closed_interval_right*:
fixes $a :: 'a::\text{euclidean_space}$
shows *closed* $\{x::'a. \forall i \in \text{Basis}. a \cdot i \leq x \cdot i\}$
by (*simp add: Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner*)

lemma *interior_halfspace_le* [*simp*]:
assumes $a \neq 0$
shows *interior* $\{x. a \cdot x \leq b\} = \{x. a \cdot x < b\}$
proof –
have $*$: $a \cdot x < b$ **if** $x: x \in S$ **and** $S: S \subseteq \{x. a \cdot x \leq b\}$ **and** *open* S **for** $S \ x$
proof –
obtain e **where** $e > 0$ **and** $e: \text{cball } x \ e \subseteq S$
using $\langle \text{open } S \rangle$ *open_contains_cball* x **by** *blast*
then have $x + (e / \text{norm } a) *_{\mathbb{R}} a \in \text{cball } x \ e$
by (*simp add: dist_norm*)
then have $x + (e / \text{norm } a) *_{\mathbb{R}} a \in S$

```

    using e by blast
  then have  $x + (e / \text{norm } a) *_{\mathbb{R}} a \in \{x. a \cdot x \leq b\}$ 
    using S by blast
  moreover have  $e * (a \cdot a) / \text{norm } a > 0$ 
    by (simp add: <0 < e> assms)
  ultimately show ?thesis
    by (simp add: algebra_simps)
qed
show ?thesis
  by (rule interior_unique) (auto simp: open_halfspace_lt *)
qed

```

```

lemma interior_halfspace_ge [simp]:
   $a \neq 0 \implies \text{interior } \{x. a \cdot x \geq b\} = \{x. a \cdot x > b\}$ 
using interior_halfspace_le [of -a -b] by simp

```

```

lemma closure_halfspace_lt [simp]:
  assumes  $a \neq 0$ 
  shows  $\text{closure } \{x. a \cdot x < b\} = \{x. a \cdot x \leq b\}$ 
proof -
  have [simp]:  $-\{x. a \cdot x < b\} = \{x. a \cdot x \geq b\}$ 
    by force
  then show ?thesis
    using interior_halfspace_ge [of a b] assms
    by (force simp: closure_interior)
qed

```

```

lemma closure_halfspace_gt [simp]:
   $a \neq 0 \implies \text{closure } \{x. a \cdot x > b\} = \{x. a \cdot x \geq b\}$ 
using closure_halfspace_lt [of -a -b] by simp

```

```

lemma interior_hyperplane [simp]:
  assumes  $a \neq 0$ 
  shows  $\text{interior } \{x. a \cdot x = b\} = \{\}$ 
proof -
  have [simp]:  $\{x. a \cdot x = b\} = \{x. a \cdot x \leq b\} \cap \{x. a \cdot x \geq b\}$ 
    by force
  then show ?thesis
    by (auto simp: assms)
qed

```

```

lemma frontier_halfspace_le:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows  $\text{frontier } \{x. a \cdot x \leq b\} = \{x. a \cdot x = b\}$ 
proof (cases a = 0)
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
    by (force simp: frontier_def closed_halfspace_le)

```

qed

```
lemma frontier_halfspace_ge:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows frontier  $\{x. a \cdot x \geq b\} = \{x. a \cdot x = b\}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
  by (force simp: frontier_def closed_halfspace_ge)
qed
```

```
lemma frontier_halfspace_lt:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows frontier  $\{x. a \cdot x < b\} = \{x. a \cdot x = b\}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
  by (force simp: frontier_def interior_open open_halfspace_lt)
qed
```

```
lemma frontier_halfspace_gt:
  assumes  $a \neq 0 \vee b \neq 0$ 
  shows frontier  $\{x. a \cdot x > b\} = \{x. a \cdot x = b\}$ 
proof (cases  $a = 0$ )
  case True with assms show ?thesis by simp
next
  case False then show ?thesis
  by (force simp: frontier_def interior_open open_halfspace_gt)
qed
```

5.1.10 Some more convenient intermediate-value theorem formulations

```
lemma connected_ivt_hyperplane:
  assumes connected  $S$  and  $xy: x \in S \ y \in S$  and  $b: \text{inner } a \ x \leq b \ b \leq \text{inner } a \ y$ 
  shows  $\exists z \in S. \text{inner } a \ z = b$ 
proof (rule ccontr)
  assume as:  $\neg (\exists z \in S. \text{inner } a \ z = b)$ 
  let ?A =  $\{x. \text{inner } a \ x < b\}$ 
  let ?B =  $\{x. \text{inner } a \ x > b\}$ 
  have open ?A open ?B
    using open_halfspace_lt and open_halfspace_gt by auto
  moreover have  $?A \cap ?B = \{\}$  by auto
  moreover have  $S \subseteq ?A \cup ?B$  using as by auto
  ultimately show False
    using  $\langle \text{connected } S \rangle$  unfolding connected_def
    by (smt (verit, del_insts) as b disjoint_iff empty_iff mem_Collect_eq xy)
```

qed

lemma *connected_ivt_component*:

fixes $x::'a::\text{euclidean_space}$
shows $\text{connected } S \implies x \in S \implies y \in S \implies x \cdot k \leq a \implies a \leq y \cdot k \implies (\exists z \in S. z \cdot k = a)$
using *connected_ivt_hyperplane*[of S x y $k::'a$ a]
by (*auto simp: inner_commute*)

5.1.11 Limit Component Bounds

lemma *Lim_component_le*:

fixes $f :: 'a \Rightarrow 'b::\text{euclidean_space}$
assumes $(f \longrightarrow l)$ *net*
and \neg (*trivial_limit net*)
and *eventually* $(\lambda x. f(x) \cdot i \leq b)$ *net*
shows $l \cdot i \leq b$
by (*rule tendsto_le*[OF *assms*(2) *tendsto_const tendsto_inner*[OF *assms*(1) *tendsto_const*] *assms*(3)])

lemma *Lim_component_ge*:

fixes $f :: 'a \Rightarrow 'b::\text{euclidean_space}$
assumes $(f \longrightarrow l)$ *net*
and \neg (*trivial_limit net*)
and *eventually* $(\lambda x. b \leq (f x) \cdot i)$ *net*
shows $b \leq l \cdot i$
by (*rule tendsto_le*[OF *assms*(2) *tendsto_inner*[OF *assms*(1) *tendsto_const*] *tendsto_const assms*(3)])

lemma *Lim_component_eq*:

fixes $f :: 'a \Rightarrow 'b::\text{euclidean_space}$
assumes *net*: $(f \longrightarrow l)$ *net* \neg *trivial_limit net*
and *ev*: *eventually* $(\lambda x. f(x) \cdot i = b)$ *net*
shows $l \cdot i = b$
using *ev*[*unfolded order_eq_iff eventually_conj_iff*]
using *Lim_component_ge*[OF *net*, of b i]
using *Lim_component_le*[OF *net*, of i b]
by *auto*

lemma *open_box*[*intro*]: *open* (*box* a b)

proof –

have *open* $(\bigcap_{i \in \text{Basis}. ((\cdot) i) - \{a \cdot i <..< b \cdot i\}})$
by (*auto intro!*: *continuous_open_vimage continuous_inner continuous_ident continuous_const*)
also have $(\bigcap_{i \in \text{Basis}. ((\cdot) i) - \{a \cdot i <..< b \cdot i\}}) = \text{box } a \ b$
by (*auto simp: box_def inner_commute*)
finally show *?thesis* .

qed

```

lemma closed_cbox[intro]:
  fixes a b :: 'a::euclidean_space
  shows closed (cbox a b)
proof -
  have closed ( $\bigcap_{i \in \text{Basis}} (\lambda x. x \cdot i) - \{a \cdot i .. b \cdot i\}$ )
    by (intro closed_INT ballI continuous_closed_vimage allI
        linear_continuous_at closed_real_atLeastAtMost finite_Basis bounded_linear_inner_left)
  also have ( $\bigcap_{i \in \text{Basis}} (\lambda x. x \cdot i) - \{a \cdot i .. b \cdot i\}$ ) = cbox a b
    by (auto simp: cbox_def)
  finally show closed (cbox a b) .
qed

```

```

lemma interior_cbox [simp]:
  fixes a b :: 'a::euclidean_space
  shows interior (cbox a b) = box a b (is ?L = ?R)
proof(rule subset_antisym)
  show ?R  $\subseteq$  ?L
    using box_subset_cbox open_box
    by (rule interior_maximal)
  {
    fix x
    assume x  $\in$  interior (cbox a b)
    then obtain s where s: open s x  $\in$  s s  $\subseteq$  cbox a b ..
    then obtain e where e>0 and e: $\forall x'. \text{dist } x' x < e \longrightarrow x' \in \text{cbox } a b$ 
      unfolding open_dist and subset_eq by auto
    {
      fix i :: 'a
      assume i: i  $\in$  Basis
      have dist (x - (e / 2) *R i) x < e
        and dist (x + (e / 2) *R i) x < e
          using norm_Basis[OF i] <e>0 by (auto simp: dist_norm)
      then have a  $\cdot$  i  $\leq$  (x - (e / 2) *R i)  $\cdot$  i and (x + (e / 2) *R i)  $\cdot$  i  $\leq$  b  $\cdot$  i
        using e[THEN spec[where x=x - (e/2) *R i]]
          and e[THEN spec[where x=x + (e/2) *R i]]
        unfolding mem_box using i by blast+
      then have a  $\cdot$  i < x  $\cdot$  i and x  $\cdot$  i < b  $\cdot$  i
        using <e>0 i
        by (auto simp: inner_diff_left inner_Basis inner_add_left)
    }
    then have x  $\in$  box a b
      unfolding mem_box by auto
  }
  then show ?L  $\subseteq$  ?R ..
qed

```

```

lemma bounded_cbox [simp]:
  fixes a :: 'a::euclidean_space
  shows bounded (cbox a b)
proof -

```

```

let ?b =  $\sum_{i \in \text{Basis}} |a \cdot i| + |b \cdot i|$ 
{
  fix x :: 'a
  assume  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i$ 
  then have  $(\sum_{i \in \text{Basis}} |x \cdot i|) \leq ?b$ 
    by (force simp: intro!: sum_mono)
  then have norm x  $\leq ?b$ 
    using norm_le_l1[of x] by auto
}
then show ?thesis
  unfolding cbox_def bounded_iff by force
qed

```

```

lemma bounded_box [simp]:
  fixes a :: 'a::euclidean_space
  shows bounded (box a b)
  by (metis bounded_cbox bounded_interior interior_cbox)

```

```

lemma not_interval_UNIV [simp]:
  fixes a :: 'a::euclidean_space
  shows cbox a b  $\neq$  UNIV box a b  $\neq$  UNIV
  using bounded_box[of a b] bounded_cbox[of a b] by force+

```

```

lemma not_interval_UNIV2 [simp]:
  fixes a :: 'a::euclidean_space
  shows UNIV  $\neq$  cbox a b UNIV  $\neq$  box a b
  using bounded_box[of a b] bounded_cbox[of a b] by force+

```

```

lemma box_midpoint:
  fixes a :: 'a::euclidean_space
  assumes box a b  $\neq$  {}
  shows  $((1/2) *_R (a + b)) \in \text{box } a \ b$ 
proof -
  have  $a \cdot i < ((1 / 2) *_R (a + b)) \cdot i \wedge ((1 / 2) *_R (a + b)) \cdot i < b \cdot i$  if  $i \in$ 
  Basis for  $i$ 
    using assms that by (auto simp: inner_add_left box_ne_empty)
  then show ?thesis unfolding mem_box by auto
qed

```

```

lemma open_cbox_convex:
  fixes x :: 'a::euclidean_space
  assumes x:  $x \in \text{box } a \ b$ 
  and y:  $y \in \text{cbox } a \ b$ 
  and e:  $0 < e \leq 1$ 
  shows  $(e *_R x + (1 - e) *_R y) \in \text{box } a \ b$ 
proof -
  {
    fix i :: 'a
    assume i:  $i \in \text{Basis}$ 

```

```

have a · i = e * (a · i) + (1 - e) * (a · i)
  unfolding left_diff_distrib by simp
also have ... < e * (x · i) + (1 - e) * (y · i)
  by (smt (verit, best) e i mem_box mult_le_cancel_left_pos mult_left_mono
x y)
finally have a · i < (e *R x + (1 - e) *R y) · i
  unfolding inner_simps by auto
moreover
{
  have b · i = e * (b · i) + (1 - e) * (b · i)
    unfolding left_diff_distrib by simp
  also have ... > e * (x · i) + (1 - e) * (y · i)
    by (smt (verit, best) e i mem_box mult_le_cancel_left_pos mult_left_mono
x y)
  finally have (e *R x + (1 - e) *R y) · i < b · i
    unfolding inner_simps by auto
}
ultimately have a · i < (e *R x + (1 - e) *R y) · i ∧ (e *R x + (1 - e) *R
y) · i < b · i
  by auto
}
then show ?thesis
  unfolding mem_box by auto
qed

```

```

lemma closure_cbox [simp]: closure (cbox a b) = cbox a b
  by (simp add: closed_cbox)

```

```

lemma closure_box [simp]:
  fixes a :: 'a::euclidean_space
  assumes box a b ≠ {}
  shows closure (box a b) = cbox a b
proof -
  have ab: a < e b
    using assms by (simp add: eucl_less_def box_ne_empty)
  let ?c = (1 / 2) *R (a + b)
  {
    fix x
    assume as: x ∈ cbox a b
    define f where [abs_def]: f n = x + (inverse (real n + 1)) *R (?c - x) for n
    {
      fix n
      assume fn: f n < e b ⟶ a < e f n ⟶ f n = x and xc: x ≠ ?c
      have *: 0 < inverse (real n + 1) inverse (real n + 1) ≤ 1
        unfolding inverse_le_1_iff by auto
      have (inverse (real n + 1)) *R ((1 / 2) *R (a + b)) + (1 - inverse (real n
+ 1)) *R x =
        x + (inverse (real n + 1)) *R (((1 / 2) *R (a + b)) - x)
        by (auto simp: algebra_simps)
    }
  }

```



```

then have  $f\ n < \varepsilon\ b$  and  $a < \varepsilon\ f\ n$ 
  using open_cbox_convex[OF box_midpoint[OF assms] as *]
  unfolding f_def by (auto simp: box_def eucl_less_def)
then have False
  using fn unfolding f_def using xc by auto
}
moreover
{
  have  $\exists N::nat. \forall n \geq N. \text{inverse}(\text{real } n + 1) < \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
    using reals_Archimedean [of  $\varepsilon$ ] that
    by (metis inverse_inverse_eq inverse_less_imp_less nat_le_real_less
order_less_trans
      reals_Archimedean2)
  then have  $(\lambda n. \text{inverse}(\text{real } n + 1)) \longrightarrow 0$ 
    unfolding lim_sequentially by(auto simp: dist_norm)
  then have  $f \longrightarrow x$ 
    unfolding f_def
    using tendsto_add[OF tendsto_const, of  $\lambda n. (\text{inverse}(\text{real } n + 1)) *_{\mathbb{R}} ((1 / 2) *_{\mathbb{R}} (a + b) - x)$  0 sequentially x]
    using tendsto_scaleR [OF tendsto_const, of  $\lambda n. \text{inverse}(\text{real } n + 1)$  0 sequentially  $((1 / 2) *_{\mathbb{R}} (a + b) - x)$ ]
    by auto
  }
  ultimately have  $x \in \text{closure}(\text{box } a\ b)$ 
    using as box_midpoint[OF assms]
    unfolding closure_def islimpt_sequential
    by (cases x=?c) (auto simp: in_box_eucl_less)
}
then show ?thesis
  using closure_minimal[OF box_subset_cbox, of a b] by blast
qed

```

lemma *bounded_subset_box_symmetric*:

fixes $S :: ('a::\text{euclidean_space})\ \text{set}$

assumes *bounded S*

obtains a **where** $S \subseteq \text{box}(-a)\ a$

proof –

obtain b **where** $b > 0$ **and** $b: \forall x \in S. \text{norm } x \leq b$

using *assms*[*unfolded* *bounded_pos*] **by** *auto*

define $a :: 'a$ **where** $a = (\sum i \in \text{Basis}. (b + 1) *_{\mathbb{R}} i)$

have $(-a) \cdot i < x \cdot i$ **and** $x \cdot i < a \cdot i$ **if** $x \in S$ **and** $i: i \in \text{Basis}$ **for** $x\ i$

using b *Basis_le_norm*[*OF* *i*, *of* *x*] **that** **by** (*auto simp: a_def*)

then have $S \subseteq \text{box}(-a)\ a$

by (*auto simp: simp add: box_def*)

then show *?thesis* ..

qed

lemma *bounded_subset_cbox_symmetric*:

fixes $S :: ('a::\text{euclidean_space})\ \text{set}$

assumes *bounded S*
obtains a where $S \subseteq \text{cbox } (-a) a$
by (*meson assms bounded_subset_box_symmetric_box_subset_cbox_order.trans*)

lemma *frontier_cbox*:
fixes $a b :: 'a::\text{euclidean_space}$
shows $\text{frontier } (\text{cbox } a b) = \text{cbox } a b - \text{box } a b$
unfolding *frontier_def* **unfolding** *interior_cbox* **and** *closure_closed[OF closed_cbox]*
..

lemma *frontier_box*:
fixes $a b :: 'a::\text{euclidean_space}$
shows $\text{frontier } (\text{box } a b) = (\text{if } \text{box } a b = \{\} \text{ then } \{\} \text{ else } \text{cbox } a b - \text{box } a b)$
by (*simp add: frontier_def interior_open open_box*)

lemma *Int_interval_mixed_eq_empty*:
fixes $a :: 'a::\text{euclidean_space}$
assumes $\text{box } c d \neq \{\}$
shows $\text{box } a b \cap \text{cbox } c d = \{\} \iff \text{box } a b \cap \text{box } c d = \{\}$
unfolding *closure_box[OF assms, symmetric]*
unfolding *open_Int_closure_eq_empty[OF open_box]* **..**

5.1.12 Class Instances

lemma *compact_lemma*:
fixes $f :: \text{nat} \Rightarrow 'a::\text{euclidean_space}$
assumes *bounded (range f)*
shows $\forall d \subseteq \text{Basis}. \exists l :: 'a. \exists r. \text{strict_mono } r \wedge (\forall e > 0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f (r n) \cdot i) (l \cdot i) < e) \text{ sequentially})$
by (*rule compact_lemma_general[where unproj= $\lambda e. \sum i \in \text{Basis}. e i *_R i$]*)
(auto intro!: assms bounded_linear_inner_left bounded_linear_image simp: euclidean_representation)

instance *euclidean_space* \subseteq *heine_borel*

proof

fix $f :: \text{nat} \Rightarrow 'a$
assume $f: \text{bounded } (\text{range } f)$
then obtain $l :: 'a$ **and** r **where** $r: \text{strict_mono } r$
and $l: \forall e > 0. \text{eventually } (\lambda n. \forall i \in \text{Basis}. \text{dist } (f (r n) \cdot i) (l \cdot i) < e) \text{ sequentially}$
using *compact_lemma [OF f]* **by** *blast*
{
fix $e :: \text{real}$
assume $e > 0$
hence $e / \text{real_of_nat } \text{DIM}('a) > 0$ **by** (*simp*)
with l **have** $\text{eventually } (\lambda n. \forall i \in \text{Basis}. \text{dist } (f (r n) \cdot i) (l \cdot i) < e / (\text{real_of_nat } \text{DIM}('a))) \text{ sequentially}$
by *simp*
moreover

```

{ fix n
  assume n:  $\forall i \in \text{Basis}. \text{dist} (f (r n) \cdot i) (l \cdot i) < e / (\text{real\_of\_nat } \text{DIM}('a))$ 
  have  $\text{dist} (f (r n)) l \leq (\sum i \in \text{Basis}. \text{dist} (f (r n) \cdot i) (l \cdot i))$ 
    using L2_set_le_sum [OF zero_le_dist] by (subst euclidean_dist_l2)
  also have  $\dots < (\sum i \in (\text{Basis}::'a \text{ set}). e / (\text{real\_of\_nat } \text{DIM}('a)))$ 
    by (meson eucl.finite_Basis n nonempty_Basis sum_strict_mono)
  finally have  $\text{dist} (f (r n)) l < e$ 
    by auto
}
ultimately have  $\forall_F n \text{ in sequentially. } \text{dist} (f (r n)) l < e$ 
  by (rule eventually_mono)
}
then have  $*(f \circ r) \longrightarrow l$ 
  unfolding o_def tendsto_iff by simp
with r show  $\exists l r. \text{strict\_mono } r \wedge (f \circ r) \longrightarrow l$ 
  by auto
qed

instance euclidean_space  $\subseteq$  banach ..

instance euclidean_space  $\subseteq$  second_countable_topology
proof
  define a where  $a f = (\sum i \in \text{Basis}. \text{fst} (f i) *_R i)$  for  $f :: 'a \Rightarrow \text{real} \times \text{real}$ 
  then have  $a: \bigwedge f. (\sum i \in \text{Basis}. \text{fst} (f i) *_R i) = a f$ 
    by simp
  define b where  $b f = (\sum i \in \text{Basis}. \text{snd} (f i) *_R i)$  for  $f :: 'a \Rightarrow \text{real} \times \text{real}$ 
  then have  $b: \bigwedge f. (\sum i \in \text{Basis}. \text{snd} (f i) *_R i) = b f$ 
    by simp
  define B where  $B = (\lambda f. \text{box} (a f) (b f)) \text{ ` } (\text{Basis} \rightarrow_E (\mathbb{Q} \times \mathbb{Q}))$ 

  have Ball B open by (simp add: B_def open_box)
  moreover have  $(\forall A. \text{open } A \longrightarrow (\exists B' \subseteq B. \bigcup B' = A))$ 
  proof safe
    fix  $A::'a \text{ set}$ 
    assume open A
    show  $\exists B' \subseteq B. \bigcup B' = A$ 
      using open_UNION_box [OF ‹open A›]
      by (smt (verit, ccfv_threshold) B_def a b image_iff mem_Collect_eq subsetI)
  qed
  ultimately
  have topological_basis B
    unfolding topological_basis_def by blast
  moreover
  have countable B
    unfolding B_def
    by (intro countable_image countable_PiE finite_Basis countable_SIGMA countable_rat)
  ultimately show  $\exists B::'a \text{ set set. countable } B \wedge \text{open} = \text{generate\_topology } B$ 
    by (blast intro: topological_basis_imp_subbasis)

```

qed

instance euclidean_space \subseteq polish_space ..

5.1.13 Compact Boxes

lemma compact_cbox [simp]:
 fixes a :: 'a::euclidean_space
 shows compact (cbox a b)
 using bounded_closed_imp_seq_compact[of cbox a b] using bounded_cbox[of a b]
 by (auto simp: compact_eq_seq_compact_metric)

proposition is_interval_compact:

is_interval S \wedge compact S \longleftrightarrow (\exists a b. S = cbox a b) (is ?lhs = ?rhs)

proof (cases S = {})

case True

with empty_as_interval show ?thesis by auto

next

case False

show ?thesis

proof

assume L: ?lhs

then have is_interval S compact S by auto

define a where $a \equiv \sum_{i \in \text{Basis}} (\text{INF } x \in S. x \cdot i) *_{\mathbb{R}} i$

define b where $b \equiv \sum_{i \in \text{Basis}} (\text{SUP } x \in S. x \cdot i) *_{\mathbb{R}} i$

have 1: $\bigwedge x i. [x \in S; i \in \text{Basis}] \implies (\text{INF } x \in S. x \cdot i) \leq x \cdot i$

by (simp add: cInf_lower bounded_inner_imp_bdd_below compact_imp_bounded

L)

have 2: $\bigwedge x i. [x \in S; i \in \text{Basis}] \implies x \cdot i \leq (\text{SUP } x \in S. x \cdot i)$

by (simp add: cSup_upper bounded_inner_imp_bdd_above compact_imp_bounded

L)

have 3: x \in S if inf: $\bigwedge i. i \in \text{Basis} \implies (\text{INF } x \in S. x \cdot i) \leq x \cdot i$

and sup: $\bigwedge i. i \in \text{Basis} \implies x \cdot i \leq (\text{SUP } x \in S. x \cdot i)$ for x

proof (rule mem_box_componentwiseI [OF ‹is_interval S›])

fix i::'a

assume i: i \in Basis

have cont: continuous_on S ($\lambda x. x \cdot i$)

by (intro continuous_intros)

obtain a where a \in S and a: $\bigwedge y. y \in S \implies a \cdot i \leq y \cdot i$

using continuous_attains_inf [OF ‹compact S› False cont] by blast

obtain b where b \in S and b: $\bigwedge y. y \in S \implies y \cdot i \leq b \cdot i$

using continuous_attains_sup [OF ‹compact S› False cont] by blast

have a \cdot i \leq (INF x \in S. x \cdot i)

by (simp add: False a cINF_greatest)

also have ... \leq x \cdot i

by (simp add: i inf)

finally have ai: a \cdot i \leq x \cdot i .

have x \cdot i \leq (SUP x \in S. x \cdot i)

```

    by (simp add: i sup)
  also have  $(\text{SUP } x \in S. x \cdot i) \leq b \cdot i$ 
    by (simp add: False b cSUP_least)
  finally have  $bi: x \cdot i \leq b \cdot i$  .
  show  $x \cdot i \in (\lambda x. x \cdot i) \text{ ` } S$ 
    apply (rule_tac  $x = \sum j \in \text{Basis}. (((\cdot)a)(i := x \cdot j))j *_R j$  in image_eqI)
    apply (simp add: i)
    apply (rule mem_is_intervalI [OF ‹is_interval S› ‹a ∈ S› ‹b ∈ S›])
    using i ai bi
    apply force
    done
qed
have  $S = \text{cbox } a \ b$ 
  by (auto simp: a_def b_def mem_box intro: 1 2 3)
then show ?rhs
  by blast
next
  assume R: ?rhs
  then show ?lhs
    using compact_cbox_is_interval_cbox by blast
qed
qed

```

5.1.14 Componentwise limits and continuity

But is the premise really necessary? Need to generalise $\text{dist } ?x \ ?y = L2_set$
 $(\lambda i. \text{dist } (?x \cdot i) \ (?y \cdot i)) \text{ Basis}$

lemma *Euclidean_dist_upper*: $i \in \text{Basis} \implies \text{dist } (x \cdot i) \ (y \cdot i) \leq \text{dist } x \ y$
 by (metis (no_types) member_le_L2_set euclidean_dist_l2 finite_Basis)

But is the premise $i \in \text{Basis}$ really necessary?

```

lemma open_preimage_inner:
  assumes open S i ∈ Basis
  shows open {x. x · i ∈ S}
proof (rule openI, simp)
  fix x
  assume x: x · i ∈ S
  with assms obtain e where 0 < e and e: ball (x · i) e ⊆ S
    by (auto simp: open_contains_ball_eq)
  have ∃ e > 0. ball (y · i) e ⊆ S if dxy: dist x y < e / 2 for y
  proof (intro exI conjI)
    have dist (x · i) (y · i) < e / 2
      by (meson ‹i ∈ Basis› dual_order.trans Euclidean_dist_upper not_le that)
    then have dist (x · i) z < e if dist (y · i) z < e / 2 for z
      by (metis dist_commute dist_triangle_half_1 that)
    then have ball (y · i) (e / 2) ⊆ ball (x · i) e
      using mem_ball by blast
    with e show ball (y · i) (e / 2) ⊆ S
  qed

```

```

      by (metis order_trans)
    qed (simp add: ‹0 < e›)
    then show  $\exists e > 0. \text{ball } x \ e \subseteq \{s. s \cdot i \in S\}$ 
      by (metis (no_types, lifting) ‹0 < e› ‹open S› half_gt_zero_iff mem_Collect_eq
mem_ball_open_contains_ball_eq subsetI)
    qed

```

proposition *tendsto_componentwise_iff*:

```

  fixes f ::  $\_ \Rightarrow 'b::\text{euclidean\_space}$ 
  shows (f  $\longrightarrow$  l) F  $\longleftrightarrow$  ( $\forall i \in \text{Basis}. ((\lambda x. (f x \cdot i)) \longrightarrow (l \cdot i)) F$ )
    (is ?lhs = ?rhs)

```

proof

```

  assume ?lhs
  then show ?rhs
    unfolding tendsto_def
    by (smt (verit) eventually_elim2 mem_Collect_eq open_preimage_inner)
next

```

```

  assume R: ?rhs
  then have  $\bigwedge e. e > 0 \implies \forall i \in \text{Basis}. \forall_F x \text{ in } F. \text{dist } (f x \cdot i) (l \cdot i) < e$ 
    unfolding tendsto_iff by blast
  then have R':  $\bigwedge e. e > 0 \implies \forall_F x \text{ in } F. \forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e$ 
    by (simp add: eventually_ball_finite_distrib [symmetric])

```

show ?lhs

unfolding tendsto_iff

proof clarify

fix e::real

assume 0 < e

have *: $L2_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} < e$

if $\forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e / \text{real DIM('b)}$ for x

proof -

have $L2_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} \leq \text{sum } (\lambda i. \text{dist } (f x \cdot i) (l \cdot i))$

Basis

by (simp add: L2_set_le_sum)

also have $\dots < \text{DIM('b)} * (e / \text{real DIM('b)})$

by (meson DIM_positive sum_bounded_above_strict that)

also have $\dots = e$

by (simp add: field_simps)

finally show $L2_set (\lambda i. \text{dist } (f x \cdot i) (l \cdot i)) \text{Basis} < e$.

qed

have $\forall_F x \text{ in } F. \forall i \in \text{Basis}. \text{dist } (f x \cdot i) (l \cdot i) < e / \text{DIM('b)}$

by (simp add: R' ‹0 < e›)

then show $\forall_F x \text{ in } F. \text{dist } (f x) l < e$

by eventually_elim (metis (full_types) * euclidean_dist_l2)

qed

qed

corollary *continuous_componentwise*:

```

  continuous F  $\longleftrightarrow$  ( $\forall i \in \text{Basis}. \text{continuous } F (\lambda x. (f x \cdot i))$ )

```

by (simp add: continuous_def tendsto_componentwise_iff [symmetric])

corollary continuous_on_componentwise:

fixes $S :: 'a :: t2_space$ set

shows $\text{continuous_on } S \ f \longleftrightarrow (\forall i \in \text{Basis}. \text{continuous_on } S \ (\lambda x. (f \ x \cdot i)))$

by (metis continuous_componentwise continuous_on_eq_continuous_within)

lemma linear_componentwise_iff:

$\text{linear } f' \longleftrightarrow (\forall i \in \text{Basis}. \text{linear } (\lambda x. f' \ x \cdot i))$ (is ?lhs \longleftrightarrow ?rhs)

proof

show ?lhs \implies ?rhs

by (simp add: Real_Vector_Spaces.linear_iff inner_left_distrib)

show ?rhs \implies ?lhs

by (simp add: linear_iff) (metis euclidean_eqI inner_left_distrib inner_scaleR_left)

qed

lemma bounded_linear_componentwise_iff:

$(\text{bounded_linear } f') \longleftrightarrow (\forall i \in \text{Basis}. \text{bounded_linear } (\lambda x. f' \ x \cdot i))$

(is ?lhs = ?rhs)

proof

assume ?rhs

then have $(\forall i \in \text{Basis}. \exists K. \forall x. |f' \ x \cdot i| \leq \text{norm } x * K)$ linear f'

by (auto simp: bounded_linear_def bounded_linear_axioms_def linear_componentwise_iff [symmetric] ball_conj_distrib)

then obtain F where $F: \bigwedge i \ x. i \in \text{Basis} \implies |f' \ x \cdot i| \leq \text{norm } x * F \ i$

by metis

have $\text{norm } (f' \ x) \leq \text{norm } x * \text{sum } F \ \text{Basis}$ for x

proof –

have $\text{norm } (f' \ x) \leq (\sum i \in \text{Basis}. |f' \ x \cdot i|)$

by (rule norm_le_l1)

also have $\dots \leq (\sum i \in \text{Basis}. \text{norm } x * F \ i)$

by (metis F sum_mono)

also have $\dots = \text{norm } x * \text{sum } F \ \text{Basis}$

by (simp add: sum_distrib_left)

finally show ?thesis .

qed

then show ?lhs

by (force simp: bounded_linear_def bounded_linear_axioms_def ⟨linear f' ⟩)

qed (simp add: bounded_linear_inner_left_comp)

5.1.15 Continuous Extension

definition clamp :: $'a :: \text{euclidean_space} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ where

$\text{clamp } a \ b \ x = (\text{if } (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$

then $(\sum i \in \text{Basis}. (\text{if } x \cdot i < a \cdot i \text{ then } a \cdot i \text{ else if } x \cdot i \leq b \cdot i \text{ then } x \cdot i \text{ else } b \cdot i) *_{\mathbb{R}} i)$

else a)

lemma clamp_in_interval[simp]:

assumes $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i$

shows $\text{clamp } a \ b \ x \in \text{cbox } a \ b$
unfolding clamp_def
using $\text{box_ne_empty}(1)[\text{of } a \ b]$ **assms by** ($\text{auto simp: cbox_def}$)

lemma $\text{clamp_cancel_cbox}[simp]$:
fixes $x \ a \ b :: 'a::\text{euclidean_space}$
assumes $x \in \text{cbox } a \ b$
shows $\text{clamp } a \ b \ x = x$
using assms
by ($\text{auto simp: clamp_def mem_box intro!: euclidean_eqI}[\text{where } 'a='a]$)

lemma $\text{clamp_empty_interval}$:
assumes $i \in \text{Basis } a \cdot i > b \cdot i$
shows $\text{clamp } a \ b = (\lambda _. a)$
using assms
by ($\text{force simp: clamp_def}[\text{abs_def}] \text{split: if_splits intro!: ext}$)

lemma $\text{dist_clamps_le_dist_args}$:
fixes $x :: 'a::\text{euclidean_space}$
shows $\text{dist } (\text{clamp } a \ b \ y) (\text{clamp } a \ b \ x) \leq \text{dist } y \ x$
proof cases
assume $le: (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$
then have $(\sum i \in \text{Basis}. (\text{dist } (\text{clamp } a \ b \ y \cdot i) (\text{clamp } a \ b \ x \cdot i))^2) \leq$
 $(\sum i \in \text{Basis}. (\text{dist } (y \cdot i) (x \cdot i))^2)$
by ($\text{auto intro!: sum_mono simp: clamp_def dist_real_def abs_le_square_iff}[\text{symmetric}]$)
then show $?thesis$
by ($\text{auto intro: real_sqrt_le_mono}$
 $\text{simp: euclidean_dist_l2}[\text{where } y=x] \text{euclidean_dist_l2}[\text{where } y=\text{clamp } a \ b$
 $x] \text{L2_set_def}$)
qed ($\text{auto simp: clamp_def}$)

lemma $\text{clamp_continuous_at}$:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{metric_space}$
and $x :: 'a$
assumes $f_cont: \text{continuous_on } (\text{cbox } a \ b) \ f$
shows $\text{continuous } (\text{at } x) (\lambda x. f (\text{clamp } a \ b \ x))$
proof cases
assume $le: (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$
show $?thesis$
unfolding $\text{continuous_at_eps_delta}$
proof safe
fix $x :: 'a$
fix $e :: \text{real}$
assume $e > 0$
moreover have $\text{clamp } a \ b \ x \in \text{cbox } a \ b$
by (simp add: le)
moreover note $f_cont[\text{simplified continuous_on_iff}]$
ultimately
obtain d **where** $d: 0 < d$


```

 $\bigwedge x'. x' \in \text{cbox } a \ b \implies \text{dist } x' \ (\text{clamp } a \ b \ x) < d \implies \text{dist } (f \ x') \ (f \ (\text{clamp } a \ b \ x)) < e$ 
  by force
  show  $\exists d > 0. \forall x'. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ (\text{clamp } a \ b \ x')) \ (f \ (\text{clamp } a \ b \ x)) < e$ 
  using le
  by (auto intro!: d clamp_in_interval dist_clamps_le_dist_args[THEN le_less_trans])
qed
qed (auto simp: clamp_empty_interval)

```

lemma *clamp_continuous_on*:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{metric\_space}$ 
assumes  $f\_cont: \text{continuous\_on } (\text{cbox } a \ b) \ f$ 
shows  $\text{continuous\_on } S \ (\lambda x. f \ (\text{clamp } a \ b \ x))$ 
using assms
by (auto intro: continuous_at_imp_continuous_on clamp_continuous_at)

```

lemma *clamp_bounded*:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{metric\_space}$ 
assumes  $\text{bounded: bounded } (f \ ' (\text{cbox } a \ b))$ 
shows  $\text{bounded } (\text{range } (\lambda x. f \ (\text{clamp } a \ b \ x)))$ 
proof cases
  assume  $le: (\forall i \in \text{Basis}. a \cdot i \leq b \cdot i)$ 
  from  $\text{bounded}$  obtain  $c$  where  $f\_bound: \forall x \in f \ ' \ \text{cbox } a \ b. \text{dist } \text{undefined } x \leq c$ 
  by (auto simp: bounded_any_center[where a=undefined])
  then show ?thesis
  by (metis bounded_bounded_subset clamp_in_interval image_mono image_subsetI
  le_range_composition)
qed (auto simp: clamp_empty_interval image_def)

```

definition *ext_cont* :: $('a::\text{euclidean_space} \Rightarrow 'b::\text{metric_space}) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b$

```

where  $\text{ext\_cont } f \ a \ b = (\lambda x. f \ (\text{clamp } a \ b \ x))$ 

```

lemma *ext_cont_cancel_cbox*[simp]:

```

fixes  $x \ a \ b :: 'a::\text{euclidean\_space}$ 
assumes  $x: x \in \text{cbox } a \ b$ 
shows  $\text{ext\_cont } f \ a \ b \ x = f \ x$ 
using assms by (simp add: ext_cont_def)

```

lemma *continuous_on_ext_cont*[continuous_intros]:

```

 $\text{continuous\_on } (\text{cbox } a \ b) \ f \implies \text{continuous\_on } S \ (\text{ext\_cont } f \ a \ b)$ 
by (auto intro!: clamp_continuous_on simp: ext_cont_def)

```

5.1.16 Separability

lemma *univ_second_countable_sequence*:

```

obtains  $B :: \text{nat} \Rightarrow 'a::\text{euclidean\_space} \ \text{set}$ 

```

```

    where inj B  $\wedge$  n. open(B n)  $\wedge$  S. open S  $\implies$   $\exists$  k. S =  $\bigcup$  {B n | n. n  $\in$  k}
  proof -
    obtain B :: 'a set set
    where countable B
      and opn:  $\bigwedge$  C. C  $\in$  B  $\implies$  open C
      and Un:  $\bigwedge$  S. open S  $\implies$   $\exists$  U. U  $\subseteq$  B  $\wedge$  S =  $\bigcup$  U
      using univ_second_countable by blast
    have *: infinite (range ( $\lambda$ n. ball (0::'a) (inverse(Suc n))))
      by (simp add: inj_on_def ball_eq_ball_iff Infinite_Set.range_inj_infinite)
    have infinite B
    proof
      assume finite B
      then have finite (Union ' (Pow B))
        by simp
      moreover have range ( $\lambda$ n. ball 0 (inverse (real (Suc n))))  $\subseteq$   $\bigcup$  ' Pow B
        by (metis (no_types, lifting) PowI image_eqI image_subset_iff Un [OF
open_ball])
      ultimately show False
        by (metis finite_subset *)
    qed
    obtain f :: nat  $\Rightarrow$  'a set where B = range f inj f
      by (blast intro: countable_as_injective_image [OF <countable B> <infinite B>])
    have *:  $\exists$  k. S =  $\bigcup$  {f n | n. n  $\in$  k} if open S for S
      using Un [OF that]
      apply clarify
      apply (rule_tac x=f-'U in exI)
      using <inj f> <B = range f> apply force
      done
    show ?thesis
      using * <B = range f> <inj f> opn that by force
  qed

```

proposition separable:

```

  fixes S :: 'a::{metric_space, second_countable_topology} set
  obtains T where countable T T  $\subseteq$  S S  $\subseteq$  closure T

```

proof -

```

  obtain B :: 'a set set
  where countable B
    and {}  $\notin$  B
    and ope:  $\bigwedge$  C. C  $\in$  B  $\implies$  openin(top_of_set S) C
    and if_ope:  $\bigwedge$  T. openin(top_of_set S) T  $\implies$   $\exists$  U. U  $\subseteq$  B  $\wedge$  T =  $\bigcup$  U
  by (meson subset_second_countable)
  then obtain f where f:  $\bigwedge$  C. C  $\in$  B  $\implies$  f C  $\in$  C
    by (metis equalsOI)
  show ?thesis
  proof
    show countable (f ' B)
      by (simp add: <countable B>)
    show f ' B  $\subseteq$  S

```

```

    using ope f openin_imp_subset by blast
  show  $S \subseteq \text{closure } (f \text{ ' } \mathcal{B})$ 
  proof (clarsimp simp: closure_approachable)
    fix x and e::real
    assume  $x \in S$   $0 < e$ 
    have openin (top_of_set S) (S  $\cap$  ball x e)
      by (simp add: openin_Int_open)
    with if_ope obtain  $\mathcal{U}$  where  $\mathcal{U} \subseteq \mathcal{B}$   $S \cap \text{ball } x \ e = \bigcup \mathcal{U}$ 
      by meson
    show  $\exists C \in \mathcal{B}. \text{dist } (f \ C) \ x < e$ 
    proof (cases  $\mathcal{U} = \{\}$ )
      case True
      then show ?thesis
        using  $\langle 0 < e \rangle$   $\mathcal{U} \langle x \in S \rangle$  by auto
    next
      case False
      then show ?thesis
        by (metis IntI Union_iff  $\mathcal{U} \langle 0 < e \rangle \langle x \in S \rangle \text{dist\_commute } \text{dist\_self } f$ 
        inf_le2 mem_ball subset_eq)
    qed
  qed
  qed
  qed

```

5.1.17 Diameter

lemma diameter_cball [simp]:

fixes $a :: 'a::\text{euclidean_space}$

shows $\text{diameter}(\text{cball } a \ r) = (\text{if } r < 0 \text{ then } 0 \text{ else } 2*r)$

proof -

have $\text{diameter}(\text{cball } a \ r) = 2*r$ if $r \geq 0$

proof (rule order_antisym)

show $\text{diameter } (\text{cball } a \ r) \leq 2*r$

proof (rule diameter_le)

fix $x \ y$ assume $x \in \text{cball } a \ r$ $y \in \text{cball } a \ r$

then have $\text{norm } (x - a) \leq r$ $\text{norm } (a - y) \leq r$

by (auto simp: dist_norm norm_minus_commute)

then have $\text{norm } (x - y) \leq r+r$

using norm_diff_triangle_le by blast

then show $\text{norm } (x - y) \leq 2*r$ by simp

qed (simp add: that)

have $2*r = \text{dist } (a + r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis})) \ (a - r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis}))$

using $\langle 0 \leq r \rangle$ that by (simp add: dist_norm flip: scaleR_2)

also have $\dots \leq \text{diameter } (\text{cball } a \ r)$

apply (rule diameter_bounded_bound)

using that by (auto simp: dist_norm)

finally show $2*r \leq \text{diameter } (\text{cball } a \ r)$.

qed

then show ?thesis by simp
qed

lemma diameter_ball [simp]:
fixes a :: 'a::euclidean_space
shows diameter(ball a r) = (if r < 0 then 0 else 2*r)
proof -
have diameter(ball a r) = 2*r if r > 0
by (metis bounded_ball diameter_closure closure_ball diameter_cball less_eq_real_def
linorder_not_less that)
then show ?thesis
by (simp add: diameter_def)
qed

lemma diameter_closed_interval [simp]: diameter {a..b} = (if b < a then 0 else
b-a)
proof -
have {a..b} = cball ((a+b)/2) ((b-a)/2)
using atLeastAtMost_eq_cball by blast
then show ?thesis
by simp
qed

lemma diameter_open_interval [simp]: diameter {a<..**b} = (if b < a then 0 else
b-a)
proof -
have {a <..**b} = ball ((a+b)/2) ((b-a)/2)
using greaterThanLessThan_eq_ball by blast
then show ?thesis
by simp
qed****

lemma diameter_cbox:
fixes a b::'a::euclidean_space
shows ($\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$) \implies diameter (cbox a b) = dist a b
by (force simp: diameter_def intro!: cSup_eq_maximum L2_set_mono
simp: euclidean_dist_l2[where 'a='a] cbox_def dist_norm)

5.1.18 Relating linear images to open/closed/interior/closure/connected

proposition open_surjective_linear_image:
fixes f :: 'a::real_normed_vector \Rightarrow 'b::euclidean_space
assumes open A linear f surj f
shows open(f ' A)
unfolding open_dist
proof clarify
fix x
assume x \in A

```

have bounded (inv f ' Basis)
  by (simp add: finite_imp_bounded)
with bounded_pos obtain B where B > 0 and B:  $\bigwedge x. x \in \text{inv } f \text{ ' Basis} \implies$ 
norm x  $\leq$  B
  by metis
obtain e where e > 0 and e:  $\bigwedge z. \text{dist } z \text{ } x < e \implies z \in A$ 
  by (metis open_dist ⟨x ∈ A⟩ ⟨open A⟩)
define δ where δ  $\equiv$  e / B / DIM('b)
show  $\exists e > 0. \forall y. \text{dist } y \text{ } (f \text{ } x) < e \longrightarrow y \in f \text{ ' } A$ 
proof (intro exI conjI)
  show δ > 0
    using ⟨e > 0⟩ ⟨B > 0⟩ by (simp add: δ_def field_split_simps)
  have y ∈ f ' A if dist y (f x) * (B * real DIM('b)) < e for y
  proof -
    define u where u  $\equiv$  y - f x
    show ?thesis
    proof (rule image_eqI)
      show y = f (x + ( $\sum i \in \text{Basis}. (u \cdot i) *_R \text{inv } f \text{ } i$ ))
      apply (simp add: linear_add linear_sum linear.scaleR ⟨linear f⟩ surj_f_inv_f
        ⟨surj f⟩)
      apply (simp add: euclidean_representation u_def)
      done
      have dist (x + ( $\sum i \in \text{Basis}. (u \cdot i) *_R \text{inv } f \text{ } i$ )) x  $\leq$  ( $\sum i \in \text{Basis}. \text{norm } ((u$ 
        · i) *_R inv f i))
        by (simp add: dist_norm sum_norm_le)
      also have ... = ( $\sum i \in \text{Basis}. |u \cdot i| * \text{norm } (\text{inv } f \text{ } i)$ )
        by simp
      also have ...  $\leq$  ( $\sum i \in \text{Basis}. |u \cdot i|$ ) * B
        by (simp add: B sum_distrib_right sum_mono mult_left_mono)
      also have ...  $\leq$  DIM('b) * dist y (f x) * B
        apply (rule mult_right_mono [OF sum_bounded_above])
        using ⟨0 < B⟩ by (auto simp: Basis_le_norm dist_norm u_def)
      also have ... < e
        by (metis mult.commute mult.left_commute that)
      finally show x + ( $\sum i \in \text{Basis}. (u \cdot i) *_R \text{inv } f \text{ } i$ ) ∈ A
        by (rule e)
    qed
  qed
qed
then show  $\forall y. \text{dist } y \text{ } (f \text{ } x) < \delta \longrightarrow y \in f \text{ ' } A$ 
  using ⟨e > 0⟩ ⟨B > 0⟩
  by (auto simp: δ_def field_split_simps)
qed
qed

corollary open_bijective_linear_image_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f bij f
  shows open(f ' A)  $\iff$  open A
proof

```

```

    assume open(f ' A)
  then show open A
    by (metis assms bij_is_inj continuous_open_vimage inj_vimage_image_eq
linear_continuous_at linear_linear)
next
  assume open A
  then show open(f ' A)
    by (simp add: assms bij_is_surj open_surjective_linear_image)
qed

```

corollary *interior_bijjective_linear_image*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f bij f
  shows interior (f ' S) = f ' interior S
  by (smt (verit) assms bij_is_inj inj_image_subset_iff interior_maximal interior_subset
open_bijjective_linear_image_eq open_interior subset_antisym subset_imageE)

```

lemma *interior_injective_linear_image*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a::euclidean_space
  assumes linear f inj f
  shows interior(f ' S) = f ' (interior S)
  by (simp add: linear_injective_imp_surjective assms bijI interior_bijjective_linear_image)

```

lemma *interior_surjective_linear_image*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a::euclidean_space
  assumes linear f surj f
  shows interior(f ' S) = f ' (interior S)
  by (simp add: assms interior_injective_linear_image linear_surjective_imp_injective)

```

lemma *interior_negations*:

```

  fixes S :: 'a::euclidean_space set
  shows interior(uminus ' S) = image uminus (interior S)
  by (simp add: bij_uminus interior_bijjective_linear_image linear_uminus)

```

lemma *connected_linear_image*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes linear f and connected s
  shows connected (f ' s)
using connected_continuous_image assms linear_continuous_on linear_conv_bounded_linear
by blast

```

5.1.19 "Isometry" (up to constant bounds) of Injective Linear Map

proposition *injective_imp_isometric*:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes s: closed s subspace s
  and f: bounded_linear f  $\forall x \in s. f x = 0 \longrightarrow x = 0$ 

```

```

  shows  $\exists e > 0. \forall x \in s. \text{norm } (f x) \geq e * \text{norm } x$ 
proof (cases  $s \subseteq \{0::'a\}$ )
  case True
  have  $\text{norm } x \leq \text{norm } (f x)$  if  $x \in s$  for  $x$ 
  proof -
    from True that have  $x = 0$  by auto
    then show ?thesis by simp
  qed
  then show ?thesis
  by (auto intro!: exI[where  $x=1$ ])
next
  case False
  interpret  $f$ : bounded_linear  $f$  by fact
  from False obtain  $a$  where  $a \neq 0$   $a \in s$ 
  by auto
  from False have  $s \neq \{\}$ 
  by auto
  let  $?S = \{f x \mid x. x \in s \wedge \text{norm } x = \text{norm } a\}$ 
  let  $?S' = \{x::'a. x \in s \wedge \text{norm } x = \text{norm } a\}$ 
  let  $?S'' = \{x::'a. \text{norm } x = \text{norm } a\}$ 

  have  $?S'' = \text{frontier } (\text{cball } 0 (\text{norm } a))$ 
  by (simp add: sphere_def dist_norm)
  then have compact  $?S''$  by (metis compact_cball compact_frontier)
  moreover have  $?S' = s \cap ?S''$  by auto
  ultimately have compact  $?S'$ 
  using closed_Int_compact[of  $s$   $?S''$ ] using  $s(1)$  by auto
  moreover have  $*:f ' ?S' = ?S$  by auto
  ultimately have compact  $?S$ 
  using compact_continuous_image[OF linear_continuous_on[OF  $f(1)$ ], of  $?S'$ ]
by auto
  then have closed  $?S$ 
  using compact_imp_closed by auto
  moreover from  $a$  have  $?S \neq \{\}$  by auto
  ultimately obtain  $b'$  where  $b' \in ?S \forall y \in ?S. \text{norm } b' \leq \text{norm } y$ 
  using distance_attains_inf[of  $?S$   $0$ ] unfolding dist_0_norm by auto
  then obtain  $b$  where  $b \in s$ 
  and  $ba: \text{norm } b = \text{norm } a$ 
  and  $b: \forall x \in \{x \in s. \text{norm } x = \text{norm } a\}. \text{norm } (f b) \leq \text{norm } (f x)$ 
  unfolding *[symmetric] unfolding image_iff by auto

  let  $?e = \text{norm } (f b) / \text{norm } b$ 
  have  $\text{norm } b > 0$ 
  using  $ba$  and  $a$  and norm_ge_zero by auto
  moreover have  $\text{norm } (f b) > 0$ 
  using  $f(2)$ [THEN bspec[where  $x=b$ ], OF  $\langle b \in s \rangle$ ]
  using  $\langle \text{norm } b > 0 \rangle$  by simp
  ultimately have  $0 < \text{norm } (f b) / \text{norm } b$  by simp
  moreover

```

```

have norm (f b) / norm b * norm x ≤ norm (f x) if x ∈ s for x
proof (cases x = 0)
  case True
  then show norm (f b) / norm b * norm x ≤ norm (f x)
  by auto
next
case False
with ⟨a ≠ 0⟩ have *: 0 < norm a / norm x
  unfolding zero_less_norm_iff[symmetric] by simp
have ∀ x ∈ s. c *R x ∈ s for c
  using s[unfolded subspace_def] by simp
with ⟨x ∈ s⟩ ⟨x ≠ 0⟩ have (norm a / norm x) *R x ∈ {x ∈ s. norm x = norm
a}
  by simp
with ⟨x ≠ 0⟩ ⟨a ≠ 0⟩ show norm (f b) / norm b * norm x ≤ norm (f x)
  using b[THEN bspec[where x=(norm a / norm x) *R x]]
  unfolding f.scaleR and ba
  by (auto simp: mult.commute pos_le_divide_eq pos_divide_le_eq)
qed
ultimately show ?thesis by auto
qed

```

```

proposition closed_injective_image_subspace:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace s bounded_linear f ∀ x ∈ s. f x = 0 → x = 0 closed s
  shows closed(f ' s)
proof –
  obtain e where e > 0 and e: ∀ x ∈ s. e * norm x ≤ norm (f x)
  using assms injective_imp_isometric by blast
  with assms show ?thesis
  by (meson complete_eq_closed complete_isometric_image)
qed

```

```

lemma closure_bounded_linear_image_subset:
  assumes f: bounded_linear f
  shows f ' closure S ⊆ closure (f ' S)
  using linear_continuous_on[OF f] closed_closure closure_subset
  by (rule image_closure_subset)

```

```

lemma closure_linear_image_subset:
  fixes f :: 'm::euclidean_space ⇒ 'n::real_normed_vector
  assumes linear f
  shows f ' (closure S) ⊆ closure (f ' S)
  using assms unfolding linear_conv_bounded_linear
  by (rule closure_bounded_linear_image_subset)

```

```

lemma closed_injective_linear_image:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space

```



```

    assumes  $S$ : closed  $S$  and  $f$ : linear  $f$  inj  $f$ 
    shows closed ( $f^{-1} S$ )
  proof -
    obtain  $g$  where  $g$ : linear  $g$   $g \circ f = \text{id}$ 
      using linear_injective_left_inverse [OF  $f$ ] by blast
    then have  $confg$ : continuous_on (range  $f$ )  $g$ 
      using linear_continuous_on_linear_conv_bounded_linear by blast
    have [simp]:  $g^{-1} f^{-1} S = S$ 
      using  $g$  by (simp add: image_comp)
    have  $cgf$ : closed ( $g^{-1} f^{-1} S$ )
      by (simp add:  $\langle g \circ f = \text{id} \rangle$   $S$  image_comp)
    have [simp]: (range  $f \cap g^{-1} S$ ) =  $f^{-1} S$ 
      using  $g$  unfolding o_def id_def image_def by auto
    metis+
    show ?thesis
  proof (rule closedin_closed_trans [of range  $f$ ])
    show closedin (top_of_set (range  $f$ )) ( $f^{-1} S$ )
      using continuous_closedin_preimage [OF  $confg$   $cgf$ ] by simp
    show closed (range  $f$ )
      using closed_injective_image_subspace  $f$  linear_conv_bounded_linear
        linear_injective_0 subspace_UNIV by blast
  qed
qed

lemma closed_injective_linear_image_eq:
  fixes  $f$  :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes  $f$ : linear  $f$  inj  $f$ 
  shows (closed (image  $f$   $s$ ))  $\iff$  closed  $s$ 
  by (metis closed_injective_linear_image_closure_eq closure_linear_image_subset
    closure_subset_eq  $f(1)$   $f(2)$  inj_image_subset_iff)

lemma closure_injective_linear_image:
  fixes  $f$  :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows  $\llbracket$  linear  $f$ ; inj  $f$   $\rrbracket \implies f^{-1} (\text{closure } S) = \text{closure } (f^{-1} S)$ 
  by (simp add: closed_injective_linear_image_closure_linear_image_subset
    closure_minimal closure_subset image_mono subset_antisym)

lemma closure_bounded_linear_image:
  fixes  $f$  :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear  $f$  bounded  $S$ 
  shows  $f^{-1} (\text{closure } S) = \text{closure } (f^{-1} S)$  (is ?lhs = ?rhs)
  proof
    show ?lhs  $\subseteq$  ?rhs
      using assms closure_linear_image_subset by blast
    show ?rhs  $\subseteq$  ?lhs
      using assms by (meson closure_minimal closure_subset compact_closure
        compact_eq_bounded_closed
          compact_continuous_image image_mono linear_continuous_on
            linear_linear)
  qed

```

```

lemma closure_scaleR:
  fixes S :: 'a::real_normed_vector set
  shows ((*R) c) ' (closure S) = closure (((*R) c) ' S) (is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
    using bounded_linear_scaleR_right by (rule closure_bounded_linear_image_subset)
  show ?rhs ⊆ ?lhs
    by (intro closure_minimal_image_mono closure_subset closed_scaling closed_closure)
qed

```

5.1.20 Some properties of a canonical subspace

```

lemma closed_substandard: closed {x::'a::euclidean_space. ∀ i∈Basis. P i ⟶ x·i = 0}
  (is closed ?A)
proof -
  let ?D = {i∈Basis. P i}
  have closed (⋂ i∈?D. {x::'a. x·i = 0})
    by (simp add: closed_INT closed_Collect_eq continuous_on_inner)
  also have (⋂ i∈?D. {x::'a. x·i = 0}) = ?A
    by auto
  finally show closed ?A .
qed

```

```

lemma closed_subspace:
  fixes S :: 'a::euclidean_space set
  assumes subspace S
  shows closed S
proof -
  have dim S ≤ card (Basis :: 'a set)
    using dim_subset_UNIV by auto
  with obtain_subset_with_card_n
  obtain d :: 'a set where cd: card d = dim S and d: d ⊆ Basis
    bymetis
  let ?t = {x::'a. ∀ i∈Basis. i ∉ d ⟶ x·i = 0}
  have ∃ f. linear f ∧ f ' {x::'a. ∀ i∈Basis. i ∉ d ⟶ x·i = 0} = S ∧
    inj_on f {x::'a. ∀ i∈Basis. i ∉ d ⟶ x·i = 0}
    using dim_substandard[of d] cd d assms
    by (intro subspace_isomorphism[OF subspace_substandard[of λi. i ∉ d]]) (auto simp: inner_Basis)
  then obtain f where f:
    linear f
    f ' {x. ∀ i∈Basis. i ∉ d ⟶ x·i = 0} = S
    inj_on f {x. ∀ i∈Basis. i ∉ d ⟶ x·i = 0}
    by blast
  interpret f: bounded_linear f
    using f by (simp add: linear_conv_bounded_linear)
  have x ∈ ?t ⟹ f x = 0 ⟹ x = 0 for x

```

```

  using f.zero d f(3)[THEN inj_onD, of x 0] by auto
  then show ?thesis
  using closed_injective_image_subspace[of ?t f] closed_substandard_subspace_substandard
  using f(2) f.bounded_linear_axioms by force
qed

```

```

lemma complete_subspace: subspace S  $\implies$  complete S
  for S :: 'a::euclidean_space set
  using complete_eq_closed closed_subspace by auto

```

```

lemma closed_span [iff]: closed (span S)
  for S :: 'a::euclidean_space set
  by (simp add: closed_subspace)

```

```

lemma dim_closure [simp]: dim (closure S) = dim S (is ?dc = ?d)
  for S :: 'a::euclidean_space set
  by (metis closed_span closure_minimal closure_subset dim_eq_span span_eq_dim
  span_superset_subset_le_dim)

```

5.1.21 Set Distance

```

lemma setdist_compact_closed:
  fixes A :: 'a::heine_borel set
  assumes compact A closed B
  and A  $\neq$  {} B  $\neq$  {}
  shows  $\exists x \in A. \exists y \in B. \text{dist } x \ y = \text{setdist } A \ B$ 
  by (metis assms infdist_attains_inf setdist_attains_inf setdist_sym)

```

```

lemma setdist_closed_compact:
  fixes S :: 'a::heine_borel set
  assumes S: closed S and T: compact T
  and S  $\neq$  {} T  $\neq$  {}
  shows  $\exists x \in S. \exists y \in T. \text{dist } x \ y = \text{setdist } S \ T$ 
  using setdist_compact_closed [OF T S  $\langle T \neq \{\} \rangle$   $\langle S \neq \{\} \rangle$ ]
  by (metis dist_commute setdist_sym)

```

```

lemma setdist_eq_0_compact_closed:
  assumes S: compact S and T: closed T
  shows  $\text{setdist } S \ T = 0 \iff S = \{\} \vee T = \{\} \vee S \cap T \neq \{\}$ 

```

```

proof (cases S = {}  $\vee$  T = {})
  case False
  then show ?thesis
  by (metis S T disjoint_iff_in_closed_iff_infdist_zero setdist_attains_inf set-
  dist_eq_0I setdist_sym)
qed auto

```

```

corollary setdist_gt_0_compact_closed:
  assumes S: compact S and T: closed T
  shows  $\text{setdist } S \ T > 0 \iff (S \neq \{\} \wedge T \neq \{\} \wedge S \cap T = \{\})$ 

```

using *setdist_pos_le* [of *S T*] *setdist_eq_0_compact_closed* [OF *assms*] **by** *linarith*

lemma *setdist_eq_0_closed_compact*:
assumes *S*: *closed S* **and** *T*: *compact T*
shows $\text{setdist } S \ T = 0 \longleftrightarrow S = \{\} \vee T = \{\} \vee S \cap T \neq \{\}$
using *setdist_eq_0_compact_closed* [OF *T S*]
by (*metis Int_commute setdist_sym*)

lemma *setdist_eq_0_bounded*:
fixes *S* :: 'a::heine_borel *set*
assumes *bounded S* \vee *bounded T*
shows $\text{setdist } S \ T = 0 \longleftrightarrow S = \{\} \vee T = \{\} \vee \text{closure } S \cap \text{closure } T \neq \{\}$
proof (*cases S = {} \vee T = {}*)
case *False*
then show *?thesis*
using *setdist_eq_0_compact_closed* [of *closure S closure T*]
setdist_eq_0_closed_compact [of *closure S closure T*] *assms*
by (*force simp: bounded_closure compact_eq_bounded_closed*)
qed *force*

lemma *setdist_eq_0_sing_1*:
 $\text{setdist } \{x\} \ S = 0 \longleftrightarrow S = \{\} \vee x \in \text{closure } S$
by (*metis in_closure_iff_infdist_zero infdist_def infdist_eq_setdist*)

lemma *setdist_eq_0_sing_2*:
 $\text{setdist } S \ \{x\} = 0 \longleftrightarrow S = \{\} \vee x \in \text{closure } S$
by (*metis setdist_eq_0_sing_1 setdist_sym*)

lemma *setdist_neq_0_sing_1*:
 $\llbracket \text{setdist } \{x\} \ S = a; a \neq 0 \rrbracket \Longrightarrow S \neq \{\} \wedge x \notin \text{closure } S$
by (*metis setdist_closure_2 setdist_empty2 setdist_eq_0I singletonI*)

lemma *setdist_neq_0_sing_2*:
 $\llbracket \text{setdist } S \ \{x\} = a; a \neq 0 \rrbracket \Longrightarrow S \neq \{\} \wedge x \notin \text{closure } S$
by (*simp add: setdist_neq_0_sing_1 setdist_sym*)

lemma *setdist_sing_in_set*:
 $x \in S \Longrightarrow \text{setdist } \{x\} \ S = 0$
by (*simp add: setdist_eq_0I*)

lemma *setdist_eq_0_closed*:
 $\text{closed } S \Longrightarrow (\text{setdist } \{x\} \ S = 0 \longleftrightarrow S = \{\} \vee x \in S)$
by (*simp add: setdist_eq_0_sing_1*)

lemma *setdist_eq_0_closedin*:
shows $\llbracket \text{closedin } (\text{top_of_set } U) \ S; x \in U \rrbracket$
 $\Longrightarrow (\text{setdist } \{x\} \ S = 0 \longleftrightarrow S = \{\} \vee x \in S)$
by (*auto simp: closedin_limpt setdist_eq_0_sing_1 closure_def*)

```

lemma setdist_gt_0_closedin:
  shows  $\llbracket \text{closedin } (\text{top\_of\_set } U) \ S; \ x \in U; \ S \neq \{\}; \ x \notin S \rrbracket$ 
     $\implies \text{setdist } \{x\} \ S > 0$ 
  using less_eq_real_def setdist_eq_0_closedin by fastforce

```

A consequence of the results above

```

lemma compact_minkowski_sum_cball:
  fixes  $A :: 'a :: \text{heine\_borel\_set}$ 
  assumes compact A
  shows  $\text{compact } (\bigcup_{x \in A} \text{cball } x \ r)$ 
proof (cases A = {})
  case False
  show ?thesis
  unfolding compact_eq_bounded_closed
  proof safe
    have  $\text{open } (\neg(\bigcup_{x \in A} \text{cball } x \ r))$ 
      unfolding open_dist
    proof safe
      fix  $x$  assume  $x: x \notin (\bigcup_{x \in A} \text{cball } x \ r)$ 
      have  $\exists x' \in \{x\}. \exists y \in A. \text{dist } x' \ y = \text{setdist } \{x\} \ A$ 
        using  $\langle A \neq \{\} \rangle$  assms
        by (intro setdist_compact_closed) (auto simp: compact_imp_closed)
      then obtain  $y$  where  $y: y \in A \ \text{dist } x \ y = \text{setdist } \{x\} \ A$ 
        by blast
      with  $x$  have  $\text{setdist } \{x\} \ A > r$ 
        by (auto simp: dist_commute)
      moreover have False if  $\text{dist } z \ x < \text{setdist } \{x\} \ A - r$   $u \in A \ z \in \text{cball } u \ r$  for
         $z \ u$ 
        by (smt (verit, del_insts) mem_cball setdist_Lipschitz setdist_sing_in_set
that)
      ultimately
      show  $\exists e > 0. \forall y. \text{dist } y \ x < e \implies y \in \neg(\bigcup_{x \in A} \text{cball } x \ r)$ 
        by (force intro!: exI[of _ setdist {x} A - r])
    qed
  thus  $\text{closed } (\bigcup_{x \in A} \text{cball } x \ r)$ 
    using closed_open by blast
  next
  from assms have bounded A
    by (simp add: compact_imp_bounded)
  then obtain  $c$  where  $c: \bigwedge y. y \in A \implies \text{dist } x \ y \leq c$ 
    unfolding bounded_def by blast
  have  $\forall y \in (\bigcup_{x \in A} \text{cball } x \ r). \text{dist } x \ y \leq c + r$ 
  proof safe
    fix  $y \ z$  assume  $*$ :  $y \in A \ z \in \text{cball } y \ r$ 
    thus  $\text{dist } x \ z \leq c + r$ 
      using  $* \ c$  [of y] cball_trans mem_cball by blast
    qed
  thus bounded  $(\bigcup_{x \in A} \text{cball } x \ r)$ 

```

```

      unfolding bounded_def by blast
    qed
  qed auto

no_notation
  eucl_less (infix <e 50)

end

```

5.2 Line Segment

```

theory Line_Segment
imports
  Convex
  Topology_Euclidean_Space
begin

```

5.2.1 Topological Properties of Convex Sets, Metric Spaces and Functions

```

lemma convex_supp_sum:
  assumes convex S and 1: supp_sum u I = 1
    and  $\bigwedge i. i \in I \implies 0 \leq u\ i \wedge (u\ i = 0 \vee f\ i \in S)$ 
  shows supp_sum ( $\lambda i. u\ i *_{\mathbb{R}} f\ i$ ) I  $\in S$ 
proof -
  have fin: finite {i  $\in I. u\ i \neq 0$ }
    using 1 sum.infinite by (force simp: supp_sum_def support_on_def)
  then have supp_sum ( $\lambda i. u\ i *_{\mathbb{R}} f\ i$ ) I = sum ( $\lambda i. u\ i *_{\mathbb{R}} f\ i$ ) {i  $\in I. u\ i \neq 0$ }
    by (force intro: sum.mono_neutral_left simp: supp_sum_def support_on_def)
  also have ...  $\in S$ 
    using 1 assms by (force simp: supp_sum_def support_on_def intro: convex_sum [OF fin <convex S>])
  finally show ?thesis .
qed

lemma sphere_eq_empty [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space}
  shows sphere a r = {}  $\iff r < 0$ 
  by (metis empty_iff linorder_not_less mem_sphere sphere_empty vector_choose_dist)

lemma cone_closure:
  fixes S :: 'a::{real_normed_vector} set
  assumes cone S
  shows cone (closure S)
  by (metis UnCI assms closure_Un_frontier closure_eq_empty closure_scaleR cone_iff)

corollary component_complement_connected:

```

```

fixes  $S :: 'a::\text{real\_normed\_vector\_set}$ 
assumes  $\text{connected } S \ C \in \text{components } (-S)$ 
shows  $\text{connected } (-C)$ 
using  $\text{component\_diff\_connected [of } S \ UNIV] \text{ assms}$ 
by  $(\text{auto simp: Compl\_eq\_Diff\_UNIV})$ 

```

proposition *clopen*:

```

fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
shows  $\text{closed } S \wedge \text{open } S \longleftrightarrow S = \{\} \vee S = UNIV$ 
using  $\text{connected\_UNIV by (force simp add: connected\_clopen)}$ 

```

corollary *compact_open*:

```

fixes  $S :: 'a :: \text{euclidean\_space\_set}$ 
shows  $\text{compact } S \wedge \text{open } S \longleftrightarrow S = \{\}$ 
by  $(\text{auto simp: compact\_eq\_bounded\_closed\_clopen})$ 

```

corollary *finite_imp_not_open*:

```

fixes  $S :: 'a::\{\text{real\_normed\_vector, perfect\_space}\} \text{ set}$ 
shows  $\llbracket \text{finite } S; \text{open } S \rrbracket \implies S = \{\}$ 
using  $\text{clopen [of } S] \text{ finite\_imp\_closed\_not\_bounded\_UNIV by blast}$ 

```

corollary *empty_interior_finite*:

```

fixes  $S :: 'a::\{\text{real\_normed\_vector, perfect\_space}\} \text{ set}$ 
shows  $\text{finite } S \implies \text{interior } S = \{\}$ 
by  $(\text{metis interior\_subset\_finite\_subset\_open\_interior [of } S] \text{ finite\_imp\_not\_open})$ 

```

Balls, being convex, are connected.

lemma *convex_local_global_minimum*:

```

fixes  $s :: 'a::\text{real\_normed\_vector\_set}$ 
assumes  $e > 0$ 
and  $\text{convex\_on } s \ f$ 
and  $\text{ball } x \ e \subseteq s$ 
and  $\forall y \in \text{ball } x \ e. f \ x \leq f \ y$ 
shows  $\forall y \in s. f \ x \leq f \ y$ 
proof  $(\text{rule ccontr})$ 
have  $x \in s$  using  $\text{assms}(1,3)$  by auto
assume  $\neg ?thesis$ 
then obtain  $y$  where  $y \in s$  and  $y: f \ x > f \ y$  by auto
then have  $xy: 0 < \text{dist } x \ y$  by auto
then obtain  $u$  where  $0 < u \ u \leq 1$  and  $u: u < e / \text{dist } x \ y$ 
using  $\text{field\_lbound\_gt\_zero [of } 1 \ e / \text{dist } x \ y] \ xy \langle e > 0 \rangle$  by auto
then have  $f \ ((1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y) \leq (1-u) * f \ x + u * f \ y$ 
using  $\langle x \in s \rangle \langle y \in s \rangle$  by  $(\text{smt (verit) assms}(2) \text{convex\_on\_def})$ 
moreover
have  $*$ :  $x - ((1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y) = u *_{\mathbb{R}} (x - y)$ 
by  $(\text{simp add: algebra\_simps})$ 
have  $(1-u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y \in \text{ball } x \ e$ 
by  $(\text{smt (verit) } * \langle 0 < u \rangle \text{dist\_norm mem\_ball norm\_scaleR pos\_less\_divide\_eq } u \ xy)$ 

```

```

then have  $f x \leq f ((1 - u) *_R x + u *_R y)$ 
  using assms(4) by auto
ultimately show False
  using mult_strict_left_mono[OF y <u>0]
  unfolding left_diff_distrib
  by auto

```

qed

```

lemma convex_ball [iff]:
  fixes  $x :: 'a::real\_normed\_vector$ 
  shows convex (ball  $x e$ )
proof (auto simp: convex_def)
  fix  $y z$ 
  assume yz:  $dist\ x\ y < e\ dist\ x\ z < e$ 
  fix  $u v :: real$ 
  assume uv:  $0 \leq u \leq v\ u + v = 1$ 
  have  $dist\ x\ (u *_R y + v *_R z) \leq u * dist\ x\ y + v * dist\ x\ z$ 
    using uv yz by (meson UNIV_I convex_def convex_on_def convex_on_dist)
  then show  $dist\ x\ (u *_R y + v *_R z) < e$ 
    using convex_bound_lt[OF yz uv] by auto

```

qed

```

lemma convex_cball [iff]:
  fixes  $x :: 'a::real\_normed\_vector$ 
  shows convex (cball  $x e$ )
proof -
  {
    fix  $y z$ 
    assume yz:  $dist\ x\ y \leq e\ dist\ x\ z \leq e$ 
    fix  $u v :: real$ 
    assume uv:  $0 \leq u \leq v\ u + v = 1$ 
    have  $dist\ x\ (u *_R y + v *_R z) \leq u * dist\ x\ y + v * dist\ x\ z$ 
      using uv yz by (meson UNIV_I convex_def convex_on_def convex_on_dist)
    then have  $dist\ x\ (u *_R y + v *_R z) \leq e$ 
      using convex_bound_le[OF yz uv] by auto
  }
  then show ?thesis by (auto simp: convex_def Ball_def)

```

qed

```

lemma connected_ball [iff]:
  fixes  $x :: 'a::real\_normed\_vector$ 
  shows connected (ball  $x e$ )
  using convex_connected convex_ball by auto

```

```

lemma connected_cball [iff]:
  fixes  $x :: 'a::real\_normed\_vector$ 
  shows connected (cball  $x e$ )
  using convex_connected convex_cball by auto

```



```

lemma bounded_convex_hull:
  fixes  $s :: 'a::real\_normed\_vector$  set
  assumes bounded s
  shows bounded (convex hull s)
proof -
  from assms obtain  $B$  where  $B: \forall x \in s. \text{norm } x \leq B$ 
  unfolding bounded_iff by auto
  show ?thesis
  by (simp add: bounded_subset[OF bounded_cball, of _ 0 B] B subsetI subset_hull)
qed

```

```

lemma finite_imp_bounded_convex_hull:
  fixes  $s :: 'a::real\_normed\_vector$  set
  shows finite s  $\implies$  bounded (convex hull s)
  using bounded_convex_hull finite_imp_bounded
  by auto

```

5.2.2 Midpoint

```

definition midpoint ::  $'a::real\_vector \Rightarrow 'a \Rightarrow 'a$ 
  where midpoint a b = (inverse (2::real)) *R (a + b)

```

```

lemma midpoint_idem [simp]: midpoint x x = x
  unfolding midpoint_def by simp

```

```

lemma midpoint_sym: midpoint a b = midpoint b a
  unfolding midpoint_def by (auto simp add: scaleR_right_distrib)

```

```

lemma midpoint_eq_iff: midpoint a b = c  $\longleftrightarrow$  a + b = c + c
proof -
  have midpoint a b = c  $\longleftrightarrow$  scaleR 2 (midpoint a b) = scaleR 2 c
  by simp
  then show ?thesis
  unfolding midpoint_def scaleR_2 [symmetric] by simp
qed

```

```

lemma
  fixes  $a::real$ 
  assumes  $a \leq b$  shows ge_midpoint_1: a  $\leq$  midpoint a b
  and le_midpoint_1: midpoint a b  $\leq$  b
  by (simp_all add: midpoint_def assms)

```

```

lemma dist_midpoint:
  fixes  $a b :: 'a::real\_normed\_vector$  shows
  dist a (midpoint a b) = (dist a b) / 2 (is ?t1)
  dist b (midpoint a b) = (dist a b) / 2 (is ?t2)
  dist (midpoint a b) a = (dist a b) / 2 (is ?t3)
  dist (midpoint a b) b = (dist a b) / 2 (is ?t4)

```

proof –

```

have *:  $\bigwedge x y :: 'a. 2 *_{\mathbb{R}} x = - y \implies \text{norm } x = (\text{norm } y) / 2$ 
  unfolding equation_minus_iff by auto
have **:  $\bigwedge x y :: 'a. 2 *_{\mathbb{R}} x = y \implies \text{norm } x = (\text{norm } y) / 2$ 
  by auto
note scaleR_right_distrib [simp]
show ?t1
  unfolding midpoint_def dist_norm
  apply (rule **)
  apply (simp add: scaleR_right_diff_distrib)
  apply (simp add: scaleR_2)
  done
show ?t2
  unfolding midpoint_def dist_norm
  apply (rule *)
  apply (simp add: scaleR_right_diff_distrib)
  apply (simp add: scaleR_2)
  done
show ?t3
  unfolding midpoint_def dist_norm
  apply (rule *)
  apply (simp add: scaleR_right_diff_distrib)
  apply (simp add: scaleR_2)
  done
show ?t4
  unfolding midpoint_def dist_norm
  apply (rule **)
  apply (simp add: scaleR_right_diff_distrib)
  apply (simp add: scaleR_2)
  done

```

qed

```

lemma midpoint_eq_endpoint [simp]:
  midpoint a b = a  $\longleftrightarrow$  a = b
  midpoint a b = b  $\longleftrightarrow$  a = b
  unfolding midpoint_eq_iff by auto

```

```

lemma midpoint_plus_self [simp]: midpoint a b + midpoint a b = a + b
  using midpoint_eq_iff by metis

```

```

lemma midpoint_linear_image:
  linear f  $\implies$  midpoint(f a)(f b) = f(midpoint a b)
  by (simp add: linear_iff midpoint_def)

```

5.2.3 Open and closed segments

```

definition closed_segment ::  $'a :: \text{real\_vector} \Rightarrow 'a \Rightarrow 'a \text{ set}$ 
  where closed_segment a b = \{(1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b \mid u :: \text{real}. 0 \leq u \wedge u \leq 1\}

```

definition *open_segment* :: 'a::real_vector \Rightarrow 'a \Rightarrow 'a set **where**
open_segment a b \equiv *closed_segment* a b - {a,b}

lemmas *segment* = *open_segment_def* *closed_segment_def*

lemma *in_segment*:

$x \in \text{closed_segment } a \ b \longleftrightarrow (\exists u. 0 \leq u \wedge u \leq 1 \wedge x = (1 - u) *_R a + u *_R b)$

$x \in \text{open_segment } a \ b \longleftrightarrow a \neq b \wedge (\exists u. 0 < u \wedge u < 1 \wedge x = (1 - u) *_R a + u *_R b)$

using *less_eq_real_def* **by** (*auto simp: segment algebra_simps*)

lemma *closed_segment_linear_image*:

closed_segment (f a) (f b) = f ' (*closed_segment* a b) **if** *linear* f

proof -

interpret *linear* f **by** *fact*

show *?thesis*

by (*force simp add: in_segment add scale*)

qed

lemma *open_segment_linear_image*:

$\llbracket \text{linear } f; \text{inj } f \rrbracket \Longrightarrow \text{open_segment } (f a) \ (f b) = f ' (\text{open_segment } a \ b)$

by (*force simp: open_segment_def closed_segment_linear_image inj_on_def*)

lemma *closed_segment_translation*:

closed_segment (c + a) (c + b) = $(\lambda x. c + x)$ ' (*closed_segment* a b) (**is** ?L =
 _ ' ?R)

proof -

have $\bigwedge x. x \in ?L \Longrightarrow x - c \in ?R \ \bigwedge x. \llbracket x \in ?R \rrbracket \Longrightarrow c + x \in ?L$

by (*auto simp: in_segment algebra_simps*)

then show *?thesis* **by** *force*

qed

lemma *open_segment_translation*:

open_segment (c + a) (c + b) = *image* $(\lambda x. c + x)$ (*open_segment* a b)

by (*simp add: open_segment_def closed_segment_translation translation_diff*)

lemma *closed_segment_of_real*:

closed_segment (*of_real* x) (*of_real* y) = *of_real* ' *closed_segment* x y

by (*simp add: closed_segment_linear_image linearI scaleR_conv_of_real*)

lemma *open_segment_of_real*:

open_segment (*of_real* x) (*of_real* y) = *of_real* ' *open_segment* x y

by (*simp add: closed_segment_of_real image_set_diff inj_of_real open_segment_def*)

lemma *closed_segment_Reals*:

$\llbracket x \in \text{Reals}; y \in \text{Reals} \rrbracket \Longrightarrow \text{closed_segment } x \ y = \text{of_real } ' \text{closed_segment } (\text{Re } x) \ (\text{Re } y)$

by (*metis closed_segment_of_real of_real_Re*)

lemma *open_segment_Reals*:

$\llbracket x \in \text{Reals}; y \in \text{Reals} \rrbracket \implies \text{open_segment } x \ y = \text{of_real } \langle \text{open_segment } (\text{Re } x) (\text{Re } y) \rangle$
by (*metis open_segment_of_real of_real_Re*)

lemma *open_segment_PairD*:

$(x, x') \in \text{open_segment } (a, a') (b, b') \implies (x \in \text{open_segment } a \ b \vee a = b) \wedge (x' \in \text{open_segment } a' \ b' \vee a' = b')$
by (*auto simp: in_segment*)

lemma *closed_segment_PairD*:

$(x, x') \in \text{closed_segment } (a, a') (b, b') \implies x \in \text{closed_segment } a \ b \wedge x' \in \text{closed_segment } a' \ b'$
by (*auto simp: closed_segment_def*)

lemma *closed_segment_translation_eq* [*simp*]:

$d + x \in \text{closed_segment } (d + a) (d + b) \longleftrightarrow x \in \text{closed_segment } a \ b$

proof –

have *: $\bigwedge d \ x \ a \ b. x \in \text{closed_segment } a \ b \implies d + x \in \text{closed_segment } (d + a) (d + b)$

using *closed_segment_translation* **by** *blast*

show *?thesis*

using * [**where** $d = -d$] * **by** *fastforce*

qed

lemma *open_segment_translation_eq* [*simp*]:

$d + x \in \text{open_segment } (d + a) (d + b) \longleftrightarrow x \in \text{open_segment } a \ b$

by (*simp add: open_segment_def*)

lemma *of_real_closed_segment* [*simp*]:

$\text{of_real } x \in \text{closed_segment } (\text{of_real } a) (\text{of_real } b) \longleftrightarrow x \in \text{closed_segment } a \ b$

by (*simp add: closed_segment_of_real_image_iff*)

lemma *of_real_open_segment* [*simp*]:

$\text{of_real } x \in \text{open_segment } (\text{of_real } a) (\text{of_real } b) \longleftrightarrow x \in \text{open_segment } a \ b$

by (*simp add: image_iff open_segment_of_real*)

lemma *convex_contains_segment*:

$\text{convex } S \longleftrightarrow (\forall a \in S. \forall b \in S. \text{closed_segment } a \ b \subseteq S)$

unfolding *convex_alt closed_segment_def* **by** *auto*

lemma *closed_segment_in_Reals*:

$\llbracket x \in \text{closed_segment } a \ b; a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies x \in \text{Reals}$

by (*meson subsetD convex_Reals convex_contains_segment*)

lemma *open_segment_in_Reals*:

$\llbracket x \in \text{open_segment } a \ b; a \in \text{Reals}; b \in \text{Reals} \rrbracket \implies x \in \text{Reals}$

by (*metis Diff_iff closed_segment_in_Reals open_segment_def*)

lemma *closed_segment_subset*: $\llbracket x \in S; y \in S; \text{convex } S \rrbracket \implies \text{closed_segment } x \ y \subseteq S$
by (*simp add: convex_contains_segment*)

lemma *closed_segment_subset_convex_hull*:
 $\llbracket x \in \text{convex hull } S; y \in \text{convex hull } S \rrbracket \implies \text{closed_segment } x \ y \subseteq \text{convex hull } S$
using *convex_contains_segment* **by** *blast*

lemma *segment_convex_hull*:
 $\text{closed_segment } a \ b = \text{convex hull } \{a, b\}$
proof –
have *: $\bigwedge x. \{x\} \neq \{\}$ **by** *auto*
show *?thesis*
unfolding *segment_convex_hull_insert[OF *]* *convex_hull_singleton*
by (*safe; rule_tac x=1 - u in exI; force*)
qed

lemma *open_closed_segment*: $u \in \text{open_segment } w \ z \implies u \in \text{closed_segment } w \ z$
by (*auto simp add: closed_segment_def open_segment_def*)

lemma *segment_open_subset_closed*:
 $\text{open_segment } a \ b \subseteq \text{closed_segment } a \ b$
by (*simp add: open_closed_segment_subsetI*)

lemma *bounded_closed_segment*:
fixes $a :: 'a::\text{real_normed_vector}$ **shows** *bounded (closed_segment a b)*
by (*simp add: bounded_convex_hull segment_convex_hull*)

lemma *bounded_open_segment*:
fixes $a :: 'a::\text{real_normed_vector}$ **shows** *bounded (open_segment a b)*
by (*rule bounded_subset [OF bounded_closed_segment segment_open_subset_closed]*)

lemmas *bounded_segment = bounded_closed_segment open_closed_segment*

lemma *ends_in_segment [iff]*: $a \in \text{closed_segment } a \ b \iff b \in \text{closed_segment } a \ b$
by (*simp_all add: hull_inc segment_convex_hull*)

lemma *eventually_closed_segment*:
fixes $x0 :: 'a::\text{real_normed_vector}$
assumes *open X0 x0 ∈ X0*
shows $\forall_F x \text{ in at } x0 \text{ within } U. \text{closed_segment } x0 \ x \subseteq X0$
proof –
from *openE[OF assms]*
obtain e **where** $0 < e \text{ ball } x0 \ e \subseteq X0$.
then have $\forall_F x \text{ in at } x0 \text{ within } U. x \in \text{ball } x0 \ e$
by (*auto simp: dist_commute eventually_at*)

```

then show ?thesis
proof eventually_elim
  case (elim x)
  have  $x0 \in \text{ball } x0 \ e$  using  $\langle e \rangle > 0$  by simp
  then have  $\text{closed\_segment } x0 \ x \subseteq \text{ball } x0 \ e$ 
    using  $\text{closed\_segment\_subset } \text{elim}$  by blast
  then show ?case
    using  $e(2)$  by auto
qed
qed

lemma  $\text{closed\_segment\_commute}$ :  $\text{closed\_segment } a \ b = \text{closed\_segment } b \ a$ 
proof -
  have  $\{a, b\} = \{b, a\}$  by auto
  thus ?thesis
    by (simp add:  $\text{segment\_convex\_hull}$ )
qed

lemma  $\text{segment\_bound1}$ :
  assumes  $x \in \text{closed\_segment } a \ b$ 
  shows  $\text{norm } (x - a) \leq \text{norm } (b - a)$ 
proof -
  obtain  $u$  where  $u: x = (1 - u) *_R a + u *_R b$   $0 \leq u \leq 1$ 
    using  $\text{assms}$  by (auto simp add:  $\text{closed\_segment\_def}$ )
  then have  $\text{norm } (u *_R b - u *_R a) \leq \text{norm } (b - a)$ 
    by (simp add:  $\text{mult\_left\_le\_one\_le\_flip}$ :  $\text{scaleR\_diff\_right}$ )
  with  $u$  show ?thesis
    by (metis  $\text{add\_diff\_cancel\_left } \text{scaleR\_collapse}$ )
qed

lemma  $\text{segment\_bound}$ :
  assumes  $x \in \text{closed\_segment } a \ b$ 
  shows  $\text{norm } (x - a) \leq \text{norm } (b - a)$   $\text{norm } (x - b) \leq \text{norm } (b - a)$ 
  by (metis  $\text{assms } \text{closed\_segment\_commute } \text{dist\_commute } \text{dist\_norm } \text{segment\_bound1}$ )

lemma  $\text{open\_segment\_bound1}$ :
  assumes  $x \in \text{open\_segment } a \ b$ 
  shows  $\text{norm } (x - a) < \text{norm } (b - a)$ 
proof -
  obtain  $u$  where  $u: x = (1 - u) *_R a + u *_R b$   $0 < u < 1$ 
    by (meson  $\text{assms } \text{in\_segment}$ )
  then have  $\text{norm } (u *_R b - u *_R a) < \text{norm } (b - a)$ 
    using  $\text{assms } \text{in\_segment}(2)$   $\text{less\_eq\_real\_def}$  by (fastforce simp flip:  $\text{scaleR\_diff\_right}$ )
  with  $u$  show ?thesis
    by (metis  $\text{add\_diff\_cancel\_left } \text{scaleR\_collapse}$ )
qed

lemma  $\text{open\_segment\_commute}$ :  $\text{open\_segment } a \ b = \text{open\_segment } b \ a$ 
  by (simp add:  $\text{closed\_segment\_commute } \text{insert\_commute } \text{open\_segment\_def}$ )

```

lemma *closed_segment_idem* [simp]: $\text{closed_segment } a \ a = \{a\}$
unfolding *segment* **by** (auto simp add: algebra_simps)

lemma *open_segment_idem* [simp]: $\text{open_segment } a \ a = \{\}$
by (simp add: open_segment_def)

lemma *closed_segment_eq_open*: $\text{closed_segment } a \ b = \text{open_segment } a \ b \cup \{a, b\}$
using *open_segment_def* **by** auto

lemma *convex_contains_open_segment*:
 $\text{convex } s \iff (\forall a \in s. \forall b \in s. \text{open_segment } a \ b \subseteq s)$
by (simp add: convex_contains_segment closed_segment_eq_open)

lemma *closed_segment_eq_real_ivl1*:
fixes $a \ b :: \text{real}$
assumes $a \leq b$
shows $\text{closed_segment } a \ b = \{a .. b\}$
proof safe
fix x
assume $x \in \text{closed_segment } a \ b$
then obtain u **where** $0 \leq u \leq 1$ **and** $x_def: x = (1 - u) * a + u * b$
by (auto simp: closed_segment_def)
have $u * a \leq u * b$ $(1 - u) * a \leq (1 - u) * b$
by (auto intro!: mult_left_mono u assms)
then show $x \in \{a .. b\}$
unfolding x_def **by** (auto simp: algebra_simps)
next
show $\bigwedge x. x \in \{a..b\} \implies x \in \text{closed_segment } a \ b$
by (force simp: closed_segment_def divide_simps algebra_simps
intro: exI[**where** $x=(x - a) / (b - a)$ **for** x])
qed

lemma *closed_segment_eq_real_ivl*:
fixes $a \ b :: \text{real}$
shows $\text{closed_segment } a \ b = (\text{if } a \leq b \text{ then } \{a .. b\} \text{ else } \{b .. a\})$
by (metis closed_segment_commute closed_segment_eq_real_ivl1 nle_le)

lemma *open_segment_eq_real_ivl*:
fixes $a \ b :: \text{real}$
shows $\text{open_segment } a \ b = (\text{if } a \leq b \text{ then } \{a < .. < b\} \text{ else } \{b < .. < a\})$
by (auto simp: closed_segment_eq_real_ivl open_segment_def split: if_split_asm)

lemma *closed_segment_real_eq*:
fixes $u :: \text{real}$ **shows** $\text{closed_segment } u \ v = (\lambda x. (v - u) * x + u) \text{ ` } \{0..1\}$
by (simp add: closed_segment_eq_real_ivl image_affinity_atLeastAtMost)

lemma *closed_segment_same_Re*:
assumes $\text{Re } a = \text{Re } b$

shows $\text{closed_segment } a \ b = \{z. \text{Re } z = \text{Re } a \wedge \text{Im } z \in \text{closed_segment } (\text{Im } a) (\text{Im } b)\}$

proof safe

fix z **assume** $z \in \text{closed_segment } a \ b$

then obtain u **where** $u: u \in \{0..1\} \ z = a + \text{of_real } u * (b - a)$

by $(\text{auto simp: closed_segment_def scaleR_conv_of_real algebra_simps})$

from assms **show** $\text{Re } z = \text{Re } a$ **by** $(\text{auto simp: } u)$

from $u(1)$ **show** $\text{Im } z \in \text{closed_segment } (\text{Im } a) (\text{Im } b)$

by $(\text{force simp: } u \ \text{closed_segment_def algebra_simps})$

next

fix z **assume** $[\text{simp}]: \text{Re } z = \text{Re } a$ **and** $\text{Im } z \in \text{closed_segment } (\text{Im } a) (\text{Im } b)$

then obtain u **where** $u: u \in \{0..1\} \ \text{Im } z = \text{Im } a + \text{of_real } u * (\text{Im } b - \text{Im } a)$

by $(\text{auto simp: closed_segment_def scaleR_conv_of_real algebra_simps})$

from $u(1)$ **show** $z \in \text{closed_segment } a \ b$ **using** assms

by $(\text{force simp: } u \ \text{closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff})$

qed

lemma $\text{closed_segment_same_Im}$:

assumes $\text{Im } a = \text{Im } b$

shows $\text{closed_segment } a \ b = \{z. \text{Im } z = \text{Im } a \wedge \text{Re } z \in \text{closed_segment } (\text{Re } a) (\text{Re } b)\}$

proof safe

fix z **assume** $z \in \text{closed_segment } a \ b$

then obtain u **where** $u: u \in \{0..1\} \ z = a + \text{of_real } u * (b - a)$

by $(\text{auto simp: closed_segment_def scaleR_conv_of_real algebra_simps})$

from assms **show** $\text{Im } z = \text{Im } a$ **by** $(\text{auto simp: } u)$

from $u(1)$ **show** $\text{Re } z \in \text{closed_segment } (\text{Re } a) (\text{Re } b)$

by $(\text{force simp: } u \ \text{closed_segment_def algebra_simps})$

next

fix z **assume** $[\text{simp}]: \text{Im } z = \text{Im } a$ **and** $\text{Re } z \in \text{closed_segment } (\text{Re } a) (\text{Re } b)$

then obtain u **where** $u: u \in \{0..1\} \ \text{Re } z = \text{Re } a + \text{of_real } u * (\text{Re } b - \text{Re } a)$

by $(\text{auto simp: closed_segment_def scaleR_conv_of_real algebra_simps})$

from $u(1)$ **show** $z \in \text{closed_segment } a \ b$ **using** assms

by $(\text{force simp: } u \ \text{closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff})$

qed

lemma $\text{dist_in_closed_segment}$:

fixes $a :: 'a :: \text{euclidean_space}$

assumes $x \in \text{closed_segment } a \ b$

shows $\text{dist } x \ a \leq \text{dist } a \ b \wedge \text{dist } x \ b \leq \text{dist } a \ b$

by $(\text{metis assms dist_commute dist_norm segment_bound}(2) \ \text{segment_bound}1)$

lemma $\text{dist_in_open_segment}$:

fixes $a :: 'a :: \text{euclidean_space}$

assumes $x \in \text{open_segment } a \ b$

shows $\text{dist } x \ a < \text{dist } a \ b \wedge \text{dist } x \ b < \text{dist } a \ b$

by $(\text{metis assms dist_commute dist_norm open_segment_bound}1 \ \text{open_segment_commute})$


```

lemma dist_decreases_open_segment_0:
  fixes  $x :: 'a :: euclidean\_space$ 
  assumes  $x \in \text{open\_segment } 0\ b$ 
  shows  $\text{dist } c\ x < \text{dist } c\ 0 \vee \text{dist } c\ x < \text{dist } c\ b$ 
proof (rule ccontr, clarsimp simp: not_less)
  obtain  $u$  where  $u: 0 \neq b \ 0 < u \ u < 1$  and  $x: x = u *_{\mathbb{R}} b$ 
  using assms by (auto simp: in_segment)
  have  $xb: x \cdot b < b \cdot b$ 
  using  $u\ x$  by auto
  assume  $\text{norm } c \leq \text{dist } c\ x$ 
  then have  $c \cdot c \leq (c - x) \cdot (c - x)$ 
  by (simp add: dist_norm norm_le)
  moreover have  $0 < x \cdot b$ 
  using  $u\ x$  by auto
  ultimately have less:  $c \cdot b < x \cdot b$ 
  by (simp add: x algebra_simps inner_commute u)
  assume  $\text{dist } c\ b \leq \text{dist } c\ x$ 
  then have  $(c - b) \cdot (c - b) \leq (c - x) \cdot (c - x)$ 
  by (simp add: dist_norm norm_le)
  then have  $(b \cdot b) * (1 - u * u) \leq 2 * (b \cdot c) * (1 - u)$ 
  by (simp add: x algebra_simps inner_commute)
  then have  $(1 + u) * (b \cdot b) * (1 - u) \leq 2 * (b \cdot c) * (1 - u)$ 
  by (simp add: algebra_simps)
  then have  $(1 + u) * (b \cdot b) \leq 2 * (b \cdot c)$ 
  using  $\langle u < 1 \rangle$  by auto
  with  $xb$  have  $c \cdot b \geq x \cdot b$ 
  by (auto simp: x algebra_simps inner_commute)
  with less show False by auto
qed

```

```

proposition dist_decreases_open_segment:
  fixes  $a :: 'a :: euclidean\_space$ 
  assumes  $x \in \text{open\_segment } a\ b$ 
  shows  $\text{dist } c\ x < \text{dist } c\ a \vee \text{dist } c\ x < \text{dist } c\ b$ 
proof -
  have  $*$ :  $x - a \in \text{open\_segment } 0\ (b - a)$  using assms
  by (metis diff_self open_segment_translation_eq uminus_add_conv_diff)
  show ?thesis
  using dist_decreases_open_segment_0 [OF  $*$ , of c-a] assms
  by (simp add: dist_norm)
qed

```

```

corollary open_segment_furthest_le:
  fixes  $a\ b\ x\ y :: 'a :: euclidean\_space$ 
  assumes  $x \in \text{open\_segment } a\ b$ 
  shows  $\text{norm } (y - x) < \text{norm } (y - a) \vee \text{norm } (y - x) < \text{norm } (y - b)$ 
  by (metis assms dist_decreases_open_segment dist_norm)

```

corollary *dist_decreases_closed_segment*:

fixes $a :: 'a :: \text{euclidean_space}$
assumes $x \in \text{closed_segment } a \ b$
shows $\text{dist } c \ x \leq \text{dist } c \ a \ \vee \ \text{dist } c \ x \leq \text{dist } c \ b$
by (*smt (verit, ccfv_threshold) Un_iff assms closed_segment_eq_open dist_norm empty_iff insertE open_segment_furthest_le*)

corollary *segment_furthest_le*:

fixes $a \ b \ x \ y :: 'a :: \text{euclidean_space}$
assumes $x \in \text{closed_segment } a \ b$
shows $\text{norm } (y - x) \leq \text{norm } (y - a) \ \vee \ \text{norm } (y - x) \leq \text{norm } (y - b)$
by (*metis assms dist_decreases_closed_segment dist_norm*)

lemma *convex_intermediate_ball*:

fixes $a :: 'a :: \text{euclidean_space}$
shows $\llbracket \text{ball } a \ r \subseteq T; T \subseteq \text{cball } a \ r \rrbracket \implies \text{convex } T$
by (*smt (verit) convex_contains_open_segment dist_decreases_open_segment mem_ball mem_cball subset_eq*)

lemma *csegment_midpoint_subset*: $\text{closed_segment } (\text{midpoint } a \ b) \ b \subseteq \text{closed_segment } a \ b$

apply (*clarsimp simp: midpoint_def in_segment*)
apply (*rule_tac x=(1 + u) / 2 in exI*)
apply (*simp add: algebra_simps add_divide_distrib diff_divide_distrib*)
by (*metis field_sum_of_halves scaleR_left.add*)

lemma *notin_segment_midpoint*:

fixes $a :: 'a :: \text{euclidean_space}$
shows $a \neq b \implies a \notin \text{closed_segment } (\text{midpoint } a \ b) \ b$
by (*auto simp: dist_midpoint dest!: dist_in_closed_segment*)

More lemmas, especially for working with the underlying formula

lemma *segment_eq_compose*:

fixes $a :: 'a :: \text{real_vector}$
shows $(\lambda u. (1 - u) *_R a + u *_R b) = (\lambda x. a + x) \circ (\lambda u. u *_R (b - a))$
by (*simp add: o_def algebra_simps*)

lemma *segment_degen_1*:

fixes $a :: 'a :: \text{real_vector}$
shows $(1 - u) *_R a + u *_R b = b \iff a = b \vee u = 1$
by (*smt (verit, best) add_right_cancel scaleR_cancel_left scaleR_collapse*)

lemma *segment_degen_0*:

fixes $a :: 'a :: \text{real_vector}$
shows $(1 - u) *_R a + u *_R b = a \iff a = b \vee u = 0$
using *segment_degen_1 [of 1-u b a]* **by** (*auto simp: algebra_simps*)

lemma *add_scaleR_degen*:

```

fixes  $a\ b :: 'a :: \text{real\_vector}$ 
assumes  $(u *_R b + v *_R a) = (u *_R a + v *_R b)$   $u \neq v$ 
shows  $a=b$ 
by (smt (verit) add_diff_cancel_left' add_diff_eq assms scaleR_cancel_left
scaleR_left.diff)

```

```

lemma closed_segment_image_interval:
  closed_segment  $a\ b = (\lambda u. (1 - u) *_R a + u *_R b)$  ' $\{0..1\}$ '
by (auto simp: set_eq_iff image_iff closed_segment_def)

```

```

lemma open_segment_image_interval:
  open_segment  $a\ b = (\text{if } a=b \text{ then } \{\} \text{ else } (\lambda u. (1 - u) *_R a + u *_R b))$  ' $\{0 <..<1\}$ '
by (auto simp: open_segment_def closed_segment_def segment_degen_0 segment_degen_1)

```

```

lemmas segment_image_interval = closed_segment_image_interval open_segment_image_interval

```

```

lemma closed_segment_neq_empty [simp]: closed_segment  $a\ b \neq \{\}$ 
by auto

```

```

lemma open_segment_eq_empty [simp]: open_segment  $a\ b = \{\} \longleftrightarrow a = b$ 
by (simp add: segment_image_interval(2))

```

```

lemma open_segment_eq_empty' [simp]:  $\{\} = \text{open\_segment } a\ b \longleftrightarrow a = b$ 
using open_segment_eq_empty by blast

```

```

lemmas segment_eq_empty = closed_segment_neq_empty open_segment_eq_empty

```

```

lemma inj_segment:
fixes  $a :: 'a :: \text{real\_vector}$ 
assumes  $a \neq b$ 
shows inj_on  $(\lambda u. (1 - u) *_R a + u *_R b)$   $I$ 

```

```

proof
fix  $x\ y$ 
assume  $(1 - x) *_R a + x *_R b = (1 - y) *_R a + y *_R b$ 
then have  $x *_R (b - a) = y *_R (b - a)$ 
by (simp add: algebra_simps)
with assms show  $x = y$ 
by (simp add: real_vector.scale_right_imp_eq)

```

```

qed

```

```

lemma finite_closed_segment [simp]: finite(closed_segment  $a\ b$ )  $\longleftrightarrow a = b$ 
using infinite_Icc [OF zero_less_one] finite_imageD [OF inj_segment [of  $a\ b$ ]]
unfolding segment_image_interval
by (smt (verit, del_insts) finite.emptyI finite_insert finite_subset image_subset_iff
insertCI segment_degen_0)

```

lemma *finite_open_segment* [*simp*]: $\text{finite}(\text{open_segment } a \ b) \longleftrightarrow a = b$
by (*auto simp: open_segment_def*)

lemmas *finite_segment* = *finite_closed_segment finite_open_segment*

lemma *closed_segment_eq_sing*: $\text{closed_segment } a \ b = \{c\} \longleftrightarrow a = c \wedge b = c$
by *auto*

lemma *open_segment_eq_sing*: $\text{open_segment } a \ b \neq \{c\}$
by (*metis finite_insert finite_open_segment insert_not_empty open_segment_image_interval*)

lemmas *segment_eq_sing* = *closed_segment_eq_sing open_segment_eq_sing*

lemma *compact_segment* [*simp*]:
fixes $a :: 'a::\text{real_normed_vector}$
shows *compact* (*closed_segment* $a \ b$)
by (*auto simp: segment_image_interval intro!: compact_continuous_image continuous_intros*)

lemma *closed_segment* [*simp*]:
fixes $a :: 'a::\text{real_normed_vector}$
shows *closed* (*closed_segment* $a \ b$)
by (*simp add: compact_imp_closed*)

lemma *closure_closed_segment* [*simp*]:
fixes $a :: 'a::\text{real_normed_vector}$
shows $\text{closure}(\text{closed_segment } a \ b) = \text{closed_segment } a \ b$
by *simp*

lemma *open_segment_bound*:
assumes $x \in \text{open_segment } a \ b$
shows $\text{norm } (x - a) < \text{norm } (b - a) \ \text{norm } (x - b) < \text{norm } (b - a)$
by (*metis assms norm_minus_commute open_segment_bound1 open_segment_commute*)+

lemma *closure_open_segment* [*simp*]:
 $\text{closure}(\text{open_segment } a \ b) = (\text{if } a = b \text{ then } \{\} \text{ else } \text{closed_segment } a \ b)$
for $a :: 'a::\text{euclidean_space}$
proof (*cases a = b*)
case *True*
then show *?thesis*
by *simp*
next
case *False*
have $\text{closure}((\lambda u. u *_R (b - a)) \ ' \{0 <.. < 1\}) = (\lambda u. u *_R (b - a)) \ ' \text{closure } \{0 <.. < 1\}$
proof (*rule closure_injective_linear_image [symmetric]*)
qed (*use False in <auto intro!: injI>*)
then have *closure*
 $((\lambda u. (1 - u) *_R a + u *_R b) \ ' \{0 <.. < 1\}) = (\lambda x. (1 - x) *_R a + x *_R b) \ ' \{0 <.. < 1\}$

```

closure {0<..\lambda x. x *_R b - x *_R a) ' $\{0<..)]
  by (simp add: segment_eq_compose field_simps scaleR_diff_left scaleR_diff_right
image_image)
  then show ?thesis
  by (simp add: segment_image_interval closure_greaterThanLessThan [symmetric])
del: closure_greaterThanLessThan
qed$ 
```

```

lemma closed_open_segment_iff [simp]:
  fixes a :: 'a::euclidean_space shows closed(open_segment a b)  $\longleftrightarrow$  a = b
  by (metis open_segment_def DiffE closure_eq closure_open_segment ends_in_segment(1)
insert_iff segment_image_interval(2))

```

```

lemma compact_open_segment_iff [simp]:
  fixes a :: 'a::euclidean_space shows compact(open_segment a b)  $\longleftrightarrow$  a = b
  by (simp add: bounded_open_segment compact_eq_bounded_closed)

```

```

lemma convex_closed_segment [iff]: convex (closed_segment a b)
  unfolding segment_convex_hull by(rule convex_convex_hull)

```

```

lemma convex_open_segment [iff]: convex (open_segment a b)
proof -
  have convex (( $\lambda u. u *_R (b - a)$ ) ' $\{0<..)
  by (rule convex_linear_image) auto
  then have convex ((+) a ' $(\lambda u. u *_R (b - a))$ ' ' $\{0<..)
  by (rule convex_translation)
  then show ?thesis
  by (simp add: image_image open_segment_image_interval segment_eq_compose
field_simps scaleR_diff_left scaleR_diff_right)
qed$$ 
```

```

lemmas convex_segment = convex_closed_segment convex_open_segment

```

```

lemma subset_closed_segment:
  closed_segment a b  $\subseteq$  closed_segment c d  $\longleftrightarrow$ 
  a  $\in$  closed_segment c d  $\wedge$  b  $\in$  closed_segment c d
  using closed_segment_subset convex_closed_segment ends_in_segment in_mono
  by blast

```

```

lemma subset_co_segment:
  closed_segment a b  $\subseteq$  open_segment c d  $\longleftrightarrow$ 
  a  $\in$  open_segment c d  $\wedge$  b  $\in$  open_segment c d
  using closed_segment_subset by blast

```

```

lemma subset_open_segment:
  fixes a :: 'a::euclidean_space
  shows open_segment a b  $\subseteq$  open_segment c d  $\longleftrightarrow$ 
  a = b  $\vee$  a  $\in$  closed_segment c d  $\wedge$  b  $\in$  closed_segment c d

```

```

      (is ?lhs = ?rhs)
proof (cases a = b)
  case True then show ?thesis by simp
next
  case False show ?thesis
  proof
    assume rhs: ?rhs
    with ⟨a ≠ b⟩ have c ≠ d
      using closed_segment_idem_singleton_iff by auto
    have ∃ uc. (1 - u) *R ((1 - ua) *R c + ua *R d) + u *R ((1 - ub) *R c +
ub *R d) =
      (1 - uc) *R c + uc *R d ∧ 0 < uc ∧ uc < 1
    if neq: (1 - ua) *R c + ua *R d ≠ (1 - ub) *R c + ub *R d c ≠ d
      and a = (1 - ua) *R c + ua *R d b = (1 - ub) *R c + ub *R d
      and u: 0 < u u < 1 and uab: 0 ≤ ua ua ≤ 1 0 ≤ ub ub ≤ 1
    for u ua ub
  proof -
    have ua ≠ ub
      using neq by auto
    moreover have (u - 1) * ua ≤ 0 using u uab
      by (simp add: mult_nonpos_nonneg)
    ultimately have lt: (u - 1) * ua < u * ub using u uab
      by (metis antisym_conv diff_ge_0_iff_ge le_less_trans mult_eq_0_iff
mult_le_0_iff not_less)
    have p * ua + q * ub < p + q if p: 0 < p and q: 0 < q for p q
  proof -
    have ¬ p ≤ 0 ∧ ¬ q ≤ 0
      using p q not_less by blast+
    then show ?thesis
      by (smt (verit) ⟨ua ≠ ub⟩ mult_cancel_left1 mult_left_le uab(2) uab(4))
  qed
    then have (1 - u) * ua + u * ub < 1 using u ⟨ua ≠ ub⟩
      by (metis diff_add_cancel diff_gt_0_iff_gt)
    with lt show ?thesis
      by (rule_tac x=ua + u*(ub-ua) in exI) (simp add: algebra_simps)
  qed
  with rhs ⟨a ≠ b⟩ ⟨c ≠ d⟩ show ?lhs
    unfolding open_segment_image_interval closed_segment_def
      by (fastforce simp add:)
next
  assume lhs: ?lhs
  with ⟨a ≠ b⟩ have c ≠ d
    by (meson finite_open_segment rev_finite_subset)
  have closure (open_segment a b) ⊆ closure (open_segment c d)
    using lhs closure_mono by blast
  then have closed_segment a b ⊆ closed_segment c d
    by (simp add: ⟨a ≠ b⟩ ⟨c ≠ d⟩)
  then show ?rhs
    by (force simp: ⟨a ≠ b⟩)

```

qed
qed

lemma *closed_segment_same_fst*:

$fst\ a = fst\ b \implies closed_segment\ a\ b = \{fst\ a\} \times closed_segment\ (snd\ a)\ (snd\ b)$
by (*auto simp: closed_segment_def scaleR_prod_def*)

lemma *closed_segment_same_snd*:

$snd\ a = snd\ b \implies closed_segment\ a\ b = closed_segment\ (fst\ a)\ (fst\ b) \times \{snd\ a\}$
by (*auto simp: closed_segment_def scaleR_prod_def*)

lemma *subset_oc_segment*:

fixes $a :: 'a::euclidean_space$

shows $open_segment\ a\ b \subseteq closed_segment\ c\ d \iff$

$a = b \vee a \in closed_segment\ c\ d \wedge b \in closed_segment\ c\ d$

(**is** $?lhs = ?rhs$)

proof

show $?lhs \implies ?rhs$

by (*metis closure_closed_segment closure_mono closure_open_segment subset_closed_segment*)

show $?rhs \implies ?lhs$

by (*meson dual_order.trans segment_open_subset_closed subset_open_segment*)

qed

lemmas *subset_segment = subset_closed_segment subset_co_segment subset_oc_segment subset_open_segment*

lemma *dist_half_times2*:

fixes $a :: 'a :: real_normed_vector$

shows $dist\ ((1 / 2) *_R\ (a + b))\ x * 2 = dist\ (a+b)\ (2 *_R\ x)$

proof -

have $norm\ ((1 / 2) *_R\ (a + b) - x) * 2 = norm\ (2 *_R\ ((1 / 2) *_R\ (a + b) - x))$

by *simp*

also have $\dots = norm\ ((a + b) - 2 *_R\ x)$

by (*simp add: real_vector.scale_right_diff_distrib*)

finally show $?thesis$

by (*simp only: dist_norm*)

qed

lemma *closed_segment_as_ball*:

$closed_segment\ a\ b = affine\ hull\ \{a,b\} \cap cball\ (inverse\ 2 *_R\ (a + b))\ (norm\ (b - a) / 2)$

proof (*cases* $b = a$)

case *True* **then show** $?thesis$ **by** (*auto simp: hull_inc*)

next

case *False*

then have $*$: $((\exists u\ v.\ x = u *_R\ a + v *_R\ b \wedge u + v = 1) \wedge$

$$\begin{aligned} & \text{dist } ((1 / 2) *_R (a + b)) x * 2 \leq \text{norm } (b - a) = \\ & (\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 \leq u \wedge u \leq 1) \text{ for } x \end{aligned}$$

proof –

have $((\exists u v. x = u *_R a + v *_R b \wedge u + v = 1) \wedge$
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 \leq \text{norm } (b - a)) =$
 $((\exists u. x = (1 - u) *_R a + u *_R b) \wedge$
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 \leq \text{norm } (b - a))$
unfolding $\text{eq_diff_eq [symmetric] by simp}$
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge$
 $\text{norm } ((a+b) - (2 *_R x)) \leq \text{norm } (b - a))$
by $(\text{simp add: dist_half_times2}) (\text{simp add: dist_norm})$
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge$
 $\text{norm } ((a+b) - (2 *_R ((1 - u) *_R a + u *_R b))) \leq \text{norm } (b - a))$
by auto
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge$
 $\text{norm } ((1 - u * 2) *_R (b - a)) \leq \text{norm } (b - a))$
by $(\text{simp add: algebra_simps scaleR_2})$
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge$
 $|1 - u * 2| * \text{norm } (b - a) \leq \text{norm } (b - a))$
by simp
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge |1 - u * 2| \leq 1)$
by $(\text{simp add: mult_le_cancel_right2 False})$
also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 \leq u \wedge u \leq 1)$
by auto
finally show $?thesis .$

qed

show $?thesis$

by $(\text{simp add: affine_hull_2 Set.set_eq_iff closed_segment_def } *)$

qed

lemma $\text{open_segment_as_ball}$:

$$\begin{aligned} & \text{open_segment } a \ b = \\ & \text{affine hull } \{a, b\} \cap \text{ball}(\text{inverse } 2 *_R (a + b))(\text{norm}(b - a) / 2) \end{aligned}$$

proof $(\text{cases } b = a)$

case True **then show** $?thesis$ **by** $(\text{auto simp: hull_inc})$

next

case False

then have $*$: $((\exists u v. x = u *_R a + v *_R b \wedge u + v = 1) \wedge$
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a)) =$
 $(\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 < u \wedge u < 1) \text{ for } x$

proof –

have $((\exists u v. x = u *_R a + v *_R b \wedge u + v = 1) \wedge$
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a)) =$
 $((\exists u. x = (1 - u) *_R a + u *_R b) \wedge$
 $\text{dist } ((1 / 2) *_R (a + b)) x * 2 < \text{norm } (b - a))$

unfolding $\text{eq_diff_eq [symmetric] by simp}$

also have $\dots = (\exists u. x = (1 - u) *_R a + u *_R b \wedge$
 $\text{norm } ((a+b) - (2 *_R x)) < \text{norm } (b - a))$
by $(\text{simp add: dist_half_times2}) (\text{simp add: dist_norm})$


```

also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
   $norm ((a+b) - (2 *_R ((1 - u) *_R a + u *_R b))) < norm (b - a)$ )
  by auto
also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
   $norm ((1 - u * 2) *_R (b - a)) < norm (b - a)$ )
  by (simp add: algebra_simps scaleR_2)
also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge$ 
   $|1 - u * 2| * norm (b - a) < norm (b - a)$ )
  by simp
also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge |1 - u * 2| < 1$ )
  by (simp add: mult_le_cancel_right2 False)
also have ... = ( $\exists u. x = (1 - u) *_R a + u *_R b \wedge 0 < u \wedge u < 1$ )
  by auto
finally show ?thesis .
qed
show ?thesis
  using False by (force simp: affine_hull_2 Set.set_eq_iff open_segment_image_interval
*)
qed

```

lemmas *segment_as_ball = closed_segment_as_ball open_segment_as_ball*

lemma *connected_segment [iff]:*
fixes $x :: 'a :: real_normed_vector$
shows *connected (closed_segment x y)*
by (simp add: convex_connected)

lemma *is_interval_closed_segment_1 [intro, simp]: is_interval (closed_segment a b) for a b::real*
unfolding *closed_segment_eq_real_ivl*
by (auto simp: is_interval_def)

lemma *IVT'_closed_segment_real:*
fixes $f :: real \Rightarrow real$
assumes $y \in \text{closed_segment } (f a) (f b)$
assumes *continuous_on (closed_segment a b) f*
shows $\exists x \in \text{closed_segment } a b. f x = y$
using *IVT'[of f a y b]*
IVT'[of -f a -y b]
IVT'[of f b y a]
IVT'[of -f b -y a] assms
by (cases a ≤ b; cases f b ≥ f a) (auto simp: closed_segment_eq_real_ivl continuous_on_minus)

5.2.4 Betweenness

definition *between = ($\lambda(a,b) x. x \in \text{closed_segment } a b$)*

lemma *betweenI:*

assumes $0 \leq u \ u \leq 1 \ x = (1 - u) *_R a + u *_R b$
shows *between* (a, b) x
using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *betweenE*:

assumes *between* (a, b) x
obtains u **where** $0 \leq u \ u \leq 1 \ x = (1 - u) *_R a + u *_R b$
using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *between_implies_scaled_diff*:

assumes *between* (S, T) X *between* (S, T) Y $S \neq Y$
obtains c **where** $(X - Y) = c *_R (S - Y)$

proof -

from $\langle \textit{between} (S, T) X \rangle$ **obtain** u_X **where** X: $X = u_X *_R S + (1 - u_X) *_R T$

by (*metis add.commute betweenE eq_diff_eq*)

from $\langle \textit{between} (S, T) Y \rangle$ **obtain** u_Y **where** Y: $Y = u_Y *_R S + (1 - u_Y) *_R T$

by (*metis add.commute betweenE eq_diff_eq*)

have $X - Y = (u_X - u_Y) *_R (S - T)$

by (*simp add: X Y scaleR_left.diff scaleR_right_diff_distrib*)

moreover from Y **have** $S - Y = (1 - u_Y) *_R (S - T)$

by (*simp add: real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib*)

moreover note $\langle S \neq Y \rangle$

ultimately have $(X - Y) = ((u_X - u_Y) / (1 - u_Y)) *_R (S - Y)$ **by** *auto*

from this that show thesis by *blast*

qed

lemma *between_mem_segment*: *between* (a,b) x $\longleftrightarrow x \in \textit{closed_segment} a b$

unfolding *between_def* **by** *auto*

lemma *between*: *between* (a, b) (x::'a::euclidean_space) $\longleftrightarrow \textit{dist} a b = (\textit{dist} a x) + (\textit{dist} x b)$

proof (*cases a = b*)

case *True*

then show *?thesis*

by (*auto simp add: between_def dist_commute*)

next

case *False*

then have *Fal*: $\textit{norm} (a - b) \neq 0$ **and** *Fal2*: $\textit{norm} (a - b) > 0$

by *auto*

have *: $\bigwedge u. a - ((1 - u) *_R a + u *_R b) = u *_R (a - b)$

by (*auto simp add: algebra_simps*)

have $\textit{norm} (a - x) *_R (x - b) = \textit{norm} (x - b) *_R (a - x)$ **if** $x = (1 - u) *_R a + u *_R b$ $0 \leq u \ u \leq 1$ **for** u

proof -

have *: $a - x = u *_R (a - b) \ x - b = (1 - u) *_R (a - b)$

unfolding *that(1)* **by** (*auto simp add: algebra_simps*)

show $\textit{norm} (a - x) *_R (x - b) = \textit{norm} (x - b) *_R (a - x)$

```

    unfolding norm_minus_commute[of x a] * using ‹0 ≤ u› ‹u ≤ 1›
    by simp
  qed
  moreover have ∃ u. x = (1 - u) *R a + u *R b ∧ 0 ≤ u ∧ u ≤ 1 if dist a b
= dist a x + dist x b
  proof -
    let ?β = norm (a - x) / norm (a - b)
    show ∃ u. x = (1 - u) *R a + u *R b ∧ 0 ≤ u ∧ u ≤ 1
    proof (intro exI conjI)
      show ?β ≤ 1
        using Fal2 unfolding that[unfolded dist_norm] norm_ge_zero by auto
      show x = (1 - ?β) *R a + (?β) *R b
      proof (subst euclidean_eq_iff; intro ballI)
        fix i :: 'a
        assume i: i ∈ Basis
        have ((1 - ?β) *R a + (?β) *R b) · i
          = ((norm (a - b) - norm (a - x)) * (a · i) + norm (a - x) * (b ·
i)) / norm (a - b)
          using Fal by (auto simp add: field_simps inner_simps)
        also have ... = x · i
        apply (rule divide_eq_imp[OF Fal])
        unfolding that[unfolded dist_norm]
        using that[unfolded dist_triangle_eq] i
        apply (subst (asm) euclidean_eq_iff)
        apply (auto simp add: field_simps inner_simps)
        done
      finally show x · i = ((1 - ?β) *R a + (?β) *R b) · i
        by auto
      qed
    qed
  qed (use Fal2 in auto)
  qed
  ultimately show ?thesis
  by (force simp add: between_def closed_segment_def dist_triangle_eq)
qed

```

lemma between_midpoint:

```

  fixes a :: 'a::euclidean_space
  shows between (a,b) (midpoint a b) (is ?t1)
  and between (b,a) (midpoint a b) (is ?t2)
  proof -
    have *: ∧ x y z. x = (1/2::real) *R z ⇒ y = (1/2) *R z ⇒ norm z = norm
x + norm y
    by auto
    show ?t1 ?t2
    unfolding between_midpoint_def dist_norm
    by (auto simp add: field_simps inner_simps euclidean_eq_iff[where 'a='a]
intro!: *)
  qed

```

lemma *between_mem_convex_hull*:

between (a,b) x \longleftrightarrow x \in convex_hull {a,b}

unfolding *between_mem_segment segment_convex_hull* ..

lemma *between_triv_iff* [simp]: *between* (a,a) b \longleftrightarrow a=b

by (auto simp: *between_def*)

lemma *between_triv1* [simp]: *between* (a,b) a

by (auto simp: *between_def*)

lemma *between_triv2* [simp]: *between* (a,b) b

by (auto simp: *between_def*)

lemma *between_commute*:

between (a,b) = *between* (b,a)

by (auto simp: *between_def closed_segment_commute*)

lemma *between_antisym*:

fixes a :: 'a :: euclidean_space

shows \llbracket *between* (b,c) a; *between* (a,c) b $\rrbracket \implies a = b$

by (auto simp: *between_dist_commute*)

lemma *between_trans*:

fixes a :: 'a :: euclidean_space

shows \llbracket *between* (b,c) a; *between* (a,c) d $\rrbracket \implies$ *between* (b,c) d

using *dist_triangle2* [of b c d] *dist_triangle3* [of b d a]

by (auto simp: *between_dist_commute*)

lemma *between_norm*:

fixes a :: 'a :: euclidean_space

shows *between* (a,b) x \longleftrightarrow norm(x - a) *_R (b - x) = norm(b - x) *_R (x - a)

by (auto simp: *between_dist_triangle_eq norm_minus_commute algebra_simps*)

lemma *between_swap*:

fixes A B X Y :: 'a::euclidean_space

assumes *between* (A, B) X

assumes *between* (A, B) Y

shows *between* (X, B) Y \longleftrightarrow *between* (A, Y) X

using *assms* **by** (auto simp add: *between*)

lemma *between_translation* [simp]: *between* (a + y, a + z) (a + x) \longleftrightarrow *between* (y,z) x

by (auto simp: *between_def*)

lemma *between_trans_2*:

fixes a :: 'a :: euclidean_space

shows \llbracket *between* (b,c) a; *between* (a,b) d $\rrbracket \implies$ *between* (c,d) a

by (*metis* *between_commute between_swap between_trans*)

```

lemma between_scaleR_lift [simp]:
  fixes  $v :: 'a::euclidean\_space$ 
  shows  $between (a *_R v, b *_R v) (c *_R v) \longleftrightarrow v = 0 \vee between (a, b) c$ 
  by (simp add: between dist_norm flip: scaleR_left_diff_distrib distrib_right)

```

```

lemma between_1:
  fixes  $x::real$ 
  shows  $between (a,b) x \longleftrightarrow (a \leq x \wedge x \leq b) \vee (b \leq x \wedge x \leq a)$ 
  by (auto simp: between_mem_segment closed_segment_eq_real_ivl)

```

end

5.3 Convex Sets and Functions on (Normed) Euclidean Spaces

```

theory Convex_Euclidean_Space
imports
  Convex Topology_Euclidean_Space Line_Segment
begin

```

5.3.1 Topological Properties of Convex Sets and Functions

```

lemma aff_dim_cball:
  fixes  $a :: 'n::euclidean\_space$ 
  assumes  $e > 0$ 
  shows  $aff\_dim (cball a e) = int (DIM('n))$ 
proof -
  have  $(\lambda x. a + x) ` (cball 0 e) \subseteq cball a e$ 
  unfolding cball_def dist_norm by auto
  then have  $aff\_dim (cball (0 :: 'n::euclidean\_space) e) \leq aff\_dim (cball a e)$ 
  using aff_dim_translation_eq[of  $a$   $cball 0 e$ ]
  aff_dim_subset[of  $(+)$   $a$   $cball 0 e$   $cball a e$ ]
  by auto
  moreover have  $aff\_dim (cball (0 :: 'n::euclidean\_space) e) = int (DIM('n))$ 
  using hull_inc[of  $(0 :: 'n::euclidean\_space)$   $cball 0 e$ ]
  centre_in_cball[of  $(0 :: 'n::euclidean\_space)$ ] assms
  by (simp add: dim_cball[of  $e$ ] aff_dim_zero[of  $cball 0 e$ ])
  ultimately show ?thesis
  using aff_dim_le_DIM[of  $cball a e$ ] by auto
qed

```

```

lemma aff_dim_open:
  fixes  $S :: 'n::euclidean\_space$  set
  assumes open S
  and  $S \neq \{\}$ 
  shows  $aff\_dim S = int (DIM('n))$ 
proof -

```

```

obtain  $x$  where  $x \in S$ 
using assms by auto
then obtain  $e$  where  $e: e > 0$   $cball\ x\ e \subseteq S$ 
using open_contains_cball[of  $S$ ] assms by auto
then have  $aff\_dim\ (cball\ x\ e) \leq aff\_dim\ S$ 
using aff_dim_subset by auto
with  $e$  show ?thesis
using aff_dim_cball[of  $e\ x$ ] aff_dim_le_DIM[of  $S$ ] by auto
qed

```

```

lemma low_dim_interior:
fixes  $S :: 'n::euclidean\_space\ set$ 
assumes  $\neg aff\_dim\ S = int\ (DIM('n))$ 
shows  $interior\ S = \{\}$ 
proof -
have  $aff\_dim(interior\ S) \leq aff\_dim\ S$ 
using interior_subset aff_dim_subset[of  $interior\ S\ S$ ] by auto
then show ?thesis
using aff_dim_open[of  $interior\ S$ ] aff_dim_le_DIM[of  $S$ ] assms by auto
qed

```

```

corollary empty_interior_lowdim:
fixes  $S :: 'n::euclidean\_space\ set$ 
shows  $dim\ S < DIM\ ('n) \implies interior\ S = \{\}$ 
by (metis low_dim_interior affine_hull_UNIV dim_affine_hull less_not_refl dim_UNIV)

```

```

corollary aff_dim_nonempty_interior:
fixes  $S :: 'a::euclidean\_space\ set$ 
shows  $interior\ S \neq \{\} \implies aff\_dim\ S = DIM('a)$ 
by (metis low_dim_interior)

```

5.3.2 Relative interior of a set

```

definition rel_interior  $S =$ 
 $\{x. \exists T. openin\ (top\_of\_set\ (affine\ hull\ S))\ T \wedge x \in T \wedge T \subseteq S\}$ 

```

```

lemma rel_interior_mono:
 $\llbracket S \subseteq T; affine\ hull\ S = affine\ hull\ T \rrbracket$ 
 $\implies (rel\_interior\ S) \subseteq (rel\_interior\ T)$ 
by (auto simp: rel_interior_def)

```

```

lemma rel_interior_maximal:
 $\llbracket T \subseteq S; openin(top\_of\_set\ (affine\ hull\ S))\ T \rrbracket \implies T \subseteq (rel\_interior\ S)$ 
by (auto simp: rel_interior_def)

```

```

lemma rel_interior:  $rel\_interior\ S = \{x \in S. \exists T. open\ T \wedge x \in T \wedge T \cap affine\ hull\ S \subseteq S\}$ 
(is ?lhs = ?rhs)
proof

```

```

show ?lhs  $\subseteq$  ?rhs
  by (force simp add: rel_interior_def openin_open)
  { fix x T
    assume *:  $x \in S$  open T  $x \in T$   $T \cap \text{affine hull } S \subseteq S$ 
    then have **:  $x \in T \cap \text{affine hull } S$ 
      using hull_inc by auto
    with * have  $\exists T_b. (\exists T_a. \text{open } T_a \wedge T_b = \text{affine hull } S \cap T_a) \wedge x \in T_b \wedge T_b \subseteq S$ 
      by (rule_tac x =  $T \cap (\text{affine hull } S)$  in exI) auto
    }
  then show ?rhs  $\subseteq$  ?lhs
    by (force simp add: rel_interior_def openin_open)
qed

```

```

lemma mem_rel_interior:  $x \in \text{rel\_interior } S \iff (\exists T. \text{open } T \wedge x \in T \cap S \wedge T \cap \text{affine hull } S \subseteq S)$ 
  by (auto simp: rel_interior)

```

```

lemma mem_rel_interior_ball:
   $x \in \text{rel\_interior } S \iff x \in S \wedge (\exists e. e > 0 \wedge \text{ball } x \ e \cap \text{affine hull } S \subseteq S)$ 
  (is ?lhs = ?rhs)

```

proof

```

  assume ?rhs then show ?lhs
    by (simp add: rel_interior) (meson Elementary_Metric_Spaces.open_ball centre_in_ball)
qed (force simp: rel_interior open_contains_ball)

```

```

lemma rel_interior_ball:
   $\text{rel\_interior } S = \{x \in S. \exists e. e > 0 \wedge \text{ball } x \ e \cap \text{affine hull } S \subseteq S\}$ 
  using mem_rel_interior_ball [of _ S] by auto

```

```

lemma mem_rel_interior_cball:
   $x \in \text{rel\_interior } S \iff x \in S \wedge (\exists e. e > 0 \wedge \text{cball } x \ e \cap \text{affine hull } S \subseteq S)$ 
  (is ?lhs = ?rhs)

```

proof

```

  assume ?rhs then obtain e where  $x \in S$   $e > 0$   $\text{cball } x \ e \cap \text{affine hull } S \subseteq S$ 
    by (auto simp: rel_interior)
  then have  $\text{ball } x \ e \cap \text{affine hull } S \subseteq S$ 
    by auto
  then show ?lhs
    using  $\langle 0 < e \rangle \langle x \in S \rangle$  rel_interior_ball by auto
qed (force simp: rel_interior open_contains_cball)

```

```

lemma rel_interior_cball:
   $\text{rel\_interior } S = \{x \in S. \exists e. e > 0 \wedge \text{cball } x \ e \cap \text{affine hull } S \subseteq S\}$ 
  using mem_rel_interior_cball [of _ S] by auto

```

```

lemma rel_interior_empty [simp]:  $\text{rel\_interior } \{\} = \{\}$ 
  by (auto simp: rel_interior_def)

```

lemma *affine_hull_sing* [simp]: $\text{affine hull } \{a :: 'n::\text{euclidean_space}\} = \{a\}$
by (*metis affine_hull_eq affine_sing*)

lemma *rel_interior_sing* [simp]:
fixes $a :: 'n::\text{euclidean_space}$ **shows** $\text{rel_interior } \{a\} = \{a\}$
proof –
have $\exists x::\text{real}. 0 < x$
using *zero_less_one* **by** *blast*
then show *?thesis*
by (*auto simp: rel_interior_ball*)
qed

lemma *subset_rel_interior*:
fixes $S T :: 'n::\text{euclidean_space}$ *set*
assumes $S \subseteq T$
and $\text{affine hull } S = \text{affine hull } T$
shows $\text{rel_interior } S \subseteq \text{rel_interior } T$
using *assms* **by** (*auto simp: rel_interior_def*)

lemma *rel_interior_subset*: $\text{rel_interior } S \subseteq S$
by (*auto simp: rel_interior_def*)

lemma *rel_interior_subset_closure*: $\text{rel_interior } S \subseteq \text{closure } S$
using *rel_interior_subset* **by** (*auto simp: closure_def*)

lemma *interior_subset_rel_interior*: $\text{interior } S \subseteq \text{rel_interior } S$
by (*auto simp: rel_interior interior_def*)

lemma *interior_rel_interior*:
fixes $S :: 'n::\text{euclidean_space}$ *set*
assumes $\text{aff_dim } S = \text{int}(\text{DIM}('n))$
shows $\text{rel_interior } S = \text{interior } S$
proof –
have $\text{affine hull } S = \text{UNIV}$
using *assms affine_hull_UNIV[of S]* **by** *auto*
then show *?thesis*
unfolding *rel_interior interior_def* **by** *auto*
qed

lemma *rel_interior_interior*:
fixes $S :: 'n::\text{euclidean_space}$ *set*
assumes $\text{affine hull } S = \text{UNIV}$
shows $\text{rel_interior } S = \text{interior } S$
using *assms* **unfolding** *rel_interior interior_def* **by** *auto*

lemma *rel_interior_open*:
fixes $S :: 'n::\text{euclidean_space}$ *set*
assumes *open S*

shows $rel_interior\ S = S$
by (metis assms interior_eq interior_subset_rel_interior rel_interior_subset set_eq_subset)

lemma interior_rel_interior_gen:
fixes $S :: 'n::euclidean_space\ set$
shows $interior\ S = (if\ aff_dim\ S = int(DIM('n))\ then\ rel_interior\ S\ else\ \{\})$
by (metis interior_rel_interior low_dim_interior)

lemma rel_interior_nonempty_interior:
fixes $S :: 'n::euclidean_space\ set$
shows $interior\ S \neq \{\} \implies rel_interior\ S = interior\ S$
by (metis interior_rel_interior_gen)

lemma affine_hull_nonempty_interior:
fixes $S :: 'n::euclidean_space\ set$
shows $interior\ S \neq \{\} \implies affine\ hull\ S = UNIV$
by (metis affine_hull_UNIV interior_rel_interior_gen)

lemma rel_interior_affine_hull [simp]:
fixes $S :: 'n::euclidean_space\ set$
shows $rel_interior\ (affine\ hull\ S) = affine\ hull\ S$
proof –
have $*$: $rel_interior\ (affine\ hull\ S) \subseteq affine\ hull\ S$
using rel_interior_subset **by** auto
{
fix x
assume $x: x \in affine\ hull\ S$
define $e :: real$ **where** $e = 1$
then have $e > 0$ $ball\ x\ e \cap affine\ hull\ (affine\ hull\ S) \subseteq affine\ hull\ S$
using hull_hull[of S] **by** auto
then have $x \in rel_interior\ (affine\ hull\ S)$
using $x\ rel_interior_ball$ [of $affine\ hull\ S$] **by** auto
}
then show $?thesis$ **using** $*$ **by** auto
qed

lemma rel_interior_UNIV [simp]: $rel_interior\ (UNIV :: ('n::euclidean_space)\ set) = UNIV$
by (metis open_UNIV rel_interior_open)

lemma rel_interior_convex_shrink:
fixes $S :: 'a::euclidean_space\ set$
assumes convex S
and $c \in rel_interior\ S$
and $x \in S$
and $0 < e$
and $e \leq 1$
shows $x - e *_R (x - c) \in rel_interior\ S$

proof –

```

obtain  $d$  where  $d > 0$  and  $d: \text{ball } c \ d \cap \text{affine hull } S \subseteq S$ 
  using  $\text{assms}(2)$  unfolding  $\text{mem\_rel\_interior\_ball}$  by  $\text{auto}$ 
  {
    fix  $y$ 
    assume  $as: \text{dist } (x - e *_{\mathbb{R}} (x - c)) \ y < e * d \ y \in \text{affine hull } S$ 
    have  $*$ :  $y = (1 - (1 - e)) *_{\mathbb{R}} ((1 / e) *_{\mathbb{R}} y - ((1 - e) / e) *_{\mathbb{R}} x) + (1 - e) *_{\mathbb{R}} x$ 
    using  $\langle e > 0 \rangle$  by ( $\text{auto simp: scaleR\_left\_diff\_distrib scaleR\_right\_diff\_distrib}$ )
    have  $x \in \text{affine hull } S$ 
    using  $\text{assms hull\_subset}[of \ S]$  by  $\text{auto}$ 
    moreover have  $1 / e + -((1 - e) / e) = 1$ 
    using  $\langle e > 0 \rangle \text{left\_diff\_distrib}[of \ 1 \ (1 - e) \ 1 / e]$  by  $\text{auto}$ 
    ultimately have  $**$ :  $(1 / e) *_{\mathbb{R}} y - ((1 - e) / e) *_{\mathbb{R}} x \in \text{affine hull } S$ 
    using  $as \ \text{affine\_affine\_hull}[of \ S] \ \text{mem\_affine}[of \ \text{affine hull } S \ y \ x \ (1 / e) - ((1 - e) / e)]$ 
    by ( $\text{simp add: algebra\_simps}$ )
    have  $c - ((1 / e) *_{\mathbb{R}} y - ((1 - e) / e) *_{\mathbb{R}} x) = (1 / e) *_{\mathbb{R}} (e *_{\mathbb{R}} c - y + (1 - e) *_{\mathbb{R}} x)$ 
    using  $\langle e > 0 \rangle$ 
    by ( $\text{auto simp: euclidean\_eq\_iff}[where \ 'a='a] \ \text{field\_simps inner\_simps}$ )
    then have  $\text{dist } c \ ((1 / e) *_{\mathbb{R}} y - ((1 - e) / e) *_{\mathbb{R}} x) = |1/e| * \text{norm } (e *_{\mathbb{R}} c - y + (1 - e) *_{\mathbb{R}} x)$ 
    unfolding  $\text{dist\_norm norm\_scaleR}[symmetric]$  by  $\text{auto}$ 
    also have  $\dots = |1/e| * \text{norm } (x - e *_{\mathbb{R}} (x - c) - y)$ 
    by ( $\text{auto intro!: arg\_cong}[where \ f=\text{norm}] \ \text{simp add: algebra\_simps}$ )
    also have  $\dots < d$ 
    using  $as[\text{unfolded dist\_norm}]$  and  $\langle e > 0 \rangle$ 
    by ( $\text{auto simp: pos\_divide\_less\_eq}[OF \ \langle e > 0 \rangle] \ \text{mult.commute}$ )
    finally have  $(1 / e) *_{\mathbb{R}} y - ((1 - e) / e) *_{\mathbb{R}} x \in S$ 
    using  $** \ d$  by  $\text{auto}$ 
    then have  $y \in S$ 
    using  $* \ \text{convexD} \ [OF \ \langle \text{convex } S \rangle] \ \text{assms}(3-5)$ 
    by ( $\text{metis diff\_add\_cancel diff\_ge\_0\_iff\_ge le\_add\_same\_cancel1 less\_eq\_real\_def}$ )
  }
then have  $\text{ball } (x - e *_{\mathbb{R}} (x - c)) \ (e * d) \cap \text{affine hull } S \subseteq S$ 
  by  $\text{auto}$ 
moreover have  $e * d > 0$ 
  using  $\langle e > 0 \rangle \ \langle d > 0 \rangle$  by  $\text{simp}$ 
moreover have  $c: c \in S$ 
  using  $\text{assms rel\_interior\_subset}$  by  $\text{auto}$ 
moreover from  $c$  have  $x - e *_{\mathbb{R}} (x - c) \in S$ 
  using  $\text{convexD\_alt}[of \ S \ x \ c \ e] \ \text{assms}$ 
  by ( $\text{metis diff\_add\_eq diff\_diff\_eq2 less\_eq\_real\_def scaleR\_diff\_left scaleR\_one scale\_right\_diff\_distrib}$ )
ultimately show  $?thesis$ 
  using  $\text{mem\_rel\_interior\_ball}[of \ x - e *_{\mathbb{R}} (x - c) \ S] \ \langle e > 0 \rangle$  by  $\text{auto}$ 
qed

```

```

lemma interior_real_atLeast [simp]:
  fixes a :: real
  shows interior {a..} = {a<..}
proof -
  {
    fix y
    have ball_y (y - a)  $\subseteq$  {a..}
      by (auto simp: dist_norm)
    moreover assume a < y
    ultimately have y  $\in$  interior {a..}
      by (force simp add: mem_interior)
  }
  moreover
  {
    fix y
    assume y  $\in$  interior {a..}
    then obtain e where e: e > 0 cball y e  $\subseteq$  {a..}
      using mem_interior_cball[of y {a..}] by auto
    moreover from e have y - e  $\in$  cball y e
      by (auto simp: cball_def dist_norm)
    ultimately have a  $\leq$  y - e by blast
    then have a < y using e by auto
  }
  ultimately show ?thesis by auto
qed

```

```

lemma continuous_ge_on_Ioo:
  assumes continuous_on {c..d} g  $\wedge$  x. x  $\in$  {c<.. $<$ d}  $\implies$  g x  $\geq$  a c < d x  $\in$ 
  {c..d}
  shows g (x::real)  $\geq$  (a::real)
proof -
  from assms(3) have {c..d} = closure {c<.. $<$ d} by (rule closure_greaterThanLessThan[symmetric])
  also from assms(2) have {c<.. $<$ d}  $\subseteq$  (g -' {a..}  $\cap$  {c..d}) by auto
  hence closure {c<.. $<$ d}  $\subseteq$  closure (g -' {a..}  $\cap$  {c..d}) by (rule closure_mono)
  also from assms(1) have closed (g -' {a..}  $\cap$  {c..d})
    by (auto simp: continuous_on_closed_vimage)
  hence closure (g -' {a..}  $\cap$  {c..d}) = g -' {a..}  $\cap$  {c..d} by simp
  finally show ?thesis using  $\langle$ x  $\in$  {c..d} $\rangle$  by auto
qed

```

```

lemma interior_real_atMost [simp]:
  fixes a :: real
  shows interior {..a} = {.. $<$ a}
proof -
  {
    fix y
    have ball_y (a - y)  $\subseteq$  {..a}
      by (auto simp: dist_norm)
    moreover assume a > y
  }

```

```

    ultimately have  $y \in \text{interior } \{..a\}$ 
      by (force simp add: mem_interior)
  }
  moreover
  {
    fix  $y$ 
    assume  $y \in \text{interior } \{..a\}$ 
    then obtain  $e$  where  $e: e > 0$   $\text{cball } y \ e \subseteq \{..a\}$ 
      using mem_interior_cball[of  $y \ \{..a\}$ ] by auto
    moreover from  $e$  have  $y + e \in \text{cball } y \ e$ 
      by (auto simp: cball_def dist_norm)
    ultimately have  $a \geq y + e$  by auto
    then have  $a > y$  using  $e$  by auto
  }
  ultimately show ?thesis by auto
qed

```

```

lemma interior_atLeastAtMost_real [simp]: interior  $\{a..b\} = \{a<..b :: real\}$ 
proof -
  have  $\{a..b\} = \{a..\} \cap \{..b\}$  by auto
  also have interior  $\dots = \{a<..\} \cap \{..<b\}$ 
    by (simp)
  also have  $\dots = \{a<..b\}$  by auto
  finally show ?thesis .
qed

```

```

lemma interior_atLeastLessThan [simp]:
  fixes  $a::real$  shows interior  $\{a..<b\} = \{a<..b\}$ 
  by (metis atLeastLessThan_def greaterThanLessThan_def interior_atLeastAtMost_real
    interior_Int interior_interior interior_real_atLeast)

```

```

lemma interior_lessThanAtMost [simp]:
  fixes  $a::real$  shows interior  $\{a<..b\} = \{a<..b\}$ 
  by (metis atLeastAtMost_def greaterThanAtMost_def interior_atLeastAtMost_real
    interior_Int
      interior_interior interior_real_atLeast)

```

```

lemma interior_greaterThanLessThan_real [simp]: interior  $\{a<..b\} = \{a<..b
:: real\}$ 
  by (metis interior_atLeastAtMost_real interior_interior)

```

```

lemma frontier_real_atMost [simp]:
  fixes  $a :: real$ 
  shows frontier  $\{..a\} = \{a\}$ 
  unfolding frontier_def by auto

```

```

lemma frontier_real_atLeast [simp]: frontier  $\{a..\} = \{a::real\}$ 
  by (auto simp: frontier_def)

```

lemma *frontier_real_greaterThan* [simp]: $\text{frontier } \{a < ..\} = \{a :: \text{real}\}$
by (auto simp: interior_open frontier_def)

lemma *frontier_real_lessThan* [simp]: $\text{frontier } \{.. < a\} = \{a :: \text{real}\}$
by (auto simp: interior_open frontier_def)

lemma *rel_interior_real_box* [simp]:
fixes $a\ b :: \text{real}$
assumes $a < b$
shows $\text{rel_interior } \{a .. b\} = \{a < .. < b\}$
proof –
have $\text{box } a\ b \neq \{\}$
using *assms*
unfolding *set_eq_iff*
by (auto intro!: *exI*[of $_ (a + b) / 2$] simp: *box_def*)
then show *?thesis*
using *interior_rel_interior_gen*[of *cbox* $a\ b$, *symmetric*]
by (simp split: *if_split_asm* del: *box_real* add: *box_real*[*symmetric*])
qed

lemma *rel_interior_real_semiline* [simp]:
fixes $a :: \text{real}$
shows $\text{rel_interior } \{a ..\} = \{a < ..\}$
proof –
have $\{a < ..\} \neq \{\}$
unfolding *set_eq_iff* **by** (auto intro!: *exI*[of $_ a + 1$])
then show *?thesis* **using** *interior_real_atLeast* *interior_rel_interior_gen*[of $\{a ..\}$]
by (auto split: *if_split_asm*)
qed

Relative open sets

definition $\text{rel_open } S \iff \text{rel_interior } S = S$

lemma *rel_open*: $\text{rel_open } S \iff \text{openin } (\text{top_of_set } (\text{affine hull } S))\ S$ (**is** *?lhs* = *?rhs*)

proof
assume *?lhs*
then show *?rhs*
unfolding *rel_open_def* *rel_interior_def*
using *openin_subopen*[of *top_of_set* (*affine hull* S) S] **by** auto
qed (auto simp: *rel_open_def* *rel_interior_def*)

lemma *openin_rel_interior*: $\text{openin } (\text{top_of_set } (\text{affine hull } S))\ (\text{rel_interior } S)$
using *openin_subopen* **by** (fastforce simp add: *rel_interior_def*)

lemma *openin_set_rel_interior*:
 $\text{openin } (\text{top_of_set } S)\ (\text{rel_interior } S)$

by (rule *openin_subset_trans* [OF *openin_rel_interior rel_interior_subset hull_subset*])

lemma *affine_rel_open*:
fixes $S :: 'n::\text{euclidean_space set}$
assumes *affine S*
shows *rel_open S*
unfolding *rel_open_def*
using *assms rel_interior_affine_hull[of S] affine_hull_eq[of S]*
by *metis*

lemma *affine_closed*:
fixes $S :: 'n::\text{euclidean_space set}$
assumes *affine S*
shows *closed S*
proof –
 {
 assume $S \neq \{\}$
 then obtain L **where** $L: \text{subspace } L \text{ affine_parallel } S \ L$
 using *assms affine_parallel_subspace[of S]* **by** *auto*
 then obtain a **where** $a: S = ((+) a ' L)$
 using *affine_parallel_def[of L S] affine_parallel_commute* **by** *auto*
 from L **have** *closed L* **using** *closed_subspace* **by** *auto*
 then have *closed S*
 using *closed_translation a* **by** *auto*
 }
then show *?thesis* **by** *auto*
qed

lemma *closure_affine_hull*:
fixes $S :: 'n::\text{euclidean_space set}$
shows $\text{closure } S \subseteq \text{affine hull } S$
by (*intro closure_minimal hull_subset affine_closed affine_affine_hull*)

lemma *closed_affine_hull [iff]*:
fixes $S :: 'n::\text{euclidean_space set}$
shows $\text{closed } (\text{affine hull } S)$
by (*metis affine_affine_hull affine_closed*)

lemma *closure_same_affine_hull [simp]*:
fixes $S :: 'n::\text{euclidean_space set}$
shows $\text{affine hull } (\text{closure } S) = \text{affine hull } S$
proof –
have $\text{affine hull } (\text{closure } S) \subseteq \text{affine hull } S$
 using *hull_mono[of closure S affine hull S affine]*
 closure_affine_hull[of S] hull_hull[of affine S]
 by *auto*
moreover have $\text{affine hull } (\text{closure } S) \supseteq \text{affine hull } S$
 using *hull_mono[of S closure S affine] closure_subset* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed

```
lemma closure_aff_dim [simp]:
  fixes S :: 'n::euclidean_space set
  shows aff_dim (closure S) = aff_dim S
proof -
  have aff_dim S ≤ aff_dim (closure S)
    using aff_dim_subset closure_subset by auto
  moreover have aff_dim (closure S) ≤ aff_dim (affine hull S)
    using aff_dim_subset closure_affine_hull by blast
  moreover have aff_dim (affine hull S) = aff_dim S
    using aff_dim_affine_hull by auto
  ultimately show ?thesis by auto
```

qed

```
lemma rel_interior_closure_convex_shrink:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  and c ∈ rel_interior S
  and x ∈ closure S
  and e > 0
  and e ≤ 1
  shows x - e *R (x - c) ∈ rel_interior S
proof -
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding mem_rel_interior_ball by auto
  have ∃ y ∈ S. norm (y - x) * (1 - e) < e * d
  proof (cases x ∈ S)
    case True
    then show ?thesis using ‹e > 0› ‹d > 0› by force
  next
    case False
    then have x: x islimpt S
      using assms(3)[unfolded closure_def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ S y ≠ x dist y x < 1
        using x[unfolded islimpt_approachable, THEN spec[where x=1]] by auto
      then show ?thesis
        unfolding True using ‹d > 0› by (force simp add: )
    next
      case False
      then have 0 < e * d / (1 - e) and *: 1 - e > 0
        using ‹e ≤ 1› ‹e > 0› ‹d > 0› by auto
      then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 - e)
        using x[unfolded islimpt_approachable, THEN spec[where x=e*d / (1 -
e)]] by auto
      then show ?thesis
```

```

      unfolding dist_norm using pos_less_divide_eq[OF *] by force
    qed
  qed
  then obtain y where y ∈ S and y: norm (y - x) * (1 - e) < e * d
    by auto
  define z where z = c + ((1 - e) / e) *R (x - y)
  have *: x - e *R (x - c) = y - e *R (y - z)
    unfolding z_def using ⟨e > 0⟩
  by (auto simp: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)
  have zball: z ∈ ball c d
    using mem_ball z_def dist_norm[of c]
    using y and assms(4,5)
  by (simp add: norm_minus_commute) (simp add: field_simps)
  have x ∈ affine hull S
    using closure_affine_hull assms by auto
  moreover have y ∈ affine hull S
    using ⟨y ∈ S⟩ hull_subset[of S] by auto
  moreover have c ∈ affine hull S
    using assms rel_interior_subset hull_subset[of S] by auto
  ultimately have z ∈ affine hull S
    using z_def affine_affine_hull[of S]
    mem_affine_3_minus [of affine hull S c x y (1 - e) / e]
    assms
  by simp
  then have z ∈ S using d zball by auto
  obtain d1 where d1 > 0 and d1: ball z d1 ≤ ball c d
    using zball open_ball[of c d] openE[of ball c d z] by auto
  then have ball z d1 ∩ affine hull S ⊆ ball c d ∩ affine hull S
    by auto
  then have ball z d1 ∩ affine hull S ⊆ S
    using d by auto
  then have z ∈ rel_interior S
    using mem_rel_interior_ball using ⟨d1 > 0⟩ ⟨z ∈ S⟩ by auto
  then have y - e *R (y - z) ∈ rel_interior S
    using rel_interior_convex_shrink[of S z y e] assms ⟨y ∈ S⟩ by auto
  then show ?thesis using * by auto
qed

```

lemma rel_interior_eq:

```

  rel_interior s = s ↔ openin(top_of_set (affine hull s)) s
using rel_open rel_open_def by blast

```

lemma rel_interior_openin:

```

  openin(top_of_set (affine hull s)) s ⇒ rel_interior s = s
by (simp add: rel_interior_eq)

```

lemma rel_interior_affine:

```

  fixes S :: 'n::euclidean_space set
  shows affine S ⇒ rel_interior S = S

```



```

using affine_rel_open rel_open_def by auto

lemma rel_interior_eq_closure:
  fixes S :: 'n::euclidean_space set
  shows rel_interior S = closure S  $\longleftrightarrow$  affine S
proof (cases S = {})
  case True
  then show ?thesis
    by auto
next
  case False show ?thesis
  proof
    assume eq: rel_interior S = closure S
    have openin (top_of_set (affine hull S)) S
      by (metis eq closure_subset openin_rel_interior rel_interior_subset subset_antisym)
    moreover have closedin (top_of_set (affine hull S)) S
      by (metis closed_subset closure_subset_eq eq hull_subset rel_interior_subset)
    ultimately have S = {}  $\vee$  S = affine hull S
      using convex_connected connected_clopen convex_affine_hull by metis
    with False have affine hull S = S
      by auto
    then show affine S
      by (metis affine_hull_eq)
  next
    assume affine S
    then show rel_interior S = closure S
      by (simp add: rel_interior_affine affine_closed)
  qed
qed

```

Relative interior preserves under linear transformations

```

lemma rel_interior_translation_aux:
  fixes a :: 'n::euclidean_space
  shows (( $\lambda x. a + x$ ) ' rel_interior S)  $\subseteq$  rel_interior (( $\lambda x. a + x$ ) ' S)
proof -
  {
    fix x
    assume x: x  $\in$  rel_interior S
    then obtain T where open T x  $\in$  T  $\cap$  S T  $\cap$  affine hull S  $\subseteq$  S
      using mem_rel_interior[of x S] by auto
    then have open (( $\lambda x. a + x$ ) ' T)
      and a + x  $\in$  (( $\lambda x. a + x$ ) ' T)  $\cap$  (( $\lambda x. a + x$ ) ' S)
      and (( $\lambda x. a + x$ ) ' T)  $\cap$  affine hull (( $\lambda x. a + x$ ) ' S)  $\subseteq$  (( $\lambda x. a + x$ ) ' S)
      using affine_hull_translation[of a S] open_translation[of T a] x by auto
    then have a + x  $\in$  rel_interior (( $\lambda x. a + x$ ) ' S)
      using mem_rel_interior[of a+x (( $\lambda x. a + x$ ) ' S)] by auto
  }

```

then show *?thesis* by *auto*
qed

lemma *rel_interior_translation*:

fixes $a :: 'n::\text{euclidean_space}$

shows $\text{rel_interior } ((\lambda x. a + x) ' S) = (\lambda x. a + x) ' \text{rel_interior } S$

proof –

have $(\lambda x. (-a) + x) ' \text{rel_interior } ((\lambda x. a + x) ' S) \subseteq \text{rel_interior } S$

using *rel_interior_translation_aux*[of $-a$ $(\lambda x. a + x) ' S$]
translation_assoc[of $-a$ a]

by *auto*

then have $(\lambda x. a + x) ' \text{rel_interior } S \supseteq \text{rel_interior } ((\lambda x. a + x) ' S)$

using *translation_inverse_subset*[of a $\text{rel_interior } ((+) a ' S)$ $\text{rel_interior } S$]

by *auto*

then show *?thesis*

using *rel_interior_translation_aux*[of a S] by *auto*

qed

lemma *affine_hull_linear_image*:

assumes *bounded_linear* f

shows $f ' (\text{affine hull } s) = \text{affine hull } f ' s$

proof –

interpret f : *bounded_linear* f by *fact*

have *affine* $\{x. f x \in \text{affine hull } f ' s\}$

unfolding *affine_def*

by (*auto simp: f.scaleR f.add affine_affine_hull*[unfolding *affine_def*, *rule_format*])

moreover have *affine* $\{x. x \in f ' (\text{affine hull } s)\}$

using *affine_affine_hull*[unfolding *affine_def*, of s]

unfolding *affine_def* by (*auto simp: f.scaleR* [*symmetric*] *f.add* [*symmetric*])

ultimately show *?thesis*

by (*auto simp: hull_inc elim!: hull_induct*)

qed

lemma *rel_interior_injective_on_span_linear_image*:

fixes $f :: 'm::\text{euclidean_space} \Rightarrow 'n::\text{euclidean_space}$

and $S :: 'm::\text{euclidean_space}$ *set*

assumes *bounded_linear* f

and *inj_on* f (*span* S)

shows $\text{rel_interior } (f ' S) = f ' (\text{rel_interior } S)$

proof –

{

fix z

assume $z: z \in \text{rel_interior } (f ' S)$

then have $z \in f ' S$

using *rel_interior_subset*[of $f ' S$] by *auto*

then obtain x where $x: x \in S$ $f x = z$ by *auto*

obtain $e2$ where $e2: e2 > 0$ $\text{cball } z e2 \cap \text{affine hull } (f ' S) \subseteq (f ' S)$

```

    using z rel_interior_cball[of f ' S] by auto
  obtain K where K: K > 0  $\wedge$  x. norm (f x)  $\leq$  norm x * K
    using assms Real_Vector_Spaces.bounded_linear.pos_bounded[of f] by auto
  define e1 where e1 = 1 / K
  then have e1: e1 > 0  $\wedge$  x. e1 * norm (f x)  $\leq$  norm x
    using K pos_le_divide_eq[of e1] by auto
  define e where e = e1 * e2
  then have e > 0 using e1 e2 by auto
  {
    fix y
    assume y: y  $\in$  cball x e  $\cap$  affine hull S
    then have h1: f y  $\in$  affine hull (f ' S)
      using affine_hull_linear_image[of f S] assms by auto
    from y have norm (x-y)  $\leq$  e1 * e2
      using cball_def[of x e] dist_norm[of x y] e_def by auto
    moreover have f x - f y = f (x - y)
      using assms linear_diff[of f x y] linear_conv_bounded_linear[of f] by auto
    moreover have e1 * norm (f (x-y))  $\leq$  norm (x - y)
      using e1 by auto
    ultimately have e1 * norm ((f x)-(f y))  $\leq$  e1 * e2
      by auto
    then have f y  $\in$  cball z e2
      using cball_def[of f x e2] dist_norm[of f x f y] e1 x by auto
    then have f y  $\in$  f ' S
      using y e2 h1 by auto
    then have y  $\in$  S
      using assms y hull_subset[of S] affine_hull_subset_span
        inj_on_image_mem_iff [OF <inj_on f (span S)>]
        by (metis Int_iff span_superset subsetCE)
  }
  then have z  $\in$  f ' (rel_interior S)
    using mem_rel_interior_cball[of x S] <e > 0> x by auto
}
moreover
{
  fix x
  assume x: x  $\in$  rel_interior S
  then obtain e2 where e2: e2 > 0 cball x e2  $\cap$  affine hull S  $\subseteq$  S
    using rel_interior_cball[of S] by auto
  have x  $\in$  S using x rel_interior_subset by auto
  then have *: f x  $\in$  f ' S by auto
  have  $\forall x \in \text{span } S. f x = 0 \longrightarrow x = 0$ 
    using assms subspace_span linear_conv_bounded_linear[of f]
      linear_injective_on_subspace_0[of f span S]
    by auto
  then obtain e1 where e1: e1 > 0  $\forall x \in \text{span } S. e1 * \text{norm } x \leq \text{norm } (f x)$ 
    using assms injective_imp_isometric[of span S f]
      subspace_span[of S] closed_subspace[of span S]
    by auto

```

```

define e where e = e1 * e2
hence e > 0 using e1 e2 by auto
{
  fix y
  assume y: y ∈ cball (f x) e ∩ affine hull (f ' S)
  then have y ∈ f ' (affine hull S)
    using affine_hull_linear_image[of f S] assms by auto
  then obtain xy where xy: xy ∈ affine hull S f xy = y by auto
  with y have norm (f x - f xy) ≤ e1 * e2
    using cball_def[of f x e] dist_norm[of f x y] e_def by auto
  moreover have f x - f xy = f (x - xy)
    using assms linear_diff[of f x xy] linear_conv_bounded_linear[of f] by auto
  moreover have *: x - xy ∈ span S
    using subspace_diff[of span S x xy] subspace_span ⟨x ∈ S⟩ xy
      affine_hull_subset_span[of S] span_superset
    by auto
  moreover from * have e1 * norm (x - xy) ≤ norm (f (x - xy))
    using e1 by auto
  ultimately have e1 * norm (x - xy) ≤ e1 * e2
    by auto
  then have xy ∈ cball x e2
    using cball_def[of x e2] dist_norm[of x xy] e1 by auto
  then have y ∈ f ' S
    using xy e2 by auto
}
then have f x ∈ rel_interior (f ' S)
  using mem_rel_interior_cball[of (f x) (f ' S)] * ⟨e > 0⟩ by auto
}
ultimately show ?thesis by auto
qed

```

```

lemma rel_interior_injective_linear_image:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes bounded_linear f
  and inj f
  shows rel_interior (f ' S) = f ' (rel_interior S)
  using assms rel_interior_injective_on_span_linear_image[of f S]
    subset_inj_on[of f UNIV span S]
  by auto

```

5.3.3 Openness and compactness are preserved by convex hull operation

```

lemma open_convex_hull[intro]:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (convex hull S)
proof (clarsimp simp: open_contains_cball convex_hull_explicit)
  fix T and u :: 'a ⇒ real

```

```

assume obt: finite T T ⊆ S ∀ x ∈ T. 0 ≤ u x sum u T = 1

from assms[unfolded open_contains_cball] obtain b
  where b: ∧ x. x ∈ S ⇒ 0 < b x ∧ cball x (b x) ⊆ S by metis
have b' : T ≠ {}
  using obt by auto
define i where i = b' T
let ?Φ = λ y. ∃ F. finite F ∧ F ⊆ S ∧ (∃ u. (∀ x ∈ F. 0 ≤ u x) ∧ sum u F = 1 ∧
(∑ v ∈ F. u v *R v) = y)
let ?a = ∑ v ∈ T. u v *R v
show ∃ e > 0. cball ?a e ⊆ {y. ?Φ y}
proof (intro exI subsetI conjI)
  show 0 < Min i
    unfolding i_def and Min_gr_iff[OF finite_imageI[OF obt(1)]] <b' T ≠ {}>
    using b' <T ⊆ S> by auto
next
fix y
assume y ∈ cball ?a (Min i)
then have y: norm (?a - y) ≤ Min i
  unfolding dist_norm[symmetric] by auto
{ fix x
  assume x ∈ T
  then have Min i ≤ b x
    by (simp add: i_def obt(1))
  then have x + (y - ?a) ∈ cball x (b x)
    using y unfolding mem_cball dist_norm by auto
  moreover have x ∈ S
    using <x ∈ T> <T ⊆ S> by auto
  ultimately have x + (y - ?a) ∈ S
    using y b by blast
}
moreover
have *: inj_on (λ v. v + (y - ?a)) T
  unfolding inj_on_def by auto
have (∑ v ∈ (λ v. v + (y - ?a))' T. u (v - (y - ?a)) *R v) = y
  unfolding sum.reindex[OF *] o_def using obt(4)
  by (simp add: sum.distrib sum_subtractf scaleR_left.sum[symmetric] scaleR_right_distrib)
ultimately show y ∈ {y. ?Φ y}
proof (intro CollectI exI conjI)
  show finite ((λ v. v + (y - ?a))' T)
    by (simp add: obt(1))
  show sum (λ v. u (v - (y - ?a))) ((λ v. v + (y - ?a))' T) = 1
    unfolding sum.reindex[OF *] o_def using obt(4) by auto
qed (use obt(1, 3) in auto)
qed
qed

lemma compact_convex_combinations:
  fixes S T :: 'a::real_normed_vector set

```

```

assumes compact S compact T
shows compact { (1 - u) *R x + u *R y | x y u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ S ∧ y
∈ T }
proof -
  let ?X = {0..1} × S × T
  let ?h = (λz. (1 - fst z) *R fst (snd z) + fst z *R snd (snd z))
  have *: { (1 - u) *R x + u *R y | x y u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ S ∧ y ∈ T } =
  ?h ` ?X
    by force
  have continuous_on ?X (λz. (1 - fst z) *R fst (snd z) + fst z *R snd (snd z))
    unfolding continuous_on by (rule ballI) (intro tendsto_intros)
  with assms show ?thesis
    by (simp add: * compact_Times compact_continuous_image)
qed

```

```

lemma finite_imp_compact_convex_hull:
  fixes S :: 'a::real_normed_vector set
  assumes finite S
  shows compact (convex hull S)
proof (cases S = {})
  case True
    then show ?thesis by simp
  next
    case False
      with assms show ?thesis
      proof (induct rule: finite_ne_induct)
        case (singleton x)
          show ?case by simp
      next
        case (insert x A)
          let ?f = λ(u, y::'a). u *R x + (1 - u) *R y
          let ?T = {0..1::real} × (convex hull A)
          have continuous_on ?T ?f
            unfolding split_def continuous_on by (intro ballI tendsto_intros)
          moreover have compact ?T
            by (intro compact_Times compact_Icc insert)
          ultimately have compact (?f ` ?T)
            by (rule compact_continuous_image)
          also have ?f ` ?T = convex hull (insert x A)
            unfolding convex_hull_insert [OF ‹A ≠ {}›]
            apply safe
            apply (rule_tac x=a in exI, simp)
            apply (rule_tac x=1 - a in exI, simp, fast)
            apply (rule_tac x=(u, b) in image_eqI, simp_all)
            done
          finally show compact (convex hull (insert x A)) .
      qed
    qed

```

```

lemma compact_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes compact S
  shows compact (convex hull S)
proof (cases S = {})
  case True
  then show ?thesis using compact_empty by simp
next
  case False
  then obtain w where w ∈ S by auto
  show ?thesis
    unfolding caratheodory[of S]
  proof (induct (DIM('a) + 1))
    case 0
    have *: {x. ∃ sa. finite sa ∧ sa ⊆ S ∧ card sa ≤ 0 ∧ x ∈ convex hull sa} = {}
      using compact_empty by auto
    from 0 show ?case unfolding * by simp
  next
    case (Suc n)
    show ?case
    proof (cases n = 0)
      case True
      have {x. ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T} = S
        unfolding set_eq_iff and mem_Collect_eq
      proof (rule, rule)
        fix x
        assume ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T
        then obtain T where T: finite T T ⊆ S card T ≤ Suc n x ∈ convex hull
          T
          by auto
        show x ∈ S
        proof (cases card T = 0)
          case True
          then show ?thesis
            using T(4) unfolding card_0_eq[OF T(1)] by simp
        next
          case False
          then have card T = Suc 0 using T(3) ⟨n=0⟩ by auto
          then obtain a where T = {a} unfolding card_Suc_eq by auto
          then show ?thesis using T(2,4) by simp
        qed
      qed
    next
      fix x assume x ∈ S
      then show ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T
        by (rule_tac x={x} in exI) (use convex_hull_singleton in auto)
      qed
    then show ?thesis using assms by simp
  next
    case False

```

```

have { $x. \exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ } =
  {( $1 - u$ ) *R  $x + u$  *R  $y \mid x y u.$ 
    $0 \leq u \wedge u \leq 1 \wedge x \in S \wedge y \in \{x. \exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n$ 
 $\wedge x \in \text{convex hull } T\}}$ }
unfolding set_eq_iff and mem_Collect_eq
proof (rule, rule)
  fix  $x$ 
  assume  $\exists u v c. x = (1 - c) *_{R} u + c *_{R} v \wedge$ 
     $0 \leq c \wedge c \leq 1 \wedge u \in S \wedge (\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n \wedge v \in$ 
convex hull } T)
  then obtain  $u v c T$  where obt:  $x = (1 - c) *_{R} u + c *_{R} v$ 
     $0 \leq c \wedge c \leq 1 \wedge u \in S \wedge \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n \wedge v \in \text{convex hull } T$ 
  by auto
  moreover have  $(1 - c) *_{R} u + c *_{R} v \in \text{convex hull insert } u T$ 
  by (meson convexD_alt convex_convex_hull hull_inc hull_mono in_mono
insertCI obt(2) obt(7) subset_insertI)
  ultimately show  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex}$ 
hull } T
    by (rule_tac x=insert u T in exI) (auto simp: card_insert_if)
next
  fix  $x$ 
  assume  $\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull } T$ 
  then obtain  $T$  where  $T: \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq \text{Suc } n \wedge x \in \text{convex hull}$ 
T
    by auto
  show  $\exists u v c. x = (1 - c) *_{R} u + c *_{R} v \wedge$ 
     $0 \leq c \wedge c \leq 1 \wedge u \in S \wedge (\exists T. \text{finite } T \wedge T \subseteq S \wedge \text{card } T \leq n \wedge v \in$ 
convex hull } T)
  proof (cases card T = Suc n)
    case False
    then have  $\text{card } T \leq n$  using  $T(3)$  by auto
    then show ?thesis
      using  $\langle w \in S \rangle$  and  $T$ 
      by (rule_tac x=w in exI, rule_tac x=x in exI, rule_tac x=1 in exI)
  auto
  next
  case True
  then obtain  $a u$  where  $au: T = \text{insert } a u \wedge a \notin u$ 
    by (metis card_le_Suc_iff order_refl)
  show ?thesis
  proof (cases u = {})
    case True
    then have  $x = a$  using  $T(4)[\text{unfolded } au]$  by auto
    show ?thesis unfolding  $\langle x = a \rangle$ 
      using  $T \langle n \neq 0 \rangle$  unfolding  $au$ 
      by (rule_tac x=a in exI, rule_tac x=a in exI, rule_tac x=1 in exI)
  force
  next
  case False

```



```

obtain  $ux\ vx\ b$  where  $obt: ux \geq 0\ vx \geq 0\ ux + vx = 1$ 
   $b \in convex\ hull\ u\ x = ux *_R\ a + vx *_R\ b$ 
  using  $T(4)[unfolded\ au\ convex\_hull\_insert[OF\ False]]$ 
  by auto
have  $*$ :  $1 - vx = ux$  using  $obt(3)$  by auto
show ?thesis
  using  $obt\ T(1-3)\ card\_insert\_disjoint[OF\_ au(2)]$  unfolding  $au\ *$ 
  by ( $rule\_tac\ x=a$  in  $exI$ ,  $rule\_tac\ x=b$  in  $exI$ ,  $rule\_tac\ x=vx$  in  $exI$ )
force
  qed
  qed
  qed
then show ?thesis
  using  $compact\_convex\_combinations[OF\ assms\ Suc]$  by simp
  qed
  qed
qed

```

5.3.4 Extremal points of a simplex are some vertices

lemma *dist_increases_online*:

fixes $a\ b\ d :: 'a::real_inner$

assumes $d \neq 0$

shows $dist\ a\ (b + d) > dist\ a\ b \vee dist\ a\ (b - d) > dist\ a\ b$

proof (*cases inner a d - inner b d > 0*)

case *True*

then have $0 < inner\ d\ d + (inner\ a\ d * 2 - inner\ b\ d * 2)$

using *assms*

by (*intro add_pos_pos*) *auto*

then show *?thesis*

unfolding *dist_norm* **and** *norm_eq_sqrt_inner* **and** *real_sqrt_less_iff*

by (*simp add: algebra_simps inner_commute*)

next

case *False*

then have $0 < inner\ d\ d + (inner\ b\ d * 2 - inner\ a\ d * 2)$

using *assms*

by (*intro add_pos_nonneg*) *auto*

then show *?thesis*

unfolding *dist_norm* **and** *norm_eq_sqrt_inner* **and** *real_sqrt_less_iff*

by (*simp add: algebra_simps inner_commute*)

qed

lemma *norm_increases_online*:

fixes $d :: 'a::real_inner$

shows $d \neq 0 \implies norm\ (a + d) > norm\ a \vee norm\ (a - d) > norm\ a$

using *dist_increases_online[of d a 0]* **unfolding** *dist_norm* **by** *auto*

lemma *simplex_furthest_lt*:

fixes $S :: 'a::real_inner\ set$

```

assumes finite S
shows  $\forall x \in \text{convex hull } S. x \notin S \longrightarrow (\exists y \in \text{convex hull } S. \text{norm } (x - a) <$ 
 $\text{norm}(y - a))$ 
using assms
proof induct
fix x S
assume as: finite S xnotinS  $\forall x \in \text{convex hull } S. x \notin S \longrightarrow (\exists y \in \text{convex hull } S. \text{norm}$ 
 $(x - a) < \text{norm } (y - a))$ 
show  $\forall xa \in \text{convex hull insert } x S. xa \notin \text{insert } x S \longrightarrow$ 
 $(\exists y \in \text{convex hull insert } x S. \text{norm } (xa - a) < \text{norm } (y - a))$ 
proof (intro impI ballI, cases S = {})
case False
fix y
assume y: y ∈ convex hull insert x S y ∉ insert x S
obtain u v b where obt: u ≥ 0 v ≥ 0 u + v = 1 b ∈ convex hull S y = u *R x +
 $v *_{R} b$ 
using y(1)[unfolded convex_hull_insert[OF False]] by auto
show  $\exists z \in \text{convex hull insert } x S. \text{norm } (y - a) < \text{norm } (z - a)$ 
proof (cases y ∈ convex hull S)
case True
then obtain z where z ∈ convex hull S norm (y - a) < norm (z - a)
using as(3)[THEN bspec[where x=y]] and y(2) by auto
then show ?thesis
by (meson hull_mono subsetD subset_insertI)
next
case False
show ?thesis
proof (cases u = 0 ∨ v = 0)
case True
with False show ?thesis
using obt y by auto
next
case False
then obtain w where w: w > 0 w < u w < v
using field_lbound_gt_zero[of u v] and obt(1,2) by auto
have  $x \neq b$ 
proof
assume  $x = b$ 
then have  $y = b$  unfolding obt(5)
using obt(3) by (auto simp: scaleR_left_distrib[symmetric])
then show False using obt(4) and False
using  $\langle x = b \rangle y(2)$  by blast
qed
then have  $*$ :  $w *_{R} (x - b) \neq 0$  using w(1) by auto
show ?thesis
using dist_increases_online[OF *, of a y]
proof (elim disjE)
assume  $\text{dist } a y < \text{dist } a (y + w *_{R} (x - b))$ 
then have  $\text{norm } (y - a) < \text{norm } ((u + w) *_{R} x + (v - w) *_{R} b - a)$ 

```

```

      unfolding dist_commute[of a]
      unfolding dist_norm obt(5)
      by (simp add: algebra_simps)
    moreover have  $(u + w) *_{\mathbb{R}} x + (v - w) *_{\mathbb{R}} b \in \text{convex hull insert } x S$ 
      unfolding convex_hull_insert[OF ‹ $S \neq \{\}$ ›]
    proof (intro CollectI conjI exI)
      show  $u + w \geq 0 \ v - w \geq 0$ 
        using obt(1) w by auto
      qed (use obt in auto)
    ultimately show ?thesis by auto
  next
    assume  $\text{dist } a \ y < \text{dist } a \ (y - w *_{\mathbb{R}} (x - b))$ 
    then have  $\text{norm } (y - a) < \text{norm } ((u - w) *_{\mathbb{R}} x + (v + w) *_{\mathbb{R}} b - a)$ 
      unfolding dist_commute[of a]
      unfolding dist_norm obt(5)
      by (simp add: algebra_simps)
    moreover have  $(u - w) *_{\mathbb{R}} x + (v + w) *_{\mathbb{R}} b \in \text{convex hull insert } x S$ 
      unfolding convex_hull_insert[OF ‹ $S \neq \{\}$ ›]
    proof (intro CollectI conjI exI)
      show  $u - w \geq 0 \ v + w \geq 0$ 
        using obt(1) w by auto
      qed (use obt in auto)
    ultimately show ?thesis by auto
  qed
qed
qed
qed
qed auto
qed (auto simp: assms)

lemma simplex_furthest_le:
  fixes  $S :: 'a::\text{real\_inner\_set}$ 
  assumes finite S
    and  $S \neq \{\}$ 
  shows  $\exists y \in S. \forall x \in \text{convex hull } S. \text{norm } (x - a) \leq \text{norm } (y - a)$ 
proof -
  have  $\text{convex hull } S \neq \{\}$ 
    using hull_subset[of S convex] and assms(2) by auto
  then obtain x where  $x \in \text{convex hull } S \ \forall y \in \text{convex hull } S. \text{norm } (y - a) \leq$ 
 $\text{norm } (x - a)$ 
    using distance_attains_sup[OF finite_imp_compact_convex_hull[OF ‹finite S›], of a]
    unfolding dist_commute[of a]
    unfolding dist_norm
    by auto
  show ?thesis
proof (cases  $x \in S$ )
  case False
  then obtain y where  $y \in \text{convex hull } S \ \text{norm } (x - a) < \text{norm } (y - a)$ 
    using simplex_furthest_lt[OF assms(1), THEN bspec[where  $x=x$ ]] and x(1)

```

```

    by auto
  then show ?thesis
    using x(2)[THEN bspec[where x=y]] by auto
next
  case True
  with x show ?thesis by auto
qed
qed

lemma simplex_furthest_le_exists:
  fixes S :: ('a::real_inner) set
  shows finite S  $\implies$   $\forall x \in (\text{convex hull } S). \exists y \in S. \text{norm } (x - a) \leq \text{norm } (y - a)$ 
  using simplex_furthest_le[of S] by (cases S = {}) auto

lemma simplex_extremal_le:
  fixes S :: 'a::real_inner set
  assumes finite S
  and S  $\neq$  {}
  shows  $\exists u \in S. \exists v \in S. \forall x \in \text{convex hull } S. \forall y \in \text{convex hull } S. \text{norm } (x - y) \leq$ 
  norm (u - v)
proof -
  have convex_hull_S  $\neq$  {}
  using hull_subset[of S convex] and assms(2) by auto
  then obtain u v where obt:  $u \in \text{convex hull } S \ v \in \text{convex hull } S$ 
   $\forall x \in \text{convex hull } S. \forall y \in \text{convex hull } S. \text{norm } (x - y) \leq \text{norm } (u - v)$ 
  using compact_sup_maxdistance[OF finite_imp_compact_convex_hull[OF
  assms(1)]]
  by (auto simp: dist_norm)
  then show ?thesis
proof (cases  $u \notin S \vee v \notin S$ , elim disjE)
  assume u  $\notin$  S
  then obtain y where  $y \in \text{convex hull } S$  norm (u - v) < norm (y - v)
  using simplex_furthest_lt[OF assms(1), THEN bspec[where x=u]] and
  obt(1)
  by auto
  then show ?thesis
  using obt(3)[THEN bspec[where x=y], THEN bspec[where x=v]] and obt(2)
  by auto
next
  assume v  $\notin$  S
  then obtain y where  $y \in \text{convex hull } S$  norm (v - u) < norm (y - u)
  using simplex_furthest_lt[OF assms(1), THEN bspec[where x=v]] and
  obt(2)
  by auto
  then show ?thesis
  using obt(3)[THEN bspec[where x=u], THEN bspec[where x=y]] and obt(1)
  by (auto simp: norm_minus_commute)
qed auto
qed

```

```

lemma simplex_extremal_le_exists:
  fixes S :: 'a::real_inner set
  shows finite S  $\implies$   $x \in \text{convex hull } S \implies y \in \text{convex hull } S \implies$ 
     $\exists u \in S. \exists v \in S. \text{norm } (x - y) \leq \text{norm } (u - v)$ 
  using convex_hull_empty simplex_extremal_le[of S]
  by(cases S = {}) auto

```

5.3.5 Closest point of a convex set is unique, with a continuous projection

```

definition closest_point :: 'a::{real_inner,heine_borel} set  $\Rightarrow$  'a  $\Rightarrow$  'a
  where closest_point S a = (SOME x.  $x \in S \wedge (\forall y \in S. \text{dist } a x \leq \text{dist } a y)$ )

```

```

lemma closest_point_exists:
  assumes closed S
  and S  $\neq$  {}
  shows closest_point_in_set: closest_point S a  $\in$  S
  and  $\forall y \in S. \text{dist } a (\text{closest\_point } S a) \leq \text{dist } a y$ 
  unfolding closest_point_def
  by (rule_tac someI2_ex, auto intro: distance_attains_inf[OF assms(1,2), of a])+

```

```

lemma closest_point_le: closed S  $\implies$   $x \in S \implies \text{dist } a (\text{closest\_point } S a) \leq \text{dist } a x$ 
  using closest_point_exists[of S] by auto

```

```

lemma closest_point_self:
  assumes  $x \in S$ 
  shows closest_point S x = x
  unfolding closest_point_def
  by (rule someI_equality, rule ex1I[of _ x]) (use assms in auto)

```

```

lemma closest_point_refl: closed S  $\implies$   $S \neq \{\}$   $\implies$  closest_point S x = x  $\longleftrightarrow$   $x \in S$ 
  using closest_point_in_set[of S x] closest_point_self[of x S]
  by auto

```

```

lemma closer_points_lemma:
  assumes inner y z > 0
  shows  $\exists u > 0. \forall v > 0. v \leq u \longrightarrow \text{norm}(v *_{\mathbb{R}} z - y) < \text{norm } y$ 
proof -
  have z: inner z z > 0
  unfolding inner_gt_zero_iff using assms by auto
  have norm (v *R z - y) < norm y
  if 0 < v and v  $\leq$  inner y z / inner z z for v
  unfolding norm_lt using z assms that
  by (simp add: field_simps inner_diff inner_commute mult_strict_left_mono[OF _ <0<v])

```

```

then show ?thesis
  using assms z
  by (rule_tac x = inner y z / inner z z in exI) auto
qed

```

```

lemma closer_point_lemma:
  assumes inner (y - x) (z - x) > 0
  shows  $\exists u > 0. u \leq 1 \wedge \text{dist } (x + u *_R (z - x)) y < \text{dist } x y$ 
proof -
  obtain u where u > 0
  and u:  $\bigwedge v. [0 < v; v \leq u] \implies \text{norm } (v *_R (z - x) - (y - x)) < \text{norm } (y - x)$ 
  using closer_points_lemma[OF assms] by auto
  show ?thesis
  using u[of min u 1] and  $\langle u > 0 \rangle$ 
  by (metis diff_diff_add dist_commute dist_norm less_eq_real_def not_less u
zero_less_one)
qed

```

```

lemma any_closest_point_dot:
  assumes convex S closed S x ∈ S y ∈ S  $\forall z \in S. \text{dist } a x \leq \text{dist } a z$ 
  shows inner (a - x) (y - x) ≤ 0
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain u where u:  $u > 0 \wedge u \leq 1 \wedge \text{dist } (x + u *_R (y - x)) a < \text{dist } x a$ 
  using closer_point_lemma[of a x y] by auto
  let ?z = (1 - u) *_R x + u *_R y
  have ?z ∈ S
  using convexD_alt[OF assms(1,3,4), of u] using u by auto
  then show False
  using assms(5)[THEN bspec[where x=?z]] and u(3)
  by (auto simp: dist_commute algebra_simps)
qed

```

```

lemma any_closest_point_unique:
  fixes x :: 'a::real_inner
  assumes convex S closed S x ∈ S y ∈ S
   $\forall z \in S. \text{dist } a x \leq \text{dist } a z \wedge \forall z \in S. \text{dist } a y \leq \text{dist } a z$ 
  shows x = y
  using any_closest_point_dot[OF assms(1-4,5)] and any_closest_point_dot[OF
assms(1-2,4,3,6)]
  unfolding norm_pths(1) and norm_le_square
  by (auto simp: algebra_simps)

```

```

lemma closest_point_unique:
  assumes convex S closed S x ∈ S  $\forall z \in S. \text{dist } a x \leq \text{dist } a z$ 
  shows x = closest_point S a
  using any_closest_point_unique[OF assms(1-3) _ assms(4), of closest_point
S a]
  using closest_point_exists[OF assms(2)] and assms(3) by auto

```

lemma *closest_point_dot*:
assumes *convex S closed S x ∈ S*
shows $\text{inner } (a - \text{closest_point } S a) (x - \text{closest_point } S a) \leq 0$
using *any_closest_point_dot[OF assms(1,2) _ assms(3)]*
by (*metis assms(2) assms(3) closest_point_in_set closest_point_le empty_iff*)

lemma *closest_point_lt*:
assumes *convex S closed S x ∈ S x ≠ closest_point S a*
shows $\text{dist } a (\text{closest_point } S a) < \text{dist } a x$
using *closest_point_unique[where a=a] closest_point_le[where a=a] assms*
by *fastforce*

lemma *setdist_closest_point*:
 $\llbracket \text{closed } S; S \neq \{\} \rrbracket \implies \text{setdist } \{a\} S = \text{dist } a (\text{closest_point } S a)$
by (*metis closest_point_exists(2) closest_point_in_set emptyE insert_iff setdist_unique*)

lemma *closest_point_lipschitz*:
assumes *convex S*
and *closed S S ≠ {}*
shows $\text{dist } (\text{closest_point } S x) (\text{closest_point } S y) \leq \text{dist } x y$
proof –
have $\text{inner } (x - \text{closest_point } S x) (\text{closest_point } S y - \text{closest_point } S x) \leq 0$
and $\text{inner } (y - \text{closest_point } S y) (\text{closest_point } S x - \text{closest_point } S y) \leq 0$
by (*simp_all add: assms closest_point_dot closest_point_in_set*)
then show *?thesis unfolding dist_norm and norm_le*
using *inner_ge_zero[of (x - closest_point S x) - (y - closest_point S y)]*
by (*simp add: inner_add inner_diff inner_commute*)
qed

lemma *continuous_at_closest_point*:
assumes *convex S*
and *closed S*
and *S ≠ {}*
shows *continuous (at x) (closest_point S)*
unfolding *continuous_at_eps_delta*
using *le_less_trans[OF closest_point_lipschitz[OF assms]] by auto*

lemma *continuous_on_closest_point*:
assumes *convex S*
and *closed S*
and *S ≠ {}*
shows *continuous_on t (closest_point S)*
by (*metis continuous_at_imp_continuous_on continuous_at_closest_point[OF assms]*)

proposition *closest_point_in_rel_interior*:
assumes *closed S S ≠ {} and x: x ∈ affine hull S*

```

    shows  $\text{closest\_point } S \ x \in \text{rel\_interior } S \iff x \in \text{rel\_interior } S$ 
  proof (cases  $x \in S$ )
    case True
      then show ?thesis
        by (simp add: closest_point_self)
    next
      case False
      then have False if asm:  $\text{closest\_point } S \ x \in \text{rel\_interior } S$ 
      proof -
        obtain  $e$  where  $e > 0$  and clox:  $\text{closest\_point } S \ x \in S$ 
          and  $e$ :  $\text{cball } (\text{closest\_point } S \ x) \ e \cap \text{affine hull } S \subseteq S$ 
          using asm mem_rel_interior_cball by blast
        then have clo_notx:  $\text{closest\_point } S \ x \neq x$ 
          using  $\langle x \notin S \rangle$  by auto
        define  $y$  where  $y \equiv \text{closest\_point } S \ x -$ 
           $(\min 1 (e / \text{norm}(\text{closest\_point } S \ x - x))) *_{\mathbb{R}} (\text{closest\_point } S$ 
 $x - x)$ 
        have  $x - y = (1 - \min 1 (e / \text{norm} (\text{closest\_point } S \ x - x))) *_{\mathbb{R}} (x -$ 
 $\text{closest\_point } S \ x)$ 
          by (simp add: y_def algebra_simps)
        then have  $\text{norm } (x - y) = \text{abs } ((1 - \min 1 (e / \text{norm} (\text{closest\_point } S \ x -$ 
 $x))) * \text{norm}(x - \text{closest\_point } S \ x))$ 
          by simp
        also have  $\dots < \text{norm}(x - \text{closest\_point } S \ x)$ 
          using clo_notx  $\langle e > 0 \rangle$ 
          by (auto simp: mult_less_cancel_right2 field_split_simps)
        finally have no_less:  $\text{norm } (x - y) < \text{norm } (x - \text{closest\_point } S \ x)$  .
        have  $y \in \text{affine hull } S$ 
          unfolding y_def
          by (meson affine_affine_hull clox hull_subset mem_affine_3_minus2 subsetD
 $x$ )
        moreover have  $\text{dist } (\text{closest\_point } S \ x) \ y \leq e$ 
          using  $\langle e > 0 \rangle$  by (auto simp: y_def min_mult_distrib_right)
        ultimately have  $y \in S$ 
          using subsetD [OF  $e$ ] by simp
        then have  $\text{dist } x (\text{closest\_point } S \ x) \leq \text{dist } x \ y$ 
          by (simp add: closest_point_le  $\langle \text{closed } S \rangle$ )
        with no_less show False
          by (simp add: dist_norm)
      qed
    moreover have  $x \notin \text{rel\_interior } S$ 
      using rel_interior_subset False by blast
    ultimately show ?thesis by blast
  qed

```

Various point-to-set separating/supporting hyperplane theorems

lemma *supporting_hyperplane_closed_point*:
 fixes $z :: 'a :: \{\text{real_inner, heine_borel}\}$


```

assumes convex S
  and closed S
  and S ≠ {}
  and z ∉ S
shows ∃ a b. ∃ y ∈ S. inner a z < b ∧ inner a y = b ∧ (∀ x ∈ S. inner a x ≥ b)
proof -
  obtain y where y ∈ S and y: ∀ x ∈ S. dist z y ≤ dist z x
    by (metis distance_attains_inf[OF assms(2-3)])
  show ?thesis
  proof (intro exI bexI conjI ballI)
    show (y - z) · z < (y - z) · y
      by (metis ⟨y ∈ S⟩ assms(4) diff_gt_0_iff_gt inner_commute inner_diff_left
inner_gt_zero_iff_right_minus_eq)
    show (y - z) · y ≤ (y - z) · x if x ∈ S for x
      proof (rule ccontr)
        have *: ∧ u. 0 ≤ u ∧ u ≤ 1 → dist z y ≤ dist z ((1 - u) *R y + u *R x)
          using assms(1)[unfolded convex_alt] and y and ⟨x ∈ S⟩ and ⟨y ∈ S⟩ by auto
        assume ¬ (y - z) · y ≤ (y - z) · x
        then obtain v where v > 0 v ≤ 1 dist (y + v *R (x - y)) z < dist y z
          using closer_point_lemma[of z y x] by (auto simp: inner_diff)
        then show False
          using *[of v] by (auto simp: dist_commute algebra_simps)
      qed
    qed (use ⟨y ∈ S⟩ in auto)
  qed

```

lemma separating_hyperplane_closed_point:

```

fixes z :: 'a::{real_inner,heine_borel}
assumes convex S
  and closed S
  and z ∉ S
shows ∃ a b. inner a z < b ∧ (∀ x ∈ S. inner a x > b)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: gt_ex)
  next
  case False
  obtain y where y ∈ S and y: ∧ x. x ∈ S ⇒ dist z y ≤ dist z x
    by (metis distance_attains_inf[OF assms(2) False])
  show ?thesis
  proof (intro exI conjI ballI)
    show (y - z) · z < inner (y - z) z + (norm (y - z))2 / 2
      using ⟨y ∈ S⟩ ⟨z ∉ S⟩ by auto
    next
    fix x
    assume x ∈ S
    have False if *: 0 < inner (z - y) (x - y)
      proof -

```

```

obtain  $u$  where  $u > 0 \ u \leq 1 \ \text{dist } (y + u *_{\mathbb{R}} (x - y)) \ z < \text{dist } y \ z$ 
using * closer_point_lemma by blast
then show False using  $y[of \ y + u *_{\mathbb{R}} (x - y)] \ \text{convexD\_alt} \ [OF \ \langle \text{convex } S \rangle]$ 
using  $\langle x \in S \rangle \ \langle y \in S \rangle$  by (auto simp: dist_commute algebra_simps)
qed
moreover have  $0 < (\text{norm } (y - z))^2$ 
using  $\langle y \in S \rangle \ \langle z \notin S \rangle$  by auto
then have  $0 < \text{inner } (y - z) (y - z)$ 
unfolding power2_norm_eq_inner by simp
ultimately show  $(y - z) \cdot z + (\text{norm } (y - z))^2 / 2 < (y - z) \cdot x$ 
by (force simp: field_simps power2_norm_eq_inner inner_commute inner_diff)
qed
qed

```

```

lemma separating_hyperplane_closed_0:
assumes convex  $(S :: 'a :: \text{euclidean\_space}) \ \text{set}$ 
and closed  $S$ 
and  $0 \notin S$ 
shows  $\exists a \ b. \ a \neq 0 \wedge 0 < b \wedge (\forall x \in S. \ \text{inner } a \ x > b)$ 
proof (cases  $S = \{\}$ )
case True
have  $(\text{SOME } i. \ i \in \text{Basis}) \neq (0 :: 'a)$ 
by (metis Basis_zero SOME_Basis)
then show ?thesis
using True zero_less_one by blast
next
case False
then show ?thesis
using False using separating_hyperplane_closed_point [OF assms]
by (metis all_not_in_conv inner_zero_left inner_zero_right less_eq_real_def not_le)
qed

```

Now set-to-set for closed/compact sets

```

lemma separating_hyperplane_closed_compact:
fixes  $S :: 'a :: \text{euclidean\_space} \ \text{set}$ 
assumes convex  $S$ 
and closed  $S$ 
and convex  $T$ 
and compact  $T$ 
and  $T \neq \{\}$ 
and  $S \cap T = \{\}$ 
shows  $\exists a \ b. \ (\forall x \in S. \ \text{inner } a \ x < b) \wedge (\forall x \in T. \ \text{inner } a \ x > b)$ 
proof (cases  $S = \{\}$ )
case True
obtain  $b$  where  $b > 0 \ \forall x \in T. \ \text{norm } x \leq b$ 
using compact_imp_bounded [OF assms(4)] unfolding bounded_pos by auto

```

```

obtain  $z :: 'a$  where  $z: \text{norm } z = b + 1$ 
  using vector_choose_size[of  $b + 1$ ] and  $b(1)$  by auto
then have  $z \notin T$  using  $b(2)$ [THEN bspec[where  $x=z$ ]] by auto
then obtain  $a b$  where  $ab: \text{inner } a z < b \ \forall x \in T. b < \text{inner } a x$ 
  using separating_hyperplane_closed_point[OF assms(3)] compact_imp_closed[OF
assms(4)], of  $z$ ]
  by auto
then show ?thesis
  using True by auto
next
  case False
then obtain  $y$  where  $y \in S$  by auto
obtain  $a b$  where  $0 < b$  and  $\S: \bigwedge x. x \in (\bigcup x \in S. \bigcup y \in T. \{x - y\}) \implies b <$ 
inner a x
  using separating_hyperplane_closed_point[OF convex_differences[OF assms(1,3)],
of  $0$ ]
  using closed_compact_differences assms by fastforce
have  $ab: b + \text{inner } a y < \text{inner } a x$  if  $x \in S \ y \in T$  for  $x \ y$ 
  using  $\S$  [of  $x-y$ ] that by (auto simp add: inner_diff_right less_diff_eq)
define  $k$  where  $k = (\text{SUP } x \in T. a \cdot x)$ 
have  $k + b / 2 < a \cdot x$  if  $x \in S$  for  $x$ 
proof -
  have  $k \leq \text{inner } a x - b$ 
    unfolding  $k\_def$ 
    using  $\langle T \neq \{\} \rangle$  ab that by (fastforce intro: cSUP_least)
  then show ?thesis
    using  $\langle 0 < b \rangle$  by auto
qed
moreover
have  $-(k + b / 2) < -a \cdot x$  if  $x \in T$  for  $x$ 
proof -
  have  $\text{inner } a x - b / 2 < k$ 
    unfolding  $k\_def$ 
  proof (subst less_cSUP_iff)
    show  $T \neq \{\}$  by fact
    show bdd_above  $((\cdot) a ' T)$ 
      using  $ab$ [rule_format, of  $y$ ]  $\langle y \in S \rangle$ 
      by (intro bdd_aboveI2[where  $M = \text{inner } a y - b$ ]) (auto simp: field_simps
intro: less_imp_le)
    show  $\exists y \in T. a \cdot x - b / 2 < a \cdot y$ 
      using  $\langle 0 < b \rangle$  that by force
  qed
then show ?thesis
  by auto
qed
ultimately show ?thesis
  by (metis inner_minus_left neg_less_iff_less)
qed

```

```

lemma separating_hyperplane_compact_closed:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes convex S
  and compact S
  and  $S \neq \{\}$ 
  and convex T
  and closed T
  and  $S \cap T = \{\}$ 
  shows  $\exists a b. (\forall x \in S. \text{inner } a \ x < b) \wedge (\forall x \in T. \text{inner } a \ x > b)$ 
proof –
  obtain  $a b$  where  $(\forall x \in T. \text{inner } a \ x < b) \wedge (\forall x \in S. b < \text{inner } a \ x)$ 
  by (metis disjoint_iff_not_equal separating_hyperplane_closed_compact assms)
  then show ?thesis
  by (metis inner_minus_left_neg_less_iff_less)
qed

```

General case without assuming closure and getting non-strict separation

```

lemma separating_hyperplane_set_0:
  assumes convex S (0::'a::euclidean_space) \notin S
  shows  $\exists a. a \neq 0 \wedge (\forall x \in S. 0 \leq \text{inner } a \ x)$ 
proof –
  let  $?k = \lambda c. \{x::'a. 0 \leq \text{inner } c \ x\}$ 
  have  $*$ : frontier (cball 0 1) \cap \bigcap f \neq \{\} if  $as: f \subseteq ?k \text{ ' } S$  finite f for f
proof –
  obtain  $c$  where  $c: f = ?k \text{ ' } c \subseteq S$  finite c
  using finite_subset_image[OF as(2,1)] by auto
  then obtain  $a b$  where  $ab: a \neq 0 \ 0 < b \ \forall x \in \text{convex hull } c. b < \text{inner } a \ x$ 
  using separating_hyperplane_closed_0[OF convex_convex_hull, of c]
  using finite_imp_compact_convex_hull[OF c(3), THEN compact_imp_closed]
and assms(2)
  using subset_hull[of convex, OF assms(1), symmetric, of c]
  by force
  have  $\text{norm } (a /_R \text{norm } a) = 1$ 
  by (simp add: ab(1))
  moreover have  $(\forall y \in c. 0 \leq y \cdot (a /_R \text{norm } a))$ 
  using hull_subset[of c convex] ab by (force simp: inner_commute)
  ultimately have  $\exists x. \text{norm } x = 1 \wedge (\forall y \in c. 0 \leq \text{inner } y \ x)$ 
  by blast
  then show frontier (cball 0 1) \cap \bigcap f \neq \{\}
  unfolding  $c(1)$  frontier_cball sphere_def dist_norm by auto
qed
have frontier (cball 0 1) \cap (\bigcap (?k ' S)) \neq \{\}
  by (rule compact_imp_fip) (use * closed_halfspace_ge in auto)
then obtain  $x$  where  $\text{norm } x = 1 \ \forall y \in S. x \in ?k \ y$ 
  unfolding frontier_cball dist_norm sphere_def by auto
then show ?thesis
  by (metis inner_commute mem_Collect_eq norm_eq_zero zero_neq_one)

```

qed

lemma *separating_hyperplane_sets*:

fixes $S T :: 'a::euclidean_space\ set$

assumes *convex S*

and *convex T*

and $S \neq \{\}$

and $T \neq \{\}$

and $S \cap T = \{\}$

shows $\exists a b. a \neq 0 \wedge (\forall x \in S. \text{inner } a\ x \leq b) \wedge (\forall x \in T. \text{inner } a\ x \geq b)$

proof -

from *separating_hyperplane_set_0*[*OF convex_differences*[*OF assms*(2,1)]]

obtain a where $a \neq 0 \forall x \in \{x - y \mid x \in T \wedge y \in S\}. 0 \leq \text{inner } a\ x$

using *assms*(3-5) by *force*

then have $*$: $\bigwedge x y. x \in T \implies y \in S \implies \text{inner } a\ y \leq \text{inner } a\ x$

by (*force simp: inner_diff*)

then have *bdd*: *bdd_above* $((\cdot) a) S$

using $\langle T \neq \{\} \rangle$ by (*auto intro: bdd_aboveI2*[*OF **])

show *?thesis*

using $\langle a \neq 0 \rangle$

by (*intro exI*[*of _ a*] *exI*[*of _ SUP* $x \in S. a \cdot x$])

(*auto intro!*: *cSUP_upper bdd cSUP_least* $\langle a \neq 0 \rangle \langle S \neq \{\} \rangle *$)

qed

5.3.6 More convexity generalities

lemma *convex_closure* [*intro,simp*]:

fixes $S :: 'a::real_normed_vector\ set$

assumes *convex S*

shows *convex (closure S)*

apply (*rule convexI*)

unfolding *closure_sequential*

apply (*elim exE*)

subgoal for $x\ y\ u\ v\ f\ g$

by (*rule_tac* $x = \lambda n. u *_{\mathbb{R}} f\ n + v *_{\mathbb{R}} g\ n$ in *exI*) (*force intro: tendsto_intros*
dest: convexD [*OF assms*])

done

lemma *convex_interior* [*intro,simp*]:

fixes $S :: 'a::real_normed_vector\ set$

assumes *convex S*

shows *convex (interior S)*

unfolding *convex_alt Ball_def mem_interior*

proof *clarify*

fix $x\ y\ u$

assume $u: 0 \leq u \leq 1$ (*real*)

fix $e\ d$

assume *ed*: $\text{ball } x\ e \subseteq S \text{ ball } y\ d \subseteq S\ 0 < d\ 0 < e$

show $\exists e > 0. \text{ball } ((1 - u) *_{\mathbb{R}} x + u *_{\mathbb{R}} y)\ e \subseteq S$

```

proof (intro exI conjI subsetI)
  fix z
  assume z: z ∈ ball ((1 - u) *R x + u *R y) (min d e)
  have (1 - u) *R (z - u *R (y - x)) + u *R (z + (1 - u) *R (y - x)) ∈ S
  proof (rule_tac assms[unfolded convex_alt, rule_format])
    show z - u *R (y - x) ∈ S z + (1 - u) *R (y - x) ∈ S
    using ed z u by (auto simp add: algebra_simps dist_norm)
  qed (use u in auto)
  then show z ∈ S
    using u by (auto simp: algebra_simps)
  qed(use u ed in auto)
qed

```

```

lemma convex_hull_eq_empty[simp]: convex hull S = {} ↔ S = {}
  using hull_subset[of S convex] convex_hull_empty by auto

```

5.3.7 Convex set as intersection of halfspaces

```

lemma convex_halfspace_intersection:
  fixes S :: ('a::euclidean_space) set
  assumes closed S convex S
  shows S = ⋂ {h. S ⊆ h ∧ (∃ a b. h = {x. inner a x ≤ b})}
proof -
  { fix z
    assume ∀ T. S ⊆ T ∧ (∃ a b. T = {x. inner a x ≤ b}) → z ∈ T z ∉ S
    then have §: ⋀ a b. S ⊆ {x. inner a x ≤ b} ⇒ z ∈ {x. inner a x ≤ b}
      by blast
    obtain a b where inner a z < b (∀ x ∈ S. inner a x > b)
      using ⟨z ∉ S⟩ assms separating_hyperplane_closed_point by blast
    then have False
      using § [of -a -b] by fastforce
    }
  then show ?thesis
    by force
  qed

```

5.3.8 Convexity of general and special intervals

```

lemma is_interval_convex:
  fixes S :: 'a::euclidean_space set
  assumes is_interval S
  shows convex S
proof (rule convexI)
  fix x y and u v :: real
  assume x ∈ S y ∈ S and uv: 0 ≤ u 0 ≤ v u + v = 1
  then have *: u = 1 - v 1 - v ≥ 0 and **: v = 1 - u 1 - u ≥ 0
    by auto
  {
    fix a b
    assume ¬ b ≤ u * a + v * b

```

```

then have  $u * a < (1 - v) * b$ 
  unfolding not_le using  $\langle 0 \leq v \rangle$  by (auto simp: field_simps)
then have  $a < b$ 
  using  $*(1)$  less_eq_real_def  $w(1)$  by auto
then have  $a \leq u * a + v * b$ 
  unfolding * using  $\langle 0 \leq v \rangle$ 
  by (auto simp: field_simps intro!: mult_right_mono)
}
moreover
{
  fix  $a b$ 
  assume  $\neg u * a + v * b \leq a$ 
  then have  $v * b > (1 - u) * a$ 
    unfolding not_le using  $\langle 0 \leq v \rangle$  by (auto simp: field_simps)
  then have  $a < b$ 
    unfolding * using  $\langle 0 \leq v \rangle$ 
    by (rule_tac mult_left_less_imp_less) (auto simp: field_simps)
  then have  $u * a + v * b \leq b$ 
    unfolding **
    using  $*(2)$   $\langle 0 \leq u \rangle$  by (auto simp: algebra_simps mult_right_mono)
}
ultimately show  $u *_R x + v *_R y \in S$ 
  using DIM_positive[where 'a='a]
  by (intro mem_is_intervalI [OF assms  $\langle x \in S \rangle \langle y \in S \rangle$ ]) (auto simp: inner_simps)
qed

```

```

lemma is_interval_connected:
  fixes  $S :: 'a::euclidean\_space$  set
  shows  $is\_interval\ S \implies connected\ S$ 
  using is_interval_convex convex_connected by auto

```

```

lemma convex_box [simp]: convex (cbox  $a$   $b$ ) convex (box  $a$  ( $b::'a::euclidean\_space$ ))
  by (auto simp add: is_interval_convex)

```

A non-singleton connected set is perfect (i.e. has no isolated points).

```

lemma connected_imp_perfect:
  fixes  $a :: 'a::metric\_space$ 
  assumes  $connected\ S$   $a \in S$  and  $S: \bigwedge x. S \neq \{x\}$ 
  shows  $a$  islimpt  $S$ 
proof -
  have False if  $a \in T$  open  $T \bigwedge y. \llbracket y \in S; y \in T \rrbracket \implies y = a$  for  $T$ 
  proof -
    obtain  $e$  where  $e > 0$  and  $e: cball\ a\ e \subseteq T$ 
      using  $\langle open\ T \rangle \langle a \in T \rangle$  by (auto simp: open_contains_cball)
    have  $openin\ (top\_of\_set\ S)\ \{a\}$ 
      unfolding openin_open using that  $\langle a \in S \rangle$  by blast
    moreover have  $closedin\ (top\_of\_set\ S)\ \{a\}$ 
      by (simp add: assms)

```

```

ultimately show False
  using ⟨connected S⟩ connected_clopen S by blast
qed
then show ?thesis
  unfolding islimpt_def by blast
qed

```

```

lemma islimpt_Ioc [simp]:
  fixes a :: real
  assumes a < b
  shows x islimpt {a <..b}  $\longleftrightarrow$  x ∈ {a..b} (is ?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (metis assms closed_atLeastAtMost closed_limpt closure_greaterThanAtMost
closure_subset islimpt_subset)
  next
    assume ?rhs
    then have x ∈ closure {a <..b}
      using assms closure_greaterThanLessThan by blast
    then show ?lhs
      by (metis (no_types) Diff_empty Diff_insert0 interior_lessThanAtMost interior_limit_point interior_subset islimpt_in_closure islimpt_subset)
qed

```

```

lemma islimpt_Ico [simp]:
  fixes a :: real
  assumes a < b shows x islimpt {a..b}  $\longleftrightarrow$  x ∈ {a..b}
  by (metis assms closure_atLeastLessThan closure_greaterThanAtMost islimpt_Ioc
limpt_of_closure)

```

```

lemma islimpt_Icc [simp]:
  fixes a :: real
  assumes a < b shows x islimpt {a..b}  $\longleftrightarrow$  x ∈ {a..b}
  by (metis assms closure_atLeastLessThan islimpt_Ico limpt_of_closure)

```

```

lemma connected_imp_perfect_aff_dim:
   $\llbracket$ connected S; aff_dim S  $\neq$  0; a ∈ S $\rrbracket \implies$  a islimpt S
  using aff_dim_sing connected_imp_perfect by blast

```

5.3.9 On *real*, *is_interval*, *convex* and *connected* are all equivalent

```

lemma mem_is_interval_1_I:
  fixes a b c :: real
  assumes is_interval S
  assumes a ∈ S c ∈ S
  assumes a ≤ b b ≤ c
  shows b ∈ S
  using assms is_interval_1 by blast

```



```

lemma is_interval_connected_1:
  fixes S :: real set
  shows is_interval S  $\longleftrightarrow$  connected S
  by (meson connected_iff_interval is_interval_1)

lemma is_interval_convex_1:
  fixes S :: real set
  shows is_interval S  $\longleftrightarrow$  convex S
  by (metis is_interval_convex convex_connected is_interval_connected_1)

lemma connected_compact_interval_1:
  connected S  $\wedge$  compact S  $\longleftrightarrow$  ( $\exists a b. S = \{a..b::real\}$ )
  by (auto simp: is_interval_connected_1 [symmetric] is_interval_compact)

lemma connected_convex_1:
  fixes S :: real set
  shows connected S  $\longleftrightarrow$  convex S
  by (metis is_interval_convex convex_connected is_interval_connected_1)

lemma connected_space_iff_is_interval_1 [iff]:
  fixes S :: real set
  shows connected_space (top_of_set S)  $\longleftrightarrow$  is_interval S
  using connectedin_topspace is_interval_connected_1 by force

lemma connected_convex_1_gen:
  fixes S :: 'a :: euclidean_space set
  assumes DIM('a) = 1
  shows connected S  $\longleftrightarrow$  convex S
proof -
  obtain f:: 'a  $\Rightarrow$  real where linf: linear f and inj f
    using subspace_isomorphism[OF subspace_UNIV subspace_UNIV,
      where 'a='a and 'b=real]
    unfolding Euclidean_Space.dim_UNIV
    by (auto simp: assms)
  then have f -' (f ' S) = S
    by (simp add: inj_vimage_image_eq)
  then show ?thesis
    by (metis connected_convex_1 convex_linear_vimage linf convex_connected
      connected_linear_image)
qed

lemma [simp]:
  fixes r s::real
  shows is_interval_io: is_interval {..

```

by (*simp_all add: is_interval_convex_1*)

5.3.10 Another intermediate value theorem formulation

lemma *ivt_increasing_component_on_1*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

assumes $a \leq b$

and *continuous_on* $\{a..b\}$ f

and $(f\ a) \cdot k \leq y \leq (f\ b) \cdot k$

shows $\exists x \in \{a..b\}. (f\ x) \cdot k = y$

proof –

have $f\ a \in f\ ` \text{cbox } a\ b$ $f\ b \in f\ ` \text{cbox } a\ b$

using $\langle a \leq b \rangle$ **by** *auto*

then show *?thesis*

using *connected_ivt_component*[*of f ` cbox a b f a f b k y*]

by (*simp add: connected_continuous_image assms*)

qed

lemma *ivt_increasing_component_1*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

shows $a \leq b \implies \forall x \in \{a..b\}. \text{continuous } (\text{at } x) f \implies$

$f\ a \cdot k \leq y \implies y \leq f\ b \cdot k \implies \exists x \in \{a..b\}. (f\ x) \cdot k = y$

by (*rule ivt_increasing_component_on_1*) (*auto simp: continuous_at_imp_continuous_on*)

lemma *ivt_decreasing_component_on_1*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

assumes $a \leq b$

and *continuous_on* $\{a..b\}$ f

and $(f\ b) \cdot k \leq y$

and $y \leq (f\ a) \cdot k$

shows $\exists x \in \{a..b\}. (f\ x) \cdot k = y$

using *ivt_increasing_component_on_1*[*of a b $\lambda x. - f\ x\ k - y$ neg_equal_iff_equal*]

using *assms continuous_on_minus* **by** *force*

lemma *ivt_decreasing_component_1*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

shows $a \leq b \implies \forall x \in \{a..b\}. \text{continuous } (\text{at } x) f \implies$

$f\ b \cdot k \leq y \implies y \leq f\ a \cdot k \implies \exists x \in \{a..b\}. (f\ x) \cdot k = y$

by (*rule ivt_decreasing_component_on_1*) (*auto simp: continuous_at_imp_continuous_on*)

5.3.11 A bound within an interval

lemma *convex_hull_eq_real_cbox*:

fixes $x\ y :: \text{real}$ **assumes** $x \leq y$

shows *convex hull* $\{x, y\} = \text{cbox } x\ y$

proof (*rule hull_unique*)

show $\{x, y\} \subseteq \text{cbox } x\ y$ **using** $\langle x \leq y \rangle$ **by** *auto*

show *convex* (*cbox x y*)

by (*rule convex_box*)

next

```

fix S assume {x, y} ⊆ S and convex S
then show cbox x y ⊆ S
  unfolding is_interval_convex_1 [symmetric] is_interval_def Basis_real_def
  by - (clarify, simp (no_asm_use), fast)
qed

```

lemma *unit_interval_convex_hull*:

```

cbox (0::'a::euclidean_space) One = convex hull {x. ∀ i∈Basis. (x·i = 0) ∨ (x·i
= 1)}

```

```

(is ?int = convex hull ?points)

```

proof -

```

have One[simp]: ∧ i. i ∈ Basis ⇒ One · i = 1

```

```

  by (simp add: inner_sum_left sum.If_cases inner_Basis)

```

```

have ?int = {x. ∀ i∈Basis. x · i ∈ cbox 0 1}

```

```

  by (auto simp: cbox_def)

```

```

also have ... = (∑ i∈Basis. (λx. x *R i) ` cbox 0 1)

```

```

  by (simp only: box_eq_set_sum_Basis)

```

```

also have ... = (∑ i∈Basis. (λx. x *R i) ` (convex hull {0, 1}))

```

```

  by (simp only: convex_hull_eq_real_cbox_zero_le_one)

```

```

also have ... = (∑ i∈Basis. convex hull ((λx. x *R i) ` {0, 1}))

```

```

  by (simp add: convex_hull_linear_image)

```

```

also have ... = convex hull (∑ i∈Basis. (λx. x *R i) ` {0, 1})

```

```

  by (simp only: convex_hull_set_sum)

```

```

also have ... = convex hull {x. ∀ i∈Basis. x·i ∈ {0, 1}}

```

```

  by (simp only: box_eq_set_sum_Basis)

```

```

also have convex hull {x. ∀ i∈Basis. x·i ∈ {0, 1}} = convex hull ?points

```

```

  by simp

```

```

finally show ?thesis .

```

qed

And this is a finite set of vertices.

lemma *unit_cube_convex_hull*:

```

obtains S :: 'a::euclidean_space set

```

```

where finite S and cbox 0 (∑ Basis) = convex hull S

```

proof

```

show finite {x::'a. ∀ i∈Basis. x · i = 0 ∨ x · i = 1}

```

```

proof (rule finite_subset, clarify)

```

```

  show finite ((λS. ∑ i∈Basis. (if i ∈ S then 1 else 0) *R i) ` Pow Basis)

```

```

    using finite_Basis by blast

```

```

  fix x :: 'a

```

```

  assume x: ∀ i∈Basis. x · i = 0 ∨ x · i = 1

```

```

  show x ∈ (λS. ∑ i∈Basis. (if i∈S then 1 else 0) *R i) ` Pow Basis

```

```

    apply (rule image_eqI[where x={i. i ∈ Basis ∧ x·i = 1}])

```

```

    using x

```

```

    by (subst euclidean_eq_iff, auto)

```

qed

```

show cbox 0 One = convex hull {x. ∀ i∈Basis. x · i = 0 ∨ x · i = 1}

```

```

  using unit_interval_convex_hull by blast

```

qed

Hence any cube (could do any nonempty interval).

lemma *cube_convex_hull*:

assumes $d > 0$

obtains $S :: 'a::euclidean_space$ set **where**

finite S **and** $\text{cbox } (x - (\sum_{i \in \text{Basis}} d *_{\mathbb{R}} i)) (x + (\sum_{i \in \text{Basis}} d *_{\mathbb{R}} i)) = \text{convex hull } S$

proof –

let $?d = (\sum_{i \in \text{Basis}} d *_{\mathbb{R}} i) :: 'a$

have $*$: $\text{cbox } (x - ?d) (x + ?d) = (\lambda y. x - ?d + (2 * d) *_{\mathbb{R}} y) \text{ ' cbox } 0 (\sum \text{Basis})$

proof (*intro set_eqI iffI*)

fix y

assume $y \in \text{cbox } (x - ?d) (x + ?d)$

then have $\text{inverse } (2 * d) *_{\mathbb{R}} (y - (x - ?d)) \in \text{cbox } 0 (\sum \text{Basis})$

using *assms* **by** (*simp add: mem_box inner_simps*) (*simp add: field_simps*)

with $\langle 0 < d \rangle$ **show** $y \in (\lambda y. x - \text{sum } ((*_{\mathbb{R}}) d) \text{Basis} + (2 * d) *_{\mathbb{R}} y) \text{ ' cbox } 0 \text{ One}$

0 One

by (*auto intro: image_eqI[where $x = \text{inverse } (2 * d) *_{\mathbb{R}} (y - (x - ?d))$]*)

next

fix y

assume $y \in (\lambda y. x - ?d + (2 * d) *_{\mathbb{R}} y) \text{ ' cbox } 0 \text{ One}$

then obtain z **where** $z: z \in \text{cbox } 0 \text{ One } y = x - ?d + (2 * d) *_{\mathbb{R}} z$

by *auto*

then show $y \in \text{cbox } (x - ?d) (x + ?d)$

using z *assms* **by** (*auto simp: mem_box inner_simps*)

qed

obtain S **where** *finite* S $\text{cbox } 0 (\sum \text{Basis} :: 'a) = \text{convex hull } S$

using *unit_cube_convex_hull* **by** *auto*

then show *?thesis*

by (*rule_tac that[of $(\lambda y. x - ?d + (2 * d) *_{\mathbb{R}} y) \text{ ' } S$]*) (*auto simp: convex_hull_affinity **)

qed

5.3.12 Representation of any interval as a finite convex hull

lemma *image_stretch_interval*:

$(\lambda x. \sum_{k \in \text{Basis}} (m k * (x \cdot k)) *_{\mathbb{R}} k) \text{ ' cbox } a (b :: 'a :: euclidean_space) =$

$(\text{if } (\text{cbox } a b) = \{\} \text{ then } \{\} \text{ else}$

$\text{cbox } (\sum_{k \in \text{Basis}} (\min (m k * (a \cdot k)) (m k * (b \cdot k))) *_{\mathbb{R}} k :: 'a)$

$(\sum_{k \in \text{Basis}} (\max (m k * (a \cdot k)) (m k * (b \cdot k))) *_{\mathbb{R}} k))$

proof *cases*

assume $*$: $\text{cbox } a b \neq \{\}$

show *?thesis*

unfolding *box_ne_empty if_not_P[OF *]*

apply (*simp add: cbox_def image_Collect set_eq_iff euclidean_eq_iff[where $'a = 'a$]* *ball_conj_distrib[symmetric]*)

apply (*subst choice_Basis_iff[symmetric]*)

proof (*intro allI ball_cong refl*)

fix $x i :: 'a$ **assume** $i \in \text{Basis}$

with $*$ **have** $a \cdot i \leq b \cdot i$

```

unfolding box_ne_empty by auto
show  $(\exists xa. x \cdot i = m\ i * xa \wedge a \cdot i \leq xa \wedge xa \leq b \cdot i) \longleftrightarrow$ 
 $\min (m\ i * (a \cdot i)) (m\ i * (b \cdot i)) \leq x \cdot i \wedge x \cdot i \leq \max (m\ i * (a \cdot i)) (m$ 
 $i * (b \cdot i))$ 
proof (cases  $m\ i = 0$ )
  case True
    with a_le_b show ?thesis by auto
  next
    case False
      then have *:  $\bigwedge a\ b. a = m\ i * b \longleftrightarrow b = a / m\ i$ 
        by (auto simp: field_simps)
      from False have
         $\min (m\ i * (a \cdot i)) (m\ i * (b \cdot i)) = (\text{if } 0 < m\ i \text{ then } m\ i * (a \cdot i) \text{ else } m$ 
 $i * (b \cdot i))$ 
         $\max (m\ i * (a \cdot i)) (m\ i * (b \cdot i)) = (\text{if } 0 < m\ i \text{ then } m\ i * (b \cdot i) \text{ else } m$ 
 $i * (a \cdot i))$ 
        using a_le_b by (auto simp: min_def max_def mult_le_cancel_left)
        with False show ?thesis using a_le_b *
        by (simp add: le_divide_eq divide_le_eq) (simp add: ac_simps)
      qed
    qed
qed simp

```

lemma interval_image_stretch_interval:

```

 $\exists u\ v. (\lambda x. \sum k \in \text{Basis}. (m\ k * (x \cdot k)) *_{\mathbb{R}} k) \text{ ` } \text{cbox } a\ (b::'a::\text{euclidean\_space}) =$ 
 $\text{cbox } u\ (v::'a::\text{euclidean\_space})$ 
unfolding image_stretch_interval by auto

```

lemma cbox_translation: $\text{cbox } (c + a)\ (c + b) = \text{image } (\lambda x. c + x)\ (\text{cbox } a\ b)$

```

using image_affinity_cbox [of 1 c a b]
using box_ne_empty [of a+c b+c] box_ne_empty [of a b]
by (auto simp: inner_left_distrib add.commute)

```

lemma cbox_image_unit_interval:

```

fixes a :: 'a::euclidean_space
assumes cbox a b  $\neq \{\}$ 
shows  $\text{cbox } a\ b =$ 
 $(+) a \text{ ` } (\lambda x. \sum k \in \text{Basis}. ((b \cdot k - a \cdot k) * (x \cdot k)) *_{\mathbb{R}} k) \text{ ` } \text{cbox } 0\ \text{One}$ 

```

using assms

```

apply (simp add: box_ne_empty image_stretch_interval cbox_translation [symmetric])
apply (simp add: min_def max_def algebra_simps sum_subtractf euclidean_representation)
done

```

lemma closed_interval_as_convex_hull:

```

fixes a :: 'a::euclidean_space
obtains S where finite S  $\text{cbox } a\ b = \text{convex hull } S$ 

```

proof (cases $\text{cbox } a\ b = \{\}$)

```

  case True with convex_hull_empty that show ?thesis
  by blast

```

```

next
case False
obtain  $S :: 'a$  set where finite  $S$  and eq:  $\text{cbox } 0 \text{ One} = \text{convex hull } S$ 
  by (blast intro: unit_cube_convex_hull)
let  $?S = ((+) a \text{ ' } (\lambda x. \sum_{k \in \text{Basis}} ((b \cdot k - a \cdot k) * (x \cdot k)) *_R k) \text{ ' } S)$ 
show thesis
proof
  show finite  $?S$ 
  by (simp add: ⟨finite  $S$ ⟩)
  have lin: linear  $(\lambda x. \sum_{k \in \text{Basis}} ((b \cdot k - a \cdot k) * (x \cdot k)) *_R k)$ 
  by (rule linear_compose_sum) (auto simp: algebra_simps linearI)
  show  $\text{cbox } a \text{ b} = \text{convex hull } ?S$ 
  using convex_hull_linear_image [OF lin]
  by (simp add: convex_hull_translation eq cbox_image_unit_interval [OF
False])
qed
qed

```

5.3.13 Bounded convex function on open set is continuous

lemma convex_on_bounded_continuous:

fixes $S :: ('a :: \text{real_normed_vector})$ set

assumes open S

and f : convex_on S f

and $\forall x \in S. |f x| \leq b$

shows continuous_on S f

proof -

have $\exists d > 0. \forall x'. \text{norm } (x' - x) < d \longrightarrow |f x' - f x| < e$ if $x \in S$ $e > 0$ for x
and $e :: \text{real}$

proof -

define B where $B = |b| + 1$

then have B : $0 < B \wedge \forall x. x \in S \implies |f x| \leq B$

using assms(3) by auto

obtain k where $k > 0$ and k : $\text{cball } x \text{ k} \subseteq S$

using ⟨ $x \in S$ ⟩ assms(1) open_contains_cball_eq by blast

show $\exists d > 0. \forall x'. \text{norm } (x' - x) < d \longrightarrow |f x' - f x| < e$

proof (intro exI conjI allI impI)

fix y

assume as: $\text{norm } (y - x) < \min (k / 2) (e / (2 * B) * k)$

show $|f y - f x| < e$

proof (cases $y = x$)

case False

define t where $t = k / \text{norm } (y - x)$

have $2 < t < 0 < t$

unfolding t_def using as False and ⟨ $k > 0$ ⟩

by (auto simp: field_simps)

have $y \in S$

apply (rule $k[\text{THEN subsetD}]$)

unfolding mem_cball dist_norm

```

    apply (rule order_trans[of _ 2 * norm (x - y)])
    using as
  by (auto simp: field_simps norm_minus_commute)
{
  define w where w = x + t *R (y - x)
  have w ∈ S
    using ⟨k>0 by (auto simp: dist_norm t_def w_def k[THEN subsetD])
  have (1 / t) *R x + - x + ((t - 1) / t) *R x = (1 / t - 1 + (t - 1) /
t) *R x
    by (auto simp: algebra_simps)
  also have ... = 0
    using ⟨t > 0 by (auto simp: field_simps)
  finally have w: (1 / t) *R w + ((t - 1) / t) *R x = y
    unfolding w_def using False and ⟨t > 0 by
    by (auto simp: algebra_simps)
  have 2: 2 * B < e * t
    unfolding t_def using ⟨0 < e⟩ ⟨0 < k⟩ ⟨B > 0 and as and False
    by (auto simp: field_simps)
  have f y - f x ≤ (f w - f x) / t
    using convex_onD [OF f, of (t - 1)/t w x] ⟨0 < t⟩ ⟨2 < t⟩ ⟨x ∈ S⟩ ⟨w
∈ S⟩
    by (simp add: w field_simps)
  also have ... < e
    using B(2)[OF ⟨w ∈ S⟩] and B(2)[OF ⟨x ∈ S⟩] 2 ⟨t > 0 by (auto simp:
field_simps)
  finally have th1: f y - f x < e .
}
}
moreover
{
  define w where w = x - t *R (y - x)
  have w ∈ S
    using ⟨k > 0 by (auto simp: dist_norm t_def w_def k[THEN subsetD])
  have (1 / (1 + t)) *R x + (t / (1 + t)) *R x = (1 / (1 + t) + t / (1 +
t)) *R x
    by (auto simp: algebra_simps)
  also have ... = x
    using ⟨t > 0 by (auto simp: field_simps)
  finally have w: (1 / (1 + t)) *R w + (t / (1 + t)) *R y = x
    unfolding w_def using False and ⟨t > 0 by
    by (auto simp: algebra_simps)
  have 2 * B < e * t
    unfolding t_def
    using ⟨0 < e⟩ ⟨0 < k⟩ ⟨B > 0 and as and False
    by (auto simp: field_simps)
  then have *: (f w - f y) / t < e
    using B(2)[OF ⟨w ∈ S⟩] and B(2)[OF ⟨y ∈ S⟩]
    using ⟨t > 0 by
    by (auto simp: field_simps)
  have f x ≤ 1 / (1 + t) * f w + (t / (1 + t)) * f y

```

```

    using convex_onD [OF f, of t / (1+t) w y] ⟨0 < t⟩ ⟨2 < t⟩ ⟨y ∈ S⟩ ⟨w
∈ S⟩
    by (simp add: w field_simps)
    also have ... = (f w + t * f y) / (1 + t)
    using ⟨t > 0⟩ by (simp add: add_divide_distrib)
    also have ... < e + f y
    using ⟨t > 0⟩ * ⟨e > 0⟩ by (auto simp: field_simps)
    finally have f x - f y < e by auto
  }
  ultimately show ?thesis by auto
qed (use ⟨0 < e⟩ in auto)
qed (use ⟨0 < e⟩ ⟨0 < k⟩ ⟨0 < B⟩ in ⟨auto simp: field_simps⟩)
qed
then show ?thesis
  by (metis continuous_on_iff dist_norm real_norm_def)
qed

```

5.3.14 Upper bound on a ball implies upper and lower bounds

```

lemma convex_bounds_lemma:
  fixes x :: 'a::real_normed_vector
  assumes f: convex_on (cball x e) f
    and b:  $\bigwedge y. y \in \text{cball } x \ e \implies f \ y \leq b$  and y:  $y \in \text{cball } x \ e$ 
  shows  $|f \ y| \leq b + 2 * |f \ x|$ 
proof (cases  $0 \leq e$ )
case True
  define z where  $z = 2 *_{\mathbb{R}} x - y$ 
  have *:  $x - (2 *_{\mathbb{R}} x - y) = y - x$ 
  by (simp add: scaleR_2)
  have z:  $z \in \text{cball } x \ e$ 
  using y unfolding z_def mem_cball dist_norm * by (auto simp: norm_minus_commute)
  have  $(1 / 2) *_{\mathbb{R}} y + (1 / 2) *_{\mathbb{R}} z = x$ 
  unfolding z_def by (auto simp: algebra_simps)
  then show  $|f \ y| \leq b + 2 * |f \ x|$ 
  using convex_onD [OF f, of 1/2 y z] b[OF y] b y z
  by (fastforce simp add: field_simps)
next
case False
  have  $\text{dist } x \ y < 0$ 
  using False y unfolding mem_cball not_le by (auto simp del: dist_not_less_zero)
  then show  $|f \ y| \leq b + 2 * |f \ x|$ 
  using zero_le_dist[of x y] by auto
qed

```

Hence a convex function on an open set is continuous

```

lemma real_of_nat_ge_one_iff:  $1 \leq \text{real } (n::\text{nat}) \iff 1 \leq n$ 
  by auto

```

```

lemma convex_on_continuous:

```



```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $open\ S\ convex\_on\ S\ f$ 
shows  $continuous\_on\ S\ f$ 
unfolding  $continuous\_on\_eq\_continuous\_at[OF\ \langle open\ S \rangle]$ 
proof
  note  $dimge1 = DIM\_positive[where\ 'a='a]$ 
  fix  $x$ 
  assume  $x \in S$ 
  then obtain  $e$  where  $e: cball\ x\ e \subseteq S\ e > 0$ 
    using  $assms(1)$  unfolding  $open\_contains\_cball$  by  $auto$ 
  define  $d$  where  $d = e / real\ DIM('a)$ 
  have  $0 < d$ 
    unfolding  $d\_def$  using  $\langle e > 0 \rangle\ dimge1$  by  $auto$ 
  let  $?d = (\sum\ i \in Basis.\ d *_{\mathbb{R}}\ i)::'a$ 
  obtain  $c$ 
    where  $c: finite\ c$  and  $c1: convex\ hull\ c \subseteq cball\ x\ e$  and  $c2: cball\ x\ d \subseteq convex$ 
     $hull\ c$ 
  proof
    define  $c$  where  $c = (\sum\ i \in Basis.\ (\lambda a.\ a *_{\mathbb{R}}\ i) \cdot \{x \cdot i - d,\ x \cdot i + d\})$ 
    show  $finite\ c$ 
      unfolding  $c\_def$  by  $(simp\ add: finite\_set\_sum)$ 
    have  $\bigwedge i. i \in Basis \implies convex\ hull\ \{x \cdot i - d,\ x \cdot i + d\} = cbox\ (x \cdot i - d)$ 
     $(x \cdot i + d)$ 
      using  $\langle 0 < d \rangle\ convex\_hull\_eq\_real\_cbox$  by  $auto$ 
    then have  $1: convex\ hull\ c = \{a. \forall i \in Basis.\ a \cdot i \in cbox\ (x \cdot i - d)\ (x \cdot i +$ 
     $d)\}$ 
      unfolding  $box\_eq\_set\_sum\_Basis\ c\_def\ convex\_hull\_set\_sum$ 
      apply  $(subst\ convex\_hull\_linear\_image\ [symmetric])$ 
      by  $(force\ simp\ add: linear\_iff\ scaleR\_add\_left)+$ 
    then have  $2: convex\ hull\ c = \{a. \forall i \in Basis.\ a \cdot i \in cball\ (x \cdot i)\ d\}$ 
      by  $(simp\ add: dist\_norm\ abs\_le\_iff\ algebra\_simps)$ 
    show  $cball\ x\ d \subseteq convex\ hull\ c$ 
      unfolding  $2$ 
      by  $(clarsimp\ simp: dist\_norm)\ (metis\ inner\_commute\ inner\_diff\_right$ 
     $norm\_bound\_Basis\_le)$ 
    have  $e': e = (\sum\ (i::'a) \in Basis.\ d)$ 
      by  $(simp\ add: d\_def)$ 
    show  $convex\ hull\ c \subseteq cball\ x\ e$ 
      unfolding  $2$ 
    proof  $clarsimp$ 
      show  $dist\ x\ y \leq e$  if  $\forall i \in Basis.\ dist\ (x \cdot i)\ (y \cdot i) \leq d$  for  $y$ 
      proof  $-$ 
        have  $\bigwedge i. i \in Basis \implies 0 \leq dist\ (x \cdot i)\ (y \cdot i)$ 
          by  $simp$ 
        have  $(\sum\ i \in Basis.\ dist\ (x \cdot i)\ (y \cdot i)) \leq e$ 
          using  $e'\ sum\_mono$  that by  $fastforce$ 
        then show  $?thesis$ 
          by  $(metis\ (mono\_tags)\ euclidean\_dist\_l2\ order\_trans\ [OF\ L2\_set\_le\_sum]$ 
     $zero\_le\_dist)$ 

```

```

      qed
    qed
  qed
  define k where k = Max (f ' c)
  have convex_on (convex hull c) f
    using assms(2) c1 convex_on_subset e(1) by blast
  then have k:  $\forall y \in \text{convex hull } c. f y \leq k$ 
    using c convex_on_convex_hull_bound k_def by fastforce
  have  $e \leq e * \text{real DIM('a)}$ 
    using e(2) real_of_nat_ge_one_iff by auto
  then have  $d \leq e$ 
    by (simp add: d_def field_split_simps)
  then have dsube:  $\text{cball } x \ d \subseteq \text{cball } x \ e$ 
    by (rule subset_cball)
  have conv: convex_on (cball x d) f
    using <convex_on (convex hull c) f> c2 convex_on_subset by blast
  then have  $\bigwedge y. y \in \text{cball } x \ d \implies |f y| \leq k + 2 * |f x|$ 
    by (rule convex_bounds_lemma) (use c2 k in blast)
  moreover have convex_on (ball x d) f
    using conv convex_on_subset by fastforce
  ultimately
  have continuous_on (ball x d) f
    by (metis convex_on_bounded_continuous Elementary_Metric_Spaces.open_ball
    mem_ball_imp_mem_cball)
  then show continuous (at x) f
    unfolding continuous_on_eq_continuous_at[OF open_ball]
    using <d > 0> by auto
qed
end

```

Chapter 6

Unsorted

theory *Starlike*

imports

Convex_Euclidean_Space

Line_Segment

begin

lemma *affine_hull_closed_segment* [*simp*]:

$\text{affine hull } (\text{closed_segment } a \ b) = \text{affine hull } \{a, b\}$

by (*simp add: segment_convex_hull*)

lemma *affine_hull_open_segment* [*simp*]:

fixes $a :: 'a::\text{euclidean_space}$

shows $\text{affine hull } (\text{open_segment } a \ b) = (\text{if } a = b \text{ then } \{\} \text{ else } \text{affine hull } \{a, b\})$

by (*metis affine_hull_convex_hull affine_hull_empty closure_open_segment closure_same_affine_hull segment_convex_hull*)

lemma *rel_interior_closure_convex_segment*:

fixes $S :: _::\text{euclidean_space set}$

assumes $\text{convex } S \ a \in \text{rel_interior } S \ b \in \text{closure } S$

shows $\text{open_segment } a \ b \subseteq \text{rel_interior } S$

proof

fix x

have [*simp*]: $(1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b = b - (1 - u) *_{\mathbb{R}} (b - a)$ **for** u

by (*simp add: algebra_simps*)

assume $x \in \text{open_segment } a \ b$

then show $x \in \text{rel_interior } S$

unfolding *closed_segment_def open_segment_def* **using** *assms*

by (*auto intro: rel_interior_closure_convex_shrink*)

qed

lemma *convex_hull_insert_segments*:

$\text{convex hull } (\text{insert } a \ S) =$

$(\text{if } S = \{\} \text{ then } \{a\} \text{ else } \bigcup x \in \text{convex hull } S. \text{closed_segment } a \ x)$

by (*force simp add: convex_hull_insert_alt in_segment*)

```

lemma Int_convex_hull_insert_rel_exterior:
  fixes z :: 'a::euclidean_space
  assumes convex C T  $\subseteq$  C and z: z  $\in$  rel_interior C and dis: disjnt S (rel_interior C)
  shows S  $\cap$  (convex hull (insert z T)) = S  $\cap$  (convex hull T) (is ?lhs = ?rhs)
proof
  have *: T = {}  $\implies$  z  $\notin$  S
  using dis z by (auto simp add: disjnt_def)
  { fix x y
    assume x  $\in$  S and y: y  $\in$  convex hull T and x  $\in$  closed_segment z y
    have y  $\in$  closure C
      by (metis y  $\langle$ convex C $\rangle$   $\langle$ T  $\subseteq$  C $\rangle$  closure_subset contra_subsetD convex_hull_eq hull_mono)
    moreover have x  $\notin$  rel_interior C
      by (meson  $\langle$ x  $\in$  S $\rangle$  dis disjnt_iff)
    moreover have x  $\in$  open_segment z y  $\cup$  {z, y}
      using  $\langle$ x  $\in$  closed_segment z y $\rangle$  closed_segment_eq_open by blast
    ultimately have x  $\in$  convex hull T
      using rel_interior_closure_convex_segment [OF  $\langle$ convex C $\rangle$  z]
      using y z by blast
  }
  with * show ?lhs  $\subseteq$  ?rhs
  by (auto simp add: convex_hull_insert_segments)
  show ?rhs  $\subseteq$  ?lhs
  by (meson hull_mono inf_mono subset_insertI subset_refl)
qed

```

6.0.1 Shrinking towards the interior of a convex set

```

lemma mem_interior_convex_shrink:
  fixes S :: 'a::euclidean_space set
  assumes convex S
  and c  $\in$  interior S
  and x  $\in$  S
  and 0 < e
  and e  $\leq$  1
  shows x - e *R (x - c)  $\in$  interior S
proof -
  obtain d where d > 0 and d: ball c d  $\subseteq$  S
  using assms(2) unfolding mem_interior by auto
  show ?thesis
  unfolding mem_interior
proof (intro exI subsetI conjI)
  fix y
  assume y  $\in$  ball (x - e *R (x - c)) (e*d)
  then have as: dist (x - e *R (x - c)) y < e * d
  by simp
  have *: y = (1 - (1 - e)) *R ((1 / e) *R y - ((1 - e) / e) *R x) + (1 - e)

```

```

*_R x
  using <e > 0> by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
  have c - ((1 / e) *_R y - ((1 - e) / e) *_R x) = (1 / e) *_R (e *_R c - y + (1
- e) *_R x)
    using <e > 0>
    by (auto simp add: euclidean_eq_iff[where 'a='a] field_simps inner_simps)
  then have dist c ((1 / e) *_R y - ((1 - e) / e) *_R x) = |1/e| * norm (e *_R c
- y + (1 - e) *_R x)
    by (simp add: dist_norm)
  also have ... = |1/e| * norm (x - e *_R (x - c) - y)
    by (auto intro!: arg_cong[where f=norm] simp add: algebra_simps)
  also have ... < d
    using as[unfolded dist_norm] and <e > 0>
    by (auto simp add: pos_divide_less_eq[OF <e > 0>] mult.commute)
  finally have (1 - (1 - e)) *_R ((1 / e) *_R y - ((1 - e) / e) *_R x) + (1 -
e) *_R x ∈ S
    using assms(3-5) d
    by (intro convexD_alt [OF <convex S>]) (auto intro: convexD_alt [OF <convex
S>])
  with <e > 0> show y ∈ S
    by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
qed (use <e>0> <d>0> in auto)
qed

```

lemma mem_interior_closure_convex_shrink:

fixes $S :: 'a::euclidean_space$ set

assumes convex S

and $c \in \text{interior } S$

and $x \in \text{closure } S$

and $0 < e$

and $e \leq 1$

shows $x - e *_R (x - c) \in \text{interior } S$

proof -

obtain d where $d > 0$ and d : ball c $d \subseteq S$

using assms(2) unfolding mem_interior by auto

have $\exists y \in S. \text{norm } (y - x) * (1 - e) < e * d$

proof (cases $x \in S$)

case True

then show ?thesis

using <e > 0> <d > 0> by force

next

case False

then have x : x islimpt S

using assms(3)[unfolded closure_def] by auto

show ?thesis

proof (cases $e = 1$)

case True

obtain y where $y \in S$ $y \neq x$ dist y $x < 1$

using x [unfolded islimpt_approachable, THEN spec[where $x=1$]] by auto

```

    then show ?thesis
      using True ⟨0 < d⟩ by auto
  next
    case False
    then have 0 < e * d / (1 - e) and *: 1 - e > 0
      using ⟨e ≤ 1⟩ ⟨e > 0⟩ ⟨d > 0⟩ by auto
    then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 - e)
      using islimpt_approachable x by blast
    then have norm (y - x) * (1 - e) < e * d
      by (metis * dist_norm mult_imp_div_pos_le not_less)
    then show ?thesis
      using ⟨y ∈ S⟩ by blast
  qed
qed
then obtain y where y ∈ S and y: norm (y - x) * (1 - e) < e * d
  by auto
define z where z = c + ((1 - e) / e) *R (x - y)
have *: x - e *R (x - c) = y - e *R (y - z)
  unfolding z_def using ⟨e > 0⟩
by (auto simp add: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)
have (1 - e) * norm (x - y) / e < d
  using y ⟨0 < e⟩ by (simp add: field_simps norm_minus_commute)
then have z ∈ interior (ball c d)
  using ⟨0 < e⟩ ⟨e ≤ 1⟩ by (simp add: interior_open[OF open_ball] z_def
dist_norm)
then have z ∈ interior S
  using d interiorI interior_ball by blast
then show ?thesis
  unfolding * using mem_interior_convex_shrink ⟨y ∈ S⟩ assms by blast
qed

lemma in_interior_closure_convex_segment:
  fixes S :: 'a::euclidean_space set
  assumes convex S and a: a ∈ interior S and b: b ∈ closure S
  shows open_segment a b ⊆ interior S
proof -
  { fix u::real
    assume u: 0 < u u < 1
    have (1 - u) *R a + u *R b = b - (1 - u) *R (b - a)
      by (simp add: algebra_simps)
    also have ... ∈ interior S using mem_interior_closure_convex_shrink [OF
assms] u
      by simp
    finally have (1 - u) *R a + u *R b ∈ interior S .
  }
then show ?thesis
  by (clarsimp simp: in_segment)
qed

```

```

lemma convex_closure_interior:
  fixes S :: 'a::euclidean_space set
  assumes convex S and int: interior S  $\neq$  {}
  shows closure(interior S) = closure S
proof -
  obtain a where a: a  $\in$  interior S
  using int by auto
  have closure S  $\subseteq$  closure(interior S)
proof
  fix x
  assume x: x  $\in$  closure S
  show x  $\in$  closure (interior S)
proof (cases x=a)
  case True
  then show ?thesis
  using  $\langle a \in \text{interior } S \rangle$  closure_subset by blast
next
  case False
  { fix e::real
  assume xnotS: x  $\notin$  interior S and 0 < e
  have  $\exists x' \in \text{interior } S. x' \neq x \wedge \text{dist } x' x < e$ 
  proof (intro bexI conjI)
    show x - min (e/2 / norm (x - a)) 1 *R (x - a)  $\neq$  x
    using False  $\langle 0 < e \rangle$  by (auto simp: algebra_simps min_def)
    show dist (x - min (e/2 / norm (x - a)) 1 *R (x - a)) x < e
    using  $\langle 0 < e \rangle$  by (auto simp: dist_norm min_def)
    show x - min (e/2 / norm (x - a)) 1 *R (x - a)  $\in$  interior S
    using  $\langle 0 < e \rangle$  False
  by (auto simp add: min_def a intro: mem_interior_closure_convex_shrink
[OF  $\langle \text{convex } S \rangle$  a x])
  qed
}
  then show ?thesis
  by (auto simp add: closure_def islimpt_approachable)
  qed
  qed
  then show ?thesis
  by (simp add: closure_mono interior_subset subset_antisym)
  qed

```

lemma openin_subset_relative_interior:

```

  fixes S :: 'a::euclidean_space set
  shows openin (top_of_set (affine hull T)) S  $\implies$  (S  $\subseteq$  rel_interior T) = (S  $\subseteq$ 
T)
  by (meson order.trans rel_interior_maximal rel_interior_subset)

```

lemma conic_hull_eq_span_affine_hull:

```

  fixes S :: 'a::euclidean_space set
  assumes 0  $\in$  rel_interior S

```

```

shows conic hull S = span S ∧ conic hull S = affine hull S
proof -
  obtain ε where ε > 0 and ε: cball 0 ε ∩ affine hull S ⊆ S
  using assms mem_rel_interior_cball by blast
  have *: affine hull S = span S
  by (meson affine_hull_span_0 assms hull_inc mem_rel_interior_cball)
  moreover
  have conic hull S ⊆ span S
  by (simp add: hull_minimal span_superset)
  moreover
  { fix x
    assume x ∈ affine hull S
    have x ∈ conic hull S
    proof (cases x=0)
      case True
      then show ?thesis
      using ⟨x ∈ affine hull S⟩ by auto
    next
      case False
      then have (ε / norm x) *R x ∈ cball 0 ε ∩ affine hull S
      using ⟨0 < ε⟩ ⟨x ∈ affine hull S⟩ * span_mul by fastforce
      then have (ε / norm x) *R x ∈ S
      by (meson ε subsetD)
      then have ∃ c xa. x = c *R xa ∧ 0 ≤ c ∧ xa ∈ S
      by (smt (verit, del_insts) ⟨0 < ε⟩ divide_nonneg_nonneg_eq_vector_fraction_iff
norm_eq_zero norm_ge_zero)
      then show ?thesis
      by (simp add: conic_hull_explicit)
    qed
  }
  then have affine hull S ⊆ conic hull S
  by auto
  ultimately show ?thesis
  by blast
qed

```

```

lemma conic_hull_eq_span:
  fixes S :: 'a::euclidean_space set
  assumes 0 ∈ rel_interior S
  shows conic hull S = span S
  by (simp add: assms conic_hull_eq_span_affine_hull)

```

```

lemma conic_hull_eq_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes 0 ∈ rel_interior S
  shows conic hull S = affine hull S
  using assms conic_hull_eq_span_affine_hull by blast

```

```

lemma conic_hull_eq_span_eq:

```



```

fixes  $S :: 'a::euclidean\_space\ set$ 
shows  $0 \in \text{rel\_interior}(\text{conic hull } S) \longleftrightarrow \text{conic hull } S = \text{span } S$  (is ?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (metis conic_hull_eq_span conic_span hull_hull hull_minimal hull_subset span_eq)
  show ?rhs  $\implies$  ?lhs
    by (metis rel_interior_affine subspace_affine subspace_span)
qed

```

```

lemma aff_dim_psubset:
   $(\text{affine hull } S) \subset (\text{affine hull } T) \implies \text{aff\_dim } S < \text{aff\_dim } T$ 
  by (metis aff_dim_affine_hull aff_dim_empty aff_dim_subset affine_affine_hull affine_dim_equal order_less_le)

```

```

lemma aff_dim_eq_full_gen:
   $S \subseteq T \implies (\text{aff\_dim } S = \text{aff\_dim } T \longleftrightarrow \text{affine hull } S = \text{affine hull } T)$ 
  by (smt (verit, del_insts) aff_dim_affine_hull2 aff_dim_psubset hull_mono psubsetI)

```

```

lemma aff_dim_eq_full:
  fixes  $S :: 'n::euclidean\_space\ set$ 
  shows  $\text{aff\_dim } S = (\text{DIM } 'n) \longleftrightarrow \text{affine hull } S = \text{UNIV}$ 
  by (metis aff_dim_UNIV aff_dim_affine_hull affine_hull_UNIV)

```

```

lemma closure_convex_Int_superset:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes  $\text{convex } S$   $\text{interior } S \neq \{\}$   $\text{interior } S \subseteq \text{closure } T$ 
  shows  $\text{closure}(S \cap T) = \text{closure } S$ 
proof -
  have  $\text{closure } S \subseteq \text{closure}(\text{interior } S)$ 
    by (simp add: convex_closure_interior assms)
  also have  $\dots \subseteq \text{closure}(S \cap T)$ 
    using interior_subset [of  $S$ ] assms
  by (metis (no_types, lifting) Int_assoc Int_lower2 closure_mono closure_open_Int_superset inf.orderE open_interior)
  finally show ?thesis
    by (simp add: closure_mono dual_order.antisym)
qed

```

6.0.2 Some obvious but surprisingly hard simplex lemmas

```

lemma simplex:
  assumes  $\text{finite } S$ 
  and  $0 \notin S$ 
  shows  $\text{convex hull } (\text{insert } 0\ S) = \{y. \exists u. (\forall x \in S. 0 \leq u\ x) \wedge \text{sum } u\ S \leq 1 \wedge \text{sum } (\lambda x. u\ x *_{\mathbb{R}} x)\ S = y\}$ 
proof -
  { fix  $x$  and  $u :: 'a \Rightarrow \text{real}$ 

```

```

    assume  $\forall x \in S. 0 \leq u \ x \ \text{sum } u \ S \leq 1$ 
    then have  $\exists v. 0 \leq v \ 0 \wedge (\forall x \in S. 0 \leq v \ x) \wedge v \ 0 + \text{sum } v \ S = 1 \wedge (\sum x \in S. v \ x \ *_{\mathbb{R}} \ x) = (\sum x \in S. u \ x \ *_{\mathbb{R}} \ x)$ 
      by (rule_tac  $x = \lambda x. \text{if } x = 0 \text{ then } 1 - \text{sum } u \ S \text{ else } u \ x$  in exI) (auto simp:
sum_delta_notmem assms if_smult)
    }
    then show ?thesis by (auto simp: convex_hull_finite_set_eq_iff assms)
qed

```

lemma *substd_simplex*:

assumes $d: d \subseteq \text{Basis}$

shows $\text{convex hull } (\text{insert } 0 \ d) =$

$\{x. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge (\sum i \in d. x \cdot i) \leq 1 \wedge (\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0)\}$

(**is** $\text{convex hull } (\text{insert } 0 \ ?p) = ?s$)

proof –

let $?D = d$

have $0 \notin ?p$

using *assms* **by** (auto simp: image_def)

from d **have** *finite* d

by (blast intro: finite_subset finite_Basis)

show ?thesis

unfolding *simplex*[OF $\langle \text{finite } d \rangle \langle 0 \notin ?p \rangle$]

proof (intro set_eqI; safe)

fix $u :: 'a \Rightarrow \text{real}$

assume *as*: $\forall x \in ?D. 0 \leq u \ x \ \text{sum } u \ ?D \leq 1$

let $?x = (\sum x \in ?D. u \ x \ *_{\mathbb{R}} \ x)$

have *ind*: $\forall i \in \text{Basis}. i \in d \longrightarrow u \ i = ?x \cdot i$

and *notind*: $(\forall i \in \text{Basis}. i \notin d \longrightarrow ?x \cdot i = 0)$

using *substdbasis_expansion_unique*[OF *assms*] **by** *blast+*

then have ****: $\text{sum } u \ ?D = \text{sum } ((\cdot) \ ?x) \ ?D$

using *assms* **by** (auto intro!: sum.cong)

show $0 \leq ?x \cdot i$ **if** $i \in \text{Basis}$ **for** i

using *as*(1) *ind* *notind* **that** **by** *fastforce*

show $\text{sum } ((\cdot) \ ?x) \ ?D \leq 1$

using **** *as*(2) **by** *linarith*

show $?x \cdot i = 0$ **if** $i \in \text{Basis}$ $i \notin d$ **for** i

using *notind* **that** **by** *blast*

next

fix x

assume $\forall i \in \text{Basis}. 0 \leq x \cdot i \ \text{sum } ((\cdot) \ x) \ ?D \leq 1 \ (\forall i \in \text{Basis}. i \notin d \longrightarrow x \cdot i = 0)$

with d **show** $\exists u. (\forall x \in ?D. 0 \leq u \ x) \wedge \text{sum } u \ ?D \leq 1 \wedge (\sum x \in ?D. u \ x \ *_{\mathbb{R}} \ x) = x$

unfolding *substdbasis_expansion_unique*[OF *assms*]

by (rule_tac $x = \text{inner } x$ in exI) *auto*

qed

qed

lemma *std_simplex*:

```

convex hull (insert 0 Basis) =
  {x::'a::euclidean_space. (∀i∈Basis. 0 ≤ x·i) ∧ sum (λi. x·i) Basis ≤ 1}
using substd_simplex[of Basis] by auto

lemma interior_std_simplex:
  interior (convex hull (insert 0 Basis)) =
    {x::'a::euclidean_space. (∀i∈Basis. 0 < x·i) ∧ sum (λi. x·i) Basis < 1}
  unfolding set_eq_iff mem_interior_std_simplex
proof (intro allI iffI CollectI; clarify)
  fix x :: 'a
  fix e
  assume e > 0 and as: ball x e ⊆ {x. (∀i∈Basis. 0 ≤ x · i) ∧ sum ((·) x) Basis
≤ 1}
  show (∀i∈Basis. 0 < x · i) ∧ sum ((·) x) Basis < 1
  proof safe
    fix i :: 'a
    assume i: i ∈ Basis
    then show 0 < x · i
      using as[THEN subsetD[where c=x - (e/2) *R i]] and ⟨e > 0⟩
      by (force simp add: inner_simps)
  next
  have **: dist x (x + (e/2) *R (SOME i. i∈Basis)) < e using ⟨e > 0⟩
  unfolding dist_norm
  by (auto intro!: mult_strict_left_mono simp: SOME_Basis)
  have ∧i. i ∈ Basis ⇒ (x + (e/2) *R (SOME i. i∈Basis)) · i =
    x·i + (if i = (SOME i. i∈Basis) then e/2 else 0)
  by (auto simp: SOME_Basis inner_Basis inner_simps)
  then have *: sum ((·) (x + (e/2) *R (SOME i. i∈Basis))) Basis =
    sum (λi. x·i + (if (SOME i. i∈Basis) = i then e/2 else 0)) Basis
  by (auto simp: intro!: sum.cong)
  have sum ((·) x) Basis < sum ((·) (x + (e/2) *R (SOME i. i∈Basis))) Basis
  using ⟨e > 0⟩ DIM_positive by (auto simp: SOME_Basis sum.distrib *)
  also have ... ≤ 1
  using ** as by force
  finally show sum ((·) x) Basis < 1 by auto
qed
next
fix x :: 'a
assume as: ∀i∈Basis. 0 < x · i sum ((·) x) Basis < 1
obtain a :: 'b where a ∈ UNIV using UNIV_witness ..
let ?d = (1 - sum ((·) x) Basis) / real (DIM('a))
show ∃e>0. ball x e ⊆ {x. (∀i∈Basis. 0 ≤ x · i) ∧ sum ((·) x) Basis ≤ 1}
  proof (rule_tac x=min (Min (((·) x) ' Basis)) D for D in exI, intro conjI
subsetI CollectI)
    fix y
    assume y: y ∈ ball x (min (Min (((·) x) ' Basis)) ?d)
    have sum ((·) y) Basis ≤ sum (λi. x·i + ?d) Basis
  proof (rule sum_mono)
    fix i :: 'a

```

```

assume  $i: i \in \text{Basis}$ 
have  $|y \cdot i - x \cdot i| \leq \text{norm } (y - x)$ 
  by (metis Basis_le_norm i inner_commute inner_diff_right)
also have  $\dots < ?d$ 
  using  $y$  by (simp add: dist_norm norm_minus_commute)
finally have  $|y \cdot i - x \cdot i| < ?d$  .
then show  $y \cdot i \leq x \cdot i + ?d$  by auto
qed
also have  $\dots \leq 1$ 
  unfolding sum.distrib sum_constant
  by (auto simp add: Suc_le_eq)
finally show  $\text{sum } ((\cdot) y) \text{Basis} \leq 1$  .
show  $(\forall i \in \text{Basis}. 0 \leq y \cdot i)$ 
proof safe
  fix  $i :: 'a$ 
  assume  $i: i \in \text{Basis}$ 
  have  $\text{norm } (x - y) < \text{Min } (((\cdot) x) \text{'Basis})$ 
    using  $y$  by (auto simp: dist_norm less_eq_real_def)
  also have  $\dots \leq x \cdot i$ 
    using  $i$  by auto
  finally have  $\text{norm } (x - y) < x \cdot i$  .
  then show  $0 \leq y \cdot i$ 
    using Basis_le_norm[OF  $i$ , of  $x - y$ ] and as(1)[rule_format, OF  $i$ ]
    by (auto simp: inner_simps)
  qed
next
  have  $\text{Min } (((\cdot) x) \text{'Basis}) > 0$ 
    using as by simp
  moreover have  $?d > 0$ 
    using as by (auto simp: Suc_le_eq)
  ultimately show  $0 < \min (\text{Min } ((\cdot) x \text{'Basis})) ((1 - \text{sum } ((\cdot) x) \text{Basis}) /$ 
real DIM('a))
    by linarith
  qed
qed

lemma interior_std_simplex_nonempty:
obtains  $a :: 'a :: \text{euclidean\_space}$  where
   $a \in \text{interior}(\text{convex hull } (\text{insert } 0 \text{Basis}))$ 
proof -
  let  $?D = \text{Basis} :: 'a \text{ set}$ 
  let  $?a = \text{sum } (\lambda b :: 'a. \text{inverse } (2 * \text{real DIM}('a)) *_{\mathbb{R}} b) \text{Basis}$ 
  {
    fix  $i :: 'a$ 
    assume  $i: i \in \text{Basis}$ 
    have  $?a \cdot i = \text{inverse } (2 * \text{real DIM}('a))$ 
      by (rule trans[of  $\text{sum } (\lambda j. \text{if } i = j \text{ then } \text{inverse } (2 * \text{real DIM}('a)) \text{ else } 0)$ 
       $?D$ ])
      (simp_all add: sum.If_cases  $i$ ) }

```

```

note ** = this
show ?thesis
proof
  show ?a ∈ interior(convex hull (insert 0 Basis))
    unfolding interior_std_simplex mem_Collect_eq
  proof safe
    fix i :: 'a
    assume i: i ∈ Basis
    show 0 < ?a · i
      unfolding **[OF i] by (auto simp add: Suc_le_eq)
  next
    have sum ((·) ?a) ?D = sum (λi. inverse (2 * real DIM('a))) ?D
      by (auto intro: sum.cong)
    also have ... < 1
      unfolding sum_constant divide_inverse[symmetric]
      by (auto simp add: field_simps)
    finally show sum ((·) ?a) ?D < 1 by auto
  qed
qed
qed

lemma rel_interior_substd_simplex:
  assumes D: D ⊆ Basis
  shows rel_interior (convex hull (insert 0 D)) =
    {x::'a::euclidean_space. (∀ i∈D. 0 < x·i) ∧ (∑ i∈D. x·i) < 1 ∧ (∀ i∈Basis.
i ∉ D → x·i = 0)}
  (is _ = ?s)
proof -
  have finite D
    using D finite_Basis finite_subset by blast
  show ?thesis
  proof (cases D = {})
    case True
      then show ?thesis
        using rel_interior_sing using euclidean_eq_iff[of _ 0] by auto
    next
      case False
        have h0: affine hull (convex hull (insert 0 D)) =
          {x::'a::euclidean_space. (∀ i∈Basis. i ∉ D → x·i = 0)}
          using affine_hull_convex_hull affine_hull_substd_basis by auto
        have aux: ∧x::'a. ∀ i∈Basis. (∀ i∈D. 0 ≤ x·i) ∧ (∀ i∈Basis. i ∉ D → x·i =
0) → 0 ≤ x·i
          by auto
        {
          fix x :: 'a::euclidean_space
          assume x: x ∈ rel_interior (convex hull (insert 0 D))
          then obtain e where e > 0 and
            ball x e ∩ {xa. (∀ i∈Basis. i ∉ D → xa·i = 0)} ⊆ convex hull (insert 0 D)
          using mem_rel_interior_ball[of x convex hull (insert 0 D)] h0 by auto
        }
  
```

```

then have as:  $\bigwedge y. \llbracket \text{dist } x \ y < e \wedge (\forall i \in \text{Basis}. i \notin D \longrightarrow y \cdot i = 0) \rrbracket \implies$ 
               $(\forall i \in D. 0 \leq y \cdot i) \wedge \text{sum } ((\cdot) \ y) \ D \leq 1$ 
  using assms by (force simp: substd_simplex)
have x0:  $(\forall i \in \text{Basis}. i \notin D \longrightarrow x \cdot i = 0)$ 
  using x_rel_interior_subset substd_simplex[OF assms] by auto
have  $(\forall i \in D. 0 < x \cdot i) \wedge \text{sum } ((\cdot) \ x) \ D < 1 \wedge (\forall i \in \text{Basis}. i \notin D \longrightarrow x \cdot i =$ 
0)
proof (intro conjI ballI)
  fix i :: 'a
  assume i  $\in D$ 
  then have  $\forall j \in D. 0 \leq (x - (e/2) *_{\mathbb{R}} i) \cdot j$ 
    using D  $\langle e > 0 \rangle$  x0
    by (intro as[THEN conjunct1]) (force simp: dist_norm inner_simps
inner_Basis)
  then show  $0 < x \cdot i$ 
    using  $\langle e > 0 \rangle \langle i \in D \rangle D$  by (force simp: inner_simps inner_Basis)
next
obtain a where a:  $a \in D$ 
  using  $\langle D \neq \{\} \rangle$  by auto
then have **:  $\text{dist } x \ (x + (e/2) *_{\mathbb{R}} a) < e$ 
  using  $\langle e > 0 \rangle$  norm_Basis[of a] D by (auto simp: dist_norm)
have  $\bigwedge i. i \in \text{Basis} \implies (x + (e/2) *_{\mathbb{R}} a) \cdot i = x \cdot i + (\text{if } i = a \text{ then } e/2$ 
else 0)
  using a D by (auto simp: inner_simps inner_Basis)
then have *:  $\text{sum } ((\cdot) \ (x + (e/2) *_{\mathbb{R}} a)) \ D = \text{sum } (\lambda i. x \cdot i + (\text{if } a = i \text{ then } e/2$ 
e/2 else 0)) D
  using D by (intro sum.cong) auto
have a  $\in \text{Basis}$ 
  using  $\langle a \in D \rangle D$  by auto
then have h1:  $(\forall i \in \text{Basis}. i \notin D \longrightarrow (x + (e/2) *_{\mathbb{R}} a) \cdot i = 0)$ 
  using x0 D  $\langle a \in D \rangle$  by (auto simp add: inner_add_left inner_Basis)
have  $\text{sum } ((\cdot) \ x) \ D < \text{sum } ((\cdot) \ (x + (e/2) *_{\mathbb{R}} a)) \ D$ 
  using  $\langle e > 0 \rangle \langle a \in D \rangle \langle \text{finite } D \rangle$  by (auto simp add: * sum.distrib)
also have ...  $\leq 1$ 
  using ** h1 as[rule_format, of  $x + (e/2) *_{\mathbb{R}} a$ ]
  by auto
finally show  $\text{sum } ((\cdot) \ x) \ D < 1 \wedge i \in \text{Basis} \implies i \notin D \longrightarrow x \cdot i = 0$ 
  using x0 by auto
qed
}
moreover
{
  fix x :: 'a::euclidean_space
  assume as:  $x \in ?s$ 
  have  $\forall i. 0 < x \cdot i \vee 0 = x \cdot i \longrightarrow 0 \leq x \cdot i$ 
    by auto
  moreover have  $\forall i. i \in D \vee i \notin D$  by auto
  ultimately
  have  $\forall i. (\forall i \in D. 0 < x \cdot i) \wedge (\forall i. i \notin D \longrightarrow x \cdot i = 0) \longrightarrow 0 \leq x \cdot i$ 

```

```

    by metis
  then have h2:  $x \in \text{convex hull } (\text{insert } 0 D)$ 
    using as assms by (force simp add: substd_simplex)
  obtain a where a:  $a \in D$ 
    using  $\langle D \neq \{\} \rangle$  by auto
  define d where  $d \equiv (1 - \text{sum } ((\cdot) x) D) / \text{real } (\text{card } D)$ 
  have  $\exists e > 0. \text{ball } x e \cap \{x. \forall i \in \text{Basis}. i \notin D \longrightarrow x \cdot i = 0\} \subseteq \text{convex hull}$ 
insert 0 D
  unfolding substd_simplex[OF assms]
proof (intro exI; safe)
  have  $0 < \text{card } D$  using  $\langle D \neq \{\} \rangle$   $\langle \text{finite } D \rangle$ 
    by (simp add: card_gt_0_iff)
  have  $\text{Min } (((\cdot) x) ' D) > 0$ 
    using as  $\langle D \neq \{\} \rangle$   $\langle \text{finite } D \rangle$  by (simp)
  moreover have  $d > 0$ 
    using as  $\langle 0 < \text{card } D \rangle$  by (auto simp: d_def)
  ultimately show  $\text{min } (\text{Min } (((\cdot) x) ' D)) d > 0$ 
    by auto
  fix y :: 'a
  assume y2:  $\forall i \in \text{Basis}. i \notin D \longrightarrow y \cdot i = 0$ 
  assume  $y \in \text{ball } x (\text{min } (\text{Min } ((\cdot) x ' D)) d)$ 
  then have  $y: \text{dist } x y < \text{min } (\text{Min } ((\cdot) x ' D)) d$ 
    by auto
  have  $\text{sum } ((\cdot) y) D \leq \text{sum } (\lambda i. x \cdot i + d) D$ 
  proof (rule sum_mono)
    fix i
    assume  $i \in D$ 
    with D have  $i: i \in \text{Basis}$ 
      by auto
    have  $|y \cdot i - x \cdot i| \leq \text{norm } (y - x)$ 
      by (metis i inner_commute inner_diff_right norm_bound_Basis_le
order_refl)
    also have  $\dots < d$ 
      by (metis dist_norm_min_less_iff_conj norm_minus_commute y)
    finally have  $|y \cdot i - x \cdot i| < d$  .
    then show  $y \cdot i \leq x \cdot i + d$  by auto
  qed
  also have  $\dots \leq 1$ 
    unfolding sum.distrib sum_constant d_def using  $\langle 0 < \text{card } D \rangle$ 
    by auto
  finally show  $\text{sum } ((\cdot) y) D \leq 1$  .

  fix i :: 'a
  assume  $i: i \in \text{Basis}$ 
  then show  $0 \leq y \cdot i$ 
  proof (cases  $i \in D$ )
    case True
      have  $\text{norm } (x - y) < x \cdot i$ 
        using y Min_gr_iff[of  $(\cdot) x ' D$  norm  $(x - y)$ ]  $\langle 0 < \text{card } D \rangle$   $\langle i \in D \rangle$ 

```

```

    by (simp add: dist_norm card_gt_0_iff)
  then show  $0 \leq y \cdot i$ 
    using Basis_le_norm[OF i, of x - y] and as(1)[rule_format]
    by (auto simp: inner_simps)
  qed (use y2 in auto)
qed
then have  $x \in \text{rel\_interior} (\text{convex hull} (\text{insert } 0 D))$ 
  using h0 h2 rel_interior_ball by force
}
ultimately have
 $\bigwedge x. x \in \text{rel\_interior} (\text{convex hull insert } 0 D) \longleftrightarrow$ 
 $x \in \{x. (\forall i \in D. 0 < x \cdot i) \wedge \text{sum } ((\cdot) x) D < 1 \wedge (\forall i \in \text{Basis}. i \notin D \longrightarrow x$ 
 $\cdot i = 0)\}$ 
  by blast
then show ?thesis by (rule set_eqI)
qed
qed

```

```

lemma rel_interior_substd_simplex_nonempty:
  assumes  $D \neq \{\}$ 
  and  $D \subseteq \text{Basis}$ 
  obtains  $a :: 'a :: \text{euclidean\_space}$ 
  where  $a \in \text{rel\_interior} (\text{convex hull} (\text{insert } 0 D))$ 
proof -
  let ?a =  $\text{sum } (\lambda b :: 'a :: \text{euclidean\_space}. \text{inverse } (2 * \text{real } (\text{card } D)) *_{\mathbb{R}} b) D$ 
  have finite D
    using assms finite_Basis infinite_super by blast
  then have d1:  $0 < \text{real } (\text{card } D)$ 
    using  $\langle D \neq \{\} \rangle$  by auto
  {
    fix i
    assume  $i \in D$ 
    have  $?a \cdot i = \text{sum } (\lambda j. \text{if } i = j \text{ then } \text{inverse } (2 * \text{real } (\text{card } D)) \text{ else } 0) D$ 
      unfolding inner_sum_left
      using  $\langle i \in D \rangle$  by (auto simp: inner_Basis subsetD[OF assms(2)] intro:
sum.cong)
    also have ... =  $\text{inverse } (2 * \text{real } (\text{card } D))$ 
      using  $\langle i \in D \rangle \langle \text{finite } D \rangle$  by auto
    finally have  $?a \cdot i = \text{inverse } (2 * \text{real } (\text{card } D))$  .
  }
  note ** = this
  show ?thesis
proof
  show  $?a \in \text{rel\_interior} (\text{convex hull} (\text{insert } 0 D))$ 
    unfolding rel_interior_substd_simplex[OF assms(2)]
  proof safe
    fix i
    assume  $i \in D$ 
    have  $0 < \text{inverse } (2 * \text{real } (\text{card } D))$ 

```



```

    using d1 by auto
  also have ... = ?a · i using **[of i] ‹i ∈ D›
    by auto
  finally show 0 < ?a · i by auto
next
have sum ((·) ?a) D = sum (λi. inverse (2 * real (card D))) D
  by (rule sum.cong) (rule refl, rule **)
also have ... < 1
  unfolding sum_constant divide_real_def[symmetric]
  by (auto simp add: field_simps)
finally show sum ((·) ?a) D < 1 by auto
next
fix i
assume i ∈ Basis and i ∉ D
have ?a ∈ span D
proof (rule span_sum[of D (λb. b /R (2 * real (card D))) D])
{
  fix x :: 'a::euclidean_space
  assume x ∈ D
  then have x ∈ span D
    using span_base[of _ D] by auto
  then have x /R (2 * real (card D)) ∈ span D
    using span_mul[of x D (inverse (real (card D)) / 2)] by auto
}
then show ∧x. x ∈ D ⇒ x /R (2 * real (card D)) ∈ span D
  by auto
qed
then show ?a · i = 0
  using ‹i ∉ D› unfolding span_substd_basis[OF assms(2)] using ‹i ∈
Basis› by auto
qed
qed
qed

```

6.0.3 Relative interior of convex set

```

lemma rel_interior_convex_nonempty_aux:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  and 0 ∈ S
  shows rel_interior S ≠ {}
proof (cases S = {0})
case True
  then show ?thesis using rel_interior_sing by auto
next
case False
  obtain B where B: independent B ∧ B ≤ S ∧ S ≤ span B ∧ card B = dim S
  using basis_exists[of S] by metis
  then have B ≠ {}

```

```

    using B assms ⟨S ≠ {0}⟩ span_empty by auto
  have insert 0 B ≤ span B
    using subspace_span[of B] subspace_0[of span B]
      span_superset by auto
  then have span (insert 0 B) ≤ span B
    using span_span[of B] span_mono[of insert 0 B span B] by blast
  then have convex hull insert 0 B ≤ span B
    using convex_hull_subset_span[of insert 0 B] by auto
  then have span (convex hull insert 0 B) ≤ span B
    using span_span[of B]
      span_mono[of convex hull insert 0 B span B] by blast
  then have *: span (convex hull insert 0 B) = span B
    using span_mono[of B convex hull insert 0 B] hull_subset[of insert 0 B] by
  auto
  then have span (convex hull insert 0 B) = span S
    using B span_mono[of B S] span_mono[of S span B]
      span_span[of B] by auto
  moreover have 0 ∈ affine hull (convex hull insert 0 B)
    using hull_subset[of convex hull insert 0 B] hull_subset[of insert 0 B] by auto
  ultimately have **: affine hull (convex hull insert 0 B) = affine hull S
    using affine_hull_span_0[of convex hull insert 0 B] affine_hull_span_0[of S]
      assms hull_subset[of S]
    by auto
  obtain d and f :: 'n ⇒ 'n where
    fd: card d = card B linear f f ' B = d
      f ' span B = {x. ∀ i ∈ Basis. i ∉ d ⟶ x · i = (0::real)} ∧ inj_on f (span B)
    and d: d ⊆ Basis
    using basis_to_substdbasis_subspace_isomorphism[of B, OF _] B by auto
  then have bounded_linear f
    using linear_conv_bounded_linear by auto
  have d ≠ {}
    using fd B ⟨B ≠ {}⟩ by auto
  have insert 0 d = f ' (insert 0 B)
    using fd linear_0 by auto
  then have (convex hull (insert 0 d)) = f ' (convex hull (insert 0 B))
    using convex_hull_linear_image[of f (insert 0 d)]
      convex_hull_linear_image[of f (insert 0 B)] ⟨linear f⟩
    by auto
  moreover have rel_interior (f ' (convex hull insert 0 B)) = f ' rel_interior
    (convex hull insert 0 B)
  proof (rule rel_interior_injective_on_span_linear_image[OF ⟨bounded_linear
  f⟩])
    show inj_on f (span (convex hull insert 0 B))
      using fd * by auto
  qed
  ultimately have rel_interior (convex hull insert 0 B) ≠ {}
    using rel_interior_substd_simplex_nonempty[OF ⟨d ≠ {}⟩ d] by fastforce
  moreover have convex hull (insert 0 B) ⊆ S
    using B assms hull_mono[of insert 0 B S convex] convex_hull_eq by auto

```

```

ultimately show ?thesis
  using subset_rel_interior[of convex hull insert 0 B S] ** by auto
qed

```

```

lemma rel_interior_eq_empty:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior S = {}  $\longleftrightarrow$  S = {}
proof -
  {
    assume S  $\neq$  {}
    then obtain a where a  $\in$  S by auto
    then have 0  $\in$  (+) (-a) ' S
      using assms exI[of ( $\lambda x. x \in S \wedge - a + x = 0$ ) a] by auto
    then have rel_interior ((+) (-a) ' S)  $\neq$  {}
      using rel_interior_convex_nonempty_aux[of (+) (-a) ' S]
        convex_translation[of S -a] assms
      by auto
    then have rel_interior S  $\neq$  {}
      using rel_interior_translation [of - a] by simp
  }
  then show ?thesis by auto
qed

```

```

lemma interior_simplex_nonempty:
  fixes S :: 'N :: euclidean_space set
  assumes independent S finite S card S = DIM('N)
  obtains a where a  $\in$  interior (convex hull (insert 0 S))
proof -
  have affine_hull (insert 0 S) = UNIV
    by (simp add: hull_inc affine_hull_span_0 dim_eq_full[symmetric]
      assms(1) assms(3) dim_eq_card_independent)
  moreover have rel_interior (convex hull insert 0 S)  $\neq$  {}
    using rel_interior_eq_empty [of convex hull (insert 0 S)] by auto
  ultimately have interior (convex hull insert 0 S)  $\neq$  {}
    by (simp add: rel_interior_interior)
  with that show ?thesis
    by auto
qed

```

```

lemma convex_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows convex (rel_interior S)
proof -
  {
    fix x y and u :: real
    assume asm: x  $\in$  rel_interior S y  $\in$  rel_interior S 0  $\leq$  u u  $\leq$  1
    then have x  $\in$  S

```

```

    using rel_interior_subset by auto
  have  $x - u *_R (x - y) \in \text{rel\_interior } S$ 
  proof (cases  $0 = u$ )
    case False
    then have  $0 < u$  using asm by auto
    then show ?thesis
      using asm rel_interior_convex_shrink[of  $S$   $y$   $x$   $u$ ] assms  $\langle x \in S \rangle$  by auto
  next
    case True
    then show ?thesis using asm by auto
  qed
  then have  $(1 - u) *_R x + u *_R y \in \text{rel\_interior } S$ 
  by (simp add: algebra_simps)
}
then show ?thesis
  unfolding convex_alt by auto
qed

```

```

lemma convex_closure_rel_interior:
  fixes  $S :: 'n::\text{euclidean\_space}$  set
  assumes convex  $S$ 
  shows  $\text{closure} (\text{rel\_interior } S) = \text{closure } S$ 
proof -
  have  $h1: \text{closure} (\text{rel\_interior } S) \leq \text{closure } S$ 
  using closure_mono[of  $\text{rel\_interior } S$   $S$ ] rel_interior_subset[of  $S$ ] by auto
  show ?thesis
  proof (cases  $S = \{\}$ )
    case False
    then obtain  $a$  where  $a \in \text{rel\_interior } S$ 
    using rel_interior_eq_empty assms by auto
    { fix  $x$ 
      assume  $x \in \text{closure } S$ 
      {
        assume  $x = a$ 
        then have  $x \in \text{closure} (\text{rel\_interior } S)$ 
        using  $a$  unfolding closure_def by auto
      }
    }
    moreover
    {
      assume  $x \neq a$ 
      {
        fix  $e :: \text{real}$ 
        assume  $e > 0$ 
        define  $e1$  where  $e1 = \min 1 (e / \text{norm } (x - a))$ 
        then have  $e1: e1 > 0 \ e1 \leq 1 \ e1 * \text{norm } (x - a) \leq e$ 
        using  $\langle x \neq a \rangle \langle e > 0 \rangle$  le_divide_eq[of  $e1$   $e$   $\text{norm } (x - a)$ ]
        by simp_all
        then have  $*$ :  $x - e1 *_R (x - a) \in \text{rel\_interior } S$ 
        using rel_interior_closure_convex_shrink[of  $S$   $a$   $e1$ ] assms  $x$   $a$   $e1\_def$ 

```

```

    by auto
    have  $\exists y. y \in \text{rel\_interior } S \wedge y \neq x \wedge \text{dist } y \ x \leq e$ 
      using *  $\langle x \neq a \rangle$  e1 by force
  }
  then have  $x \text{ islimpt } \text{rel\_interior } S$ 
    unfolding islimpt_approachable_le by auto
  then have  $x \in \text{closure}(\text{rel\_interior } S)$ 
    unfolding closure_def by auto
  }
  ultimately have  $x \in \text{closure}(\text{rel\_interior } S)$  by auto
}
then show ?thesis using h1 by auto
qed auto
qed

```

lemma empty_interior_subset_hyperplane_aux:

```

  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes convex  $S$   $0 \in S$  and empty_int:  $\text{interior } S = \{\}$ 
  shows  $\exists a \ b. a \neq 0 \wedge S \subseteq \{x. a \cdot x = b\}$ 
proof -
  have False if  $\bigwedge a. a = 0 \vee (\forall b. \exists T \in S. a \cdot T \neq b)$ 
  proof -
    have  $\text{rel\_int: } \text{rel\_interior } S \neq \{\}$ 
      using assms rel_interior_eq_empty by auto
    moreover
    have  $\text{dim } S \neq \text{dim } (\text{UNIV}::'a \text{ set})$ 
      by (metis aff_dim_zero affine_hull_UNIV  $\langle 0 \in S \rangle$  dim_UNIV empty_int
hull_inc rel_int rel_interior_interior)
    then obtain a where  $a \neq 0$  and  $a: \text{span } S \subseteq \{x. a \cdot x = 0\}$ 
      using lowdim_subset_hyperplane
      by (metis dim_UNIV dim_subset_UNIV order_less_le)
    have  $\text{span } \text{UNIV} = \text{span } S$ 
      by (metis span_base span_not_UNIV_orthogonal that)
    then have  $\text{UNIV} \subseteq \text{affine hull } S$ 
      by (simp add:  $\langle 0 \in S \rangle$  hull_inc affine_hull_span_0)
    ultimately show False
      using  $\langle \text{rel\_interior } S \neq \{\} \rangle$  empty_int rel_interior_interior by blast
  qed
  then show ?thesis
    by blast
qed

```

lemma empty_interior_subset_hyperplane:

```

  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes convex  $S$  and int:  $\text{interior } S = \{\}$ 
  obtains a b where  $a \neq 0$   $S \subseteq \{x. a \cdot x = b\}$ 
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis

```

```

    using that by blast
next
case False
then obtain u where u ∈ S
  by blast
have ∃ a b. a ≠ 0 ∧ (λx. x - u) ' S ⊆ {x. a · x = b}
proof (rule empty_interior_subset_hyperplane_aux)
  show convex ((λx. x - u) ' S)
    using ⟨convex S⟩ by force
  show 0 ∈ (λx. x - u) ' S
    by (simp add: ⟨u ∈ S⟩)
  show interior ((λx. x - u) ' S) = {}
    by (simp add: int_interior_translation_subtract)
qed
then obtain a b where a ≠ 0 and ab: (λx. x - u) ' S ⊆ {x. a · x = b}
  by metis
then have S ⊆ {x. a · x = b + (a · u)}
  using ab by (auto simp: algebra_simps)
then show ?thesis
  using ⟨a ≠ 0⟩ that by auto
qed

lemma rel_interior_same_affine_hull:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows affine hull (rel_interior S) = affine hull S
  by (metis assms closure_same_affine_hull convex_closure_rel_interior)

lemma rel_interior_aff_dim:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows aff_dim (rel_interior S) = aff_dim S
  by (metis aff_dim_affine_hull2 assms rel_interior_same_affine_hull)

lemma rel_interior_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior (rel_interior S) = rel_interior S
proof -
  have openin (top_of_set (affine hull (rel_interior S))) (rel_interior S)
    using openin_rel_interior[of S] rel_interior_same_affine_hull[of S] assms by
  auto
  then show ?thesis
    using rel_interior_def by auto
qed

lemma rel_interior_rel_open:
  fixes S :: 'n::euclidean_space set
  assumes convex S

```

shows $\text{rel_open } (\text{rel_interior } S)$
unfolding rel_open_def **using** $\text{rel_interior_rel_interior}$ *assms* **by** *auto*

lemma *convex_rel_interior_closure_aux*:

fixes $x\ y\ z :: 'n::\text{euclidean_space}$

assumes $0 < a\ 0 < b\ (a + b) *_{\mathbb{R}} z = a *_{\mathbb{R}} x + b *_{\mathbb{R}} y$

obtains e **where** $0 < e\ e < 1\ z = y - e *_{\mathbb{R}} (y - x)$

proof –

define e **where** $e = a / (a + b)$

have $z = (1 / (a + b)) *_{\mathbb{R}} ((a + b) *_{\mathbb{R}} z)$

using *assms* **by** (*simp add: eq_vector_fraction_iff*)

also have $\dots = (1 / (a + b)) *_{\mathbb{R}} (a *_{\mathbb{R}} x + b *_{\mathbb{R}} y)$

using *assms* *scaleR_cancel_left*[of $1/(a+b)$] $(a + b) *_{\mathbb{R}} z = a *_{\mathbb{R}} x + b *_{\mathbb{R}} y$

by *auto*

also have $\dots = y - e *_{\mathbb{R}} (y - x)$

using e_def *assms*

by (*simp add: divide_simps vector_fraction_eq_iff*) (*simp add: algebra_simps*)

finally have $z = y - e *_{\mathbb{R}} (y - x)$

by *auto*

moreover have $e > 0\ e < 1$ **using** e_def *assms* **by** *auto*

ultimately show *?thesis* **using** *that*[of e] **by** *auto*

qed

lemma *convex_rel_interior_closure*:

fixes $S :: 'n::\text{euclidean_space}$ *set*

assumes *convex* S

shows $\text{rel_interior } (\text{closure } S) = \text{rel_interior } S$

proof (*cases* $S = \{\}$)

case *True*

then show *?thesis*

using *assms* *rel_interior_eq_empty* **by** *auto*

next

case *False*

have $\text{rel_interior } (\text{closure } S) \supseteq \text{rel_interior } S$

using *subset_rel_interior*[of S *closure* S] *closure_same_affine_hull* *closure_subset*

by *auto*

moreover

{

fix z

assume $z: z \in \text{rel_interior } (\text{closure } S)$

obtain x **where** $x: x \in \text{rel_interior } S$

using $\langle S \neq \{\} \rangle$ *assms* *rel_interior_eq_empty* **by** *auto*

have $z \in \text{rel_interior } S$

proof (*cases* $x = z$)

case *True*

then show *?thesis* **using** x **by** *auto*

next

case *False*

obtain e **where** $e: e > 0\ \text{cball } z\ e \cap \text{affine hull closure } S \leq \text{closure } S$

```

    using z rel_interior_cball[of closure S] by auto
  hence *:  $0 < e/\text{norm}(z-x)$  using e False by auto
  define y where  $y = z + (e/\text{norm}(z-x)) *_R (z-x)$ 
  have yball:  $y \in \text{cball } z \ e$ 
    using y_def dist_norm[of z y] e by auto
  have x  $\in$  affine hull closure S
    using x rel_interior_subset_closure hull_inc[of x closure S] by blast
  moreover have  $z \in$  affine hull closure S
    using z rel_interior_subset_hull_subset[of closure S] by blast
  ultimately have  $y \in$  affine hull closure S
    using y_def affine_affine_hull[of closure S]
    mem_affine_3_minus [of affine hull closure S z z x e/norm(z-x)] by auto
  then have  $y \in$  closure S using e yball by auto
  have  $(1 + (e/\text{norm}(z-x))) *_R z = (e/\text{norm}(z-x)) *_R x + y$ 
    using y_def by (simp add: algebra_simps)
  then obtain e1 where  $0 < e1$   $e1 < 1$   $z = y - e1 *_R (y - x)$ 
    using * convex_rel_interior_closure_aux[of e / norm (z - x) 1 z x y]
    by (auto simp add: algebra_simps)
  then show ?thesis
    using rel_interior_closure_convex_shrink assms x  $\langle y \in$  closure S  $\rangle$ 
    by fastforce
qed
}
ultimately show ?thesis by auto
qed

```

lemma *convex_interior_closure:*

```

  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows interior (closure S) = interior S
  using closure_aff_dim[of S] interior_rel_interior_gen[of S]
    interior_rel_interior_gen[of closure S]
    convex_rel_interior_closure[of S] assms
  by auto

```

lemma *open_subset_closure_of_interval:*

```

  assumes open U is_interval S
  shows  $U \subseteq$  closure S  $\longleftrightarrow$   $U \subseteq$  interior S
  by (metis assms convex_interior_closure is_interval_convex open_subset_interior)

```

lemma *closure_eq_rel_interior_eq:*

```

  fixes S1 S2 :: 'n::euclidean_space set
  assumes convex S1
    and convex S2
  shows closure S1 = closure S2  $\longleftrightarrow$  rel_interior S1 = rel_interior S2
  by (metis convex_rel_interior_closure convex_closure_rel_interior assms)

```

lemma *closure_eq_between:*

```

  fixes S1 S2 :: 'n::euclidean_space set

```



```

assumes convex S1
and convex S2
shows closure S1 = closure S2  $\longleftrightarrow$  rel_interior S1  $\leq$  S2  $\wedge$  S2  $\subseteq$  closure S1
(is ?A  $\longleftrightarrow$  ?B)
proof
  assume ?A
  then show ?B
    by (metis assms closure_subset convex_rel_interior_closure rel_interior_subset)
next
  assume ?B
  then have closure S1  $\subseteq$  closure S2
    by (metis assms(1) convex_closure_rel_interior_closure_mono)
  moreover from  $\langle ?B \rangle$  have closure S1  $\supseteq$  closure S2
    by (metis closed_closure_closure_minimal)
  ultimately show ?A ..
qed

lemma open_inter_closure_rel_interior:
  fixes S A :: 'n::euclidean_space set
  assumes convex S
  and open A
  shows A  $\cap$  closure S =  $\{\}$   $\longleftrightarrow$  A  $\cap$  rel_interior S =  $\{\}$ 
  by (metis assms convex_closure_rel_interior open_Int_closure_eq_empty)

lemma rel_interior_open_segment:
  fixes a :: 'a :: euclidean_space
  shows rel_interior(open_segment a b) = open_segment a b
proof (cases a = b)
  case True then show ?thesis by auto
next
  case False then
    have open_segment a b = affine_hull {a, b}  $\cap$  ball ((a + b) /R 2) (norm (b -
a) / 2)
    by (simp add: open_segment_as_ball)
    then show ?thesis
      unfolding rel_interior_eq_openin_open
      by (metis Elementary_Metric_Spaces.open_ball False affine_hull_open_segment)
qed

lemma rel_interior_closed_segment:
  fixes a :: 'a :: euclidean_space
  shows rel_interior(closed_segment a b) =
    (if a = b then {a} else open_segment a b)
proof (cases a = b)
  case True then show ?thesis by auto
next
  case False then show ?thesis
    by simp
    (metis closure_open_segment convex_open_segment convex_rel_interior_closure

```

rel_interior_open_segment)

qed

lemmas *rel_interior_segment* = *rel_interior_closed_segment* *rel_interior_open_segment*

6.0.4 The relative frontier of a set

definition *rel_frontier* $S = \text{closure } S - \text{rel_interior } S$

lemma *rel_frontier_empty* [*simp*]: *rel_frontier* $\{\} = \{\}$
by (*simp* *add: rel_frontier_def*)

lemma *rel_frontier_eq_empty*:
fixes $S :: 'n::\text{euclidean_space}$ *set*
shows *rel_frontier* $S = \{\} \longleftrightarrow \text{affine } S$
unfolding *rel_frontier_def*
using *rel_interior_subset_closure* **by** (*auto simp add: rel_interior_eq_closure*
[*symmetric*])

lemma *rel_frontier_singleton* [*simp*]:
fixes $a :: 'n::\text{euclidean_space}$
shows *rel_frontier* $\{a\} = \{\}$
by (*simp add: rel_frontier_def*)

lemma *rel_frontier_affine_hull*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
shows *rel_frontier* $S \subseteq \text{affine hull } S$
using *closure_affine_hull* *rel_frontier_def* **by** *fastforce*

lemma *rel_frontier_cball* [*simp*]:
fixes $a :: 'n::\text{euclidean_space}$
shows *rel_frontier*(*cball* a r) = (if $r = 0$ then $\{\}$ else *sphere* a r)

proof (*cases* *rule: linorder_cases* [*of* r 0])

case *less* **then show** *?thesis*
by (*force simp: sphere_def*)

next

case *equal* **then show** *?thesis* **by** *simp*

next

case *greater* **then show** *?thesis*

by *simp* (*metis* *centre_in_ball* *empty_iff* *frontier_cball* *frontier_def* *interior_cball* *interior_rel_interior_gen* *rel_frontier_def*)

qed

lemma *rel_frontier_translation*:
fixes $a :: 'a::\text{euclidean_space}$
shows *rel_frontier*(($\lambda x. a + x$) ' S) = ($\lambda x. a + x$) '*(rel_frontier* S)
by (*simp add: rel_frontier_def* *translation_diff* *rel_interior_translation* *closure_translation*)

```

lemma rel_frontier_nonempty_interior:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{interior } S \neq \{\} \implies \text{rel\_frontier } S = \text{frontier } S$ 
  by (metis frontier_def interior_rel_interior_gen rel_frontier_def)

lemma rel_frontier_frontier:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{affine hull } S = \text{UNIV} \implies \text{rel\_frontier } S = \text{frontier } S$ 
  by (simp add: frontier_def rel_frontier_def rel_interior_interior)

lemma closest_point_in_rel_frontier:
   $\llbracket \text{closed } S; S \neq \{\}; x \in \text{affine hull } S - \text{rel\_interior } S \rrbracket$ 
   $\implies \text{closest\_point } S x \in \text{rel\_frontier } S$ 
  by (simp add: closest_point_in_rel_interior closest_point_in_set rel_frontier_def)

lemma closed_rel_frontier [iff]:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{closed } (\text{rel\_frontier } S)$ 
proof -
  have *:  $\text{closedin } (\text{top\_of\_set } (\text{affine hull } S)) (\text{closure } S - \text{rel\_interior } S)$ 
  by (simp add: closed_subset closedin_diff closure_affine_hull openin_rel_interior)
  show ?thesis
  proof (rule closedin_closed_trans[of affine hull S rel_frontier S])
    show  $\text{closedin } (\text{top\_of\_set } (\text{affine hull } S)) (\text{rel\_frontier } S)$ 
    by (simp add: * rel_frontier_def)
  qed simp
qed

lemma closed_rel_boundary:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{closed } S \implies \text{closed}(S - \text{rel\_interior } S)$ 
  by (metis closed_rel_frontier closure_closed rel_frontier_def)

lemma compact_rel_boundary:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{compact } S \implies \text{compact}(S - \text{rel\_interior } S)$ 
  by (metis bounded_diff closed_rel_boundary closure_eq compact_closure compact_imp_closed)

lemma bounded_rel_frontier:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{bounded } S \implies \text{bounded}(\text{rel\_frontier } S)$ 
  by (simp add: bounded_closure bounded_diff rel_frontier_def)

lemma compact_rel_frontier_bounded:
  fixes  $S :: 'n::\text{euclidean\_space set}$ 
  shows  $\text{bounded } S \implies \text{compact}(\text{rel\_frontier } S)$ 
using bounded_rel_frontier closed_rel_frontier compact_eq_bounded_closed by
  blast

```

```

lemma compact_rel_frontier:
  fixes S :: 'n::euclidean_space set
  shows compact S  $\implies$  compact(rel_frontier S)
by (meson compact_eq_bounded_closed compact_rel_frontier_bounded)

lemma convex_same_rel_interior_closure:
  fixes S :: 'n::euclidean_space set
  shows  $\llbracket$ convex S; convex T $\rrbracket$ 
 $\implies$  rel_interior S = rel_interior T  $\longleftrightarrow$  closure S = closure T
by (simp add: closure_eq_rel_interior_eq)

lemma convex_same_rel_interior_closure_straddle:
  fixes S :: 'n::euclidean_space set
  shows  $\llbracket$ convex S; convex T $\rrbracket$ 
 $\implies$  rel_interior S = rel_interior T  $\longleftrightarrow$ 
rel_interior S  $\subseteq$  T  $\wedge$  T  $\subseteq$  closure S
by (simp add: closure_eq_between_convex_same_rel_interior_closure)

lemma convex_rel_frontier_aff_dim:
  fixes S1 S2 :: 'n::euclidean_space set
  assumes convex S1
  and convex S2
  and S2  $\neq$  {}
  and S1  $\leq$  rel_frontier S2
  shows aff_dim S1 < aff_dim S2
proof -
  have S1  $\subseteq$  closure S2
  using assms unfolding rel_frontier_def by auto
  then have *: affine hull S1  $\subseteq$  affine hull S2
  using hull_mono[of S1 closure S2] closure_same_affine_hull[of S2] by blast
  then have aff_dim S1  $\leq$  aff_dim S2
  using * aff_dim_affine_hull[of S1] aff_dim_affine_hull[of S2]
  aff_dim_subset[of affine hull S1 affine hull S2]
  by auto
  moreover
  {
  assume eq: aff_dim S1 = aff_dim S2
  then have S1  $\neq$  {}
  using aff_dim_empty[of S1] aff_dim_empty[of S2]  $\langle$ S2  $\neq$  {} $\rangle$  by auto
  have **: affine hull S1 = affine hull S2
  by (simp_all add: * eq  $\langle$ S1  $\neq$  {} $\rangle$  affine_dim_equal)
  obtain a where a: a  $\in$  rel_interior S1
  using  $\langle$ S1  $\neq$  {} $\rangle$  rel_interior_eq_empty assms by auto
  obtain T where T: open T a  $\in$  T  $\cap$  S1 T  $\cap$  affine hull S1  $\subseteq$  S1
  using mem_rel_interior[of a S1] a by auto
  then have a  $\in$  T  $\cap$  closure S2
  using a assms unfolding rel_frontier_def by auto
  then obtain b where b: b  $\in$  T  $\cap$  rel_interior S2

```

```

    using open_inter_closure_rel_interior[of S2 T] assms T by auto
  then have b ∈ affine hull S1
    using rel_interior_subset hull_subset[of S2] ** by auto
  then have b ∈ S1
    using T b by auto
  then have False
    using b assms unfolding rel_frontier_def by auto
}
ultimately show ?thesis
  using less_le by auto
qed

```

lemma convex_rel_interior_if:

```

  fixes S :: 'n::euclidean_space set
  assumes convex S
    and z ∈ rel_interior S
  shows ∀ x ∈ affine hull S. ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m → (1 - e) *R x
+ e *R z ∈ S)
proof -
  obtain e1 where e1: e1 > 0 ∧ cball z e1 ∩ affine hull S ⊆ S
    using mem_rel_interior_cball[of z S] assms by auto
  {
    fix x
    assume x: x ∈ affine hull S
    {
      assume x ≠ z
      define m where m = 1 + e1/norm(x-z)
      hence m > 1 using e1 ⟨x ≠ z⟩ by auto
      {
        fix e
        assume e: e > 1 ∧ e ≤ m
        have z ∈ affine hull S
          using assms rel_interior_subset hull_subset[of S] by auto
        then have *: (1 - e)*R x + e *R z ∈ affine hull S
          using mem_affine[of affine hull S x z (1-e) e] affine_affine_hull[of S] x
          by auto
        have norm (z + e *R x - (x + e *R z)) = norm ((e - 1) *R (x - z))
          by (simp add: algebra_simps)
        also have ... = (e - 1) * norm (x - z)
          using norm_scaleR e by auto
        also have ... ≤ (m - 1) * norm (x - z)
          using e mult_right_mono[of _ _ norm(x-z)] by auto
        also have ... = (e1 / norm (x - z)) * norm (x - z)
          using m_def by auto
        also have ... = e1
          using ⟨x ≠ z⟩ e1 by simp
        finally have **: norm (z + e *R x - (x + e *R z)) ≤ e1
          by auto
        have (1 - e)*R x + e *R z ∈ cball z e1

```

```

    using m_def **
    unfolding cball_def dist_norm
    by (auto simp add: algebra_simps)
    then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
    using e * e1 by auto
  }
  then have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
  using <m> 1 by auto
}
moreover
{
  assume x = z
  define m where  $m = 1 + e1$ 
  then have  $m > 1$ 
  using e1 by auto
  {
    fix e
    assume e:  $e > 1 \wedge e \leq m$ 
    then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
    using e1 x <x = z> by (auto simp add: algebra_simps)
    then have  $(1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
    using e by auto
  }
  then have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
  using <m > 1 by auto
}
ultimately have  $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
by blast
}
then show ?thesis by auto
qed

```

```

lemma convex_rel_interior_if2:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  assumes z ∈ rel_interior S
  shows  $\forall x \in \text{affine hull } S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
  using convex_rel_interior_if[of S z] assms by auto

```

```

lemma convex_rel_interior_only_if:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  and S ≠ {}
  assumes  $\forall x \in S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
  shows z ∈ rel_interior S
proof -

```

```

obtain  $x$  where  $x: x \in \text{rel\_interior } S$ 
  using  $\text{rel\_interior\_eq\_empty } \text{assms}$  by  $\text{auto}$ 
then have  $x \in S$ 
  using  $\text{rel\_interior\_subset}$  by  $\text{auto}$ 
then obtain  $e$  where  $e: e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S$ 
  using  $\text{assms}$  by  $\text{auto}$ 
define  $y$  where  $[\text{abs\_def}]: y = (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z$ 
then have  $y \in S$  using  $e$  by  $\text{auto}$ 
define  $e1$  where  $e1 = 1/e$ 
then have  $0 < e1 \wedge e1 < 1$  using  $e$  by  $\text{auto}$ 
then have  $z = y - (1 - e1) *_{\mathbb{R}} (y - x)$ 
  using  $e1\_def$   $y\_def$  by  $(\text{auto } \text{simp } \text{add: algebra\_simps})$ 
then show  $?thesis$ 
  using  $\text{rel\_interior\_convex\_shrink}[of\ S\ x\ y\ 1-e1]$   $\langle 0 < e1 \wedge e1 < 1 \rangle$   $\langle y \in S \rangle$ 
 $x$   $\text{assms}$ 
  by  $\text{auto}$ 
qed

```

```

lemma  $\text{convex\_rel\_interior\_iff}$ :
  fixes  $S :: 'n::\text{euclidean\_space } \text{set}$ 
  assumes  $\text{convex } S$ 
  and  $S \neq \{\}$ 
  shows  $z \in \text{rel\_interior } S \iff (\forall x \in S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
  using  $\text{assms } \text{hull\_subset}[of\ S\ \text{affine}]$ 
   $\text{convex\_rel\_interior\_if}[of\ S\ z]$   $\text{convex\_rel\_interior\_only\_if}[of\ S\ z]$ 
  by  $\text{auto}$ 

```

```

lemma  $\text{convex\_rel\_interior\_iff2}$ :
  fixes  $S :: 'n::\text{euclidean\_space } \text{set}$ 
  assumes  $\text{convex } S$ 
  and  $S \neq \{\}$ 
  shows  $z \in \text{rel\_interior } S \iff (\forall x \in \text{affine hull } S. \exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in S)$ 
  using  $\text{assms } \text{hull\_subset}[of\ S]$   $\text{convex\_rel\_interior\_if2}[of\ S\ z]$   $\text{convex\_rel\_interior\_only\_if}[of\ S\ z]$ 
  by  $\text{auto}$ 

```

```

lemma  $\text{convex\_interior\_iff}$ :
  fixes  $S :: 'n::\text{euclidean\_space } \text{set}$ 
  assumes  $\text{convex } S$ 
  shows  $z \in \text{interior } S \iff (\forall x. \exists e. e > 0 \wedge z + e *_{\mathbb{R}} x \in S)$ 
proof  $(\text{cases } \text{aff\_dim } S = \text{int } \text{DIM } ('n))$ 
  case  $\text{False}$ 
  { assume  $z \in \text{interior } S$ 
    then have  $\text{False}$ 
      using  $\text{False } \text{interior\_rel\_interior\_gen}[of\ S]$  by  $\text{auto}$  }
  moreover
  { assume  $r: \forall x. \exists e. e > 0 \wedge z + e *_{\mathbb{R}} x \in S$ 
    { fix  $x$ 

```

```

obtain  $e1$  where  $e1: e1 > 0 \wedge z + e1 *_R (x - z) \in S$ 
  using  $r$  by auto
obtain  $e2$  where  $e2: e2 > 0 \wedge z + e2 *_R (z - x) \in S$ 
  using  $r$  by auto
define  $x1$  where [abs_def]:  $x1 = z + e1 *_R (x - z)$ 
then have  $x1: x1 \in \text{affine hull } S$ 
  using  $e1$  hull_subset[of  $S$ ] by auto
define  $x2$  where [abs_def]:  $x2 = z + e2 *_R (z - x)$ 
then have  $x2: x2 \in \text{affine hull } S$ 
  using  $e2$  hull_subset[of  $S$ ] by auto
have  $*$ :  $e1/(e1+e2) + e2/(e1+e2) = 1$ 
  using add_divide_distrib[of  $e1$   $e2$   $e1+e2$ ]  $e1$   $e2$  by simp
then have  $z = (e2/(e1+e2)) *_R x1 + (e1/(e1+e2)) *_R x2$ 
  by (simp add: x1_def x2_def algebra_simps) (simp add: * pth_8)
then have  $z: z \in \text{affine hull } S$ 
  using mem_affine[of affine hull  $S$   $x1$   $x2$   $e2/(e1+e2)$   $e1/(e1+e2)$ ]
     $x1$   $x2$  affine_affine_hull[of  $S$ ]  $*$ 
  by auto
have  $x1 - x2 = (e1 + e2) *_R (x - z)$ 
  using x1_def x2_def by (auto simp add: algebra_simps)
then have  $x = z + (1/(e1+e2)) *_R (x1 - x2)$ 
  using  $e1$   $e2$  by simp
then have  $x \in \text{affine hull } S$ 
  using mem_affine_3_minus[of affine hull  $S$   $z$   $x1$   $x2$   $1/(e1+e2)$ ]
     $x1$   $x2$   $z$  affine_affine_hull[of  $S$ ]
  by auto
}
then have affine hull  $S = UNIV$ 
  by auto
then have aff_dim  $S = \text{int } DIM('n)$ 
  using aff_dim_affine_hull[of  $S$ ] by (simp)
then have False
  using False by auto
}
ultimately show ?thesis by auto
next
case True
then have  $S \neq \{\}$ 
  using aff_dim_empty[of  $S$ ] by auto
have  $*$ : affine hull  $S = UNIV$ 
  using True affine_hull_UNIV by auto
{
  assume  $z \in \text{interior } S$ 
  then have  $z \in \text{rel\_interior } S$ 
    using True interior_rel_interior_gen[of  $S$ ] by auto
  then have  $**$ :  $\forall x. \exists e. e > 1 \wedge (1 - e) *_R x + e *_R z \in S$ 
    using convex_rel_interior_iff2[of  $S$   $z$ ] assms  $\langle S \neq \{\} \rangle$   $*$  by auto
  fix  $x$ 
  obtain  $e1$  where  $e1: e1 > 1 (1 - e1) *_R (z - x) + e1 *_R z \in S$ 

```



```

    using **[rule_format, of z-x] by auto
  define e where [abs_def]: e = e1 - 1
  then have (1 - e1) *R (z - x) + e1 *R z = z + e *R x
    by (simp add: algebra_simps)
  then have e > 0 z + e *R x ∈ S
    using e1 e_def by auto
  then have ∃ e. e > 0 ∧ z + e *R x ∈ S
    by auto
}
moreover
{
  assume r: ∀ x. ∃ e. e > 0 ∧ z + e *R x ∈ S
  {
    fix x
    obtain e1 where e1: e1 > 0 z + e1 *R (z - x) ∈ S
      using r[rule_format, of z-x] by auto
    define e where e = e1 + 1
    then have z + e1 *R (z - x) = (1 - e) *R x + e *R z
      by (simp add: algebra_simps)
    then have e > 1 (1 - e) *R x + e *R z ∈ S
      using e1 e_def by auto
    then have ∃ e. e > 1 ∧ (1 - e) *R x + e *R z ∈ S by auto
  }
  then have z ∈ rel_interior S
    using convex_rel_interior_iff2[of S z] assms ⟨S ≠ {}⟩ by auto
  then have z ∈ interior S
    using True interior_rel_interior_gen[of S] by auto
}
ultimately show ?thesis by auto
qed

```

Relative interior and closure under common operations

lemma *rel_interior_inter_aux*: $\bigcap \{ \text{rel_interior } S \mid S. S \in I \} \subseteq \bigcap I$

proof -

```

{ fix y
  assume y ∈ ⋂ { rel_interior S | S. S ∈ I }
  then have y: ∀ S ∈ I. y ∈ rel_interior S
    by auto
  { fix S
    assume S ∈ I
    then have y ∈ S
      using rel_interior_subset y by auto
  }
  then have y ∈ ⋂ I by auto
}
then show ?thesis by auto
qed

```

```

lemma convex_closure_rel_interior_Int:
  assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{convex } (S :: 'n::\text{euclidean\_space set})$ 
    and  $\bigcap (\text{rel\_interior } ' \mathcal{F}) \neq \{\}$ 
  shows  $\bigcap (\text{closure } ' \mathcal{F}) \subseteq \text{closure } (\bigcap (\text{rel\_interior } ' \mathcal{F}))$ 
proof -
  obtain x where x:  $\forall S \in \mathcal{F}. x \in \text{rel\_interior } S$ 
    using assms by auto
  show ?thesis
proof
  fix y
  assume y:  $y \in \bigcap (\text{closure } ' \mathcal{F})$ 
  show  $y \in \text{closure } (\bigcap (\text{rel\_interior } ' \mathcal{F}))$ 
proof (cases  $y=x$ )
  case True
  with closure_subset x show ?thesis
    by fastforce
next
  case False
  show ?thesis
proof (clarsimp simp: closure_approachable_le)
  fix  $\varepsilon :: \text{real}$ 
  assume e:  $\varepsilon > 0$ 
  define e1 where  $e1 = \min 1 (\varepsilon / \text{norm } (y - x))$ 
  then have e1:  $e1 > 0 \ e1 \leq 1 \ e1 * \text{norm } (y - x) \leq \varepsilon$ 
    using  $\langle y \neq x \rangle \langle \varepsilon > 0 \rangle \text{le\_divide\_eq}[of e1 \ \varepsilon \ \text{norm } (y - x)]$ 
    by simp_all
  define z where  $z = y - e1 *_{\mathbb{R}} (y - x)$ 
  {
  fix S
  assume  $S \in \mathcal{F}$ 
  then have  $z \in \text{rel\_interior } S$ 
    using rel_interior_closure_convex_shrink[of S x y e1] assms x y e1
  z_def
  by auto
  }
  then have *:  $z \in \bigcap (\text{rel\_interior } ' \mathcal{F})$ 
    by auto
  show  $\exists x \in \bigcap (\text{rel\_interior } ' \mathcal{F}). \text{dist } x \ y \leq \varepsilon$ 
  using  $\langle y \neq x \rangle \ z\_def \ * \ e1 \ e \ \text{dist\_norm}[of z y]$ 
  by force
qed
qed
qed
qed

```

```

lemma closure_Inter_convex:
  fixes  $\mathcal{F} :: 'n::\text{euclidean\_space set set}$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies \text{convex } S$  and  $\bigcap (\text{rel\_interior } ' \mathcal{F}) \neq \{\}$ 

```

```

shows closure( $\bigcap \mathcal{F}$ ) =  $\bigcap$ (closure ‘  $\mathcal{F}$ )
proof –
  have  $\bigcap$ (closure ‘  $\mathcal{F}$ )  $\leq$  closure ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ ))
    by (meson assms convex_closure_rel_interior_Int)
  moreover
  have closure ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ ))  $\subseteq$  closure ( $\bigcap \mathcal{F}$ )
    using rel_interior_inter_aux closure_mono[of  $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )  $\bigcap \mathcal{F}$ ]
    by auto
  ultimately show ?thesis
    using closure_Int[of  $\mathcal{F}$ ] by blast
qed

```

```

lemma closure_Inter_convex_open:
  ( $\bigwedge S::'n::euclidean\_space\ set. S \in \mathcal{F} \implies convex\ S \wedge open\ S$ )
   $\implies$  closure( $\bigcap \mathcal{F}$ ) = (if  $\bigcap \mathcal{F} = \{\}$  then  $\{\}$  else  $\bigcap$ (closure ‘  $\mathcal{F}$ ))
  by (simp add: closure_Inter_convex_rel_interior_open)

```

```

lemma convex_Inter_rel_interior_same_closure:
  fixes  $\mathcal{F} :: 'n::euclidean\_space\ set\ set$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies convex\ S$ 
  and  $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )  $\neq \{\}$ 
  shows closure ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )) = closure ( $\bigcap \mathcal{F}$ )
proof –
  have  $\bigcap$ (closure ‘  $\mathcal{F}$ )  $\subseteq$  closure ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ ))
    by (meson assms convex_closure_rel_interior_Int)
  moreover
  have closure ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ ))  $\subseteq$  closure ( $\bigcap \mathcal{F}$ )
    by (metis Setcompr_eq_image closure_mono rel_interior_inter_aux)
  ultimately show ?thesis
    by (simp add: assms closure_Inter_convex)
qed

```

```

lemma convex_rel_interior_Inter:
  fixes  $\mathcal{F} :: 'n::euclidean\_space\ set\ set$ 
  assumes  $\bigwedge S. S \in \mathcal{F} \implies convex\ S$ 
  and  $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )  $\neq \{\}$ 
  shows rel_interior ( $\bigcap \mathcal{F}$ )  $\subseteq$   $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )
proof –
  have convex ( $\bigcap \mathcal{F}$ )
    using assms convex_Inter by auto
  moreover
  have convex ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ ))
    using assms by (metis convex_rel_interior convex_INT)
  ultimately
  have rel_interior ( $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )) = rel_interior ( $\bigcap \mathcal{F}$ )
    using convex_Inter_rel_interior_same_closure assms
    closure_eq_rel_interior_eq[of  $\bigcap$ (rel_interior ‘  $\mathcal{F}$ )  $\bigcap \mathcal{F}$ ]
    by blast
  then show ?thesis

```

using *rel_interior_subset*[of $\bigcap (\text{rel_interior } ' \mathcal{F})$] by auto
qed

lemma *convex_rel_interior_finite_Inter*:

fixes $\mathcal{F} :: 'n::\text{euclidean_space set set}$

assumes $\bigwedge S. S \in \mathcal{F} \implies \text{convex } S$

and $\bigcap (\text{rel_interior } ' \mathcal{F}) \neq \{\}$

and *finite* \mathcal{F}

shows $\text{rel_interior } (\bigcap \mathcal{F}) = \bigcap (\text{rel_interior } ' \mathcal{F})$

proof -

have $\bigcap \mathcal{F} \neq \{\}$

using *assms rel_interior_inter_aux*[of \mathcal{F}] by auto

have *convex* $(\bigcap \mathcal{F})$

using *convex_Inter assms* by auto

show ?thesis

proof (cases $\mathcal{F} = \{\}$)

case *True*

then show ?thesis

using *Inter_empty rel_interior_UNIV* by auto

next

case *False*

{

fix z

assume $z: z \in \bigcap (\text{rel_interior } ' \mathcal{F})$

{

fix x

assume $x: x \in \bigcap \mathcal{F}$

{

fix S

assume $S: S \in \mathcal{F}$

then have $z \in \text{rel_interior } S \ x \in S$

using $z \ x$ by auto

then have $\exists m. m > 1 \wedge (\forall e. e > 1 \wedge e \leq m \longrightarrow (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}}$

$z \in S)$

using *convex_rel_interior_if*[of $S \ z$] *S assms hull_subset*[of S] by auto

}

then obtain mS where

$mS: \forall S \in \mathcal{F}. mS \ S > 1 \wedge (\forall e. e > 1 \wedge e \leq mS \ S \longrightarrow (1 - e) *_{\mathbb{R}} x + e$

$*_{\mathbb{R}} z \in S)$ by *metis*

define e where $e = \text{Min } (mS ' \mathcal{F})$

then have $e \in mS ' \mathcal{F}$ using *assms* $\langle \mathcal{F} \neq \{\} \rangle$ by *simp*

then have $e > 1$ using mS by auto

moreover have $\forall S \in \mathcal{F}. e \leq mS \ S$

using e_def *assms* by auto

ultimately have $\exists e > 1. (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in \bigcap \mathcal{F}$

using mS by auto

}

then have $z \in \text{rel_interior } (\bigcap \mathcal{F})$

using *convex_rel_interior_iff*[of $\bigcap \mathcal{F} \ z$] $\langle \bigcap \mathcal{F} \neq \{\} \rangle$ $\langle \text{convex } (\bigcap \mathcal{F}) \rangle$ by

```

auto
}
then show ?thesis
  using convex_rel_interior_Inter[of  $\mathcal{F}$ ] assms by auto
qed
qed

lemma closure_Int_convex:
  fixes  $S T :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  and convex  $T$ 
  assumes  $rel\_interior\ S \cap rel\_interior\ T \neq \{\}$ 
  shows  $closure\ (S \cap T) = closure\ S \cap closure\ T$ 
  using closure_Inter_convex[of  $\{S, T\}$ ] assms by auto

lemma convex_rel_interior_inter_two:
  fixes  $S T :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  and convex  $T$ 
  and  $rel\_interior\ S \cap rel\_interior\ T \neq \{\}$ 
  shows  $rel\_interior\ (S \cap T) = rel\_interior\ S \cap rel\_interior\ T$ 
  using convex_rel_interior_finite_Inter[of  $\{S, T\}$ ] assms by auto

lemma convex_affine_closure_Int:
  fixes  $S T :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  and affine  $T$ 
  and  $rel\_interior\ S \cap T \neq \{\}$ 
  shows  $closure\ (S \cap T) = closure\ S \cap T$ 
  by (metis affine_imp_convex assms closure_Int_convex rel_interior_affine rel_interior_eq_closure)

lemma connected_component_1_gen:
  fixes  $S :: 'a :: euclidean\_space$  set
  assumes  $DIM('a) = 1$ 
  shows  $connected\_component\ S\ a\ b \iff closed\_segment\ a\ b \subseteq S$ 
  unfolding connected_component_def
  by (metis (no_types, lifting) assms subsetD subsetI convex_contains_segment convex_segment(1)
    ends_in_segment connected_convex_1_gen)

lemma connected_component_1:
  fixes  $S :: real$  set
  shows  $connected\_component\ S\ a\ b \iff closed\_segment\ a\ b \subseteq S$ 
  by (simp add: connected_component_1_gen)

lemma convex_affine_rel_interior_Int:
  fixes  $S T :: 'n::euclidean\_space$  set
  assumes convex  $S$ 
  and affine  $T$ 

```

```

    and rel_interior S ∩ T ≠ {}
  shows rel_interior (S ∩ T) = rel_interior S ∩ T
  by (simp add: affine_imp_convex assms convex_rel_interior_inter_two rel_interior_affine)

```

```

lemma convex_affine_rel_frontier_Int:
  fixes S T :: 'n::euclidean_space set
  assumes convex S
    and affine T
    and interior S ∩ T ≠ {}
  shows rel_frontier(S ∩ T) = frontier S ∩ T
using assms
  unfolding rel_frontier_def frontier_def
  using convex_affine_closure_Int convex_affine_rel_interior_Int rel_interior_nonempty_interior
  by fastforce

```

```

lemma rel_interior_convex_Int_affine:
  fixes S :: 'a::euclidean_space set
  assumes convex S affine T interior S ∩ T ≠ {}
  shows rel_interior(S ∩ T) = interior S ∩ T
  by (metis Int_emptyI assms convex_affine_rel_interior_Int empty_iff interior_rel_interior_gen)

```

```

lemma subset_rel_interior_convex:
  fixes S T :: 'n::euclidean_space set
  assumes convex S
    and convex T
    and S ≤ closure T
    and ¬ S ⊆ rel_frontier T
  shows rel_interior S ⊆ rel_interior T
proof -
  have *: S ∩ closure T = S
    using assms by auto
  have ¬ rel_interior S ⊆ rel_frontier T
    using closure_mono[of rel_interior S rel_frontier T] closed_rel_frontier[of T]
      closure_closed[of S] convex_closure_rel_interior[of S] closure_subset[of S]
  assms
  by auto
  then have rel_interior S ∩ rel_interior (closure T) ≠ {}
    using assms rel_frontier_def[of T] rel_interior_subset convex_rel_interior_closure[of T]
  by auto
  then have rel_interior S ∩ rel_interior T = rel_interior (S ∩ closure T)
    using assms convex_closure convex_rel_interior_inter_two[of S closure T]
      convex_rel_interior_closure[of T]
  by auto
  also have ... = rel_interior S
    using * by auto
  finally show ?thesis
    by auto

```

qed

lemma *rel_interior_convex_linear_image*:

fixes $f :: 'm::euclidean_space \Rightarrow 'n::euclidean_space$

assumes *linear f*

and *convex S*

shows $f^{-1}(\text{rel_interior } S) = \text{rel_interior } (f^{-1} S)$

proof (cases $S = \{\}$)

case *True*

then show *?thesis*

using *assms* by auto

next

case *False*

interpret *linear f* by fact

have $*$: $f^{-1}(\text{rel_interior } S) \subseteq f^{-1} S$

unfolding *image_mono* using *rel_interior_subset* by auto

have $f^{-1} S \subseteq f^{-1}(\text{closure } S)$

unfolding *image_mono* using *closure_subset* by auto

also have $\dots = f^{-1}(\text{closure } (\text{rel_interior } S))$

using *convex_closure_rel_interior* *assms* by auto

also have $\dots \subseteq \text{closure } (f^{-1}(\text{rel_interior } S))$

using *closure_linear_image_subset* *assms* by auto

finally have $\text{closure } (f^{-1} S) = \text{closure } (f^{-1} \text{rel_interior } S)$

using *closure_mono*[of $f^{-1} S$ $\text{closure } (f^{-1} \text{rel_interior } S)$] *closure_closure*

closure_mono[of $f^{-1} \text{rel_interior } S$ $f^{-1} S$] $*$

by auto

then have $\text{rel_interior } (f^{-1} S) = \text{rel_interior } (f^{-1} \text{rel_interior } S)$

using *assms* *convex_rel_interior*

linear_conv_bounded_linear[of f] *convex_linear_image*[of $_ S$]

convex_linear_image[of $_ \text{rel_interior } S$]

closure_eq_rel_interior_eq[of $f^{-1} S$ $f^{-1} \text{rel_interior } S$]

by auto

then have $\text{rel_interior } (f^{-1} S) \subseteq f^{-1} \text{rel_interior } S$

using *rel_interior_subset* by auto

moreover

{

fix z

assume $z \in f^{-1} \text{rel_interior } S$

then obtain $z1$ where $z1: z1 \in \text{rel_interior } S$ $f z1 = z$ by auto

{

fix x

assume $x \in f^{-1} S$

then obtain $x1$ where $x1: x1 \in S$ $f x1 = x$ by auto

then obtain e where $e: e > 1$ $(1 - e) *_R x1 + e *_R z1 \in S$

using *convex_rel_interior_iff*[of S $z1$] $\langle \text{convex } S \rangle$ $x1$ $z1$ by auto

moreover have $f((1 - e) *_R x1 + e *_R z1) = (1 - e) *_R x + e *_R z$

using $x1$ $z1$ by (*simp add: linear_add linear_scale* $\langle \text{linear } f \rangle$)

ultimately have $(1 - e) *_R x + e *_R z \in f^{-1} S$

using *imageI*[of $(1 - e) *_R x1 + e *_R z1$ S f] by auto

```

    then have  $\exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in f^{-1} S$ 
      using  $e$  by auto
  }
  then have  $z \in \text{rel\_interior } (f^{-1} S)$ 
    using  $\text{convex\_rel\_interior\_iff}[of f^{-1} S z]$   $\langle \text{convex } S \rangle$   $\langle \text{linear } f \rangle$ 
       $\langle S \neq \{\} \rangle$   $\text{convex\_linear\_image}[of f S]$   $\text{linear\_conv\_bounded\_linear}[of f]$ 
    by auto
  }
  ultimately show ?thesis by auto
qed

```

lemma *rel_interior_convex_linear_preimage*:

fixes $f :: 'm::\text{euclidean_space} \Rightarrow 'n::\text{euclidean_space}$

assumes *linear* f

and *convex* S

and $f^{-1}(\text{rel_interior } S) \neq \{\}$

shows $\text{rel_interior } (f^{-1} S) = f^{-1}(\text{rel_interior } S)$

proof -

interpret *linear* f by *fact*

have $S \neq \{\}$

using *assms* by *auto*

have *nonemp*: $f^{-1} S \neq \{\}$

by (*metis* *assms*(3)) *rel_interior_subset subset_empty vimage_mono*)

then have $S \cap (\text{range } f) \neq \{\}$

by *auto*

have *conv*: *convex* $(f^{-1} S)$

using *convex_linear_vimage* *assms* by *auto*

then have *conv* $(S \cap \text{range } f)$

by (*simp* *add*: *assms*(2)) *convex_Int convex_linear_image linear_axioms*)

{

fix z

assume $z \in f^{-1}(\text{rel_interior } S)$

then have $z: f z \in \text{rel_interior } S$

by *auto*

{

fix x

assume $x \in f^{-1} S$

then have $f x \in S$ by *auto*

then obtain e where $e: e > 1 \wedge (1 - e) *_{\mathbb{R}} f x + e *_{\mathbb{R}} f z \in S$

using *convex_rel_interior_iff*[*of* $S f z$] z *assms* $\langle S \neq \{\} \rangle$ by *auto*

moreover have $(1 - e) *_{\mathbb{R}} f x + e *_{\mathbb{R}} f z = f((1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z)$

using $\langle \text{linear } f \rangle$ by (*simp* *add*: *linear_iff*)

ultimately have $\exists e. e > 1 \wedge (1 - e) *_{\mathbb{R}} x + e *_{\mathbb{R}} z \in f^{-1} S$

using e by *auto*

}

then have $z \in \text{rel_interior } (f^{-1} S)$

using *convex_rel_interior_iff*[*of* $f^{-1} S z$] *conv* *nonemp* by *auto*

}

moreover


```

{
  fix z
  assume z: z ∈ rel_interior (f -' S)
  {
    fix x
    assume x ∈ S ∩ range f
    then obtain y where y: f y = x y ∈ f -' S by auto
    then obtain e where e: e > 1 (1 - e) *R y + e *R z ∈ f -' S
      using convex_rel_interior_iff[of f -' S z] z conv by auto
    moreover have (1 - e) *R x + e *R f z = f ((1 - e) *R y + e *R z)
      using ⟨linear f⟩ y by (simp add: linear_iff)
    ultimately have ∃ e. e > 1 ∧ (1 - e) *R x + e *R f z ∈ S ∩ range f
      using e by auto
  }
  then have f z ∈ rel_interior (S ∩ range f)
    using ⟨convex (S ∩ (range f))⟩ ⟨S ∩ range f ≠ {}⟩
      convex_rel_interior_iff[of S ∩ (range f) f z]
    by auto
  moreover have affine (range f)
    by (simp add: linear_axioms linear_subspace_image subspace_imp_affine)
  ultimately have f z ∈ rel_interior S
    using convex_affine_rel_interior_Int[of S range f] assms by auto
  then have z ∈ f -' (rel_interior S)
    by auto
}
ultimately show ?thesis by auto
qed

lemma rel_interior_Times:
  fixes S :: 'n::euclidean_space set
    and T :: 'm::euclidean_space set
  assumes convex S
    and convex T
  shows rel_interior (S × T) = rel_interior S × rel_interior T
proof (cases S = {} ∨ T = {})
  case True
  then show ?thesis
    by auto
next
  case False
  then have S ≠ {} T ≠ {}
    by auto
  then have ri: rel_interior S ≠ {} rel_interior T ≠ {}
    using rel_interior_eq_empty assms by auto
  then have fst -' rel_interior S ≠ {}
    using fst_vimage_eq_Times[of rel_interior S] by auto
  then have rel_interior ((fst :: 'n * 'm ⇒ 'n) -' S) = fst -' rel_interior S
    using linear_fst ⟨convex S⟩ rel_interior_convex_linear_preimage[of fst S] by
  auto

```

```

then have s: rel_interior (S × (UNIV :: 'm set)) = rel_interior S × UNIV
  by (simp add: fst_vimage_eq_Times)
from ri have snd - ' rel_interior T ≠ {}
  using snd_vimage_eq_Times[of rel_interior T] by auto
then have rel_interior ((snd :: 'n * 'm ⇒ 'm) - ' T) = snd - ' rel_interior T
  using linear_snd ⟨convex T⟩ rel_interior_convex_linear_preimage[of snd T]
by auto
then have t: rel_interior ((UNIV :: 'n set) × T) = UNIV × rel_interior T
  by (simp add: snd_vimage_eq_Times)
from s t have *: rel_interior (S × (UNIV :: 'm set)) ∩ rel_interior ((UNIV ::
'n set) × T) =
  rel_interior S × rel_interior T by auto
have S × T = S × (UNIV :: 'm set) ∩ (UNIV :: 'n set) × T
  by auto
then have rel_interior (S × T) = rel_interior ((S × (UNIV :: 'm set)) ∩
((UNIV :: 'n set) × T))
  by auto
also have ... = rel_interior (S × (UNIV :: 'm set)) ∩ rel_interior ((UNIV ::
'n set) × T)
  using * ri assms convex_Times
  by (subst convex_rel_interior_inter_two) auto
finally show ?thesis using * by auto
qed

```

lemma *rel_interior_scaleR*:

```

fixes S :: 'n::euclidean_space set
assumes c ≠ 0
shows ((*R) c) ' (rel_interior S) = rel_interior (((*R) c) ' S)
using rel_interior_injective_linear_image[of ((*R) c) S]
  linear_conv_bounded_linear[of (*R) c] linear_scaleR_injective_scaleR[of c]
assms
by auto

```

lemma *rel_interior_convex_scaleR*:

```

fixes S :: 'n::euclidean_space set
assumes convex S
shows ((*R) c) ' (rel_interior S) = rel_interior (((*R) c) ' S)
by (metis assms linear_scaleR rel_interior_convex_linear_image)

```

lemma *convex_rel_open_scaleR*:

```

fixes S :: 'n::euclidean_space set
assumes convex S
  and rel_open S
shows convex (((*R) c) ' S) ∧ rel_open (((*R) c) ' S)
by (metis assms convex_scaling rel_interior_convex_scaleR rel_open_def)

```

lemma *convex_rel_open_finite_Inter*:

```

fixes F :: 'n::euclidean_space set set
assumes ∧S. S ∈ F ⇒ convex S ∧ rel_open S

```

```

    and finite  $\mathcal{F}$ 
  shows  $\text{convex } (\bigcap \mathcal{F}) \wedge \text{rel\_open } (\bigcap \mathcal{F})$ 
proof (cases  $\bigcap \{\text{rel\_interior } S \mid S. S \in \mathcal{F}\} = \{\}$ )
  case True
  then have  $\bigcap \mathcal{F} = \{\}$ 
    using assms unfolding rel_open_def by auto
  then show ?thesis
    unfolding rel_open_def by auto
next
  case False
  then have  $\text{rel\_open } (\bigcap \mathcal{F})$ 
    using assms convex_rel_interior_finite_Inter[of  $\mathcal{F}$ ] by (force simp: rel_open_def)
  then show ?thesis
    using convex_Inter assms by auto
qed

lemma convex_rel_open_linear_image:
  fixes  $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes linear  $f$ 
    and convex  $S$ 
    and rel_open  $S$ 
  shows  $\text{convex } (f \text{ ` } S) \wedge \text{rel\_open } (f \text{ ` } S)$ 
  by (metis assms convex_linear_image rel_interior_convex_linear_image rel_open_def)

lemma convex_rel_open_linear_preimage:
  fixes  $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes linear  $f$ 
    and convex  $S$ 
    and rel_open  $S$ 
  shows  $\text{convex } (f \text{ - ` } S) \wedge \text{rel\_open } (f \text{ - ` } S)$ 
proof (cases  $f \text{ - ` } (\text{rel\_interior } S) = \{\}$ )
  case True
  then have  $f \text{ - ` } S = \{\}$ 
    using assms unfolding rel_open_def by auto
  then show ?thesis
    unfolding rel_open_def by auto
next
  case False
  then have  $\text{rel\_open } (f \text{ - ` } S)$ 
    using assms unfolding rel_open_def
    using rel_interior_convex_linear_preimage[of  $f$   $S$ ]
    by auto
  then show ?thesis
    using convex_linear_vimage assms
    by auto
qed

lemma rel_interior_projection:
  fixes  $S :: ('m::\text{euclidean\_space} \times 'n::\text{euclidean\_space}) \text{ set}$ 

```

```

    and f :: 'm::euclidean_space ⇒ 'n::euclidean_space set
  assumes convex S
    and f = (λy. {z. (y, z) ∈ S})
  shows (y, z) ∈ rel_interior S ⟷ (y ∈ rel_interior {y. (f y ≠ {})}) ∧ z ∈
rel_interior (f y)
proof -
  {
    fix y
    assume y ∈ {y. f y ≠ {}}
    then obtain z where (y, z) ∈ S
      using assms by auto
    then have ∃x. x ∈ S ∧ y = fst x
      by auto
    then obtain x where x ∈ S y = fst x
      by blast
    then have y ∈ fst ` S
      unfolding image_def by auto
  }
  then have fst ` S = {y. f y ≠ {}}
    unfolding fst_def using assms by auto
  then have h1: fst ` rel_interior S = rel_interior {y. f y ≠ {}}
    using rel_interior_convex_linear_image[of fst S] assms linear_fst by auto
  {
    fix y
    assume y ∈ rel_interior {y. f y ≠ {}}
    then have y ∈ fst ` rel_interior S
      using h1 by auto
    then have *: rel_interior S ∩ fst -` {y} ≠ {}
      by auto
    moreover have aff: affine (fst -` {y})
      unfolding affine_alt by (simp add: algebra_simps)
    ultimately have **: rel_interior (S ∩ fst -` {y}) = rel_interior S ∩ fst -`
{y}
      using convex_affine_rel_interior_Int[of S fst -` {y}] assms by auto
    have conv: convex (S ∩ fst -` {y})
      using convex_Int assms aff affine_imp_convex by auto
    {
      fix x
      assume x ∈ f y
      then have (y, x) ∈ S ∩ (fst -` {y})
        using assms by auto
      moreover have x = snd (y, x) by auto
      ultimately have x ∈ snd ` (S ∩ fst -` {y})
        by blast
    }
  }
  then have snd ` (S ∩ fst -` {y}) = f y
    using assms by auto
  then have ***: rel_interior (f y) = snd ` rel_interior (S ∩ fst -` {y})
    using rel_interior_convex_linear_image[of snd S ∩ fst -` {y}] linear_snd

```

```

conv
  by auto
  {
    fix z
    assume  $z \in \text{rel\_interior } (f y)$ 
    then have  $z \in \text{snd } ' \text{rel\_interior } (S \cap \text{fst } - ' \{y\})$ 
      using *** by auto
    moreover have  $\{y\} = \text{fst } ' \text{rel\_interior } (S \cap \text{fst } - ' \{y\})$ 
      using * ** rel_interior_subset by auto
    ultimately have  $(y, z) \in \text{rel\_interior } (S \cap \text{fst } - ' \{y\})$ 
      by force
    then have  $(y, z) \in \text{rel\_interior } S$ 
      using ** by auto
  }
  moreover
  {
    fix z
    assume  $(y, z) \in \text{rel\_interior } S$ 
    then have  $(y, z) \in \text{rel\_interior } (S \cap \text{fst } - ' \{y\})$ 
      using ** by auto
    then have  $z \in \text{snd } ' \text{rel\_interior } (S \cap \text{fst } - ' \{y\})$ 
      by (metis Range_iff_snd_eq_Range)
    then have  $z \in \text{rel\_interior } (f y)$ 
      using *** by auto
  }
  ultimately have  $\bigwedge z. (y, z) \in \text{rel\_interior } S \longleftrightarrow z \in \text{rel\_interior } (f y)$ 
    by auto
}
then have h2:  $\bigwedge y z. y \in \text{rel\_interior } \{t. f t \neq \{\}\} \implies$ 
   $(y, z) \in \text{rel\_interior } S \longleftrightarrow z \in \text{rel\_interior } (f y)$ 
  by auto
{
  fix y z
  assume asm:  $(y, z) \in \text{rel\_interior } S$ 
  then have  $y \in \text{fst } ' \text{rel\_interior } S$ 
    by (metis Domain_iff_fst_eq_Domain)
  then have  $y \in \text{rel\_interior } \{t. f t \neq \{\}\}$ 
    using h1 by auto
  then have  $y \in \text{rel\_interior } \{t. f t \neq \{\}\}$  and  $(z \in \text{rel\_interior } (f y))$ 
    using h2 asm by auto
}
then show ?thesis using h2 by blast
qed

```

lemma rel_frontier_Times:

```

fixes  $S :: 'n::\text{euclidean\_space}$  set
  and  $T :: 'm::\text{euclidean\_space}$  set
assumes convex S
  and convex T

```

shows $rel_frontier\ S \times rel_frontier\ T \subseteq rel_frontier\ (S \times T)$
 by (force simp: rel_frontier_def rel_interior_Times assms closure_Times)

Relative interior of convex cone

lemma cone_rel_interior:

fixes $S :: 'm::euclidean_space\ set$

assumes cone S

shows cone $(\{0\} \cup rel_interior\ S)$

proof (cases $S = \{0\}$)

case True

then show ?thesis

by (simp add: cone_0)

next

case False

then have *: $0 \in S \wedge (\forall c. c > 0 \longrightarrow (*_R)\ c \cdot S = S)$

using cone_iff[of S] assms by auto

then have *: $0 \in (\{0\} \cup rel_interior\ S)$

and $\forall c. c > 0 \longrightarrow (*_R)\ c \cdot (\{0\} \cup rel_interior\ S) = (\{0\} \cup rel_interior\ S)$

by (auto simp add: rel_interior_scaleR)

then show ?thesis

using cone_iff[of $\{0\} \cup rel_interior\ S$] by auto

qed

lemma rel_interior_convex_cone_aux:

fixes $S :: 'm::euclidean_space\ set$

assumes convex S

shows $(c, x) \in rel_interior\ (cone\ hull\ (\{1 :: real\} \times S)) \iff$

$c > 0 \wedge x \in ((*_R)\ c \cdot (rel_interior\ S))$

proof (cases $S = \{0\}$)

case True

then show ?thesis

by (simp add: cone_hull_empty)

next

case False

then obtain s where $s \in S$ by auto

have conv: convex $(\{1 :: real\} \times S)$

using convex_Times[of $\{1 :: real\}$ S] assms convex_singleton[of $1 :: real$]

by auto

define f where $f\ y = \{z. (y, z) \in cone\ hull\ (\{1 :: real\} \times S)\}$ for y

then have *: $(c, x) \in rel_interior\ (cone\ hull\ (\{1 :: real\} \times S)) =$

$(c \in rel_interior\ \{y. f\ y \neq \{0\}\} \wedge x \in rel_interior\ (f\ c))$

using convex_cone_hull[of $\{1 :: real\} \times S$] conv

by (subst rel_interior_projection) auto

{

fix $y :: real$

assume $y \geq 0$

then have $y *_R\ (1, s) \in cone\ hull\ (\{1 :: real\} \times S)$

using cone_hull_expl[of $\{1 :: real\} \times S$] $\langle s \in S \rangle$ by auto

```

    then have  $f y \neq \{\}$ 
      using  $f\_def$  by auto
  }
  then have  $\{y. f y \neq \{\}\} = \{0..\}$ 
    using  $f\_def$   $cone\_hull\_expl$ [of  $\{1 :: real\} \times S$ ] by auto
  then have **:  $rel\_interior \{y. f y \neq \{\}\} = \{0<..\}$ 
    using  $rel\_interior\_real\_semiline$  by auto
  {
    fix  $c :: real$ 
    assume  $c > 0$ 
    then have  $f c = ((*_R) c ' S)$ 
      using  $f\_def$   $cone\_hull\_expl$ [of  $\{1 :: real\} \times S$ ] by auto
    then have  $rel\_interior (f c) = (*_R) c ' rel\_interior S$ 
      using  $rel\_interior\_convex\_scaleR$ [of  $S c$ ] assms by auto
  }
  then show ?thesis using * ** by auto
qed

```

lemma $rel_interior_convex_cone$:

fixes $S :: 'm::euclidean_space$ set

assumes $convex S$

shows $rel_interior (cone_hull (\{1 :: real\} \times S)) =$

$\{(c, c *_R x) \mid c x. c > 0 \wedge x \in rel_interior S\}$

(**is** ?lhs = ?rhs)

proof –

```

  {
    fix  $z$ 
    assume  $z \in ?lhs$ 
    have *:  $z = (fst z, snd z)$ 
      by auto
    then have  $z \in ?rhs$ 
      using  $rel\_interior\_convex\_cone\_aux$ [of  $S$   $fst z$   $snd z$ ] assms  $\langle z \in ?lhs \rangle$  by

```

fastforce

```

  }

```

moreover

```

  {
    fix  $z$ 
    assume  $z \in ?rhs$ 
    then have  $z \in ?lhs$ 
      using  $rel\_interior\_convex\_cone\_aux$ [of  $S$   $fst z$   $snd z$ ] assms
      by auto
  }

```

ultimately show ?thesis **by** *blast*

qed

lemma $convex_hull_finite_union$:

assumes $finite I$

assumes $\forall i \in I. convex (S i) \wedge (S i) \neq \{\}$

shows $convex_hull (\bigcup (S ' I)) =$

```

    {sum (λi. c i *R s i) I | c s. (∀i∈I. c i ≥ 0) ∧ sum c I = 1 ∧ (∀i∈I. s i ∈ S
i)}
  (is ?lhs = ?rhs)
proof -
  have ?lhs ⊇ ?rhs
  proof
    fix x
    assume x ∈ ?rhs
    then obtain c s where *: sum (λi. c i *R s i) I = x sum c I = 1
      (∀i∈I. c i ≥ 0) ∧ (∀i∈I. s i ∈ S i) by auto
    then have ∀i∈I. s i ∈ convex hull (∪(S ' I))
      using hull_subset[of ∪(S ' I) convex] by auto
    then show x ∈ ?lhs
      unfolding *(1)[symmetric]
      using * assms convex_convex_hull
      by (subst convex_sum) auto
  qed
  {
    fix i
    assume i ∈ I
    with assms have ∃p. p ∈ S i by auto
  }
  then obtain p where p: ∀i∈I. p i ∈ S i bymetis
  {
    fix i
    assume i ∈ I
    {
      fix x
      assume x ∈ S i
      define c where c j = (if j = i then 1::real else 0) for j
      then have *: sum c I = 1
        using ⟨finite I⟩ ⟨i ∈ I⟩ sum.delta[of I i λj::'a. 1::real]
        by auto
      define s where s j = (if j = i then x else p j) for j
      then have ∀j. c j *R s j = (if j = i then x else 0)
        using c_def by (auto simp add: algebra_simps)
      then have x = sum (λi. c i *R s i) I
        using s_def c_def ⟨finite I⟩ ⟨i ∈ I⟩ sum.delta[of I i λj::'a. x]
        by auto
      moreover have (∀i∈I. 0 ≤ c i) ∧ sum c I = 1 ∧ (∀i∈I. s i ∈ S i)
        using * c_def s_def p ⟨x ∈ S i⟩ by auto
      ultimately have x ∈ ?rhs
        by force
    }
  }
  then have ?rhs ⊇ S i by auto
}
then have *: ?rhs ⊇ ∪(S ' I) by auto
{

```



```

fix u v :: real
assume uv: u ≥ 0 ∧ v ≥ 0 ∧ u + v = 1
fix x y
assume xy: x ∈ ?rhs ∧ y ∈ ?rhs
from xy obtain c s where
  xc: x = sum (λi. c i *R s i) I ∧ (∀i∈I. c i ≥ 0) ∧ sum c I = 1 ∧ (∀i∈I. s
i ∈ S i)
  by auto
from xy obtain d t where
  yc: y = sum (λi. d i *R t i) I ∧ (∀i∈I. d i ≥ 0) ∧ sum d I = 1 ∧ (∀i∈I. t
i ∈ S i)
  by auto
define e where e i = u * c i + v * d i for i
have ge0: ∀i∈I. e i ≥ 0
  using e_def xc yc uv by simp
have sum (λi. u * c i) I = u * sum c I
  by (simp add: sum_distrib_left)
moreover have sum (λi. v * d i) I = v * sum d I
  by (simp add: sum_distrib_left)
ultimately have sum1: sum e I = 1
  using e_def xc yc uv by (simp add: sum.distrib)
define q where q i = (if e i = 0 then p i else (u * c i / e i) *R s i + (v * d i
/ e i) *R t i)
  for i
  {
  fix i
  assume i: i ∈ I
  have q i ∈ S i
  proof (cases e i = 0)
  case True
  then show ?thesis using i p q_def by auto
  next
  case False
  then show ?thesis
  using mem_convex_alt[of S i s i t i u * (c i) v * (d i)]
    mult_nonneg_nonneg[of u c i] mult_nonneg_nonneg[of v d i]
    assms q_def e_def i False xc yc uv
  by (auto simp del: mult_nonneg_nonneg)
  qed
  }
then have qs: ∀i∈I. q i ∈ S i by auto
  {
  fix i
  assume i: i ∈ I
  have (u * c i) *R s i + (v * d i) *R t i = e i *R q i
  proof (cases e i = 0)
  case True
  have ge: u * (c i) ≥ 0 ∧ v * d i ≥ 0
  using xc yc uv i by simp

```

```

moreover from ge have  $u * c \ i \leq 0 \wedge v * d \ i \leq 0$ 
  using True e_def i by simp
ultimately have  $u * c \ i = 0 \wedge v * d \ i = 0$  by auto
with True show ?thesis by auto
next
case False
then have  $(u * (c \ i) / (e \ i)) *_R (s \ i) + (v * (d \ i) / (e \ i)) *_R (t \ i) = q \ i$ 
  using q_def by auto
then have  $e \ i *_R ((u * (c \ i) / (e \ i)) *_R (s \ i) + (v * (d \ i) / (e \ i)) *_R (t \ i))$ 
   $= (e \ i) *_R (q \ i)$  by auto
with False show ?thesis by (simp add: algebra_simps)
qed
}
then have  $*$ :  $\forall i \in I. (u * c \ i) *_R s \ i + (v * d \ i) *_R t \ i = e \ i *_R q \ i$ 
by auto
have  $u *_R x + v *_R y = \text{sum } (\lambda i. (u * c \ i) *_R s \ i + (v * d \ i) *_R t \ i) \ I$ 
using xc yc by (simp add: algebra_simps scaleR_right.sum sum.distrib)
also have  $\dots = \text{sum } (\lambda i. e \ i *_R q \ i) \ I$ 
using  $*$  by auto
finally have  $u *_R x + v *_R y = \text{sum } (\lambda i. (e \ i) *_R (q \ i)) \ I$ 
by auto
then have  $u *_R x + v *_R y \in ?rhs$ 
using ge0 sum1 qs by auto
}
then have convex ?rhs unfolding convex_def by auto
then show ?thesis
  using  $\langle ?lhs \supseteq ?rhs \rangle * \text{hull\_minimal}[of \bigcup (S \ ' I) \ ?rhs \ \text{convex}]$ 
  by blast
qed

lemma convex_hull_union_two:
fixes  $S \ T :: 'm::\text{euclidean\_space} \ \text{set}$ 
assumes convex S
  and  $S \neq \{\}$ 
  and convex T
  and  $T \neq \{\}$ 
shows convex hull  $(S \cup T) =$ 
 $\{u *_R s + v *_R t \mid u \ v \ s \ t. u \geq 0 \wedge v \geq 0 \wedge u + v = 1 \wedge s \in S \wedge t \in T\}$ 
(is ?lhs = ?rhs)
proof
define  $I :: \text{nat set}$  where  $I = \{1, 2\}$ 
define  $s$  where  $s \ i = (\text{if } i = (1::\text{nat}) \ \text{then } S \ \text{else } T)$  for  $i$ 
have  $\bigcup (s \ ' I) = S \cup T$ 
using s_def I_def by auto
then have convex hull  $(\bigcup (s \ ' I)) = \text{convex hull } (S \cup T)$ 
by auto
moreover have convex hull  $\bigcup (s \ ' I) =$ 
 $\{\sum_{i \in I} c \ i *_R s \ a \ i \mid c \ sa. (\forall i \in I. 0 \leq c \ i) \wedge \text{sum } c \ I = 1 \wedge (\forall i \in I. s \ a \ i \in s$ 
 $i)\}$ 

```

```

    using assms s_def I_def
    by (subst convex_hull_finite_union) auto
  moreover have
    { $\sum_{i \in I}. c_i *_{\mathbb{R}} s_i \mid c \text{ sa. } (\forall i \in I. 0 \leq c_i) \wedge \text{sum } c \ I = 1 \wedge (\forall i \in I. s_i \in s)$ }  $\leq$  ?rhs
    using s_def I_def by auto
  ultimately show ?lhs  $\subseteq$  ?rhs by auto
  {
    fix x
    assume x  $\in$  ?rhs
    then obtain u v s t where *:  $x = u *_{\mathbb{R}} s + v *_{\mathbb{R}} t \wedge u \geq 0 \wedge v \geq 0 \wedge u + v = 1 \wedge s \in S \wedge t \in T$ 
    by auto
    then have  $x \in \text{convex hull } \{s, t\}$ 
    using convex_hull_2[of s t] by auto
    then have  $x \in \text{convex hull } (S \cup T)$ 
    using * hull_mono[of {s, t} S  $\cup$  T] by auto
  }
  then show ?lhs  $\supseteq$  ?rhs by blast
qed

```

proposition ray_to_rel_frontier:

fixes $a :: 'a::\text{real_inner}$

assumes bounded S

and $a: a \in \text{rel_interior } S$

and aff: $(a + l) \in \text{affine hull } S$

and $l \neq 0$

obtains d where $0 < d$ $(a + d *_{\mathbb{R}} l) \in \text{rel_frontier } S$

$\wedge e. [0 \leq e; e < d] \implies (a + e *_{\mathbb{R}} l) \in \text{rel_interior } S$

proof –

have aaff: $a \in \text{affine hull } S$

by (meson a_hull_subset_rel_interior_subset_rev_subsetD)

let ?D = $\{d. 0 < d \wedge a + d *_{\mathbb{R}} l \notin \text{rel_interior } S\}$

obtain B where $B > 0$ and $B: S \subseteq \text{ball } a \ B$

using bounded_subset_ballD [OF \langle bounded S \rangle] by blast

have $a + (B / \text{norm } l) *_{\mathbb{R}} l \notin \text{ball } a \ B$

by (simp add: dist_norm \langle l \neq 0 \rangle)

with B have $a + (B / \text{norm } l) *_{\mathbb{R}} l \notin \text{rel_interior } S$

using rel_interior_subset_subsetCE by blast

with $\langle B > 0 \rangle$ $\langle l \neq 0 \rangle$ have nonMT: $?D \neq \{\}$

using divide_pos_pos_zero_less_norm_iff by fastforce

have bdd: bdd_below ?D

by (metis (no_types, lifting) bdd_belowI le_less mem_Collect_eq)

have relin_Ex: $\bigwedge x. x \in \text{rel_interior } S \implies$

$\exists e > 0. \forall x' \in \text{affine hull } S. \text{dist } x' \ x < e \implies x' \in \text{rel_interior } S$

using openin_rel_interior [of S] by (simp add: openin_euclidean_subtopology_iff)

define d where $d = \text{Inf } ?D$

obtain ε where $0 < \varepsilon$ and $\varepsilon: \bigwedge \eta. [0 \leq \eta; \eta < \varepsilon] \implies (a + \eta *_{\mathbb{R}} l) \in \text{rel_interior } S$

S

```

proof –
  obtain  $e$  where  $e > 0$ 
    and  $e: \bigwedge x'. x' \in \text{affine hull } S \implies \text{dist } x' a < e \implies x' \in \text{rel\_interior } S$ 
    using  $\text{relin\_Ex } a$  by  $\text{blast}$ 
  show  $\text{thesis}$ 
  proof ( $\text{rule\_tac } \varepsilon = e / \text{norm } l$  in  $\text{that}$ )
    show  $0 < e / \text{norm } l$  by ( $\text{simp add: } \langle 0 < e \rangle \langle l \neq 0 \rangle$ )
  next
    show  $a + \eta *_R l \in \text{rel\_interior } S$  if  $0 \leq \eta$   $\eta < e / \text{norm } l$  for  $\eta$ 
    proof ( $\text{rule } e$ )
      show  $a + \eta *_R l \in \text{affine hull } S$ 
      by ( $\text{metis (no\_types) add\_diff\_cancel\_left' aff affine\_affine\_hull mem\_affine\_3\_minus}$ 
 $\text{aaff}$ )
      show  $\text{dist } (a + \eta *_R l) a < e$ 
      using  $\text{that}$  by ( $\text{simp add: } \langle l \neq 0 \rangle \text{dist\_norm\_pos\_less\_divide\_eq}$ )
    qed
  qed
  qed
  have  $\text{inint: } \bigwedge e. \llbracket 0 \leq e; e < d \rrbracket \implies a + e *_R l \in \text{rel\_interior } S$ 
    unfolding  $d\_def$  using  $cInf\_lower$  [ $OF\_bdd$ ]
    by ( $\text{metis (no\_types, lifting) a add.right\_neutral le\_less mem\_Collect\_eq}$ 
 $\text{not\_less real\_vector.scale\_zero\_left}$ )
  have  $\varepsilon \leq d$ 
    unfolding  $d\_def$ 
    using  $\varepsilon \text{ dual\_order.strict\_implies\_order le\_less\_linear}$ 
    by ( $\text{blast intro: cInf\_greatest [OF nonMT]}$ )
  with  $\langle 0 < \varepsilon \rangle$  have  $0 < d$  by  $\text{simp}$ 
  have  $a + d *_R l \notin \text{rel\_interior } S$ 
  proof
    assume  $\text{adl: } a + d *_R l \in \text{rel\_interior } S$ 
    obtain  $e$  where  $e > 0$ 
    and  $e: \bigwedge x'. x' \in \text{affine hull } S \implies \text{dist } x' (a + d *_R l) < e \implies x' \in \text{rel\_interior}$ 
 $S$ 
    using  $\text{relin\_Ex adl}$  by  $\text{blast}$ 
    have  $d + e / \text{norm } l \leq x$ 
    if  $0 < x$  and  $\text{nonrel: } a + x *_R l \notin \text{rel\_interior } S$  for  $x$ 
    proof ( $\text{cases } x < d$ )
      case  $\text{True}$  with  $\text{inint nonrel } \langle 0 < x \rangle$ 
      show  $?thesis$  by  $\text{auto}$ 
    next
      case  $\text{False}$ 
      then have  $\text{dle: } x < d + e / \text{norm } l \implies \text{dist } (a + x *_R l) (a + d *_R l) < e$ 
      by ( $\text{simp add: field\_simps } \langle l \neq 0 \rangle$ )
      have  $\text{ain: } a + x *_R l \in \text{affine hull } S$ 
      by ( $\text{metis add\_diff\_cancel\_left' aff affine\_affine\_hull mem\_affine\_3\_minus}$ 
 $\text{aaff}$ )
      show  $?thesis$ 
      using  $e$  [ $OF \text{ ain}$ ]  $\text{nonrel dle}$  by  $\text{force}$ 
    qed
  
```

```

then
  have  $d + e / \text{norm } l \leq \text{Inf } \{d. 0 < d \wedge a + d *_R l \notin \text{rel\_interior } S\}$ 
    by (force simp add: intro: cInf_greatest [OF nonMT])
  then show False
    using  $\langle 0 < e \rangle \langle l \neq 0 \rangle$  by (simp add: d_def [symmetric] field_simps)
qed
moreover
have  $\exists y \in S. \text{dist } y (a + d *_R l) < \eta$  if  $0 < \eta$  for  $\eta :: \text{real}$ 
proof -
  have  $1: a + (d - \min d (\eta / 2 / \text{norm } l)) *_R l \in S$ 
  proof (rule subsetD [OF rel_interior_subset inint])
    show  $d - \min d (\eta / 2 / \text{norm } l) < d$ 
      using  $\langle l \neq 0 \rangle \langle 0 < d \rangle \langle 0 < \eta \rangle$  by auto
  qed auto
  have  $\text{norm } l * \min d (\eta / (\text{norm } l * 2)) \leq \text{norm } l * (\eta / (\text{norm } l * 2))$ 
    by (metis min_def mult_left_mono norm_ge_zero order_refl)
  also have  $\dots < \eta$ 
    using  $\langle l \neq 0 \rangle \langle 0 < \eta \rangle$  by (simp add: field_simps)
  finally have  $2: \text{norm } l * \min d (\eta / (\text{norm } l * 2)) < \eta$  .
  show ?thesis
    using 1 2  $\langle 0 < d \rangle \langle 0 < \eta \rangle$ 
    by (rule_tac  $x = a + (d - \min d (\eta / 2 / \text{norm } l)) *_R l$  in bexI) (auto simp:
algebra_simps)
qed
then have  $a + d *_R l \in \text{closure } S$ 
  by (auto simp: closure_approachable)
ultimately have  $\text{infront}: a + d *_R l \in \text{rel\_frontier } S$ 
  by (simp add: rel_frontier_def)
show ?thesis
  by (rule that [OF  $\langle 0 < d \rangle \text{infront inint}$ ])
qed

corollary ray_to_frontier:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes bounded S
    and  $a: a \in \text{interior } S$ 
    and  $l \neq 0$ 
  obtains  $d$  where  $0 < d (a + d *_R l) \in \text{frontier } S$ 
     $\wedge e. [0 \leq e; e < d] \implies (a + e *_R l) \in \text{interior } S$ 
proof -
  have  $\S: \text{interior } S = \text{rel\_interior } S$ 
    using a rel_interior_nonempty_interior by auto
  then have  $a \in \text{rel\_interior } S$ 
    using a by simp
  moreover have  $a + l \in \text{affine hull } S$ 
    using a affine_hull_nonempty_interior by blast
  ultimately show thesis
    by (metis  $\S \langle \text{bounded } S \rangle \langle l \neq 0 \rangle \text{frontier\_def ray\_to\_rel\_frontier rel\_frontier\_def}$ 
that)

```

qed

lemma *segment_to_rel_frontier_aux*:

fixes $x :: 'a::\text{euclidean_space}$

assumes *convex* S *bounded* S and $x: x \in \text{rel_interior } S$ and $y: y \in S$ and $xy:$
 $x \neq y$

obtains z where $z \in \text{rel_frontier } S$ $y \in \text{closed_segment } x z$
 $\text{open_segment } x z \subseteq \text{rel_interior } S$

proof –

have $x + (y - x) \in \text{affine hull } S$

using *hull_inc* [OF y] by auto

then obtain d where $0 < d$ and $df: (x + d *_R (y-x)) \in \text{rel_frontier } S$

and $di: \bigwedge e. [0 \leq e; e < d] \implies (x + e *_R (y-x)) \in \text{rel_interior } S$

by (*rule ray_to_rel_frontier* [OF $\langle \text{bounded } S \rangle x$]) (*use xy in auto*)

show ?thesis

proof

show $x + d *_R (y - x) \in \text{rel_frontier } S$

by (*simp add: df*)

next

have $\text{open_segment } x y \subseteq \text{rel_interior } S$

using *rel_interior_closure_convex_segment* [OF $\langle \text{convex } S \rangle x$] *closure_subset*
 y by *blast*

moreover have $x + d *_R (y - x) \in \text{open_segment } x y$ if $d < 1$

using $xy \langle 0 < d \rangle$ that by (*force simp: in_segment algebra_simps*)

ultimately have $1 \leq d$

using *df rel_frontier_def* by *fastforce*

moreover have $x = (1 / d) *_R x + ((d - 1) / d) *_R x$

by (*metis* $\langle 0 < d \rangle$ *add commute add_divide_distrib diff_add_cancel divide_self_if_less_irrefl scaleR_add_left scaleR_one*)

ultimately show $y \in \text{closed_segment } x (x + d *_R (y - x))$

unfolding *in_segment*

by (*rule_tac* $x=1/d$ in *exI*) (*auto simp: algebra_simps*)

next

show $\text{open_segment } x (x + d *_R (y - x)) \subseteq \text{rel_interior } S$

proof (*rule rel_interior_closure_convex_segment* [OF $\langle \text{convex } S \rangle x$])

show $x + d *_R (y - x) \in \text{closure } S$

using *df rel_frontier_def* by auto

qed

qed

qed

lemma *segment_to_rel_frontier*:

fixes $x :: 'a::\text{euclidean_space}$

assumes $S: \text{convex } S$ *bounded* S and $x: x \in \text{rel_interior } S$

and $y: y \in S$ and $xy: \neg(x = y \wedge S = \{x\})$

obtains z where $z \in \text{rel_frontier } S$ $y \in \text{closed_segment } x z$
 $\text{open_segment } x z \subseteq \text{rel_interior } S$

proof (*cases* $x=y$)

```

case True
with xy have S ≠ {x}
  by blast
with True show ?thesis
  by (metis Set.set_insert all_not_in_conv ends_in_segment(1) insert_iff segment_to_rel_frontier_aux[OF S x] that y)
next
case False
then show ?thesis
  using segment_to_rel_frontier_aux [OF S x y] that by blast
qed

```

proposition *rel_frontier_not_sing*:

```

fixes a :: 'a::euclidean_space
assumes bounded S
  shows rel_frontier S ≠ {a}
proof (cases S = {})
case True then show ?thesis by simp
next
case False
then obtain z where z ∈ S
  by blast
then show ?thesis
proof (cases S = {z})
case True then show ?thesis by simp
next
case False
then obtain w where w ∈ S w ≠ z
  using ⟨z ∈ S⟩ by blast
show ?thesis
proof
assume rel_frontier S = {a}
then consider w ∉ rel_frontier S | z ∉ rel_frontier S
  using ⟨w ≠ z⟩ by auto
then show False
proof cases
case 1
then have w: w ∈ rel_interior S
  using ⟨w ∈ S⟩ closure_subset rel_frontier_def by fastforce
have w + (w - z) ∈ affine hull S
  by (metis ⟨w ∈ S⟩ ⟨z ∈ S⟩ affine_affine_hull hull_inc mem_affine_3_minus scaleR_one)
then obtain e where 0 < e (w + e *R (w - z)) ∈ rel_frontier S
  using ⟨w ≠ z⟩ ⟨z ∈ S⟩ by (metis assms ray_to_rel_frontier_right_minus_eq w)
moreover obtain d where 0 < d (w + d *R (z - w)) ∈ rel_frontier S
  using ray_to_rel_frontier [OF ⟨bounded S⟩ w, of 1 *R (z - w)] ⟨w ≠ z⟩
  ⟨z ∈ S⟩
  by (metis add commute add.right_neutral diff_add_cancel hull_inc

```

```

scaleR_one)
  ultimately have  $d *_R (z - w) = e *_R (w - z)$ 
    using  $\langle \text{rel\_frontier } S = \{a\} \rangle$  by force
  moreover have  $e \neq -d$ 
    using  $\langle 0 < e \rangle \langle 0 < d \rangle$  by force
  ultimately show False
    by (metis (no_types, lifting)  $\langle w \neq z \rangle$  eq_iff_diff_eq_0 minus_diff_eq
real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus)
  next
  case 2
  then have  $z: z \in \text{rel\_interior } S$ 
    using  $\langle z \in S \rangle$  closure_subset_rel_frontier_def by fastforce
  have  $z + (z - w) \in \text{affine hull } S$ 
  by (metis  $\langle z \in S \rangle \langle w \in S \rangle$  affine_affine_hull hull_inc mem_affine_3_minus
scaleR_one)
  then obtain  $e$  where  $0 < e (z + e *_R (z - w)) \in \text{rel\_frontier } S$ 
  using  $\langle w \neq z \rangle \langle w \in S \rangle$  by (metis assms ray_to_rel_frontier_right_minus_eq
z)
  moreover obtain  $d$  where  $0 < d (z + d *_R (w - z)) \in \text{rel\_frontier } S$ 
  using ray_to_rel_frontier [OF  $\langle \text{bounded } S \rangle z$ , of  $1 *_R (w - z)$ ]  $\langle w \neq z \rangle$ 
 $\langle w \in S \rangle$ 
  by (metis add.commute add.right_neutral diff_add_cancel hull_inc
scaleR_one)
  ultimately have  $d *_R (w - z) = e *_R (z - w)$ 
    using  $\langle \text{rel\_frontier } S = \{a\} \rangle$  by force
  moreover have  $e \neq -d$ 
    using  $\langle 0 < e \rangle \langle 0 < d \rangle$  by force
  ultimately show False
    by (metis (no_types, lifting)  $\langle w \neq z \rangle$  eq_iff_diff_eq_0 minus_diff_eq
real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus)
  qed
  qed
  qed
  qed

```

6.0.5 Convexity on direct sums

```

lemma closure_sum:
  fixes  $S T :: 'a::\text{real\_normed\_vector\_set}$ 
  shows  $\text{closure } S + \text{closure } T \subseteq \text{closure } (S + T)$ 
  unfolding set_plus_image closure_Times [symmetric] split_def
  by (intro closure_bounded_linear_image_subset bounded_linear_add
bounded_linear_fst bounded_linear_snd)

```

```

lemma fst_snd_linear: linear  $(\lambda(x,y). x + y)$ 
  unfolding linear_iff by (simp add: algebra_simps)

```

```

lemma rel_interior_sum:
  fixes  $S T :: 'n::\text{euclidean\_space\_set}$ 

```



```

assumes convex S
and convex T
shows rel_interior (S + T) = rel_interior S + rel_interior T
proof -
  have rel_interior S + rel_interior T =  $(\lambda(x,y). x + y) \text{ ` } (rel\_interior S \times rel\_interior T)$ 
  by (simp add: set_plus_image)
  also have ... =  $(\lambda(x,y). x + y) \text{ ` } rel\_interior (S \times T)$ 
  using rel_interior_Times assms by auto
  also have ... = rel_interior (S + T)
  using fst_snd_linear convex_Times assms
    rel_interior_convex_linear_image[of  $(\lambda(x,y). x + y) S \times T$ ]
  by (auto simp add: set_plus_image)
  finally show ?thesis ..
qed

```

```

lemma rel_interior_sum_gen:
fixes S :: 'a  $\Rightarrow$  'n::euclidean_space set
assumes  $\bigwedge i. i \in I \implies convex (S i)$ 
shows rel_interior (sum S I) = sum  $(\lambda i. rel\_interior (S i)) I$ 
using rel_interior_sum rel_interior_sing[of 0] assms
by (subst sum_set_cond_linear[of convex], auto simp add: convex_set_plus)

```

```

lemma convex_rel_open_direct_sum:
fixes S T :: 'n::euclidean_space set
assumes convex S
and rel_open S
and convex T
and rel_open T
shows convex (S  $\times$  T)  $\wedge$  rel_open (S  $\times$  T)
by (metis assms convex_Times rel_interior_Times rel_open_def)

```

```

lemma convex_rel_open_sum:
fixes S T :: 'n::euclidean_space set
assumes convex S
and rel_open S
and convex T
and rel_open T
shows convex (S + T)  $\wedge$  rel_open (S + T)
by (metis assms convex_set_plus rel_interior_sum rel_open_def)

```

```

lemma convex_hull_finite_union_cones:
assumes finite I
and  $I \neq \{\}$ 
assumes  $\bigwedge i. i \in I \implies convex (S i) \wedge cone (S i) \wedge S i \neq \{\}$ 
shows convex hull  $(\bigcup (S \text{ ` } I)) = sum S I$ 
  (is ?lhs = ?rhs)
proof -
  {

```

```

fix  $x$ 
assume  $x \in ?lhs$ 
then obtain  $c\ xs$  where
   $x: x = \text{sum } (\lambda i. c\ i *_{\mathbb{R}} xs\ i)\ I \wedge (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. xs$ 
 $i \in S\ i)$ 
  using convex_hull_finite_union[of  $I\ S$ ] assms by auto
define  $s$  where  $s\ i = c\ i *_{\mathbb{R}} xs\ i$  for  $i$ 
have  $\forall i \in I. s\ i \in S\ i$ 
  using s_def  $x$  assms by (simp add: mem_cone)
moreover have  $x = \text{sum } s\ I$  using  $x\ s\_def$  by auto
ultimately have  $x \in ?rhs$ 
  using set_sum_alt[of  $I\ S$ ] assms by auto
}
moreover
{
  fix  $x$ 
assume  $x \in ?rhs$ 
then obtain  $s$  where  $x: x = \text{sum } s\ I \wedge (\forall i \in I. s\ i \in S\ i)$ 
  using set_sum_alt[of  $I\ S$ ] assms by auto
define  $xs$  where  $xs\ i = \text{of\_nat}(\text{card } I) *_{\mathbb{R}} s\ i$  for  $i$ 
then have  $x = \text{sum } (\lambda i. (1 :: \text{real}) / \text{of\_nat}(\text{card } I)) *_{\mathbb{R}} xs\ i)\ I$ 
  using  $x$  assms by auto
moreover have  $\forall i \in I. xs\ i \in S\ i$ 
  using  $xs\_def$  assms by (simp add: cone_def)
moreover have  $\forall i \in I. (1 :: \text{real}) / \text{of\_nat}(\text{card } I) \geq 0$ 
  by auto
moreover have  $\text{sum } (\lambda i. (1 :: \text{real}) / \text{of\_nat}(\text{card } I))\ I = 1$ 
  using assms by auto
ultimately have  $x \in ?lhs$ 
  using assms
  apply (simp add: convex_hull_finite_union[of  $I\ S$ ])
  by (rule_tac  $x = (\lambda i. 1 / (\text{card } I))$  in  $exI$ ) auto
}
ultimately show  $?thesis$  by auto

```

qed

lemma *convex_hull_union_cones_two*:

fixes $S\ T :: 'm::\text{euclidean_space}\ \text{set}$

assumes *convex* S

and *cone* S

and $S \neq \{\}$

assumes *convex* T

and *cone* T

and $T \neq \{\}$

shows *convex hull* $(S \cup T) = S + T$

proof –

define $I :: \text{nat}\ \text{set}$ **where** $I = \{1, 2\}$

define A **where** $A\ i = (\text{if } i = (1::\text{nat}) \text{ then } S \text{ else } T)$ **for** i

have $\bigcup (A\ ` I) = S \cup T$

```

  using A_def I_def by auto
  then have convex_hull (⋃ (A ' I)) = convex_hull (S ∪ T)
    by auto
  moreover have convex_hull ⋃ (A ' I) = sum A I
    using A_def I_def
    by (metis assms convex_hull_finite_union_cones empty_iff finite.emptyI fi-
nite.insertI insertI1)
  moreover have sum A I = S + T
    using A_def I_def by (force simp add: set_plus_def)
  ultimately show ?thesis by auto
qed

```

lemma *rel_interior_convex_hull_union*:

```

  fixes S :: 'a ⇒ 'n::euclidean_space set
  assumes finite I
    and ∀ i ∈ I. convex (S i) ∧ S i ≠ {}
  shows rel_interior (convex_hull (⋃ (S ' I))) =
    {sum (λ i. c i *R s i) I | c s. (∀ i ∈ I. c i > 0) ∧ sum c I = 1 ∧
    (∀ i ∈ I. s i ∈ rel_interior (S i))}
  (is ?lhs = ?rhs)
proof (cases I = {})
  case True
    then show ?thesis
      using convex_hull_empty by auto
  next
  case False
    define C0 where C0 = convex_hull (⋃ (S ' I))
    have ∀ i ∈ I. C0 ≥ S i
      unfolding C0_def using hull_subset[of ⋃ (S ' I)] by auto
    define K0 where K0 = cone_hull ({1 :: real} × C0)
    define K where K i = cone_hull ({1 :: real} × S i) for i
    have ∀ i ∈ I. K i ≠ {}
      unfolding K_def using assms
      by (simp add: cone_hull_empty_iff[symmetric])
    have convK: ∀ i ∈ I. convex (K i)
      unfolding K_def
      by (simp add: assms(2) convex_Times convex_cone_hull)
    have K0 ⊇ K i if i ∈ I for i
      unfolding K0_def K_def
      by (simp add: Sigma_mono ⟨∀ i ∈ I. S i ⊆ C0⟩ hull_mono that)
    then have K0 ⊇ ⋃ (K ' I) by auto
    moreover have convex K0
      unfolding K0_def by (simp add: C0_def convex_Times convex_cone_hull)
    ultimately have geq: K0 ⊇ convex_hull (⋃ (K ' I))
      using hull_minimal[of _ K0 convex] by blast
    have ∀ i ∈ I. K i ⊇ {1 :: real} × S i
      using K_def by (simp add: hull_subset)
    then have ⋃ (K ' I) ⊇ {1 :: real} × ⋃ (S ' I)
      by auto

```

```

then have convex_hull  $\bigcup (K \text{ ' } I) \supseteq \text{convex\_hull } (\{1 :: \text{real}\} \times \bigcup (S \text{ ' } I))$ 
  by (simp add: hull_mono)
then have convex_hull  $\bigcup (K \text{ ' } I) \supseteq \{1 :: \text{real}\} \times C0$ 
  unfolding C0_def
  using convex_hull_Times[of  $\{1 :: \text{real}\} \bigcup (S \text{ ' } I)$ ] convex_hull_singleton
  by auto
moreover have cone (convex_hull  $\bigcup (K \text{ ' } I)$ )
  by (simp add: K_def cone_Union cone_cone_hull cone_convex_hull)
ultimately have convex_hull  $\bigcup (K \text{ ' } I) \supseteq K0$ 
  unfolding K0_def
  using hull_minimal[of  $\_ \text{convex\_hull } \bigcup (K \text{ ' } I)$ ] cone]
  by blast
then have  $K0 = \text{convex\_hull } \bigcup (K \text{ ' } I)$ 
  using geq by auto
also have  $\dots = \text{sum } K \ I$ 
  using assms False  $\langle \forall i \in I. K \ i \neq \{\} \rangle$  cone_hull_eq convK
  by (intro convex_hull_finite_union_cones; fastforce simp: K_def)
finally have  $K0 = \text{sum } K \ I$  by auto
then have *:  $\text{rel\_interior } K0 = \text{sum } (\lambda i. (\text{rel\_interior } (K \ i))) \ I$ 
  using rel_interior_sum_gen[of  $I \ K$ ] convK by auto
{
  fix  $x$ 
  assume  $x \in ?lhs$ 
  then have  $(1 :: \text{real}, x) \in \text{rel\_interior } K0$ 
    using K0_def C0_def rel_interior_convex_cone_aux[of  $C0 \ 1 :: \text{real} \ x$ ] convex_convex_hull
    by auto
  then obtain  $k$  where  $k: (1 :: \text{real}, x) = \text{sum } k \ I \wedge (\forall i \in I. k \ i \in \text{rel\_interior } (K \ i))$ 
    using  $\langle \text{finite } I \rangle$  * set_sum_alt[of  $I \ \lambda i. \text{rel\_interior } (K \ i)$ ] by auto
  {
    fix  $i$ 
    assume  $i \in I$ 
    then have  $\text{convex } (S \ i) \wedge k \ i \in \text{rel\_interior } (\text{cone\_hull } \{1\} \times S \ i)$ 
      using  $k \ K\_def$  assms by auto
    then have  $\exists ci \ si. k \ i = (ci, ci *_R \ si) \wedge 0 < ci \wedge si \in \text{rel\_interior } (S \ i)$ 
      using rel_interior_convex_cone[of  $S \ i$ ] by auto
  }
  then obtain  $c \ s$  where  $cs: \forall i \in I. k \ i = (c \ i, c \ i *_R \ s \ i) \wedge 0 < c \ i \wedge s \ i \in \text{rel\_interior } (S \ i)$ 
    by metis
  then have  $x = (\sum i \in I. c \ i *_R \ s \ i) \wedge \text{sum } c \ I = 1$ 
    using  $k$  by (simp add: sum_prod)
  then have  $x \in ?rhs$ 
    using  $k \ cs$  by auto
}
moreover
{
  fix  $x$ 

```

```

assume  $x \in ?rhs$ 
then obtain  $c\ s$  where  $cs: x = \text{sum } (\lambda i. c\ i *_R\ s\ i)\ I \wedge$ 
   $(\forall i \in I. c\ i > 0) \wedge \text{sum } c\ I = 1 \wedge (\forall i \in I. s\ i \in \text{rel\_interior } (S\ i))$ 
by auto
define  $k$  where  $k\ i = (c\ i, c\ i *_R\ s\ i)$  for  $i$ 
{
  fix  $i$  assume  $i \in I$ 
  then have  $k\ i \in \text{rel\_interior } (K\ i)$ 
    using  $k\_def\ K\_def\ assms\ cs\ \text{rel\_interior\_convex\_cone}$ [of  $S\ i$ ]
    by auto
}
then have  $(1, x) \in \text{rel\_interior } K0$ 
  using  $*\ \text{set\_sum\_alt}$ [of  $I\ (\lambda i. \text{rel\_interior } (K\ i))$ ] assms  $cs$ 
  by ( $\text{simp add: } k\_def$ ) ( $\text{metis (mono\_tags, lifting) sum\_prod}$ )
then have  $x \in ?lhs$ 
  using  $K0\_def\ C0\_def\ \text{rel\_interior\_convex\_cone\_aux}$ [of  $C0\ 1\ x$ ]
  by auto
}
ultimately show  $?thesis$  by blast
qed

```

lemma $\text{convex_le_Inf_differential}$:

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $\text{convex\_on } I\ f$ 
  and  $x \in \text{interior } I$ 
  and  $y \in I$ 
shows  $f\ y \geq f\ x + \text{Inf } ((\lambda t. (f\ x - f\ t) / (x - t)) \text{ ` } (\{x <.. \} \cap I)) * (y - x)$ 
  (is  $\_ \geq \_ + \text{Inf } (?F\ x) * (y - x)$ )
proof ( $\text{cases rule: linorder\_cases}$ )
  assume  $x < y$ 
  moreover
  have  $\text{open } (\text{interior } I)$  by auto
  from  $\text{openE}$ [OF  $\text{this } \langle x \in \text{interior } I \rangle$ ]
  obtain  $e$  where  $0 < e$   $\text{ball } x\ e \subseteq \text{interior } I$  .
  moreover define  $t$  where  $t = \min (x + e / 2) ((x + y) / 2)$ 
  ultimately have  $x < t < y$   $t \in \text{ball } x\ e$ 
    by ( $\text{auto simp: dist\_real\_def field\_simps split: split\_min}$ )
  with  $\langle x \in \text{interior } I \rangle\ e\ \text{interior\_subset}$ [of  $I$ ] have  $t \in I$   $x \in I$  by auto

  define  $K$  where  $K = x - e / 2$ 
  with  $\langle 0 < e \rangle$  have  $K \in \text{ball } x\ e$   $K < x$ 
    by ( $\text{auto simp: dist\_real\_def}$ )
  then have  $K \in I$ 
    using  $\langle \text{interior } I \subseteq I \rangle\ e(2)$  by blast

  have  $\text{Inf } (?F\ x) \leq (f\ x - f\ y) / (x - y)$ 
  proof ( $\text{intro bdd\_belowI cInf\_lower2}$ )
    show  $(f\ x - f\ t) / (x - t) \in ?F\ x$ 

```

```

    using ⟨t ∈ I⟩ ⟨x < t⟩ by auto
    show (f x - f t) / (x - t) ≤ (f x - f y) / (x - y)
      using ⟨convex_on I f⟩ ⟨x ∈ I⟩ ⟨y ∈ I⟩ ⟨x < t⟩ ⟨t < y⟩
      by (rule convex_on_slope_le)
  next
    fix y
    assume y ∈ ?F x
    with order_trans[OF convex_on_slope_le[OF ⟨convex_on I f⟩ ⟨K ∈ I⟩ _ ⟨K
< x⟩ _]]
    show (f K - f x) / (K - x) ≤ y by auto
  qed
  then show ?thesis
    using ⟨x < y⟩ by (simp add: field_simps)
next
  assume y < x
  moreover
  have open (interior I) by auto
  from openE[OF this ⟨x ∈ interior I⟩]
  obtain e where e: 0 < e ball x e ⊆ interior I .
  moreover define t where t = x + e / 2
  ultimately have x < t t ∈ ball x e
    by (auto simp: dist_real_def field_simps)
  with ⟨x ∈ interior I⟩ e interior_subset[of I] have t ∈ I x ∈ I by auto

  have (f x - f y) / (x - y) ≤ Inf (?F x)
  proof (rule cInf_greatest)
    have (f x - f y) / (x - y) = (f y - f x) / (y - x)
      using ⟨y < x⟩ by (auto simp: field_simps)
    also
    fix z
    assume z ∈ ?F x
    with order_trans[OF convex_on_slope_le[OF ⟨convex_on I f⟩ ⟨y ∈ I⟩ _ ⟨y
< x⟩]]
    have (f y - f x) / (y - x) ≤ z
      by auto
    finally show (f x - f y) / (x - y) ≤ z .
  next
    have x + e / 2 ∈ ball x e
      using e by (auto simp: dist_real_def)
    with e interior_subset[of I] have x + e / 2 ∈ {x<..} ∩ I
      by auto
    then show ?F x ≠ {}
      by blast
  qed
  then show ?thesis
    using ⟨y < x⟩ by (simp add: field_simps)
qed simp

```

6.0.6 Explicit formulas for interior and relative interior of convex hull

lemma *at_within_cbox_finite*:

assumes $x \in \text{cbox } a \ b \ x \notin S$ *finite S*
 shows $(\text{at } x \text{ within } \text{cbox } a \ b - S) = \text{at } x$

proof –

have $\text{interior } (\text{cbox } a \ b - S) = \text{cbox } a \ b - S$
 using $\langle \text{finite } S \rangle$ by (simp add: *interior_diff_finite_imp_closed*)

then show *?thesis*

using *at_within_interior_assms* by fastforce

qed

lemma *affine_independent_convex_affine_hull*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes $\neg \text{affine_dependent } S \ T \subseteq S$

shows $\text{convex hull } T = \text{affine hull } T \cap \text{convex hull } S$

proof –

have *fin*: *finite S finite T* using *assms affine_independent_finite_finite_subset* by auto

have $\text{convex hull } T \subseteq \text{affine hull } T$

using *convex_hull_subset_affine_hull* by blast

moreover have $\text{convex hull } T \subseteq \text{convex hull } S$

using *assms hull_mono* by blast

moreover have $\text{affine hull } T \cap \text{convex hull } S \subseteq \text{convex hull } T$

proof –

have $0: \bigwedge u. \text{sum } u \ S = 0 \implies (\forall v \in S. u \ v = 0) \vee (\sum v \in S. u \ v \ *_R \ v) \neq 0$

using *affine_dependent_explicit_finite_assms(1) fin(1)* by auto

show *?thesis*

proof (clarsimp simp add: *affine_hull_finite fin*)

fix u

assume $S: (\sum v \in T. u \ v \ *_R \ v) \in \text{convex hull } S$

and $T1: \text{sum } u \ T = 1$

then obtain v where $v: \forall x \in S. 0 \leq v \ x \ \text{sum } v \ S = 1 \ (\sum x \in S. v \ x \ *_R \ x) =$
 $(\sum v \in T. u \ v \ *_R \ v)$

by (auto simp add: *convex_hull_finite fin*)

{ fix x

assume $x \in T$

then have $S: S = (S - T) \cup T$ — split into separate cases

using *assms* by auto

have [simp]: $(\sum x \in T. v \ x \ *_R \ x) + (\sum x \in S - T. v \ x \ *_R \ x) = (\sum x \in T. u$
 $x \ *_R \ x)$

$\text{sum } v \ T + \text{sum } v \ (S - T) = 1$

using *v fin S*

by (auto simp: *sum.union_disjoint [symmetric] Un_commute*)

have $(\sum x \in S. \text{if } x \in T \text{ then } v \ x - u \ x \ \text{else } v \ x) = 0$

$(\sum x \in S. (\text{if } x \in T \text{ then } v \ x - u \ x \ \text{else } v \ x) \ *_R \ x) = 0$

using *v fin T1*

by (subst *S*, subst *sum.union_disjoint*, auto simp: *algebra_simps sum_subtractf*)+

} note [simp] = *this*

```

have ( $\forall x \in T. 0 \leq u x$ )
  using 0 [of  $\lambda x. \text{if } x \in T \text{ then } v x - u x \text{ else } v x$ ]  $\langle T \subseteq S \rangle v(1)$  by fastforce
then show ( $\sum v \in T. u v *_{\mathbb{R}} v$ )  $\in \text{convex hull } T$ 
  using 0 [of  $\lambda x. \text{if } x \in T \text{ then } v x - u x \text{ else } v x$ ]  $\langle T \subseteq S \rangle T1$ 
  by (fastforce simp add: convex_hull_finite fin)
qed
qed
ultimately show ?thesis
  by blast
qed

```

```

lemma affine_independent_span_eq:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\neg \text{affine\_dependent } S$   $\text{card } S = \text{Suc } (\text{DIM } ('a))$ 
  shows  $\text{affine hull } S = \text{UNIV}$ 
proof (cases  $S = \{\}$ )
  case True then show ?thesis
    using assms by simp
  next
  case False
    then obtain  $a T$  where  $T: a \notin T$   $S = \text{insert } a T$ 
      by blast
    then have  $\text{fin}: \text{finite } T$  using assms
      by (metis finite_insert affine_independent_finite)
    have  $\text{UNIV} \subseteq (+) a \text{ 'span } ((\lambda x. x - a) \text{ ' } T)$ 
    proof (intro card_ge_dim_independent Fun.vimage_subsetD)
      show  $\text{independent } ((\lambda x. x - a) \text{ ' } T)$ 
        using  $T$  affine_dependent_iff_dependent assms(1) by auto
      show  $\text{dim } ((+) a \text{ ' UNIV}) \leq \text{card } ((\lambda x. x - a) \text{ ' } T)$ 
        using assms  $T$  fin by (auto simp: card_image inj_on_def)
    qed (use surj_plus in auto)
    then show ?thesis
      using  $T(2)$  affine_hull_insert_span_gen equalityI by fastforce
qed

```

```

lemma affine_independent_span_gt:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\text{ind}: \neg \text{affine\_dependent } S$  and  $\text{dim}: \text{DIM } ('a) < \text{card } S$ 
  shows  $\text{affine hull } S = \text{UNIV}$ 
proof (intro affine_independent_span_eq [OF ind] antisym)
  show  $\text{card } S \leq \text{Suc } \text{DIM } ('a)$ 
    using affine_independent_finite affine_dependent_biggerset ind by fastforce
  show  $\text{Suc } \text{DIM } ('a) \leq \text{card } S$ 
    using Suc_leI dim by blast
qed

```

```

lemma empty_interior_affine_hull:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\text{finite } S$  and  $\text{dim}: \text{card } S \leq \text{DIM } ('a)$ 

```



```

  shows interior(affine hull S) = {}
  using assms
proof (induct S rule: finite_induct)
  case (insert x S)
  then have dim (span ((λy. y - x) ' S)) < DIM('a)
  by (auto simp: Suc_le_lessD card_image_le dual_order.trans intro!: dim_le_card'[THEN
le_less_trans])
  then show ?case
  by (simp add: empty_interior_lowdim affine_hull_insert_span_gen interior_translation)
qed auto

```

```

lemma empty_interior_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes finite S and dim: card S ≤ DIM ('a)
  shows interior(convex hull S) = {}
  by (metis Diff_empty Diff_eq_empty_iff convex_hull_subset_affine_hull
interior_mono empty_interior_affine_hull [OF assms])

```

```

lemma explicit_subset_rel_interior_convex_hull:
  fixes S :: 'a::euclidean_space set
  shows finite S
    ⇒ {y. ∃ u. (∀ x ∈ S. 0 < u x ∧ u x < 1) ∧ sum u S = 1 ∧ sum (λx. u x
*_R x) S = y}
    ⊆ rel_interior (convex hull S)
  by (force simp add: rel_interior_convex_hull_union [where S=λx. {x} and
I=S, simplified])

```

```

lemma explicit_subset_rel_interior_convex_hull_minimal:
  fixes S :: 'a::euclidean_space set
  shows finite S
    ⇒ {y. ∃ u. (∀ x ∈ S. 0 < u x) ∧ sum u S = 1 ∧ sum (λx. u x *_R x) S =
y}
    ⊆ rel_interior (convex hull S)
  by (force simp add: rel_interior_convex_hull_union [where S=λx. {x} and
I=S, simplified])

```

```

lemma rel_interior_convex_hull_explicit:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows rel_interior(convex hull S) =
    {y. ∃ u. (∀ x ∈ S. 0 < u x) ∧ sum u S = 1 ∧ sum (λx. u x *_R x) S = y}
    (is ?lhs = ?rhs)

```

```

proof
  show ?rhs ≤ ?lhs
  by (simp add: aff_independent_finite explicit_subset_rel_interior_convex_hull_minimal
assms)
next
  show ?lhs ≤ ?rhs
  proof (cases ∃ a. S = {a})

```

```

case True then show  $?lhs \leq ?rhs$ 
  by force
next
case False
have fs: finite S
  using assms by (simp add: aff_independent_finite)
  { fix a b and d::real
    assume ab:  $a \in S \ b \in S \ a \neq b$ 
    then have S:  $S = (S - \{a,b\}) \cup \{a,b\}$  — split into separate cases
      by auto
      have  $(\sum_{x \in S}. \text{if } x = a \text{ then } -d \text{ else if } x = b \text{ then } d \text{ else } 0) = 0$ 
         $(\sum_{x \in S}. (\text{if } x = a \text{ then } -d \text{ else if } x = b \text{ then } d \text{ else } 0) *_R x) = d *_R b$ 
      -  $d *_R a$ 
      using ab fs
      by (subst S, subst sum.union_disjoint, auto)+
    } note [simp] = this
  { fix y
    assume y:  $y \in \text{convex hull } S \ y \notin ?rhs$ 
    have *: False if
      ua:  $\forall x \in S. 0 \leq u \ x \ \text{sum } u \ S = 1 \ \neg \ 0 < u \ a \ a \in S$ 
      and yT:  $y = (\sum_{x \in S}. u \ x *_R x) \ y \in T \ \text{open } T$ 
      and sb:  $T \cap \text{affine hull } S \subseteq \{w. \exists u. (\forall x \in S. 0 \leq u \ x) \wedge \text{sum } u \ S = 1 \wedge$ 
       $(\sum_{x \in S}. u \ x *_R x) = w\}$ 
      for u T a
      proof -
        have ua0:  $u \ a = 0$ 
          using ua by auto
        obtain b where b:  $b \in S \ a \neq b$ 
          using ua False by auto
        obtain e where e:  $0 < e \ \text{ball } (\sum_{x \in S}. u \ x *_R x) \ e \subseteq T$ 
          using yT by (auto elim: openE)
        with b obtain d where d:  $0 < d \ \text{norm}(d *_R (a-b)) < e$ 
          by (auto intro: that [of e / 2 / norm(a-b)])
        have  $(\sum_{x \in S}. u \ x *_R x) \in \text{affine hull } S$ 
          using yT y by (metis affine_hull_convex_hull hull_redundant_eq)
        then have  $(\sum_{x \in S}. u \ x *_R x) - d *_R (a - b) \in \text{affine hull } S$ 
          using ua b by (auto simp: hull_inc intro: mem_affine_3_minus2)
        then have  $y - d *_R (a - b) \in T \cap \text{affine hull } S$ 
          using d e yT by auto
        then obtain v where v:  $\forall x \in S. 0 \leq v \ x$ 
           $\text{sum } v \ S = 1$ 
           $(\sum_{x \in S}. v \ x *_R x) = (\sum_{x \in S}. u \ x *_R x) - d *_R (a - b)$ 
          using subsetD [OF sb] yT
          by auto
        have aff:  $\bigwedge u. \text{sum } u \ S = 0 \implies (\forall v \in S. u \ v = 0) \vee (\sum_{v \in S}. u \ v *_R v) \neq 0$ 
          using assms by (simp add: affine_dependent_explicit_finite fs)
        show False
          using ua b d v aff [of  $\lambda x. (v \ x - u \ x) - (\text{if } x = a \text{ then } -d \text{ else if } x = b$ 
           $\text{then } d \text{ else } 0)$ ]

```

```

      by (auto simp: algebra_simps sum_subtractf sum.distrib)
    qed
  have  $y \notin \text{rel\_interior}(\text{convex hull } S)$ 
    using  $y \text{ convex\_hull\_finite [OF fs]}$  *
    apply simp
  by (metis (no_types, lifting) IntD1 affine_hull_convex_hull mem_rel_interior)
} with rel_interior_subset show ?lhs  $\leq$  ?rhs
  by blast
qed
qed

```

lemma *interior_convex_hull_explicit_minimal*:

```

  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\neg \text{affine\_dependent } S$ 
  shows
    interior(convex hull  $S$ ) =
      (if  $\text{card}(S) \leq \text{DIM}('a)$  then {}
       else  $\{y. \exists u. (\forall x \in S. 0 < u x) \wedge \text{sum } u S = 1 \wedge (\sum_{x \in S} u x *_R x) = y\}$ )
    (is _ = (if _ then _ else ?rhs))
  proof -
    { assume  $S: \neg \text{card } S \leq \text{DIM}('a)$ 
      have interior (convex hull  $S$ ) = rel_interior(convex hull  $S$ )
        using assms  $S$  by (simp add: affine_independent_span_gt_rel_interior_interior)
      then have interior(convex hull  $S$ ) = ?rhs
        by (simp add: assms  $S$  rel_interior_convex_hull_explicit)
    }
  then show ?thesis
    by (auto simp: aff_independent_finite_empty_interior_convex_hull assms)
  qed

```

lemma *interior_convex_hull_explicit*:

```

  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\neg \text{affine\_dependent } S$ 
  shows
    interior(convex hull  $S$ ) =
      (if  $\text{card}(S) \leq \text{DIM}('a)$  then {}
       else  $\{y. \exists u. (\forall x \in S. 0 < u x \wedge u x < 1) \wedge \text{sum } u S = 1 \wedge (\sum_{x \in S} u x *_R x) = y\}$ )
  proof -
    { fix  $u :: 'a \Rightarrow \text{real}$  and  $a$ 
      assume  $\text{card } \text{Basis} < \text{card } S$  and  $u: \bigwedge x. x \in S \implies 0 < u x$  and  $\text{sum } u S = 1$  and
       $a: a \in S$ 
      then have  $cs: \text{Suc } 0 < \text{card } S$ 
        by (metis DIM_positive less_trans_Suc)
      obtain  $b$  where  $b: b \in S$  and  $a \neq b$ 
      proof (cases  $S \leq \{a\}$ )
        case True
          then show thesis

```

```

    using cs_subset_singletonD by fastforce
qed blast
have u a + u b ≤ sum u {a,b}
  using a b by simp
also have ... ≤ sum u S
  using a b u
by (intro Groups_Big.sum_mono2) (auto simp: less_imp_le aff_independent_finite
assms)
  finally have u a < 1
    using ⟨b ∈ S⟩ u by fastforce
} note [simp] = this
show ?thesis
  using assms by (force simp add: not_le interior_convex_hull_explicit_minimal)
qed

```

```

lemma interior_closed_segment_ge2:
  fixes a :: 'a::euclidean_space
  assumes 2 ≤ DIM('a)
  shows interior(closed_segment a b) = {}
using assms unfolding segment_convex_hull
proof -
  have card {a, b} ≤ DIM('a)
  using assms
  by (simp add: card_insert_if_linear not_less_eq_eq numeral_2_eq_2)
  then show interior (convex_hull {a, b}) = {}
  by (metis empty_interior_convex_hull finite.insertI finite.emptyI)
qed

```

```

lemma interior_open_segment:
  fixes a :: 'a::euclidean_space
  shows interior(open_segment a b) =
    (if 2 ≤ DIM('a) then {} else open_segment a b)
proof (cases 2 ≤ DIM('a))
  case True
  then have interior (open_segment a b) = {}
  using interior_closed_segment_ge2 interior_mono_segment_open_subset_closed
  by blast
  with True show ?thesis
  by auto
next
  case ge2: False
  have interior (open_segment a b) = open_segment a b
  proof (cases a = b)
    case True then show ?thesis by auto
  next
    case False
    with ge2 have affine_hull (open_segment a b) = UNIV
    by (simp add: False affine_independent_span_gt)
    then show interior (open_segment a b) = open_segment a b
  next

```

```

    using rel_interior_interior rel_interior_open_segment by blast
  qed
  with ge2 show ?thesis
  by auto
qed

lemma interior_closed_segment:
  fixes a :: 'a::euclidean_space
  shows interior(closed_segment a b) =
    (if 2 ≤ DIM('a) then {} else open_segment a b)
proof (cases a = b)
  case True then show ?thesis by simp
next
  case False
  then have closure (open_segment a b) = closed_segment a b
  by simp
  then show ?thesis
  by (metis (no_types) convex_interior_closure convex_open_segment interior_open_segment)
qed

lemmas interior_segment = interior_closed_segment interior_open_segment

lemma closed_segment_eq [simp]:
  fixes a :: 'a::euclidean_space
  shows closed_segment a b = closed_segment c d ↔ {a,b} = {c,d}
proof
  assume abcd: closed_segment a b = closed_segment c d
  show {a,b} = {c,d}
  proof (cases a=b ∨ c=d)
    case True with abcd show ?thesis by force
  next
    case False
    then have neq: a ≠ b ∧ c ≠ d by force
    have *: closed_segment c d - {a, b} = rel_interior (closed_segment c d)
    using neq abcd by (metis (no_types) open_segment_def rel_interior_closed_segment)
    have b ∈ {c, d}
    proof -
      have insert b (closed_segment c d) = closed_segment c d
      using abcd by blast
      then show ?thesis
      by (metis DiffD2 Diff_insert2 False * insertI1 insert_Diff_if open_segment_def rel_interior_closed_segment)
    qed
    moreover have a ∈ {c, d}
    by (metis Diff_iff False * abcd ends_in_segment(1) insertI1 open_segment_def rel_interior_closed_segment)
    ultimately show {a, b} = {c, d}
    using neq by fastforce
  qed

```

```

qed
next
  assume  $\{a,b\} = \{c,d\}$ 
  then show  $\text{closed\_segment } a \ b = \text{closed\_segment } c \ d$ 
    by (simp add: segment_convex_hull)
qed

```

```

lemma closed_open_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{closed\_segment } a \ b \neq \text{open\_segment } c \ d$ 
by (metis DiffE closed_segment_neq_empty closure_closed_segment closure_open_segment ends_in_segment(1) insertI1 open_segment_def)

```

```

lemma open_closed_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{open\_segment } a \ b \neq \text{closed\_segment } c \ d$ 
using closed_open_segment_eq by blast

```

```

lemma open_segment_eq [simp]:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  shows  $\text{open\_segment } a \ b = \text{open\_segment } c \ d \iff a = b \wedge c = d \vee \{a,b\} = \{c,d\}$ 
  (is  $?lhs = ?rhs$ )

```

```

proof
  assume  $abcd: ?lhs$ 
  show  $?rhs$ 
  proof (cases  $a=b \vee c=d$ )
    case True with  $abcd$  show  $?thesis$ 
      using finite_open_segment by fastforce
    next
    case False
      then have  $a2: a \neq b \wedge c \neq d$  by force
      with  $abcd$  show  $?rhs$ 
        unfolding open_segment_def
        by (metis (no_types)  $abcd$  closed_segment_eq closure_open_segment)
  qed
next
  assume  $?rhs$ 
  then show  $?lhs$ 
    by (metis Diff_cancel convex_hull_singleton insert_absorb2 open_segment_def segment_convex_hull)
qed

```

6.0.7 Similar results for closure and (relative or absolute) frontier

```

lemma closure_convex_hull [simp]:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  shows  $\text{compact } S \implies \text{closure}(\text{convex hull } S) = \text{convex hull } S$ 

```

by (simp add: compact_imp_closed compact_convex_hull)

lemma *rel_frontier_convex_hull_explicit*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes \neg *affine_dependent* S

shows $\text{rel_frontier}(\text{convex_hull } S) =$

$\{y. \exists u. (\forall x \in S. 0 \leq u x) \wedge (\exists x \in S. u x = 0) \wedge \text{sum } u S = 1 \wedge \text{sum}$
 $(\lambda x. u x *_{\mathbb{R}} x) S = y\}$

proof –

have $fs: \text{finite } S$

using *assms* **by** (simp add: *aff_independent_finite*)

have $\bigwedge u y v.$

$\llbracket y \in S; u y = 0; \text{sum } u S = 1; \forall x \in S. 0 < v x;$

$\text{sum } v S = 1; (\sum_{x \in S} v x *_{\mathbb{R}} x) = (\sum_{x \in S} u x *_{\mathbb{R}} x) \rrbracket$

$\implies \exists u. \text{sum } u S = 0 \wedge (\exists v \in S. u v \neq 0) \wedge (\sum_{v \in S} u v *_{\mathbb{R}} v) = 0$

apply (*rule_tac* $x = \lambda x. u x - v x$ **in** *exI*)

apply (*force simp: sum_subtractf scaleR_diff_left*)

done

then show *?thesis*

using fs *assms*

apply (simp add: *rel_frontier_def finite_imp_compact rel_interior_convex_hull_explicit*)

apply (*auto simp: convex_hull_finite*)

apply (*metis less_eq_real_def*)

by (simp add: *affine_dependent_explicit_finite*)

qed

lemma *frontier_convex_hull_explicit*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes \neg *affine_dependent* S

shows $\text{frontier}(\text{convex_hull } S) =$

$\{y. \exists u. (\forall x \in S. 0 \leq u x) \wedge (\text{DIM } ('a) < \text{card } S \longrightarrow (\exists x \in S. u x = 0))$
 \wedge
 $\text{sum } u S = 1 \wedge \text{sum } (\lambda x. u x *_{\mathbb{R}} x) S = y\}$

proof –

have $fs: \text{finite } S$

using *assms* **by** (simp add: *aff_independent_finite*)

show *?thesis*

proof (*cases* $\text{DIM } ('a) < \text{card } S$)

case *True*

with *assms fs* **show** *?thesis*

by (simp add: *rel_frontier_def frontier_def rel_frontier_convex_hull_explicit*
 $[\text{symmetric}]$

interior_convex_hull_explicit_minimal rel_interior_convex_hull_explicit)

next

case *False*

then have $\text{card } S \leq \text{DIM } ('a)$

by *linarith*

then show *?thesis*

using *assms fs*

```

apply (simp add: frontier_def interior_convex_hull_explicit finite_imp_compact)
apply (simp add: convex_hull_finite)
done
qed
qed

```

lemma *rel_frontier_convex_hull_cases*:

```

fixes  $S :: 'a::euclidean\_space$  set
assumes  $\neg$  affine_dependent  $S$ 
shows  $rel\_frontier(convex\ hull\ S) = \bigcup \{convex\ hull\ (S - \{x\}) \mid x. x \in S\}$ 
proof -
have  $fs$ : finite  $S$ 
using assms by (simp add: aff_independent_finite)
{ fix  $u\ a$ 
have  $\forall x \in S. 0 \leq u\ x \implies a \in S \implies u\ a = 0 \implies sum\ u\ S = 1 \implies$ 
 $\exists x\ v. x \in S \wedge$ 
 $(\forall x \in S - \{x\}. 0 \leq v\ x) \wedge$ 
 $sum\ v\ (S - \{x\}) = 1 \wedge (\sum x \in S - \{x\}. v\ x *_{\mathbb{R}} x) = (\sum x \in S. u$ 
 $x *_{\mathbb{R}} x)$ 
apply (rule_tac  $x=a$  in exI)
apply (rule_tac  $x=u$  in exI)
apply (simp add: Groups_Big.sum_diff1  $fs$ )
done }
moreover
{ fix  $a\ u$ 
have  $a \in S \implies \forall x \in S - \{a\}. 0 \leq u\ x \implies sum\ u\ (S - \{a\}) = 1 \implies$ 
 $\exists v. (\forall x \in S. 0 \leq v\ x) \wedge$ 
 $(\exists x \in S. v\ x = 0) \wedge sum\ v\ S = 1 \wedge (\sum x \in S. v\ x *_{\mathbb{R}} x) = (\sum x \in S -$ 
 $\{a\}. u\ x *_{\mathbb{R}} x)$ 
apply (rule_tac  $x=\lambda x. if\ x = a\ then\ 0\ else\ u\ x$  in exI)
apply (auto simp: sum.If_cases Diff_eq if_smult  $fs$ )
done }
ultimately show ?thesis
using assms
apply (simp add: rel_frontier_convex_hull_explicit)
apply (auto simp add: convex_hull_finite  $fs$  Union_SetCompr_eq)
done
qed

```

lemma *frontier_convex_hull_eq_rel_frontier*:

```

fixes  $S :: 'a::euclidean\_space$  set
assumes  $\neg$  affine_dependent  $S$ 
shows  $frontier(convex\ hull\ S) =$ 
 $(if\ card\ S \leq DIM\ ('a)\ then\ convex\ hull\ S\ else\ rel\_frontier(convex\ hull\ S))$ 
using assms
unfolding rel_frontier_def frontier_def
by (simp add: affine_independent_span_gt rel_interior_interior
finite_imp_compact empty_interior_convex_hull aff_independent_finite)

```



```

lemma frontier_convex_hull_cases:
  fixes S :: 'a::euclidean_space set
  assumes  $\neg$  affine_dependent S
  shows frontier(convex hull S) =
    (if card S  $\leq$  DIM ('a) then convex hull S else  $\bigcup$ {convex hull (S - {x})
|x. x  $\in$  S})
by (simp add: assms frontier_convex_hull_eq_rel_frontier rel_frontier_convex_hull_cases)

```

```

lemma in_frontier_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes finite S card S  $\leq$  Suc (DIM ('a)) x  $\in$  S
  shows x  $\in$  frontier(convex hull S)
proof (cases affine_dependent S)
  case True
  with assms obtain y where y  $\in$  S and y: y  $\in$  affine hull (S - {y})
  by (auto simp: affine_dependent_def)
  moreover have x  $\in$  closure (convex hull S)
  by (meson closure_subset hull_inc subset_eq  $\langle$ x  $\in$  S $\rangle$ )
  moreover have x  $\notin$  interior (convex hull S)
  using assms
  by (metis Suc_mono affine_hull_convex_hull affine_hull_nonempty_interior
 $\langle$ y  $\in$  S $\rangle$  y card.remove empty_iff empty_interior_affine_hull finite_Diff hull_redundant
insert_Diff interior_UNIV not_less)
  ultimately show ?thesis
  unfolding frontier_def by blast
  next
  case False
  { assume card S = Suc (card Basis)
  then have cs: Suc 0 < card S
  by (simp)
  with subset_singletonD have  $\exists$  y  $\in$  S. y  $\neq$  x
  by (cases S  $\leq$  {x}) fastforce+
  } note [dest!] = this
  show ?thesis using assms
  unfolding frontier_convex_hull_cases [OF False] Union_SetCompr_eq
  by (auto simp: le_Suc_eq hull_inc)
qed

```

```

lemma not_in_interior_convex_hull:
  fixes S :: 'a::euclidean_space set
  assumes finite S card S  $\leq$  Suc (DIM ('a)) x  $\in$  S
  shows x  $\notin$  interior(convex hull S)
using in_frontier_convex_hull [OF assms]
by (metis Diff_iff frontier_def)

```

```

lemma interior_convex_hull_eq_empty:
  fixes S :: 'a::euclidean_space set
  assumes card S = Suc (DIM ('a))
  shows interior(convex hull S) = {}  $\longleftrightarrow$  affine_dependent S

```

```

proof
  show affine_dependent S  $\implies$  interior (convex hull S) = {}
  proof (clarsimp simp: affine_dependent_def)
    fix a b
    assume b  $\in$  S b  $\in$  affine hull (S - {b})
    then have interior(affine hull S) = {} using assms
    by (metis DIM_positive One_nat_def Suc_mono card.remove card.infinite
    empty_interior_affine_hull eq_iff_hull_redundant insert_Diff not_less zero_le_one)
    then show interior (convex hull S) = {}
    using affine_hull_nonempty_interior by fastforce
  qed
next
  show interior (convex hull S) = {}  $\implies$  affine_dependent S
  by (metis affine_hull_convex_hull affine_hull_empty affine_independent_span_eq
  assms convex_convex_hull empty_not_UNIV rel_interior_eq_empty rel_interior_interior)
  qed

```

6.0.8 Coplanarity, and collinearity in terms of affine hull

definition *coplanar* **where**

$$\text{coplanar } S \equiv \exists u v w. S \subseteq \text{affine hull } \{u, v, w\}$$

lemma *collinear_affine_hull*:

$$\text{collinear } S \longleftrightarrow (\exists u v. S \subseteq \text{affine hull } \{u, v\})$$

proof (cases S={})

case True **then show** ?thesis

by simp

next

case False

then obtain x **where** x: x \in S **by** auto

{ **fix** u

assume *: $\bigwedge x y. [x \in S; y \in S] \implies \exists c. x - y = c *_R u$

have $\bigwedge y c. x - y = c *_R u \implies \exists a b. y = a *_R x + b *_R (x + u) \wedge a + b = 1$

by (rule_tac x=1+c **in** exI, rule_tac x=-c **in** exI, simp add: algebra_simps)

then have $\exists u v. S \subseteq \{a *_R u + b *_R v \mid a b. a + b = 1\}$

using * [OF x] **by** (rule_tac x=x **in** exI, rule_tac x=x+u **in** exI, force)

} **moreover**

{ **fix** u v x y

assume *: $S \subseteq \{a *_R u + b *_R v \mid a b. a + b = 1\}$

have $\exists c. x - y = c *_R (v - u)$ **if** x \in S y \in S

proof -

obtain a r **where** a + r = 1 x = a *_R u + r *_R v

using * $\langle x \in S \rangle$ **by** blast

moreover

obtain b s **where** b + s = 1 y = b *_R u + s *_R v

using * $\langle y \in S \rangle$ **by** blast

ultimately have x - y = (r-s) *_R (v-u)

by (simp add: algebra_simps) (metis scaleR_left.add)

then show ?thesis

```

      by blast
    qed
  } ultimately
  show ?thesis
  unfolding collinear_def affine_hull_2
    by blast
qed

lemma collinear_closed_segment [simp]: collinear (closed_segment a b)
  by (metis affine_hull_convex_hull collinear_affine_hull hull_subset segment_convex_hull)

lemma collinear_open_segment [simp]: collinear (open_segment a b)
  unfolding open_segment_def
  by (metis convex_hull_subset_affine_hull segment_convex_hull dual_order.trans
    convex_hull_subset_affine_hull Diff_subset collinear_affine_hull)

lemma collinear_between_cases:
  fixes c :: 'a::euclidean_space
  shows collinear {a,b,c}  $\longleftrightarrow$  between (b,c) a  $\vee$  between (c,a) b  $\vee$  between (a,b) c
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain u v where uv:  $\bigwedge x. x \in \{a, b, c\} \implies \exists c. x = u + c *R v$ 
    by (auto simp: collinear_alt)
  show ?rhs
    using uv [of a] uv [of b] uv [of c] by (auto simp: between_1)
next
  assume ?rhs
  then show ?lhs
    unfolding between_mem_convex_hull
    by (metis (no_types, opaque_lifting) collinear_closed_segment collinear_subset
    hull_redundant hull_subset insert_commute segment_convex_hull)
qed

lemma subset_continuous_image_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes continuous_on (closed_segment a b) f
  shows closed_segment (f a) (f b)  $\subseteq$  image f (closed_segment a b)
  by (metis connected_segment convex_contains_segment ends_in_segment imageI
    is_interval_connected_1 is_interval_convex connected_continuous_image
    [OF assms])

lemma continuous_injective_image_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes contf: continuous_on (closed_segment a b) f
    and injf: inj_on f (closed_segment a b)
  shows f ` (closed_segment a b) = closed_segment (f a) (f b)
proof

```

```

show closed_segment (f a) (f b) ⊆ f ' closed_segment a b
  by (metis subset_continuous_image_segment_1 contf)
show f ' closed_segment a b ⊆ closed_segment (f a) (f b)
proof (cases a = b)
  case True
  then show ?thesis by auto
next
case False
then have fnot: f a ≠ f b
  using inj_onD injf by fastforce
moreover
have f a ∉ open_segment (f c) (f b) if c: c ∈ closed_segment a b for c
proof (clarsimp simp add: open_segment_def)
  assume fa: f a ∈ closed_segment (f c) (f b)
  moreover have closed_segment (f c) (f b) ⊆ f ' closed_segment c b
  by (meson closed_segment_subset contf continuous_on_subset convex_closed_segment
ends_in_segment(2) subset_continuous_image_segment_1 that)
  ultimately have f a ∈ f ' closed_segment c b
  by blast
  then have a: a ∈ closed_segment c b
  by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
that)
  have cb: closed_segment c b ⊆ closed_segment a b
  by (simp add: closed_segment_subset that)
show f a = f c
proof (rule between_antisym)
  show between (f c, f b) (f a)
  by (simp add: between_mem_segment fa)
  show between (f a, f b) (f c)
  by (metis a cb between_antisym between_mem_segment between_triv1
subset_iff)
qed
qed
moreover
have f b ∉ open_segment (f a) (f c) if c: c ∈ closed_segment a b for c
proof (clarsimp simp add: open_segment_def fnot eq_commute)
  assume fb: f b ∈ closed_segment (f a) (f c)
  moreover have closed_segment (f a) (f c) ⊆ f ' closed_segment a c
  by (meson contf continuous_on_subset ends_in_segment(1) subset_closed_segment
subset_continuous_image_segment_1 that)
  ultimately have f b ∈ f ' closed_segment a c
  by blast
  then have b: b ∈ closed_segment a c
  by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
that)
  have ca: closed_segment a c ⊆ closed_segment a b
  by (simp add: closed_segment_subset that)
show f b = f c
proof (rule between_antisym)

```

```

    show between (f c, f a) (f b)
      by (simp add: between_commute between_mem_segment fb)
    show between (f b, f a) (f c)
      by (metis b between_antisym between_commute between_mem_segment
between_triv2 that)
    qed
  qed
  ultimately show ?thesis
    by (force simp: closed_segment_eq_real_ivl open_segment_eq_real_ivl split:
if_split_asm)
  qed
qed

```

```

lemma continuous_injective_image_open_segment_1:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes contf: continuous_on (closed_segment a b) f
    and injf: inj_on f (closed_segment a b)
    shows f ` (open_segment a b) = open_segment (f a) (f b)
proof -
  have f ` (open_segment a b) = f ` (closed_segment a b) - {f a, f b}
    by (metis (no_types, opaque_lifting) empty_subsetI ends_in_segment im-
age_insert image_is_empty inj_on_image_set_diff injf insert_subset open_segment_def
segment_open_subset_closed)
  also have ... = open_segment (f a) (f b)
    using continuous_injective_image_segment_1 [OF assms]
    by (simp add: open_segment_def inj_on_image_set_diff [OF injf])
  finally show ?thesis .
qed

```

```

lemma collinear_imp_coplanar:
  collinear s ==> coplanar s
by (metis collinear_affine_hull coplanar_def insert_absorb2)

```

```

lemma collinear_small:
  assumes finite s card s  $\leq$  2
  shows collinear s
proof -
  have card s = 0  $\vee$  card s = 1  $\vee$  card s = 2
    using assms by linarith
  then show ?thesis using assms
    using card_eq_SucD numeral_2_eq_2 by (force simp: card_1_singleton_iff)
qed

```

```

lemma coplanar_small:
  assumes finite s card s  $\leq$  3
  shows coplanar s
proof -
  consider card s  $\leq$  2 | card s = Suc (Suc (Suc 0))
    using assms by linarith

```

```

then show ?thesis
proof cases
  case 1
    then show ?thesis
    by (simp add: ⟨finite s⟩ collinear_imp_coplanar collinear_small)
  next
    case 2
      then show ?thesis
      using hull_subset [of {_,_,_}]
      by (fastforce simp: coplanar_def dest!: card_eq_SucD)
qed
qed

lemma coplanar_empty: coplanar {}
by (simp add: coplanar_small)

lemma coplanar_sing: coplanar {a}
by (simp add: coplanar_small)

lemma coplanar_2: coplanar {a,b}
by (auto simp: card_insert_if coplanar_small)

lemma coplanar_3: coplanar {a,b,c}
by (auto simp: card_insert_if coplanar_small)

lemma collinear_affine_hull_collinear: collinear(affine hull s)  $\longleftrightarrow$  collinear s
unfolding collinear_affine_hull
by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_affine_hull_coplanar: coplanar(affine hull s)  $\longleftrightarrow$  coplanar s
unfolding coplanar_def
by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_linear_image:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes coplanar S linear f shows coplanar(f ' S)
proof -
  { fix u v w
    assume S  $\subseteq$  affine hull {u, v, w}
    then have f ' S  $\subseteq$  f ' (affine hull {u, v, w})
      by (simp add: image_mono)
    then have f ' S  $\subseteq$  affine hull (f ' {u, v, w})
      by (metis assms(2) linear_conv_bounded_linear_affine_hull_linear_image)
  } then
  show ?thesis
  by auto (meson assms(1) coplanar_def)
qed

lemma coplanar_translation_imp:

```

```

  assumes coplanar S shows coplanar (( $\lambda$ x. a + x) ' S)
proof -
  obtain u v w where S  $\subseteq$  affine hull {u,v,w}
  by (meson assms coplanar_def)
  then have (+) a ' S  $\subseteq$  affine hull {u + a, v + a, w + a}
  using affine_hull_translation [of a {u,v,w} for u v w]
  by (force simp: add commute)
  then show ?thesis
  unfolding coplanar_def by blast
qed

lemma coplanar_translation_eq: coplanar(( $\lambda$ x. a + x) ' S)  $\longleftrightarrow$  coplanar S
  by (metis (no_types) coplanar_translation_imp translation_galois)

lemma coplanar_linear_image_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f inj f shows coplanar(f ' S) = coplanar S
proof
  assume coplanar S
  then show coplanar (f ' S)
  using assms(1) coplanar_linear_image by blast
next
  obtain g where g: linear g g  $\circ$  f = id
  using linear_injective_left_inverse [OF assms]
  by blast
  assume coplanar (f ' S)
  then show coplanar S
  by (metis coplanar_linear_image g(1) g(2) id_apply image_comp image_id)
qed

lemma coplanar_subset:  $\llbracket$ coplanar t; S  $\subseteq$  t $\rrbracket \Longrightarrow$  coplanar S
  by (meson coplanar_def order_trans)

lemma affine_hull_3_imp_collinear: c  $\in$  affine hull {a,b}  $\Longrightarrow$  collinear {a,b,c}
  by (metis collinear_2 collinear_affine_hull collinear_hull_redundant insert_commute)

lemma collinear_3_imp_in_affine_hull:
  assumes collinear {a,b,c} a  $\neq$  b shows c  $\in$  affine hull {a,b}
proof -
  obtain u x y where b - a = y *R u c - a = x *R u
  using assms unfolding collinear_def by auto
  with  $\langle$ a  $\neq$  b $\rangle$  have  $\exists$  v. c = (1 - x / y) *R a + v *R b  $\wedge$  1 - x / y + v = 1
  by (simp add: algebra_simps)
  then show ?thesis
  by (simp add: hull_inc mem_affine)
qed

lemma collinear_3_affine_hull:
  assumes a  $\neq$  b

```

shows $\text{collinear } \{a,b,c\} \longleftrightarrow c \in \text{affine hull } \{a,b\}$
using $\text{affine_hull_3_imp_collinear}$ *assms* $\text{collinear_3_imp_in_affine_hull}$ **by**
blast

lemma $\text{collinear_3_eq_affine_dependent}$:
 $\text{collinear}\{a,b,c\} \longleftrightarrow a = b \vee a = c \vee b = c \vee \text{affine_dependent } \{a,b,c\}$
proof (*cases* $a = b \vee a = c \vee b = c$)
 case *True*
 then show *?thesis*
 by (*auto simp: insert_commute*)
next
 case *False*
 then have $\text{collinear}\{a,b,c\}$ **if** $\text{affine_dependent } \{a,b,c\}$
 using that unfolding $\text{affine_dependent_def}$
 by (*auto simp: insert_Diff_if; metis affine_hull_3_imp_collinear insert_commute*)
 moreover
 have $\text{affine_dependent } \{a,b,c\}$ **if** $\text{collinear}\{a,b,c\}$
 using *False* **that by** (*auto simp: affine_dependent_def collinear_3_affine_hull*
insert_Diff_if)
 ultimately
 show *?thesis*
 using *False* **by** *blast*
qed

lemma $\text{affine_dependent_imp_collinear_3}$:
 $\text{affine_dependent } \{a,b,c\} \implies \text{collinear}\{a,b,c\}$
by (*simp add: collinear_3_eq_affine_dependent*)

lemma collinear_3 : *NO_MATCH* $0 x \implies \text{collinear } \{x,y,z\} \longleftrightarrow \text{collinear } \{0, x-y, z-y\}$
by (*auto simp add: collinear_def*)

lemma $\text{collinear_3_expand}$:
 $\text{collinear}\{a,b,c\} \longleftrightarrow a = c \vee (\exists u. b = u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} c)$
proof –
 have $\text{collinear}\{a,b,c\} = \text{collinear}\{a,c,b\}$
 by (*simp add: insert_commute*)
 also have $\dots = \text{collinear } \{0, a - c, b - c\}$
 by (*simp add: collinear_3*)
 also have $\dots \longleftrightarrow (a = c \vee b = c \vee (\exists ca. b - c = ca *_{\mathbb{R}} (a - c)))$
 by (*simp add: collinear_lemma*)
 also have $\dots \longleftrightarrow a = c \vee (\exists u. b = u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} c)$
 by (*cases* $a = c \vee b = c$) (*auto simp: algebra_simps*)
 finally show *?thesis* .
qed

lemma collinear_aff_dim : $\text{collinear } S \longleftrightarrow \text{aff_dim } S \leq 1$
proof
 assume $\text{collinear } S$


```

then obtain  $u$  and  $v :: 'a$  where  $\text{aff\_dim } S \leq \text{aff\_dim } \{u,v\}$ 
  by (metis  $\langle \text{collinear } S \rangle \text{aff\_dim\_affine\_hull aff\_dim\_subset collinear\_affine\_hull}$ )
then show  $\text{aff\_dim } S \leq 1$ 
  using order_trans by fastforce
next
  assume  $\text{aff\_dim } S \leq 1$ 
  then have  $le1: \text{aff\_dim } (\text{affine hull } S) \leq 1$ 
    by simp
  obtain  $B$  where  $B \subseteq S$  and  $B: \neg \text{affine\_dependent } B \text{ affine hull } S = \text{affine hull } B$ 
    using affine_basis_exists [of  $S$ ] by auto
  then have finite  $B$   $\text{card } B \leq 2$ 
    using  $B$   $le1$  by (auto simp: affine_independent_iff_card)
  then have collinear  $B$ 
    by (rule collinear_small)
  then show collinear  $S$ 
    by (metis  $\langle \text{affine hull } S = \text{affine hull } B \rangle \text{collinear\_affine\_hull\_collinear}$ )
qed

```

```

lemma collinear_midpoint:  $\text{collinear}\{a, \text{midpoint } a \ b, b\}$ 
proof -
  have  $\S: \llbracket a \neq \text{midpoint } a \ b; b - \text{midpoint } a \ b \neq -1 *_{\mathbb{R}} (a - \text{midpoint } a \ b) \rrbracket \implies$ 
 $b = \text{midpoint } a \ b$ 
    by (simp add: algebra_simps)
  show ?thesis
    by (auto simp: collinear_3 collinear_lemma intro: \S)
qed

```

```

lemma midpoint_collinear:
  fixes  $a \ b \ c :: 'a::\text{real\_normed\_vector}$ 
  assumes  $a \neq c$ 
  shows  $b = \text{midpoint } a \ c \iff \text{collinear}\{a,b,c\} \wedge \text{dist } a \ b = \text{dist } b \ c$ 
proof -
  have  $*$ :  $a - (u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} c) = (1 - u) *_{\mathbb{R}} (a - c)$ 
 $u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} c - c = u *_{\mathbb{R}} (a - c)$ 
 $|1 - u| = |u| \iff u = 1/2$  for  $u::\text{real}$ 
    by (auto simp: algebra_simps)
  have  $b = \text{midpoint } a \ c \implies \text{collinear}\{a,b,c\}$ 
    using collinear_midpoint by blast
  moreover have  $b = \text{midpoint } a \ c \iff \text{dist } a \ b = \text{dist } b \ c$  if  $\text{collinear}\{a,b,c\}$ 
proof -
  consider  $a = c \mid u$  where  $b = u *_{\mathbb{R}} a + (1 - u) *_{\mathbb{R}} c$ 
    using  $\langle \text{collinear } \{a,b,c\} \rangle$  unfolding collinear_3_expand by blast
  then show ?thesis
proof cases
  case 2
  with assms have  $\text{dist } a \ b = \text{dist } b \ c \implies b = \text{midpoint } a \ c$ 
    by (simp add: dist_norm * midpoint_def scaleR_add_right del: divide_const_simps)
  then show ?thesis

```

```

      by (auto simp: dist_midpoint)
    qed (use assms in auto)
  qed
  ultimately show ?thesis by blast
qed

```

```

lemma between_imp_collinear:
  fixes x :: 'a :: euclidean_space
  assumes between (a,b) x
  shows collinear {a,x,b}
proof (cases x = a ∨ x = b ∨ a = b)
  case True with assms show ?thesis
    by (auto simp: dist_commute)
next
  case False
  then have False if  $\bigwedge c. b - x \neq c *_R (a - x)$ 
    using that [of  $-(\text{norm}(b - x) / \text{norm}(x - a))$ ] assms
  by (simp add: between_norm vector_add_divide_simps flip: real_vector.scale_minus_right)
  then show ?thesis
    by (auto simp: collinear_3 collinear_lemma)
qed

```

```

lemma midpoint_between:
  fixes a b :: 'a::euclidean_space
  shows  $b = \text{midpoint } a \ c \longleftrightarrow \text{between } (a,c) \ b \wedge \text{dist } a \ b = \text{dist } b \ c$ 
proof (cases a = c)
  case False
  show ?thesis
    using False between_imp_collinear between_midpoint(1) midpoint_collinear
  by blast
qed (auto simp: dist_commute)

```

```

lemma collinear_triples:
  assumes  $a \neq b$ 
  shows  $\text{collinear}(\text{insert } a (\text{insert } b \ S)) \longleftrightarrow (\forall x \in S. \text{collinear}\{a,b,x\})$ 
  (is ?lhs = ?rhs)
proof safe
  fix x
  assume ?lhs and  $x \in S$ 
  then show collinear {a, b, x}
    using collinear_subset by force
next
  assume ?rhs
  then have  $\forall x \in S. \text{collinear}\{a,x,b\}$ 
    by (simp add: insert_commute)
  then have *:  $\exists u. x = u *_R a + (1 - u) *_R b$  if  $x \in \text{insert } a (\text{insert } b \ S)$  for x
    using that assms collinear_3_expand by fastforce+
  have  $\exists c. x - y = c *_R (b - a)$ 
    if  $x: x \in \text{insert } a (\text{insert } b \ S)$  and  $y: y \in \text{insert } a (\text{insert } b \ S)$  for x y

```

```

proof -
  obtain  $u\ v$  where  $x = u *_R a + (1 - u) *_R b$   $y = v *_R a + (1 - v) *_R b$ 
  using  $*\ x\ y$  by presburger
  then have  $x - y = (v - u) *_R (b - a)$ 
  by (simp add: scale_left_diff_distrib scale_right_diff_distrib)
  then show ?thesis ..
qed
then show ?lhs
  unfolding collinear_def by metis
qed

```

```

lemma collinear_4_3:
  assumes  $a \neq b$ 
  shows  $\text{collinear}\ \{a,b,c,d\} \longleftrightarrow \text{collinear}\{a,b,c\} \wedge \text{collinear}\{a,b,d\}$ 
  using collinear_triples [OF assms, of  $\{c,d\}$ ] by (force simp:)

```

```

lemma collinear_3_trans:
  assumes  $\text{collinear}\{a,b,c\}$   $\text{collinear}\{b,c,d\}$   $b \neq c$ 
  shows  $\text{collinear}\{a,b,d\}$ 
proof -
  have  $\text{collinear}\{b,c,a,d\}$ 
  by (metis (full_types) assms collinear_4_3 insert_commute)
  then show ?thesis
  by (simp add: collinear_subset)
qed

```

```

lemma affine_hull_2_alt:
  fixes  $a\ b :: 'a::\text{real\_vector}$ 
  shows  $\text{affine\_hull}\ \{a,b\} = \text{range}\ (\lambda u. a + u *_R (b - a))$ 
proof -
  have  $1: u *_R a + v *_R b = a + v *_R (b - a)$  if  $u + v = 1$  for  $u\ v$ 
  using that
  by (simp add: algebra_simps flip: scaleR_add_left)
  have  $2: a + u *_R (b - a) = (1 - u) *_R a + u *_R b$  for  $u$ 
  by (auto simp: algebra_simps)
  show ?thesis
  by (force simp add: affine_hull_2 dest: 1 intro!: 2)
qed

```

```

lemma interior_convex_hull_3_minimal:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $\neg \text{collinear}\{a,b,c\}$  and  $2: \text{DIM}('a) = 2$ 
  shows  $\text{interior}(\text{convex\_hull}\ \{a,b,c\}) =$ 
     $\{v. \exists x\ y\ z. 0 < x \wedge 0 < y \wedge 0 < z \wedge x + y + z = 1 \wedge x *_R a + y *_R b$ 
     $+ z *_R c = v\}$ 
    (is ?lhs = ?rhs)
proof
  have  $abc: a \neq b \wedge a \neq c \wedge b \neq c \rightarrow \text{affine\_dependent}\ \{a, b, c\}$ 
  using assms by (auto simp: collinear_3_eq_affine_dependent)

```

```

with ? show ?lhs ⊆ ?rhs
  by (fastforce simp add: interior_convex_hull_explicit_minimal)
show ?rhs ⊆ ?lhs
  using abc ?
  apply (clarsimp simp add: interior_convex_hull_explicit_minimal)
  subgoal for x y z
    by (rule_tac x=λr. (if r=a then x else if r=b then y else if r=c then z else
0) in exI) auto
  done
qed

```

6.0.9 Basic lemmas about hyperplanes and halfspaces

lemma *halfspace_Int_eq*:

$$\{x. a \cdot x \leq b\} \cap \{x. b \leq a \cdot x\} = \{x. a \cdot x = b\}$$

$$\{x. b \leq a \cdot x\} \cap \{x. a \cdot x \leq b\} = \{x. a \cdot x = b\}$$

by *auto*

lemma *hyperplane_eq_Ex*:

assumes $a \neq 0$ obtains x where $a \cdot x = b$

by (rule_tac $x = (b / (a \cdot a)) *_{\mathbb{R}} a$ in that) (simp add: assms)

lemma *hyperplane_eq_empty*:

$$\{x. a \cdot x = b\} = \{\} \longleftrightarrow a = 0 \wedge b \neq 0$$

using *hyperplane_eq_Ex*

by (metis (mono_tags, lifting) empty_Collect_eq inner_zero_left)

lemma *hyperplane_eq_UNIV*:

$$\{x. a \cdot x = b\} = \text{UNIV} \longleftrightarrow a = 0 \wedge b = 0$$

proof –

have $a = 0 \wedge b = 0$ if $\text{UNIV} \subseteq \{x. a \cdot x = b\}$

using *subsetD* [OF that, where $c = ((b+1) / (a \cdot a)) *_{\mathbb{R}} a$]

by (simp add: field_split_simps split: if_split_asm)

then show ?thesis by force

qed

lemma *halfspace_eq_empty_lt*:

$$\{x. a \cdot x < b\} = \{\} \longleftrightarrow a = 0 \wedge b \leq 0$$

proof –

have $a = 0 \wedge b \leq 0$ if $\{x. a \cdot x < b\} \subseteq \{\}$

using *subsetD* [OF that, where $c = ((b-1) / (a \cdot a)) *_{\mathbb{R}} a$]

by (force simp add: field_split_simps split: if_split_asm)

then show ?thesis by force

qed

lemma *halfspace_eq_empty_gt*:

$$\{x. a \cdot x > b\} = \{\} \longleftrightarrow a = 0 \wedge b \geq 0$$

using *halfspace_eq_empty_lt* [of $-a -b$]

by *simp*

```

lemma halfspace_eq_empty_le:
  {x. a · x ≤ b} = {} ↔ a = 0 ∧ b < 0
proof -
  have a = 0 ∧ b < 0 if {x. a · x ≤ b} ⊆ {}
    using subsetD [OF that, where c = ((b-1) / (a · a)) *R a]
    by (force simp add: field_split_simps split: if_split_asm)
  then show ?thesis by force
qed

```

```

lemma halfspace_eq_empty_ge:
  {x. a · x ≥ b} = {} ↔ a = 0 ∧ b > 0
using halfspace_eq_empty_le [of -a -b] by simp

```

6.0.10 Use set distance for an easy proof of separation properties

```

proposition separation_closures:
  fixes S :: 'a::euclidean_space set
  assumes S ∩ closure T = {} T ∩ closure S = {}
  obtains U V where U ∩ V = {} open U open V S ⊆ U T ⊆ V
proof (cases S = {} ∨ T = {})
  case True with that show ?thesis by auto
next
  case False
  define f where f ≡ λx. setdist {x} T - setdist {x} S
  have contf: continuous_on UNIV f
    unfolding f_def by (intro continuous_intros continuous_on_setdist)
  show ?thesis
proof (rule_tac U = {x. f x > 0} and V = {x. f x < 0} in that)
  show {x. 0 < f x} ∩ {x. f x < 0} = {}
    by auto
  show open {x. 0 < f x}
    by (simp add: open_Collect_less contf)
  show open {x. f x < 0}
    by (simp add: open_Collect_less contf)
  have ∧x. x ∈ S ⇒ setdist {x} T ≠ 0 ∧ x. x ∈ T ⇒ setdist {x} S ≠ 0
    by (meson False assms disjoint_iff setdist_eq_0_sing_1)+
  then show S ⊆ {x. 0 < f x} T ⊆ {x. f x < 0}
    using less_eq_real_def by (fastforce simp add: f_def setdist_sing_in_set)+
qed
qed

```

```

lemma separation_normal:
  fixes S :: 'a::euclidean_space set
  assumes closed S closed T S ∩ T = {}
  obtains U V where open U open V S ⊆ U T ⊆ V U ∩ V = {}
using separation_closures [of S T]
by (metis assms closure_closed disjnt_def inf_commute)

```

```

lemma separation_normal_local:
  fixes S :: 'a::euclidean_space set
  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and S ∩ T = {}
  obtains S' T' where openin (top_of_set U) S'
    openin (top_of_set U) T'
    S ⊆ S' T ⊆ T' S' ∩ T' = {}
proof (cases S = {} ∨ T = {})
  case True with that show ?thesis
    using UT US by (blast dest: closedin_subset)
next
  case False
  define f where f ≡ λx. setdist {x} T - setdist {x} S
  have contf: continuous_on U f
    unfolding f_def by (intro continuous_intros)
  show ?thesis
  proof (rule_tac S' = (U ∩ f -' {0<..}) and T' = (U ∩ f -' {..<0}) in that)
    show (U ∩ f -' {0<..}) ∩ (U ∩ f -' {..<0}) = {}
      by auto
    show openin (top_of_set U) (U ∩ f -' {0<..})
      by (rule continuous_openin_preimage [where T=UNIV]) (simp_all add:
contf)
    next
      show openin (top_of_set U) (U ∩ f -' {..<0})
        by (rule continuous_openin_preimage [where T=UNIV]) (simp_all add:
contf)
    next
      have S ⊆ U T ⊆ U
        using closedin_imp_subset assms by blast+
      then show S ⊆ U ∩ f -' {0<..} T ⊆ U ∩ f -' {..<0}
        using assms False by (force simp add: f_def setdist_sing_in_set intro!:
setdist_gt_0_closedin)+
    qed
  qed

```

```

lemma separation_normal_compact:
  fixes S :: 'a::euclidean_space set
  assumes compact S closed T S ∩ T = {}
  obtains U V where open U compact(closure U) open V S ⊆ U T ⊆ V U ∩ V
= {}
proof -
  have closed S bounded S
    using assms by (auto simp: compact_eq_bounded_closed)
  then obtain r where r>0 and r: S ⊆ ball 0 r
    by (auto dest!: bounded_subset_ballD)
  have **: closed (T ∪ - ball 0 r) S ∩ (T ∪ - ball 0 r) = {}
    using assms r by blast+

```

```

then obtain  $U V$  where  $UV: \text{open } U \text{ open } V \ S \subseteq U \ T \cup - \text{ball } 0 \ r \subseteq V \ U \cap V = \{\}$ 
by (meson  $\langle \text{closed } S \rangle \text{ separation\_normal}$ )
then have  $\text{compact}(\text{closure } U)$ 
by (meson  $\text{bounded\_ball bounded\_subset compact\_closure compl\_le\_swap2 disjoint\_eq\_subset\_Compl le\_sup\_iff}$ )
with  $UV$  show thesis
using that by auto
qed

```

6.0.11 Connectedness of the intersection of a chain

proposition *connected_chain*:

```

fixes  $\mathcal{F} :: 'a :: \text{euclidean\_space set set}$ 
assumes  $cc: \bigwedge S. S \in \mathcal{F} \implies \text{compact } S \wedge \text{connected } S$ 
and  $\text{linear}: \bigwedge S \ T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
shows  $\text{connected}(\bigcap \mathcal{F})$ 
proof (cases  $\mathcal{F} = \{\}$ )
case True then show ?thesis
by auto
next
case False
then have  $cf: \text{compact}(\bigcap \mathcal{F})$ 
by (simp add: cc compact_Inter)
have  $\text{False}$  if  $AB: \text{closed } A \ \text{closed } B \ A \cap B = \{\}$ 
and  $ABeq: A \cup B = \bigcap \mathcal{F}$  and  $A \neq \{\}$   $B \neq \{\}$  for  $A \ B$ 
proof –
obtain  $U V$  where  $\text{open } U \ \text{open } V \ A \subseteq U \ B \subseteq V \ U \cap V = \{\}$ 
using separation_normal [OF AB] by metis
obtain  $K$  where  $K \in \mathcal{F}$   $\text{compact } K$ 
using  $cc$  False by blast
then obtain  $N$  where  $\text{open } N$  and  $K \subseteq N$ 
by blast
let  $\mathcal{C} = \text{insert } (U \cup V) ((\lambda S. N - S) \ ' \mathcal{F})$ 
obtain  $\mathcal{D}$  where  $\mathcal{D} \subseteq \mathcal{C}$   $\text{finite } \mathcal{D}$   $K \subseteq \bigcup \mathcal{D}$ 
proof (rule compactE [OF  $\langle \text{compact } K \rangle$ ])
show  $K \subseteq \bigcup (\text{insert } (U \cup V) ((-) \ N \ ' \mathcal{F}))$ 
using  $\langle K \subseteq N \rangle \ ABeq \ \langle A \subseteq U \rangle \ \langle B \subseteq V \rangle$  by auto
show  $\bigwedge B. B \in \text{insert } (U \cup V) ((-) \ N \ ' \mathcal{F}) \implies \text{open } B$ 
by (auto simp:  $\langle \text{open } U \rangle \ \langle \text{open } V \rangle \ \text{open\_Un} \ \langle \text{open } N \rangle \ cc \ \text{compact\_imp\_closed open\_Diff}$ )
qed
then have  $\text{finite}(\mathcal{D} - \{U \cup V\})$ 
by blast
moreover have  $\mathcal{D} - \{U \cup V\} \subseteq (\lambda S. N - S) \ ' \mathcal{F}$ 
using  $\langle \mathcal{D} \subseteq \mathcal{C} \rangle$  by blast
ultimately obtain  $\mathcal{G}$  where  $\mathcal{G} \subseteq \mathcal{F}$   $\text{finite } \mathcal{G}$  and  $\text{Deq}: \mathcal{D} - \{U \cup V\} = (\lambda S. N - S) \ ' \mathcal{G}$ 
using finite_subset_image by metis

```

```

obtain  $J$  where  $J \in \mathcal{F}$  and  $J: (\bigcup S \in \mathcal{G}. N - S) \subseteq N - J$ 
proof (cases  $\mathcal{G} = \{\}$ )
  case True
    with  $\langle \mathcal{F} \neq \{\} \rangle$  that show ?thesis
      by auto
  next
    case False
      have  $\bigwedge S T. \llbracket S \in \mathcal{G}; T \in \mathcal{G} \rrbracket \implies S \subseteq T \vee T \subseteq S$ 
        by (meson  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  in_mono local.linear)
      with  $\langle \text{finite } \mathcal{G} \rangle \langle \mathcal{G} \neq \{\} \rangle$ 
      have  $\exists J \in \mathcal{G}. (\bigcup S \in \mathcal{G}. N - S) \subseteq N - J$ 
      proof induction
        case (insert  $X \mathcal{H}$ )
          show ?case
          proof (cases  $\mathcal{H} = \{\}$ )
            case True then show ?thesis by auto
          next
            case False
              then have  $\bigwedge S T. \llbracket S \in \mathcal{H}; T \in \mathcal{H} \rrbracket \implies S \subseteq T \vee T \subseteq S$ 
                by (simp add: insert.prems)
              with insert.IH False obtain  $J \in \mathcal{H}$  and  $J: (\bigcup Y \in \mathcal{H}. N - Y)$ 
                 $\subseteq N - J$ 
                by metis
              have  $N - J \subseteq N - X \vee N - X \subseteq N - J$ 
                by (meson Diff_mono  $\langle J \in \mathcal{H} \rangle$  insert.prems(2) insert_iff order_refl)
              then show ?thesis
              proof
                assume  $N - J \subseteq N - X$  with  $J$  show ?thesis
                  by auto
                next
                  assume  $N - X \subseteq N - J$ 
                  with  $J$  have  $N - X \cup \bigcup ((-) N \setminus \mathcal{H}) \subseteq N - J$ 
                    by auto
                  with  $\langle J \in \mathcal{H} \rangle$  show ?thesis
                    by blast
                qed
              qed
            qed simp
            with  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  show ?thesis by (blast intro: that)
          qed
        have  $K \subseteq \bigcup (\text{insert } (U \cup V) (\mathcal{D} - \{U \cup V\}))$ 
          using  $\langle K \subseteq \bigcup \mathcal{D} \rangle$  by auto
        also have  $\dots \subseteq (U \cup V) \cup (N - J)$ 
          by (metis (no_types, opaque_lifting) Deq Un_subset_iff Un_upper2 J Union_insert
order_trans sup_ge1)
        finally have  $J \cap K \subseteq U \cup V$ 
          by blast
        moreover have connected( $J \cap K$ )
          by (metis Int_absorb1  $\langle J \in \mathcal{F} \rangle \langle K \in \mathcal{F} \rangle$  cc inf.orderE local.linear)

```



```

moreover have  $U \cap (J \cap K) \neq \{\}$ 
  using ABeq  $\langle J \in \mathcal{F} \rangle \langle K \in \mathcal{F} \rangle \langle A \neq \{\} \rangle \langle A \subseteq U \rangle$  by blast
moreover have  $V \cap (J \cap K) \neq \{\}$ 
  using ABeq  $\langle J \in \mathcal{F} \rangle \langle K \in \mathcal{F} \rangle \langle B \neq \{\} \rangle \langle B \subseteq V \rangle$  by blast
ultimately show False
  using connectedD [of  $J \cap K \ U \ V$ ]  $\langle \text{open } U \rangle \langle \text{open } V \rangle \langle U \cap V = \{\} \rangle$  by
auto
qed
with cf show ?thesis
  by (auto simp: connected_closed_set compact_imp_closed)
qed

```

lemma *connected_chain_gen*:

```

fixes  $\mathcal{F} :: 'a :: \text{euclidean\_space set set}$ 
assumes  $X: X \in \mathcal{F}$  compact  $X$ 
  and cc:  $\bigwedge T. T \in \mathcal{F} \implies \text{closed } T \wedge \text{connected } T$ 
  and linear:  $\bigwedge S \ T. S \in \mathcal{F} \wedge T \in \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
shows connected( $\bigcap \mathcal{F}$ )
proof -
  have  $\bigcap \mathcal{F} = (\bigcap T \in \mathcal{F}. X \cap T)$ 
    using  $X$  by blast
  moreover have connected ( $\bigcap T \in \mathcal{F}. X \cap T$ )
    proof (rule connected_chain)
      show  $\bigwedge T. T \in (\bigcap) X \ ' \mathcal{F} \implies \text{compact } T \wedge \text{connected } T$ 
        using cc  $X$  by auto (metis inf.absorb2 inf.orderE local.linear)
      show  $\bigwedge S \ T. S \in (\bigcap) X \ ' \mathcal{F} \wedge T \in (\bigcap) X \ ' \mathcal{F} \implies S \subseteq T \vee T \subseteq S$ 
        using local.linear by blast
    qed
  ultimately show ?thesis
    by metis
qed

```

lemma *connected_nest*:

```

fixes  $S :: 'a :: \text{linorder} \Rightarrow 'b :: \text{euclidean\_space set}$ 
assumes  $S: \bigwedge n. \text{compact}(S \ n) \wedge \text{connected}(S \ n)$ 
  and nest:  $\bigwedge m \ n. m \leq n \implies S \ n \subseteq S \ m$ 
shows connected( $\bigcap (\text{range } S)$ )
proof (rule connected_chain)
  show  $\bigwedge A \ T. A \in \text{range } S \wedge T \in \text{range } S \implies A \subseteq T \vee T \subseteq A$ 
    by (metis image_iff le_cases nest)
qed (use S in blast)

```

lemma *connected_nest_gen*:

```

fixes  $S :: 'a :: \text{linorder} \Rightarrow 'b :: \text{euclidean\_space set}$ 
assumes  $S: \bigwedge n. \text{closed}(S \ n) \wedge \text{connected}(S \ n) \text{ compact}(S \ k)$ 
  and nest:  $\bigwedge m \ n. m \leq n \implies S \ n \subseteq S \ m$ 
shows connected( $\bigcap (\text{range } S)$ )
proof (rule connected_chain_gen [of S k])
  show  $\bigwedge A \ T. A \in \text{range } S \wedge T \in \text{range } S \implies A \subseteq T \vee T \subseteq A$ 

```

by (*metis imageE le_cases nest*)
 qed (*use S in auto*)

6.0.12 Proper maps, including projections out of compact sets

lemma *finite_indexed_bound*:

assumes A : *finite* $A \wedge x. x \in A \implies \exists n::'a::linorder. P x n$
 shows $\exists m. \forall x \in A. \exists k \leq m. P x k$

using A

proof (*induction A*)

case *empty* then show ?case by force

next

case (*insert a A*)

then obtain $m n$ where $\forall x \in A. \exists k \leq m. P x k P a n$

by force

then show ?case

by (*metis dual_order.trans insert_iff le_cases*)

qed

proposition *proper_map*:

fixes $f :: 'a::heine_borel \Rightarrow 'b::heine_borel$

assumes *closedin* (*top_of_set S*) K

and *com*: $\bigwedge U. [U \subseteq T; \text{compact } U] \implies \text{compact } (S \cap f^{-1} U)$

and $f^{-1} S \subseteq T$

shows *closedin* (*top_of_set T*) ($f^{-1} K$)

proof –

have $K \subseteq S$

using *assms closedin_imp_subset* by *metis*

obtain C where *closed* C and *Keq*: $K = S \cap C$

using *assms* by (*auto simp: closedin_closed*)

have $*$: $y \in f^{-1} K$ if $y \in T$ and y : y *islimpt* $f^{-1} K$ for y

proof –

obtain h where $\forall n. (\exists x \in K. h n = f x) \wedge h n \neq y$ *inj* h and *hlim*: ($h \longrightarrow y$) *sequentially*

using $\langle y \in T \rangle y$ by (*force simp: limpt_sequential_inj*)

then obtain X where X : $\bigwedge n. X n \in K \wedge h n = f (X n) \wedge h n \neq y$

by *metis*

then have *fX*: $\bigwedge n. f (X n) = h n$

by *metis*

define Ψ where $\Psi \equiv \lambda n. \{a \in K. f a \in \text{insert } y (\text{range } (\lambda i. f (X (n + i))))\}$

have *compact* ($C \cap (S \cap f^{-1} \text{insert } y (\text{range } (\lambda i. f (X (n + i))))$) for n

proof (*intro closed_Int_compact [OF <closed C> com] compact_sequence_with_limit*)

show *insert* y ($\text{range } (\lambda i. f (X (n + i)))$) $\subseteq T$

using $X \langle K \subseteq S \rangle \langle f^{-1} S \subseteq T \rangle \langle y \in T \rangle$ by *blast*

show $(\lambda i. f (X (n + i))) \longrightarrow y$

by (*simp add: fX add.commute [of n] LIMSEQ_ignore_initial_segment [OF*

hlim])

qed

```

then have comf: compact ( $\Psi$  n) for n
  by (simp add: Keq Int_def  $\Psi$ _def conj_commute)
have ne:  $\bigcap \mathcal{F} \neq \{\}$ 
  if finite  $\mathcal{F}$ 
  and  $\mathcal{F}$ :  $\bigwedge t. t \in \mathcal{F} \implies (\exists n. t = \Psi n)$ 
  for  $\mathcal{F}$ 
proof -
  obtain m where  $m$ :  $\bigwedge t. t \in \mathcal{F} \implies \exists k \leq m. t = \Psi k$ 
  by (rule exE [OF finite_indexed_bound [OF <finite  $\mathcal{F}$ >  $\mathcal{F}$ ]], force+)
  have  $X m \in \bigcap \mathcal{F}$ 
  using  $X$  le_Suc_ex by (fastforce simp:  $\Psi$ _def dest: m)
  then show ?thesis by blast
qed
have ( $\bigcap n. \Psi n$ )  $\neq \{\}$ 
proof (rule compact_fip_Heine_Borel)
  show  $\bigwedge \mathcal{F}'. [\text{finite } \mathcal{F}'; \mathcal{F}' \subseteq \text{range } \Psi] \implies \bigcap \mathcal{F}' \neq \{\}$ 
  by (meson ne rangeE subset_eq)
qed (use comf in blast)
then obtain x where  $x \in K \bigwedge n. (f x = y \vee (\exists u. f x = h (n + u)))$ 
  by (force simp add:  $\Psi$ _def fX)
then show ?thesis
  unfolding image_iff by (metis <inj h> le_add1 not_less_eq_eq rangeI
range_ex1_eq)
qed
with assms closedin_subset show ?thesis
  by (force simp: closedin_limpt)
qed

```

6.0.13 Closure of conic hulls

proposition *closedin_conic_hull*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes *compact* T $0 \notin T$ $T \subseteq S$

shows *closedin* (*top_of_set* (*conic hull* S)) (*conic hull* T)

proof -

have **: *compact* ($\{0..\} \times T \cap (\lambda z. \text{fst } z *_R \text{snd } z) - 'K$) (is *compact* ? L)

if $K \subseteq (\lambda z. \text{fst } z *_R \text{snd } z) - (\{0..\} \times S)$ *compact* K for K

proof -

obtain *r* where $r > 0$ and r : $\bigwedge x. x \in K \implies \text{norm } x \leq r$

by (metis <compact K > *bounded_normE compact_imp_bounded*)

show ?thesis

unfolding *compact_eq_bounded_closed*

proof

have *bounded* ($\{0..r / \text{setdist}\{0\}T\} \times T$)

by (simp add: *assms(1) bounded_Times compact_imp_bounded*)

moreover have ? $L \subseteq (\{0..r / \text{setdist}\{0\}T\} \times T)$

proof *clarsimp*

fix *a b*

assume $a *_R b \in K$ and $b \in T$ and $0 \leq a$

```

have setdist {0} T ≠ 0
  using ⟨b ∈ T⟩ assms compact_imp_closed setdist_eq_0_closed by auto
then have T0: setdist {0} T > 0
  using less_eq_real_def by fastforce
then have a * setdist {0} T ≤ r
  by (smt (verit, ccfv_SIG) ⟨0 ≤ a⟩ ⟨a *R b ∈ K⟩ ⟨b ∈ T⟩ dist_0_norm
mult_mono' norm_scaleR r setdist_le_dist singletonI)
  with T0 ⟨r>0⟩ show a ≤ r / setdist {0} T
    by (simp add: divide_simps)
qed
ultimately show bounded ?L
  by (meson bounded_subset)
show closed ?L
proof (rule continuous_closed_preimage)
  show continuous_on ({0..} × T) (λz. fst z *R snd z)
    by (intro continuous_intros)
  show closed ({0::real..} × T)
    by (simp add: assms(1) closed_Times compact_imp_closed)
  show closed K
    by (simp add: compact_imp_closed that(2))
qed
qed
qed
show ?thesis
  unfolding conic_hull_as_image
proof (rule proper_map)
  show compact ({0..} × T ∩ (λz. fst z *R snd z) -' K) (is compact ?L)
    if K ⊆ (λz. (fst z) *R snd z) '({0..} × S) compact K for K
  proof -
    obtain r where r > 0 and r: ∧x. x ∈ K ⇒ norm x ≤ r
      by (metis ⟨compact K⟩ bounded_normE compact_imp_bounded)
    show ?thesis
      unfolding compact_eq_bounded_closed
    proof
      have bounded ({0..r / setdist{0}T} × T)
        by (simp add: assms(1) bounded_Times compact_imp_bounded)
      moreover have ?L ⊆ ({0..r / setdist{0}T} × T)
      proof clarsimp
        fix a b
        assume a *R b ∈ K and b ∈ T and 0 ≤ a
        have setdist {0} T ≠ 0
          using ⟨b ∈ T⟩ assms compact_imp_closed setdist_eq_0_closed by auto
        then have T0: setdist {0} T > 0
          using less_eq_real_def by fastforce
        then have a * setdist {0} T ≤ r
          by (smt (verit, ccfv_SIG) ⟨0 ≤ a⟩ ⟨a *R b ∈ K⟩ ⟨b ∈ T⟩ dist_0_norm
mult_mono' norm_scaleR r setdist_le_dist singletonI)
        with T0 ⟨r>0⟩ show a ≤ r / setdist {0} T
          by (simp add: divide_simps)
      qed
    qed
  qed

```

```

qed
ultimately show bounded ?L
  by (meson bounded_subset)
show closed ?L
proof (rule continuous_closed_preimage)
  show continuous_on ({0..} × T) (λz. fst z *R snd z)
    by (intro continuous_intros)
  show closed ({0::real..} × T)
    by (simp add: assms(1) closed_Times compact_imp_closed)
  show closed K
    by (simp add: compact_imp_closed that(2))
qed
qed
qed
show (λz. fst z *R snd z) ‘({0::real..} × T) ⊆ (λz. fst z *R snd z) ‘({0..} ×
S)
  using ‹T ⊆ S› by force
qed auto
qed

```

```

lemma closed_conic_hull:
  fixes S :: 'a::euclidean_space set
  assumes 0 ∈ rel_interior S ∨ compact S ∧ 0 ∉ S
  shows closed(conic hull S)
  using assms
proof
  assume 0 ∈ rel_interior S
  then show closed (conic hull S)
    by (simp add: conic_hull_eq_span)
next
  assume compact S ∧ 0 ∉ S
  then have closedin (top_of_set UNIV) (conic hull S)
    using closedin_conic_hull by force
  then show closed (conic hull S)
    by simp
qed

```

```

lemma conic_closure:
  fixes S :: 'a::euclidean_space set
  shows conic S ⇒ conic(closure S)
  by (meson Convex.cone_def cone_closure conic_def)

```

```

lemma closure_conic_hull:
  fixes S :: 'a::euclidean_space set
  assumes 0 ∈ rel_interior S ∨ bounded S ∧ ~ (0 ∈ closure S)
  shows closure(conic hull S) = conic hull (closure S)
  using assms
proof
  assume 0 ∈ rel_interior S

```

```

then show closure (conic hull S) = conic hull closure S
  by (metis closed_affine_hull_closure_closed closure_same_affine_hull closure_subset conic_hull_eq_affine_hull subsetD subset_rel_interior)
next
  have  $\bigwedge x. x \in \text{conic hull closure } S \implies x \in \text{closure (conic hull } S)$ 
  by (metis (no_types, opaque_lifting) closure_mono conic_closure conic_conic_hull subset_eq subset_hull)
  moreover
  assume bounded S  $\wedge 0 \notin \text{closure } S$ 
  then have  $\bigwedge x. x \in \text{closure (conic hull } S) \implies x \in \text{conic hull closure } S$ 
  by (metis closed_conic_hull_closure_Un_frontier closure_closed closure_mono compact_closure hull_Un_subset le_sup_iff subsetD)
  ultimately show closure (conic hull S) = conic hull closure S
  by blast
qed

```

```

lemma compact_continuous_image_eq:
  fixes f :: 'a::heine_borel  $\Rightarrow$  'b::heine_borel
  assumes f: inj_on f S
  shows continuous_on S f  $\longleftrightarrow (\forall T. \text{compact } T \wedge T \subseteq S \longrightarrow \text{compact}(f \text{ ` } T))$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs then show ?rhs
  by (metis continuous_on_subset compact_continuous_image)
next
  assume RHS: ?rhs
  obtain g where gf:  $\bigwedge x. x \in S \implies g (f x) = x$ 
  by (metis inv_into_f_f)
  then have *:  $(S \cap f \text{ ` } U) = g \text{ ` } U$  if  $U \subseteq f \text{ ` } S$  for U
  using that by fastforce
  have gfm:  $g \text{ ` } f \text{ ` } S \subseteq S$  using gf by auto
  have **: compact (f ` S  $\cap$  g ` C) if C:  $C \subseteq S$  compact C for C
  proof -
    obtain h where h C  $\in C \wedge h C \notin S \vee \text{compact } (f \text{ ` } C)$ 
    by (force simp: C RHS)
    moreover have f ` C = (f ` S  $\cap$  g ` C)
    using C gf by auto
    ultimately show ?thesis
    using C by auto
  qed
  show ?lhs
  using proper_map [OF _ _ gfm] **
  by (simp add: continuous_on_closed * closedin_imp_subset)
qed

```

6.0.14 Trivial fact: convexity equals connectedness for collinear sets

```

lemma convex_connected_collinear:

```

```

fixes S :: 'a::euclidean_space set
assumes collinear S
shows convex S  $\longleftrightarrow$  connected S
proof
  assume convex S
  then show connected S
    using convex_connected by blast
next
assume S: connected S
show convex S
proof (cases S = {})
  case True
  then show ?thesis by simp
next
  case False
  then obtain a where a  $\in$  S by auto
  have collinear (affine hull S)
    by (simp add: assms collinear_affine_hull_collinear)
  then obtain z where z  $\neq$  0  $\wedge$   $x \in$  affine hull S  $\implies \exists c. x - a = c *_R z$ 
    by (meson  $\langle a \in S \rangle$  collinear_hull_inc)
  then obtain f where f:  $\wedge x. x \in$  affine hull S  $\implies x - a = f x *_R z$ 
    by metis
  then have inj_f: inj_on f (affine hull S)
    by (metis diff_add_cancel inj_onI)
  have diff:  $x - y = (f x - f y) *_R z$  if  $x: x \in$  affine hull S and  $y: y \in$  affine
hull S for x y
  proof -
    have f x *_R z = x - a
      by (simp add: f_hull_inc x)
    moreover have f y *_R z = y - a
      by (simp add: f_hull_inc y)
    ultimately show ?thesis
      by (simp add: scaleR_left.diff)
  qed
  have cont_f: continuous_on (affine hull S) f
  proof (clarsimp simp: dist_norm continuous_on_iff diff)
    show  $\wedge x e. 0 < e \implies \exists d > 0. \forall y \in$  affine hull S.  $|f y - f x| * \text{norm } z < d$ 
 $\longrightarrow |f y - f x| < e$ 
    by (metis  $\langle z \neq 0 \rangle$  mult_pos_pos mult_less_cancel_right_pos zero_less_norm_iff)
  qed
  then have conn_fS: connected (f ' S)
  by (meson S connected_continuous_image continuous_on_subset hull_subset)
  show ?thesis
  proof (clarsimp simp: convex_contains_segment)
    fix x y z
    assume x  $\in$  S y  $\in$  S z  $\in$  closed_segment x y
    have False if z  $\notin$  S
    proof -
      have f ' (closed_segment x y) = closed_segment (f x) (f y)

```

```

proof (rule continuous_injective_image_segment_1)
  show continuous_on (closed_segment x y) f
  by (meson ⟨x ∈ S⟩ ⟨y ∈ S⟩ convex_affine_hull convex_contains_segment
hull_inc continuous_on_subset [OF cont_f])
  show inj_on f (closed_segment x y)
  by (meson ⟨x ∈ S⟩ ⟨y ∈ S⟩ convex_affine_hull convex_contains_segment
hull_inc inj_on_subset [OF inj_f])
  qed
  then have fz: f z ∈ closed_segment (f x) (f y)
  using ⟨z ∈ closed_segment x y⟩ by blast
  have z ∈ affine_hull S
  by (meson ⟨x ∈ S⟩ ⟨y ∈ S⟩ ⟨z ∈ closed_segment x y⟩ convex_affine_hull
convex_contains_segment hull_inc subset_eq)
  then have fz_notin: f z ∉ f' S
  using hull_subset inj_f inj_onD that by fastforce
  moreover have {..proof -
    consider f x ≤ f z ∧ f z ≤ f y | f y ≤ f z ∧ f z ≤ f x
    using fz
    by (auto simp add: closed_segment_eq_real_ivl split: if_split_asm)
  then have {..by cases (use fz_notin ⟨x ∈ S⟩ ⟨y ∈ S⟩ in ⟨auto simp: image_iff⟩)
  then show {..using ⟨x ∈ S⟩ ⟨y ∈ S⟩ by blast+
  qed
  ultimately show False
    using connectedD [OF conn_fS, of {..by force
  qed
  then show z ∈ S by meson
  qed
qed
qed
qed

```

lemma compact_convex_collinear_segment_alt:

fixes S :: 'a::euclidean_space set

assumes S ≠ {} compact S connected S collinear S

obtains a b **where** S = closed_segment a b

proof -

obtain ξ **where** ξ ∈ S **using** ⟨S ≠ {}⟩ **by** auto

have collinear (affine_hull S)

by (simp add: assms collinear_affine_hull_collinear)

then obtain z **where** z ≠ 0 ∧ x. x ∈ affine_hull S ⇒ ∃ c. x - ξ = c *_R z

by (meson ⟨ξ ∈ S⟩ collinear_hull_inc)

then obtain f **where** f: ∧x. x ∈ affine_hull S ⇒ x - ξ = f x *_R z

bymetis

let ?g = λr. r *_R z + ξ

have gf: ?g (f x) = x **if** x ∈ affine_hull S **for** x

by (metis diff_add_cancel f **that**)

then have inj_f: inj_on f (affine_hull S)


```

    by (metis inj_onI)
  have diff:  $x - y = (f x - f y) *_R z$  if  $x: x \in \text{affine hull } S$  and  $y: y \in \text{affine hull } S$  for  $x y$ 
  proof -
    have  $f x *_R z = x - \xi$ 
      by (simp add: f_hull_inc x)
    moreover have  $f y *_R z = y - \xi$ 
      by (simp add: f_hull_inc y)
    ultimately show ?thesis
      by (simp add: scaleR_left.diff)
  qed
  have cont_f: continuous_on (affine hull S) f
  proof (clarsimp simp: dist_norm continuous_on_iff diff)
    show  $\bigwedge x e. 0 < e \implies \exists d > 0. \forall y \in \text{affine hull } S. |f y - f x| *_R z < d \implies |f y - f x| < e$ 
      by (metis  $\langle z \neq 0 \rangle$  mult_pos_pos mult_less_cancel_right_pos zero_less_norm_iff)
  qed
  then have connected (f ' S)
    by (meson  $\langle \text{connected } S \rangle$  connected_continuous_image continuous_on_subset hull_subset)
  moreover have compact (f ' S)
    by (meson  $\langle \text{compact } S \rangle$  compact_continuous_image_eq cont_f hull_subset inj_f)
  ultimately obtain  $x y$  where  $f ' S = \{x..y\}$ 
    by (meson connected_compact_interval_1)
  then have fS_eq:  $f ' S = \text{closed\_segment } x y$ 
    using  $\langle S \neq \{\} \rangle$  closed_segment_eq_real_ivl by auto
  obtain  $a b$  where  $a \in S$   $f a = x$   $b \in S$   $f b = y$ 
    by (metis (full_types) ends_in_segment fS_eq imageE)
  have  $f '( \text{closed\_segment } a b ) = \text{closed\_segment } (f a) (f b)$ 
  proof (rule continuous_injective_image_segment_1)
    show continuous_on (closed_segment a b) f
      by (meson  $\langle a \in S \rangle \langle b \in S \rangle$  convex_affine_hull convex_contains_segment hull_inc continuous_on_subset [OF cont_f])
    show inj_on f (closed_segment a b)
      by (meson  $\langle a \in S \rangle \langle b \in S \rangle$  convex_affine_hull convex_contains_segment hull_inc inj_on_subset [OF inj_f])
  qed
  then have  $f '( \text{closed\_segment } a b ) = f ' S$ 
    by (simp add:  $\langle f a = x \rangle \langle f b = y \rangle$  fS_eq)
  then have ?g '  $f '( \text{closed\_segment } a b ) = ?g ' f ' S$ 
    by simp
  moreover have  $(\lambda x. f x *_R z + \xi) ' \text{closed\_segment } a b = \text{closed\_segment } a b$ 
    unfolding image_def using  $\langle a \in S \rangle \langle b \in S \rangle$ 
    by (safe; metis (mono_tags, lifting) convex_affine_hull convex_contains_segment gf_hull_subset subsetCE)
  ultimately have closed_segment a b = S
    using gf by (simp add: image_comp_o_def hull_inc cong: image_cong)
  then show ?thesis

```

using that by blast
qed

lemma compact_convex_collinear_segment:
fixes $S :: 'a::euclidean_space$ set
assumes $S \neq \{\}$ compact S convex S collinear S
obtains a b where $S = \text{closed_segment } a$ b
using assms convex_connected_collinear compact_convex_collinear_segment_alt
by blast

lemma proper_map_from_compact:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes $\text{contf: continuous_on } S$ f and $\text{imf: } f \in S \rightarrow T$ and compact S
 $\text{closedin (top_of_set } T) K$
shows compact $(S \cap f^{-1} K)$
by (rule closedin_compact [OF \langle compact S \rangle] continuous_closedin_preimage_gen
assms)+

lemma proper_map_fst:
assumes compact T $K \subseteq S$ compact K
shows compact $(S \times T \cap \text{fst}^{-1} K)$
proof -
have $(S \times T \cap \text{fst}^{-1} K) = K \times T$
using assms by auto
then show ?thesis by (simp add: assms compact_Times)
qed

lemma closed_map_fst:
fixes $S :: 'a::euclidean_space$ set and $T :: 'b::euclidean_space$ set
assumes compact T closedin (top_of_set $(S \times T)$) c
shows closedin (top_of_set S) $(\text{fst}^{-1} c)$
proof -
have *: $\text{fst}^{-1} (S \times T) \subseteq S$
by auto
show ?thesis
using proper_map [OF __ *] by (simp add: proper_map_fst assms)
qed

lemma proper_map_snd:
assumes compact S $K \subseteq T$ compact K
shows compact $(S \times T \cap \text{snd}^{-1} K)$
proof -
have $(S \times T \cap \text{snd}^{-1} K) = S \times K$
using assms by auto
then show ?thesis by (simp add: assms compact_Times)
qed

lemma closed_map_snd:

```

fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $compact\ S\ closedin\ (top\_of\_set\ (S \times T))\ c$ 
shows  $closedin\ (top\_of\_set\ T)\ (snd\ 'c)$ 
proof -
have  $*: snd\ 'c\ (S \times T) \subseteq T$ 
by auto
show ?thesis
using  $proper\_map\ [OF\ \_ \_]\ * \mathbf{by}\ (simp\ add:\ proper\_map\_snd\ assms)$ 
qed

```

```

lemma closedin_compact_projection:
fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $compact\ S$  and  $clo:\ closedin\ (top\_of\_set\ (S \times T))\ U$ 
shows  $closedin\ (top\_of\_set\ T)\ \{y.\ \exists x.\ x \in S \wedge (x, y) \in U\}$ 
proof -
have  $U \subseteq S \times T$ 
by  $(metis\ clo\ closedin\_imp\_subset)$ 
then have  $\{y.\ \exists x.\ x \in S \wedge (x, y) \in U\} = snd\ 'U$ 
by force
moreover have  $closedin\ (top\_of\_set\ T)\ (snd\ 'U)$ 
by  $(rule\ closed\_map\_snd\ [OF\ assms])$ 
ultimately show ?thesis
by simp
qed

```

```

lemma closed_compact_projection:
fixes  $S :: 'a::euclidean\_space\ set$ 
and  $T :: ('a * 'b::euclidean\_space)\ set$ 
assumes  $compact\ S$  and  $clo:\ closed\ T$ 
shows  $closed\ \{y.\ \exists x.\ x \in S \wedge (x, y) \in T\}$ 
proof -
have  $*: \{y.\ \exists x.\ x \in S \wedge Pair\ x\ y \in T\} = \{y.\ \exists x.\ x \in S \wedge Pair\ x\ y \in ((S \times UNIV) \cap T)\}$ 
by auto
show ?thesis
unfolding  $*$ 
by  $(intro\ clo\ closedin\_closed\_Int\ closedin\_closed\_trans\ [OF\ \_ \_]\ closed\_UNIV)$ 
closedin_compact_projection  $[OF\ \langle compact\ S \rangle]$ 
qed

```

Representing affine hull as a finite intersection of hyperplanes

```

proposition affine_hull_convex_Int_nonempty_interior:
fixes  $S :: 'a::real\_normed\_vector\ set$ 
assumes  $convex\ S\ S \cap interior\ T \neq \{\}$ 
shows  $affine\ hull\ (S \cap T) = affine\ hull\ S$ 
proof
show  $affine\ hull\ (S \cap T) \subseteq affine\ hull\ S$ 

```

```

    by (simp add: hull_mono)
next
obtain  $a$  where  $a \in S$   $a \in T$  and  $at: a \in \text{interior } T$ 
  using assms interior_subset by blast
then obtain  $e$  where  $e > 0$  and  $e: \text{cball } a \ e \subseteq T$ 
  using mem_interior_cball by blast
have  $*$ :  $x \in (+) \ a \ \text{'span } ((\lambda x. x - a) \ \text{'}(S \cap T))$  if  $x \in S$  for  $x$ 
proof (cases  $x = a$ )
  case True with that span_0 eq_add_iff image_def mem_Collect_eq show
?thesis
    by blast
next
case False
define  $k$  where  $k = \min (1/2) (e / \text{norm } (x-a))$ 
have  $k: 0 < k \ k < 1$ 
  using  $\langle e > 0 \rangle$  False by (auto simp: k_def)
then have  $xa: (x-a) = \text{inverse } k \ *_R \ k \ *_R (x-a)$ 
  by simp
have  $e / \text{norm } (x - a) \geq k$ 
  using k_def by linarith
then have  $a + k \ *_R (x - a) \in \text{cball } a \ e$ 
  using  $\langle 0 < k \rangle$  False
  by (simp add: dist_norm) (simp add: field_simps)
then have  $T: a + k \ *_R (x - a) \in T$ 
  using  $e$  by blast
have  $S: a + k \ *_R (x - a) \in S$ 
  using  $k \ \langle a \in S \rangle$  convexD [OF  $\langle \text{convex } S \rangle \ \langle a \in S \rangle \ \langle x \in S \rangle$ , of  $1-k \ k$ ]
  by (simp add: algebra_simps)
have  $\text{inverse } k \ *_R \ k \ *_R (x-a) \in \text{span } ((\lambda x. x - a) \ \text{'}(S \cap T))$ 
  by (intro span_mul [OF span_base] image_eqI [where  $x = a + k \ *_R (x -$ 
 $a)$ ]) (auto simp: S T)
  with  $xa$  image_iff show ?thesis by fastforce
qed
have  $S \subseteq \text{affine hull } (S \cap T)$ 
  by (force simp: *  $\langle a \in S \rangle \ \langle a \in T \rangle$  hull_inc affine_hull_span_gen [of  $a$ ])
then show  $\text{affine hull } S \subseteq \text{affine hull } (S \cap T)$ 
  by (simp add: subset_hull)
qed

```

corollary *affine_hull_convex_Int_open*:

```

fixes  $S :: 'a::\text{real\_normed\_vector}$  set
assumes convex S open T S \cap T \neq \{\}
shows  $\text{affine hull } (S \cap T) = \text{affine hull } S$ 
using affine_hull_convex_Int_nonempty_interior assms interior_eq by blast

```

corollary *affine_hull_affine_Int_nonempty_interior*:

```

fixes  $S :: 'a::\text{real\_normed\_vector}$  set
assumes affine S S \cap interior T \neq \{\}
shows  $\text{affine hull } (S \cap T) = \text{affine hull } S$ 

```

by (simp add: affine_hull_convex_Int_nonempty_interior affine_imp_convex assms)

corollary *affine_hull_affine_Int_open*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes $\text{affine } S \text{ open } T \ S \cap T \neq \{\}$

shows $\text{affine hull } (S \cap T) = \text{affine hull } S$

by (simp add: affine_hull_convex_Int_open affine_imp_convex assms)

corollary *affine_hull_convex_Int_openin*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes $\text{convex } S \text{ openin } (\text{top_of_set } (\text{affine hull } S)) \ T \ S \cap T \neq \{\}$

shows $\text{affine hull } (S \cap T) = \text{affine hull } S$

using assms **unfolding** *openin_open*

by (metis affine_hull_convex_Int_open hull_subset inf.orderE inf_assoc)

corollary *affine_hull_openin*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes $\text{openin } (\text{top_of_set } (\text{affine hull } T)) \ S \ S \neq \{\}$

shows $\text{affine hull } S = \text{affine hull } T$

using assms **unfolding** *openin_open*

by (metis affine_affine_hull affine_hull_affine_Int_open hull_hull)

corollary *affine_hull_open*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes $\text{open } S \ S \neq \{\}$

shows $\text{affine hull } S = \text{UNIV}$

by (metis affine_hull_convex_Int_nonempty_interior assms convex_UNIV hull_UNIV inf_top.left_neutral interior_open)

lemma *aff_dim_convex_Int_nonempty_interior*:

fixes $S :: 'a::\text{euclidean_space_set}$

shows $\llbracket \text{convex } S; S \cap \text{interior } T \neq \{\} \rrbracket \implies \text{aff_dim}(S \cap T) = \text{aff_dim } S$

using *aff_dim_affine_hull2 affine_hull_convex_Int_nonempty_interior* by blast

lemma *aff_dim_convex_Int_open*:

fixes $S :: 'a::\text{euclidean_space_set}$

shows $\llbracket \text{convex } S; \text{open } T; S \cap T \neq \{\} \rrbracket \implies \text{aff_dim}(S \cap T) = \text{aff_dim } S$

using *aff_dim_convex_Int_nonempty_interior interior_eq* by blast

lemma *affine_hull_Diff*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes *ope*: $\text{openin } (\text{top_of_set } (\text{affine hull } S)) \ S$ and *finite* $F \ F \subset S$

shows $\text{affine hull } (S - F) = \text{affine hull } S$

proof –

have *clo*: $\text{closedin } (\text{top_of_set } S) \ F$

using assms *finite_imp_closedin* by auto

moreover have $S - F \neq \{\}$

using assms by auto

ultimately show *?thesis*
by (*metis ope closedin_def topspace_euclidean_subtopology affine_hull_openin openin_trans*)
qed

lemma *affine_hull_halfspace_lt*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{affine_hull } \{x. a \cdot x < r\} = (\text{if } a = 0 \wedge r \leq 0 \text{ then } \{\} \text{ else } \text{UNIV})$
using *halfspace_eq_empty_lt* [of a r]
by (*simp add: open_halfspace_lt affine_hull_open*)

lemma *affine_hull_halfspace_le*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{affine_hull } \{x. a \cdot x \leq r\} = (\text{if } a = 0 \wedge r < 0 \text{ then } \{\} \text{ else } \text{UNIV})$
proof (*cases a = 0*)
case *True* **then show** *?thesis* **by** *simp*
next
case *False*
then have $\text{affine_hull_closure } \{x. a \cdot x < r\} = \text{UNIV}$
using *affine_hull_halfspace_lt closure_same_affine_hull* **by** *fastforce*
moreover have $\{x. a \cdot x < r\} \subseteq \{x. a \cdot x \leq r\}$
by (*simp add: Collect_mono*)
ultimately show *?thesis* **using** *False antisym_conv hull_mono top_greatest*
by (*metis affine_hull_halfspace_lt*)
qed

lemma *affine_hull_halfspace_gt*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{affine_hull } \{x. a \cdot x > r\} = (\text{if } a = 0 \wedge r \geq 0 \text{ then } \{\} \text{ else } \text{UNIV})$
using *halfspace_eq_empty_gt* [of r a]
by (*simp add: open_halfspace_gt affine_hull_open*)

lemma *affine_hull_halfspace_ge*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{affine_hull } \{x. a \cdot x \geq r\} = (\text{if } a = 0 \wedge r > 0 \text{ then } \{\} \text{ else } \text{UNIV})$
using *affine_hull_halfspace_le* [of $-a$ $-r$] **by** *simp*

lemma *aff_dim_halfspace_lt*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{aff_dim } \{x. a \cdot x < r\} =$
 $(\text{if } a = 0 \wedge r \leq 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
by *simp* (*metis aff_dim_open halfspace_eq_empty_lt open_halfspace_lt*)

lemma *aff_dim_halfspace_le*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{aff_dim } \{x. a \cdot x \leq r\} =$
 $(\text{if } a = 0 \wedge r < 0 \text{ then } -1 \text{ else } \text{DIM}('a))$
proof –
have $\text{int } (\text{DIM}('a)) = \text{aff_dim } (\text{UNIV}::'a \text{ set})$

```

  by (simp)
  then have aff_dim (affine hull {x. a · x ≤ r}) = DIM('a) if (a = 0 → r ≥
0)
  using that by (simp add: affine_hull_halfspace_le not_less)
  then show ?thesis
  by (force)
qed

```

```

lemma aff_dim_halfspace_gt:
  fixes a :: 'a::euclidean_space
  shows aff_dim {x. a · x > r} =
    (if a = 0 ∧ r ≥ 0 then -1 else DIM('a))
by simp (metis aff_dim_open_halfspace_eq_empty_gt open_halfspace_gt)

```

```

lemma aff_dim_halfspace_ge:
  fixes a :: 'a::euclidean_space
  shows aff_dim {x. a · x ≥ r} =
    (if a = 0 ∧ r > 0 then -1 else DIM('a))
using aff_dim_halfspace_le [of -a -r] by simp

```

```

proposition aff_dim_eq_hyperplane:
  fixes S :: 'a::euclidean_space set
  shows aff_dim S = DIM('a) - 1 ↔ (∃ a b. a ≠ 0 ∧ affine hull S = {x. a · x
= b})
  (is ?lhs = ?rhs)
proof (cases S = {})
  case True then show ?thesis
  by (auto simp: dest: hyperplane_eq_Ex)
next
  case False
  then obtain c where c ∈ S by blast
  show ?thesis
  proof (cases c = 0)
    case True
    have ?lhs ↔ (∃ a. a ≠ 0 ∧ span ((λx. x - c) ' S) = {x. a · x = 0})
    by (simp add: aff_dim_eq_dim [of c] ⟨c ∈ S⟩ hull_inc dim_eq_hyperplane
del: One_nat_def)
    also have ... ↔ ?rhs
    using span_zero [of S] True ⟨c ∈ S⟩ affine_hull_span_0 hull_inc
    by (fastforce simp add: affine_hull_span_gen [of c] ⟨c = 0⟩)
    finally show ?thesis .
  next
  case False
  have xc_im: x ∈ (+) c ' {y. a · y = 0} if a · x = a · c for a x
  proof -
    have ∃ y. a · y = 0 ∧ c + y = x
    by (metis that add commute diff_add_cancel inner_commute inner_diff_left
right_minus_eq)
    then show x ∈ (+) c ' {y. a · y = 0}

```

```

    by blast
  qed
  have 2: span ((λx. x - c) ' S) = {x. a · x = 0}
    if (+) c ' span ((λx. x - c) ' S) = {x. a · x = b} for a b
  proof -
    have b = a · c
      using span_0 that by fastforce
    with that have (+) c ' span ((λx. x - c) ' S) = {x. a · x = a · c}
      by simp
    then have span ((λx. x - c) ' S) = (λx. x - c) ' {x. a · x = a · c}
      by (metis (no_types) image_cong translation_galois uminus_add_conv_diff)
    also have ... = {x. a · x = 0}
      by (force simp: inner_distrib inner_diff_right
        intro: image_eqI [where x=x+c for x])
    finally show ?thesis .
  qed
  have ?lhs = (∃ a. a ≠ 0 ∧ span ((λx. x - c) ' S) = {x. a · x = 0})
    by (simp add: aff_dim_eq_dim [of c] ⟨c ∈ S⟩ hull_inc dim_eq_hyperplane
  del: One_nat_def)
  also have ... = ?rhs
    by (fastforce simp add: affine_hull_span_gen [of c] ⟨c ∈ S⟩ hull_inc inner_distrib
  intro: xc_im intro!: 2)
  finally show ?thesis .
  qed
  qed

```

```

corollary aff_dim_hyperplane [simp]:
  fixes a :: 'a::euclidean_space
  shows a ≠ 0 ⇒ aff_dim {x. a · x = r} = DIM('a) - 1
  by (metis aff_dim_eq_hyperplane affine_hull_eq affine_hyperplane)

```

6.0.15 Some stepping theorems

```

lemma aff_dim_insert:
  fixes a :: 'a::euclidean_space
  shows aff_dim (insert a S) = (if a ∈ affine hull S then aff_dim S else aff_dim
  S + 1)
  proof (cases S = {})
    case True then show ?thesis
      by simp
  next
    case False
    then obtain x s' where S: S = insert x s' x ∉ s'
      by (meson Set.set_insert all_not_in_conv)
    show ?thesis using S
      by (force simp add: affine_hull_insert_span_gen span_zero insert_commute
  [of a] aff_dim_eq_dim [of x] dim_insert)
  qed

```



```

lemma affine_dependent_choose:
  fixes a :: 'a :: euclidean_space
  assumes ¬(affine_dependent S)
  shows affine_dependent(insert a S)  $\longleftrightarrow$  a  $\notin$  S  $\wedge$  a  $\in$  affine hull S
    (is ?lhs = ?rhs)
proof safe
  assume affine_dependent (insert a S) and a  $\in$  S
  then show False
    using  $\langle a \in S \rangle$  assms insert_absorb by fastforce
next
  assume lhs: affine_dependent (insert a S)
  then have a  $\notin$  S
    by (metis (no_types) assms insert_absorb)
  moreover have finite S
    using affine_independent_iff_card assms by blast
  moreover have aff_dim (insert a S)  $\neq$  int (card S)
    using  $\langle$ finite S $\rangle$  affine_independent_iff_card  $\langle a \notin S \rangle$  lhs by fastforce
  ultimately show a  $\in$  affine hull S
    by (metis aff_dim_affine_independent aff_dim_insert assms)
next
  assume a  $\notin$  S and a  $\in$  affine hull S
  show affine_dependent (insert a S)
    by (simp add:  $\langle a \in$  affine hull S $\rangle$   $\langle a \notin S \rangle$  affine_dependent_def)
qed

lemma affine_independent_insert:
  fixes a :: 'a :: euclidean_space
  shows  $\llbracket \neg$  affine_dependent S; a  $\notin$  affine hull S $\rrbracket \implies \neg$  affine_dependent(insert
a S)
  by (simp add: affine_dependent_choose)

lemma subspace_bounded_eq_trivial:
  fixes S :: 'a::real_normed_vector set
  assumes subspace S
  shows bounded S  $\longleftrightarrow$  S = {0}
proof -
  have False if bounded S x  $\in$  S x  $\neq$  0 for x
  proof -
    obtain B where B:  $\bigwedge y. y \in S \implies \text{norm } y < B \ B > 0$ 
      using  $\langle$ bounded S $\rangle$  by (force simp: bounded_pos_less)
    have (B / norm x)  $\cdot_R$  x  $\in$  S
      using assms subspace_mul  $\langle x \in S \rangle$  by auto
    moreover have norm ((B / norm x)  $\cdot_R$  x) = B
      using that B by (simp add: algebra_simps)
    ultimately show False using B by force
  qed
  then have bounded S  $\implies$  S = {0}
    using assms subspace_0 by fastforce
  then show ?thesis

```

by *blast*
qed

lemma *affine_bounded_eq_trivial*:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes *affine S*
shows $\text{bounded } S \longleftrightarrow S = \{\} \vee (\exists a. S = \{a\})$
proof (*cases S = {}*)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then obtain b **where** $b \in S$ **by** *blast*
with *False assms*
have $\text{bounded } S \implies S = \{b\}$
using *affine_diffs_subspace [OF assms ⟨b ∈ S⟩]*
by (*metis (no_types, lifting) ab_group_add_class.ab_left_minus bounded_translation image_empty image_insert subspace_bounded_eq_trivial translation_invert*)
then show *?thesis* **by** *auto*
qed

lemma *affine_bounded_eq_lowdim*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes *affine S*
shows $\text{bounded } S \longleftrightarrow \text{aff_dim } S \leq 0$
proof
show $\text{aff_dim } S \leq 0 \implies \text{bounded } S$
by (*metis aff_dim_singular aff_dim_subset affine_dim_equal affine_singular all_not_in_conv assms bounded_empty bounded_insert dual_order.antisym empty_subsetI insert_subset*)
qed (*use affine_bounded_eq_trivial assms in fastforce*)

lemma *bounded_hyperplane_eq_trivial_0*:
fixes $a :: 'a::\text{euclidean_space}$
assumes $a \neq 0$
shows $\text{bounded } \{x. a \cdot x = 0\} \longleftrightarrow \text{DIM}('a) = 1$
proof
assume $\text{bounded } \{x. a \cdot x = 0\}$
then have $\text{aff_dim } \{x. a \cdot x = 0\} \leq 0$
by (*simp add: affine_bounded_eq_lowdim affine_hyperplane*)
with *assms* **show** $\text{DIM}('a) = 1$
by (*simp add: le_Suc_eq*)
next
assume $\text{DIM}('a) = 1$
then show $\text{bounded } \{x. a \cdot x = 0\}$
by (*simp add: affine_bounded_eq_lowdim affine_hyperplane assms*)
qed

lemma *bounded_hyperplane_eq_trivial*:

```

fixes  $a :: 'a::euclidean\_space$ 
shows  $\text{bounded } \{x. a \cdot x = r\} \longleftrightarrow (\text{if } a = 0 \text{ then } r \neq 0 \text{ else } \text{DIM}('a) = 1)$ 
proof -
  { assume  $r \neq 0 \ a \neq 0$ 
    have  $\text{aff\_dim } \{x. y \cdot x = 0\} = \text{aff\_dim } \{x. a \cdot x = r\}$  if  $y \neq 0$  for  $y::'a$ 
      by (metis that  $\langle a \neq 0 \rangle$  aff_dim_hyperplane)
    then have  $\text{bounded } \{x. a \cdot x = r\} = (\text{DIM}('a) = \text{Suc } 0)$ 
    by (metis One_nat_def  $\langle a \neq 0 \rangle$  affine_bounded_eq_lowdim affine_hyperplane
      bounded_hyperplane_eq_trivial_0)
  }
  then show ?thesis
    by (auto simp: bounded_hyperplane_eq_trivial_0)
qed

```

6.0.16 General case without assuming closure and getting non-strict separation

proposition *separating_hyperplane_closed_point_inset:*

```

fixes  $S :: 'a::euclidean\_space$  set
assumes convex S closed S S  $\neq \{\}$   $z \notin S$ 
obtains  $a \ b$  where  $a \in S \ (a - z) \cdot z < b \ \wedge \ x \in S \implies b < (a - z) \cdot x$ 
proof -
  obtain  $y$  where  $y \in S$  and  $y: \bigwedge u. u \in S \implies \text{dist } z \ y \leq \text{dist } z \ u$ 
    using distance_attains_inf [of S z] assms by auto
  then have  $*$ :  $(y - z) \cdot z < (y - z) \cdot z + (\text{norm } (y - z))^2 / 2$ 
    using  $\langle y \in S \rangle \ \langle z \notin S \rangle$  by auto
  show ?thesis
proof (rule that [OF  $\langle y \in S \rangle *$ ])
    fix  $x$ 
    assume  $x \in S$ 
    have  $yz: 0 < (y - z) \cdot (y - z)$ 
      using  $\langle y \in S \rangle \ \langle z \notin S \rangle$  by auto
    { assume  $0: 0 < ((z - y) \cdot (x - y))$ 
      with any_closest_point_dot [OF  $\langle \text{convex } S \rangle \ \langle \text{closed } S \rangle$ ]
      have False
        using  $y \ \langle x \in S \rangle \ \langle y \in S \rangle$  not_less by blast
    }
    then have  $0 \leq ((y - z) \cdot (x - y))$ 
      by (force simp: not_less inner_diff_left)
    with  $yz$  have  $0 < 2 * ((y - z) \cdot (x - y)) + (y - z) \cdot (y - z)$ 
      by (simp add: algebra_simps)
    then show  $(y - z) \cdot z + (\text{norm } (y - z))^2 / 2 < (y - z) \cdot x$ 
      by (simp add: field_simps inner_diff_left inner_diff_right dot_square_norm
        [symmetric])
    qed
qed

```

lemma *separating_hyperplane_closed_0_inset:*

```

fixes  $S :: 'a::euclidean\_space$  set

```

assumes $\text{convex } S \text{ closed } S \neq \{\} \ 0 \notin S$
obtains $a \ b$ **where** $a \in S \ a \neq 0 \ 0 < b \ \wedge x. x \in S \implies a \cdot x > b$
using $\text{separating_hyperplane_closed_point_inset}$ [*OF assms*] **by** simp ($\text{metis } \langle 0 \notin S \rangle$)

proposition $\text{separating_hyperplane_set_0_in_span}$:

fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{convex } S \ S \neq \{\} \ 0 \notin S$
obtains a **where** $a \in \text{span } S \ a \neq 0 \ \wedge x. x \in S \implies 0 \leq a \cdot x$
proof –
define k **where** [*abs_def*]: $k \ c = \{x. 0 \leq c \cdot x\}$ **for** $c :: 'a$
have $\text{span } S \cap \text{frontier } (\text{cball } 0 \ 1) \cap \bigcap f' \neq \{\}$
if $f': \text{finite } f' \ f' \subseteq k \ 'S$ **for** f'
proof –
obtain C **where** $C \subseteq S$ $\text{finite } C$ **and** $C: f' = k \ 'C$
using $\text{finite_subset_image}$ [*OF f'*] **by** blast
obtain a **where** $a \in S \ a \neq 0$
using $\langle S \neq \{\} \ \langle 0 \notin S \rangle \ \text{ex_in_conv}$ **by** blast
then have $\text{norm } (a /_{\mathbb{R}} (\text{norm } a)) = 1$
by simp
moreover have $a /_{\mathbb{R}} (\text{norm } a) \in \text{span } S$
by ($\text{simp add: } \langle a \in S \rangle \ \text{span_scale } \text{span_base}$)
ultimately have $\text{ass: } a /_{\mathbb{R}} (\text{norm } a) \in \text{span } S \cap \text{sphere } 0 \ 1$
by simp
show *?thesis*
proof ($\text{cases } C = \{\}$)
case True **with** $C \ \text{ass}$ **show** *?thesis*
by auto
next
case False
have $\text{closed } (\text{convex hull } C)$
using $\langle \text{finite } C \rangle \ \text{compact_eq_bounded_closed } \text{finite_imp_compact_convex_hull}$
by auto
moreover have $\text{convex hull } C \neq \{\}$
by ($\text{simp add: } \text{False}$)
moreover have $0 \notin \text{convex hull } C$
by ($\text{metis } \langle C \subseteq S \rangle \ \langle \text{convex } S \rangle \ \langle 0 \notin S \rangle \ \text{convex_hull_subset_hull_same}$
 $\text{insert_absorb } \text{insert_subset}$)
ultimately obtain $a \ b$
where $a \in \text{convex hull } C \ a \neq 0 \ 0 < b$
and $\text{ab: } \wedge x. x \in \text{convex hull } C \implies a \cdot x > b$
using $\text{separating_hyperplane_closed_0_inset}$ **by** blast
then have $a \in S$
by ($\text{metis } \langle C \subseteq S \rangle \ \text{assms}(1) \ \text{subsetCE } \text{subset_hull}$)
moreover have $\text{norm } (a /_{\mathbb{R}} (\text{norm } a)) = 1$
using $\langle a \neq 0 \rangle$ **by** simp
moreover have $a /_{\mathbb{R}} (\text{norm } a) \in \text{span } S$
by ($\text{simp add: } \langle a \in S \rangle \ \text{span_scale } \text{span_base}$)

```

ultimately have ass:  $a /_R (\text{norm } a) \in \text{span } S \cap \text{sphere } 0 \ 1$ 
  by simp
have  $\bigwedge x. \llbracket a \neq 0; x \in C \rrbracket \implies 0 \leq x \cdot a$ 
  using  $ab \langle 0 < b \rangle$  by (metis hull_inc inner_commute less_eq_real_def
less_trans)
then have aa:  $a /_R (\text{norm } a) \in (\bigcap c \in C. \{x. 0 \leq c \cdot x\})$ 
  by (auto simp add: field_split_simps)
show ?thesis
  unfolding C k_def
  using ass aa Int_iff empty_iff by force
qed
qed
moreover have  $\bigwedge T. T \in k \text{ ' } S \implies \text{closed } T$ 
  using closed_halfspace_ge k_def by blast
ultimately have  $(\text{span } S \cap \text{frontier}(\text{cball } 0 \ 1)) \cap (\bigcap (k \text{ ' } S)) \neq \{\}$ 
  by (metis compact_imp_fip closed_Int_compact closed_span compact_cball
compact_frontier)
then show ?thesis
  unfolding set_eq_iff k_def
  by simp (metis inner_commute norm_eq_zero that zero_neq_one)
qed

```

lemma separating_hyperplane_set_point_inaff:

```

fixes S :: 'a::euclidean_space set
assumes convex S S  $\neq \{\}$  and zno:  $z \notin S$ 
obtains a b where  $(z + a) \in \text{affine hull } (\text{insert } z \ S)$ 
  and  $a \neq 0$  and  $a \cdot z \leq b$ 
  and  $\bigwedge x. x \in S \implies a \cdot x \geq b$ 
proof -
from separating_hyperplane_set_0_inspan [of image  $(\lambda x. -z + x) \ S$ ]
have convex  $((+) (-z) \text{ ' } S)$ 
  using  $\langle \text{convex } S \rangle$  by simp
moreover have  $(+) (-z) \text{ ' } S \neq \{\}$ 
  by (simp add:  $\langle S \neq \{\} \rangle$ )
moreover have  $0 \notin (+) (-z) \text{ ' } S$ 
  using zno by auto
ultimately obtain a where  $a \in \text{span } ((+) (-z) \text{ ' } S)$   $a \neq 0$ 
  and a:  $\bigwedge x. x \in ((+) (-z) \text{ ' } S) \implies 0 \leq a \cdot x$ 
  using separating_hyperplane_set_0_inspan [of image  $(\lambda x. -z + x) \ S$ ]
  by blast
then have szx:  $\bigwedge x. x \in S \implies a \cdot z \leq a \cdot x$ 
  by (metis (no_types, lifting) imageI inner_minus_right inner_right_distrib
minus_add neg_le_0_iff_le neg_le_iff_le real_add_le_0_iff)
moreover
have  $z + a \in \text{affine hull } (\text{insert } z \ S)$ 
  using  $\langle a \in \text{span } ((+) (-z) \text{ ' } S) \rangle$  affine_hull_insert_span_gen by blast
ultimately show ?thesis
  using  $\langle a \neq 0 \rangle$  szx that by auto

```

qed

proposition *supporting_hyperplane_rel_boundary:*

fixes $S :: 'a::\text{euclidean_space}$ set
assumes $\text{convex } S$ $x \in S$ **and** $x \notin \text{rel_interior } S$
obtains a **where** $a \neq 0$
and $\bigwedge y. y \in S \implies a \cdot x \leq a \cdot y$
and $\bigwedge y. y \in \text{rel_interior } S \implies a \cdot x < a \cdot y$

proof –

obtain a b **where** $\text{aff}: (x + a) \in \text{affine hull } (\text{insert } x (\text{rel_interior } S))$
and $a \neq 0$ **and** $a \cdot x \leq b$
and $\text{ageb}: \bigwedge u. u \in (\text{rel_interior } S) \implies a \cdot u \geq b$
using *separating_hyperplane_set_point_inaff* [of $\text{rel_interior } S$ x] *assms*
by (*auto simp: rel_interior_eq_empty convex_rel_interior*)
have $\text{le_ay}: a \cdot x \leq a \cdot y$ **if** $y \in S$ **for** y

proof –

have $\text{con}: \text{continuous_on } (\text{closure } (\text{rel_interior } S)) ((\cdot) a)$
by (*rule continuous_intros continuous_on_subset | blast*)
have $y: y \in \text{closure } (\text{rel_interior } S)$
using $\langle \text{convex } S \rangle$ *closure_def convex_closure_rel_interior* $\langle y \in S \rangle$
by *fastforce*
show *?thesis*
using *continuous_ge_on_closure* [OF $\text{con } y$] ageb $\langle a \cdot x \leq b \rangle$
by *fastforce*

qed

have $\exists: a \cdot x < a \cdot y$ **if** $y \in \text{rel_interior } S$ **for** y

proof –

obtain e **where** $0 < e$ $y \in S$ **and** $e: \text{cball } y e \cap \text{affine hull } S \subseteq S$
using $\langle y \in \text{rel_interior } S \rangle$ **by** (*force simp: rel_interior_cball*)
define y' **where** $y' = y - (e / \text{norm } a) *_{\mathbb{R}} ((x + a) - x)$
have $y' \in \text{cball } y e$
unfolding y'_def **using** $\langle 0 < e \rangle$ **by** *force*
moreover **have** $y' \in \text{affine hull } S$
unfolding y'_def
by (*metis* $\langle x \in S \rangle \langle y \in S \rangle \langle \text{convex } S \rangle$ *aff_affine_affine_hull hull_redundant*
rel_interior_same_affine_hull hull_inc mem_affine_3_minus2)
ultimately **have** $y' \in S$
using e **by** *auto*
have $a \cdot x \leq a \cdot y$
using le_ay $\langle a \neq 0 \rangle \langle y \in S \rangle$ **by** *blast*
moreover **have** $a \cdot x \neq a \cdot y$
using le_ay [OF $\langle y' \in S \rangle$] $\langle a \neq 0 \rangle \langle 0 < e \rangle$ *not_le*
by (*fastforce simp add: y'_def inner_diff dot_square_norm power2_eq_square*)
ultimately **show** *?thesis* **by** *force*

qed

show *?thesis*

by (*rule that* [OF $\langle a \neq 0 \rangle \text{le_ay } \exists$])

qed

```

lemma supporting_hyperplane_relative_frontier:
  fixes S :: 'a::euclidean_space set
  assumes convex S x ∈ closure S x ∉ rel_interior S
  obtains a where a ≠ 0
    and  $\bigwedge y. y \in \text{closure } S \implies a \cdot x \leq a \cdot y$ 
    and  $\bigwedge y. y \in \text{rel\_interior } S \implies a \cdot x < a \cdot y$ 
using supporting_hyperplane_rel_boundary [of closure S x]
by (metis assms convex_closure convex_rel_interior_closure)

```

6.0.17 Some results on decomposing convex hulls: intersections, simplicial subdivision

```

lemma
  fixes S :: 'a::euclidean_space set
  assumes  $\neg \text{affine\_dependent}(S \cup T)$ 
  shows convex_hull_Int_subset:  $\text{convex hull } S \cap \text{convex hull } T \subseteq \text{convex hull } (S \cap T)$  (is ?C)
    and affine_hull_Int_subset:  $\text{affine hull } S \cap \text{affine hull } T \subseteq \text{affine hull } (S \cap T)$  (is ?A)
proof -
  have [simp]: finite S finite T
  using aff_independent_finite assms by blast+
  have sum_u (S ∩ T) = 1 ∧
    ( $\sum v \in S \cap T. u \ v \ *_R \ v$ ) = ( $\sum v \in S. u \ v \ *_R \ v$ )
  if [simp]: sum_u S = 1
    sum_v T = 1
    and eq: ( $\sum x \in T. v \ x \ *_R \ x$ ) = ( $\sum x \in S. u \ x \ *_R \ x$ ) for u v
proof -
  define f where  $f \ x = (\text{if } x \in S \text{ then } u \ x \ \text{else } 0) - (\text{if } x \in T \text{ then } v \ x \ \text{else } 0)$ 
for x
  have sum_f (S ∪ T) = 0
  by (simp add: f_def sum_Un sum_subtractf flip: sum.inter_restrict)
  moreover have ( $\sum x \in (S \cup T). f \ x \ *_R \ x$ ) = 0
  by (simp add: eq f_def sum_Un scaleR_left_diff_distrib sum_subtractf
if_smult_flip: sum.inter_restrict cong: if_cong)
  ultimately have  $\bigwedge v. v \in S \cup T \implies f \ v = 0$ 
  using aff_independent_finite assms unfolding affine_dependent_explicit
  by blast
  then have u [simp]:  $\bigwedge x. x \in S \implies u \ x = (\text{if } x \in T \text{ then } v \ x \ \text{else } 0)$ 
  by (simp add: f_def) presburger
  have sum_u (S ∩ T) = sum_u S
  by (simp add: sum.inter_restrict)
  then have sum_u (S ∩ T) = 1
  using that by linarith
  moreover have ( $\sum v \in S \cap T. u \ v \ *_R \ v$ ) = ( $\sum v \in S. u \ v \ *_R \ v$ )
  by (auto simp: if_smult sum.inter_restrict intro: sum.cong)
  ultimately show ?thesis
  by force
qed

```

```

    then show ?A ?C
      by (auto simp: convex_hull_finite affine_hull_finite)
qed

```

```

proposition affine_hull_Int:
  fixes S :: 'a::euclidean_space set
  assumes  $\neg$  affine_dependent (S  $\cup$  T)
  shows affine_hull (S  $\cap$  T) = affine_hull S  $\cap$  affine_hull T
  by (simp add: affine_hull_Int_subset assms hull_mono subset_antisym)

```

```

proposition convex_hull_Int:
  fixes S :: 'a::euclidean_space set
  assumes  $\neg$  affine_dependent (S  $\cup$  T)
  shows convex_hull (S  $\cap$  T) = convex_hull S  $\cap$  convex_hull T
  by (simp add: convex_hull_Int_subset assms hull_mono subset_antisym)

```

```

proposition
  fixes S :: 'a::euclidean_space set set
  assumes  $\neg$  affine_dependent ( $\bigcup$  S)
  shows affine_hull_Inter: affine_hull ( $\bigcap$  S) = ( $\bigcap$  T $\in$ S. affine_hull T) (is ?A)
  and convex_hull_Inter: convex_hull ( $\bigcap$  S) = ( $\bigcap$  T $\in$ S. convex_hull T) (is ?C)

```

```

proof -
  have finite S
    using aff_independent_finite assms finite_UnionD by blast
  then have ?A  $\wedge$  ?C using assms
  proof (induction S rule: finite_induct)
    case empty then show ?case by auto
  next
    case (insert T F)
    then show ?case
  proof (cases F={})
    case True then show ?thesis by simp
  next
    case False
    with insert.prem1 have [simp]:  $\neg$  affine_dependent (T  $\cup$   $\bigcap$  F)
      by (auto intro: affine_dependent_subset)
    have [simp]:  $\neg$  affine_dependent ( $\bigcup$  F)
      using affine_independent_subset insert.prem1 by fastforce
    show ?thesis
      by (simp add: affine_hull_Int convex_hull_Int insert.IH)
  qed
  qed
  then show ?A ?C
    by auto
  qed

```

```

proposition in_convex_hull_exchange_unique:
  fixes S :: 'a::euclidean_space set

```



```

assumes naff:  $\neg$  affine_dependent S and a:  $a \in \text{convex\_hull } S$ 
  and S:  $T \subseteq S \ T' \subseteq S$ 
  and x:  $x \in \text{convex\_hull } (\text{insert } a \ T)$ 
  and x':  $x \in \text{convex\_hull } (\text{insert } a \ T')$ 
  shows  $x \in \text{convex\_hull } (\text{insert } a \ (T \cap T'))$ 
proof (cases  $a \in S$ )
  case True
  then have  $\neg$  affine_dependent (insert a  $T \cup \text{insert } a \ T'$ )
    using affine_dependent_subset assms by auto
  then have  $x \in \text{convex\_hull } (\text{insert } a \ T \cap \text{insert } a \ T')$ 
    by (metis IntI convex_hull_Int x x')
  then show ?thesis
    by simp
  next
  case False
  then have anot:  $a \notin T \ a \notin T'$ 
    using assms by auto
  have [simp]: finite S
    by (simp add: aff_independent_finite assms)
  then obtain b where b0:  $\bigwedge s. s \in S \implies 0 \leq b \ s$ 
    and b1:  $\text{sum } b \ S = 1$  and aeq:  $a = (\sum s \in S. b \ s \ *_R \ s)$ 
    using a by (auto simp: convex_hull_finite)
  have fin [simp]: finite T finite T'
    using assms infinite_super  $\langle \text{finite } S \rangle$  by blast+
  then obtain c c' where c0:  $\bigwedge t. t \in \text{insert } a \ T \implies 0 \leq c \ t$ 
    and c1:  $\text{sum } c \ (\text{insert } a \ T) = 1$ 
    and xeq:  $x = (\sum t \in \text{insert } a \ T. c \ t \ *_R \ t)$ 
    and c'0:  $\bigwedge t. t \in \text{insert } a \ T' \implies 0 \leq c' \ t$ 
    and c'1:  $\text{sum } c' \ (\text{insert } a \ T') = 1$ 
    and x'eq:  $x = (\sum t \in \text{insert } a \ T'. c' \ t \ *_R \ t)$ 
    using x x' by (auto simp: convex_hull_finite)
  with fin anot
  have sumTT':  $\text{sum } c \ T = 1 - c \ a \ \text{sum } c' \ T' = 1 - c' \ a$ 
    and wsumT:  $(\sum t \in T. c \ t \ *_R \ t) = x - c \ a \ *_R \ a$ 
    by simp_all
  have wsumT':  $(\sum t \in T'. c' \ t \ *_R \ t) = x - c' \ a \ *_R \ a$ 
    using x'eq fin anot by simp
  define cc where cc  $\equiv \lambda x. \text{if } x \in T \text{ then } c \ x \ \text{else } 0$ 
  define cc' where cc'  $\equiv \lambda x. \text{if } x \in T' \text{ then } c' \ x \ \text{else } 0$ 
  define dd where dd  $\equiv \lambda x. cc \ x - cc' \ x + (c \ a - c' \ a) \ *_R \ x$ 
  have sumSS':  $\text{sum } cc \ S = 1 - c \ a \ \text{sum } cc' \ S = 1 - c' \ a$ 
    unfolding cc_def cc'_def using S
    by (simp_all add: Int_absorb1 Int_absorb2 sum_subtractf sum.inter_restrict
    [symmetric] sumTT')
  have wsumSS:  $(\sum t \in S. cc \ t \ *_R \ t) = x - c \ a \ *_R \ a \ (\sum t \in S. cc' \ t \ *_R \ t) = x - c' \ a \ *_R \ a$ 
    unfolding cc_def cc'_def using S
    by (simp_all add: Int_absorb1 Int_absorb2 if_smult sum.inter_restrict [symmetric]
    wsumT wsumT' cong: if_cong)

```

```

have sum_dd0: sum dd S = 0
  unfolding dd_def using S
  by (simp add: sumSS' comm_monoid_add_class.sum.distrib sum_subtractf
      algebra_simps sum_distrib_right [symmetric] b1)
have (∑ v∈S. (b v * x) *R v) = x *R (∑ v∈S. b v *R v) for x
  by (simp add: pth_5 real_vector.scale_sum_right mult.commute)
then have *: (∑ v∈S. (b v * x) *R v) = x *R a for x
  using aeq by blast
have (∑ v ∈ S. dd v *R v) = 0
  unfolding dd_def using S
  by (simp add: * wsumSS sum.distrib sum_subtractf algebra_simps)
then have dd0: dd v = 0 if v ∈ S for v
  using naff [unfolded affine_dependent_explicit_not_ex, rule_format, of S dd]
  using that sum_dd0 by force
consider c' a ≤ c a | c a ≤ c' a by linarith
then show ?thesis
proof cases
case 1
then have sum_cc S ≤ sum cc' S
  by (simp add: sumSS')
then have le: cc x ≤ cc' x if x ∈ S for x
  using dd0 [OF that] 1 b0 mult_left_mono that
  by (fastforce simp add: dd_def algebra_simps)
have cc0: cc x = 0 if x ∈ S x ∉ T ∩ T' for x
  using le [OF ⟨x ∈ S⟩] that c0
  by (force simp: cc_def cc'_def split: if_split_asm)
have ge0: ∀ x∈T ∩ T'. 0 ≤ (cc(a := c a)) x
  by (simp add: c0 cc_def)
have sum (cc(a := c a)) (insert a (T ∩ T')) = c a + sum (cc(a := c a)) (T
∩ T')
  by (simp add: anot)
also have ... = c a + sum (cc(a := c a)) S
  using ⟨T ⊆ S⟩ False cc0 cc_def ⟨a ∉ S⟩ by (fastforce intro!: sum.mono_neutral_left
split: if_split_asm)
also have ... = c a + (1 - c a)
  by (metis ⟨a ∉ S⟩ fun_upd_other sum.cong sumSS'(1))
finally have 1: sum (cc(a := c a)) (insert a (T ∩ T')) = 1
  by simp
have (∑ x∈insert a (T ∩ T'). (cc(a := c a)) x *R x) = c a *R a + (∑ x ∈ T
∩ T'. (cc(a := c a)) x *R x)
  by (simp add: anot)
also have ... = c a *R a + (∑ x ∈ S. (cc(a := c a)) x *R x)
  using ⟨T ⊆ S⟩ False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
also have ... = c a *R a + x - c a *R a
  by (simp add: wsumSS ⟨a ∉ S⟩ if_smult sum_delta_notmem)
finally have self: (∑ x∈insert a (T ∩ T'). (cc(a := c a)) x *R x) = x
  by simp
show ?thesis

```

```

    by (force simp: convex_hull_finite c0 intro!: ge0 1 self exI [where x = cc(a
:= c a)])
  next
    case 2
    then have sum cc' S ≤ sum cc S
      by (simp add: sumSS')
    then have le: cc' x ≤ cc x if x ∈ S for x
      using dd0 [OF that] 2 b0 mult_left_mono that
      by (fastforce simp add: dd_def algebra_simps)
    have cc0: cc' x = 0 if x ∈ S x ∉ T ∩ T' for x
      using le [OF ⟨x ∈ S⟩] that c'0
      by (force simp: cc_def cc'_def split: if_split_asm)
    have ge0: ∀ x ∈ T ∩ T'. 0 ≤ (cc'(a := c' a)) x
      by (simp add: c'0 cc'_def)
    have sum (cc'(a := c' a)) (insert a (T ∩ T')) = c' a + sum (cc'(a := c' a))
(T ∩ T')
      by (simp add: anot)
    also have ... = c' a + sum (cc'(a := c' a)) S
      using ⟨T ⊆ S⟩ False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
    also have ... = c' a + (1 - c' a)
      by (metis ⟨a ∉ S⟩ fun_upd_other sum.cong sumSS')
    finally have 1: sum (cc'(a := c' a)) (insert a (T ∩ T')) = 1
      by simp
    have (∑ x ∈ insert a (T ∩ T'). (cc'(a := c' a)) x *R x) = c' a *R a + (∑ x ∈
T ∩ T'. (cc'(a := c' a)) x *R x)
      by (simp add: anot)
    also have ... = c' a *R a + (∑ x ∈ S. (cc'(a := c' a)) x *R x)
      using ⟨T ⊆ S⟩ False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
    also have ... = c a *R a + x - c a *R a
      by (simp add: wsumSS ⟨a ∉ S⟩ if_smult sum_delta_notmem)
    finally have self: (∑ x ∈ insert a (T ∩ T'). (cc'(a := c' a)) x *R x) = x
      by simp
    show ?thesis
      by (force simp: convex_hull_finite c'0 intro!: ge0 1 self exI [where x = cc'(a
:= c' a)])
  qed
qed

```

corollary *convex_hull_exchange_Int:*

fixes $a :: 'a::\text{euclidean_space}$

assumes $\neg \text{affine_dependent } S \ a \in \text{convex_hull } S \ T \subseteq S \ T' \subseteq S$

shows $(\text{convex_hull } (\text{insert } a \ T)) \cap (\text{convex_hull } (\text{insert } a \ T')) =$
 $\text{convex_hull } (\text{insert } a \ (T \cap T'))$ **(is ?lhs = ?rhs)**

proof (rule subset_antisym)

show $?lhs \subseteq ?rhs$

using *in_convex_hull_exchange_unique* **assms** **by** *blast*

show $?rhs \subseteq ?lhs$

by (metis hull_mono inf_le1 inf_le2 insert_inter_insert le_inf_iff)
qed

lemma Int_closed_segment:

fixes $b :: 'a::\text{euclidean_space}$

assumes $b \in \text{closed_segment } a \ c \vee \neg \text{collinear}\{a,b,c\}$

shows $\text{closed_segment } a \ b \cap \text{closed_segment } b \ c = \{b\}$

proof (cases $c = a$)

case True

then show ?thesis

using assms collinear_3_eq_affine_dependent by fastforce

next

case False

from assms show ?thesis

proof

assume $b \in \text{closed_segment } a \ c$

moreover have $\neg \text{affine_dependent } \{a, c\}$

by (simp)

ultimately show ?thesis

using False convex_hull_exchange_Int [of $\{a,c\}$ b $\{a\}$ $\{c\}$]

by (simp add: segment_convex_hull_insert_commute)

next

assume $\text{ncoll}: \neg \text{collinear } \{a, b, c\}$

have False if $\text{closed_segment } a \ b \cap \text{closed_segment } b \ c \neq \{b\}$

proof -

have $b \in \text{closed_segment } a \ b$ and $b \in \text{closed_segment } b \ c$

by auto

with that obtain d where $b \neq d$ $d \in \text{closed_segment } a \ b$ $d \in \text{closed_segment}$

$b \ c$

by force

then have $d: \text{collinear } \{a, d, b\}$ $\text{collinear } \{b, d, c\}$

by (auto simp: between_mem_segment between_imp_collinear)

have $\text{collinear } \{a, b, c\}$

by (metis (full_types) $\langle b \neq d \rangle$ collinear_3_trans d insert_commute)

with ncoll show False ..

qed

then show ?thesis

by blast

qed

qed

lemma affine_hull_finite_intersection_hyperplanes:

fixes $S :: 'a::\text{euclidean_space}$ set

obtains \mathcal{F} where

finite \mathcal{F}

$\text{of_nat } (\text{card } \mathcal{F}) + \text{aff_dim } S = \text{DIM}('a)$

$\text{affine hull } S = \bigcap \mathcal{F}$

$\bigwedge h. h \in \mathcal{F} \implies \exists a \ b. a \neq 0 \wedge h = \{x. a \cdot x = b\}$

proof -

```

obtain  $b$  where  $b \subseteq S$ 
  and  $indb: \neg \text{affine\_dependent } b$ 
  and  $eq: \text{affine\_hull } S = \text{affine\_hull } b$ 
  using  $\text{affine\_basis\_exists}$  by  $\text{blast}$ 
obtain  $c$  where  $indc: \neg \text{affine\_dependent } c$  and  $b \subseteq c$ 
  and  $affc: \text{affine\_hull } c = \text{UNIV}$ 
  by ( $\text{metis extend\_to\_affine\_basis affine\_UNIV hull\_same indb subset\_UNIV}$ )
then have  $\text{finite } c$ 
  by ( $\text{simp add: aff\_independent\_finite}$ )
then have  $fbc: \text{finite } b \text{ card } b \leq \text{card } c$ 
  using  $\langle b \subseteq c \rangle \text{infinite\_super}$  by ( $\text{auto simp: card\_mono}$ )
have  $imeq: (\lambda x. \text{affine\_hull } x) \text{ ' } ((\lambda a. c - \{a\}) \text{ ' } (c - b)) = ((\lambda a. \text{affine\_hull } (c - \{a\})) \text{ ' } (c - b))$ 
by  $\text{blast}$ 
have  $\text{card\_cb}: (\text{card } (c - b)) + \text{aff\_dim } S = \text{DIM}('a)$ 
proof -
  have  $\text{aff}: \text{aff\_dim } (\text{UNIV}::'a \text{ set}) = \text{aff\_dim } c$ 
  by ( $\text{metis aff\_dim\_affine\_hull affc}$ )
  have  $\text{aff\_dim } b = \text{aff\_dim } S$ 
  by ( $\text{metis (no\_types) aff\_dim\_affine\_hull eq}$ )
  then have  $\text{int } (\text{card } b) = 1 + \text{aff\_dim } S$ 
  by ( $\text{simp add: aff\_dim\_affine\_independent indb}$ )
  then show  $?thesis$ 
  using  $fbc \text{ aff}$ 
  by ( $\text{simp add: } \langle \neg \text{affine\_dependent } c \rangle \langle b \subseteq c \rangle \text{aff\_dim\_affine\_independent}$ 
 $\text{card\_Diff\_subset\_of\_nat\_diff}$ )
qed
show  $?thesis$ 
proof ( $\text{cases } c = b$ )
  case  $\text{True}$  show  $?thesis$ 
  proof
    show  $\text{int } (\text{card } \{\}) + \text{aff\_dim } S = \text{int } \text{DIM}('a)$ 
    using  $\text{True card\_cb}$  by  $\text{auto}$ 
    show  $\text{affine\_hull } S = \bigcap \{\}$ 
    using  $\text{True affc eq}$  by  $\text{blast}$ 
  qed  $\text{auto}$ 
next
  case  $\text{False}$ 
  have  $ind: \neg \text{affine\_dependent } (\bigcup a \in c - b. c - \{a\})$ 
  by ( $\text{rule affine\_independent\_subset [OF indc]} \text{ auto}$ )
  have  $*$ :  $1 + \text{aff\_dim } (c - \{t\}) = \text{int } (\text{DIM}('a))$  if  $t: t \in c$  for  $t$ 
  proof -
    have  $\text{insert } t \ c = c$ 
    using  $t$  by  $\text{blast}$ 
    then show  $?thesis$ 
    by ( $\text{metis (full\_types) add.commute aff\_dim\_affine\_hull aff\_dim\_insert}$ 
 $\text{aff\_dim\_UNIV affc affine\_dependent\_def indc insert\_Diff\_single } t$ )
  qed
  let  $?F = (\lambda x. \text{affine\_hull } x) \text{ ' } ((\lambda a. c - \{a\}) \text{ ' } (c - b))$ 

```

```

show ?thesis
proof
  have card ((λa. affine hull (c - {a})) ' (c - b)) = card (c - b)
  proof (rule card_image)
    show inj_on (λa. affine hull (c - {a})) (c - b)
    unfolding inj_on_def
    by (metis Diff_eq_empty_iff Diff_iff indc affine_dependent_def hull_subset
insert_iff)
  qed
  then show int (card ?F) + aff_dim S = int DIM('a)
    by (simp add: imeq card_cb)
  show affine hull S = ⋂ ?F
    by (metis Diff_eq_empty_iff INT_simps(4) UN_singleton order_refl ⟨b ⊆
c⟩ False eq_double_diff affine_hull_Inter [OF ind])
  have ∧a. [a ∈ c; a ∉ b] ⇒ aff_dim (c - {a}) = int (DIM('a) - Suc 0)
    by (metis * DIM_ge_Suc0 One_nat_def add_diff_cancel_left' int_ops(2)
of_nat_diff)
  then show ∧h. h ∈ ?F ⇒ ∃ a b. a ≠ 0 ∧ h = {x. a · x = b}
    by (auto simp only: One_nat_def aff_dim_eq_hyperplane [symmetric])
  qed (use ⟨finite c⟩ in auto)
qed
qed

lemma affine_hyperplane_sums_eq_UNIV_0:
  fixes S :: 'a :: euclidean_space set
  assumes affine S
  and 0 ∈ S and w ∈ S
  and a · w ≠ 0
  shows {x + y | x y. x ∈ S ∧ a · y = 0} = UNIV
proof -
  have subspace S
    by (simp add: assms subspace_affine)
  have span1: span {y. a · y = 0} ⊆ span {x + y | x y. x ∈ S ∧ a · y = 0}
    using ⟨0 ∈ S⟩ add.left_neutral by (intro span_mono) force
  have w ∉ span {y. a · y = 0}
    using ⟨a · w ≠ 0⟩ span_induct subspace_hyperplane by auto
  moreover have w ∈ span {x + y | x y. x ∈ S ∧ a · y = 0}
    using ⟨w ∈ S⟩
    by (metis (mono_tags, lifting) inner_zero_right mem_Collect_eq pth_d span_base)
  ultimately have span2: span {y. a · y = 0} ≠ span {x + y | x y. x ∈ S ∧ a ·
y = 0}
    by blast
  have a ≠ 0 using assms inner_zero_left by blast
  then have DIM('a) - 1 = dim {y. a · y = 0}
    by (simp add: dim_hyperplane)
  also have ... < dim {x + y | x y. x ∈ S ∧ a · y = 0}
    using span1 span2 by (blast intro: dim_subset)
  finally have DIM('a) - 1 < dim {x + y | x y. x ∈ S ∧ a · y = 0} .
  then have DD: dim {x + y | x y. x ∈ S ∧ a · y = 0} = DIM('a)

```

using *antisym dim_subset_UNIV lowdim_subset_hyperplane not_le* by *fast-force*

have *subs*: *subspace* $\{x + y \mid x \ y. \ x \in S \wedge a \cdot y = 0\}$
 using *subspace_sums* [*OF* \langle *subspace S* \rangle *subspace_hyperplane*] by *simp*
 moreover have *span* $\{x + y \mid x \ y. \ x \in S \wedge a \cdot y = 0\} = UNIV$
 using *DD dim_eq_full* by *blast*
 ultimately show *?thesis*
 by (*simp add: subs*) (*metis (lifting) span_eq_iff subs*)

qed

proposition *affine_hyperplane_sums_eq_UNIV*:

fixes *S* :: 'a :: *euclidean_space set*
 assumes *affine S*
 and $S \cap \{v. \ a \cdot v = b\} \neq \{\}$
 and $S - \{v. \ a \cdot v = b\} \neq \{\}$
 shows $\{x + y \mid x \ y. \ x \in S \wedge a \cdot y = b\} = UNIV$

proof (*cases a = 0*)

case *True* with *assms* show *?thesis*
 by (*auto simp: if_splits*)

next

case *False*

obtain *c* where $c \in S$ and $c: \ a \cdot c = b$

using *assms* by *force*

with *affine_diffs_subspace* [*OF* \langle *affine S* \rangle]

have *subspace* $((+) \ (- \ c) \ 'S)$ by *blast*

then have *aff*: *affine* $((+) \ (- \ c) \ 'S)$

by (*simp add: subspace_imp_affine*)

have *0*: $0 \in (+) \ (- \ c) \ 'S$

by (*simp add: c*)

obtain *d* where $d \in S$ and $a \cdot d \neq b$ and *dc*: $d - c \in (+) \ (- \ c) \ 'S$

using *assms* by *auto*

then have *adc*: $a \cdot (d - c) \neq 0$

by (*simp add: c inner_diff_right*)

define *U* where $U \equiv \{x + y \mid x \ y. \ x \in (+) \ (- \ c) \ 'S \wedge a \cdot y = 0\}$

have $u + v \in (+) \ (c + c) \ 'U$

if $u \in S$ $b = a \cdot v$ for $u \ v$

proof

show $u + v = c + c + (u + v - c - c)$

by (*simp add: algebra_simps*)

have $\exists x \ y. \ u + v - c - c = x + y \wedge (\exists xa \in S. \ x = xa - c) \wedge a \cdot y = 0$

proof (*intro exI conjI*)

show $u + v - c - c = (u - c) + (v - c) \wedge a \cdot (v - c) = 0$

by (*simp_all add: algebra_simps c that*)

qed (*use that in auto*)

then show $u + v - c - c \in U$

by (*auto simp: U_def image_def*)

qed

moreover have $\llbracket a \cdot v = 0; \ u \in S \rrbracket$

$\implies \exists x \ ya. \ v + (u + c) = x + ya \wedge x \in S \wedge a \cdot ya = b$ for $v \ u$

by (*metis add.left commute c inner_right distrib pth d*)
ultimately have $\{x + y \mid x y. x \in S \wedge a \cdot y = b\} = (+) (c+c) \text{ ' } U$
 by (*fastforce simp: algebra_simps U_def*)
also have $\dots = \text{range } ((+) (c + c))$
 by (*simp only: U_def affine_hyperplane_sums_eq_UNIV_0 [OF aff 0 dc adc]*)
also have $\dots = UNIV$
 by *simp*
finally show *?thesis* .
qed

lemma *aff_dim_sums_Int_0*:
assumes *affine S*
and *affine T*
and $0 \in S \ 0 \in T$
shows $\text{aff_dim } \{x + y \mid x y. x \in S \wedge y \in T\} = (\text{aff_dim } S + \text{aff_dim } T) - \text{aff_dim}(S \cap T)$
proof –
have $0 \in \{x + y \mid x y. x \in S \wedge y \in T\}$
using *assms by force*
then have $0: 0 \in \text{affine hull } \{x + y \mid x y. x \in S \wedge y \in T\}$
by (*metis (lifting) hull_inc*)
have *sub: subspace S subspace T*
using *assms by (auto simp: subspace_affine)*
show *?thesis*
using *dim_sums_Int [OF sub] by (simp add: aff_dim_zero assms 0 hull_inc)*
qed

proposition *aff_dim_sums_Int*:
assumes *affine S*
and *affine T*
and $S \cap T \neq \{\}$
shows $\text{aff_dim } \{x + y \mid x y. x \in S \wedge y \in T\} = (\text{aff_dim } S + \text{aff_dim } T) - \text{aff_dim}(S \cap T)$
proof –
obtain *a* **where** $a: a \in S \ a \in T$ **using** *assms by force*
have *aff: affine ((+) (-a) ' S) affine ((+) (-a) ' T)*
using *affine_translation [symmetric, of - a] assms by (simp_all cong: image_cong_simp)*
have *zero: 0 ∈ ((+) (-a) ' S) 0 ∈ ((+) (-a) ' T)*
using *a assms by auto*
have $\{x + y \mid x y. x \in (+) (-a) \text{ ' } S \wedge y \in (+) (-a) \text{ ' } T\} =$
 $(+) (-2 *_{\mathbb{R}} a) \text{ ' } \{x + y \mid x y. x \in S \wedge y \in T\}$
by (*force simp: algebra_simps scaleR_2*)
moreover have $(+) (-a) \text{ ' } S \cap (+) (-a) \text{ ' } T = (+) (-a) \text{ ' } (S \cap T)$
by *auto*
ultimately show *?thesis*
using *aff_dim_sums_Int_0 [OF aff zero] aff_dim_translation_eq*
by (*metis (lifting)*)
qed


```

lemma aff_dim_affine_Int_hyperplane:
  fixes a :: 'a::euclidean_space
  assumes affine S
  shows aff_dim(S ∩ {x. a · x = b}) =
    (if S ∩ {v. a · v = b} = {} then - 1
     else if S ⊆ {v. a · v = b} then aff_dim S
     else aff_dim S - 1)
proof (cases a = 0)
  case True with assms show ?thesis
    by auto
next
  case False
  then have aff_dim (S ∩ {x. a · x = b}) = aff_dim S - 1
    if x ∈ S a · x ≠ b and non: S ∩ {v. a · v = b} ≠ {} for x
  proof -
    have [simp]: {x + y | x y. x ∈ S ∧ a · y = b} = UNIV
      using affine_hyperplane_sums_eq_UNIV [OF assms non] that by blast
    show ?thesis
      using aff_dim_sums_Int [OF assms affine_hyperplane non]
      by (simp add: of_nat_diff False)
  qed
  then show ?thesis
    by (metis (mono_tags, lifting) inf.orderE aff_dim_empty_eq mem_Collect_eq
        subsetI)
  qed

lemma aff_dim_lt_full:
  fixes S :: 'a::euclidean_space set
  shows aff_dim S < DIM('a) ⟷ (affine hull S ≠ UNIV)
by (metis (no_types) aff_dim_affine_hull aff_dim_le_DIM aff_dim_UNIV affine_hull_UNIV
    less_le)

lemma aff_dim_openin:
  fixes S :: 'a::euclidean_space set
  assumes ope: openin (top_of_set T) S and affine T S ≠ {}
  shows aff_dim S = aff_dim T
proof -
  show ?thesis
  proof (rule order_antisym)
    show aff_dim S ≤ aff_dim T
      by (blast intro: aff_dim_subset [OF openin_imp_subset] ope)
  next
    obtain a where a ∈ S
      using ⟨S ≠ {}⟩ by blast
    have S ⊆ T
      using ope openin_imp_subset by auto
    then have a ∈ T
      using ⟨a ∈ S⟩ by auto
  qed

```

```

then have subT': subspace ((λx. - a + x) ' T)
  using affine_diffs_subspace ⟨affine T⟩ by auto
then obtain B where Bsub: B ⊆ ((λx. - a + x) ' T) and po: pairwise
orthogonal B
  and eq1: ⋀x. x ∈ B ⇒ norm x = 1 and independent B
  and cardB: card B = dim ((λx. - a + x) ' T)
  and spanB: span B = ((λx. - a + x) ' T)
  by (rule orthonormal_basis_subspace) auto
obtain e where 0 < e and e: cball a e ∩ T ⊆ S
  by (meson ⟨a ∈ S⟩ openin_contains_cball ope)
have aff_dim T = aff_dim ((λx. - a + x) ' T)
  by (metis aff_dim_translation_eq)
also have ... = dim ((λx. - a + x) ' T)
  using aff_dim_subspace subT' by blast
also have ... = card B
  by (simp add: cardB)
also have ... = card ((λx. e *R x) ' B)
  using ⟨0 < e⟩ by (force simp: inj_on_def card_image)
also have ... ≤ dim ((λx. - a + x) ' S)
proof -
  have e': cball 0 e ∩ (λx. x - a) ' T ⊆ (λx. x - a) ' S
    using e by (auto simp: dist_norm norm_minus_commute subset_eq)
  have (λx. e *R x) ' B ⊆ cball 0 e ∩ (λx. x - a) ' T
    using Bsub ⟨0 < e⟩ eq1 subT' ⟨a ∈ T⟩ by (auto simp: subspace_def)
  then have (λx. e *R x) ' B ⊆ (λx. x - a) ' S
    using e' by blast
  moreover
  have inj_on ((*R) e) (span B)
    using ⟨0 < e⟩ inj_on_def by fastforce
  then have independent ((λx. e *R x) ' B)
    using linear_scale_self ⟨independent B⟩ linear_dependent_inj_imageD by
blast
  ultimately show ?thesis
    by (auto simp: intro!: independent_card_le_dim)
qed
also have ... = aff_dim S
  using ⟨a ∈ S⟩ aff_dim_eq_dim hull_inc by (force cong: image_cong_simp)
finally show aff_dim T ≤ aff_dim S .
qed
qed

```

lemma *dim_openin*:

```

fixes S :: 'a::euclidean_space set
assumes ope: openin (top_of_set T) S and subspace T S ≠ {}
shows dim S = dim T

```

proof (rule order_antisym)

```

show dim S ≤ dim T
  by (metis ope dim_subset openin_subset topspace_euclidean_subtopology)
next

```

```

have dim T = aff_dim S
  using aff_dim_openin
  by (metis aff_dim_subspace ⟨subspace T⟩ ⟨S ≠ {}⟩ ope subspace_affine)
also have ... ≤ dim S
  by (metis aff_dim_subset aff_dim_subspace dim_span span_superset
    subspace_span)
finally show dim T ≤ dim S by simp
qed

```

6.0.18 Lower-dimensional affine subsets are nowhere dense

proposition *dense_complement_subspace*:

```

fixes S :: 'a :: euclidean_space set
assumes dim_less: dim T < dim S and subspace S shows closure(S - T) = S
proof -
  have closure(S - U) = S if dim U < dim S U ⊆ S for U
  proof -
    have span U ⊂ span S
      by (metis neq_iff psubsetI span_eq_dim span_mono that)
    then obtain a where a ≠ 0 a ∈ span S and a: ⋀y. y ∈ span U ⇒ orthogonal
    a y

```

```

      using orthogonal_to_subspace_exists_gen by metis
    show ?thesis
  proof
    have closed S
      by (simp add: ⟨subspace S⟩ closed_subspace)
    then show closure(S - U) ⊆ S
      by (simp add: closure_minimal)
    show S ⊆ closure(S - U)
    proof (clarsimp simp: closure_approachable)
      fix x and e::real
      assume x ∈ S 0 < e
      show ∃y ∈ S - U. dist y x < e
      proof (cases x ∈ U)
        case True
        let ?y = x + (e/2 / norm a) *R a
        show ?thesis
        proof
          show dist ?y x < e
            using ⟨0 < e⟩ by (simp add: dist_norm)
        next
          have ?y ∈ S
            by (metis ⟨a ∈ span S⟩ ⟨x ∈ S⟩ assms(2) span_eq_iff subspace_add
              subspace_scale)
          moreover have ?y ∉ U
          proof -
            have e/2 / norm a ≠ 0
              using ⟨0 < e⟩ ⟨a ≠ 0⟩ by auto
            then show ?thesis

```

```

      by (metis True ⟨a ≠ 0⟩ a orthogonal_scaleR orthogonal_self
real_vector.scale_eq_0_iff span_add_eq span_base)
    qed
    ultimately show ?y ∈ S - U by blast
  qed
next
case False
with ⟨0 < e⟩ ⟨x ∈ S⟩ show ?thesis by force
qed
qed
qed
qed
moreover have S - S ∩ T = S - T
  by blast
moreover have dim (S ∩ T) < dim S
  by (metis dim_less dim_subset inf.cobounded2 inf.orderE inf.strict_boundedE
not_le)
ultimately show ?thesis
  by force
qed

```

corollary *dense_complement_affine*:

```

  fixes S :: 'a :: euclidean_space set
  assumes less: aff_dim T < aff_dim S and affine S shows closure(S - T) = S
proof (cases S ∩ T = {})
  case True
  then show ?thesis
    by (metis Diff_triv affine_hull_eq ⟨affine S⟩ closure_same_affine_hull clo-
sure_subset hull_subset subset_antisym)
  next
  case False
  then obtain z where z: z ∈ S ∩ T by blast
  then have subspace ((+) (- z) ' S)
    by (meson IntD1 affine_diffs_subspace ⟨affine S⟩)
  moreover have int (dim ((+) (- z) ' T)) < int (dim ((+) (- z) ' S))
thm aff_dim_eq_dim
    using z less by (simp add: aff_dim_eq_dim_subtract [of z] hull_inc cong:
image_cong_simp)
  ultimately have closure(((+) (- z) ' S) - ((+) (- z) ' T)) = ((+) (- z) ' S)
    by (simp add: dense_complement_subspace)
  then show ?thesis
    by (metis closure_translation translation_diff translation_invert)
qed

```

corollary *dense_complement_openin_affine_hull*:

```

  fixes S :: 'a :: euclidean_space set
  assumes less: aff_dim T < aff_dim S
  and ope: openin (top_of_set (affine hull S)) S
  shows closure(S - T) = closure S

```

```

proof –
  have affine hull  $S - T \subseteq \text{affine hull } S$ 
    by blast
  then have  $\text{closure } (S \cap \text{closure } (\text{affine hull } S - T)) = \text{closure } (S \cap (\text{affine hull } S - T))$ 
    by (rule closure_openin_Int_closure [OF ope])
  then show ?thesis
    by (metis Int_Diff_aff_dim_affine_hull affine_affine_hull dense_complement_affine_hull_subset inf.orderE less)
qed

```

corollary *dense_complement_convex*:

```

fixes  $S :: 'a :: \text{euclidean\_space set}$ 
assumes  $\text{aff\_dim } T < \text{aff\_dim } S$  convex  $S$ 
shows  $\text{closure}(S - T) = \text{closure } S$ 

```

proof

```

show  $\text{closure } (S - T) \subseteq \text{closure } S$ 
  by (simp add: closure_mono)
have  $\text{closure } (\text{rel\_interior } S - T) = \text{closure } (\text{rel\_interior } S)$ 
  by (simp add: assms dense_complement_openin_affine_hull openin_rel_interior_rel_interior_aff_dim_rel_interior_same_affine_hull)
then show  $\text{closure } S \subseteq \text{closure } (S - T)$ 
  by (metis Diff_mono <convex S> closure_mono convex_closure_rel_interior_order_refl rel_interior_subset)
qed

```

corollary *dense_complement_convex_closed*:

```

fixes  $S :: 'a :: \text{euclidean\_space set}$ 
assumes  $\text{aff\_dim } T < \text{aff\_dim } S$  convex  $S$  closed  $S$ 
shows  $\text{closure}(S - T) = S$ 
by (simp add: assms dense_complement_convex)

```

6.0.19 Parallel slices, etc

If we take a slice out of a set, we can do it perpendicularly, with the normal vector to the slice parallel to the affine hull.

proposition *affine_parallel_slice*:

```

fixes  $S :: 'a :: \text{euclidean\_space set}$ 
assumes affine  $S$ 
  and  $S \cap \{x. a \cdot x \leq b\} \neq \{\}$ 
  and  $\neg (S \subseteq \{x. a \cdot x \leq b\})$ 
obtains  $a' b'$  where  $a' \neq 0$ 
   $S \cap \{x. a' \cdot x \leq b'\} = S \cap \{x. a \cdot x \leq b\}$ 
   $S \cap \{x. a' \cdot x = b'\} = S \cap \{x. a \cdot x = b\}$ 
   $\bigwedge w. w \in S \implies (w + a') \in S$ 

```

proof (*cases* $S \cap \{x. a \cdot x = b\} = \{\}$)

case *True*

```

then obtain  $u v$  where  $u \in S$   $v \in S$   $a \cdot u \leq b$   $a \cdot v > b$ 
using assms by (auto simp: not_le)

```

```

define  $\eta$  where  $\eta = u + ((b - a \cdot u) / (a \cdot v - a \cdot u)) *_R (v - u)$ 
have  $\eta \in S$ 
  by (simp add:  $\eta\_def$   $\langle u \in S \rangle \langle v \in S \rangle \langle affine\ S \rangle mem\_affine\_3\_minus$ )
moreover have  $a \cdot \eta = b$ 
  using  $\langle a \cdot u \leq b \rangle \langle b < a \cdot v \rangle$ 
  by (simp add:  $\eta\_def$  algebra_simps) (simp add: field_simps)
ultimately have False
  using True by force
then show ?thesis ..
next
case False
then obtain  $z$  where  $z \in S$  and  $z: a \cdot z = b$ 
  using assms by auto
with affine_diffs_subspace [OF  $\langle affine\ S \rangle$ ]
have sub: subspace  $((+) (- z) ' S)$  by blast
then have aff: affine  $((+) (- z) ' S)$  and span: span  $((+) (- z) ' S) = ((+) (- z) ' S)$ 
  by (auto simp: subspace_imp_affine)
obtain  $a' a''$  where  $a': a' \in span\ ((+) (- z) ' S)$  and  $a: a = a' + a''$ 
  and  $\bigwedge w. w \in span\ ((+) (- z) ' S) \implies orthogonal\ a''\ w$ 
  using orthogonal_subspace_decomp_exists [of  $(+) (- z) ' S\ a]$  by metis
then have  $\bigwedge w. w \in S \implies a'' \cdot (w - z) = 0$ 
  by (simp add: span_base orthogonal_def)
then have  $a'': \bigwedge w. w \in S \implies a'' \cdot w = (a - a') \cdot z$ 
  by (simp add: a inner_diff_right)
then have  $ba'': \bigwedge w. w \in S \implies a'' \cdot w = b - a' \cdot z$ 
  by (simp add: inner_diff_left z)
show ?thesis
proof (cases  $a' = 0$ )
case True
  with a assms True  $a''$  diff_zero less_irrefl show ?thesis
  by auto
next
case False
show ?thesis
proof
show  $S \cap \{x. a' \cdot x \leq a' \cdot z\} = S \cap \{x. a \cdot x \leq b\}$ 
   $S \cap \{x. a' \cdot x = a' \cdot z\} = S \cap \{x. a \cdot x = b\}$ 
  by (auto simp: a  $ba''$  inner_left_distrib)
have  $\bigwedge w. w \in (+) (- z) ' S \implies (w + a') \in (+) (- z) ' S$ 
  by (metis subspace_add  $a'$  span_eq_iff sub)
then show  $\bigwedge w. w \in S \implies (w + a') \in S$ 
  by fastforce
qed (use False in auto)
qed
qed

```

lemma *diffs_affine_hull_span*:
 assumes $a \in S$

```

  shows  $(\lambda x. x - a) \text{ ' } (\text{affine hull } S) = \text{span } ((\lambda x. x - a) \text{ ' } S)$ 
proof -
  have *:  $(\lambda x. x - a) \text{ ' } (S - \{a\}) = ((\lambda x. x - a) \text{ ' } S) - \{0\}$ 
    by (auto simp: algebra_simps)
  show ?thesis
    by (auto simp add: algebra_simps affine_hull_span2 [OF assms] *)
qed

```

```

lemma aff_dim_dim_affine_diffs:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $\text{affine } S \ a \in S$ 
  shows  $\text{aff\_dim } S = \text{dim } ((\lambda x. x - a) \text{ ' } S)$ 
proof -
  obtain  $B$  where  $\text{aff}: \text{affine hull } B = \text{affine hull } S$ 
    and  $\text{ind}: \neg \text{affine\_dependent } B$ 
    and  $\text{card}: \text{of\_nat } (\text{card } B) = \text{aff\_dim } S + 1$ 
    using  $\text{aff\_dim\_basis\_exists}$  by blast
  then have  $B \neq \{\}$  using  $\text{assms}$ 
    by (metis  $\text{affine\_hull\_eq\_empty ex\_in\_conv}$ )
  then obtain  $c$  where  $c \in B$  by auto
  then have  $c \in S$ 
    by (metis  $\text{aff affine\_hull\_eq } \langle \text{affine } S \rangle \text{ hull\_inc}$ )
  have  $xy: x - c = y - a \iff y = x + 1 *_R (a - c)$  for  $x \ y \ c$  and  $a::'a$ 
    by (auto simp: algebra_simps)
  have *:  $(\lambda x. x - c) \text{ ' } S = (\lambda x. x - a) \text{ ' } S$ 
    using  $\text{assms } \langle c \in S \rangle$ 
    by (auto simp:  $\text{image\_iff } xy; \text{metis mem\_affine\_3\_minus\_pth\_1}$ )
  have  $\text{aff}S: \text{affine hull } S = S$ 
    by (simp add:  $\langle \text{affine } S \rangle$ )
  have  $\text{aff\_dim } S = \text{of\_nat } (\text{card } B) - 1$ 
    using  $\text{card}$  by simp
  also have  $\dots = \text{dim } ((\lambda x. x - c) \text{ ' } B)$ 
    using  $\text{affine\_independent\_card\_dim\_diffs}$  [OF  $\text{ind } \langle c \in B \rangle$ ]
    by (simp add:  $\text{affine\_independent\_card\_dim\_diffs}$  [OF  $\text{ind } \langle c \in B \rangle$ ])
  also have  $\dots = \text{dim } ((\lambda x. x - c) \text{ ' } (\text{affine hull } B))$ 
    by (simp add:  $\text{diffs\_affine\_hull\_span } \langle c \in B \rangle$ )
  also have  $\dots = \text{dim } ((\lambda x. x - a) \text{ ' } S)$ 
    by (simp add:  $\text{aff}S \text{ aff } *$ )
  finally show ?thesis .
qed

```

```

lemma aff_dim_linear_image_le:
  assumes  $\text{linear } f$ 
  shows  $\text{aff\_dim}(f \text{ ' } S) \leq \text{aff\_dim } S$ 
proof -
  have  $\text{aff\_dim } (f \text{ ' } T) \leq \text{aff\_dim } T$  if  $\text{affine } T$  for  $T$ 
  proof (cases  $T = \{\}$ )
    case True then show ?thesis by (simp add:  $\text{aff\_dim\_geq}$ )
  next

```

```

case False
then obtain a where a ∈ T by auto
have 1: ((λx. x - f a) ' f ' T) = {x - f a | x. x ∈ f ' T}
  by auto
have 2: {x - f a | x. x ∈ f ' T} = f ' ((λx. x - a) ' T)
  by (force simp: linear_diff [OF assms])
have aff_dim (f ' T) = int (dim {x - f a | x. x ∈ f ' T})
  by (simp add: ⟨a ∈ T⟩ hull_inc aff_dim_eq_dim [of f a] 1 cong: im-
age_cong_simp)
also have ... = int (dim (f ' ((λx. x - a) ' T)))
  by (force simp: linear_diff [OF assms] 2)
also have ... ≤ int (dim ((λx. x - a) ' T))
  by (simp add: dim_image_le [OF assms])
also have ... ≤ aff_dim T
  by (simp add: aff_dim_dim_affine_diffs [symmetric] ⟨a ∈ T⟩ ⟨affine T⟩)
finally show ?thesis .
qed
then
have aff_dim (f ' (affine hull S)) ≤ aff_dim (affine hull S)
  using affine_affine_hull [of S] by blast
then show ?thesis
  using affine_hull_linear_image assms linear_conv_bounded_linear by fast-
force
qed

```

```

lemma aff_dim_injective_linear_image [simp]:
  assumes linear f inj f
  shows aff_dim (f ' S) = aff_dim S
proof (rule antisym)
  show aff_dim (f ' S) ≤ aff_dim S
    by (simp add: aff_dim_linear_image_le assms(1))
next
  obtain g where linear g g ∘ f = id
    using assms(1) assms(2) linear_injective_left_inverse by blast
  then have aff_dim S ≤ aff_dim (g ' f ' S)
    by (simp add: image_comp)
  also have ... ≤ aff_dim (f ' S)
    by (simp add: ⟨linear g⟩ aff_dim_linear_image_le)
  finally show aff_dim S ≤ aff_dim (f ' S) .
qed

```

```

lemma choose_affine_subset:
  assumes affine S -1 ≤ d and dle: d ≤ aff_dim S
  obtains T where affine T T ⊆ S aff_dim T = d
proof (cases d = -1 ∨ S={})
  case True with assms show ?thesis
    by (metis aff_dim_empty affine_empty bot.extremum that eq_iff)
next

```



```

case False
with assms obtain a where a ∈ S 0 ≤ d by auto
with assms have ss: subspace ((+) (- a) ' S)
  by (simp add: affine_diffs_subspace_subtract cong: image_cong_simp)
have nat d ≤ dim ((+) (- a) ' S)
  by (metis aff_dim_subspace aff_dim_translation_eq dle nat_int nat_mono
ss)
then obtain T where subspace T and Tsb: T ⊆ span ((+) (- a) ' S)
  and Tdim: dim T = nat d
  using choose_subspace_of_subspace [of nat d (+) (- a) ' S] by blast
then have affine T
  using subspace_affine by blast
then have affine ((+) a ' T)
  by (metis affine_hull_eq affine_hull_translation)
moreover have (+) a ' T ⊆ S
proof -
  have T ⊆ (+) (- a) ' S
  by (metis (no_types) span_eq_iff Tsb ss)
  then show (+) a ' T ⊆ S
  using add_ac by auto
qed
moreover have aff_dim ((+) a ' T) = d
  by (simp add: aff_dim_subspace Tdim ⟨0 ≤ d⟩ ⟨subspace T⟩ aff_dim_translation_eq)
ultimately show ?thesis
  by (rule that)
qed

```

6.0.20 Paracompactness

proposition *paracompact*:

fixes $S :: 'a :: \{\text{metric_space}, \text{second_countable_topology}\}$ set

assumes $S \subseteq \bigcup \mathcal{C}$ and $opC: \bigwedge T. T \in \mathcal{C} \implies \text{open } T$

obtains \mathcal{C}' where $S \subseteq \bigcup \mathcal{C}'$

and $\bigwedge U. U \in \mathcal{C}' \implies \text{open } U \wedge (\exists T. T \in \mathcal{C} \wedge U \subseteq T)$

and $\bigwedge x. x \in S$

$\implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U. U \in \mathcal{C}' \wedge (U \cap V \neq \{\})\}$

proof (cases $S = \{\}$)

case True with that show ?thesis by blast

next

case False

have $\exists T U. x \in U \wedge \text{open } U \wedge \text{closure } U \subseteq T \wedge T \in \mathcal{C}$ if $x \in S$ for x

proof -

obtain T where $x \in T T \in \mathcal{C}$ open T

using assms $\langle x \in S \rangle$ by blast

then obtain e where $e > 0$ cball $x e \subseteq T$

by (force simp: open_contains_cball)

then show ?thesis

by (meson open_ball ⟨ $T \in \mathcal{C}$ ⟩ ball_subset_cball centre_in_ball closed_cball
closure_minimal dual_order.trans)

```

qed
then obtain F G where Gin: x ∈ G x and oG: open (G x)
  and clos: closure (G x) ⊆ F x and Fin: F x ∈ C
if x ∈ S for x
  by metis
then obtain F where F ⊆ G ' S countable F ∪ F = ∪ (G ' S)
  using Lindelof [of G ' S] by (metis image_iff)
then obtain K where K: K ⊆ S countable K and eq: ∪ (G ' K) = ∪ (G ' S)
  by (metis countable_subset_image)
with False Gin have K ≠ {} by force
then obtain a :: nat ⇒ 'a where range a = K
  by (metis range_from_nat_into countable K)
then have odif: ∧ n. open (F (a n) - ∪ {closure (G (a m)) | m. m < n})
  using ⟨K ⊆ S⟩ Fin opC by (fastforce simp add:)
let ?C = range (λ n. F (a n) - ∪ {closure (G (a m)) | m. m < n})
have enum_S: ∃ n. x ∈ F (a n) ∧ x ∈ G (a n) if x ∈ S for x
proof -
  have ∃ y ∈ K. x ∈ G y using eq that Gin by fastforce
  then show ?thesis
    using clos K ⟨range a = K⟩ closure_subset by blast
qed
show ?thesis
proof
  show S ⊆ Union ?C
  proof
    fix x assume x ∈ S
    define n where n ≡ LEAST n. x ∈ F (a n)
    have n: x ∈ F (a n)
      using enum_S [OF ⟨x ∈ S⟩] by (force simp: n_def intro: LeastI)
    have notn: x ∉ F (a m) if m < n for m
      using that not_less_Least by (force simp: n_def)
    then have x ∉ ∪ {closure (G (a m)) | m. m < n}
      using n ⟨K ⊆ S⟩ ⟨range a = K⟩ clos notn by fastforce
    with n show x ∈ Union ?C
      by blast
  qed
  show ∧ U. U ∈ ?C ⇒ open U ∧ (∃ T. T ∈ C ∧ U ⊆ T)
    using Fin ⟨K ⊆ S⟩ ⟨range a = K⟩ by (auto simp: odif)
  show ∃ V. open V ∧ x ∈ V ∧ finite {U. U ∈ ?C ∧ (U ∩ V ≠ {})} if x ∈ S
for x
proof -
  obtain n where n: x ∈ F (a n) x ∈ G (a n)
    using ⟨x ∈ S⟩ enum_S by auto
  have {U ∈ ?C. U ∩ G (a n) ≠ {}} ⊆ (λ n. F (a n) - ∪ {closure (G (a m))
|m. m < n}) ' atMost n
  proof clarsimp
    fix k assume (F (a k) - ∪ {closure (G (a m)) | m. m < k}) ∩ G (a n) ≠
  {}
    then have k ≤ n

```

```

    by auto (metis closure_subset not_le subsetCE)
  then show  $F(a\ k) = \bigcup \{ \text{closure}(G(a\ m)) \mid m. m < k \}$ 
     $\in (\lambda n. F(a\ n) = \bigcup \{ \text{closure}(G(a\ m)) \mid m. m < n \}) \text{ ' } \{..n\}$ 
  by force
qed
  moreover have finite  $((\lambda n. F(a\ n) = \bigcup \{ \text{closure}(G(a\ m)) \mid m. m < n \}) \text{ ' } \text{atMost } n)$ 
  by force
  ultimately have *: finite  $\{U \in ?C. U \cap G(a\ n) \neq \{\}\}$ 
  using finite_subset by blast
  have  $a\ n \in S$ 
  using  $\langle K \subseteq S \rangle \langle \text{range } a = K \rangle$  by blast
  then show ?thesis
  by (blast intro: oG n *)
qed
qed
qed

corollary paracompact_closedin:
  fixes  $S :: 'a :: \{ \text{metric\_space, second\_countable\_topology} \}$  set
  assumes cin: closedin (top_of_set U) S
    and oin:  $\bigwedge T. T \in \mathcal{C} \implies \text{openin}(\text{top\_of\_set } U) T$ 
    and  $S \subseteq \bigcup \mathcal{C}$ 
  obtains  $\mathcal{C}'$  where  $S \subseteq \bigcup \mathcal{C}'$ 
    and  $\bigwedge V. V \in \mathcal{C}' \implies \text{openin}(\text{top\_of\_set } U) V \wedge (\exists T. T \in \mathcal{C} \wedge V \subseteq T)$ 
    and  $\bigwedge x. x \in U \implies \exists V. \text{openin}(\text{top\_of\_set } U) V \wedge x \in V \wedge \text{finite } \{X. X \in \mathcal{C}' \wedge (X \cap V \neq \{\})\}$ 

proof -
  have  $\exists Z. \text{open } Z \wedge (T = U \cap Z)$  if  $T \in \mathcal{C}$  for  $T$ 
  using oin [OF that] by (auto simp: openin_open)
  then obtain  $F$  where opF:  $\text{open}(F\ T)$  and intF:  $U \cap F\ T = T$  if  $T \in \mathcal{C}$  for  $T$ 
  by metis
  obtain  $K$  where  $K$ : closed  $K$   $U \cap K = S$ 
  using cin by (auto simp: closedin_closed)
  have 1:  $U \subseteq \bigcup (\text{insert } (-\ K) (F\ \mathcal{C}))$ 
  by clarsimp (metis Int_iff Union_iff  $\langle U \cap K = S \rangle \langle S \subseteq \bigcup \mathcal{C} \rangle$  subsetD intF)
  have 2:  $\bigwedge T. T \in \text{insert } (-\ K) (F\ \mathcal{C}) \implies \text{open } T$ 
  using  $\langle \text{closed } K \rangle$  by (auto simp: opF)
  obtain  $\mathcal{D}$  where  $U \subseteq \bigcup \mathcal{D}$ 
    and D1:  $\bigwedge U. U \in \mathcal{D} \implies \text{open } U \wedge (\exists T. T \in \text{insert } (-\ K) (F\ \mathcal{C}) \wedge U \subseteq T)$ 
    and D2:  $\bigwedge x. x \in U \implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{D}. U \cap V \neq \{\}\}$ 
  by (blast intro: paracompact [OF 1 2])
  let  $?C = \{U \cap V \mid V. V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$ 
  show ?thesis

```

```

proof (rule_tac  $C' = \{U \cap V \mid V. V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$ ) in that
  show  $S \subseteq \bigcup ?C$ 
    using  $\langle U \cap K = S \rangle \langle U \subseteq \bigcup \mathcal{D} \rangle K$  by (blast dest!: subsetD)
  show  $\bigwedge V. V \in ?C \implies \text{openin } (\text{top\_of\_set } U) V \wedge (\exists T. T \in \mathcal{C} \wedge V \subseteq T)$ 
    using D1 intF by fastforce
  have *:  $\{X. (\exists V. X = U \cap V \wedge V \in \mathcal{D} \wedge V \cap K \neq \{\}) \wedge X \cap (U \cap V) \neq \{\}\} \subseteq$ 
     $(\lambda x. U \cap x) \text{ ` } \{U \in \mathcal{D}. U \cap V \neq \{\}\}$  for V
    by blast
  show  $\exists V. \text{openin } (\text{top\_of\_set } U) V \wedge x \in V \wedge \text{finite } \{X \in ?C. X \cap V \neq \{\}\}$ 
    if  $x \in U$  for x
  proof -
    from D2 [OF that] obtain V where open V  $x \in V$  finite  $\{U \in \mathcal{D}. U \cap V \neq \{\}\}$ 
    by auto
    with * show ?thesis
    by (rule_tac  $x=U \cap V$  in exI) (auto intro: that finite_subset [OF *])
  qed
qed
qed

```

corollary *paracompact_closed:*

```

fixes S :: 'a :: {metric_space, second_countable_topology} set
assumes closed S
  and opC:  $\bigwedge T. T \in \mathcal{C} \implies \text{open } T$ 
  and  $S \subseteq \bigcup \mathcal{C}$ 
obtains C' where  $S \subseteq \bigcup C'$ 
  and  $\bigwedge U. U \in C' \implies \text{open } U \wedge (\exists T. T \in \mathcal{C} \wedge U \subseteq T)$ 
  and  $\bigwedge x. \exists V. \text{open } V \wedge x \in V \wedge$ 
    finite  $\{U. U \in C' \wedge (U \cap V \neq \{\})\}$ 
by (rule paracompact_closedin [of UNIV S C]) (auto simp: assms)

```

6.0.21 Closed-graph characterization of continuity

lemma *continuous_closed_graph_gen:*

```

fixes T :: 'b::real_normed_vector set
assumes contf: continuous_on S f and fim:  $f \text{ ` } S \subseteq T$ 
  shows closedin (top_of_set (S × T)) (( $\lambda x. \text{Pair } x (f x)$ ) ` S)

```

proof -

```

have eq: (( $\lambda x. \text{Pair } x (f x)$ ) ` S) = (S × T ∩ ( $\lambda z. (f \circ \text{fst})z - \text{snd } z$ ) - ` {0})
  using fim by auto

```

show ?thesis

unfolding eq

by (intro continuous_intros continuous_closedin_preimage continuous_on_subset [OF contf]) auto

qed

lemma *continuous_closed_graph_eq:*

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $compact\ T$  and  $fm: f \in S \rightarrow T$ 
shows  $continuous\_on\ S\ f \longleftrightarrow$ 
   $closedin\ (top\_of\_set\ (S \times T))\ ((\lambda x. Pair\ x\ (f\ x))\ 'S)$ 
  (is  $?lhs = ?rhs$ )
proof -
  have  $?lhs$  if  $?rhs$ 
  proof ( $clarsimp\ simp\ add: continuous\_on\_closed\_gen\ [OF\ fm]$ )
    fix  $U$ 
    assume  $U: closedin\ (top\_of\_set\ T)\ U$ 
    have  $eq: (S \cap f\ -' U) = fst\ '(((\lambda x. Pair\ x\ (f\ x))\ 'S) \cap (S \times U))$ 
      by ( $force\ simp: image\_iff$ )
    show  $closedin\ (top\_of\_set\ S)\ (S \cap f\ -' U)$ 
      by ( $simp\ add: U\ closedin\_Int\ closedin\_Times\ closed\_map\_fst\ [OF\ \langle compact\ T \rangle]$  that eq)
    qed
  with  $continuous\_closed\_graph\_gen\ assms$  show  $?thesis$  by  $blast$ 
qed

```

```

lemma  $continuous\_closed\_graph:$ 
  fixes  $f :: 'a::topological\_space \Rightarrow 'b::real\_normed\_vector$ 
  assumes  $closed\ S$  and  $contf: continuous\_on\ S\ f$ 
  shows  $closed\ ((\lambda x. Pair\ x\ (f\ x))\ 'S)$ 
proof ( $rule\ closedin\_closed\_trans$ )
  show  $closedin\ (top\_of\_set\ (S \times UNIV))\ ((\lambda x. (x, f\ x))\ 'S)$ 
    by ( $rule\ continuous\_closed\_graph\_gen\ [OF\ contf\ subset\_UNIV]$ )
qed ( $simp\ add: \langle closed\ S \rangle\ closed\_Times$ )

```

```

lemma  $continuous\_from\_closed\_graph:$ 
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $compact\ T$  and  $fm: f \in S \rightarrow T$  and  $clo: closed\ ((\lambda x. Pair\ x\ (f\ x))\ 'S)$ 
  shows  $continuous\_on\ S\ f$ 
    using  $fm\ clo$ 
    by ( $auto\ intro: closed\_subset\ simp: continuous\_closed\_graph\_eq\ [OF\ \langle compact\ T \rangle\ fm]$ )

```

```

lemma  $continuous\_on\_Un\_local\_open:$ 
  assumes  $opS: openin\ (top\_of\_set\ (S \cup T))\ S$ 
    and  $opT: openin\ (top\_of\_set\ (S \cup T))\ T$ 
    and  $contf: continuous\_on\ S\ f$  and  $contg: continuous\_on\ T\ f$ 
  shows  $continuous\_on\ (S \cup T)\ f$ 
    using  $pastng\_lemma\ [of\ \{S, T\}\ top\_of\_set\ (S \cup T)\ id\ euclidean\ \lambda i. f\ f]\ contf\ contg\ opS\ opT$ 
    by ( $simp\ add: subtopology\_subtopology$ ) ( $metis\ inf.absorb2\ openin\_imp\_subset$ )

```

```

lemma  $continuous\_on\_cases\_local\_open:$ 
  assumes  $opS: openin\ (top\_of\_set\ (S \cup T))\ S$ 

```

```

and opT: openin (top_of_set ( $S \cup T$ )) T
and contf: continuous_on S f and contg: continuous_on T g
and fg:  $\bigwedge x. x \in S \wedge \neg P x \vee x \in T \wedge P x \implies f x = g x$ 
shows continuous_on ( $S \cup T$ ) ( $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ )
proof -
  have  $\bigwedge x. x \in S \implies (\text{if } P x \text{ then } f x \text{ else } g x) = f x$   $\bigwedge x. x \in T \implies (\text{if } P x \text{ then } f x \text{ else } g x) = g x$ 
    by (simp_all add: fg)
  then have continuous_on S ( $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ ) continuous_on T ( $\lambda x. \text{if } P x \text{ then } f x \text{ else } g x$ )
    by (simp_all add: contf contg cong: continuous_on_cong)
  then show ?thesis
    by (rule continuous_on_Un_local_open [OF opS opT])
qed

```

6.0.22 The union of two collinear segments is another segment

```

proposition in_convex_hull_exchange:
  fixes a :: 'a::euclidean_space
  assumes a:  $a \in \text{convex hull } S$  and xS:  $x \in \text{convex hull } S$ 
  obtains b where  $b \in S \wedge x \in \text{convex hull } (\text{insert } a (S - \{b\}))$ 
proof (cases a  $\in S$ )
  case True
    with xS insert_Diff that show ?thesis by fastforce
  next
    case False
    show ?thesis
    proof (cases finite S  $\wedge$  card S  $\leq$  Suc (DIM('a)))
      case True
        then obtain u where u0:  $\bigwedge i. i \in S \implies 0 \leq u i$  and u1:  $\text{sum } u S = 1$ 
          and ua:  $(\sum_{i \in S}. u i *_{\mathbb{R}} i) = a$ 
          using a by (auto simp: convex_hull_finite)
        obtain v where v0:  $\bigwedge i. i \in S \implies 0 \leq v i$  and v1:  $\text{sum } v S = 1$ 
          and vx:  $(\sum_{i \in S}. v i *_{\mathbb{R}} i) = x$ 
          using True xS by (auto simp: convex_hull_finite)
        show ?thesis
        proof (cases  $\exists b. b \in S \wedge v b = 0$ )
          case True
            then obtain b where  $b \in S \wedge v b = 0$ 
              by blast
            show ?thesis
          proof
            have fin: finite ( $\text{insert } a (S - \{b\})$ )
              using sum.infinite v1 by fastforce
            show  $x \in \text{convex hull } \text{insert } a (S - \{b\})$ 
              unfolding convex_hull_finite [OF fin] mem_Collect_eq
            proof (intro conjI exI ballI)
              have  $(\sum x \in \text{insert } a (S - \{b\}). \text{if } x = a \text{ then } 0 \text{ else } v x) =$ 

```

```

      ( $\sum x \in S - \{b\}. \text{if } x = a \text{ then } 0 \text{ else } v x$ )
    using fin by (force intro: sum.mono_neutral_right)
  also have ... = ( $\sum x \in S - \{b\}. v x$ )
    using b False by (auto intro!: sum.cong split: if_split_asm)
  also have ... = ( $\sum x \in S. v x$ )
    by (metis  $\langle v b = 0 \rangle$  diff_zero sum.infinite sum_diff1 u1 zero_neq_one)
  finally show ( $\sum x \in \text{insert } a (S - \{b\}). \text{if } x = a \text{ then } 0 \text{ else } v x = 1$ )
    by (simp add: v1)
  show  $\bigwedge x. x \in \text{insert } a (S - \{b\}) \implies 0 \leq (\text{if } x = a \text{ then } 0 \text{ else } v x)$ 
    by (auto simp: v0)
  have ( $\sum x \in \text{insert } a (S - \{b\}). (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x =$ 
    ( $\sum x \in S - \{b\}. (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x$ )
    using fin by (force intro: sum.mono_neutral_right)
  also have ... = ( $\sum x \in S - \{b\}. v x *_R x$ )
    using b False by (auto intro!: sum.cong split: if_split_asm)
  also have ... = ( $\sum x \in S. v x *_R x$ )
    by (metis (no_types, lifting) b(2) diff_zero fin finite.emptyI finite_Diff2
    finite_insert scale_eq_0_iff sum_diff1)
  finally show ( $\sum x \in \text{insert } a (S - \{b\}). (\text{if } x = a \text{ then } 0 \text{ else } v x) *_R x =$ 
 $x$ 
    by (simp add: vx)
  qed
qed (rule  $\langle b \in S \rangle$ )
next
case False
have le_Max:  $u i / v i \leq \text{Max } ((\lambda i. u i / v i) ' S)$  if  $i \in S$  for  $i$ 
  by (simp add: True that)
have Max  $((\lambda i. u i / v i) ' S) \in (\lambda i. u i / v i) ' S$ 
  using True v1 by (auto intro: Max_in)
then obtain b where  $b \in S$  and beq:  $\text{Max } ((\lambda b. u b / v b) ' S) = u b / v b$ 
  by blast
then have  $0 \neq u b / v b$ 
  using le_Max beq divide_le_0_iff le_numeral_extra(2) sum_nonpos u1
  by (metis False eq_iff v0)
then have  $0 < u b$  and  $0 < v b$ 
  using False  $\langle b \in S \rangle$  u0 v0 by force+
have fin: finite (insert  $a (S - \{b\})$ )
  using sum.infinite v1 by fastforce
show ?thesis
proof
  show  $x \in \text{convex hull insert } a (S - \{b\})$ 
    unfolding convex_hull_finite [OF fin] mem_Collect_eq
  proof (intro conjI exI ballI)
    have ( $\sum x \in \text{insert } a (S - \{b\}). \text{if } x=a \text{ then } v b / u b \text{ else } v x - (v b / u$ 
 $b) * u x =$ 
       $v b / u b + (\sum x \in S - \{b\}. v x - (v b / u b) * u x)$ 
      using  $\langle a \notin S \rangle \langle b \in S \rangle$  True
      by (auto intro!: sum.cong split: if_split_asm)
    also have ... =  $v b / u b + (\sum x \in S - \{b\}. v x) - (v b / u b) * (\sum x \in$ 

```

```

S - {b}. u x)
  by (simp add: Groups_Big.sum_subtractf sum_distrib_left)
  also have ... = (∑ x∈S. v x)
    using ⟨0 < u b⟩ True by (simp add: Groups_Big.sum_diff1 u1
field_simps)
  finally show sum (λx. if x=a then v b / u b else v x - (v b / u b) * u x)
(insert a (S - {b})) = 1
  by (simp add: v1)
  show 0 ≤ (if i = a then v b / u b else v i - v b / u b * u i)
  if i ∈ insert a (S - {b}) for i
  using ⟨0 < u b⟩ ⟨0 < v b⟩ v0 [of i] le_Max [of i] beq that False
  by (auto simp: field_simps split: if_split_asm)
  have (∑ x∈insert a (S - {b}). (if x=a then v b / u b else v x - v b / u b
* u x) *R x) =
    (v b / u b) *R a + (∑ x∈S - {b}. (v x - v b / u b * u x) *R x)
  using ⟨a ∉ S⟩ ⟨b ∈ S⟩ True by (auto intro!: sum.cong split: if_split_asm)
  also have ... = (v b / u b) *R a + (∑ x ∈ S - {b}. v x *R x) - (v b / u
b) *R (∑ x ∈ S - {b}. u x *R x)
  by (simp add: Groups_Big.sum_subtractf scaleR_left_diff_distrib
sum_distrib_left scale_sum_right)
  also have ... = (∑ x∈S. v x *R x)
  using ⟨0 < u b⟩ True by (simp add: ua vx Groups_Big.sum_diff1
algebra_simps)
  finally
  show (∑ x∈insert a (S - {b}). (if x=a then v b / u b else v x - v b / u
b * u x) *R x) = x
  by (simp add: vx)
qed
qed (rule ⟨b ∈ S⟩)
qed
next
case False
obtain T where finite T T ⊆ S and caT: card T ≤ Suc (DIM('a)) and xT:
x ∈ convex hull T
  using xS by (auto simp: caratheodory [of S])
with False obtain b where b: b ∈ S b ∉ T
  by (metis antisym subsetI)
show ?thesis
proof
  show x ∈ convex hull insert a (S - {b})
    using ⟨T ⊆ S⟩ b by (blast intro: subsetD [OF hull_mono xT])
  qed (rule ⟨b ∈ S⟩)
qed
qed

```

lemma *convex_hull_exchange_Union:*

fixes $a :: 'a::euclidean_space$

assumes $a \in \text{convex hull } S$

shows $\text{convex hull } S = (\bigcup b \in S. \text{convex hull } (\text{insert } a (S - \{b\})))$ (**is** $?lhs =$


```

?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs
    by (blast intro: in_convex_hull_exchange [OF assms])
  show ?rhs  $\subseteq$  ?lhs
    proof clarify
      fix x b
      assume  $b \in S$   $x \in \text{convex hull insert } a (S - \{b\})$ 
      then show  $x \in \text{convex hull } S$  if  $b \in S$ 
        by (metis (no_types) that assms order_refl hull_mono hull_redundant insert_Diff_single insert_subset subsetCE)
      qed
    qed

lemma Un_closed_segment:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $b \in \text{closed\_segment } a c$ 
  shows  $\text{closed\_segment } a b \cup \text{closed\_segment } b c = \text{closed\_segment } a c$ 
proof (cases  $c = a$ )
  case True
    with assms show ?thesis by simp
  next
    case False
    with assms have  $\text{convex hull } \{a, b\} \cup \text{convex hull } \{b, c\} = (\bigcup_{ba \in \{a, c\}} \text{convex hull insert } b (\{a, c\} - \{ba\}))$ 
      by (auto simp: insert_Diff_if insert_commute)
    then show ?thesis
      using convex_hull_exchange_Union
      by (metis assms segment_convex_hull)
  qed

lemma Un_open_segment:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $b \in \text{open\_segment } a c$ 
  shows  $\text{open\_segment } a b \cup \{b\} \cup \text{open\_segment } b c = \text{open\_segment } a c$  (is ?lhs = ?rhs)
proof -
  have  $b \in \text{closed\_segment } a c$ 
    by (simp add: assms open_closed_segment)
  have *: ?rhs  $\subseteq$  insert b (open_segment a b  $\cup$  open_segment b c)
    if  $\{b, c, a\} \cup \text{open\_segment } a b \cup \text{open\_segment } b c = \{c, a\} \cup ?rhs$ 
  proof -
    have insert a (insert c (insert b (open_segment a b  $\cup$  open_segment b c))) = insert a (insert c (?rhs))
      using that by (simp add: insert_commute)
    then show ?thesis
      by (metis (no_types) Diff_cancel Diff_eq_empty_iff Diff_insert2 open_segment_def)
  qed
show ?thesis

```

```

proof
  show ?lhs  $\subseteq$  ?rhs
    by (simp add: assms b subset_open_segment)
  show ?rhs  $\subseteq$  ?lhs
    using Un_closed_segment [OF b] *
    by (simp add: closed_segment_eq_open insert_commute)
qed
qed

```

6.0.23 Covering an open set by a countable chain of compact sets

```

proposition open_Union_compact_subsets:
  fixes S :: 'a::euclidean_space set
  assumes open S
  obtains C where  $\bigwedge n. \text{compact}(C\ n) \wedge n. C\ n \subseteq S$ 
     $\bigwedge n. C\ n \subseteq \text{interior}(C(\text{Suc}\ n))$ 
     $\bigcup (\text{range}\ C) = S$ 
     $\bigwedge K. [\text{compact}\ K; K \subseteq S] \implies \exists N. \forall n \geq N. K \subseteq (C\ n)$ 
proof (cases S = {})
  case True
    then show ?thesis
      by (rule_tac C =  $\lambda n. \{\}$  in that) auto
  next
    case False
      then obtain a where a  $\in$  S
        by auto
      let ?C =  $\lambda n. \text{cball}\ a\ (\text{real}\ n) - (\bigcup x \in -S. \bigcup e \in \text{ball}\ 0\ (1 / \text{real}(\text{Suc}\ n)). \{x + e\})$ 
      have  $\exists N. \forall n \geq N. K \subseteq (f\ n)$ 
        if  $\bigwedge n. \text{compact}(f\ n)$  and sub_int:  $\bigwedge n. f\ n \subseteq \text{interior}(f(\text{Suc}\ n))$ 
        and eq:  $\bigcup (\text{range}\ f) = S$  and compact K  $K \subseteq S$  for f K
      proof -
        have *:  $\forall n. f\ n \subseteq (\bigcup n. \text{interior}(f\ n))$ 
          by (meson Sup_upper2 UNIV_I  $\langle \bigwedge n. f\ n \subseteq \text{interior}(f(\text{Suc}\ n)) \rangle$  image_iff)
        have mono:  $\bigwedge m\ n. m \leq n \implies f\ m \subseteq f\ n$ 
          by (meson dual_order.trans interior_subset lift_Suc_mono_le sub_int)
        obtain I where finite I and I:  $K \subseteq (\bigcup i \in I. \text{interior}(f\ i))$ 
        proof (rule compactE_image [OF  $\langle \text{compact}\ K \rangle$ ])
          show  $K \subseteq (\bigcup n. \text{interior}(f\ n))$ 
            using  $\langle K \subseteq S \rangle \langle \bigcup (f\ ` UNIV) = S \rangle$  * by blast
        qed auto
      { fix n
        assume n:  $\text{Max}\ I \leq n$ 
        have  $(\bigcup i \in I. \text{interior}(f\ i)) \subseteq f\ n$ 
          by (rule UN_least) (meson dual_order.trans interior_subset mono I Max_ge [OF  $\langle \text{finite}\ I \rangle$ ] n)
        then have  $K \subseteq f\ n$ 
          using I by auto
      }

```

```

}
then show ?thesis
  by blast
qed
moreover have  $\exists f. (\forall n. \text{compact}(f\ n)) \wedge (\forall n. (f\ n) \subseteq S) \wedge (\forall n. (f\ n) \subseteq \text{interior}(f(\text{Suc}\ n))) \wedge$ 
   $((\bigcup (\text{range}\ f) = S))$ 
proof (intro exI conjI allI)
show  $\bigwedge n. \text{compact}\ (?C\ n)$ 
  by (auto simp: compact_diff_open_sums)
show  $\bigwedge n. ?C\ n \subseteq S$ 
  by auto
show  $?C\ n \subseteq \text{interior}\ (?C\ (\text{Suc}\ n))$  for  $n$ 
proof (simp add: interior_diff, rule Diff_mono)
show  $\text{cball}\ a\ (\text{real}\ n) \subseteq \text{ball}\ a\ (1 + \text{real}\ n)$ 
  by (simp add: cball_subset_ball_iff)
have  $cl: \text{closed}\ (\bigcup_{x \in -S} \bigcup_{e \in \text{cball}\ 0\ (1 / (2 + \text{real}\ n))}. \{x + e\})$ 
  using assms by (auto intro: closed_compact_sums)
have  $\text{closure}\ (\bigcup_{x \in -S} \bigcup_{y \in \text{ball}\ 0\ (1 / (2 + \text{real}\ n))}. \{x + y\})$ 
   $\subseteq (\bigcup_{x \in -S} \bigcup_{e \in \text{cball}\ 0\ (1 / (2 + \text{real}\ n))}. \{x + e\})$ 
  by (intro closure_minimal UN_mono ball_subset_cball order_refl cl)
also have  $\dots \subseteq (\bigcup_{x \in -S} \bigcup_{y \in \text{ball}\ 0\ (1 / (1 + \text{real}\ n))}. \{x + y\})$ 
  by (simp add: cball_subset_ball_iff field_split_simps UN_mono)
finally show  $\text{closure}\ (\bigcup_{x \in -S} \bigcup_{y \in \text{ball}\ 0\ (1 / (2 + \text{real}\ n))}. \{x + y\})$ 
   $\subseteq (\bigcup_{x \in -S} \bigcup_{y \in \text{ball}\ 0\ (1 / (1 + \text{real}\ n))}. \{x + y\}) .$ 
qed
have  $S \subseteq \bigcup (\text{range}\ ?C)$ 
proof
fix  $x$ 
assume  $x: x \in S$ 
then obtain  $e$  where  $e > 0$  and  $e: \text{ball}\ x\ e \subseteq S$ 
  using assms open_contains_ball by blast
then obtain  $N1$  where  $N1 > 0$  and  $N1: \text{real}\ N1 > 1/e$ 
  using reals_Archimedean2
  by (metis divide_less_0_iff less_eq_real_def neq0_conv not_le of_nat_0
of_nat_1 of_nat_less_0_iff)
obtain  $N2$  where  $N2: \text{norm}(x - a) \leq \text{real}\ N2$ 
  by (meson real_arch_simple)
have  $N12: \text{inverse}((N1 + N2) + 1) \leq \text{inverse}(N1)$ 
  using  $\langle N1 > 0 \rangle$  by (auto simp: field_split_simps)
have  $x \neq y + z$  if  $y \notin S$   $\text{norm}\ z < 1 / (1 + (\text{real}\ N1 + \text{real}\ N2))$  for  $y\ z$ 
proof -
have  $e * \text{real}\ N1 < e * (1 + (\text{real}\ N1 + \text{real}\ N2))$ 
  by (simp add:  $\langle 0 < e \rangle$ )
then have  $1 / (1 + (\text{real}\ N1 + \text{real}\ N2)) < e$ 
  using  $N1 \langle e > 0 \rangle$ 
  by (metis divide_less_eq less_trans mult.commute of_nat_add of_nat_less_0_iff
of_nat_Suc)
then have  $x - z \in \text{ball}\ x\ e$ 

```

```

    using that by simp
  then have  $x - z \in S$ 
    using e by blast
  with that show ?thesis
    by auto
qed
with N2 show  $x \in \bigcup (\text{range } ?C)$ 
  by (rule_tac a = N1+N2 in UN_I) (auto simp: dist_norm norm_minus_commute)
qed
then show  $\bigcup (\text{range } ?C) = S$  by auto
qed
ultimately show ?thesis
  using that by metis
qed

```

6.0.24 Orthogonal complement

definition *orthogonal_comp* $(_^\perp [80] 80)$
 where *orthogonal_comp* $W \equiv \{x. \forall y \in W. \text{orthogonal } y \ x\}$

proposition *subspace_orthogonal_comp*: *subspace* (W^\perp)
unfolding *subspace_def* *orthogonal_comp_def* *orthogonal_def*
 by (auto simp: inner_right_distrib)

lemma *orthogonal_comp_anti_mono*:

assumes $A \subseteq B$
 shows $B^\perp \subseteq A^\perp$

proof

fix x assume $x \in B^\perp$

show $x \in \text{orthogonal_comp } A$ using x unfolding *orthogonal_comp_def*
 by (simp add: *orthogonal_def*, metis *assms in_mono*)

qed

lemma *orthogonal_comp_null* [*simp*]: $\{0\}^\perp = \text{UNIV}$
 by (auto simp: *orthogonal_comp_def* *orthogonal_def*)

lemma *orthogonal_comp_UNIV* [*simp*]: $\text{UNIV}^\perp = \{0\}$
unfolding *orthogonal_comp_def* *orthogonal_def*
 by auto (use *inner_eq_zero_iff* in blast)

lemma *orthogonal_comp_subset*: $U \subseteq U^{\perp\perp}$
 by (auto simp: *orthogonal_comp_def* *orthogonal_def* *inner_commute*)

lemma *subspace_sum_minimal*:

assumes $S \subseteq U$ $T \subseteq U$ *subspace* U
 shows $S + T \subseteq U$

proof

fix x

assume $x \in S + T$

```

then obtain  $xs\ xt$  where  $xs \in S\ xt \in T\ x = xs+xt$ 
  by (meson set_plus_elim)
then show  $x \in U$ 
  by (meson assms subsetCE subspace_add)
qed

```

proposition *subspace_sum_orthogonal_comp*:

```

fixes  $U :: 'a :: euclidean\_space\ set$ 
assumes subspace U
shows  $U + U^\perp = UNIV$ 
proof -
  obtain  $B$  where  $B \subseteq U$ 
    and ortho: pairwise orthogonal B  $\wedge x. x \in B \implies norm\ x = 1$ 
    and independent B card B = dim U span B = U
    using orthonormal_basis_subspace [OF assms] by metis
  then have finite B
    by (simp add: indep_card_eq_dim_span)
  have  $*$ :  $\forall x \in B. \forall y \in B. x \cdot y = (if\ x=y\ then\ 1\ else\ 0)$ 
    using ortho norm_eq_1 by (auto simp: orthogonal_def pairwise_def)
  { fix  $v$ 
    let  $?u = \sum b \in B. (v \cdot b) *_{\mathbb{R}} b$ 
    have  $v = ?u + (v - ?u)$ 
      by simp
    moreover have  $?u \in U$ 
      by (metis (no_types, lifting) span B = U assms subspace_sum span_base span_mul)
    moreover have  $(v - ?u) \in U^\perp$ 
      proof (clarsimp simp: orthogonal_comp_def orthogonal_def)
        fix  $y$ 
        assume  $y \in U$ 
        with  $\langle span\ B = U \rangle$  span_finite [OF finite B]
        obtain  $u$  where  $u = (\sum b \in B. u\ b *_{\mathbb{R}} b)$ 
          by auto
        have  $b \cdot (v - ?u) = 0$  if  $b \in B$  for  $b$ 
          using that finite B
          by (simp add: algebra_simps inner_sum_right if_distrib [of (*) v for v])
        inner_commute cong: if_cong
        then show  $y \cdot (v - ?u) = 0$ 
          by (simp add: u inner_sum_left)
      qed
    ultimately have  $v \in U + U^\perp$ 
      using set_plus_intro by fastforce
  } then show ?thesis
    by auto
qed

```

lemma *orthogonal_Int_0*:

```

assumes subspace U
shows  $U \cap U^\perp = \{0\}$ 

```

using *orthogonal_comp_def* *orthogonal_self*
 by (force *simp*: *assms* *subspace_0* *subspace_orthogonal_comp*)

lemma *orthogonal_comp_self*:

fixes $U :: 'a :: \text{euclidean_space}$ *set*

assumes *subspace* U

shows $U^{\perp\perp} = U$

proof

have ssU' : *subspace* (U^\perp)

by (*simp* *add*: *subspace_orthogonal_comp*)

have $u \in U$ if $u \in U^{\perp\perp}$ for u

proof –

obtain v w where $u = v+w$ $v \in U$ $w \in U^\perp$

using *subspace_sum_orthogonal_comp* [*OF* *assms*] *set_plus_elim* by *blast*

then have $u-v \in U^\perp$

by *simp*

moreover have $v \in U^{\perp\perp}$

using $\langle v \in U \rangle$ *orthogonal_comp_subset* by *blast*

then have $u-v \in U^{\perp\perp}$

by (*simp* *add*: *subspace_diff* *subspace_orthogonal_comp* *that*)

ultimately have $u-v = 0$

using *orthogonal_Int_0* ssU' by *blast*

with $\langle v \in U \rangle$ show *?thesis*

by *auto*

qed

then show $U^{\perp\perp} \subseteq U$

by *auto*

qed (use *orthogonal_comp_subset* in *auto*)

lemma *ker_orthogonal_comp_adjoint*:

fixes $f :: 'm :: \text{euclidean_space} \Rightarrow 'n :: \text{euclidean_space}$

assumes *linear* f

shows $f^{-1} \{0\} = (\text{range } (\text{adjoint } f))^\perp$

proof –

have $\bigwedge x. [\forall y. y \cdot f x = 0] \implies f x = 0$

using *assms* *inner_commute_all_zero_iff* by *metis*

then show *?thesis*

using *assms*

by (*auto* *simp*: *orthogonal_comp_def* *orthogonal_def* *adjoint_works* *inner_commute*)

qed

6.0.25 A non-injective linear function maps into a hyperplane.

lemma *linear_surj_adj_imp_inj*:

fixes $f :: 'm :: \text{euclidean_space} \Rightarrow 'n :: \text{euclidean_space}$

assumes *linear* f *surj* (*adjoint* f)

shows *inj* f

proof –

```

have  $\exists x. y = \text{adjoint } f \ x$  for  $y$ 
  using assms by (simp add: surjD)
then show inj f
  using assms unfolding inj_on_def image_def
  by (metis (no_types) adjoint_works euclidean_eqI)
qed

```

— <https://mathonline.wikidot.com/injectivity-and-surjectivity-of-the-adjoint-of-a-linear-map>

```

lemma surj_adjoint_iff_inj [simp]:
  fixes  $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes linear f
  shows  $\text{surj } (\text{adjoint } f) \longleftrightarrow \text{inj } f$ 
proof
  assume surj (adjoint f)
  then show inj f
    by (simp add: assms linear_surj_adj_imp_inj)
next
  assume inj f
  have  $f - \{0\} = \{0\}$ 
    using assms  $\langle \text{inj } f \rangle$  linear_0 linear_injective_0 by fastforce
  moreover have  $f - \{0\} = \text{range } (\text{adjoint } f)^\perp$ 
    by (intro ker_orthogonal_comp_adjoint assms)
  ultimately have  $\text{range } (\text{adjoint } f)^{\perp\perp} = \text{UNIV}$ 
    by (metis orthogonal_comp_null)
  then show surj (adjoint f)
    using adjoint_linear  $\langle \text{linear } f \rangle$ 
    by (subst (asm) orthogonal_comp_self)
    (simp add: adjoint_linear linear_subspace_image)
qed

```

```

lemma inj_adjoint_iff_surj [simp]:
  fixes  $f :: 'm::\text{euclidean\_space} \Rightarrow 'n::\text{euclidean\_space}$ 
  assumes linear f
  shows  $\text{inj } (\text{adjoint } f) \longleftrightarrow \text{surj } f$ 
proof
  assume inj (adjoint f)
  have  $(\text{adjoint } f) - \{0\} = \{0\}$ 
    by (metis  $\langle \text{inj } (\text{adjoint } f) \rangle$  adjoint_linear assms surj_adjoint_iff_inj ker_orthogonal_comp_adjoint orthogonal_comp_UNIV)
  then have  $(\text{range}(f))^\perp = \{0\}$ 
    by (metis (no_types, opaque_lifting) adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint set_zero)
  then show surj f
    by (metis  $\langle \text{inj } (\text{adjoint } f) \rangle$  adjoint_adjoint adjoint_linear assms surj_adjoint_iff_inj)
next
  assume surj f
  then have  $\text{range } f = (\text{adjoint } f - \{0\})^\perp$ 
    by (simp add: adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint)
  then have  $\{0\} = \text{adjoint } f - \{0\}$ 

```

```

    using <surj f> adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint
  by force
  then show inj (adjoint f)
    by (simp add: <surj f> adjoint_adjoint adjoint_linear assms linear_surj_adj_imp_inj)
  qed

```

```

lemma linear_singular_into_hyperplane:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes linear f
  shows  $\neg$  inj f  $\longleftrightarrow$   $(\exists a. a \neq 0 \wedge (\forall x. a \cdot f x = 0))$  (is _ = ?rhs)
proof
  assume  $\neg$ inj f
  then show ?rhs
    using all_zero_iff
    by (metis (no_types, opaque_lifting) adjoint_clauses(2) adjoint_linear assms
        linear_injective_0 linear_injective_imp_surjective linear_surj_adj_imp_inj)
next
  assume ?rhs
  then show  $\neg$ inj f
    by (metis assms linear_injective_isomorphism all_zero_iff)
qed

```

```

lemma linear_singular_image_hyperplane:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes linear f  $\neg$ inj f
  obtains a where  $a \neq 0 \wedge S. f ' S \subseteq \{x. a \cdot x = 0\}$ 
  using assms by (fastforce simp add: linear_singular_into_hyperplane)

```

end

6.1 Path-Connectedness

```

theory Path_Connected
imports
  Starlike
  T1_Spaces
begin

```

6.1.1 Paths and Arcs

```

definition path :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  bool
  where path g  $\equiv$  continuous_on {0..1} g

```

```

definition pathstart :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  'a
  where pathstart g  $\equiv$  g 0

```

```

definition pathfinish :: (real  $\Rightarrow$  'a::topological_space)  $\Rightarrow$  'a
  where pathfinish g  $\equiv$  g 1

```


definition *path_image* :: $(\text{real} \Rightarrow 'a::\text{topological_space}) \Rightarrow 'a \text{ set}$
where $\text{path_image } g \equiv g \text{ ` } \{0 .. 1\}$

definition *reversepath* :: $(\text{real} \Rightarrow 'a::\text{topological_space}) \Rightarrow \text{real} \Rightarrow 'a$
where $\text{reversepath } g \equiv (\lambda x. g(1 - x))$

definition *joinpaths* :: $(\text{real} \Rightarrow 'a::\text{topological_space}) \Rightarrow (\text{real} \Rightarrow 'a) \Rightarrow \text{real} \Rightarrow 'a$
(infixr +++ **75)**
where $g1 \text{ +++ } g2 \equiv (\lambda x. \text{if } x \leq 1/2 \text{ then } g1 (2 * x) \text{ else } g2 (2 * x - 1))$

definition *loop_free* :: $(\text{real} \Rightarrow 'a::\text{topological_space}) \Rightarrow \text{bool}$
where $\text{loop_free } g \equiv \forall x \in \{0..1\}. \forall y \in \{0..1\}. g \ x = g \ y \longrightarrow x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$

definition *simple_path* :: $(\text{real} \Rightarrow 'a::\text{topological_space}) \Rightarrow \text{bool}$
where $\text{simple_path } g \equiv \text{path } g \wedge \text{loop_free } g$

definition *arc* :: $(\text{real} \Rightarrow 'a :: \text{topological_space}) \Rightarrow \text{bool}$
where $\text{arc } g \equiv \text{path } g \wedge \text{inj_on } g \ \{0..1\}$

6.1.2 Invariance theorems

lemma *path_eq*: $\text{path } p \Longrightarrow (\bigwedge t. t \in \{0..1\} \Longrightarrow p \ t = q \ t) \Longrightarrow \text{path } q$
using *continuous_on_eq path_def* **by** *blast*

lemma *path_continuous_image*: $\text{path } g \Longrightarrow \text{continuous_on } (\text{path_image } g) \ f \Longrightarrow \text{path}(f \circ g)$
unfolding *path_def path_image_def*
using *continuous_on_compose* **by** *blast*

lemma *path_translation_eq*:
fixes $g :: \text{real} \Rightarrow 'a :: \text{real_normed_vector}$
shows $\text{path}((\lambda x. a + x) \circ g) = \text{path } g$
using *continuous_on_translation_eq path_def* **by** *blast*

lemma *path_linear_image_eq*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *linear f inj f*
shows $\text{path}(f \circ g) = \text{path } g$

proof –

from *linear_injective_left_inverse [OF assms]*

obtain h **where** $h: \text{linear } h \ h \circ f = \text{id}$

by *blast*

with *assms* **show** *?thesis*

by *(metis comp_assoc id_comp linear_continuous_on linear_linear path_continuous_image)*

qed

lemma *pathstart_translation*: $\text{pathstart}((\lambda x. a + x) \circ g) = a + \text{pathstart } g$
by *(simp add: pathstart_def)*

lemma *pathstart_linear_image_eq*: $\text{linear } f \implies \text{pathstart}(f \circ g) = f(\text{pathstart } g)$
by (*simp add: pathstart_def*)

lemma *pathfinish_translation*: $\text{pathfinish}((\lambda x. a + x) \circ g) = a + \text{pathfinish } g$
by (*simp add: pathfinish_def*)

lemma *pathfinish_linear_image*: $\text{linear } f \implies \text{pathfinish}(f \circ g) = f(\text{pathfinish } g)$
by (*simp add: pathfinish_def*)

lemma *path_image_translation*: $\text{path_image}((\lambda x. a + x) \circ g) = (\lambda x. a + x) \text{ ` } (\text{path_image } g)$
by (*simp add: image_comp path_image_def*)

lemma *path_image_linear_image*: $\text{linear } f \implies \text{path_image}(f \circ g) = f \text{ ` } (\text{path_image } g)$
by (*simp add: image_comp path_image_def*)

lemma *reversepath_translation*: $\text{reversepath}((\lambda x. a + x) \circ g) = (\lambda x. a + x) \circ \text{reversepath } g$
by (*rule ext*) (*simp add: reversepath_def*)

lemma *reversepath_linear_image*: $\text{linear } f \implies \text{reversepath}(f \circ g) = f \circ \text{reversepath } g$
by (*rule ext*) (*simp add: reversepath_def*)

lemma *joinpaths_translation*:
 $((\lambda x. a + x) \circ g1) +++ ((\lambda x. a + x) \circ g2) = (\lambda x. a + x) \circ (g1 +++ g2)$
by (*rule ext*) (*simp add: joinpaths_def*)

lemma *joinpaths_linear_image*: $\text{linear } f \implies (f \circ g1) +++ (f \circ g2) = f \circ (g1 +++ g2)$
by (*rule ext*) (*simp add: joinpaths_def*)

lemma *loop_free_translation_eq*:
fixes $g :: \text{real} \Rightarrow 'a::\text{euclidean_space}$
shows $\text{loop_free}((\lambda x. a + x) \circ g) = \text{loop_free } g$
by (*simp add: loop_free_def*)

lemma *simple_path_translation_eq*:
fixes $g :: \text{real} \Rightarrow 'a::\text{euclidean_space}$
shows $\text{simple_path}((\lambda x. a + x) \circ g) = \text{simple_path } g$
by (*simp add: simple_path_def loop_free_translation_eq path_translation_eq*)

lemma *loop_free_linear_image_eq*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $\text{linear } f \text{ inj } f$
shows $\text{loop_free}(f \circ g) = \text{loop_free } g$
using *assms inj_on_eq_iff* [*of f*] **by** (*auto simp: loop_free_def*)

lemma *simple_path_linear_image_eq*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes *linear f inj f*
shows $simple_path(f \circ g) = simple_path\ g$
using *assms*
by (*simp add: loop_free_linear_image_eq path_linear_image_eq simple_path_def*)

lemma *simple_pathI* [*intro?*]:
assumes *path p*
assumes $\bigwedge x\ y. 0 \leq x \implies x < y \implies y \leq 1 \implies p\ x = p\ y \implies x = 0 \wedge y = 1$
shows *simple_path p*
unfolding *simple_path_def loop_free_def*
proof (*intro ballI conjI impI*)
fix $x\ y$ **assume** $x \in \{0..1\}\ y \in \{0..1\}\ p\ x = p\ y$
thus $x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$
by (*metis assms(2) atLeastAtMost_iff linorder_less_linear*)
qed *fact+*

lemma *arcD*: $arc\ p \implies p\ x = p\ y \implies x \in \{0..1\} \implies y \in \{0..1\} \implies x = y$
by (*auto simp: arc_def inj_on_def*)

lemma *arc_translation_eq*:
fixes $g :: real \Rightarrow 'a::euclidean_space$
shows $arc((\lambda x. a + x) \circ g) \longleftrightarrow arc\ g$
by (*auto simp: arc_def inj_on_def path_translation_eq*)

lemma *arc_linear_image_eq*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes *linear f inj f*
shows $arc(f \circ g) = arc\ g$
using *assms inj_on_eq_iff [of f]*
by (*auto simp: arc_def inj_on_def path_linear_image_eq*)

6.1.3 Basic lemmas about paths

lemma *path_of_real: path complex_of_real*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_const: path* $(\lambda t. a)$ **for** $a::'a::real_normed_vector$
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_minus: path* $g \implies path\ (\lambda t. -\ g\ t)$ **for** $g::real \Rightarrow 'a::real_normed_vector$
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_add: [path f; path g] \implies path* $(\lambda t. f\ t + g\ t)$ **for** $f::real \Rightarrow 'a::real_normed_vector$
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_diff: [path f; path g] \implies path* $(\lambda t. f\ t - g\ t)$ **for** $f::real \Rightarrow 'a::real_normed_vector$

unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_mult*: $\llbracket \text{path } f; \text{ path } g \rrbracket \implies \text{path } (\lambda t. f t * g t)$ **for** $f :: \text{real} \Rightarrow 'a :: \text{real_normed_field}$
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *pathin_iff_path_real* [*simp*]: $\text{pathin euclideanreal } g \longleftrightarrow \text{path } g$
by (*simp add: pathin_def path_def*)

lemma *continuous_on_path*: $\text{path } f \implies t \subseteq \{0..1\} \implies \text{continuous_on } t f$
using *continuous_on_subset path_def* **by** *blast*

lemma *inj_on_imp_loop_free*: $\text{inj_on } g \{0..1\} \implies \text{loop_free } g$
by (*simp add: inj_onD loop_free_def*)

lemma *arc_imp_simple_path*: $\text{arc } g \implies \text{simple_path } g$
by (*simp add: arc_def inj_on_imp_loop_free simple_path_def*)

lemma *arc_imp_path*: $\text{arc } g \implies \text{path } g$
using *arc_def* **by** *blast*

lemma *arc_imp_inj_on*: $\text{arc } g \implies \text{inj_on } g \{0..1\}$
by (*auto simp: arc_def*)

lemma *simple_path_imp_path*: $\text{simple_path } g \implies \text{path } g$
using *simple_path_def* **by** *blast*

lemma *loop_free_cases*: $\text{loop_free } g \implies \text{inj_on } g \{0..1\} \vee \text{pathfinish } g = \text{pathstart } g$
by (*force simp: inj_on_def loop_free_def pathfinish_def pathstart_def*)

lemma *simple_path_cases*: $\text{simple_path } g \implies \text{arc } g \vee \text{pathfinish } g = \text{pathstart } g$
using *arc_def loop_free_cases simple_path_def* **by** *blast*

lemma *simple_path_imp_arc*: $\text{simple_path } g \implies \text{pathfinish } g \neq \text{pathstart } g \implies \text{arc } g$
using *simple_path_cases* **by** *auto*

lemma *arc_distinct_ends*: $\text{arc } g \implies \text{pathfinish } g \neq \text{pathstart } g$
unfolding *arc_def inj_on_def pathfinish_def pathstart_def*
by *fastforce*

lemma *arc_simple_path*: $\text{arc } g \longleftrightarrow \text{simple_path } g \wedge \text{pathfinish } g \neq \text{pathstart } g$
using *arc_distinct_ends arc_imp_simple_path simple_path_cases* **by** *blast*

lemma *simple_path_eq_arc*: $\text{pathfinish } g \neq \text{pathstart } g \implies (\text{simple_path } g = \text{arc } g)$
by (*simp add: arc_simple_path*)

lemma *path_image_const* [*simp*]: $\text{path_image } (\lambda t. a) = \{a\}$

```

by (force simp: path_image_def)

lemma path_image_nonempty [simp]: path_image g ≠ {}
  unfolding path_image_def image_is_empty box_eq_empty
  by auto

lemma pathstart_in_path_image[intro]: pathstart g ∈ path_image g
  unfolding pathstart_def path_image_def
  by auto

lemma pathfinish_in_path_image[intro]: pathfinish g ∈ path_image g
  unfolding pathfinish_def path_image_def
  by auto

lemma connected_path_image[intro]: path g ⇒ connected (path_image g)
  unfolding path_def path_image_def
  using connected_continuous_image connected_Icc by blast

lemma compact_path_image[intro]: path g ⇒ compact (path_image g)
  unfolding path_def path_image_def
  using compact_continuous_image connected_Icc by blast

lemma reversepath_reversepath[simp]: reversepath (reversepath g) = g
  unfolding reversepath_def
  by auto

lemma pathstart_reversepath[simp]: pathstart (reversepath g) = pathfinish g
  unfolding pathstart_def reversepath_def pathfinish_def
  by auto

lemma pathfinish_reversepath[simp]: pathfinish (reversepath g) = pathstart g
  unfolding pathstart_def reversepath_def pathfinish_def
  by auto

lemma reversepath_o: reversepath g = g ∘ (-)1
  by (auto simp: reversepath_def)

lemma pathstart_join[simp]: pathstart (g1 +++ g2) = pathstart g1
  unfolding pathstart_def joinpaths_def pathfinish_def
  by auto

lemma pathfinish_join[simp]: pathfinish (g1 +++ g2) = pathfinish g2
  unfolding pathstart_def joinpaths_def pathfinish_def
  by auto

lemma path_image_reversepath[simp]: path_image (reversepath g) = path_image
g
  by (metis cancel_comm_monoid_add_class.diff_cancel diff_zero image_comp
image_diff_atLeastAtMost path_image_def reversepath_o)

```

```

lemma path_reversepath [simp]: path (reversepath g)  $\longleftrightarrow$  path g
  by (metis continuous_on_compose continuous_on_op_minus image_comp image_ident path_def path_image_def path_image_reversepath reversepath_o reversepath_reversepath)

lemma arc_reversepath:
  assumes arc g shows arc(reversepath g)
proof -
  have injg: inj_on g {0..1}
    using assms
    by (simp add: arc_def)
  have **:  $\bigwedge x y::\text{real}. 1-x = 1-y \implies x = y$ 
    by simp
  show ?thesis
    using assms by (clarsimp simp: arc_def intro!: inj_onI) (simp add: inj_onD reversepath_def **)
qed

lemma loop_free_reversepath:
  assumes loop_free g shows loop_free(reversepath g)
  using assms by (simp add: reversepath_def loop_free_def Ball_def) (smt (verit))

lemma simple_path_reversepath: simple_path g  $\implies$  simple_path (reversepath g)
  by (simp add: loop_free_reversepath simple_path_def)

lemmas reversepath_simps =
  path_reversepath path_image_reversepath pathstart_reversepath pathfinish_reversepath

lemma path_join[simp]:
  assumes pathfinish g1 = pathstart g2
  shows path (g1 +++ g2)  $\longleftrightarrow$  path g1  $\wedge$  path g2
  unfolding path_def pathfinish_def pathstart_def
proof safe
  assume cont: continuous_on {0..1} (g1 +++ g2)
  have g1: continuous_on {0..1} g1  $\longleftrightarrow$  continuous_on {0..1} ((g1 +++ g2)  $\circ$  ( $\lambda x. x / 2$ ))
    by (intro continuous_on_cong refl) (auto simp: joinpaths_def)
  have g2: continuous_on {0..1} g2  $\longleftrightarrow$  continuous_on {0..1} ((g1 +++ g2)  $\circ$  ( $\lambda x. x / 2 + 1/2$ ))
    using assms
    by (intro continuous_on_cong refl) (auto simp: joinpaths_def pathfinish_def pathstart_def)
  show continuous_on {0..1} g1 and continuous_on {0..1} g2
  unfolding g1 g2
    by (auto intro!: continuous_intros continuous_on_subset[OF cont] simp del: o_apply)
next
  assume g1g2: continuous_on {0..1} g1 continuous_on {0..1} g2

```

```

have 01: {0 .. 1} = {0..1/2} ∪ {1/2 .. 1::real}
  by auto
{
  fix x :: real
  assume 0 ≤ x and x ≤ 1
  then have x ∈ (λx. x * 2) ‘ {0..1 / 2}
    by (intro image_eqI[where x=x/2]) auto
}
note 1 = this
{
  fix x :: real
  assume 0 ≤ x and x ≤ 1
  then have x ∈ (λx. x * 2 - 1) ‘ {1 / 2..1}
    by (intro image_eqI[where x=x/2 + 1/2]) auto
}
note 2 = this
show continuous_on {0..1} (g1 +++ g2)
  using assms
  unfolding joinpaths_def 01
  apply (intro continuous_on_cases closed_atLeastAtMost g1g2[THEN contin-
uous_on_compose2] continuous_intros)
  apply (auto simp: field_simps pathfinish_def pathstart_def intro!: 1 2)
  done
qed

```

6.1.4 Path Images

lemma *bounded_path_image*: $\text{path } g \implies \text{bounded}(\text{path_image } g)$
 by (simp add: compact_imp_bounded compact_path_image)

lemma *closed_path_image*:
 fixes $g :: \text{real} \Rightarrow 'a::t2_space$
 shows $\text{path } g \implies \text{closed}(\text{path_image } g)$
 by (metis compact_path_image compact_imp_closed)

lemma *connected_simple_path_image*: $\text{simple_path } g \implies \text{connected}(\text{path_image } g)$
 by (metis connected_path_image simple_path_imp_path)

lemma *compact_simple_path_image*: $\text{simple_path } g \implies \text{compact}(\text{path_image } g)$
 by (metis compact_path_image simple_path_imp_path)

lemma *bounded_simple_path_image*: $\text{simple_path } g \implies \text{bounded}(\text{path_image } g)$
 by (metis bounded_path_image simple_path_imp_path)

lemma *closed_simple_path_image*:
 fixes $g :: \text{real} \Rightarrow 'a::t2_space$
 shows $\text{simple_path } g \implies \text{closed}(\text{path_image } g)$
 by (metis closed_path_image simple_path_imp_path)

lemma *connected_arc_image*: $\text{arc } g \implies \text{connected}(\text{path_image } g)$
by (*metis connected_path_image arc_imp_path*)

lemma *compact_arc_image*: $\text{arc } g \implies \text{compact}(\text{path_image } g)$
by (*metis compact_path_image arc_imp_path*)

lemma *bounded_arc_image*: $\text{arc } g \implies \text{bounded}(\text{path_image } g)$
by (*metis bounded_path_image arc_imp_path*)

lemma *closed_arc_image*:
fixes $g :: \text{real} \Rightarrow 'a::t2_space$
shows $\text{arc } g \implies \text{closed}(\text{path_image } g)$
by (*metis closed_path_image arc_imp_path*)

lemma *path_image_join_subset*: $\text{path_image } (g1 +++ g2) \subseteq \text{path_image } g1 \cup \text{path_image } g2$
unfolding *path_image_def joinpaths_def*
by *auto*

lemma *subset_path_image_join*:
assumes $\text{path_image } g1 \subseteq S$ **and** $\text{path_image } g2 \subseteq S$
shows $\text{path_image } (g1 +++ g2) \subseteq S$
using *path_image_join_subset[of g1 g2]* **and** *assms*
by *auto*

lemma *path_image_join*:
assumes $\text{pathfinish } g1 = \text{pathstart } g2$
shows $\text{path_image}(g1 +++ g2) = \text{path_image } g1 \cup \text{path_image } g2$
proof –
have $\text{path_image } g1 \subseteq \text{path_image } (g1 +++ g2)$
proof (*clarsimp simp: path_image_def joinpaths_def*)
fix $u::\text{real}$
assume $0 \leq u \leq 1$
then show $g1 \ u \in (\lambda x. g1 \ (2 * x)) \ '(\{0..1\} \cap \{x. x * 2 \leq 1\})$
by (*rule_tac x=u/2 in image_eqI*) *auto*
qed
moreover
have $g2 \ u \in (\lambda x. g2 \ (2 * x - 1)) \ '(\{0..1\} \cap \{x. \neg x * 2 \leq 1\})$
if $0 < u \leq 1$ **for** u
using *that assms*
by (*rule_tac x=(u+1)/2 in image_eqI*) (*auto simp: field_simps pathfinish_def pathstart_def*)
have $g2 \ 0 \in (\lambda x. g1 \ (2 * x)) \ '(\{0..1\} \cap \{x. x * 2 \leq 1\})$
using *assms*
by (*rule_tac x=1/2 in image_eqI*) (*auto simp: pathfinish_def pathstart_def*)
then have $\text{path_image } g2 \subseteq \text{path_image } (g1 +++ g2)$
by (*auto simp: path_image_def joinpaths_def intro!: §*)
ultimately show *?thesis*

using path_image_join_subset by blast
qed

lemma not_in_path_image_join:
assumes $x \notin \text{path_image } g1$ and $x \notin \text{path_image } g2$
shows $x \notin \text{path_image } (g1 +++ g2)$
using assms and path_image_join_subset[of $g1$ $g2$]
by auto

lemma pathstart_compose: $\text{pathstart}(f \circ p) = f(\text{pathstart } p)$
by (simp add: pathstart_def)

lemma pathfinish_compose: $\text{pathfinish}(f \circ p) = f(\text{pathfinish } p)$
by (simp add: pathfinish_def)

lemma path_image_compose: $\text{path_image } (f \circ p) = f' (\text{path_image } p)$
by (simp add: image_comp path_image_def)

lemma path_compose_join: $f \circ (p +++ q) = (f \circ p) +++ (f \circ q)$
by (rule ext) (simp add: joinpaths_def)

lemma path_compose_reversepath: $f \circ \text{reversepath } p = \text{reversepath}(f \circ p)$
by (rule ext) (simp add: reversepath_def)

lemma joinpaths_eq:
 $(\bigwedge t. t \in \{0..1\} \implies p \ t = p' \ t) \implies$
 $(\bigwedge t. t \in \{0..1\} \implies q \ t = q' \ t)$
 $\implies t \in \{0..1\} \implies (p +++ q) \ t = (p' +++ q') \ t$
by (auto simp: joinpaths_def)

lemma loop_free_inj_on: $\text{loop_free } g \implies \text{inj_on } g \ \{0 <..< 1\}$
by (force simp: inj_on_def loop_free_def)

lemma simple_path_inj_on: $\text{simple_path } g \implies \text{inj_on } g \ \{0 <..< 1\}$
using loop_free_inj_on simple_path_def by auto

6.1.5 Simple paths with the endpoints removed

lemma simple_path_endless:
assumes simple_path c
shows $\text{path_image } c - \{\text{pathstart } c, \text{pathfinish } c\} = c' \ \{0 <..< 1\}$ (is ?lhs =
?rhs)

proof

show ?lhs \subseteq ?rhs

using less_eq_real_def by (auto simp: path_image_def pathstart_def pathfin-

ish_def)

show ?rhs \subseteq ?lhs

using assms

apply (simp add: image_subset_iff path_image_def pathstart_def pathfin-

```

ish_def simple_path_def loop_free_def Ball_def)
  by (smt (verit))
qed

```

```

lemma connected_simple_path_endless:
  assumes simple_path c
  shows connected(path_image c - {pathstart c,pathfinish c})
proof -
  have continuous_on {0<..<1} c
    using assms by (simp add: simple_path_def continuous_on_path path_def
subset_iff)
  then have connected (c ' {0<..<1})
    using connected_Ioo connected_continuous_image by blast
  then show ?thesis
    using assms by (simp add: simple_path_endless)
qed

```

```

lemma nonempty_simple_path_endless:
  simple_path c  $\implies$  path_image c - {pathstart c,pathfinish c}  $\neq$  {}
  by (simp add: simple_path_endless)

```

```

lemma simple_path_continuous_image:
  assumes simple_path f continuous_on (path_image f) g inj_on g (path_image
f)
  shows simple_path (g  $\circ$  f)
  unfolding simple_path_def
proof
  show path (g  $\circ$  f)
    using assms unfolding simple_path_def by (intro path_continuous_image)
  auto
  from assms have [simp]: g (f x) = g (f y)  $\iff$  f x = f y if x  $\in$  {0..1} y  $\in$  {0..1}
  for x y
    unfolding inj_on_def path_image_def using that by fastforce
  show loop_free (g  $\circ$  f)
    using assms(1) by (auto simp: loop_free_def simple_path_def)
qed

```

6.1.6 The operations on paths

```

lemma path_image_subset_reversepath: path_image(reversepath g)  $\leq$  path_image
g
  by simp

```

```

lemma path_imp_reversepath: path g  $\implies$  path(reversepath g)
  by simp

```

```

lemma half_bounded_equal: 1  $\leq$  x * 2  $\implies$  x * 2  $\leq$  1  $\iff$  x = (1/2::real)
  by simp

```

lemma *continuous_on_joinpaths*:

assumes *continuous_on* $\{0..1\}$ $g1$ *continuous_on* $\{0..1\}$ $g2$ *pathfinish* $g1 =$
pathstart $g2$
shows *continuous_on* $\{0..1\}$ $(g1 +++ g2)$
using *assms path_def path_join* **by** *blast*

lemma *path_join_imp*: $\llbracket \text{path } g1; \text{path } g2; \text{pathfinish } g1 = \text{pathstart } g2 \rrbracket \implies$
 $\text{path}(g1 +++ g2)$
by *simp*

lemma *arc_join*:

assumes *arc* $g1$ *arc* $g2$
pathfinish $g1 = \text{pathstart } g2$
path_image $g1 \cap \text{path_image } g2 \subseteq \{\text{pathstart } g2\}$
shows *arc* $(g1 +++ g2)$

proof –

have *injg1*: *inj_on* $g1$ $\{0..1\}$ **and** *injg2*: *inj_on* $g2$ $\{0..1\}$
and *g11*: $g1$ $1 = g2$ 0 **and** *sb*: $g1$ ‘ $\{0..1\} \cap g2$ ‘ $\{0..1\} \subseteq \{g2$ $0\}$
using *assms*
by (*auto simp: arc_def pathfinish_def pathstart_def path_image_def*)
{ fix x **and** y :*real*
assume xy : $g2$ $(2 * x - 1) = g1$ $(2 * y) x \leq 1$ $0 \leq y$ $y * 2 \leq 1$ $\neg x * 2 \leq 1$
then have $g1$ $(2 * y) = g2$ 0
using *sb* **by** *force*
then have *False*
using xy *inj_onD injg2* **by** *fastforce*
} **note** $*$ = *this*
have *inj_on* $(g1 +++ g2)$ $\{0..1\}$
using *inj_onD [OF injg1] inj_onD [OF injg2] **
by (*simp add: inj_on_def joinpaths_def Ball_def*) (*smt (verit)*)
then show *?thesis*
using *arc_def assms path_join_imp* **by** *blast*

qed

lemma *simple_path_join_loop*:

assumes *arc* $g1$ *arc* $g2$
pathfinish $g1 = \text{pathstart } g2$ *pathfinish* $g2 = \text{pathstart } g1$
path_image $g1 \cap \text{path_image } g2 \subseteq \{\text{pathstart } g1, \text{pathstart } g2\}$
shows *simple_path* $(g1 +++ g2)$

proof –

have *injg1*: *inj_on* $g1$ $\{0..1\}$ **and** *injg2*: *inj_on* $g2$ $\{0..1\}$
using *assms* **by** (*auto simp add: arc_def*)
have *g12*: $g1$ $1 = g2$ 0
and *g21*: $g2$ $1 = g1$ 0
and *sb*: $g1$ ‘ $\{0..1\} \cap g2$ ‘ $\{0..1\} \subseteq \{g1$ $0, g2$ $0\}$
using *assms*
by (*simp_all add: arc_def pathfinish_def pathstart_def path_image_def*)
{ fix x **and** y :*real*
assume *g2_eq*: $g2$ $(2 * x - 1) = g1$ $(2 * y)$

```

    and  $xyI: x \neq 1 \vee y \neq 0$ 
    and  $xy: x \leq 1 \ 0 \leq y \ y * 2 \leq 1 \ \neg x * 2 \leq 1$ 
  then consider  $g1 \ (2 * y) = g1 \ 0 \mid g1 \ (2 * y) = g2 \ 0$ 
    using sb by force
  then have False
  proof cases
    case 1
      then have  $y = 0$ 
        using  $xy \ g2\_eq$  by (auto dest!: inj_onD [OF injg1])
      then show ?thesis
        using  $xy \ g2\_eq \ xyI$  by (auto dest: inj_onD [OF injg2] simp flip: g21)
    next
      case 2
        then have  $2*x = 1$ 
          using  $g2\_eq \ g12 \ inj\_onD \ [OF \ injg2] \ atLeastAtMost\_iff \ xy(1) \ xy(4)$  by
fastforce
        with  $xy$  show False by auto
      qed
    } note * = this
  have loop_free( $g1 \ +++ \ g2$ )
    using inj_onD [OF injg1] inj_onD [OF injg2] *
  by (simp add: loop_free_def joinpaths_def Ball_def) (smt (verit))
  then show ?thesis
  by (simp add: arc_imp_path assms simple_path_def)
qed

```

lemma *reversepath_joinpaths*:

```

  pathfinish  $g1 = pathstart \ g2 \implies reversepath(g1 \ +++ \ g2) = reversepath \ g2$ 
+++ reversepath  $g1$ 
  unfolding reversepath_def pathfinish_def pathstart_def joinpaths_def
  by (rule ext) (auto simp: mult.commute)

```

6.1.7 Some reversed and "if and only if" versions of joining theorems

lemma *path_join_path_ends*:

```

  fixes  $g1 :: real \Rightarrow 'a::metric\_space$ 
  assumes  $path(g1 \ +++ \ g2) \ path \ g2$ 
  shows  $pathfinish \ g1 = pathstart \ g2$ 
  proof (rule ccontr)
    define  $e$  where  $e = dist \ (g1 \ 1) \ (g2 \ 0)$ 
    assume Neg:  $pathfinish \ g1 \neq pathstart \ g2$ 
    then have  $0 < dist \ (pathfinish \ g1) \ (pathstart \ g2)$ 
      by auto
    then have  $e > 0$ 
      by (metis  $e\_def \ pathfinish\_def \ pathstart\_def$ )
    then have  $\forall e > 0. \exists d > 0. \forall x' \in \{0..1\}. dist \ x' \ 0 < d \implies dist \ (g2 \ x') \ (g2 \ 0) < e$ 
      using  $\langle path \ g2 \rangle \ atLeastAtMost\_iff \ zero\_le\_one$  unfolding  $path\_def \ continuous\_on\_iff$ 

```

```

  by blast
  then obtain d1 where d1 > 0
    and d1:  $\bigwedge x'. \llbracket x' \in \{0..1\}; \text{norm } x' < d1 \rrbracket \implies \text{dist } (g2 \ x') \ (g2 \ 0) < e/2$ 
    by (metis <0 < e> half_gt_zero_iff norm_conv_dist)
  obtain d2 where d2 > 0
    and d2:  $\bigwedge x'. \llbracket x' \in \{0..1\}; \text{dist } x' \ (1/2) < d2 \rrbracket$ 
       $\implies \text{dist } ((g1 \ +++ \ g2) \ x') \ (g1 \ 1) < e/2$ 
    using assms(1) <e > 0> unfolding path_def continuous_on_iff
    apply (drule_tac x=1/2 in bspec, simp)
    apply (drule_tac x=e/2 in spec, force simp: joinpaths_def)
  done
  have int01_1:  $\min \ (1/2) \ (\min \ d1 \ d2) / 2 \in \{0..1\}$ 
    using <d1 > 0> <d2 > 0> by (simp add: min_def)
  have dist1:  $\text{norm } (\min \ (1 / 2) \ (\min \ d1 \ d2) / 2) < d1$ 
    using <d1 > 0> <d2 > 0> by (simp add: min_def dist_norm)
  have int01_2:  $1/2 + \min \ (1/2) \ (\min \ d1 \ d2) / 4 \in \{0..1\}$ 
    using <d1 > 0> <d2 > 0> by (simp add: min_def)
  have dist2:  $\text{dist } (1 / 2 + \min \ (1 / 2) \ (\min \ d1 \ d2) / 4) \ (1 / 2) < d2$ 
    using <d1 > 0> <d2 > 0> by (simp add: min_def dist_norm)
  have [simp]:  $\neg \min \ (1 / 2) \ (\min \ d1 \ d2) \leq 0$ 
    using <d1 > 0> <d2 > 0> by (simp add: min_def)
  have dist (g2 (min (1 / 2) (min d1 d2) / 2)) (g1 1) < e/2
    dist (g2 (min (1 / 2) (min d1 d2) / 2)) (g2 0) < e/2
    using d1 [OF int01_1 dist1] d2 [OF int01_2 dist2] by (simp_all add: join-
paths_def)
  then have dist (g1 1) (g2 0) < e/2 + e/2
    using dist_triangle_half_r e_def by blast
  then show False
    by (simp add: e_def [symmetric])
qed

```

```

lemma path_join_eq [simp]:
  fixes g1 :: real  $\Rightarrow$  'a::metric_space
  assumes path g1 path g2
  shows path(g1 +++ g2)  $\longleftrightarrow$  pathfinish g1 = pathstart g2
  using assms by (metis path_join_path_ends path_join_imp)

```

```

lemma simple_path_joinE:
  assumes simple_path(g1 +++ g2) and pathfinish g1 = pathstart g2
  obtains arc g1 arc g2
    path_image g1  $\cap$  path_image g2  $\subseteq$  {pathstart g1, pathstart g2}

```

proof –

```

  have *:  $\bigwedge x \ y. \llbracket 0 \leq x; x \leq 1; 0 \leq y; y \leq 1; (g1 \ +++ \ g2) \ x = (g1 \ +++ \ g2) \ y \rrbracket$ 
     $\implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ 

```

```

  using assms by (simp add: simple_path_def loop_free_def)

```

```

  have path g1

```

```

    using assms path_join simple_path_imp_path by blast

```

```

  moreover have inj_on g1 {0..1}

```

```

  proof (clarsimp simp: inj_on_def)

```

```

    fix x y
    assume g1 x = g1 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1
    then show x = y
      using * [of x/2 y/2] by (simp add: joinpaths_def split_ifs)
    qed
    ultimately have arc g1
      using assms by (simp add: arc_def)
    have [simp]: g2 0 = g1 1
      using assms by (metis pathfinish_def pathstart_def)
    have path g2
      using assms path_join simple_path_imp_path by blast
    moreover have inj_on g2 {0..1}
    proof (clarsimp simp: inj_on_def)
      fix x y
      assume g2 x = g2 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1
      then show x = y
        using * [of (x+1) / 2 (y+1) / 2]
        by (force simp: joinpaths_def split_ifs field_split_simps)
    qed
    ultimately have arc g2
      using assms by (simp add: arc_def)
    have g2 y = g1 0 ∨ g2 y = g1 1
      if g1 x = g2 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1 for x y
      using * [of x / 2 (y + 1) / 2] that
      by (auto simp: joinpaths_def split_ifs field_split_simps)
    then have path_image g1 ∩ path_image g2 ⊆ {pathstart g1, pathstart g2}
      by (fastforce simp: pathstart_def pathfinish_def path_image_def)
    with ⟨arc g1⟩ ⟨arc g2⟩ show ?thesis using that by blast
  qed

lemma simple_path_join_loop_eq:
  assumes pathfinish g2 = pathstart g1 pathfinish g1 = pathstart g2
  shows simple_path(g1 +++ g2) ↔
    arc g1 ∧ arc g2 ∧ path_image g1 ∩ path_image g2 ⊆ {pathstart g1,
pathstart g2}
  by (metis assms simple_path_joinE simple_path_join_loop)

lemma arc_join_eq:
  assumes pathfinish g1 = pathstart g2
  shows arc(g1 +++ g2) ↔
    arc g1 ∧ arc g2 ∧ path_image g1 ∩ path_image g2 ⊆ {pathstart g2}
    (is ?lhs = ?rhs)

proof
  assume ?lhs then show ?rhs
    using reversepath_simps assms
    by (smt (verit, best) Int_commute arc_reversepath arc_simple_path in_mono
insertE pathstart_join
reversepath_joinpaths simple_path_joinE subsetI)
next

```

```

assume ?rhs then show ?lhs
  using assms
  by (fastforce simp: pathfinish_def pathstart_def intro!: arc_join)
qed

```

```

lemma arc_join_eq_alt:
  pathfinish g1 = pathstart g2
   $\implies \text{arc}(g1 \text{ +++ } g2) \longleftrightarrow \text{arc } g1 \wedge \text{arc } g2 \wedge \text{path\_image } g1 \cap \text{path\_image } g2$ 
   $= \{\text{pathstart } g2\}$ 
  using pathfinish_in_path_image by (fastforce simp: arc_join_eq)

```

Symmetry and loops

```

lemma path_sym:
   $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket \implies \text{path}(p \text{ +++ } q) \longleftrightarrow$ 
   $\text{path}(q \text{ +++ } p)$ 
  by auto

```

```

lemma simple_path_sym:
   $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket$ 
   $\implies \text{simple\_path}(p \text{ +++ } q) \longleftrightarrow \text{simple\_path}(q \text{ +++ } p)$ 
  by (metis (full_types) inf_commute insert_commute simple_path_joinE simple_path_join_loop)

```

```

lemma path_image_sym:
   $\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } p \rrbracket$ 
   $\implies \text{path\_image}(p \text{ +++ } q) = \text{path\_image}(q \text{ +++ } p)$ 
  by (simp add: path_image_join sup_commute)

```

```

lemma simple_path_joinI:
  assumes arc p1 arc p2 pathfinish p1 = pathstart p2
  assumes path_image p1  $\cap$  path_image p2  $\subseteq$  {pathstart p2}
   $\subseteq \text{insert}(\text{pathstart } p2) (\text{if } \text{pathstart } p1 = \text{pathfinish } p2 \text{ then } \{\text{pathstart } p1\}$ 
  else  $\{\}$ )
  shows simple_path (p1 +++ p2)
  by (smt (verit, best) Int_commute arc_imp_simple_path arc_join assms insert_commute simple_path_join_loop simple_path_sym)

```

```

lemma simple_path_join3I:
  assumes arc p1 arc p2 arc p3
  assumes path_image p1  $\cap$  path_image p2  $\subseteq$  {pathstart p2}
  assumes path_image p2  $\cap$  path_image p3  $\subseteq$  {pathstart p3}
  assumes path_image p1  $\cap$  path_image p3  $\subseteq$  {pathstart p1}  $\cap$  {pathfinish p3}
  assumes pathfinish p1 = pathstart p2 pathfinish p2 = pathstart p3
  shows simple_path (p1 +++ p2 +++ p3)
  using assms by (intro simple_path_joinI arc_join) (auto simp: path_image_join)

```

6.1.8 The joining of paths is associative

```

lemma path_assoc:

```

$\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } r \rrbracket$
 $\implies \text{path}(p \text{ +++ } (q \text{ +++ } r)) \longleftrightarrow \text{path}((p \text{ +++ } q) \text{ +++ } r)$
 by *simp*

lemma *simple_path_assoc*:

assumes $\text{pathfinish } p = \text{pathstart } q$ $\text{pathfinish } q = \text{pathstart } r$
shows $\text{simple_path } (p \text{ +++ } (q \text{ +++ } r)) \longleftrightarrow \text{simple_path } ((p \text{ +++ } q) \text{ +++ } r)$

proof (*cases pathstart p = pathfinish r*)

case *True* **show** *?thesis*

proof

assume $\text{simple_path } (p \text{ +++ } q \text{ +++ } r)$

with *assms True* **show** $\text{simple_path } ((p \text{ +++ } q) \text{ +++ } r)$

by (*fastforce simp add: simple_path_join_loop_eq arc_join_eq path_image_join*
dest: arc_distinct_ends [of r])

next

assume *0*: $\text{simple_path } ((p \text{ +++ } q) \text{ +++ } r)$

with *assms True* **have** $q: \text{pathfinish } r \notin \text{path_image } q$

using *arc_distinct_ends*

by (*fastforce simp add: simple_path_join_loop_eq arc_join_eq path_image_join*)

have $\text{pathstart } r \notin \text{path_image } p$

using *assms*

by (*metis 0 IntI arc_distinct_ends arc_join_eq_alt empty_iff insert_iff*
pathfinish_in_path_image pathfinish_join simple_path_joinE)

with *assms 0 q True* **show** $\text{simple_path } (p \text{ +++ } q \text{ +++ } r)$

by (*auto simp: simple_path_join_loop_eq arc_join_eq path_image_join*
dest!: subsetD [OF _ IntI])

qed

next

case *False*

{ **fix** $x :: 'a$

assume $a: \text{path_image } p \cap \text{path_image } q \subseteq \{\text{pathstart } q\}$

$(\text{path_image } p \cup \text{path_image } q) \cap \text{path_image } r \subseteq \{\text{pathstart } r\}$

$x \in \text{path_image } p \implies x \in \text{path_image } r$

have $\text{pathstart } r \in \text{path_image } q$

by (*metis assms(2) pathfinish_in_path_image*)

with *a* **have** $x = \text{pathstart } q$

by *blast*

}

with *False assms* **show** *?thesis*

by (*auto simp: simple_path_eq_arc simple_path_join_loop_eq arc_join_eq*
path_image_join)

qed

lemma *arc_assoc*:

$\llbracket \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } r \rrbracket$

$\implies \text{arc}(p \text{ +++ } (q \text{ +++ } r)) \longleftrightarrow \text{arc}((p \text{ +++ } q) \text{ +++ } r)$

by (*simp add: arc_simple_path simple_path_assoc*)

6.1.9 Subpath

definition *subpath* :: $real \Rightarrow real \Rightarrow (real \Rightarrow 'a) \Rightarrow real \Rightarrow 'a::real_normed_vector$
where *subpath* *a b g* $\equiv \lambda x. g((b - a) * x + a)$

lemma *path_image_subpath_gen*:
fixes *g* :: $_ \Rightarrow 'a::real_normed_vector$
shows $path_image(subpath\ u\ v\ g) = g\ ' (closed_segment\ u\ v)$
by (*auto simp add: closed_segment_real_eq path_image_def subpath_def*)

lemma *path_image_subpath*:
fixes *g* :: $real \Rightarrow 'a::real_normed_vector$
shows $path_image(subpath\ u\ v\ g) = (if\ u \leq v\ then\ g\ '\{u..v\}\ else\ g\ '\{v..u\})$
by (*simp add: path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_image_subpath_commute*:
fixes *g* :: $real \Rightarrow 'a::real_normed_vector$
shows $path_image(subpath\ u\ v\ g) = path_image(subpath\ v\ u\ g)$
by (*simp add: path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_subpath [simp]*:
fixes *g* :: $real \Rightarrow 'a::real_normed_vector$
assumes $path\ g\ u \in \{0..1\}\ v \in \{0..1\}$
shows $path(subpath\ u\ v\ g)$
proof –
have $continuous_on\ \{u..v\}\ g\ continuous_on\ \{v..u\}\ g$
using *assms continuous_on_path by fastforce+*
then have $continuous_on\ \{0..1\}\ (g \circ (\lambda x. ((v-u) * x + u)))$
by (*intro continuous_intros; simp add: image_affinity_atLeastAtMost [where c=u]*)
then show *?thesis*
by (*simp add: path_def subpath_def*)
qed

lemma *pathstart_subpath [simp]*: $pathstart(subpath\ u\ v\ g) = g(u)$
by (*simp add: pathstart_def subpath_def*)

lemma *pathfinish_subpath [simp]*: $pathfinish(subpath\ u\ v\ g) = g(v)$
by (*simp add: pathfinish_def subpath_def*)

lemma *subpath_trivial [simp]*: $subpath\ 0\ 1\ g = g$
by (*simp add: subpath_def*)

lemma *subpath_reversepath*: $subpath\ 1\ 0\ g = reversepath\ g$
by (*simp add: reversepath_def subpath_def*)

lemma *reversepath_subpath*: $reversepath(subpath\ u\ v\ g) = subpath\ v\ u\ g$
by (*simp add: reversepath_def subpath_def algebra_simps*)

lemma *subpath_translation*: $subpath\ u\ v\ ((\lambda x. a + x) \circ g) = (\lambda x. a + x) \circ subpath$

$u\ v\ g$

by (*rule ext*) (*simp add: subpath_def*)

lemma *subpath_image*: $\text{subpath } u\ v\ (f \circ g) = f \circ \text{subpath } u\ v\ g$

by (*rule ext*) (*simp add: subpath_def*)

lemma *affine_ineq*:

fixes $x :: 'a::\text{linordered_idom}$

assumes $x \leq 1\ v \leq u$

shows $v + x * u \leq u + x * v$

proof –

have $(1-x)*(u-v) \geq 0$

using *assms* **by** *auto*

then show *?thesis*

by (*simp add: algebra_simps*)

qed

lemma *sum_le_prod1*:

fixes $a::\text{real}$ **shows** $\llbracket a \leq 1; b \leq 1 \rrbracket \implies a + b \leq 1 + a * b$

by (*metis add.commute affine_ineq mult.right_neutral*)

lemma *simple_path_subpath_eq*:

$\text{simple_path}(\text{subpath } u\ v\ g) \longleftrightarrow$

$\text{path}(\text{subpath } u\ v\ g) \wedge u \neq v \wedge$

$(\forall x\ y. x \in \text{closed_segment } u\ v \wedge y \in \text{closed_segment } u\ v \wedge g\ x = g\ y$

$\longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u)$

(*is ?lhs = ?rhs*)

proof

assume *?lhs*

then have $p: \text{path } (\lambda x. g ((v - u) * x + u))$

and $\text{sim}: (\bigwedge x\ y. \llbracket x \in \{0..1\}; y \in \{0..1\}; g ((v - u) * x + u) = g ((v - u) * y + u) \rrbracket$

$\implies x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0)$

by (*auto simp: simple_path_def loop_free_def subpath_def*)

{ **fix** $x\ y$

assume $x \in \text{closed_segment } u\ v\ y \in \text{closed_segment } u\ v\ g\ x = g\ y$

then have $x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$

using *sim* [*of* $(x-u)/(v-u)$ $(y-u)/(v-u)$] *p*

by (*auto split: if_split_asm simp add: closed_segment_real_eq image_affinity_atLeastAtMost*)

(*simp_all add: field_split_simps*)

} **moreover**

have $\text{path}(\text{subpath } u\ v\ g) \wedge u \neq v$

using *sim* [*of* 1/3 2/3] *p*

by (*auto simp: subpath_def*)

ultimately show *?rhs*

by *metis*

next

assume *?rhs*

then

```

have d1:  $\bigwedge x y. \llbracket g x = g y; u \leq x; x \leq v; u \leq y; y \leq v \rrbracket \implies x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ 
and d2:  $\bigwedge x y. \llbracket g x = g y; v \leq x; x \leq u; v \leq y; y \leq u \rrbracket \implies x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$ 
and ne:  $u < v \vee v < u$ 
and psp: path (subpath u v g)
by (auto simp: closed_segment_real_eq image_affinity_atLeastAtMost)
have [simp]:  $\bigwedge x. u + x * v = v + x * u \longleftrightarrow u=v \vee x=1$ 
by algebra
show ?lhs using psp ne
unfolding simple_path_def loop_free_def subpath_def
by (fastforce simp add: algebra_simps affine_ineq mult_left_mono crossprod_uct_eq dest: d1 d2)
qed

```

lemma *arc_subpath_eq*:

```

 $\text{arc}(\text{subpath } u \ v \ g) \longleftrightarrow \text{path}(\text{subpath } u \ v \ g) \wedge u \neq v \wedge \text{inj\_on } g \ (\text{closed\_segment } u \ v)$ 
by (smt (verit, best) arc_simple_path closed_segment_commute ends_in_segment(2) inj_on_def pathfinish_subpath pathstart_subpath simple_path_subpath_eq)

```

lemma *simple_path_subpath*:

```

assumes simple_path g u ∈ {0..1} v ∈ {0..1} u ≠ v
shows simple_path(subpath u v g)
using assms
unfolding simple_path_subpath_eq
by (force simp: simple_path_def loop_free_def closed_segment_real_eq image_affinity_atLeastAtMost)

```

lemma *arc_simple_path_subpath*:

```

 $\llbracket \text{simple\_path } g; u \in \{0..1\}; v \in \{0..1\}; g \ u \neq g \ v \rrbracket \implies \text{arc}(\text{subpath } u \ v \ g)$ 
by (force intro: simple_path_subpath simple_path_imp_arc)

```

lemma *arc_subpath_arc*:

```

 $\llbracket \text{arc } g; u \in \{0..1\}; v \in \{0..1\}; u \neq v \rrbracket \implies \text{arc}(\text{subpath } u \ v \ g)$ 
by (meson arc_def arc_imp_simple_path arc_simple_path_subpath inj_onD)

```

lemma *arc_simple_path_subpath_interior*:

```

 $\llbracket \text{simple\_path } g; u \in \{0..1\}; v \in \{0..1\}; u \neq v; |u-v| < 1 \rrbracket \implies \text{arc}(\text{subpath } u \ v \ g)$ 
by (force simp: simple_path_def loop_free_def intro: arc_simple_path_subpath)

```

lemma *path_image_subpath_subset*:

```

 $\llbracket u \in \{0..1\}; v \in \{0..1\} \rrbracket \implies \text{path\_image}(\text{subpath } u \ v \ g) \subseteq \text{path\_image } g$ 
by (metis atLeastAtMost_iff atLeastatMost_subset_iff path_image_def path_image_subpath_subset_image_iff)

```

lemma *join_subpaths_middle*: *subpath (0) ((1 / 2)) p +++ subpath ((1 / 2)) 1*

$p = p$
by (*rule ext*) (*simp add: joinpaths_def subpath_def field_split_simps*)

6.1.10 There is a subpath to the frontier

lemma *subpath_to_frontier_explicit*:

fixes $S :: 'a::metric_space\ set$
assumes $g: path\ g$ **and** $pathfinish\ g \notin S$
obtains u **where** $0 \leq u \leq 1$
 $\bigwedge x. 0 \leq x \wedge x < u \implies g\ x \in interior\ S$
 $(g\ u \notin interior\ S) (u = 0 \vee g\ u \in closure\ S)$

proof –

have $gcon: continuous_on\ \{0..1\}\ g$
using g **by** (*simp add: path_def*)
moreover **have** $bounded\ (\{u. g\ u \in closure\ (-\ S)\} \cap \{0..1\})$
using *compact_eq_bounded_closed* **by** *fastforce*
ultimately **have** $com: compact\ (\{0..1\} \cap \{u. g\ u \in closure\ (-\ S)\})$
using *closed_vimage_Int*
by (*metis (full_types) Int_commute closed_atLeastAtMost closed_closure compact_eq_bounded_closed vimage_def*)
have $1 \in \{u. g\ u \in closure\ (-\ S)\}$
using *assms* **by** (*simp add: pathfinish_def closure_def*)
then **have** $dis: \{0..1\} \cap \{u. g\ u \in closure\ (-\ S)\} \neq \{\}$
using *atLeastAtMost_iff zero_le_one* **by** *blast*
then **obtain** u **where** $0 \leq u \leq 1$ **and** $gu: g\ u \in closure\ (-\ S)$
and $umin: \bigwedge t. [0 \leq t; t \leq 1; g\ t \in closure\ (-\ S)] \implies u \leq t$
using *compact_attains_inf [OF com dis]* **by** *fastforce*
then **have** $umin': \bigwedge t. [0 \leq t; t \leq 1; t < u] \implies g\ t \in S$
using *closure_def* **by** *fastforce*
have $\S: g\ u \in closure\ S$ **if** $u \neq 0$
proof –
have $u > 0$ **using** *that* $\langle 0 \leq u \rangle$ **by** *auto*
{ **fix** $e::real$ **assume** $e > 0$
obtain d **where** $d > 0$ **and** $d: \bigwedge x'. [x' \in \{0..1\}; dist\ x'\ u \leq d] \implies dist\ (g\ x')\ (g\ u) < e$
using *continuous_onE [OF gcon _ <e > 0] <0 ≤ _> <_ ≤ 1> atLeastAtMost_iff* **by** *auto*
have $*$: $dist\ (max\ 0\ (u - d / 2))\ u \leq d$
using $\langle 0 \leq u \rangle \langle u \leq 1 \rangle \langle d > 0 \rangle$ **by** (*simp add: dist_real_def*)
have $\exists y \in S. dist\ y\ (g\ u) < e$
using $\langle 0 < u \rangle \langle u \leq 1 \rangle \langle d > 0 \rangle$
by (*force intro: d [OF _ *] umin'*)
}
then **show** *?thesis*
by (*simp add: frontier_def closure_approachable*)
qed
show *?thesis*
proof
show $\bigwedge x. 0 \leq x \wedge x < u \implies g\ x \in interior\ S$

```

    using ‹ $u \leq 1$ › interior_closure uin by fastforce
  show  $g\ u \notin \text{interior } S$ 
    by (simp add: gu interior_closure)
  qed (use ‹ $0 \leq u$ › ‹ $u \leq 1$ › § in auto)
qed

lemma subpath_to_frontier_strong:
  assumes  $g: \text{path } g$  and  $\text{pathfinish } g \notin S$ 
  obtains  $u$  where  $0 \leq u$   $u \leq 1$   $g\ u \notin \text{interior } S$ 
     $u = 0 \vee (\forall x. 0 \leq x \wedge x < 1 \longrightarrow \text{subpath } 0\ u\ g\ x \in \text{interior } S)$ 
 $\wedge g\ u \in \text{closure } S$ 
proof -
  obtain  $u$  where  $0 \leq u$   $u \leq 1$ 
    and  $gxin: \bigwedge x. 0 \leq x \wedge x < u \implies g\ x \in \text{interior } S$ 
    and  $gunot: (g\ u \notin \text{interior } S)$  and  $u0: (u = 0 \vee g\ u \in \text{closure } S)$ 
  using subpath_to_frontier_explicit [OF assms] by blast
  show ?thesis
  proof
    show  $g\ u \notin \text{interior } S$ 
      using gunot by blast
    qed (use ‹ $0 \leq u$ › ‹ $u \leq 1$ ›  $u0$  in ‹(force simp: subpath_def  $gxin$ )›)
  qed

lemma subpath_to_frontier:
  assumes  $g: \text{path } g$  and  $g0: \text{pathstart } g \in \text{closure } S$  and  $g1: \text{pathfinish } g \notin S$ 
  obtains  $u$  where  $0 \leq u$   $u \leq 1$   $g\ u \in \text{frontier } S$   $\text{path\_image}(\text{subpath } 0\ u\ g) - \{g\ u\} \subseteq \text{interior } S$ 
proof -
  obtain  $u$  where  $0 \leq u$   $u \leq 1$ 
    and  $notin: g\ u \notin \text{interior } S$ 
    and  $disj: u = 0 \vee$ 
       $(\forall x. 0 \leq x \wedge x < 1 \longrightarrow \text{subpath } 0\ u\ g\ x \in \text{interior } S) \wedge g\ u \in$ 
 $\text{closure } S$ 
    (is  $\_ \vee ?P$ )
  using subpath_to_frontier_strong [OF  $g\ g1$ ] by blast
  show ?thesis
  proof
    show  $g\ u \in \text{frontier } S$ 
      by (metis DiffI disj frontier_def  $g0$  notin pathstart_def)
    show  $\text{path\_image}(\text{subpath } 0\ u\ g) - \{g\ u\} \subseteq \text{interior } S$ 
      using disj
    proof
      assume  $u = 0$ 
      then show ?thesis
        by (simp add: path_image_subpath)
    next
      assume  $P: ?P$ 
      show ?thesis
        proof (clarsimp simp add: path_image_subpath_gen)

```

```

fix y
assume y: y ∈ closed_segment 0 u g y ∉ interior S
with ⟨0 ≤ u⟩ have 0 ≤ y y ≤ u
  by (auto simp: closed_segment_eq_real_ivl split: if_split_asm)
then have y=u ∨ subpath 0 u g (y/u) ∈ interior S
  using P less_eq_real_def by force
then show g y = g u
  using y by (auto simp: subpath_def split: if_split_asm)
qed
qed
qed (use ⟨0 ≤ u⟩ ⟨u ≤ 1⟩ in auto)
qed

```

lemma exists_path_subpath_to_frontier:

```

fixes S :: 'a::real_normed_vector set
assumes path g pathstart g ∈ closure S pathfinish g ∉ S
obtains h where path h pathstart h = pathstart g path_image h ⊆ path_image
g
      path_image h - {pathfinish h} ⊆ interior S
      pathfinish h ∈ frontier S

```

proof –

```

obtain u where u: 0 ≤ u u ≤ 1 g u ∈ frontier S (path_image(subpath 0 u g) –
{g u}) ⊆ interior S
  using subpath_to_frontier [OF assms] by blast
show ?thesis
proof
  show path_image (subpath 0 u g) ⊆ path_image g
    by (simp add: path_image_subpath_subset u)
  show pathstart (subpath 0 u g) = pathstart g
    by (metis pathstart_def pathstart_subpath)
  qed (use assms u in ⟨auto simp: path_image_subpath⟩)
qed

```

lemma exists_path_subpath_to_frontier_closed:

```

fixes S :: 'a::real_normed_vector set
assumes S: closed S and g: path g and g0: pathstart g ∈ S and g1: pathfinish
g ∉ S
obtains h where path h pathstart h = pathstart g path_image h ⊆ path_image
g ∩ S
      pathfinish h ∈ frontier S
by (smt (verit, del_insts) Diff_iff Int_iff S closure_closed exists_path_subpath_to_frontier
      frontier_def g g0 g1 interior_subset singletonD subset_eq)

```

6.1.11 Shift Path to Start at Some Given Point

definition shiftpath :: real ⇒ (real ⇒ 'a::topological_space) ⇒ real ⇒ 'a
where shiftpath a f = (λx. if (a + x) ≤ 1 then f (a + x) else f (a + x - 1))

lemma *shiftpath_alt_def*: $\text{shiftpath } a \ f = (\lambda x. \text{if } x \leq 1-a \text{ then } f(a+x) \text{ else } f(a+x-1))$

by (*auto simp: shiftpath_def*)

lemma *pathstart_shiftpath*: $a \leq 1 \implies \text{pathstart}(\text{shiftpath } a \ g) = g \ a$

unfolding *pathstart_def shiftpath_def* **by** *auto*

lemma *pathfinish_shiftpath*:

assumes $0 \leq a$

and $\text{pathfinish } g = \text{pathstart } g$

shows $\text{pathfinish}(\text{shiftpath } a \ g) = g \ a$

using *assms*

unfolding *pathstart_def pathfinish_def shiftpath_def*

by *auto*

lemma *endpoints_shiftpath*:

assumes $\text{pathfinish } g = \text{pathstart } g$

and $a \in \{0 \dots 1\}$

shows $\text{pathfinish}(\text{shiftpath } a \ g) = g \ a$

and $\text{pathstart}(\text{shiftpath } a \ g) = g \ a$

using *assms*

by (*simp_all add: pathstart_shiftpath pathfinish_shiftpath*)

lemma *closed_shiftpath*:

assumes $\text{pathfinish } g = \text{pathstart } g$

and $a \in \{0..1\}$

shows $\text{pathfinish}(\text{shiftpath } a \ g) = \text{pathstart}(\text{shiftpath } a \ g)$

using *endpoints_shiftpath[OF assms]*

by *auto*

lemma *path_shiftpath*:

assumes $\text{path } g$

and $\text{pathfinish } g = \text{pathstart } g$

and $a \in \{0..1\}$

shows $\text{path}(\text{shiftpath } a \ g)$

proof –

have *: $\{0 \dots 1\} = \{0 \dots 1-a\} \cup \{1-a \dots 1\}$

using *assms(3)* **by** *auto*

have **: $\bigwedge x. x + a = 1 \implies g(x+a-1) = g(x+a)$

by (*smt (verit, best) assms(2) pathfinish_def pathstart_def*)

show *?thesis*

unfolding *path_def shiftpath_def* *

proof (*rule continuous_on_closed_Un*)

have *contg*: $\text{continuous_on } \{0..1\} \ g$

using $\langle \text{path } g \rangle$ *path_def* **by** *blast*

show $\text{continuous_on } \{0..1-a\} \ (\lambda x. \text{if } a+x \leq 1 \text{ then } g(a+x) \text{ else } g(a+x-1))$

proof (*rule continuous_on_eq*)

show $\text{continuous_on } \{0..1-a\} \ (g \circ (+) \ a)$

```

    by (intro continuous_intros continuous_on_subset [OF contg]) (use ⟨a ∈
{0..1}⟩ in auto)
  qed auto
  show continuous_on {1-a..1} (λx. if a + x ≤ 1 then g (a + x) else g (a + x
- 1))
  proof (rule continuous_on_eq)
    show continuous_on {1-a..1} (g ∘ (+) (a - 1))
    by (intro continuous_intros continuous_on_subset [OF contg]) (use ⟨a ∈
{0..1}⟩ in auto)
  qed (auto simp: ** add commute add_diff_eq)
  qed auto
qed

```

```

lemma shiftpath_shiftpath:
  assumes pathfinish g = pathstart g
    and a ∈ {0..1}
    and x ∈ {0..1}
  shows shiftpath (1 - a) (shiftpath a g) x = g x
  using assms
  unfolding pathfinish_def pathstart_def shiftpath_def
  by auto

```

```

lemma path_image_shiftpath:
  assumes a: a ∈ {0..1}
    and pathfinish g = pathstart g
  shows path_image (shiftpath a g) = path_image g
  proof -
    { fix x
      assume g: g 1 = g 0 x ∈ {0..1::real} and gne: ∧y. y∈{0..1} ∩ {x. ¬ a + x
≤ 1} ⇒ g x ≠ g (a + y - 1)
      then have ∃y∈{0..1} ∩ {x. a + x ≤ 1}. g x = g (a + y)
      proof (cases a ≤ x)
        case False
        then show ?thesis
          apply (rule_tac x=1 + x - a in bexI)
          using g gne[of 1 + x - a] a by (force simp: field_simps)+
        next
        case True
        then show ?thesis
          using g a by (rule_tac x=x - a in bexI) (auto simp: field_simps)
      qed
    }
  then show ?thesis
  using assms
  unfolding shiftpath_def path_image_def pathfinish_def pathstart_def
  by (auto simp: image_iff)
qed

```

```

lemma loop_free_shiftpath:

```



```

assumes loop_free g pathfinish g = pathstart g and a:  $0 \leq a \leq 1$ 
shows loop_free (shiftpath a g)
unfolding loop_free_def
proof (intro conjI impI ballI)
show  $x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$ 
if  $x \in \{0..1\}$   $y \in \{0..1\}$  shiftpath a g  $x = \text{shiftpath a g y}$  for x y
using that a assms unfolding shiftpath_def loop_free_def
by (smt (verit, ccfv_threshold) atLeastAtMost_iff)
qed

```

```

lemma simple_path_shiftpath:
assumes simple_path g pathfinish g = pathstart g and a:  $0 \leq a \leq 1$ 
shows simple_path (shiftpath a g)
using assms loop_free_shiftpath path_shiftpath simple_path_def by fastforce

```

6.1.12 Straight-Line Paths

```

definition linepath :: 'a::real_normed_vector  $\Rightarrow$  'a  $\Rightarrow$  real  $\Rightarrow$  'a
where linepath a b =  $(\lambda x. (1 - x) *_R a + x *_R b)$ 

```

```

lemma pathstart_linepath[simp]: pathstart (linepath a b) = a
unfolding pathstart_def linepath_def
by auto

```

```

lemma pathfinish_linepath[simp]: pathfinish (linepath a b) = b
unfolding pathfinish_def linepath_def
by auto

```

```

lemma linepath_inner: linepath a b  $x \cdot v = \text{linepath } (a \cdot v) (b \cdot v) x$ 
by (simp add: linepath_def algebra_simps)

```

```

lemma Re_linepath': Re (linepath a b x) = linepath (Re a) (Re b) x
by (simp add: linepath_def)

```

```

lemma Im_linepath': Im (linepath a b x) = linepath (Im a) (Im b) x
by (simp add: linepath_def)

```

```

lemma linepath_0': linepath a b 0 = a
by (simp add: linepath_def)

```

```

lemma linepath_1': linepath a b 1 = b
by (simp add: linepath_def)

```

```

lemma continuous_linepath_at[intro]: continuous (at x) (linepath a b)
unfolding linepath_def
by (intro continuous_intros)

```

```

lemma continuous_on_linepath [intro,continuous_intros]: continuous_on s (linepath
a b)

```

using *continuous_linepath_at*
by (*auto intro!*: *continuous_at_imp_continuous_on*)

lemma *path_linepath[iff]*: *path (linepath a b)*
unfolding *path_def*
by (*rule continuous_on_linepath*)

lemma *path_image_linepath[simp]*: *path_image (linepath a b) = closed_segment a b*
unfolding *path_image_def segment_linepath_def*
by *auto*

lemma *reversepath_linepath[simp]*: *reversepath (linepath a b) = linepath b a*
unfolding *reversepath_def linepath_def*
by *auto*

lemma *linepath_0 [simp]*: *linepath 0 b x = x *_R b*
by (*simp add: linepath_def*)

lemma *linepath_cnj*: *cnj (linepath a b x) = linepath (cnj a) (cnj b) x*
by (*simp add: linepath_def*)

lemma *arc_linepath*:
assumes $a \neq b$ **shows** [*simp*]: *arc (linepath a b)*
proof –
{
 fix $x y :: \text{real}$
 assume $x *_{\mathbb{R}} b + y *_{\mathbb{R}} a = x *_{\mathbb{R}} a + y *_{\mathbb{R}} b$
 then have $(x - y) *_{\mathbb{R}} a = (x - y) *_{\mathbb{R}} b$
 by (*simp add: algebra_simps*)
 with *assms* **have** $x = y$
 by *simp*
}
then show *?thesis*
unfolding *arc_def inj_on_def*
by (*fastforce simp: algebra_simps linepath_def*)
qed

lemma *simple_path_linepath[intro]*: $a \neq b \implies \text{simple_path (linepath a b)}$
by (*simp add: arc_imp_simple_path*)

lemma *linepath_trivial [simp]*: *linepath a a x = a*
by (*simp add: linepath_def real_vector.scale_left_diff_distrib*)

lemma *linepath_refl*: *linepath a a = ($\lambda x. a$)*
by *auto*

lemma *subpath_refl*: *subpath a a g = linepath (g a) (g a)*
by (*simp add: subpath_def linepath_def algebra_simps*)

lemma *linepath_of_real*: $(\text{linepath } (\text{of_real } a) (\text{of_real } b) x) = \text{of_real } ((1 - x)*a + x*b)$

by (*simp add: scaleR_conv_of_real linepath_def*)

lemma *of_real_linepath*: $\text{of_real } (\text{linepath } a b x) = \text{linepath } (\text{of_real } a) (\text{of_real } b) x$

by (*metis linepath_of_real mult.right_neutral of_real_def real_scaleR_def*)

lemma *inj_on_linepath*:

assumes $a \neq b$ **shows** *inj_on* (*linepath* $a b$) $\{0..1\}$

using *arc_imp_inj_on arc_linepath assms* **by** *blast*

lemma *linepath_le_1*:

fixes $a::'a::\text{linordered_idom}$ **shows** $\llbracket a \leq 1; b \leq 1; 0 \leq u; u \leq 1 \rrbracket \implies (1 - u) * a + u * b \leq 1$

using *mult_left_le [of a 1-u] mult_left_le [of b u]* **by** *auto*

lemma *linepath_in_path*:

shows $x \in \{0..1\} \implies \text{linepath } a b x \in \text{closed_segment } a b$

by (*auto simp: segment linepath_def*)

lemma *linepath_image_01*: $\text{linepath } a b \text{ ' } \{0..1\} = \text{closed_segment } a b$

by (*auto simp: segment linepath_def*)

lemma *linepath_in_convex_hull*:

fixes $x::\text{real}$

assumes $a \in \text{convex_hull } S$

and $b \in \text{convex_hull } S$

and $0 \leq x \leq 1$

shows $\text{linepath } a b x \in \text{convex_hull } S$

by (*meson assms atLeastAtMost_iff convex_contains_segment convex_convex_hull linepath_in_path subset_eq*)

lemma *Re_linepath*: $\text{Re}(\text{linepath } (\text{of_real } a) (\text{of_real } b) x) = (1 - x)*a + x*b$

by (*simp add: linepath_def*)

lemma *Im_linepath*: $\text{Im}(\text{linepath } (\text{of_real } a) (\text{of_real } b) x) = 0$

by (*simp add: linepath_def*)

lemma *bounded_linear_linepath*:

assumes *bounded_linear* f

shows $f (\text{linepath } a b x) = \text{linepath } (f a) (f b) x$

proof –

interpret f : *bounded_linear* f **by** *fact*

show *?thesis* **by** (*simp add: linepath_def f.add f.scale*)

qed

lemma *bounded_linear_linepath'*:

assumes *bounded_linear* *f*
shows $f \circ \text{linepath } a \ b = \text{linepath } (f \ a) \ (f \ b)$
using *bounded_linear_linepath*[*OF assms*] **by** (*simp add: fun_eq_iff*)

lemma *linepath_cnj'*: $\text{cnj} \circ \text{linepath } a \ b = \text{linepath } (\text{cnj } a) \ (\text{cnj } b)$
by (*simp add: linepath_def fun_eq_iff*)

lemma *differentiable_linepath* [*intro*]: *linepath* *a b* *differentiable at x within A*
by (*auto simp: linepath_def*)

lemma *has_vector_derivative_linepath_within*:
(*linepath* *a b* *has_vector_derivative* (*b - a*)) (*at x within S*)
by (*force intro: derivative_eq_intros simp add: linepath_def has_vector_derivative_def algebra_simps*)

6.1.13 Segments via convex hulls

lemma *segments_subset_convex_hull*:
closed_segment *a b* \subseteq (*convex hull* {*a,b,c*})
closed_segment *a c* \subseteq (*convex hull* {*a,b,c*})
closed_segment *b c* \subseteq (*convex hull* {*a,b,c*})
closed_segment *b a* \subseteq (*convex hull* {*a,b,c*})
closed_segment *c a* \subseteq (*convex hull* {*a,b,c*})
closed_segment *c b* \subseteq (*convex hull* {*a,b,c*})
by (*auto simp: segment_convex_hull linepath_of_real elim!: rev_subsetD [OF _ hull_mono]*)

lemma *midpoints_in_convex_hull*:
assumes $x \in \text{convex hull } s$ $y \in \text{convex hull } s$
shows $\text{midpoint } x \ y \in \text{convex hull } s$
using *assms closed_segment_subset_convex_hull csegment_midpoint_subset* **by** *blast*

lemma *not_in_interior_convex_hull_3*:
fixes *a :: complex*
shows $a \notin \text{interior}(\text{convex hull } \{a,b,c\})$
 $b \notin \text{interior}(\text{convex hull } \{a,b,c\})$
 $c \notin \text{interior}(\text{convex hull } \{a,b,c\})$
by (*auto simp: card_insert_le_m1 not_in_interior_convex_hull*)

lemma *midpoint_in_closed_segment* [*simp*]: $\text{midpoint } a \ b \in \text{closed_segment } a \ b$
using *midpoints_in_convex_hull segment_convex_hull* **by** *blast*

lemma *midpoint_in_open_segment* [*simp*]: $\text{midpoint } a \ b \in \text{open_segment } a \ b \longleftrightarrow a \neq b$
by (*simp add: open_segment_def*)

lemma *continuous_IVT_local_extremum*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow \text{real}$

```

assumes contf: continuous_on (closed_segment a b) f
and ab:  $a \neq b \implies f\ a = f\ b$ 
obtains z where  $z \in \text{open\_segment } a\ b$ 
       $(\forall w \in \text{closed\_segment } a\ b. (f\ w) \leq (f\ z)) \vee$ 
       $(\forall w \in \text{closed\_segment } a\ b. (f\ z) \leq (f\ w))$ 
proof –
  obtain c where  $c \in \text{closed\_segment } a\ b$  and  $c: \bigwedge y. y \in \text{closed\_segment } a\ b$ 
 $\implies f\ y \leq f\ c$ 
  using continuous_attains_sup [of closed_segment a b f] contf by auto
  moreover
  obtain d where  $d \in \text{closed\_segment } a\ b$  and  $d: \bigwedge y. y \in \text{closed\_segment } a\ b$ 
 $\implies f\ d \leq f\ y$ 
  using continuous_attains_inf [of closed_segment a b f] contf by auto
  ultimately show ?thesis
  by (smt (verit) UnE ab closed_segment_eq_open empty_iff_insert_iff midpoint_in_open_segment that)
qed

```

An injective map into \mathbb{R} is also an open map w.r.T. the universe, and conversely.

proposition *injective_eq_1d_open_map_UNIV*:

```

fixes f :: real  $\Rightarrow$  real
assumes contf: continuous_on S f and S: is_interval S
shows inj_on f S  $\longleftrightarrow (\forall T. \text{open } T \wedge T \subseteq S \longrightarrow \text{open}(f\ ' T))$ 
      (is ?lhs = ?rhs)

```

proof *safe*

```

fix T
assume injf: ?lhs and open T and  $T \subseteq S$ 
have  $\exists U. \text{open } U \wedge f\ x \in U \wedge U \subseteq f\ ' T$  if  $x \in T$  for x
proof –
  obtain  $\delta$  where  $\delta > 0$  and  $\delta: \text{cball } x\ \delta \subseteq T$ 
  using  $\langle \text{open } T \rangle \langle x \in T \rangle$  open_contains_cball_eq by blast
  show ?thesis
proof (intro exI conjI)
  have closed_segment  $(x - \delta)$   $(x + \delta) = \{x - \delta..x + \delta\}$ 
  using  $\langle 0 < \delta \rangle$  by (auto simp: closed_segment_eq_real_ivl)
  also have  $\dots \subseteq S$ 
  using  $\delta \langle T \subseteq S \rangle$  by (auto simp: dist_norm subset_eq)
  finally have  $f\ ' (\text{open\_segment } (x - \delta)$   $(x + \delta)) = \text{open\_segment } (f\ (x - \delta))$   $(f\ (x + \delta))$ 
  using continuous_injective_image_open_segment_1
  by (metis continuous_on_subset [OF contf] inj_on_subset [OF injf])
  then show open  $(f\ ' \{x - \delta <.. <x + \delta\})$ 
  using  $\langle 0 < \delta \rangle$  by (simp add: open_segment_eq_real_ivl)
  show  $f\ x \in f\ ' \{x - \delta <.. <x + \delta\}$ 
  by (auto simp:  $\langle \delta > 0 \rangle$ )
  show  $f\ ' \{x - \delta <.. <x + \delta\} \subseteq f\ ' T$ 
  using  $\delta$  by (auto simp: dist_norm subset_iff)
qed

```

```

qed
with open_subopen show open (f ' T)
  by blast
next
assume R: ?rhs
have False if xy: x ∈ S y ∈ S and f x = f y x ≠ y for x y
proof -
  have open (f ' open_segment x y)
    using R
  by (metis S convex_contains_open_segment is_interval_convex open_greaterThanLessThan
open_segment_eq_real_ivl xy)
  moreover
  have continuous_on (closed_segment x y) f
  by (meson S closed_segment_subset contf continuous_on_subset is_interval_convex
that)
  then obtain ξ where ξ ∈ open_segment x y
    and ξ: (∀ w ∈ closed_segment x y. (f w) ≤ (f ξ)) ∨
      (∀ w ∈ closed_segment x y. (f ξ) ≤ (f w))
    using continuous_IVT_local_extremum [of x y f] ⟨f x = f y⟩ ⟨x ≠ y⟩ by blast
  ultimately obtain e where e>0 and e: ∧u. dist u (f ξ) < e ⇒ u ∈ f '
open_segment x y
    using open_dist by (metis image_eqI)
  have fin: f ξ + (e/2) ∈ f ' open_segment x y f ξ - (e/2) ∈ f ' open_segment
x y
    using e [of f ξ + (e/2)] e [of f ξ - (e/2)] ⟨e > 0⟩ by (auto simp: dist_norm)
  show ?thesis
    using ξ ⟨0 < e⟩ fin open_closed_segment by fastforce
qed
then show ?lhs
  by (force simp: inj_on_def)
qed

```

6.1.14 Bounding a point away from a path

```

lemma not_on_path_ball:
  fixes g :: real ⇒ 'a::heine_borel
  assumes path g
  and z: z ∉ path_image g
  shows ∃ e > 0. ball z e ∩ path_image g = {}
proof -
  have closed (path_image g)
  by (simp add: ⟨path g⟩ closed_path_image)
  then obtain a where a ∈ path_image g ∀ y ∈ path_image g. dist z a ≤ dist z y
  by (auto intro: distance_attains_inf[OF path_image_nonempty, of g z])
  then show ?thesis
  by (rule_tac x=dist z a in exI) (use dist_commute z in auto)
qed

lemma not_on_path_cball:

```

```

fixes  $g :: \text{real} \Rightarrow 'a::\text{heine\_borel}$ 
assumes  $\text{path } g$ 
and  $z \notin \text{path\_image } g$ 
shows  $\exists e > 0. \text{cball } z \ e \cap (\text{path\_image } g) = \{\}$ 
by (smt (verit, ccfv\_threshold) open\_ball assms centre\_in\_ball inf.orderE inf\_assoc
inf\_bot\_right not\_on\_path\_ball open\_contains\_cball\_eq)

```

6.1.15 Path component

Original formalization by Tom Hales

```

definition  $\text{path\_component } S \ x \ y \equiv$ 
 $(\exists g. \text{path } g \wedge \text{path\_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)$ 

```

abbreviation

```

 $\text{path\_component\_set } S \ x \equiv \text{Collect } (\text{path\_component } S \ x)$ 

```

```

lemmas  $\text{path\_defs} = \text{path\_def } \text{pathstart\_def } \text{pathfinish\_def } \text{path\_image\_def } \text{path\_component\_def}$ 

```

lemma $\text{path_component_mem}$:

```

assumes  $\text{path\_component } S \ x \ y$ 
shows  $x \in S$  and  $y \in S$ 
using assms
unfolding  $\text{path\_defs}$ 
by auto

```

lemma $\text{path_component_refl}$:

```

assumes  $x \in S$ 
shows  $\text{path\_component } S \ x \ x$ 
using assms
unfolding  $\text{path\_defs}$ 
by (metis (full\_types) assms continuous\_on\_const image\_subset\_iff path\_image\_def)

```

lemma $\text{path_component_refl_eq}$: $\text{path_component } S \ x \ x \longleftrightarrow x \in S$

```

by (auto intro!:  $\text{path\_component\_mem}$   $\text{path\_component\_refl}$ )

```

lemma $\text{path_component_sym}$: $\text{path_component } S \ x \ y \Longrightarrow \text{path_component } S \ y \ x$

```

unfolding  $\text{path\_component\_def}$ 
by (metis (no\_types) path\_image\_reversepath path\_reversepath pathfinish\_reversepath
pathstart\_reversepath)

```

lemma $\text{path_component_trans}$:

```

assumes  $\text{path\_component } S \ x \ y$  and  $\text{path\_component } S \ y \ z$ 
shows  $\text{path\_component } S \ x \ z$ 
using assms
unfolding  $\text{path\_component\_def}$ 
by (metis path\_join pathfinish\_join pathstart\_join subset\_path\_image\_join)

```

lemma $\text{path_component_of_subset}$: $S \subseteq T \Longrightarrow \text{path_component } S \ x \ y \Longrightarrow \text{path_component } T \ x \ y$

unfolding *path_component_def* **by** *auto*

lemma *path_component_linepath*:
 fixes $S :: 'a::\text{real_normed_vector_set}$
 shows $\text{closed_segment } a \ b \subseteq S \implies \text{path_component } S \ a \ b$
unfolding *path_component_def* **by** *fastforce*

Path components as sets

lemma *path_component_set*:
 $\text{path_component_set } S \ x =$
 $\{y. (\exists g. \text{path } g \wedge \text{path_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)\}$
by (*auto simp: path_component_def*)

lemma *path_component_subset*: $\text{path_component_set } S \ x \subseteq S$
by (*auto simp: path_component_mem(2)*)

lemma *path_component_eq_empty*: $\text{path_component_set } S \ x = \{\} \longleftrightarrow x \notin S$
using *path_component_mem path_component_refl_eq*
by *fastforce*

lemma *path_component_mono*:
 $S \subseteq T \implies (\text{path_component_set } S \ x) \subseteq (\text{path_component_set } T \ x)$
by (*simp add: Collect_mono path_component_of_subset*)

lemma *path_component_eq*:
 $y \in \text{path_component_set } S \ x \implies \text{path_component_set } S \ y = \text{path_component_set } S \ x$
by (*metis (no_types, lifting) Collect_cong mem_Collect_eq path_component_sym path_component_trans*)

6.1.16 Path connectedness of a space

definition *path_connected* $S \longleftrightarrow$
 $(\forall x \in S. \forall y \in S. \exists g. \text{path } g \wedge \text{path_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y)$

lemma *path_connectedin_iff_path_connected_real* [*simp*]:
 $\text{path_connectedin euclideanreal } S \longleftrightarrow \text{path_connected } S$
by (*simp add: path_connectedin path_connected_def path_defs image_subset_iff_funcset*)

lemma *path_connected_component*: $\text{path_connected } S \longleftrightarrow (\forall x \in S. \forall y \in S. \text{path_component } S \ x \ y)$
unfolding *path_connected_def path_component_def* **by** *auto*

lemma *path_connected_component_set*: $\text{path_connected } S \longleftrightarrow (\forall x \in S. \text{path_component_set } S \ x = S)$
unfolding *path_connected_component path_component_subset*
using *path_component_mem* **by** *blast*

lemma *path_component_maximal*:

$\llbracket x \in T; \text{path_connected } T; T \subseteq S \rrbracket \implies T \subseteq (\text{path_component_set } S \ x)$
by (*metis path_component_mono path_connected_component_set*)

lemma *convex_imp_path_connected*:

fixes $S :: 'a::\text{real_normed_vector_set}$
assumes *convex S*
shows *path_connected S*
unfolding *path_connected_def*
using *assms convex_contains_segment* **by** *fastforce*

lemma *path_connected_UNIV* [*iff*]: *path_connected (UNIV :: 'a::real_normed_vector set)*

by (*simp add: convex_imp_path_connected*)

lemma *path_component_UNIV*: *path_component_set UNIV x = (UNIV :: 'a::real_normed_vector set)*

using *path_connected_component_set* **by** *auto*

lemma *path_connected_imp_connected*:

assumes *path_connected S*
shows *connected S*

proof (*rule connectedI*)

fix $e1 \ e2$

assume *as*: $\text{open } e1 \ \text{open } e2 \ S \subseteq e1 \cup e2 \ e1 \cap e2 \cap S = \{\} \ e1 \cap S \neq \{\} \ e2 \cap S \neq \{\}$

then obtain $x1 \ x2$ **where** $\text{obt}: x1 \in e1 \cap S \ x2 \in e2 \cap S$

by *auto*

then obtain g **where** $g: \text{path } g \ \text{path_image } g \subseteq S$ **and** $\text{pg}: \text{pathstart } g = x1$
 $\text{pathfinish } g = x2$

using *assms[unfolded path_connected_def, rule_format, of x1 x2]* **by** *auto*

have $*$: *connected* $\{0..1\}::\text{real}$

by (*auto intro!: convex_connected*)

have $\{0..1\} \subseteq \{x \in \{0..1\}. g \ x \in e1\} \cup \{x \in \{0..1\}. g \ x \in e2\}$

using *as(3) g(2)[unfolded path_defs]* **by** *blast*

moreover have $\{x \in \{0..1\}. g \ x \in e1\} \cap \{x \in \{0..1\}. g \ x \in e2\} = \{\}$

using *as(4) g(2)[unfolded path_defs]*

unfolding *subset_eq*

by *auto*

moreover have $\{x \in \{0..1\}. g \ x \in e1\} \neq \{\} \wedge \{x \in \{0..1\}. g \ x \in e2\} \neq \{\}$

by (*smt (verit, ccfv_threshold) IntE atLeastAtMost_iff empty_iff pg mem_Collect_eq obt pathfinish_def pathstart_def*)

ultimately show *False*

using $*$ [*unfolded connected_local not_ex, rule_format,*

of $\{0..1\} \cap g - 'e1 \ \{0..1\} \cap g - 'e2$]

using *continuous_openin_preimage_gen[OF g(1)[unfolded path_def] as(1)]*

using *continuous_openin_preimage_gen[OF g(1)[unfolded path_def] as(2)]*

by *auto*

qed

```
lemma open_path_component:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (path_component_set S x)
  unfolding open_contains_ball
  by (metis assms centre_in_ball convex_ball convex_imp_path_connected equals0D
    openE
    path_component_eq path_component_eq_empty path_component_maximal)
```

```
lemma open_non_path_component:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (S - path_component_set S x)
  unfolding open_contains_ball
proof
  fix y
  assume y: y ∈ S - path_component_set S x
  then obtain e where e: e > 0 ball y e ⊆ S
    using assms openE by auto
  show ∃ e > 0. ball y e ⊆ S - path_component_set S x
  proof (intro exI conjI subsetI DiffI notI)
    show ∧ x. x ∈ ball y e ⇒ x ∈ S
      using e by blast
    show False if z ∈ ball y e z ∈ path_component_set S x for z
      by (metis (no_types, lifting) Diff_iff centre_in_ball convex_ball convex_imp_path_connected
        path_component_eq path_component_maximal subsetD that y e)
  qed (use e in auto)
qed
```

```
lemma connected_open_path_connected:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  and connected S
  shows path_connected S
  unfolding path_connected_component_set
proof (rule, rule, rule path_component_subset, rule)
  fix x y
  assume x ∈ S and y ∈ S
  show y ∈ path_component_set S x
  proof (rule ccontr)
    assume ¬ ?thesis
    moreover have path_component_set S x ∩ S ≠ {}
      using ⟨x ∈ S⟩ path_component_eq_empty path_component_subset[of S x]
      by auto
    ultimately
    show False
```

```

using ⟨ $y \in S$ ⟩ open_non_path_component[OF ⟨open S⟩] open_path_component[OF
⟨open S⟩]
using ⟨connected S⟩[unfolded connected_def not_ex, rule_format,
of path_component_set S x S - path_component_set S x]
by auto
qed
qed

```

```

lemma path_connected_continuous_image:
assumes contf: continuous_on S f
and path_connected S
shows path_connected (f ` S)
unfolding path_connected_def
proof clarsimp
fix  $x\ y$ 
assume  $x \in S$  and  $y \in S$ 
with ⟨path_connected S⟩
show  $\exists g. \text{path } g \wedge \text{path\_image } g \subseteq f ` S \wedge \text{pathstart } g = f\ x \wedge \text{pathfinish } g = f\ y$ 
unfolding path_defs path_connected_def
using continuous_on_subset[OF contf]
by (smt (verit, ccfv_threshold) continuous_on_compose2 image_eqI image_subset_iff)
qed

```

```

lemma path_connected_translationI:
fixes  $a :: 'a :: \text{topological\_group\_add}$ 
assumes path_connected S shows path_connected (( $\lambda x. a + x$ ) ` S)
by (intro path_connected_continuous_image assms continuous_intros)

```

```

lemma path_connected_translation:
fixes  $a :: 'a :: \text{topological\_group\_add}$ 
shows path_connected (( $\lambda x. a + x$ ) ` S) = path_connected S
proof -
have  $\forall x\ y. (+) (x::'a) ` (+) (0 - x) ` y = y$ 
by (simp add: image_image)
then show ?thesis
by (metis (no_types) path_connected_translationI)
qed

```

```

lemma path_connected_segment [simp]:
fixes  $a :: 'a :: \text{real\_normed\_vector}$ 
shows path_connected (closed_segment a b)
by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_open_segment [simp]:
fixes  $a :: 'a :: \text{real\_normed\_vector}$ 
shows path_connected (open_segment a b)
by (simp add: convex_imp_path_connected)

```

```

lemma homeomorphic_path_connectedness:

```

1000

S homeomorphic $T \implies \text{path_connected } S \iff \text{path_connected } T$
unfolding *homeomorphic_def* *homeomorphism_def* **by** (*metis path_connected_continuous_image*)

lemma *path_connected_empty* [*simp*]: *path_connected* {}
unfolding *path_connected_def* **by** *auto*

lemma *path_connected_singleton* [*simp*]: *path_connected* {*a*}
unfolding *path_connected_def* *pathstart_def* *pathfinish_def* *path_image_def*
using *path_def* **by** *fastforce*

lemma *path_connected_Un*:
assumes *path_connected S*
and *path_connected T*
and $S \cap T \neq \{\}$
shows *path_connected* ($S \cup T$)
unfolding *path_connected_component*
proof (*intro ballI*)
fix $x\ y$
assume $x: x \in S \cup T$ **and** $y: y \in S \cup T$
from *assms* **obtain** z **where** $z: z \in S\ z \in T$
by *auto*
with $x\ y$ **show** *path_component* ($S \cup T$) $x\ y$
by (*smt* (*verit*) *assms*(1,2) *in_mono* *mem_Collect_eq* *path_component_eq*
path_component_maximal
sup.bounded_iff *sup.cobounded2* *sup_ge1*)
qed

lemma *path_connected_UNION*:
assumes $\bigwedge i. i \in A \implies \text{path_connected } (S\ i)$
and $\bigwedge i. i \in A \implies z \in S\ i$
shows *path_connected* ($\bigcup_{i \in A}. S\ i$)
unfolding *path_connected_component*
proof *clarify*
fix $x\ i\ y\ j$
assume $*$: $i \in A\ x \in S\ i\ j \in A\ y \in S\ j$
then **have** *path_component* ($S\ i$) $x\ z$ **and** *path_component* ($S\ j$) $z\ y$
using *assms* **by** (*simp_all* *add: path_connected_component*)
then **have** *path_component* ($\bigcup_{i \in A}. S\ i$) $x\ z$ **and** *path_component* ($\bigcup_{i \in A}. S\ i$)
 $z\ y$
using $*(1,3)$ **by** (*meson SUP_upper* *path_component_of_subset*)
then **show** *path_component* ($\bigcup_{i \in A}. S\ i$) $x\ y$
by (*rule path_component_trans*)
qed

lemma *path_component_path_image_pathstart*:
assumes $p: \text{path } p$ **and** $x: x \in \text{path_image } p$
shows *path_component* (*path_image* p) (*pathstart* p) x
proof –
obtain y **where** $x: x = p\ y$ **and** $y: 0 \leq y\ y \leq 1$

```

  using x by (auto simp: path_image_def)
show ?thesis
  unfolding path_component_def
proof (intro exI conjI)
  have continuous_on ((* y ' {0..1}) p
    by (simp add: continuous_on_path image_mult_atLeastAtMost_if p y)
  then have continuous_on {0..1} (p o ((* y))
    using continuous_on_compose continuous_on_mult_const by blast
  then show path ( $\lambda u. p (y * u)$ )
    by (simp add: path_def)
  show path_image ( $\lambda u. p (y * u)$ )  $\subseteq$  path_image p
    using y mult_le_one by (fastforce simp: path_image_def image_iff)
qed (auto simp: pathstart_def pathfinish_def x)
qed

lemma path_connected_path_image: path p  $\implies$  path_connected(path_image p)
  unfolding path_connected_component
  by (meson path_component_path_image_pathstart path_component_sym path_component_trans)

lemma path_connected_path_component [simp]:
  path_connected (path_component_set S x)
  by (smt (verit) mem_Collect_eq path_component_def path_component_eq path_component_maximal

    path_connected_component path_connected_path_image pathstart_in_path_image)

lemma path_component:
  path_component S x y  $\longleftrightarrow$  ( $\exists t. \text{path\_connected } t \wedge t \subseteq S \wedge x \in t \wedge y \in t$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis path_component_def path_connected_path_image pathfinish_in_path_image
      pathstart_in_path_image)
  next
  assume ?rhs then show ?lhs
    by (meson path_component_of_subset path_connected_component)
qed

lemma path_component_path_component [simp]:
  path_component_set (path_component_set S x) x = path_component_set S x
  by (metis (full_types) mem_Collect_eq path_component_eq_empty path_component_refl
    path_connected_component_set path_connected_path_component)

lemma path_component_subset_connected_component:
  (path_component_set S x)  $\subseteq$  (connected_component_set S x)
proof (cases x  $\in$  S)
  case True show ?thesis
    by (simp add: True connected_component_maximal path_component_refl path_component_subset
      path_connected_imp_connected)
  next

```

```

    case False then show ?thesis
      using path_component_eq_empty by auto
qed

```

6.1.17 Lemmas about path-connectedness

```

lemma path_connected_linear_image:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes path_connected  $S$  bounded_linear  $f$ 
  shows path_connected  $(f \ ` S)$ 
  by (auto simp: linear_continuous_on assms path_connected_continuous_image)

```

```

lemma is_interval_path_connected: is_interval  $S \implies$  path_connected  $S$ 
  by (simp add: convex_imp_path_connected is_interval_convex)

```

```

lemma path_connected_Ioi[simp]: path_connected  $\{a<..\}$  for  $a :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Ici[simp]: path_connected  $\{a.. \}$  for  $a :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Iio[simp]: path_connected  $\{..<a \}$  for  $a :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Iic[simp]: path_connected  $\{..a \}$  for  $a :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Ioo[simp]: path_connected  $\{a<..b \}$  for  $a b :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Ioc[simp]: path_connected  $\{a<..b \}$  for  $a b :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connected_Ico[simp]: path_connected  $\{a..b \}$  for  $a b :: real$ 
  by (simp add: convex_imp_path_connected)

```

```

lemma path_connectedin_path_image:
  assumes pathin  $X$   $g$  shows path_connectedin  $X$   $(g \ ` (\{0..1\}))$ 
  unfolding pathin_def
proof (rule path_connectedin_continuous_map_image)
  show continuous_map (subtopology euclideanreal  $\{0..1\}$ )  $X$   $g$ 
    using assms pathin_def by blast
qed (auto simp: is_interval_1 is_interval_path_connected)

```

```

lemma path_connected_space_subconnected:
  path_connected_space  $X \iff$ 
   $(\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \exists S. \text{path\_connectedin } X \ S \wedge x \in S \wedge y \in S)$ 
  by (metis path_connectedin_path_connectedin_topspace path_connected_space_def)

```

lemma *connectedin_path_image*: $\text{pathin } X \ g \implies \text{connectedin } X \ (g \text{ ‘ } (\{0..1\}))$
by (*simp add: path_connectedin_imp_connectedin path_connectedin_path_image*)

lemma *compactin_path_image*: $\text{pathin } X \ g \implies \text{compactin } X \ (g \text{ ‘ } (\{0..1\}))$
unfolding *pathin_def*
by (*rule image_compactin [of top_of_set {0..1}]*) *auto*

lemma *linear_homeomorphism_image*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *linear f inj f*
obtains g **where** *homeomorphism (f ‘ S) S g f*
proof –
obtain g **where** *linear g g o f = id*
using *assms linear_injective_left_inverse* **by** *blast*
then have *homeomorphism (f ‘ S) S g f*
using *assms unfolding homeomorphism_def*
by (*auto simp: eq_id_iff [symmetric] image_comp linear_conv_bounded_linear linear_continuous_on*)
then show *thesis ..*
qed

lemma *linear_homeomorphic_image*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *linear f inj f*
shows S *homeomorphic f ‘ S*
by (*meson homeomorphic_def homeomorphic_sym linear_homeomorphism_image [OF assms]*)

lemma *path_connected_Times*:
assumes *path_connected s path_connected t*
shows *path_connected (s × t)*
proof (*simp add: path_connected_def Sigma_def, clarify*)
fix $x1 \ y1 \ x2 \ y2$
assume $x1 \in s \ y1 \in t \ x2 \in s \ y2 \in t$
obtain g **where** *path g* **and** $g: \text{path_image } g \subseteq s$ **and** $gs: \text{pathstart } g = x1$ **and**
 $gf: \text{pathfinish } g = x2$
using $\langle x1 \in s \rangle \langle x2 \in s \rangle$ *assms* **by** (*force simp: path_connected_def*)
obtain h **where** *path h* **and** $h: \text{path_image } h \subseteq t$ **and** $hs: \text{pathstart } h = y1$ **and**
 $hf: \text{pathfinish } h = y2$
using $\langle y1 \in t \rangle \langle y2 \in t \rangle$ *assms* **by** (*force simp: path_connected_def*)
have *path* $(\lambda z. (x1, h z))$
using $\langle \text{path } h \rangle$
unfolding *path_def*
by (*intro continuous_intros continuous_on_compose2 [where g = Pair _]; force*)
moreover have *path* $(\lambda z. (g z, y2))$
using $\langle \text{path } g \rangle$

```

unfolding path_def
  by (intro continuous_intros continuous_on_compose2 [where g = Pair _];
force)
ultimately have 1: path ((λz. (x1, h z)) +++ (λz. (g z, y2)))
  by (metis hf gs path_join_imp pathstart_def pathfinish_def)
have path_image ((λz. (x1, h z)) +++ (λz. (g z, y2))) ⊆ path_image (λz. (x1,
h z)) ∪ path_image (λz. (g z, y2))
  by (rule Path_Connected.path_image_join_subset)
also have ... ⊆ (∪ x∈s. ∪ x1∈t. {(x, x1)})
  using g h ⟨x1 ∈ s⟩ ⟨y2 ∈ t⟩ by (force simp: path_image_def)
finally have 2: path_image ((λz. (x1, h z)) +++ (λz. (g z, y2))) ⊆ (∪ x∈s.
∪ x1∈t. {(x, x1)}) .
show ∃ g. path g ∧ path_image g ⊆ (∪ x∈s. ∪ x1∈t. {(x, x1)}) ∧
  pathstart g = (x1, y1) ∧ pathfinish g = (x2, y2)
  using 1 2 gf hs
  by (metis (no_types, lifting) pathfinish_def pathfinish_join pathstart_def path-
start_join)
qed

```

```

lemma is_interval_path_connected_1:
  fixes s :: real set
  shows is_interval s ⟷ path_connected s
  using is_interval_connected_1 is_interval_path_connected path_connected_imp_connected
by blast

```

6.1.18 Path components

```

lemma Union_path_component [simp]:
  Union {path_component_set S x | x. x ∈ S} = S
  using path_component_subset path_component_refl by blast

```

```

lemma path_component_disjoint:
  disjnt (path_component_set S a) (path_component_set S b) ⟷
  (a ⊄ path_component_set S b)
  unfolding disjnt_iff
  using path_component_sym path_component_trans by blast

```

```

lemma path_component_eq_eq:
  path_component S x = path_component S y ⟷
  (x ∉ S) ∧ (y ∉ S) ∨ x ∈ S ∧ y ∈ S ∧ path_component S x y
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs then show ?rhs
    by (metis (no_types) path_component_mem(1) path_component_refl)
next
  assume ?rhs then show ?lhs
  proof
    assume x ∉ S ∧ y ∉ S then show ?lhs
      by (metis Collect_empty_eq_bot path_component_eq_empty)

```



```

next
  assume  $S: x \in S \wedge y \in S \wedge \text{path\_component } S \ x \ y$  show ?lhs
  by (rule ext) (metis  $S \ \text{path\_component\_trans} \ \text{path\_component\_sym}$ )
qed
qed

lemma path_component_unique:
  assumes  $x \in C \ C \subseteq S \ \text{path\_connected } C$ 
     $\bigwedge C'. \llbracket x \in C'; C' \subseteq S; \text{path\_connected } C' \rrbracket \implies C' \subseteq C$ 
  shows  $\text{path\_component\_set } S \ x = C$ 
  by (smt (verit, best) Collect_cong assms  $\text{path\_component} \ \text{path\_component\_of\_subset}$ 
     $\text{path\_connected\_component\_set}$ )

```

```

lemma path_component_intermediate_subset:
   $\text{path\_component\_set } U \ a \subseteq T \wedge T \subseteq U$ 
   $\implies \text{path\_component\_set } T \ a = \text{path\_component\_set } U \ a$ 
  by (metis (no_types) path_component_mono path_component_path_component
    subset_antisym)

```

```

lemma complement_path_component_Union:
  fixes  $x :: 'a :: \text{topological\_space}$ 
  shows  $S - \text{path\_component\_set } S \ x =$ 
     $\bigcup (\{\text{path\_component\_set } S \ y \mid y. y \in S\} - \{\text{path\_component\_set } S \ x\})$ 
proof -
  have *:  $(\bigwedge x. x \in S - \{a\} \implies \text{disjnt } a \ x) \implies \bigcup S - a = \bigcup (S - \{a\})$ 
  for  $a :: 'a \ \text{set}$  and  $S$ 
  by (auto simp: disjnt_def)
  have  $\bigwedge y. y \in \{\text{path\_component\_set } S \ x \mid x. x \in S\} - \{\text{path\_component\_set } S \ x\}$ 
     $\implies \text{disjnt } (\text{path\_component\_set } S \ x) \ y$ 
  using path_component_disjoint path_component_eq by fastforce
  then have  $\bigcup \{\text{path\_component\_set } S \ x \mid x. x \in S\} - \text{path\_component\_set } S \ x$ 
  =
     $\bigcup (\{\text{path\_component\_set } S \ y \mid y. y \in S\} - \{\text{path\_component\_set } S \ x\})$ 
  by (meson *)
  then show ?thesis by simp
qed

```

6.1.19 Path components

```

definition path_component_of
  where  $\text{path\_component\_of } X \ x \ y \equiv \exists g. \text{pathin } X \ g \wedge g \ 0 = x \wedge g \ 1 = y$ 

```

```

abbreviation path_component_of_set
  where  $\text{path\_component\_of\_set } X \ x \equiv \text{Collect } (\text{path\_component\_of } X \ x)$ 

```

```

definition path_components_of ::  $'a \ \text{topology} \Rightarrow 'a \ \text{set} \ \text{set}$ 
  where  $\text{path\_components\_of } X \equiv \text{path\_component\_of\_set } X \ ` \ \text{topspace } X$ 

```

lemma *pathin_canon_iff*: $\text{pathin } (\text{top_of_set } T) \ g \longleftrightarrow \text{path } g \wedge g \in \{0..1\} \rightarrow T$

by (*simp add: path_def pathin_def image_subset_iff funcset*)

lemma *path_component_of_canon_iff* [*simp*]:

$\text{path_component_of } (\text{top_of_set } T) \ a \ b \longleftrightarrow \text{path_component } T \ a \ b$

by (*simp add: path_component_of_def pathin_canon_iff path_defs image_subset_iff funcset*)

lemma *path_component_in_topospace*:

$\text{path_component_of } X \ x \ y \implies x \in \text{topspace } X \wedge y \in \text{topspace } X$

by (*auto simp: path_component_of_def pathin_def continuous_map_def*)

lemma *path_component_of_refl*:

$\text{path_component_of } X \ x \ x \longleftrightarrow x \in \text{topspace } X$

by (*metis path_component_in_topospace path_component_of_def pathin_const*)

lemma *path_component_of_sym*:

assumes $\text{path_component_of } X \ x \ y$

shows $\text{path_component_of } X \ y \ x$

using *assms*

apply (*clarsimp simp: path_component_of_def pathin_def*)

apply (*rule_tac x=g o ($\lambda t. 1 - t$) in exI*)

apply (*auto intro!: continuous_map_compose simp: continuous_map_in_subtopology continuous_on_op_minus*)

done

lemma *path_component_of_sym_iff*:

$\text{path_component_of } X \ x \ y \longleftrightarrow \text{path_component_of } X \ y \ x$

by (*metis path_component_of_sym*)

lemma *continuous_map_cases_le*:

assumes *contp*: $\text{continuous_map } X \ \text{euclideanreal } p$

and *contq*: $\text{continuous_map } X \ \text{euclideanreal } q$

and *contf*: $\text{continuous_map } (\text{subtopology } X \ \{x. x \in \text{topspace } X \wedge p \ x \leq q \ x\})$

$Y \ f$

and *contg*: $\text{continuous_map } (\text{subtopology } X \ \{x. x \in \text{topspace } X \wedge q \ x \leq p \ x\})$

$Y \ g$

and *fg*: $\bigwedge x. \llbracket x \in \text{topspace } X; p \ x = q \ x \rrbracket \implies f \ x = g \ x$

shows $\text{continuous_map } X \ Y \ (\lambda x. \text{if } p \ x \leq q \ x \text{ then } f \ x \text{ else } g \ x)$

proof –

have $\text{continuous_map } X \ Y \ (\lambda x. \text{if } q \ x - p \ x \in \{0..\} \text{ then } f \ x \text{ else } g \ x)$

proof (*rule continuous_map_cases_function*)

show $\text{continuous_map } X \ \text{euclideanreal } (\lambda x. q \ x - p \ x)$

by (*intro contp contq continuous_intros*)

show $\text{continuous_map } (\text{subtopology } X \ \{x \in \text{topspace } X. q \ x - p \ x \in \text{euclideanreal closure_of } \{0..\}\}) \ Y \ f$

by (*simp add: contf*)

show $\text{continuous_map } (\text{subtopology } X \ \{x \in \text{topspace } X. q \ x - p \ x \in \text{euclideanreal closure_of } (\text{topspace euclideanreal} - \{0..\})\}) \ Y \ g$

```

    by (simp add: contg_flip: Compl_eq_Diff_UNIV)
  qed (auto simp: fg)
  then show ?thesis
    by simp
  qed

```

lemma *continuous_map_cases_lt:*

```

  assumes contp: continuous_map X euclideanreal p
    and contq: continuous_map X euclideanreal q
    and contf: continuous_map (subtopology X {x. x ∈ topspace X ∧ p x ≤ q x})
  Y f
    and contg: continuous_map (subtopology X {x. x ∈ topspace X ∧ q x ≤ p x})
  Y g
    and fg:  $\bigwedge x. [x \in \text{topspace } X; p \ x = q \ x] \implies f \ x = g \ x$ 
  shows continuous_map X Y ( $\lambda x. \text{if } p \ x < q \ x \text{ then } f \ x \text{ else } g \ x$ )
  proof -
    have continuous_map X Y ( $\lambda x. \text{if } q \ x - p \ x \in \{0 < ..\} \text{ then } f \ x \text{ else } g \ x$ )
    proof (rule continuous_map_cases_function)
      show continuous_map X euclideanreal ( $\lambda x. q \ x - p \ x$ )
        by (intro contp contq continuous_intros)
      show continuous_map (subtopology X {x ∈ topspace X. q x - p x ∈ euclideanreal
closure_of {0 < ..}}) Y f
        by (simp add: contf)
      show continuous_map (subtopology X {x ∈ topspace X. q x - p x ∈ euclideanreal
closure_of (topspace euclideanreal - {0 < ..}}) Y g
        by (simp add: contg_flip: Compl_eq_Diff_UNIV)
    qed (auto simp: fg)
  then show ?thesis
    by simp
  qed

```

lemma *path_component_of_trans:*

```

  assumes path_component_of X x y and path_component_of X y z
  shows path_component_of X x z
  unfolding path_component_of_def pathin_def
  proof -
    let ?T01 = top_of_set {0..1::real}
    obtain g1 g2 where g1: continuous_map ?T01 X g1 x = g1 0 y = g1 1
      and g2: continuous_map ?T01 X g2 g2 0 = g2 1 z = g2 1
      using assms unfolding path_component_of_def pathin_def by blast
    let ?g =  $\lambda x. \text{if } x \leq 1/2 \text{ then } (g1 \circ (\lambda t. 2 * t)) \ x \text{ else } (g2 \circ (\lambda t. 2 * t - 1)) \ x$ 
    show  $\exists g. \text{continuous\_map } ?T01 \ X \ g \wedge g \ 0 = x \wedge g \ 1 = z$ 
    proof (intro exI conjI)
      show continuous_map (subtopology euclideanreal {0..1}) X ?g
      proof (intro continuous_map_cases_le continuous_map_compose, force, force)
        show continuous_map (subtopology ?T01 {x ∈ topspace ?T01. x ≤ 1/2})
?T01 ((*)) 2)
        by (auto simp: continuous_map_in_subtopology continuous_map_from_subtopology)
      have continuous_map

```

```

      (subtopology (top_of_set {0..1}) {x. 0 ≤ x ∧ x ≤ 1 ∧ 1 ≤ x * 2})
      euclideanreal (λt. 2 * t - 1)
    by (intro continuous_intros) (force intro: continuous_map_from_subtopology)
    then show continuous_map (subtopology ?T01 {x ∈ topspace ?T01. 1/2 ≤
x}) ?T01 (λt. 2 * t - 1)
      by (force simp: continuous_map_in_subtopology)
    show (g1 ∘ (*) 2) x = (g2 ∘ (λt. 2 * t - 1)) x if x ∈ topspace ?T01 x =
1/2 for x
      using that by (simp add: g2(2) mult.commute continuous_map_from_subtopology)
    qed (auto simp: g1 g2)
  qed (auto simp: g1 g2)
qed

```

lemma *path_component_of_mono*:

```

[[path_component_of (subtopology X S) x y; S ⊆ T]] ⇒ path_component_of
(subtopology X T) x y
  unfolding path_component_of_def
  by (metis subsetD pathin_subtopology)

```

lemma *path_component_of*:

```

path_component_of X x y ↔ (∃ T. path_connectedin X T ∧ x ∈ T ∧ y ∈ T)
  (is ?lhs = ?rhs)

```

proof

assume ?lhs then show ?rhs

```

  by (metis atLeastAtMost_iff image_eqI order_refl path_component_of_def
path_connectedin_path_image zero_le_one)

```

next

assume ?rhs then show ?lhs

```

  by (metis path_component_of_def path_connectedin)

```

qed

lemma *path_component_of_set*:

```

path_component_of X x y ↔ (∃ g. pathin X g ∧ g 0 = x ∧ g 1 = y)
  by (auto simp: path_component_of_def)

```

lemma *path_component_of_subset_topspace*:

```

Collect(path_component_of X x) ⊆ topspace X
  using path_component_in_topspace by fastforce

```

lemma *path_component_of_eq_empty*:

```

Collect(path_component_of X x) = {} ↔ (x ∉ topspace X)
  using path_component_in_topspace path_component_of_refl by fastforce

```

lemma *path_connected_space_iff_path_component*:

```

path_connected_space X ↔ (∀ x ∈ topspace X. ∀ y ∈ topspace X. path_component_of
X x y)

```

```

  by (simp add: path_component_of_path_connected_space_subconnected)

```

lemma *path_connected_space_imp_path_component_of*:

```

[[path_connected_space X; a ∈ topspace X; b ∈ topspace X]]
  ⇒ path_component_of X a b
by (simp add: path_connected_space_iff_path_component)

lemma path_connected_space_path_component_set:
  path_connected_space X ↔ (∀ x ∈ topspace X. Collect(path_component_of X
x) = topspace X)
  using path_component_of_subset_topspace_path_connected_space_iff_path_component
by fastforce

lemma path_component_of_maximal:
  [[path_connectedin X s; x ∈ s]] ⇒ s ⊆ Collect(path_component_of X x)
  using path_component_of by fastforce

lemma path_component_of_equiv:
  path_component_of X x y ↔ x ∈ topspace X ∧ y ∈ topspace X ∧ path_component_of
X x = path_component_of X y
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding fun_eq_iff path_component_in_topspace
  by (metis path_component_in_topspace path_component_of_sym path_component_of_trans)
qed (simp add: path_component_of_refl)

lemma path_component_of_disjoint:
  disjnt (Collect (path_component_of X x)) (Collect (path_component_of X y))
↔
  ~ (path_component_of X x y)
  by (force simp: disjnt_def path_component_of_eq_empty path_component_of_equiv)

lemma path_component_of_eq:
  path_component_of X x = path_component_of X y ↔
  (x ∉ topspace X) ∧ (y ∉ topspace X) ∨
  x ∈ topspace X ∧ y ∈ topspace X ∧ path_component_of X x y
  by (metis Collect_empty_eq_bot path_component_of_eq_empty path_component_of_equiv)

lemma path_component_of_aux:
  path_component_of X x y
  ⇒ path_component_of (subtopology X (Collect (path_component_of X
x))) x y
  by (meson path_component_of_path_component_of_maximal path_connectedin_subtopology)

lemma path_connectedin_path_component_of:
  path_connectedin X (Collect (path_component_of X x))
proof -
  have topspace (subtopology X (path_component_of_set X x)) = path_component_of_set
X x
  by (meson path_component_of_subset_topspace topspace_subtopology_subset)

```

then have $\text{path_connected_space}$ (subtopology X (path_component_of_set X x))
by (metis mem_Collect_eq path_component_of_aux path_component_of_equiv
path_connected_space_iff_path_component)
then show ?thesis
by (simp add: path_component_of_subset_topspace path_connectedin_def)
qed

lemma path_connectedin_euclidean [simp]:
 $\text{path_connectedin euclidean } S \longleftrightarrow \text{path_connected } S$
by (auto simp: path_connectedin_def path_connected_space_iff_path_component
path_connected_component)

lemma path_connected_space_euclidean_subtopology [simp]:
 $\text{path_connected_space}(\text{subtopology euclidean } S) \longleftrightarrow \text{path_connected } S$
using path_connectedin_topspace **by** force

lemma Union_path_components_of:
 $\bigcup (\text{path_components_of } X) = \text{topspace } X$
by (auto simp: path_components_of_def path_component_of_equiv)

lemma path_components_of_maximal:
 $\llbracket C \in \text{path_components_of } X; \text{path_connectedin } X \ S; \sim \text{disjnt } C \ S \rrbracket \implies S \subseteq C$
by (smt (verit, ccfv_SIG) disjnt_iff_imageE mem_Collect_eq path_component_of_equiv

path_component_of_maximal path_components_of_def)

lemma pairwise_disjoint_path_components_of:
 $\text{pairwise disjnt } (\text{path_components_of } X)$
by (auto simp: path_components_of_def pairwise_def path_component_of_disjoint
path_component_of_equiv)

lemma complement_path_components_of_Union:
 $C \in \text{path_components_of } X \implies \text{topspace } X - C = \bigcup (\text{path_components_of } X - \{C\})$
by (metis Union_path_components_of bot.extremum ccpo_Sup_singleton diff_Union_pairwise_disjoin

insert_subsetI pairwise_disjoint_path_components_of)

lemma nonempty_path_components_of:
assumes $C \in \text{path_components_of } X$ **shows** $C \neq \{\}$
by (metis assms imageE path_component_of_eq_empty path_components_of_def)

lemma path_components_of_subset: $C \in \text{path_components_of } X \implies C \subseteq \text{topspace } X$
by (auto simp: path_components_of_def path_component_of_equiv)

lemma path_connectedin_path_components_of:
 $C \in \text{path_components_of } X \implies \text{path_connectedin } X \ C$

by (auto simp: path_components_of_def path_connectedin_path_component_of)

lemma path_component_in_path_components_of:

Collect (path_component_of X a) ∈ path_components_of X \longleftrightarrow a ∈ topspace X

by (metis imageI nonempty_path_components_of_path_component_of_eq_empty path_components_of_def)

lemma path_connectedin_Union:

assumes A: $\bigwedge S. S \in \mathcal{A} \implies \text{path_connectedin } X \ S$ and $\bigcap \mathcal{A} \neq \{\}$

shows path_connectedin X $(\bigcup \mathcal{A})$

proof –

obtain a where $\bigwedge S. S \in \mathcal{A} \implies a \in S$

using assms **by** blast

then have $\bigwedge x. x \in \text{topspace } (\text{subtopology } X \ (\bigcup \mathcal{A})) \implies \text{path_component_of } (\text{subtopology } X \ (\bigcup \mathcal{A})) \ a \ x$

unfolding topspace_subtopology_path_component_of

by (metis (full_types) IntD2 Union_iff Union_upper A path_connectedin_subtopology)

then show ?thesis

using A **unfolding** path_connectedin_def

by (metis Sup_le_iff path_component_of_equiv path_connected_space_iff_path_component)

qed

lemma path_connectedin_Un:

$\llbracket \text{path_connectedin } X \ S; \text{path_connectedin } X \ T; S \cap T \neq \{\} \rrbracket$

$\implies \text{path_connectedin } X \ (S \cup T)$

by (blast intro: path_connectedin_Union [of {S,T}, simplified])

lemma path_connected_space_iff_components_eq:

path_connected_space X \longleftrightarrow

$(\forall C \in \text{path_components_of } X. \forall C' \in \text{path_components_of } X. C = C')$

unfolding path_components_of_def

proof (intro iffI ballI)

assume $\forall C \in \text{path_component_of_set } X \ ' \ \text{topspace } X.$

$\forall C' \in \text{path_component_of_set } X \ ' \ \text{topspace } X. C = C'$

then show path_connected_space X

using path_component_of_refl path_connected_space_iff_path_component **by**

fastforce

qed (auto simp: path_connected_space_path_component_set)

lemma path_components_of_eq_empty:

path_components_of X = $\{\}$ \longleftrightarrow X = trivial_topology

by (metis image_is_empty path_components_of_def subtopology_eq_discrete_topology_empty)

lemma path_components_of_empty_space:

path_components_of trivial_topology = $\{\}$

by (simp add: path_components_of_eq_empty)

lemma path_components_of_subset_singleton:

1012

$path_components_of\ X \subseteq \{S\} \longleftrightarrow$
 $path_connected_space\ X \wedge (topspace\ X = \{\} \vee topspace\ X = S)$
proof (cases $topspace\ X = \{\}$)
 case *True*
 then show *?thesis*
 by (auto simp: $path_components_of_empty_space\ path_connected_space_topspace_empty$)
next
 case *False*
 have ($path_components_of\ X = \{S\}$) $\longleftrightarrow (path_connected_space\ X \wedge topspace\ X = S)$
 by (metis *False Set.set_insert ex_in_conv insert_iff path_component_in_path_components_of*

 $path_connected_space_iff_components_eq\ path_connected_space_path_component_set$)
 with *False show ?thesis*
 by (simp add: $path_components_of_eq_empty\ subset_singleton_iff$)
qed

lemma $path_connected_space_iff_components_subset_singleton$:
 $path_connected_space\ X \longleftrightarrow (\exists a. path_components_of\ X \subseteq \{a\})$
by (simp add: $path_components_of_subset_singleton$)

lemma $path_components_of_eq_singleton$:
 $path_components_of\ X = \{S\} \longleftrightarrow path_connected_space\ X \wedge topspace\ X \neq \{\} \wedge S = topspace\ X$
by (metis *cSup_singleton insert_not_empty path_components_of_subset_singleton subset_singleton_iff*)

lemma $path_components_of_path_connected_space$:
 $path_connected_space\ X \implies path_components_of\ X = (if\ topspace\ X = \{\} then\ \{\} else\ \{topspace\ X\})$
by (simp add: $path_components_of_eq_empty\ path_components_of_eq_singleton$)

lemma $path_component_subset_connected_component_of$:
 $path_component_of_set\ X\ x \subseteq connected_component_of_set\ X\ x$
proof (cases $x \in topspace\ X$)
 case *True*
 then show *?thesis*
 by (simp add: $connected_component_of_maximal\ path_component_of_refl\ path_connectedin_imp_connectedin\ path_connectedin_path_component_of$)
next
 case *False*
 then show *?thesis*
 using $path_component_of_eq_empty$ **by** *fastforce*
qed

lemma $exists_path_component_of_superset$:
assumes $S: path_connectedin\ X\ S$ **and** $ne: topspace\ X \neq \{\}$
obtains C **where** $C \in path_components_of\ X\ S \subseteq C$
by (metis $S\ ne\ ex_in_conv\ path_component_in_path_components_of\ path_component_of_maximal$)

path_component_of_subset_topspace subset_eq that)

lemma *path_component_of_eq_overlap:*

$path_component_of\ X\ x = path_component_of\ X\ y \longleftrightarrow$

$(x \notin topspace\ X) \wedge (y \notin topspace\ X) \vee$

$Collect\ (path_component_of\ X\ x) \cap Collect\ (path_component_of\ X\ y) \neq \{\}$

by (*metis disjoint_def empty_iff inf_bot_right mem_Collect_eq path_component_of_disjoint path_component_of_eq path_component_of_eq_empty*)

lemma *path_component_of_nonoverlap:*

$Collect\ (path_component_of\ X\ x) \cap Collect\ (path_component_of\ X\ y) = \{\}$

\longleftrightarrow

$(x \notin topspace\ X) \vee (y \notin topspace\ X) \vee$

$path_component_of\ X\ x \neq path_component_of\ X\ y$

by (*metis inf.idem path_component_of_eq_empty path_component_of_eq_overlap*)

lemma *path_component_of_overlap:*

$Collect\ (path_component_of\ X\ x) \cap Collect\ (path_component_of\ X\ y) \neq \{\}$

\longleftrightarrow

$x \in topspace\ X \wedge y \in topspace\ X \wedge path_component_of\ X\ x = path_component_of\ X\ y$

by (*meson path_component_of_nonoverlap*)

lemma *path_components_of_disjoint:*

$\llbracket C \in path_components_of\ X; C' \in path_components_of\ X \rrbracket \implies disjoint\ C\ C'$

$\longleftrightarrow C \neq C'$

by (*auto simp: path_components_of_def path_component_of_disjoint path_component_of_equiv*)

lemma *path_components_of_overlap:*

$\llbracket C \in path_components_of\ X; C' \in path_components_of\ X \rrbracket \implies C \cap C' \neq \{\}$

$\longleftrightarrow C = C'$

by (*auto simp: path_components_of_def path_component_of_equiv*)

lemma *path_component_of_unique:*

$\llbracket x \in C; path_connectedin\ X\ C; \bigwedge C'. \llbracket x \in C'; path_connectedin\ X\ C' \rrbracket \implies C' \subseteq C \rrbracket$

$\implies Collect\ (path_component_of\ X\ x) = C$

by (*meson subsetD_eq_iff path_component_of_maximal path_connectedin_path_component_of*)

lemma *path_component_of_discrete_topology [simp]:*

$Collect\ (path_component_of\ (discrete_topology\ U)\ x) = (if\ x \in U\ then\ \{x\}\ else\ \{\})$

proof –

have $\bigwedge C'. \llbracket x \in C'; path_connectedin\ (discrete_topology\ U)\ C' \rrbracket \implies C' \subseteq \{x\}$

by (*metis path_connectedin_discrete_topology subsetD singletonD*)

then have $x \in U \implies Collect\ (path_component_of\ (discrete_topology\ U)\ x) = \{x\}$

by (*simp add: path_component_of_unique*)

then show *?thesis*

1014

using *path_component_in_topospace* **by** *fastforce*
qed

lemma *path_component_of_discrete_topology_iff* [*simp*]:
 path_component_of (*discrete_topology* *U*) *x y* $\longleftrightarrow x \in U \wedge y=x$
by (*metis empty_iff insertI1 mem_Collect_eq path_component_of_discrete_topology singletonD*)

lemma *path_components_of_discrete_topology* [*simp*]:
 path_components_of (*discrete_topology* *U*) = $(\lambda x. \{x\}) \text{ ' } U$
by (*auto simp: path_components_of_def image_def fun_eq_iff*)

lemma *homeomorphic_map_path_component_of*:
 assumes *f: homeomorphic_map X Y f* **and** *x: x \in topspace X*
 shows *Collect (path_component_of Y (f x)) = f ' Collect(path_component_of X x)*

proof –

obtain *g* **where** *g: homeomorphic_maps X Y f g*

using *f homeomorphic_map_maps* **by** *blast*

show *?thesis*

proof

have *Collect (path_component_of Y (f x)) \subseteq topspace Y*

by (*simp add: path_component_of_subset_topospace*)

moreover have *g ' Collect(path_component_of Y (f x)) \subseteq Collect (path_component_of X (g (f x)))*

using *f g x unfolding homeomorphic_maps_def*

by (*metis image_Collect_subsetI image_eqI mem_Collect_eq path_component_of_equiv path_component_of_maximal*

path_connectedin_continuous_map_image path_connectedin_path_component_of)

ultimately show *Collect (path_component_of Y (f x)) \subseteq f ' Collect (path_component_of X x)*

using *g x unfolding homeomorphic_maps_def continuous_map_def image_iff subset_iff*

by *metis*

show *f ' Collect (path_component_of X x) \subseteq Collect (path_component_of Y (f x))*

proof (*rule path_component_of_maximal*)

show *path_connectedin Y (f ' Collect (path_component_of X x))*

by (*meson f homeomorphic_map_path_connectedness_eq path_connectedin_path_component_of*)

qed (*simp add: path_component_of_refl x*)

qed

qed

lemma *homeomorphic_map_path_components_of*:
 assumes *homeomorphic_map X Y f*
 shows *path_components_of Y = (image f) ' (path_components_of X)*
 unfolding *path_components_of_def homeomorphic_imp_surjective_map [OF assms, symmetric]*
 using *assms homeomorphic_map_path_component_of* **by** *fastforce*

6.1.20 Paths and path-connectedness

lemma *path_connected_space_quotient_map_image*:

$\llbracket \text{quotient_map } X \ Y \ q; \text{ path_connected_space } X \rrbracket \implies \text{path_connected_space } Y$
by (*metis path_connectedin_continuous_map_image path_connectedin_topspace*
quotient_imp_continuous_map quotient_imp_surjective_map)

lemma *path_connected_space_retraction_map_image*:

$\llbracket \text{retraction_map } X \ Y \ r; \text{ path_connected_space } X \rrbracket \implies \text{path_connected_space } Y$
using *path_connected_space_quotient_map_image retraction_imp_quotient_map*
by *blast*

lemma *path_connected_space_prod_topology*:

$\text{path_connected_space}(\text{prod_topology } X \ Y) \longleftrightarrow$
 $\text{topspace}(\text{prod_topology } X \ Y) = \{\} \vee \text{path_connected_space } X \wedge \text{path_connected_space } Y$

proof (*cases topspace(prod_topology X Y) = {}*)

case *True*

then show *?thesis*

using *path_connected_space_topspace_empty* **by** *force*

next

case *False*

have *path_connected_space (prod_topology X Y)*

if *X: path_connected_space X* **and** *Y: path_connected_space Y*

proof (*clarsimp simp: path_connected_space_def*)

fix *x y x' y'*

assume *x ∈ topspace X* **and** *y ∈ topspace Y* **and** *x' ∈ topspace X* **and** *y' ∈ topspace Y*

obtain *f* **where** *pathin X f f 0 = x f 1 = x'*

by (*meson X ⟨x ∈ topspace X⟩ ⟨x' ∈ topspace X⟩ path_connected_space_def*)

obtain *g* **where** *pathin Y g g 0 = y g 1 = y'*

by (*meson Y ⟨y ∈ topspace Y⟩ ⟨y' ∈ topspace Y⟩ path_connected_space_def*)

show $\exists h. \text{pathin } (\text{prod_topology } X \ Y) \ h \wedge h \ 0 = (x, y) \wedge h \ 1 = (x', y')$

proof (*intro exI conjI*)

show *pathin (prod_topology X Y) (λt. (f t, g t))*

using *⟨pathin X f⟩ ⟨pathin Y g⟩* **by** (*simp add: continuous_map_paired pathin_def*)

show $(\lambda t. (f \ t, \ g \ t)) \ 0 = (x, \ y)$

using *⟨f 0 = x⟩ ⟨g 0 = y⟩* **by** *blast*

show $(\lambda t. (f \ t, \ g \ t)) \ 1 = (x', \ y')$

using *⟨f 1 = x'⟩ ⟨g 1 = y'⟩* **by** *blast*

qed

qed

then show *?thesis*

by (*metis False path_connected_space_quotient_map_image prod_topology_trivial1*
prod_topology_trivial2

quotient_mapfst quotient_map_snd topspace_discrete_topology)

qed

lemma *path_connectedin_Times*:

$path_connectedin (prod_topology X Y) (S \times T) \longleftrightarrow$
 $S = \{\} \vee T = \{\} \vee path_connectedin X S \wedge path_connectedin Y T$
by (*auto simp add: path_connectedin_def subtopology_Times path_connected_space_prod_topology*)

6.1.21 Path components

lemma *path_component_of_subtopology_eq:*

$path_component_of (subtopology X U) x = path_component_of X x \longleftrightarrow path_component_of_set$
 $X x \subseteq U$
(is ?lhs = ?rhs)

proof

show *?lhs \implies ?rhs*

by (*metis path_connectedin_path_component_of path_connectedin_subtopology*)

next

show *?rhs \implies ?lhs*

unfolding *fun_eq_iff*

by (*metis path_connectedin_subtopology path_component_of path_component_of_aux path_component_of_mono*)

qed

lemma *path_components_of_subtopology:*

assumes $C \in path_components_of X C \subseteq U$

shows $C \in path_components_of (subtopology X U)$

using *assms path_component_of_refl path_component_of_subtopology_eq topspace_subtopology*

by (*smt (verit) imageE path_component_in_path_components_of path_components_of_def*)

lemma *path_imp_connected_component_of:*

$path_component_of X x y \implies connected_component_of X x y$

by (*metis in_mono mem_Collect_eq path_component_subset_connected_component_of*)

lemma *finite_path_components_of_finite:*

$finite(tospace X) \implies finite(path_components_of X)$

by (*simp add: Union_path_components_of_finite_UnionD*)

lemma *path_component_of_continuous_image:*

$\llbracket continuous_map X X' f; path_component_of X x y \rrbracket \implies path_component_of$
 $X' (f x) (f y)$

by (*meson image_eqI path_component_of path_connectedin_continuous_map_image*)

lemma *path_component_of_pair [simp]:*

$path_component_of_set (prod_topology X Y) (x,y) =$

$path_component_of_set X x \times path_component_of_set Y y$ **(is ?lhs = ?rhs)**

proof (*cases ?lhs = \{\}*)

case *True*

then show *?thesis*

by (*metis Sigma_empty1 Sigma_empty2 mem_Sigma_iff path_component_of_eq_empty topspace_prod_topology*)

next

case *False*

```

then have path_component_of X x path_component_of Y y
  using path_component_of_eq_empty path_component_of_refl by fastforce+
moreover
  have path_connectedin (prod_topology X Y) (path_component_of_set X x ×
path_component_of_set Y y)
    by (metis path_connectedin_Times path_connectedin_path_component_of)
  moreover have path_component_of X x a path_component_of Y y b
    if (x, y) ∈ C' (a,b) ∈ C' and path_connectedin (prod_topology X Y) C' for
C' a b
    by (smt (verit, best) that continuous_map_fst continuous_map_snd fst_conv
snd_conv path_component_of path_component_of_continuous_image)+
  ultimately show ?thesis
    by (intro path_component_of_unique) auto
qed

```

```

lemma path_components_of_prod_topology:
  path_components_of (prod_topology X Y) =
  (λ(C,D). C × D) ' (path_components_of X × path_components_of Y)
by (force simp add: image_iff path_components_of_def)

```

```

lemma path_components_of_prod_topology':
  path_components_of (prod_topology X Y) =
  {C × D | C D. C ∈ path_components_of X ∧ D ∈ path_components_of Y}
by (auto simp: path_components_of_prod_topology)

```

```

lemma path_component_of_product_topology:
  path_component_of_set (product_topology X I) f =
  (if f ∈ extensional I then PiE I (λi. path_component_of_set (X i) (f i)) else
  {})
  (is ?lhs = ?rhs)

```

```

proof (cases path_component_of_set (product_topology X I) f = {})
case True
  then show ?thesis
    by (smt (verit) PiE_eq_empty_iff PiE_iff path_component_of_eq_empty
topspace_product_topology)
next
case False
  then have [simp]: f ∈ extensional I
    by (auto simp: path_component_of_eq_empty PiE_iff path_component_of_equiv)
  show ?thesis
proof (intro path_component_of_unique)
  show f ∈ ?rhs
    using False path_component_of_eq_empty path_component_of_refl by force
  show path_connectedin (product_topology X I) (if f ∈ extensional I then ΠE
i∈I. path_component_of_set (X i) (f i) else {})
    by (simp add: path_connectedin_PiE path_connectedin_path_component_of)
  fix C'
  assume f ∈ C' and C': path_connectedin (product_topology X I) C'
  show C' ⊆ ?rhs

```

```

proof –
  have  $C' \subseteq \text{extensional } I$ 
    using  $\text{PiE\_def } C' \text{ path\_connectedin\_subset\_topspace}$  by fastforce
  with  $\langle f \in C' \rangle C'$  show ?thesis
    apply (clarsimp simp: PiE\_iff subset\_iff)
    by (smt (verit, ccfv\_threshold) continuous\_map\_product\_projection path\_component\_of
path\_component\_of\_continuous\_image)
  qed
qed
qed

```

```

lemma path\_components\_of\_product\_topology:
   $\text{path\_components\_of } (\text{product\_topology } X \ I) =$ 
   $\{\text{PiE } I \ B \mid B. \forall i \in I. B \ i \in \text{path\_components\_of } (X \ i)\}$  (is ?lhs=?rhs)

```

```

proof
  show ?lhs  $\subseteq$  ?rhs
    unfolding path\_components\_of\_def image\_subset\_iff
    by (smt (verit) image\_iff mem\_Collect\_eq path\_component\_of\_product\_topology
topspace\_product\_topology\_alt)
  next
    show ?rhs  $\subseteq$  ?lhs
    proof
      fix  $F$ 
      assume  $F \in ?rhs$ 
      then obtain  $B$  where  $B: F = \text{PiE } I \ B$ 
      and  $\forall i \in I. \exists x \in \text{topspace } (X \ i). B \ i = \text{path\_component\_of\_set } (X \ i) \ x$ 
      by (force simp add: path\_components\_of\_def image\_iff)
      then obtain  $f$  where  $f \text{top}: \bigwedge i. i \in I \implies f \ i \in \text{topspace } (X \ i)$ 
      and  $BF: \bigwedge i. i \in I \implies B \ i = \text{path\_component\_of\_set } (X \ i) \ (f \ i)$ 
      by metis
      then have  $F = \text{path\_component\_of\_set } (\text{product\_topology } X \ I) \ (\text{restrict } f \ I)$ 
      by (metis (mono\_tags, lifting) B PiE\_cong path\_component\_of\_product\_topology
restrict\_apply' restrict\_extensional)
      then show  $F \in ?lhs$ 
      by (simp add: ftop path\_component\_in\_path\_components\_of)
    qed
  qed

```

6.1.22 Sphere is path-connected

```

lemma path\_connected\_punctured\_universe:
  assumes  $2 \leq \text{DIM}('a::\text{euclidean\_space})$ 
  shows path\_connected  $(- \{a::'a\})$ 

```

```

proof –
  let  $?A = \{x::'a. \exists i \in \text{Basis}. x \cdot i < a \cdot i\}$ 
  let  $?B = \{x::'a. \exists i \in \text{Basis}. a \cdot i < x \cdot i\}$ 

```

```

  have  $A: \text{path\_connected } ?A$ 
    unfolding Collect\_bex\_eq

```

```

proof (rule path_connected_UNION)
  fix  $i :: 'a$ 
  assume  $i \in \text{Basis}$ 
  then show  $(\sum_{i \in \text{Basis}} (a \cdot i - 1) *_{\mathbb{R}} i) \in \{x :: 'a. x \cdot i < a \cdot i\}$ 
    by simp
  show path_connected  $\{x. x \cdot i < a \cdot i\}$ 
    using convex_imp_path_connected [OF convex_halfspace_lt, of  $i \ a \cdot i$ ]
    by (simp add: inner_commute)
qed
have  $B: \text{path\_connected } ?B$ 
  unfolding Collect_bex_eq
proof (rule path_connected_UNION)
  fix  $i :: 'a$ 
  assume  $i \in \text{Basis}$ 
  then show  $(\sum_{i \in \text{Basis}} (a \cdot i + 1) *_{\mathbb{R}} i) \in \{x :: 'a. a \cdot i < x \cdot i\}$ 
    by simp
  show path_connected  $\{x. a \cdot i < x \cdot i\}$ 
    using convex_imp_path_connected [OF convex_halfspace_gt, of  $a \cdot i \ i$ ]
    by (simp add: inner_commute)
qed
obtain  $S :: 'a \text{ set where } S \subseteq \text{Basis and card } S = \text{Suc } (\text{Suc } 0)$ 
  using obtain_subset_with_card_n [OF assms] by (force simp add: eval_nat_numeral)
then obtain  $b0 \ b1 :: 'a \text{ where } b0 \in \text{Basis and } b1 \in \text{Basis and } b0 \neq b1$ 
  unfolding card_Suc_eq by auto
then have  $a + b0 - b1 \in ?A \cap ?B$ 
  by (auto simp: inner_simps inner_Basis)
then have  $?A \cap ?B \neq \{\}$ 
  by fast
with  $A \ B$  have path_connected  $(?A \cup ?B)$ 
  by (rule path_connected_Un)
also have  $?A \cup ?B = \{x. \exists i \in \text{Basis}. x \cdot i \neq a \cdot i\}$ 
  unfolding neq_iff_bex_disj_distrib Collect_disj_eq ..
also have  $\dots = \{x. x \neq a\}$ 
  unfolding euclidean_eq_iff [where  $'a = 'a$ ]
  by (simp add: Bex_def)
also have  $\dots = - \{a\}$ 
  by auto
finally show ?thesis .
qed

corollary connected_punctured_universe:
   $2 \leq \text{DIM}('N :: \text{euclidean\_space}) \implies \text{connected}(- \{a :: 'N\})$ 
  by (simp add: path_connected_punctured_universe path_connected_imp_connected)

proposition path_connected_sphere:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $2 \leq \text{DIM}('a)$ 
  shows path_connected(sphere  $a \ r$ )
proof (cases  $r \ 0 :: \text{real}$  rule: linorder_cases)

```

```

case greater
then have eq: (sphere (0::'a) r) = ( $\lambda x. (r / \text{norm } x) *_R x$ ) ‘(- {0::'a})
  by (force simp: image_iff split: if_split_asm)
have continuous_on (- {0::'a}) ( $\lambda x. (r / \text{norm } x) *_R x$ )
  by (intro continuous_intros) auto
then have path_connected (( $\lambda x. (r / \text{norm } x) *_R x$ ) ‘(- {0::'a}))
  by (intro path_connected_continuous_image path_connected_punctured_universe
assms)
with eq have path_connected((+) a ‘(sphere (0::'a) r))
  by (simp add: path_connected_translation)
then show ?thesis
  by (metis add.right_neutral sphere_translation)
qed auto

```

```

lemma connected_sphere:
  fixes a :: 'a :: euclidean_space
  assumes  $2 \leq \text{DIM}(\text{'a})$ 
  shows connected(sphere a r)
using path_connected_sphere [OF assms]
by (simp add: path_connected_imp_connected)

```

```

corollary path_connected_complement_bounded_convex:
  fixes S :: 'a :: euclidean_space set
  assumes bounded S convex S and  $2: 2 \leq \text{DIM}(\text{'a})$ 
  shows path_connected (- S)
proof (cases S = {})
  case True then show ?thesis
    using convex_imp_path_connected by auto
  next
    case False
    then obtain a where a ∈ S by auto
    have § [rule_format]:  $\forall y \in S. \forall u. 0 \leq u \wedge u \leq 1 \longrightarrow (1 - u) *_R a + u *_R y \in S$ 
    using ⟨convex S⟩ ⟨a ∈ S⟩ by (simp add: convex_alt)
    { fix x y assume x ∉ S y ∉ S
      then have x ≠ a y ≠ a using ⟨a ∈ S⟩ by auto
      then have bxy: bounded(insert x (insert y S))
        by (simp add: convex_bounded)
      then obtain B::real where B:  $0 < B$  and Bx: norm (a - x) < B and By:
norm (a - y) < B
        and S ⊆ ball a B
      using bounded_subset_ballD [OF bxy, of a] by (auto simp: dist_norm)
      define C where C = B / norm(x - a)
      let ?Cx = a + C *_R (x - a)
      { fix u
        assume u:  $(1 - u) *_R x + u *_R ?Cx \in S$  and  $0 \leq u \leq 1$ 
        have CC:  $1 \leq 1 + (C - 1) * u$ 
          using ⟨x ≠ a⟩ ⟨ $0 \leq u$ ⟩ Bx

```



```

    by (auto simp add: C_def norm_minus_commute)
    have *:  $\bigwedge v. (1 - u) *_{\mathbb{R}} x + u *_{\mathbb{R}} (a + v *_{\mathbb{R}} (x - a)) = a + (1 + (v - 1) * u) *_{\mathbb{R}} (x - a)$ 
    by (simp add: algebra_simps)
    have a +  $((1 / (1 + C * u - u)) *_{\mathbb{R}} x + ((u / (1 + C * u - u)) *_{\mathbb{R}} a + (C * u / (1 + C * u - u)) *_{\mathbb{R}} x)) =$ 
       $(1 + (u / (1 + C * u - u))) *_{\mathbb{R}} a + ((1 / (1 + C * u - u)) + (C * u / (1 + C * u - u))) *_{\mathbb{R}} x$ 
    by (simp add: algebra_simps)
    also have ... =  $(1 + (u / (1 + C * u - u))) *_{\mathbb{R}} a + (1 + (u / (1 + C * u - u))) *_{\mathbb{R}} x$ 
    using CC by (simp add: field_simps)
    also have ... =  $x + (1 + (u / (1 + C * u - u))) *_{\mathbb{R}} a + (u / (1 + C * u - u)) *_{\mathbb{R}} x$ 
    by (simp add: algebra_simps)
    also have ... =  $x + ((1 / (1 + C * u - u)) *_{\mathbb{R}} a + ((u / (1 + C * u - u)) *_{\mathbb{R}} x + (C * u / (1 + C * u - u)) *_{\mathbb{R}} a))$ 
    using CC by (simp add: field_simps) (simp add: add_divide_distrib scaleR_add_left)
    finally have xeq:  $(1 - 1 / (1 + (C - 1) * u)) *_{\mathbb{R}} a + (1 / (1 + (C - 1) * u)) *_{\mathbb{R}} (a + (1 + (C - 1) * u) *_{\mathbb{R}} (x - a)) = x$ 
    by (simp add: algebra_simps)
    have False
    using  $\S [of a + (1 + (C - 1) * u) *_{\mathbb{R}} (x - a) 1 / (1 + (C - 1) * u)]$ 
    using  $u \langle x \neq a \rangle \langle x \notin S \rangle \langle 0 \leq u \rangle CC$ 
    by (auto simp: xeq *)
  }
  then have pcx: path_component (- S) x ?Cxa
  by (force simp: closed_segment_def intro!: path_component_linepath)
  define D where  $D = B / \text{norm}(y - a)$  — massive duplication with the proof above
  let ?Dya =  $a + D *_{\mathbb{R}} (y - a)$ 
  { fix u
    assume u:  $(1 - u) *_{\mathbb{R}} y + u *_{\mathbb{R}} ?Dya \in S$  and  $0 \leq u \leq 1$ 
    have DD:  $1 \leq 1 + (D - 1) * u$ 
    using  $\langle y \neq a \rangle \langle 0 \leq u \rangle By$ 
    by (auto simp add: D_def norm_minus_commute)
    have *:  $\bigwedge v. (1 - u) *_{\mathbb{R}} y + u *_{\mathbb{R}} (a + v *_{\mathbb{R}} (y - a)) = a + (1 + (v - 1) * u) *_{\mathbb{R}} (y - a)$ 
    by (simp add: algebra_simps)
    have a +  $((1 / (1 + D * u - u)) *_{\mathbb{R}} y + ((u / (1 + D * u - u)) *_{\mathbb{R}} a + (D * u / (1 + D * u - u)) *_{\mathbb{R}} y)) =$ 
       $(1 + (u / (1 + D * u - u))) *_{\mathbb{R}} a + ((1 / (1 + D * u - u)) + (D * u / (1 + D * u - u))) *_{\mathbb{R}} y$ 
    by (simp add: algebra_simps)
    also have ... =  $(1 + (u / (1 + D * u - u))) *_{\mathbb{R}} a + (1 + (u / (1 + D * u - u))) *_{\mathbb{R}} y$ 
    using DD by (simp add: field_simps)
    also have ... =  $y + (1 + (u / (1 + D * u - u))) *_{\mathbb{R}} a + (u / (1 + D * u - u)) *_{\mathbb{R}} y$ 

```

```

- u)) *R y
  by (simp add: algebra_simps)
  also have ... = y + ((1 / (1 + D * u - u)) *R a +
    ((u / (1 + D * u - u)) *R y + (D * u / (1 + D * u - u)) *R a))
    using DD by (simp add: field_simps) (simp add: add_divide_distrib
scaleR_add_left)
  finally have xeq: (1 - 1 / (1 + (D - 1) * u)) *R a + (1 / (1 + (D - 1)
* u)) *R (a + (1 + (D - 1) * u) *R (y - a)) = y
  by (simp add: algebra_simps)
  have False
  using § [of a + (1 + (D - 1) * u) *R (y - a) 1 / (1 + (D - 1) * u)]
  using u ⟨y ≠ a⟩ ⟨y ∉ S⟩ ⟨0 ≤ u⟩ DD
  by (auto simp: xeq *)
}
then have pdy: path_component (- S) y ?Dya
  by (force simp: closed_segment_def intro!: path_component_linepath)
have pyx: path_component (- S) ?Dya ?Cxa
proof (rule path_component_of_subset)
  show sphere a B ⊆ - S
  using ⟨S ⊆ ball a B⟩ by (force simp: ball_def dist_norm norm_minus_commute)
  have aB: ?Dya ∈ sphere a B ?Cxa ∈ sphere a B
  using ⟨x ≠ a⟩ using ⟨y ≠ a⟩ B by (auto simp: dist_norm C_def D_def)
  then show path_component (sphere a B) ?Dya ?Cxa
  using path_connected_sphere [OF 2] path_connected_component by blast
qed
have path_component (- S) x y
  by (metis path_component_trans path_component_sym pcx pdy pyx)
}
then show ?thesis
  by (auto simp: path_connected_component)
qed

lemma connected_complement_bounded_convex:
  fixes S :: 'a :: euclidean_space set
  assumes bounded S convex S 2 ≤ DIM('a)
  shows connected (- S)
  using path_connected_complement_bounded_convex [OF assms] path_connected_imp_connected
  by blast

lemma connected_diff_ball:
  fixes S :: 'a :: euclidean_space set
  assumes connected S cball a r ⊆ S 2 ≤ DIM('a)
  shows connected (S - ball a r)
proof (rule connected_diff_open_from_closed [OF ball_subset_cball])
  show connected (cball a r - ball a r)
  using assms connected_sphere by (auto simp: cball_diff_eq_sphere)
qed (auto simp: assms dist_norm)

proposition connected_open_delete:

```

```

  assumes open S connected S and 2: 2 ≤ DIM('N::euclidean_space)
    shows connected(S - {a::'N})
proof (cases a ∈ S)
  case True
  with ⟨open S⟩ obtain ε where ε > 0 and ε: cball a ε ⊆ S
    using open_contains_cball_eq by blast
  define b where b ≡ a + ε *R (SOME i. i ∈ Basis)
  have dist a b = ε
    by (simp add: b_def dist_norm SOME_Basis ⟨0 < ε⟩ less_imp_le)
  with ε have b ∈ ⋂{S - ball a r | r. 0 < r ∧ r < ε}
    by auto
  then have nonemp: (⋂{S - ball a r | r. 0 < r ∧ r < ε}) = {} ⇒ False
    by auto
  have con: ⋀r. r < ε ⇒ connected (S - ball a r)
    using ε by (force intro: connected_diff_ball [OF ⟨connected S⟩ _ 2])
  have x ∈ ⋃{S - ball a r | r. 0 < r ∧ r < ε} if x ∈ S - {a} for x
    using that ⟨0 < ε⟩
    by (intro UnionI [of S - ball a (min ε (dist a x) / 2)]) auto
  then have S - {a} = ⋃{S - ball a r | r. 0 < r ∧ r < ε}
    by auto
  then show ?thesis
    by (auto intro: connected_Union con dest!: nonemp)
next
  case False then show ?thesis
    by (simp add: ⟨connected S⟩)
qed

```

corollary *path_connected_open_delete:*

```

  assumes open S connected S and 2: 2 ≤ DIM('N::euclidean_space)
    shows path_connected(S - {a::'N})
  by (simp add: asms connected_open_delete connected_open_path_connected
    open_delete)

```

corollary *path_connected_punctured_ball:*

```

  2 ≤ DIM('N::euclidean_space) ⇒ path_connected(ball a r - {a::'N})
  by (simp add: path_connected_open_delete)

```

corollary *connected_punctured_ball:*

```

  2 ≤ DIM('N::euclidean_space) ⇒ connected(ball a r - {a::'N})
  by (simp add: connected_open_delete)

```

corollary *connected_open_delete_finite:*

```

  fixes S T::'a::euclidean_space set
  assumes S: open S connected S and 2: 2 ≤ DIM('a) and finite T
    shows connected(S - T)
  using ⟨finite T⟩ S
proof (induct T)
  case empty
  show ?case using ⟨connected S⟩ by simp

```

```

next
  case (insert x T)
  then have connected (S - T)
    by auto
  moreover have open (S - T)
    using finite_imp_closed[OF ‹finite T› ‹open S›] by auto
  ultimately have connected (S - T - {x})
    using connected_open_delete[OF _ _ 2] by auto
  thus ?case by (metis Diff_insert)
qed

```

```

lemma sphere_1D_doubleton_zero:
  assumes 1: DIM('a) = 1 and r > 0
  obtains x y::'a::euclidean_space
    where sphere 0 r = {x,y} ∧ dist x y = 2*r
proof -
  obtain b::'a where b: Basis = {b}
    using 1 card_1_singletonE by blast
  show ?thesis
  proof (intro that conjI)
    have x = norm x *R b ∨ x = - norm x *R b if r = norm x for x
  proof -
    have xb: (x · b) *R b = x
      using euclidean_representation [of x, unfolded b] by force
    then have norm ((x · b) *R b) = norm x
      by simp
    with b have |x · b| = norm x
      using norm_Basis by (simp add: b)
    with xb show ?thesis
      by (metis (mono_tags, opaque_lifting) abs_eq_iff abs_norm_cancel)
  qed
  with ‹r > 0› b show sphere 0 r = {r *R b, - r *R b}
    by (force simp: sphere_def dist_norm)
  have dist (r *R b) (- r *R b) = norm (r *R b + r *R b)
    by (simp add: dist_norm)
  also have ... = norm ((2*r) *R b)
    by (metis mult_2 scaleR_add_left)
  also have ... = 2*r
    using ‹r > 0› b norm_Basis by fastforce
  finally show dist (r *R b) (- r *R b) = 2*r .
qed
qed

```

```

lemma sphere_1D_doubleton:
  fixes a :: 'a :: euclidean_space
  assumes DIM('a) = 1 and r > 0
  obtains x y where sphere a r = {x,y} ∧ dist x y = 2*r
  using sphere_1D_doubleton_zero [OF assms] dist_add_cancel image_empty
  image_insert

```

```

  by (metis (no_types, opaque_lifting) add.right_neutral sphere_translation)

lemma psubset_sphere_Compl_connected:
  fixes S :: 'a::euclidean_space set
  assumes S: S ⊆ sphere a r and 0 < r and 2: 2 ≤ DIM('a)
  shows connected(− S)
proof −
  have S ⊆ sphere a r
    using S by blast
  obtain b where dist a b = r and b ∉ S
    using S mem_sphere by blast
  have CS: − S = {x. dist a x ≤ r ∧ (x ∉ S)} ∪ {x. r ≤ dist a x ∧ (x ∉ S)}
    by auto
  have {x. dist a x ≤ r ∧ x ∉ S} ∩ {x. r ≤ dist a x ∧ x ∉ S} ≠ {}
    using ⟨b ∉ S⟩ ⟨dist a b = r⟩ by blast
  moreover have connected {x. dist a x ≤ r ∧ x ∉ S}
    using assms
    by (force intro: connected_intermediate_closure [of ball a r])
  moreover have connected {x. r ≤ dist a x ∧ x ∉ S}
  proof (rule connected_intermediate_closure [of − cball a r])
    show {x. r ≤ dist a x ∧ x ∉ S} ⊆ closure (− cball a r)
      using interior_closure by (force intro: connected_complement_bounded_convex)
  qed (use assms connected_complement_bounded_convex in auto)
  ultimately show ?thesis
    by (simp add: CS connected_Un)
qed

```

6.1.23 Every annulus is a connected set

```

lemma path_connected_2DIM_I:
  fixes a :: 'N::euclidean_space
  assumes 2: 2 ≤ DIM('N) and pc: path_connected {r. 0 ≤ r ∧ P r}
  shows path_connected {x. P(norm(x − a))}
proof −
  have {x. P(norm(x − a))} = (+) a ‘ {x. P(norm x)}
    by force
  moreover have path_connected {x::'N. P(norm x)}
  proof −
    let ?D = {x. 0 ≤ x ∧ P x} × sphere (0::'N) 1
    have x ∈ (λz. fst z *R snd z) ‘ ?D
      if P (norm x) for x::'N
    proof (cases x=0)
      case True
      with that show ?thesis
        apply (simp add: image_iff)
      by (metis (no_types) mem_sphere_0 order_refl vector_choose_size zero_le_one)
    next
      case False
      with that show ?thesis

```

```

    by (rule_tac x=(norm x, x /R norm x) in image_eqI) auto
  qed
  then have *: {x::'N. P(norm x)} = (λz. fst z *R snd z) ' ?D
    by auto
  have continuous_on ?D (λz::real×'N. fst z *R snd z)
    by (intro continuous_intros)
  moreover have path_connected ?D
    by (metis path_connected_Times [OF pc] path_connected_sphere 2)
  ultimately show ?thesis
    by (simp add: * path_connected_continuous_image)
  qed
  ultimately show ?thesis
    using path_connected_translation by metis
  qed

```

proposition *path_connected_annulus*:

```

  fixes a :: 'N::euclidean_space
  assumes 2 ≤ DIM('N)
  shows path_connected {x. r1 < norm(x - a) ∧ norm(x - a) < r2}
        path_connected {x. r1 < norm(x - a) ∧ norm(x - a) ≤ r2}
        path_connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) < r2}
        path_connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) ≤ r2}
  by (auto simp: is_interval_def intro!: is_interval_convex convex_imp_path_connected
    path_connected_2DIM_I [OF assms])

```

proposition *connected_annulus*:

```

  fixes a :: 'N::euclidean_space
  assumes 2 ≤ DIM('N::euclidean_space)
  shows connected {x. r1 < norm(x - a) ∧ norm(x - a) < r2}
        connected {x. r1 < norm(x - a) ∧ norm(x - a) ≤ r2}
        connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) < r2}
        connected {x. r1 ≤ norm(x - a) ∧ norm(x - a) ≤ r2}
  by (auto simp: path_connected_annulus [OF assms] path_connected_imp_connected)

```

6.1.24 Relations between components and path components

lemma *open_connected_component*:

```

  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (connected_component_set S x)
proof (clarsimp simp: open_contains_ball)
  fix y
  assume xy: connected_component S x y
  then obtain e where e>0 ball y e ⊆ S
    using assms connected_component_in openE by blast
  then show ∃ e>0. ball y e ⊆ connected_component_set S x
    by (metis xy centre_in_ball connected_ball connected_component_eq_eq con-
      nected_component_in_connected_component_maximal)
  qed

```

corollary *open_components*:

fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\llbracket \text{open } u; S \in \text{components } u \rrbracket \implies \text{open } S$
by (*simp add: components_iff*) (*metis open_connected_component*)

lemma *in_closure_connected_component*:

fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $x: x \in S$ **and** $S: \text{open } S$
shows $x \in \text{closure } (\text{connected_component_set } S \ y) \longleftrightarrow x \in \text{connected_component_set } S \ y$
proof –
have $x \text{ islimpt } \text{connected_component_set } S \ y \implies \text{connected_component } S \ y \ x$
by (*metis (no_types, lifting) S connected_component_eq connected_component_refl islimptE mem_Collect_eq open_connected_component x*)
then show *?thesis*
by (*auto simp: closure_def*)

qed

lemma *connected_disjoint_Union_open_pick*:

assumes *pairwise_disjnt B*
 $\bigwedge S. S \in A \implies \text{connected } S \wedge S \neq \{\}$
 $\bigwedge S. S \in B \implies \text{open } S$
 $\bigcup A \subseteq \bigcup B$
 $S \in A$
obtains T **where** $T \in B \ S \subseteq T \ S \cap \bigcup (B - \{T\}) = \{\}$

proof –

have $S \subseteq \bigcup B \ \text{connected } S \ S \neq \{\}$
using *assms* $\langle S \in A \rangle$ **by** *blast+*
then obtain T **where** $T \in B \ S \cap T \neq \{\}$
by (*metis Sup_inf_eq_bot_iff inf.absorb_iff2 inf_commute*)
have $1: \text{open } T$ **by** (*simp add:* $\langle T \in B \rangle$ *assms*)
have $2: \text{open } (\bigcup (B - \{T\}))$ **using** *assms* **by** *blast*
have $3: S \subseteq T \cup \bigcup (B - \{T\})$ **using** $\langle S \subseteq \bigcup B \rangle$ **by** *blast*
have $T \cap \bigcup (B - \{T\}) = \{\}$ **using** $\langle T \in B \rangle$ $\langle \text{pairwise_disjnt } B \rangle$
by (*auto simp: pairwise_def disjnt_def*)
then have $4: T \cap \bigcup (B - \{T\}) \cap S = \{\}$ **by** *auto*
from *connectedD* [*OF* $\langle \text{connected } S \rangle$ $1 \ 2 \ 4 \ 3$]
have $S \cap \bigcup (B - \{T\}) = \{\}$
by (*auto simp: Int_commute* $\langle S \cap T \neq \{\} \rangle$)
with $\langle T \in B \rangle$ 3 **that** **show** *?thesis*
by (*metis IntI UnE empty_iff subsetD subsetI*)

qed

lemma *connected_disjoint_Union_open_subset*:

assumes $A: \text{pairwise_disjnt } A$ **and** $B: \text{pairwise_disjnt } B$
and $SA: \bigwedge S. S \in A \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$
and $SB: \bigwedge S. S \in B \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$
and *eq* [*simp*]: $\bigcup A = \bigcup B$

shows $A \subseteq B$
proof
fix S
assume $S \in A$
obtain T **where** $T \in B$ $S \subseteq T$ $S \cap \bigcup(B - \{T\}) = \{\}$
using SA SB $\langle S \in A \rangle$ *connected_disjoint_Union_open_pick* [*OF B, of A*] *eq*
order_refl **by** *blast*
moreover obtain S' **where** $S' \in A$ $T \subseteq S'$ $T \cap \bigcup(A - \{S'\}) = \{\}$
using SA SB $\langle T \in B \rangle$ *connected_disjoint_Union_open_pick* [*OF A, of B*] *eq*
order_refl **by** *blast*
ultimately have $S' = S$
by (*metis A Int_subset_iff SA* $\langle S \in A \rangle$ *disjnt_def inf.orderE pairwise_def*)
with $\langle T \subseteq S' \rangle$ **have** $T \subseteq S$ **by** *simp*
with $\langle S \subseteq T \rangle$ **have** $S = T$ **by** *blast*
with $\langle T \in B \rangle$ **show** $S \in B$ **by** *simp*
qed

lemma *connected_disjoint_Union_open_unique*:
assumes A : *pairwise disjnt A* **and** B : *pairwise disjnt B*
and SA : $\bigwedge S. S \in A \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$
and SB : $\bigwedge S. S \in B \implies \text{open } S \wedge \text{connected } S \wedge S \neq \{\}$
and *eq* [*simp*]: $\bigcup A = \bigcup B$
shows $A = B$
by (*metis subset_antisym connected_disjoint_Union_open_subset assms*)

proposition *components_open_unique*:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes *pairwise disjnt A* $\bigcup A = S$
 $\bigwedge X. X \in A \implies \text{open } X \wedge \text{connected } X \wedge X \neq \{\}$
shows *components S = A*
proof –
have *open S* **using** *assms* **by** *blast*
show *?thesis*
proof (*rule connected_disjoint_Union_open_unique*)
show *disjoint (components S)*
by (*simp add: components_eq disjnt_def pairwise_def*)
qed (*use* $\langle \text{open } S \rangle$ **in** $\langle \text{simp_all add: assms open_components in_components_connected in_components_nonempty} \rangle$)
qed

6.1.25 Existence of unbounded components

lemma *cobounded_unbounded_component*:
fixes $S :: 'a :: \text{euclidean_space_set}$
assumes *bounded* $(-S)$
shows $\exists x. x \in S \wedge \neg \text{bounded} (\text{connected_component_set } S \ x)$
proof –
obtain $i::'a$ **where** $i: i \in \text{Basis}$
using *nonempty_Basis* **by** *blast*


```

obtain  $B$  where  $B > 0 \ -S \subseteq \text{ball } 0 \ B$ 
  using bounded_subset_ballD [OF assms, of 0] by auto
then have  $*$ :  $\bigwedge x. B \leq \text{norm } x \implies x \in S$ 
  by (force simp: ball_def dist_norm)
have unbounded_inner:  $\neg \text{bounded } \{x. \text{inner } i \ x \geq B\}$ 
proof (clarsimp simp: bounded_def dist_norm)
  fix  $e \ x$ 
  show  $\exists y. B \leq i \cdot y \wedge \neg \text{norm } (x - y) \leq e$ 
  using  $i$ 
  by (rule_tac x=x + (max B e + 1 + |i \cdot x|) *R i in exI) (auto simp:
inner_right_distrib)
qed
have  $\S$ :  $\bigwedge x. B \leq i \cdot x \implies x \in S$ 
  using  $*$  Basis_le_norm [OF i] by (metis abs_ge_self inner_commute order_trans)
have  $\{x. B \leq i \cdot x\} \subseteq \text{connected\_component\_set } S \ (B *R i)$ 
  by (intro connected_component_maximal) (auto simp: i intro: convex_connected convex_halfspace_ge [of B] \S)
then have  $\neg \text{bounded } (\text{connected\_component\_set } S \ (B *R i))$ 
  using bounded_subset unbounded_inner by blast
moreover have  $B *R i \in S$ 
  by (rule *) (simp add: norm_Basis [OF i])
ultimately show ?thesis
  by blast
qed

lemma cobounded_unique_unbounded_component:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  assumes  $bs$ : bounded  $(-S)$  and  $2 \leq \text{DIM}('a)$ 
  and  $bo$ :  $\neg \text{bounded}(\text{connected\_component\_set } S \ x)$ 
   $\neg \text{bounded}(\text{connected\_component\_set } S \ y)$ 
  shows  $\text{connected\_component\_set } S \ x = \text{connected\_component\_set } S \ y$ 
proof -
  obtain  $i :: 'a$  where  $i: i \in \text{Basis}$ 
  using nonempty_Basis by blast
  obtain  $B$  where  $B > 0$  and  $B: -S \subseteq \text{ball } 0 \ B$ 
  using bounded_subset_ballD [OF bs, of 0] by auto
then have  $*$ :  $\bigwedge x. B \leq \text{norm } x \implies x \in S$ 
  by (force simp: ball_def dist_norm)
  obtain  $x' \ y'$  where  $x': \text{connected\_component } S \ x \ x' \ \text{norm } x' > B$ 
and  $y': \text{connected\_component } S \ y \ y' \ \text{norm } y' > B$ 
  using  $\langle B > 0 \rangle$  bo bounded_pos by (metis linorder_not_le mem_Collect_eq)
have  $x' y': \text{connected\_component } S \ x' \ y'$ 
  unfolding connected_component_def
proof (intro exI conjI)
  show connected  $(- \text{ball } 0 \ B :: 'a \text{ set})$ 
  using assms by (auto intro: connected_complement_bounded_convex)
qed (use x' y' dist_norm * in auto)
show ?thesis

```

using $x' y' x'y'$
by (*metis connected_component_eq mem_Collect_eq*)
qed

lemma *cobounded_unbounded_components*:
fixes $S :: 'a :: euclidean_space\ set$
shows $\text{bounded } (-S) \implies \exists c. c \in \text{components } S \wedge \neg \text{bounded } c$
by (*metis cobounded_unbounded_component components_def imageI*)

lemma *cobounded_unique_unbounded_components*:
fixes $S :: 'a :: euclidean_space\ set$
shows $\llbracket \text{bounded } (-S); c \in \text{components } S; \neg \text{bounded } c; c' \in \text{components } S; \neg \text{bounded } c'; 2 \leq \text{DIM}('a) \rrbracket \implies c' = c$
unfolding *components_iff*
by (*metis cobounded_unique_unbounded_component*)

lemma *cobounded_has_bounded_component*:
fixes $S :: 'a :: euclidean_space\ set$
assumes $\text{bounded } (-S) \neg \text{connected } S \ 2 \leq \text{DIM}('a)$
obtains C **where** $C \in \text{components } S \ \text{bounded } C$
by (*meson cobounded_unique_unbounded_components connected_eq_connected_components_eq assms*)

6.1.26 The *inside* and *outside* of a Set

The *inside* comprises the points in a bounded connected component of the set's complement. The *outside* comprises the points in unbounded connected component of the complement.

definition *inside where*
 $\text{inside } S \equiv \{x. (x \notin S) \wedge \text{bounded}(\text{connected_component_set } (-S) x)\}$

definition *outside where*
 $\text{outside } S \equiv -S \cap \{x. \neg \text{bounded}(\text{connected_component_set } (-S) x)\}$

lemma *outside*: $\text{outside } S = \{x. \neg \text{bounded}(\text{connected_component_set } (-S) x)\}$
by (*auto simp: outside_def*) (*metis Compl_iff bounded_empty connected_component_eq_empty*)

lemma *inside_no_overlap* [*simp*]: $\text{inside } S \cap S = \{\}$
by (*auto simp: inside_def*)

lemma *outside_no_overlap* [*simp*]:
 $\text{outside } S \cap S = \{\}$
by (*auto simp: outside_def*)

lemma *inside_Int_outside* [*simp*]: $\text{inside } S \cap \text{outside } S = \{\}$
by (*auto simp: inside_def outside_def*)

lemma *inside_Un_outside* [*simp*]: $\text{inside } S \cup \text{outside } S = (-S)$
by (*auto simp: inside_def outside_def*)

lemma *inside_eq_outside*:

$inside\ S = outside\ S \longleftrightarrow S = UNIV$
by (*auto simp: inside_def outside_def*)

lemma *inside_outside*: $inside\ S = \neg (S \cup outside\ S)$

by (*force simp: inside_def outside*)

lemma *outside_inside*: $outside\ S = \neg (S \cup inside\ S)$

by (*auto simp: inside_outside*) (*metis IntI equals0D outside_no_overlap*)

lemma *union_with_inside*: $S \cup inside\ S = \neg outside\ S$

by (*auto simp: inside_outside*) (*simp add: outside_inside*)

lemma *union_with_outside*: $S \cup outside\ S = \neg inside\ S$

by (*simp add: inside_outside*)

lemma *outside_mono*: $S \subseteq T \implies outside\ T \subseteq outside\ S$

by (*auto simp: outside bounded_subset connected_component_mono*)

lemma *inside_mono*: $S \subseteq T \implies inside\ S - T \subseteq inside\ T$

by (*auto simp: inside_def bounded_subset connected_component_mono*)

lemma *segment_bound_lemma*:

fixes $u::real$

assumes $x \geq B\ y \geq B\ 0 \leq u\ u \leq 1$

shows $(1 - u) * x + u * y \geq B$

by (*smt (verit) assms convex_bound_le ge_iff_diff_ge_0 minus_add_distrib mult_minus_right neg_le_iff_le*)

lemma *cobounded_outside*:

fixes $S :: 'a :: real_normed_vector\ set$

assumes *bounded S* **shows** $\neg outside\ S$

proof -

obtain B **where** $B > 0\ S \subseteq ball\ 0\ B$

using *bounded_subset_ballD [OF assms, of 0]* **by** *auto*

{ **fix** $x::'a$ **and** $C::real$

assume $Bno: B \leq norm\ x$ **and** $C: 0 < C$

have $\exists y. connected_component\ (\neg S)\ x\ y \wedge norm\ y > C$

proof (*cases x = 0*)

case *True* **with** $B\ Bno$ **show** *?thesis* **by** *force*

next

case *False*

have $closed_segment\ x\ (((B + C) / norm\ x) *_R\ x) \subseteq \neg ball\ 0\ B$

proof

fix w

assume $w \in closed_segment\ x\ (((B + C) / norm\ x) *_R\ x)$

then obtain u **where**

$w: w = (1 - u + u * (B + C) / norm\ x) *_R\ x\ 0 \leq u\ u \leq 1$

```

    by (auto simp add: closed_segment_def real_vector_class.scaleR_add_left
[symmetric])
  with False B C have  $B \leq (1 - u) * norm\ x + u * (B + C)$ 
    using segment_bound_lemma [of B norm x B + C u] Bno
    by simp
  with False B C show  $w \in -\ ball\ 0\ B$ 
    using distrib_right [of _ _ norm x]
    by (simp add: ball_def w not_less)
  qed
  also have  $\dots \subseteq -S$ 
    by (simp add: B)
  finally have  $\exists T. connected\ T \wedge T \subseteq -S \wedge x \in T \wedge ((B + C) / norm\ x)$ 
*_R x  $\in T$ 
    by (rule_tac x=closed_segment x (((B+C)/norm x) *_R x) in exI) simp
  with False B
  show ?thesis
  by (rule_tac x=((B+C)/norm x) *_R x in exI) (simp add: connected_component_def)
  qed
}
then show ?thesis
  apply (simp add: outside_def assms)
  apply (rule bounded_subset [OF bounded_ball [of 0 B]])
  apply (force simp: dist_norm not_less bounded_pos)
  done
qed

```

lemma *unbounded_outside:*

```

  fixes S :: 'a::{real_normed_vector, perfect_space} set
  shows bounded S  $\implies \neg bounded(outside\ S)$ 
  using cobounded_imp_unbounded cobounded_outside by blast

```

lemma *bounded_inside:*

```

  fixes S :: 'a::{real_normed_vector, perfect_space} set
  shows bounded S  $\implies bounded(inside\ S)$ 
  by (simp add: bounded_Int cobounded_outside inside_outside)

```

lemma *connected_outside:*

```

  fixes S :: 'a::euclidean_space set
  assumes bounded S  $2 \leq DIM('a)$ 
  shows connected(outside S)
  apply (clarsimp simp add: connected_iff_connected_component outside)
  apply (rule_tac S=connected_component_set (- S) x in connected_component_of_subset)
  apply (metis (no_types) assms cobounded_unbounded_component cobounded_unique_unbounded_com
connected_component_eq_eq connected_component_idemp double_complement mem_Collect_eq)
  by (simp add: Collect_mono connected_component_eq)

```

lemma *outside_connected_component_lt:*

```

  outside S = {x.  $\forall B. \exists y. B < norm(y) \wedge connected\_component\ (- S)\ x\ y$ }

```

proof –

```

have  $\bigwedge x B. x \in \text{outside } S \implies \exists y. B < \text{norm } y \wedge \text{connected\_component } (- S)$ 
 $x y$ 
  by (metis boundedI linorder_not_less mem_Collect_eq outside)
moreover
  have  $\bigwedge x. \forall B. \exists y. B < \text{norm } y \wedge \text{connected\_component } (- S) x y \implies x \in$ 
  outside S
  by (metis bounded_pos linorder_not_less mem_Collect_eq outside)
ultimately show ?thesis by auto
qed

```

lemma outside_connected_component_le:

```

  outside S = {x.  $\forall B. \exists y. B \leq \text{norm}(y) \wedge \text{connected\_component } (- S) x y$ }
apply (simp add: outside_connected_component_lt Set.set_eq_iff)
by (meson gt_ex leD le_less_linear less_imp_le order.trans)

```

lemma not_outside_connected_component_lt:

```

  fixes S :: 'a::euclidean_space set
  assumes S: bounded S and 2 ≤ DIM('a)
  shows - (outside S) = {x.  $\forall B. \exists y. B < \text{norm}(y) \wedge \neg \text{connected\_component}$ 
  (- S) x y}
proof -
  obtain B::real where B: 0 < B and Bno:  $\bigwedge x. x \in S \implies \text{norm } x \leq B$ 
  using S [simplified bounded_pos] by auto
  have cyz: connected_component (- S) y z
  if yz: B < norm z B < norm y for y::'a and z::'a
  proof -
  have connected_component (- cball 0 B) y z
  using assms yz
  by (force simp: dist_norm intro: connected_componentI [OF __ subset_refl]
  connected_complement_bounded_convex)
  then show ?thesis
  by (metis connected_component_of_subset Bno Compl_anti_mono mem_cball_0
  subset_iff)
  qed
  show ?thesis
  apply (auto simp: outside_bounded_pos)
  apply (metis Compl_iff bounded_iff cobounded_imp_unbounded mem_Collect_eq
  not_le)
  by (metis B connected_component_trans cyz not_le)
qed

```

lemma not_outside_connected_component_le:

```

  fixes S :: 'a::euclidean_space set
  assumes S: bounded S 2 ≤ DIM('a)
  shows - (outside S) = {x.  $\forall B. \exists y. B \leq \text{norm}(y) \wedge \neg \text{connected\_component } (-$ 
  S) x y}
unfolding not_outside_connected_component_lt [OF assms]
by (metis (no_types, opaque_lifting) dual_order.strict_trans1 gt_ex pinf(8))

```

lemma *inside_connected_component_lt*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes S : *bounded* S $2 \leq \text{DIM}('a)$
shows $\text{inside } S = \{x. (x \notin S) \wedge (\forall B. \exists y. B < \text{norm}(y) \wedge \neg \text{connected_component } (-S) x y)\}$
by (*auto simp: inside_outside not_outside_connected_component_lt [OF assms]*)

lemma *inside_connected_component_le*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes S : *bounded* S $2 \leq \text{DIM}('a)$
shows $\text{inside } S = \{x. (x \notin S) \wedge (\forall B. \exists y. B \leq \text{norm}(y) \wedge \neg \text{connected_component } (-S) x y)\}$
by (*auto simp: inside_outside not_outside_connected_component_le [OF assms]*)

lemma *inside_subset*:
assumes *connected* U **and** \neg *bounded* U **and** $T \cup U = -S$
shows $\text{inside } S \subseteq T$
using *bounded_subset [of connected_component_set (-S) U] assms*
by (*metis (no_types, lifting) ComplI Un_iff connected_component_maximal_inside_def mem_Collect_eq subsetI*)

lemma *frontier_not_empty*:
fixes $S :: 'a :: \text{real_normed_vector}$ *set*
shows $\llbracket S \neq \{\}; S \neq \text{UNIV} \rrbracket \implies \text{frontier } S \neq \{\}$
using *connected_Int_frontier [of UNIV S]* **by** *auto*

lemma *frontier_eq_empty*:
fixes $S :: 'a :: \text{real_normed_vector}$ *set*
shows $\text{frontier } S = \{\} \iff S = \{\} \vee S = \text{UNIV}$
using *frontier_UNIV frontier_empty frontier_not_empty* **by** *blast*

lemma *frontier_of_connected_component_subset*:
fixes $S :: 'a::\text{real_normed_vector}$ *set*
shows $\text{frontier}(\text{connected_component_set } S x) \subseteq \text{frontier } S$
proof –
{ **fix** y
assume $y1: y \in \text{closure}(\text{connected_component_set } S x)$
and $y2: y \notin \text{interior}(\text{connected_component_set } S x)$
have $y \in \text{closure } S$
using $y1$ *closure_mono connected_component_subset* **by** *blast*
moreover **have** $z \in \text{interior}(\text{connected_component_set } S x)$
if $0 < e$ $\text{ball } y e \subseteq \text{interior } S$ $\text{dist } y z < e$ **for** $e z$
proof –
have $\text{ball } y e \subseteq \text{connected_component_set } S y$
using *connected_component_maximal that interior_subset*
by (*metis centre_in_ball connected_ball subset_trans*)
then **show** *?thesis*
using $y1$ **apply** (*simp add: closure_approachable open_contains_ball_eq [OF open_interior]*)

```

    by (metis connected_component_eq dist_commute mem_Collect_eq mem_ball
        mem_interior subsetD ‹0 < e› y2)
  qed
  then have  $y \notin \text{interior } S$ 
    using y2 by (force simp: open_contains_ball_eq [OF open_interior])
  ultimately have  $y \in \text{frontier } S$ 
    by (auto simp: frontier_def)
}
then show ?thesis by (auto simp: frontier_def)
qed

```

lemma *frontier_Union_subset_closure*:

```

  fixes  $F :: 'a::\text{real\_normed\_vector set set}$ 
  shows  $\text{frontier}(\bigcup F) \subseteq \text{closure}(\bigcup t \in F. \text{frontier } t)$ 
  proof -
    have  $\exists y \in F. \exists y \in \text{frontier } y. \text{dist } y \ x < e$ 
      if  $T \in F \ y \in T \ \text{dist } y \ x < e$ 
       $x \notin \text{interior}(\bigcup F) \ 0 < e$  for  $x \ y \ e \ T$ 
    proof (cases  $x \in T$ )
      case True with that show ?thesis
        by (metis Diff_iff Sup_upper closure_subset contra_subsetD dist_self frontier_def interior_mono)
      next
      case False
        have  $\S: \text{closed\_segment } x \ y \cap T \neq \{\}$   $\text{closed\_segment } x \ y - T \neq \{\}$ 
          using ‹ $y \in T$ › False by blast+
        obtain  $c$  where  $c \in \text{closed\_segment } x \ y \ c \in \text{frontier } T$ 
          using False connected_Int_frontier [OF connected_segment  $\S$ ] by auto
        with that show ?thesis
          by (smt (verit) dist_norm segment_bound1)
    qed
  then show ?thesis
    by (fastforce simp add: frontier_def closure_approachable)
  qed

```

lemma *frontier_Union_subset*:

```

  fixes  $F :: 'a::\text{real\_normed\_vector set set}$ 
  shows  $\text{finite } F \implies \text{frontier}(\bigcup F) \subseteq (\bigcup t \in F. \text{frontier } t)$ 
  by (metis closed_UN closure_closed frontier_Union_subset_closure frontier_closed)

```

lemma *frontier_of_components_subset*:

```

  fixes  $S :: 'a::\text{real\_normed\_vector set}$ 
  shows  $C \in \text{components } S \implies \text{frontier } C \subseteq \text{frontier } S$ 
  by (metis Path_Connected.frontier_of_connected_component_subset components_iff)

```

lemma *frontier_of_components_closed_complement*:

```

  fixes  $S :: 'a::\text{real\_normed\_vector set}$ 
  shows  $\llbracket \text{closed } S; C \in \text{components } (- S) \rrbracket \implies \text{frontier } C \subseteq S$ 
  using frontier_complement frontier_of_components_subset frontier_subset_eq

```

by *blast*

lemma *frontier_minimal_separating_closed*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes *closed* S

and *nconn*: $\neg \text{connected}(-S)$

and $C: C \in \text{components}(-S)$

and *conn*: $\bigwedge T. \llbracket \text{closed } T; T \subset S \rrbracket \implies \text{connected}(-T)$

shows $\text{frontier } C = S$

proof (rule *ccontr*)

assume $\text{frontier } C \neq S$

then have $\text{frontier } C \subset S$

using *frontier_of_components_closed_complement* [*OF* $\langle \text{closed } S \rangle C$] **by** *blast*

then have $\text{connected}(-(\text{frontier } C))$

by (*simp add: conn*)

have $\neg \text{connected}(-(\text{frontier } C))$

unfolding *connected_def not_not*

proof (*intro exI conjI*)

show *open* C

using $C \langle \text{closed } S \rangle \text{open_components}$ **by** *blast*

show *open* $(- \text{closure } C)$

by *blast*

show $C \cap - \text{closure } C \cap - \text{frontier } C = \{\}$

using *closure_subset* **by** *blast*

show $C \cap - \text{frontier } C \neq \{\}$

using $C \langle \text{open } C \rangle \text{components_eq_frontier_disjoint_eq}$ **by** *fastforce*

show $- \text{frontier } C \subseteq C \cup - \text{closure } C$

by (*simp add: $\langle \text{open } C \rangle \text{closed_Compl_frontier_closures}$*)

then show $- \text{closure } C \cap - \text{frontier } C \neq \{\}$

by (*metis* $C \text{Compl_Diff_eq } Un_Int_eq(4) Un_commute \langle \text{frontier } C \subset S \rangle$

$\langle \text{open } C \rangle \text{compl_le_compl_iff_frontier_def_in_components_subset_interior_eq } leD$
sup_bot.right_neutral)

qed

then show *False*

using $\langle \text{connected}(-\text{frontier } C) \rangle$ **by** *blast*

qed

lemma *connected_component_UNIV* [*simp*]:

fixes $x :: 'a::\text{real_normed_vector}$

shows *connected_component_set* $UNIV\ x = UNIV$

using *connected_iff_eq_connected_component_set* [*of* $UNIV::'a\ \text{set}$] *connected_UNIV*

by *auto*

lemma *connected_component_eq_UNIV*:

fixes $x :: 'a::\text{real_normed_vector}$

shows *connected_component_set* $s\ x = UNIV \iff s = UNIV$

using *connected_component_in_connected_component_UNIV* **by** *blast*

lemma *components_UNIV* [*simp*]: $\text{components } UNIV = \{UNIV :: 'a::\text{real_normed_vector}\}$


```

set}
  by (auto simp: components_eq_sing_iff)

lemma interior_inside_frontier:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S
  shows interior S  $\subseteq$  inside (frontier S)
proof -
  { fix x y
    assume x: x  $\in$  interior S and y: y  $\notin$  S
    and cc: connected_component (- frontier S) x y
    have connected_component_set (- frontier S) x  $\cap$  frontier S  $\neq$  {}
    proof (rule connected_Int_frontier; simp add: set_eq_iff)
      show  $\exists u.$  connected_component (- frontier S) x u  $\wedge$  u  $\in$  S
      by (meson cc connected_component_in connected_component_refl_eq interior_subset subsetD x)
    show  $\exists u.$  connected_component (- frontier S) x u  $\wedge$  u  $\notin$  S
      using y cc by blast
    qed
    then have bounded (connected_component_set (- frontier S) x)
      using connected_component_in by auto
  }
  then show ?thesis
    using bounded_subset [OF assms]
    by (metis (no_types, lifting) Diff_iff frontier_def inside_def mem_Collect_eq subsetI)
qed

lemma inside_empty [simp]: inside {} = ({} :: 'a :: {real_normed_vector, perfect_space} set)
  by (simp add: inside_def)

lemma outside_empty [simp]: outside {} = (UNIV :: 'a :: {real_normed_vector, perfect_space} set)
  using inside_empty inside_Un_outside by blast

lemma inside_same_component:
   $\llbracket$ connected_component (- S) x y; x  $\in$  inside S $\rrbracket \implies y \in$  inside S
  using connected_component_eq connected_component_in
  by (fastforce simp add: inside_def)

lemma outside_same_component:
   $\llbracket$ connected_component (- S) x y; x  $\in$  outside S $\rrbracket \implies y \in$  outside S
  using connected_component_eq connected_component_in
  by (fastforce simp add: outside_def)

lemma convex_in_outside:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes S: convex S and z: z  $\notin$  S

```

```

    shows  $z \in \text{outside } S$ 
  proof (cases  $S = \{\}$ )
    case True then show ?thesis by simp
  next
    case False then obtain  $a$  where  $a \in S$  by blast
    with  $z$  have  $z \neq a$  by auto
    { assume bounded (connected_component_set (- S) z)
      with bounded_pos_less obtain  $B > 0$  and  $B$ :  $\bigwedge x. \text{connected\_component} (- S) z x \implies \text{norm } x < B$ 
        by (metis mem_Collect_eq)
      define  $C$  where  $C = (B + 1 + \text{norm } z) / \text{norm } (z - a)$ 
      have  $C > 0$ 
        using  $\langle 0 < B \rangle$   $z \neq a$  by (simp add: C_def field_split_simps add_strict_increasing)
      have  $|\text{norm } (z + C *_{\mathbb{R}} (z - a)) - \text{norm } (C *_{\mathbb{R}} (z - a))| \leq \text{norm } z$ 
        by (metis add_diff_cancel norm_triangle_ineq3)
      moreover have  $\text{norm } (C *_{\mathbb{R}} (z - a)) > \text{norm } z + B$ 
        using  $z \neq a$   $\langle B > 0 \rangle$  by (simp add: C_def le_max_iff_disj)
      ultimately have  $C$ :  $\text{norm } (z + C *_{\mathbb{R}} (z - a)) > B$  by linarith
      { fix  $u :: \text{real}$ 
        assume  $u$ :  $0 \leq u \leq 1$  and  $ins$ :  $(1 - u) *_{\mathbb{R}} z + u *_{\mathbb{R}} (z + C *_{\mathbb{R}} (z - a)) \in S$ 
        then have  $C_{pos}$ :  $1 + u * C > 0$ 
          by (meson  $\langle 0 < C \rangle$  add_pos_nonneg less_eq_real_def zero_le_mult_iff zero_less_one)
        then have  $*$ :  $(1 / (1 + u * C)) *_{\mathbb{R}} z + (u * C / (1 + u * C)) *_{\mathbb{R}} z = z$ 
          by (simp add: scaleR_add_left [symmetric] field_split_simps)
        then have False
          using convexD_alt [OF  $S \langle a \in S \rangle ins$ , of  $1 / (u * C + 1)$ ]  $\langle C > 0 \rangle \langle z \notin S \rangle$ 
           $C_{pos} u$ 
          by (simp add: * field_split_simps)
        } note contra = this
      have connected_component (- S) z (z + C *_{\mathbb{R}} (z - a))
      proof (rule connected_componentI [OF connected_segment])
        show closed_segment z (z + C *_{\mathbb{R}} (z - a))  $\subseteq - S$ 
          using contra by (force simp add: closed_segment_def)
        qed auto
      then have False
        using  $z \neq a$   $B$  [of  $z + C *_{\mathbb{R}} (z - a)$ ]  $C$ 
        by (auto simp: field_split_simps max_mult_distrib_right)
      }
    } then show ?thesis
      by (auto simp: outside_def z)
  qed

```

lemma *outside_convex*:

```

  fixes  $S :: 'a :: \{\text{real\_normed\_vector}, \text{perfect\_space}\}$  set
  assumes convex  $S$ 
  shows  $\text{outside } S = - S$ 
  by (metis ComplD assms convex_in_outside equalityI inside_Un_outside subsetI sup.cobounded2)

```

```

lemma outside_singleton [simp]:
  fixes  $x :: 'a :: \{real\_normed\_vector, perfect\_space\}$ 
  shows  $outside \{x\} = -\{x\}$ 
  by (auto simp: outside_convex)

lemma inside_convex:
  fixes  $S :: 'a :: \{real\_normed\_vector, perfect\_space\}$  set
  shows  $convex S \implies inside S = \{\}$ 
  by (simp add: inside_outside outside_convex)

lemma inside_singleton [simp]:
  fixes  $x :: 'a :: \{real\_normed\_vector, perfect\_space\}$ 
  shows  $inside \{x\} = \{\}$ 
  by (auto simp: inside_convex)

lemma outside_subset_convex:
  fixes  $S :: 'a :: \{real\_normed\_vector, perfect\_space\}$  set
  shows  $\llbracket convex T; S \subseteq T \rrbracket \implies - T \subseteq outside S$ 
  using outside_convex outside_mono by blast

lemma outside_Un_outside_Un:
  fixes  $S :: 'a :: real\_normed\_vector$  set
  assumes  $S \cap outside(T \cup U) = \{\}$ 
  shows  $outside(T \cup U) \subseteq outside(T \cup S)$ 
proof
  fix  $x$ 
  assume  $x: x \in outside (T \cup U)$ 
  have  $Y \subseteq - S$  if  $connected Y$   $Y \subseteq - T$   $Y \subseteq - U$   $x \in Y$   $u \in Y$  for  $u \in Y$ 
  proof -
    have  $Y \subseteq connected\_component\_set (- (T \cup U)) x$ 
    by (simp add: connected_component_maximal that)
    also have  $\dots \subseteq outside(T \cup U)$ 
    by (metis (mono_tags, lifting) Collect_mono mem_Collect_eq outside_outside_same_component x)
    finally have  $Y \subseteq outside(T \cup U)$  .
    with assms show ?thesis by auto
  qed
  with  $x$  show  $x \in outside (T \cup S)$ 
  by (simp add: outside_connected_component_lt connected_component_def)
meson
qed

lemma outside_frontier_misses_closure:
  fixes  $S :: 'a :: real\_normed\_vector$  set
  assumes bounded  $S$ 
  shows  $outside(frontier S) \subseteq - closure S$ 
  using assms frontier_def interior_inside_frontier outside_inside by fastforce

```

```

lemma outside_frontier_eq_complement_closure:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes bounded S convex S
  shows outside(frontier S) = - closure S
by (metis Diff_subset assms convex_closure frontier_def outside_frontier_misses_closure
    outside_subset_convex subset_antisym)

```

```

lemma inside_frontier_eq_interior:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  shows  $\llbracket$ bounded S; convex S $\rrbracket \implies$  inside(frontier S) = interior S
unfolding inside_outside outside_frontier_eq_complement_closure
using closure_subset interior_subset by (auto simp: frontier_def)

```

```

lemma open_inside:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows open (inside S)
proof -
  { fix x assume x: x  $\in$  inside S
    have open (connected_component_set (- S) x)
      using assms open_connected_component by blast
    then obtain e where e: e>0 and e:  $\bigwedge y. \text{dist } y \ x < e \longrightarrow$  connected_component
      (- S) x y
      using dist_not_less_zero
      apply (simp add: open_dist)
      by (metis (no_types, lifting) Compl_iff connected_component_refl_eq in-
        side_def mem_Collect_eq x)
    then have  $\exists e>0. \text{ball } x \ e \subseteq$  inside S
      by (metis e dist_commute inside_same_component mem_ball subsetI x)
  }
  then show ?thesis
    by (simp add: open_contains_ball)
qed

```

```

lemma open_outside:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
  shows open (outside S)
proof -
  { fix x assume x: x  $\in$  outside S
    have open (connected_component_set (- S) x)
      using assms open_connected_component by blast
    then obtain e where e: e>0 and e:  $\bigwedge y. \text{dist } y \ x < e \longrightarrow$  connected_component
      (- S) x y
      using dist_not_less_zero x
      by (auto simp add: open_dist outside_def intro: connected_component_refl)
    then have  $\exists e>0. \text{ball } x \ e \subseteq$  outside S
      by (metis e dist_commute outside_same_component mem_ball subsetI x)
  }

```

```

then show ?thesis
  by (simp add: open_contains_ball)
qed

```

```

lemma closure_inside_subset:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes closed  $S$ 
  shows  $\text{closure}(\text{inside } S) \subseteq S \cup \text{inside } S$ 
  by (metis assms closure_minimal open_closed open_outside sup.cobounded2 union_with_inside)

```

```

lemma frontier_inside_subset:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes closed  $S$ 
  shows  $\text{frontier}(\text{inside } S) \subseteq S$ 
  using assms closure_inside_subset frontier_closures frontier_disjoint_eq open_inside
  by fastforce

```

```

lemma closure_outside_subset:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes closed  $S$ 
  shows  $\text{closure}(\text{outside } S) \subseteq S \cup \text{outside } S$ 
  by (metis assms closed_open closure_minimal inside_outside open_inside sup_ge2)

```

```

lemma closed_path_image_Un_inside:
  fixes  $g :: real \Rightarrow 'a :: real\_normed\_vector$ 
  assumes path  $g$ 
  shows closed  $(\text{path\_image } g \cup \text{inside } (\text{path\_image } g))$ 
  by (simp add: assms closed_Comp closed_path_image open_outside union_with_inside)

```

```

lemma frontier_outside_subset:
  fixes  $S :: 'a::real\_normed\_vector\ set$ 
  assumes closed  $S$ 
  shows  $\text{frontier}(\text{outside } S) \subseteq S$ 
  unfolding frontier_def
  by (metis Diff_subset_conv assms closure_outside_subset interior_eq open_outside sup_aci(1))

```

```

lemma inside_complement_unbounded_connected_empty:
   $\llbracket \text{connected } (- S); \neg \text{bounded } (- S) \rrbracket \Longrightarrow \text{inside } S = \{\}$ 
  using inside_subset by blast

```

```

lemma inside_bounded_complement_connected_empty:
  fixes  $S :: 'a::\{real\_normed\_vector, perfect\_space\}\ set$ 
  shows  $\llbracket \text{connected } (- S); \text{bounded } S \rrbracket \Longrightarrow \text{inside } S = \{\}$ 
  by (metis inside_complement_unbounded_connected_empty cobounded_imp_unbounded)

```

```

lemma inside_inside:
  assumes  $S \subseteq \text{inside } T$ 
  shows  $\text{inside } S - T \subseteq \text{inside } T$ 

```

unfolding *inside_def*

proof *clarify*

fix *x*

assume *x: x ∉ T x ∉ S and bo: bounded (connected_component_set (- S) x)*

show *bounded (connected_component_set (- T) x)*

proof *(cases S ∩ connected_component_set (- T) x = {})*

case *True then show ?thesis*

by *(metis bounded_subset [OF bo] compl_le_compl_iff connected_component_idemp connected_component_mono disjoint_eq_subset_Cmpl double_compl)*

next

case *False*

then obtain *y where y: y ∈ S y ∈ connected_component_set (- T) x*

by *(meson disjoint_iff)*

then have *bounded (connected_component_set (- T) y)*

using *assms [unfolded inside_def] by blast*

with *y show ?thesis*

by *(metis connected_component_eq)*

qed

qed

lemma *inside_inside_subset: inside (inside S) ⊆ S*

using *inside_inside union_with_outside by fastforce*

lemma *inside_outside_intersect_connected:*

[[connected T; inside S ∩ T ≠ {}; outside S ∩ T ≠ {}]] ⇒ S ∩ T ≠ {}

apply *(simp add: inside_def outside_def ex_in_conv [symmetric] disjoint_eq_subset_Cmpl, clarify)*

by *(metis compl_le_swap1 connected_componentI connected_component_eq mem_Collect_eq)*

lemma *outside_bounded_nonempty:*

fixes *S :: 'a :: {real_normed_vector, perfect_space} set*

assumes *bounded S shows outside S ≠ {}*

using *assms unbounded_outside by force*

lemma *outside_compact_in_open:*

fixes *S :: 'a :: {real_normed_vector, perfect_space} set*

assumes *S: compact S and T: open T and S ⊆ T T ≠ {}*

shows *outside S ∩ T ≠ {}*

proof *-*

have *outside S ≠ {}*

by *(simp add: compact_imp_bounded outside_bounded_nonempty S)*

with *assms obtain a b where a: a ∈ outside S and b: b ∈ T by auto*

show *?thesis*

proof *(cases a ∈ T)*

case *True with a show ?thesis by blast*

next

case *False*

have *front: frontier T ⊆ - S*

using *⟨S ⊆ T⟩ frontier_disjoint_eq T by auto*

```

{ fix  $\gamma$ 
  assume path  $\gamma$  and pimng_sbs: path_image  $\gamma - \{\text{pathfinish } \gamma\} \subseteq \text{interior}$ 
  ( $- T$ )
    and pf: pathfinish  $\gamma \in \text{frontier } T$  and ps: pathstart  $\gamma = a$ 
  define c where c = pathfinish  $\gamma$ 
  have c  $\in -S$  unfolding c_def using front pf by blast
  moreover have open ( $-S$ ) using S compact_imp_closed by blast
  ultimately obtain  $\varepsilon::\text{real}$  where  $\varepsilon > 0$  and  $\varepsilon: \text{cball } c \varepsilon \subseteq -S$ 
    using open_contains_cball[of  $-S$ ] S by blast
  then obtain d where d  $\in T$  and d: dist d c  $< \varepsilon$ 
    using closure_approachable [of c T] pf unfolding c_def
    by (metis Diff_iff frontier_def)
  then have d  $\in -S$  using  $\varepsilon$ 
    using dist_commute by (metis contra_subsetD mem_cball not_le
not_less_iff_gr_or_eq)
  have pimng_sbs_cos: path_image  $\gamma \subseteq -S$ 
    using  $\langle c \in -S \rangle \langle S \subseteq T \rangle$  c_def interior_subset pimng_sbs by fastforce
  have closed_segment c d  $\leq \text{cball } c \varepsilon$ 
    by (metis  $\langle 0 < \varepsilon \rangle$  centre_in_cball closed_segment_subset convex_cball d
dist_commute less_eq_real_def mem_cball)
  with  $\varepsilon$  have closed_segment c d  $\subseteq -S$  by blast
  moreover have con_gcd: connected (path_image  $\gamma \cup \text{closed\_segment } c d$ )
  by (rule connected_Un) (auto simp: c_def  $\langle \text{path } \gamma \rangle$  connected_path_image)
  ultimately have connected_component ( $- S$ ) a d
    unfolding connected_component_def using pimng_sbs_cos ps by blast
  then have outside S  $\cap T \neq \{\}$ 
    using outside_same_component [OF  $- a$ ] by (metis IntI  $\langle d \in T \rangle$ 
empty_iff)
} note * = this
have pal: pathstart (linepath a b)  $\in \text{closure } (- T)$ 
  by (auto simp: False closure_def)
show ?thesis
  by (rule exists_path_subpath_to_frontier [OF path_linepath pal  $- *$ ]) (auto
simp: b)
qed
qed

```

lemma inside_inside_compact_connected:

```

fixes S :: 'a :: euclidean_space set
assumes S: closed S and T: compact T and connected T S  $\subseteq \text{inside } T$ 
shows inside S  $\subseteq \text{inside } T$ 
proof (cases inside T =  $\{\}$ )
case True with assms show ?thesis by auto
next
case False
consider DIM('a) = 1 | DIM('a)  $\geq 2$ 
  using antisym_not_less_eq_eq by fastforce
then show ?thesis
proof cases

```

```

    case 1 then show ?thesis
      using connected_convex_1_gen assms False inside_convex by blast
next
case 2
have bounded S
using assms by (meson bounded_inside bounded_subset compact_imp_bounded)
then have coms: compact S
  by (simp add: S compact_eq_bounded_closed)
then have bst: bounded (S ∪ T)
  by (simp add: compact_imp_bounded T)
then obtain r where 0 < r and r: S ∪ T ⊆ ball 0 r
  using bounded_subset_ballD by blast
have outst: outside S ∩ outside T ≠ {}
by (metis bounded_Un bounded_subset bst cobounded_outside disjoint_eq_subset_Compl
unbounded_outside)
have S ∩ T = {} using assms
  by (metis disjoint_iff_not_equal inside_no_overlap subsetCE)
moreover have outside S ∩ inside T ≠ {}
  by (meson False assms(4) compact_eq_bounded_closed coms open_inside
outside_compact_in_open T)
ultimately have inside S ∩ T = {}
  using inside_outside_intersect_connected [OF ⟨connected T⟩, of S]
  by (metis 2 compact_eq_bounded_closed coms connected_outside inf commute
inside_outside_intersect_connected outst)
then show ?thesis
  using inside_inside [OF ⟨S ⊆ inside T⟩] by blast
qed
qed

lemma connected_with_inside:
  fixes S :: 'a :: real_normed_vector set
  assumes S: closed S and cons: connected S
  shows connected(S ∪ inside S)
proof (cases S ∪ inside S = UNIV)
case True with assms show ?thesis by auto
next
case False
  then obtain b where b: b ∉ S b ∉ inside S by blast
  have *: ∃ y T. y ∈ S ∧ connected T ∧ a ∈ T ∧ y ∈ T ∧ T ⊆ (S ∪ inside S)
    if a ∈ S ∪ inside S for a
    using that
  proof
  assume a ∈ S then show ?thesis
    using cons by blast
  next
  assume a: a ∈ inside S
  then have ain: a ∈ closure (inside S)
    by (simp add: closure_def)
  obtain h where h: path h pathstart h = a

```



```

      path_image h - {pathfinish h}  $\subseteq$  interior (inside S)
      pathfinish h  $\in$  frontier (inside S)
    using ain b
  by (metis exists_path_subpath_to_frontier path_linepath pathfinish_linepath
pathstart_linepath)
  moreover
  have h1S: pathfinish h  $\in$  S
    using S h frontier_inside_subset by blast
  moreover
  have path_image h  $\subseteq$  S  $\cup$  inside S
    using IntD1 S h1S h interior_eq open_inside by fastforce
  ultimately show ?thesis by blast
qed
show ?thesis
  apply (simp add: connected_iff_connected_component)
  apply (clarsimp simp add: connected_component_def dest!: *)
  subgoal for x y u u' T t'
    by (rule_tac x = S  $\cup$  T  $\cup$  t' in exI) (auto intro!: connected_Un cons)
  done
qed

```

The proof is virtually the same as that above.

```

lemma connected_with_outside:
  fixes S :: 'a :: real_normed_vector set
  assumes S: closed S and cons: connected S
  shows connected(S  $\cup$  outside S)
proof (cases S  $\cup$  outside S = UNIV)
  case True with assms show ?thesis by auto
next
  case False
  then obtain b where b: b  $\notin$  S b  $\notin$  outside S by blast
  have *:  $\exists y T. y \in S \wedge$  connected T  $\wedge a \in T \wedge y \in T \wedge T \subseteq$  (S  $\cup$  outside S)
  if a  $\in$  (S  $\cup$  outside S) for a
  using that proof
  assume a  $\in$  S then show ?thesis
    by (rule_tac x=a in exI, rule_tac x={a} in exI, simp)
  next
  assume a: a  $\in$  outside S
  then have ain: a  $\in$  closure (outside S)
    by (simp add: closure_def)
  obtain h where h: path h pathstart h = a
    path_image h - {pathfinish h}  $\subseteq$  interior (outside S)
    pathfinish h  $\in$  frontier (outside S)
  using ain b
  by (metis exists_path_subpath_to_frontier path_linepath pathfinish_linepath
pathstart_linepath)
  moreover
  have h1S: pathfinish h  $\in$  S
    using S frontier_outside_subset h(4) by blast

```

```

moreover
  have path_image  $h \subseteq S \cup \text{outside } S$ 
    using IntD1 S h1S h interior_eq open_outside by fastforce
  ultimately show ?thesis
    by blast
qed
show ?thesis
  apply (simp add: connected_iff_connected_component)
  apply (clarsimp simp add: connected_component_def dest!: *)
  subgoal for  $x\ y\ u\ u'\ T\ t'$ 
    by (rule_tac x=(S  $\cup$  T  $\cup$  t') in exI) (auto intro!: connected_Un cons)
  done
qed

lemma inside_inside_eq_empty [simp]:
  fixes  $S :: 'a :: \{\text{real\_normed\_vector, perfect\_space}\}$  set
  assumes  $S$ : closed S and cons: connected S
  shows inside (inside S) = {}
proof –
  have connected (– inside S)
    by (metis S connected_with_outside cons union_with_outside)
  then show ?thesis
    by (metis bounded_Un inside_complement_unbounded_connected_empty unbounded_outside union_with_outside)
qed

lemma inside_in_components:
   $\text{inside } S \in \text{components } (– S) \iff \text{connected}(\text{inside } S) \wedge \text{inside } S \neq \{\}$  (is
?lhs = ?rhs)
proof
  assume  $R$ : ?rhs
  then have  $\bigwedge x. \llbracket x \in S; x \in \text{inside } S \rrbracket \implies \neg \text{connected } (\text{inside } S)$ 
    by (simp add: inside_outside)
  with  $R$  show ?lhs
    unfolding in_components_maximal
    by (auto intro: inside_same_component connected_componentI)
qed (simp add: in_components_maximal)

```

The proof is like that above.

```

lemma outside_in_components:
   $\text{outside } S \in \text{components } (– S) \iff \text{connected}(\text{outside } S) \wedge \text{outside } S \neq \{\}$  (is
?lhs = ?rhs)
proof
  assume  $R$ : ?rhs
  then have  $\bigwedge x. \llbracket x \in S; x \in \text{outside } S \rrbracket \implies \neg \text{connected } (\text{outside } S)$ 
    by (meson disjoint_iff outside_no_overlap)
  with  $R$  show ?lhs
    unfolding in_components_maximal
    by (auto intro: outside_same_component connected_componentI)

```

qed (simp add: in_components_maximal)

lemma bounded_unique_outside:

fixes $S :: 'a :: \text{euclidean_space set}$

assumes $\text{bounded } S \text{ DIM}('a) \geq 2$

shows $(c \in \text{components } (- S) \wedge \neg \text{bounded } c) \longleftrightarrow c = \text{outside } S$

using *assms*

by (metis *cobounded_unique_unbounded_components_connected_outside_double_compl_outside_bounded_nonempty_outside_in_components_unbounded_outside*)

6.1.27 Condition for an open map's image to contain a ball

proposition *ball_subset_open_map_image*:

fixes $f :: 'a :: \text{heine_borel} \Rightarrow 'b :: \{\text{real_normed_vector, heine_borel}\}$

assumes *contf*: *continuous_on* (closure S) f

and *oint*: *open* ($f \text{ ` interior } S$)

and *le_no*: $\bigwedge z. z \in \text{frontier } S \implies r \leq \text{norm}(f z - f a)$

and *bounded* $S a \in S \ 0 < r$

shows $\text{ball } (f a) r \subseteq f \text{ ` } S$

proof (cases $f \text{ ` } S = \text{UNIV}$)

case *True* then show *?thesis* by *simp*

next

case *False*

then have *closed* (*frontier* ($f \text{ ` } S$)) *frontier* ($f \text{ ` } S$) $\neq \{\}$

using $\langle a \in S \rangle$ by (auto *simp*: *frontier_eq_empty*)

then obtain w where $w: w \in \text{frontier } (f \text{ ` } S)$

and *dw_le*: $\bigwedge y. y \in \text{frontier } (f \text{ ` } S) \implies \text{norm } (f a - w) \leq \text{norm } (f a - y)$

by (auto *simp* add: *dist_norm_intro*: *distance_attains_inf* [of *frontier*($f \text{ ` } S$) $f a$])

then obtain ξ where $\xi: \bigwedge n. \xi n \in f \text{ ` } S$ and *tendsw*: $\xi \longrightarrow w$

by (metis *Diff_iff_frontier_def_closure_sequential*)

then have $\bigwedge n. \exists x \in S. \xi n = f x$ by *force*

then obtain z where $zs: \bigwedge n. z n \in S$ and *fz*: $\bigwedge n. \xi n = f (z n)$

by *metis*

then obtain $y K$ where $y: y \in \text{closure } S$ and *strict_mono* ($K :: \text{nat} \Rightarrow \text{nat}$)

and *Klim*: $(z \circ K) \longrightarrow y$

using $\langle \text{bounded } S \rangle$

unfolding *compact_closure* [*symmetric*] *compact_def* by (meson *closure_subset_subset_iff*)

then have *ftendsw*: $((\lambda n. f (z n)) \circ K) \longrightarrow w$

by (metis *LIMSEQ_subseq_LIMSEQ* *fun.map_cong0* *fz* *tendsw*)

have *zKs*: $\bigwedge n. (z \circ K) n \in S$ by (simp add: *zs*)

have *fz*: $f \circ z = \xi \ (\lambda n. f (z n)) = \xi$

using *fz* by *auto*

then have $(\xi \circ K) \longrightarrow f y$

by (metis (no_types) *Klim* *zKs* *y* *contf* *comp_assoc* *continuous_on_closure_sequentially*)

with *fz* have *wy*: $w = f y$ using *fz* *LIMSEQ_unique* *ftendsw* by *auto*

have $r \leq \text{norm } (f y - f a)$

```

proof (rule le_no)
  show  $y \in \text{frontier } S$ 
  using  $w$   $wy$  oint by (force simp: imageI image_mono interiorI interior_subset
frontier_def y)
qed
then have  $\bigwedge y. \llbracket \text{norm } (f a - y) < r; y \in \text{frontier } (f ' S) \rrbracket \implies \text{False}$ 
  by (metis dw_le norm_minus_commute not_less order_trans wy)
then have  $\text{ball } (f a) r \cap \text{frontier } (f ' S) = \{\}$ 
  by (metis disjoint_iff_not_equal dist_norm mem_ball)
moreover
have  $\text{ball } (f a) r \cap f ' S \neq \{\}$ 
  using  $\langle a \in S \rangle \langle 0 < r \rangle$  centre_in_ball by blast
ultimately show ?thesis
  by (meson connected_Int_frontier connected_ball diff_shunt_var)
qed

```

Special characterizations of classes of functions into and out of \mathbf{R} .

lemma Hausdorff_space_euclidean [simp]: Hausdorff_space (euclidean :: 'a::metric_space topology)

```

proof -
  have  $\exists U V. \text{open } U \wedge \text{open } V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U V$ 
  if  $x \neq y$  for  $x y :: 'a$ 
  proof (intro exI conjI)
  let  $?r = \text{dist } x y / 2$ 
  have [simp]:  $?r > 0$ 
  by (simp add: that)
  show  $\text{open } (\text{ball } x ?r) \text{ open } (\text{ball } y ?r) x \in (\text{ball } x ?r) y \in (\text{ball } y ?r)$ 
  by (auto simp add: that)
  show  $\text{disjnt } (\text{ball } x ?r) (\text{ball } y ?r)$ 
  unfolding disjnt_def by (simp add: disjoint_ballI)
qed
then show ?thesis
  by (simp add: Hausdorff_space_def)
qed

```

proposition embedding_map_into_euclideanreal:

assumes path_connected_space X
shows embedding_map X euclideanreal $f \longleftrightarrow$
continuous_map X euclideanreal $f \wedge \text{inj_on } f$ (topspace X)

proof safe

show continuous_map X euclideanreal f

if embedding_map X euclideanreal f

using continuous_map_in_subtopology homeomorphic_imp_continuous_map

that

unfolding embedding_map_def **by** blast

show inj_on f (topspace X)

if embedding_map X euclideanreal f

using that homeomorphic_imp_injective_map

```

  unfolding embedding_map_def by blast
show embedding_map X euclideanreal f
  if cont: continuous_map X euclideanreal f and inj: inj_on f (topspace X)
proof -
  obtain g where gf:  $\bigwedge x. x \in \text{topspace } X \implies g (f x) = x$ 
    using inv_into_f_f [OF inj] by auto
  show ?thesis
  unfolding embedding_map_def homeomorphic_map_maps homeomorphic_maps_def
proof (intro exI conjI)
  show continuous_map X (top_of_set (f ' topspace X)) f
    by (simp add: cont continuous_map_in_subtopology)
  let ?S = f ' topspace X
  have eq:  $\{x \in ?S. g x \in U\} = f ' U$  if openin X U for U
    using openin_subset [OF that] by (auto simp: gf)
  have 1:  $g ' ?S \subseteq \text{topspace } X$ 
    using eq by blast
  have openin (top_of_set ?S)  $\{x \in ?S. g x \in T\}$ 
    if openin X T for T
  proof -
    have  $T \subseteq \text{topspace } X$ 
      by (simp add: openin_subset that)
    have RR:  $\forall x \in ?S \cap g - ' T. \exists d > 0. \forall x' \in ?S \cap \text{ball } x \ d. g x' \in T$ 
      proof (clarsimp simp add: gf)
        have pcS: path_connectedin euclidean ?S
          using assms cont path_connectedin_continuous_map_image path_connectedin_topspace
        by blast
      show  $\exists d > 0. \forall x' \in f ' \text{topspace } X \cap \text{ball } (f x) \ d. g x' \in T$ 
        if  $x \in T$  for  $x$ 
      proof -
        have  $x: x \in \text{topspace } X$ 
          using  $\langle T \subseteq \text{topspace } X \rangle \langle x \in T \rangle$  by blast
        obtain  $u \ v \ d$  where  $0 < d \ u \in \text{topspace } X \ v \in \text{topspace } X$ 
          and sub_fuv:  $?S \cap \{f x - d .. f x + d\} \subseteq \{f u..f v\}$ 
        proof (cases  $\exists u \in \text{topspace } X. f u < f x$ )
        case True
        then obtain  $u$  where  $u: u \in \text{topspace } X \ f u < f x ..$ 
        show ?thesis
        proof (cases  $\exists v \in \text{topspace } X. f x < f v$ )
        case True
        then obtain  $v$  where  $v: v \in \text{topspace } X \ f x < f v ..$ 
        show ?thesis
        proof
        let  $?d = \min (f x - f u) (f v - f x)$ 
        show  $0 < ?d$ 
          by (simp add:  $\langle f u < f x \rangle \langle f x < f v \rangle$ )
        show  $f ' \text{topspace } X \cap \{f x - ?d..f x + ?d\} \subseteq \{f u..f v\}$ 
          by fastforce
        qed (auto simp: u v)
      next

```

```

case False
show ?thesis
proof
  let ?d = f x - f u
  show  $0 < ?d$ 
  by (simp add: u)
  show  $f \text{ ' } \textit{topspace } X \cap \{f x - ?d..f x + ?d\} \subseteq \{f u..f x\}$ 
  using x u False by auto
qed (auto simp: x u)
qed
next
case False
note no_u = False
show ?thesis
proof (cases  $\exists v \in \textit{topspace } X. f x < f v$ )
  case True
  then obtain v where v: v \in \textit{topspace } X f x < f v ..
  show ?thesis
  proof
    let ?d = f v - f x
    show  $0 < ?d$ 
    by (simp add: v)
    show  $f \text{ ' } \textit{topspace } X \cap \{f x - ?d..f x + ?d\} \subseteq \{f x..f v\}$ 
    using False by auto
  qed (auto simp: x v)
  next
  case False
  show ?thesis
  proof
    show  $f \text{ ' } \textit{topspace } X \cap \{f x - 1..f x + 1\} \subseteq \{f x..f x\}$ 
    using False no_u by fastforce
  qed (auto simp: x)
  qed
then obtain h where pathin X h h 0 = u h 1 = v
  using assms unfolding path_connected_space_def by blast
obtain C where compactin X C connectedin X C u \in C v \in C
proof
  show compactin X (h \text{ ' } \{0..1\})
  using that by (simp add: \langle pathin X h \rangle compactin_path_image)
  show connectedin X (h \text{ ' } \{0..1\})
  using  $\langle \textit{pathin } X h \rangle \textit{connectedin\_path\_image}$  by blast
qed (use \langle h 0 = u \rangle \langle h 1 = v \rangle in auto)
have continuous_map (subtopology euclideanreal (?S \cap \{f x - d .. f x + d\})) (subtopology X C) g
proof (rule continuous_inverse_map)
  show compact_space (subtopology X C)
  using  $\langle \textit{compactin } X C \rangle \textit{compactin\_subspace}$  by blast
  show continuous_map (subtopology X C) euclideanreal f

```

```

    by (simp add: cont continuous_map_from_subtopology)
  have  $\{f\ u \ ..\ f\ v\} \subseteq f\ ' \ topspace\ (subtopology\ X\ C)$ 
  proof (rule connected_contains_Icc)
    show connected  $(f\ ' \ topspace\ (subtopology\ X\ C))$ 
      using connectedin_continuous_map_image [OF cont]
      by (simp add:  $\langle compactin\ X\ C \rangle \langle connectedin\ X\ C \rangle compactin\_subset\_topspace\ inf\_absorb2$ )
    show  $f\ u \in f\ ' \ topspace\ (subtopology\ X\ C)$ 
      by (simp add:  $\langle u \in C \rangle \langle u \in topspace\ X \rangle$ )
    show  $f\ v \in f\ ' \ topspace\ (subtopology\ X\ C)$ 
      by (simp add:  $\langle v \in C \rangle \langle v \in topspace\ X \rangle$ )
  qed
  then show  $f\ ' \ topspace\ X \cap \{f\ x - d .. f\ x + d\} \subseteq f\ ' \ topspace\ (subtopology\ X\ C)$ 
    using sub_fuv by blast
  qed (auto simp: gf)
  then have contg: continuous_map (subtopology euclideanreal  $(?S \cap \{f\ x - d .. f\ x + d\})$ )  $X\ g$ 
    using continuous_map_in_subtopology by blast
  have  $\exists e > 0. \forall x \in ?S \cap \{f\ x - d .. f\ x + d\} \cap ball\ (f\ x)\ e. g\ x \in T$ 
    using openin_continuous_map_preimage [OF contg  $\langle openin\ X\ T \rangle$ ]  $x \in T \rangle \langle 0 < d \rangle$ 
    unfolding openin_euclidean_subtopology_iff
    by (force simp: gf dist_commute)
  then obtain e where  $e > 0 \wedge (\forall x \in f\ ' \ topspace\ X \cap \{f\ x - d .. f\ x + d\} \cap ball\ (f\ x)\ e. g\ x \in T)$ 
    by metis
  with  $\langle 0 < d \rangle$  have  $min\ d\ e > 0 \forall u. u \in topspace\ X \longrightarrow |f\ x - f\ u| < min\ d\ e \longrightarrow u \in T$ 
    using dist_real_def gf by force+
  then show ?thesis
    by (metis (full_types) Int_iff dist_real_def image_iff mem_ball gf)
  qed
  qed
  then obtain d where  $d: \bigwedge r. r \in ?S \cap g\ -\ ' \ T \implies d\ r > 0 \wedge (\forall x \in ?S \cap ball\ r\ (d\ r). g\ x \in T)$ 
    by metis
  show ?thesis
    unfolding openin_subtopology
  proof (intro exI conjI)
    show  $\{x \in ?S. g\ x \in T\} = (\bigcup r \in ?S \cap g\ -\ ' \ T. ball\ r\ (d\ r)) \cap f\ ' \ topspace\ X$ 
      using d by (auto simp: gf)
    qed auto
  qed
  then show continuous_map (top_of_set ?S)  $X\ g$ 
    by (simp add: 1 continuous_map)
  qed (auto simp: gf)
  qed

```

qed

An injective function into \mathbb{R} is a homeomorphism and so an open map.

lemma *injective_into_1d_eq_homeomorphism*:

fixes $f :: 'a::\text{topological_space} \Rightarrow \text{real}$

assumes f : *continuous_on* S f **and** S : *path_connected* S

shows *inj_on* f $S \iff (\exists g. \text{homeomorphism } S (f \text{ ' } S) f g)$

proof

show $\exists g. \text{homeomorphism } S (f \text{ ' } S) f g$

if *inj_on* f S

proof –

have *embedding_map* (*top_of_set* S) *euclideanreal* f

using *that_embedding_map_into_euclideanreal* [*of_top_of_set* S f] *assms* **by**

auto

then show *?thesis*

unfolding *embedding_map_def* *topspace_euclidean_subtopology*

by (*metis* f *homeomorphic_map_closedness_eq* *homeomorphism_injective_closed_map* *that*)

qed

qed (*metis* *homeomorphism_def* *inj_onI*)

lemma *injective_into_1d_imp_open_map*:

fixes $f :: 'a::\text{topological_space} \Rightarrow \text{real}$

assumes *continuous_on* S f *path_connected* S *inj_on* f S *openin* (*subtopology* *euclidean* S) T

shows *openin* (*subtopology* *euclidean* ($f \text{ ' } S$)) ($f \text{ ' } T$)

using *assms* *homeomorphism_imp_open_map* *injective_into_1d_eq_homeomorphism*

by *blast*

lemma *homeomorphism_into_1d*:

fixes $f :: 'a::\text{topological_space} \Rightarrow \text{real}$

assumes *path_connected* S *continuous_on* S f $f \text{ ' } S = T$ *inj_on* f S

shows $\exists g. \text{homeomorphism } S T f g$

using *assms* *injective_into_1d_eq_homeomorphism* **by** *blast*

6.1.28 Rectangular paths

definition *rectpath* **where**

rectpath $a1$ $a3 = (\text{let } a2 = \text{Complex } (\text{Re } a3) (\text{Im } a1); a4 = \text{Complex } (\text{Re } a1) (\text{Im } a3)$

in *linepath* $a1$ $a2$ +++ *linepath* $a2$ $a3$ +++ *linepath* $a3$ $a4$ +++ *linepath* $a4$ $a1$)

lemma *path_rectpath* [*simp*, *intro*]: *path* (*rectpath* a b)

by (*simp* *add*: *Let_def* *rectpath_def*)

lemma *pathstart_rectpath* [*simp*]: *pathstart* (*rectpath* $a1$ $a3$) = $a1$

by (*simp* *add*: *rectpath_def* *Let_def*)

lemma *pathfinish_rectpath* [*simp*]: $\text{pathfinish} (\text{rectpath } a1 \ a3) = a1$
by (*simp add: rectpath_def Let_def*)

lemma *simple_path_rectpath* [*simp, intro*]:
assumes $\text{Re } a1 \neq \text{Re } a3 \ \text{Im } a1 \neq \text{Im } a3$
shows *simple_path* (*rectpath* *a1 a3*)
unfolding *rectpath_def Let_def* **using** *assms*
by (*intro simple_path_join_loop arc_join arc_linepath*)
(auto simp: complex_eq_iff path_image_join closed_segment_same_Re closed_segment_same_Im)

lemma *path_image_rectpath*:
assumes $\text{Re } a1 \leq \text{Re } a3 \ \text{Im } a1 \leq \text{Im } a3$
shows $\text{path_image} (\text{rectpath } a1 \ a3) =$
 $\{z. \text{Re } z \in \{\text{Re } a1, \text{Re } a3\} \wedge \text{Im } z \in \{\text{Im } a1.. \text{Im } a3\}\} \cup$
 $\{z. \text{Im } z \in \{\text{Im } a1, \text{Im } a3\} \wedge \text{Re } z \in \{\text{Re } a1.. \text{Re } a3\}\}$ (**is** *?lhs = ?rhs*)
proof –
define *a2 a4* **where** $a2 = \text{Complex} (\text{Re } a3) (\text{Im } a1)$ **and** $a4 = \text{Complex} (\text{Re}$
 $a1) (\text{Im } a3)$
have *?lhs* = $\text{closed_segment } a1 \ a2 \cup \text{closed_segment } a2 \ a3 \cup$
 $\text{closed_segment } a4 \ a3 \cup \text{closed_segment } a1 \ a4$
by (*simp_all add: rectpath_def Let_def path_image_join closed_segment_commute*
a2_def a4_def Un_assoc)
also have $\dots = ?rhs$ **using** *assms*
by (*auto simp: rectpath_def Let_def path_image_join a2_def a4_def*
closed_segment_same_Re closed_segment_same_Im closed_segment_eq_real_ivl)
finally show *?thesis* .

qed

lemma *path_image_rectpath_subset_cbox*:
assumes $\text{Re } a \leq \text{Re } b \ \text{Im } a \leq \text{Im } b$
shows $\text{path_image} (\text{rectpath } a \ b) \subseteq \text{cbox } a \ b$
using *assms* **by** (*auto simp: path_image_rectpath_in_cbox_complex_iff*)

lemma *path_image_rectpath_inter_box*:
assumes $\text{Re } a \leq \text{Re } b \ \text{Im } a \leq \text{Im } b$
shows $\text{path_image} (\text{rectpath } a \ b) \cap \text{box } a \ b = \{\}$
using *assms* **by** (*auto simp: path_image_rectpath_in_box_complex_iff*)

lemma *path_image_rectpath_cbox_minus_box*:
assumes $\text{Re } a \leq \text{Re } b \ \text{Im } a \leq \text{Im } b$
shows $\text{path_image} (\text{rectpath } a \ b) = \text{cbox } a \ b - \text{box } a \ b$
using *assms* **by** (*auto simp: path_image_rectpath_in_cbox_complex_iff in_box_complex_iff*)

end

6.2 Neighbourhood bases and Locally path-connected spaces

```

theory Locally
  imports
    Path_Connected Function_Topology Sum_Topology
begin

```

6.2.1 Neighbourhood Bases

Useful for "local" properties of various kinds

```

definition neighbourhood_base_at where
  neighbourhood_base_at  $x P X \equiv$ 
     $\forall W. \text{openin } X W \wedge x \in W$ 
     $\longrightarrow (\exists U V. \text{openin } X U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W)$ 

```

```

definition neighbourhood_base_of where
  neighbourhood_base_of  $P X \equiv$ 
     $\forall x \in \text{topspace } X. \text{neighbourhood_base_at } x P X$ 

```

```

lemma neighbourhood_base_of:
  neighbourhood_base_of  $P X \longleftrightarrow$ 
     $(\forall W x. \text{openin } X W \wedge x \in W$ 
       $\longrightarrow (\exists U V. \text{openin } X U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$ 
unfolding neighbourhood_base_at_def neighbourhood_base_of_def
using openin_subset by blast

```

```

lemma neighbourhood_base_at_mono:
   $\llbracket \text{neighbourhood_base_at } x P X; \bigwedge S. \llbracket P S; x \in S \rrbracket \Longrightarrow Q S \rrbracket \Longrightarrow \text{neighbourhood_base_at } x Q X$ 
unfolding neighbourhood_base_at_def
by (meson subset_eq)

```

```

lemma neighbourhood_base_of_mono:
   $\llbracket \text{neighbourhood_base_of } P X; \bigwedge S. P S \Longrightarrow Q S \rrbracket \Longrightarrow \text{neighbourhood_base_of } Q X$ 
unfolding neighbourhood_base_of_def
using neighbourhood_base_at_mono by force

```

```

lemma open_neighbourhood_base_at:
   $(\bigwedge S. \llbracket P S; x \in S \rrbracket \Longrightarrow \text{openin } X S)$ 
     $\Longrightarrow \text{neighbourhood_base_at } x P X \longleftrightarrow (\forall W. \text{openin } X W \wedge x \in W \longrightarrow$ 
       $(\exists U. P U \wedge x \in U \wedge U \subseteq W))$ 
unfolding neighbourhood_base_at_def
by (meson subsetD)

```

```

lemma open_neighbourhood_base_of:
   $(\forall S. P S \longrightarrow \text{openin } X S)$ 

```

$\implies \text{neighbourhood_base_of } P X \longleftrightarrow (\forall W x. \text{openin } X W \wedge x \in W \longrightarrow (\exists U. P U \wedge x \in U \wedge U \subseteq W))$

by (smt (verit) neighbourhood_base_of_subsetD)

lemma neighbourhood_base_of_open_subset:

$\llbracket \text{neighbourhood_base_of } P X; \text{openin } X S \rrbracket$

$\implies \text{neighbourhood_base_of } P (\text{subtopology } X S)$

by (smt (verit, ccfv_SIG) neighbourhood_base_of_openin_open_subtopology_subset_trans)

lemma neighbourhood_base_of_topology_base:

$\text{openin } X = \text{arbitrary_union_of } (\lambda W. W \in \mathcal{B})$

$\implies \text{neighbourhood_base_of } P X \longleftrightarrow$

$(\forall W x. W \in \mathcal{B} \wedge x \in W \longrightarrow (\exists U V. \text{openin } X U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$

by (smt (verit, del_insts) neighbourhood_base_of_openin_topology_base_unique_subset_trans)

lemma neighbourhood_base_at_unlocalized:

assumes $\bigwedge S T. \llbracket P S; \text{openin } X T; x \in T; T \subseteq S \rrbracket \implies P T$

shows neighbourhood_base_at $x P X$

$\longleftrightarrow (x \in \text{topspace } X \longrightarrow (\exists U V. \text{openin } X U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq \text{topspace } X))$

(is ?lhs = ?rhs)

proof

assume $R: ?rhs$ **show** ?lhs

unfolding neighbourhood_base_at_def

proof clarify

fix W

assume $\text{openin } X W x \in W$

then have $x \in \text{topspace } X$

using openin_subset **by** blast

with R **obtain** $U V$ **where** $\text{openin } X U P V x \in U U \subseteq V V \subseteq \text{topspace } X$

by blast

then show $\exists U V. \text{openin } X U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W$

by (metis IntI <openin X W> <x ∈ W> assms inf_le1 inf_le2 openin_Int)

qed

qed (simp add: neighbourhood_base_at_def)

lemma neighbourhood_base_at_with_subset:

$\llbracket \text{openin } X U; x \in U \rrbracket$

$\implies (\text{neighbourhood_base_at } x P X \longleftrightarrow \text{neighbourhood_base_at } x (\lambda T. T \subseteq U \wedge P T) X)$

unfolding neighbourhood_base_at_def **by** (metis IntI Int_subset_iff openin_Int)

lemma neighbourhood_base_of_with_subset:

$\text{neighbourhood_base_of } P X \longleftrightarrow \text{neighbourhood_base_of } (\lambda t. t \subseteq \text{topspace } X \wedge P t) X$

using neighbourhood_base_at_with_subset

by (fastforce simp add: neighbourhood_base_of_def)

6.2.2 Locally path-connected spaces

definition *weakly_locally_path_connected_at*

where *weakly_locally_path_connected_at* $x X \equiv$ neighbourhood_base_at x (path_connectedin X) X

definition *locally_path_connected_at*

where *locally_path_connected_at* $x X \equiv$ neighbourhood_base_at x (λU . openin $X U \wedge$ path_connectedin $X U$) X

definition *locally_path_connected_space*

where *locally_path_connected_space* $X \equiv$ neighbourhood_base_of (path_connectedin X) X

lemma *locally_path_connected_space_alt:*

locally_path_connected_space $X \longleftrightarrow$ neighbourhood_base_of (λU . openin $X U \wedge$ path_connectedin $X U$) X

(is ?P = ?Q)

and *locally_path_connected_space_eq_open_path_component_of:*

locally_path_connected_space $X \longleftrightarrow$

($\forall U x$. openin $X U \wedge x \in U \longrightarrow$ openin X (Collect (path_component_of (subtopology $X U$) x)))

(is ?P = ?R)

proof –

have ?P if ?Q

using *locally_path_connected_space_def* neighbourhood_base_of_mono that by auto

moreover have ?R if P: ?P

proof –

show ?thesis

proof clarify

fix $U y$

assume openin $X U y \in U$

have $\exists T$. openin $X T \wedge x \in T \wedge T \subseteq$ Collect (path_component_of (subtopology $X U$) y)

if path_component_of (subtopology $X U$) $y x$ **for** x

proof –

have $x \in U$

using path_component_of_equiv that topspace_subtopology **by** fastforce

then have $\exists Ua$. openin $X Ua \wedge (\exists V$. path_connectedin $X V \wedge x \in Ua \wedge Ua \subseteq V \wedge V \subseteq U)$

using P

by (simp add: \langle openin $X U \rangle$ locally_path_connected_space_def neighbourhood_base_of)

then show ?thesis

by (metis dual_order.trans path_component_of_equiv path_component_of_maximal

```

path_connectedin_subtopology_subset_iff that)
  qed
  then show openin X (Collect (path_component_of (subtopology X U) y))
    using openin_subopen by force
  qed
  qed
  moreover have ?Q if ?R
    by (smt (verit) mem_Collect_eq open_neighbourhood_base_of openin_subset
      path_component_of_refl
      path_connectedin_path_component_of path_connectedin_subtopology that
      topspace_subtopology_subset)
  ultimately show ?P = ?Q ?P = ?R
    by blast+
  qed

```

lemma *locally_path_connected_space:*

```

locally_path_connected_space X
 $\longleftrightarrow (\forall V x. \text{openin } X V \wedge x \in V \longrightarrow (\exists U. \text{openin } X U \wedge \text{path\_connectedin } X U \wedge x \in U \wedge U \subseteq V))$ 
  by (simp add: locally_path_connected_space_alt open_neighbourhood_base_of)

```

lemma *locally_path_connected_space_open_path_components:*

```

locally_path_connected_space X  $\longleftrightarrow$ 
  ( $\forall U C. \text{openin } X U \wedge C \in \text{path\_components\_of}(\text{subtopology } X U) \longrightarrow \text{openin } X C$ )
  apply (simp add: locally_path_connected_space_eq_open_path_component_of
    path_components_of_def)
  by (smt (verit, ccfv_threshold) Int_iff image_iff openin_subset subset_iff)

```

lemma *openin_path_component_of_locally_path_connected_space:*

```

locally_path_connected_space X  $\implies \text{openin } X (\text{Collect } (\text{path\_component\_of } X x))$ 
  using locally_path_connected_space_eq_open_path_component_of openin_subopen
  path_component_of_eq_empty
  by fastforce

```

lemma *openin_path_components_of_locally_path_connected_space:*

```

 $\llbracket \text{locally\_path\_connected\_space } X; C \in \text{path\_components\_of } X \rrbracket \implies \text{openin } X C$ 
  using locally_path_connected_space_open_path_components by force

```

lemma *closedin_path_components_of_locally_path_connected_space:*

```

 $\llbracket \text{locally\_path\_connected\_space } X; C \in \text{path\_components\_of } X \rrbracket \implies \text{closedin } X C$ 
  unfolding closedin_def
  by (metis Diff_iff complement_path_components_of_Union openin_clauses(3)
    openin_closedin_eq
    openin_path_components_of_locally_path_connected_space)

```

lemma *closedin_path_component_of_locally_path_connected_space*:
assumes *locally_path_connected_space X*
shows *closedin X (Collect (path_component_of X x))*
proof (*cases x ∈ topspace X*)
case *True*
then show *?thesis*
by (*simp add: assms closedin_path_components_of_locally_path_connected_space path_component_in_path_components_of*)
next
case *False*
then show *?thesis*
by (*metis closedin_empty_path_component_of_eq_empty*)
qed

lemma *weakly_locally_path_connected_at*:
weakly_locally_path_connected_at x X \longleftrightarrow
 $(\forall V. \text{openin } X \ V \wedge x \in V$
 $\longrightarrow (\exists U. \text{openin } X \ U \wedge x \in U \wedge U \subseteq V \wedge$
 $(\forall y \in U. \exists C. \text{path_connectedin } X \ C \wedge C \subseteq V \wedge x \in C \wedge y \in C)))$
(is ?lhs = ?rhs)

proof
assume *?lhs* **then show** *?rhs*
by (*smt (verit) neighbourhood_base_at_def subset_iff weakly_locally_path_connected_at_def*)
next
have $*$: $\exists V. \text{path_connectedin } X \ V \wedge U \subseteq V \wedge V \subseteq W$
if $(\forall y \in U. \exists C. \text{path_connectedin } X \ C \wedge C \subseteq W \wedge x \in C \wedge y \in C)$
for $W \ U$
proof (*intro exI conjI*)
let $?V = (\text{Collect } (\text{path_component_of } (\text{subtopology } X \ W) \ x))$
show $\text{path_connectedin } X \ (\text{Collect } (\text{path_component_of } (\text{subtopology } X \ W) \ x))$
by (*meson path_connectedin_path_component_of_path_connectedin_subtopology*)
show $U \subseteq ?V$
by (*auto simp: path_component_of_path_connectedin_subtopology that*)
show $?V \subseteq W$
by (*meson path_connectedin_path_component_of_path_connectedin_subtopology*)
qed
assume *?rhs*
then show *?lhs*
unfolding *weakly_locally_path_connected_at_def neighbourhood_base_at_def*
by (*metis **)
qed

lemma *locally_path_connected_space_im_kleinen*:
locally_path_connected_space X \longleftrightarrow
 $(\forall V \ x. \text{openin } X \ V \wedge x \in V$
 $\longrightarrow (\exists U. \text{openin } X \ U \wedge$
 $x \in U \wedge U \subseteq V \wedge$
 $(\forall y \in U. \exists C. \text{path_connectedin } X \ C \wedge$

```

      ( $C \subseteq V \wedge x \in C \wedge y \in C$ ))
    (is ?lhs = ?rhs)
  proof
    show ?lhs  $\implies$  ?rhs
      by (metis locally_path_connected_space)
    assume ?rhs
    then show ?lhs
      unfolding locally_path_connected_space_def neighbourhood_base_of
      by (metis neighbourhood_base_at_def weakly_locally_path_connected_at weakly_locally_path_connected_at_def)
  qed

lemma locally_path_connected_space_open_subset:
  [[locally_path_connected_space X; openin X S]]
   $\implies$  locally_path_connected_space (subtopology X S)
  by (smt (verit, best) locally_path_connected_space openin_open_subtopology
    path_connectedin_subtopology subset_trans)

lemma locally_path_connected_space_quotient_map_image:
  assumes f: quotient_map X Y f and X: locally_path_connected_space X
  shows locally_path_connected_space Y
  unfolding locally_path_connected_space_open_path_components
  proof clarify
    fix V C
    assume V: openin Y V and c:  $C \in$  path_components_of (subtopology Y V)
    then have sub:  $C \subseteq$  topspace Y
      using path_connectedin_path_components_of path_connectedin_subset_topspace
      path_connectedin_subtopology by blast
    have openin X {x  $\in$  topspace X. f x  $\in$  C}
      proof (subst openin_subopen, clarify)
        fix x
        assume x: x  $\in$  topspace X and f x  $\in$  C
        let ?T = Collect (path_component_of (subtopology X {z  $\in$  topspace X. f z  $\in$ 
          V}) x)
        show  $\exists T$ . openin X T  $\wedge$  x  $\in$  T  $\wedge$  T  $\subseteq$  {x  $\in$  topspace X. f x  $\in$  C}
          proof (intro exI conjI)
            have *:  $\exists U$ . openin X U  $\wedge$  ?T  $\in$  path_components_of (subtopology X U)
              proof (intro exI conjI)
                show openin X ({z  $\in$  topspace X. f z  $\in$  V})
                  using V f openin_subset quotient_map_def by fastforce
                have x  $\in$  topspace (subtopology X {z  $\in$  topspace X. f z  $\in$  V})
                  using ⟨f x  $\in$  C⟩ c path_components_of_subset x by force
                then show Collect (path_component_of (subtopology X {z  $\in$  topspace X.
                  f z  $\in$  V}) x)
                   $\in$  path_components_of (subtopology X {z  $\in$  topspace X. f z  $\in$  V})
                  by (meson path_component_in_path_components_of)
              qed
            with X show openin X ?T
              using locally_path_connected_space_open_path_components by blast
            show x: x  $\in$  ?T
  end

```

```

using * nonempty_path_components_of_path_component_of_eq_path_component_of_eq_empty
by fastforce
  have  $f' \ ?T \subseteq C$ 
  proof (rule path_components_of_maximal)
    show  $C \in \text{path\_components\_of} \ (\text{subtopology } Y \ V)$ 
      by (simp add: c)
    show  $\text{path\_connectedin} \ (\text{subtopology } Y \ V) \ (f' \ ?T)$ 
      proof –
        have  $\text{quotient\_map} \ (\text{subtopology } X \ \{a \in \text{topspace } X. \ f \ a \in V\}) \ (\text{subtopology } Y \ V) \ f$ 
          by (simp add: V f quotient_map_restriction)
        then show ?thesis
          by (meson path_connectedin_continuous_map_image_path_connectedin_path_component_of_quotient_imp_continuous_map)
        qed
    show  $\neg \text{disjnt } C \ (f' \ ?T)$ 
      by (metis (no_types, lifting) <f x ∈ C> x disjnt_iff_image_eqI)
    qed
  then show  $?T \subseteq \{x \in \text{topspace } X. \ f \ x \in C\}$ 
    by (force simp: path_component_of_equiv)
  qed
qed
then show openin Y C
  using f sub by (simp add: quotient_map_def)
qed

```

```

lemma homeomorphic_locally_path_connected_space:
  assumes  $X \ \text{homeomorphic\_space } Y$ 
  shows  $\text{locally\_path\_connected\_space } X \iff \text{locally\_path\_connected\_space } Y$ 
  using assms
  unfolding homeomorphic_space_def homeomorphic_map_def homeomorphic_maps_map
  by (metis locally_path_connected_space_quotient_map_image)

```

```

lemma locally_path_connected_space_retraction_map_image:
   $\llbracket \text{retraction\_map } X \ Y \ r; \text{locally\_path\_connected\_space } X \rrbracket$ 
   $\implies \text{locally\_path\_connected\_space } Y$ 
  using Abstract_Topology.retraction_imp_quotient_map locally_path_connected_space_quotient_map
by blast

```

```

lemma locally_path_connected_space_euclideanreal:  $\text{locally\_path\_connected\_space euclideanreal}$ 
  unfolding locally_path_connected_space_def neighbourhood_base_of
  proof clarsimp
  fix  $W$  and  $x :: \text{real}$ 
  assume  $\text{open } W \ x \in W$ 
  then obtain  $e$  where  $e > 0$  and  $e: \bigwedge x'. |x' - x| < e \implies x' \in W$ 
    by (auto simp: open_real)
  then show  $\exists U. \text{open } U \wedge (\exists V. \text{path\_connected } V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W)$ 

```


by (force intro!: convex_imp_path_connected exI [where $x = \{x - e <..< x + e\}$])
qed

lemma *locally_path_connected_space_discrete_topology*:
 locally_path_connected_space (discrete_topology U)
 using locally_path_connected_space_open_path_components by fastforce

lemma *path_component_eq_connected_component_of*:
 assumes locally_path_connected_space X
 shows path_component_of_set X x = connected_component_of_set X x
proof (cases $x \in \text{topspace } X$)
 case True
 have path_component_of_set X x \subseteq connected_component_of_set X x
 by (simp add: path_component_subset_connected_component_of)
 moreover have closedin X (path_component_of_set X x)
 by (simp add: assms closedin_path_component_of_locally_path_connected_space)
 moreover have openin X (path_component_of_set X x)
 by (simp add: assms openin_path_component_of_locally_path_connected_space)
 moreover have path_component_of_set X x \neq {}
 by (meson True path_component_of_eq_empty)
 ultimately show ?thesis
 using connectedin_connected_component_of [of X x] unfolding connecte-
 din_def
 by (metis closedin_subset_topspace connected_space_clopen_in
 subset_openin_subtopology_topspace_subtopology_subset)
 next
 case False
 then show ?thesis
 using connected_component_of_eq_empty path_component_of_eq_empty by
 fastforce
 qed

lemma *path_components_eq_connected_components_of*:
 locally_path_connected_space X \implies (path_components_of X = connected_components_of
 X)
 by (simp add: path_components_of_def connected_components_of_def image_def
 path_component_eq_connected_component_of)

lemma *path_connected_eq_connected_space*:
 locally_path_connected_space X
 \implies path_connected_space X \iff connected_space X
 by (metis connected_components_of_subset_sing path_components_eq_connected_components_of
 path_components_of_subset_singleton)

lemma *locally_path_connected_space_product_topology*:
 locally_path_connected_space (product_topology X I) \iff
 (product_topology X I) = trivial_topology \vee
 finite {i. i \in I \wedge \sim path_connected_space(X i)} \wedge
 (\forall i \in I. locally_path_connected_space(X i))

```

    (is ?lhs  $\longleftrightarrow$  ?empty  $\vee$  ?rhs)
proof (cases ?empty)
  case True
    then show ?thesis
    by (simp add: locally_path_connected_space_def neighbourhood_base_of openin_closedin_eq)
  next
  case False
    then obtain z where z: z  $\in$  ( $\prod_E i \in I. \text{topspace } (X i)$ )
    using discrete_topology_unique_derived_set by (fastforce iff: null_topspace_iff_trivial)
    have ?rhs if L: ?lhs
    proof -
      obtain U C where U: openin (product_topology X I) U
        and V: path_connectedin (product_topology X I) C
        and z  $\in$  U U  $\subseteq$  C and Csub: C  $\subseteq$  ( $\prod_E i \in I. \text{topspace } (X i)$ )
      by (metis L locally_path_connected_space openin_topspace topspace_product_topology
z)
      then obtain V where finV: finite {i  $\in$  I. V i  $\neq$  topspace (X i)}
        and XV:  $\bigwedge i. i \in I \implies \text{openin } (X i) (V i)$  and z  $\in$  PiE I V and subU: PiE
I V  $\subseteq$  U
      by (force simp: openin_product_topology_alt)
      show ?thesis
      proof (intro conjI ballI)
        have path_connected_space (X i) if i  $\in$  I V i = topspace (X i) for i
        proof -
          have pc: path_connectedin (X i) (( $\lambda x. x i$ ) ' C)
          by (metis V continuous_map_product_projection path_connectedin_continuous_map_image
that(1))
          moreover have (( $\lambda x. x i$ ) ' C) = topspace (X i)
          proof
            show ( $\lambda x. x i$ ) ' C  $\subseteq$  topspace (X i)
            by (simp add: pc path_connectedin_subset_topspace)
            have V i  $\subseteq$  ( $\lambda x. x i$ ) ' ( $\prod_E i \in I. V i$ )
            by (metis  $\langle z \in \text{PiE } I V \rangle$  empty_iff_image_projection_PiE order_refl
that(1))
            also have ...  $\subseteq$  ( $\lambda x. x i$ ) ' U
            using subU by blast
            finally show topspace (X i)  $\subseteq$  ( $\lambda x. x i$ ) ' C
            using  $\langle U \subseteq C \rangle$  that by blast
          qed
          ultimately show ?thesis
          by (simp add: path_connectedin_topspace)
        qed
      then have {i  $\in$  I.  $\neg$  path_connected_space (X i)}  $\subseteq$  {i  $\in$  I. V i  $\neq$  topspace
(X i)}
      by blast
      with finV show finite {i  $\in$  I.  $\neg$  path_connected_space (X i)}
      using finite_subset by blast
    next
    show locally_path_connected_space (X i) if i  $\in$  I for i

```

```

    by (meson False L locally_path_connected_space_quotient_map_image
quotient_map_product_projection that)
  qed
  qed
  moreover have ?lhs if R: ?rhs
  proof (clarsimp simp add: locally_path_connected_space_def neighbourhood_base_of)
    fix F z
    assume openin (product_topology X I) F and z ∈ F
    then obtain W where finW: finite {i ∈ I. W i ≠ topspace (X i)}
      and opeW:  $\bigwedge i. i \in I \implies \text{openin } (X i) (W i)$  and  $z \in \text{Pi}_E I W \text{ Pi}_E I$ 
    W ⊆ F
    by (auto simp: openin_product_topology_alt)
    have  $\forall i \in I. \exists U C. \text{openin } (X i) U \wedge \text{path\_connectedin } (X i) C \wedge z i \in U \wedge$ 
      U ⊆ C ∧ C ⊆ W i ∧
      (W i = topspace (X i) ∧
      path_connected_space (X i)  $\longrightarrow$  U = topspace (X i) ∧ C =
      topspace (X i))
      (is  $\forall i \in I. ?\Phi i$ )
    proof
      fix i assume i ∈ I
      have locally_path_connected_space (X i)
      by (simp add: R <i ∈ I>)
      moreover have *:  $\text{openin } (X i) (W i) \ z i \in W i$ 
      using <z ∈ Pi_E I W> opeW <i ∈ I> by auto
      ultimately obtain U C where UC:  $\text{openin } (X i) U \text{ path\_connectedin } (X i)$ 
        C z i ∈ U U ⊆ C C ⊆ W i
      using <i ∈ I> by (force simp: locally_path_connected_space_def neighbourhood_base_of)
      show ?Φ i
      by (metis UC * openin_subset path_connectedin_topspace)
    qed
    then obtain U C where
      *:  $\bigwedge i. i \in I \implies \text{openin } (X i) (U i) \wedge \text{path\_connectedin } (X i) (C i) \wedge z i \in$ 
      (U i) ∧ (U i) ⊆ (C i) ∧ (C i) ⊆ W i ∧
      (W i = topspace (X i) ∧ path_connected_space (X i)
       $\longrightarrow$  (U i) = topspace (X i) ∧ (C i) = topspace (X i))
    by metis
    let ?A = {i ∈ I.  $\neg \text{path\_connected\_space } (X i)$ } ∪ {i ∈ I. W i ≠ topspace (X i)}
    have {i ∈ I. U i ≠ topspace (X i)} ⊆ ?A
    by (clarsimp simp add: *)
    moreover have finite ?A
    by (simp add: that finW)
    ultimately have finite {i ∈ I. U i ≠ topspace (X i)}
    using finite_subset by auto
    with * have openin (product_topology X I) (Pi_E I U)
    by (simp add: openin_PiE_gen)
    then show  $\exists U. \text{openin } (product\_topology X I) U \wedge$ 
      ( $\exists V. \text{path\_connectedin } (product\_topology X I) V \wedge z \in U \wedge U \subseteq V$ )

```

1064

```

 $\wedge V \subseteq F$ )
  by (metis (no_types, opaque_lifting) subsetI  $\langle z \in Pi_E I W \rangle \langle Pi_E I W \subseteq F \rangle$ )
* path_connected_in_PiE
  PiE_iff PiE_mono order.trans)
qed
ultimately show ?thesis
  using False by blast
qed
```

```

lemma locally_path_connected_is_realinterval:
  assumes is_interval S
  shows locally_path_connected_space (subtopology euclideanreal S)
  unfolding locally_path_connected_space_def
proof (clarsimp simp add: neighbourhood_base_of_open_in_subtopology_alt)
  fix a U
  assume a  $\in S$  and a  $\in U$  and open U
  then obtain r where r > 0 and r: ball a r  $\subseteq U$ 
    by (metis open_contains_ball_eq)
  show  $\exists W. \text{open } W \wedge (\exists V. \text{path\_connected\_in } (top\_of\_set S) V \wedge a \in W \wedge S$ 
 $\cap W \subseteq V \wedge V \subseteq S \wedge V \subseteq U)$ 
  proof (intro exI conjI)
    show path_connected_in (top_of_set S) (S  $\cap$  ball a r)
    by (simp add: assms is_interval_Int is_interval_ball_real is_interval_path_connected
path_connected_in_subtopology)
    show a  $\in$  ball a r
    by (simp add:  $\langle 0 < r \rangle$ )
  qed (use  $\langle 0 < r \rangle$  r in auto)
qed
```

```

lemma locally_path_connected_real_interval:
  locally_path_connected_space (subtopology euclideanreal {a..b})
  locally_path_connected_space (subtopology euclideanreal {a<.. $<b$ })
  using locally_path_connected_is_realinterval
  by (auto simp add: is_interval_1)
```

```

lemma locally_path_connected_space_prod_topology:
  locally_path_connected_space (prod_topology X Y)  $\longleftrightarrow$ 
  (prod_topology X Y) = trivial_topology  $\vee$ 
  locally_path_connected_space X  $\wedge$  locally_path_connected_space Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    using locally_path_connected_space_discrete_topology by force
next
  case False
  then have ne: X  $\neq$  trivial_topology Y  $\neq$  trivial_topology
    by simp_all
  show ?thesis
  proof
```

```

  assume ?lhs then show ?rhs
    by (meson locally_path_connected_space_quotient_map_image ne(1) ne(2)
quotient_map_fst quotient_map_snd)
  next
    assume ?rhs
  with False have X: locally_path_connected_space X and Y: locally_path_connected_space
Y
    by auto
  show ?lhs
    unfolding locally_path_connected_space_def neighbourhood_base_of
  proof clarify
    fix UV x y
    assume UV: openin (prod_topology X Y) UV and (x,y) ∈ UV
    obtain A B where W12: openin X A ∧ openin Y B ∧ x ∈ A ∧ y ∈ B ∧ A
× B ⊆ UV
    using X Y by (metis UV ⟨(x,y) ∈ UV⟩ openin_prod_topology_alt)
    then obtain C D K L
    where openin X C path_connectedin X K x ∈ C C ⊆ K K ⊆ A
      openin Y D path_connectedin Y L y ∈ D D ⊆ L L ⊆ B
    by (metis X Y locally_path_connected_space)
    with W12 ⟨openin X C⟩ ⟨openin Y D⟩
  show ∃ U V. openin (prod_topology X Y) U ∧ path_connectedin (prod_topology
X Y) V ∧ (x, y) ∈ U ∧ U ⊆ V ∧ V ⊆ UV
    apply (rule_tac x=C × D in exI)
    apply (rule_tac x=K × L in exI)
    apply (fastforce simp: openin_prod_Times_iff path_connectedin_Times)
  done
  qed
  qed
  qed

```

lemma *locally_path_connected_space_sum_topology:*

locally_path_connected_space(sum_topology X I) ↔
(∀ i ∈ I. locally_path_connected_space (X i)) (is ?lhs=?rhs)

proof

assume ?lhs then show ?rhs

by (smt (verit) homeomorphic_locally_path_connected_space locally_path_connected_space_open_subset
topological_property_of_sum_component)

next

assume R: ?rhs

show ?lhs

proof (clarsimp simp add: locally_path_connected_space_def neighbourhood_base_of
forall_openin_sum_topology imp_conjL)

fix W i x

assume ope: ∀ i ∈ I. openin (X i) (W i)

and i ∈ I and x ∈ W i

then obtain U V where U: openin (X i) U and V: path_connectedin (X i)

V

and x ∈ U U ⊆ V V ⊆ W i

```

    by (metis R ⟨i ∈ I⟩ ⟨x ∈ W i⟩ locally_path_connected_space)
  show ∃ U. openin (sum_topology X I) U ∧ (∃ V. path_connectedin (sum_topology
X I) V ∧ (i, x) ∈ U ∧ U ⊆ V ∧ V ⊆ Sigma I W)
  proof (intro exI conjI)
    show openin (sum_topology X I) (Pair i ‘ U)
      by (meson U ⟨i ∈ I⟩ open_map_component_injection open_map_def)
    show path_connectedin (sum_topology X I) (Pair i ‘ V)
      by (meson V ⟨i ∈ I⟩ continuous_map_component_injection path_connectedin_continuous_map_in
show Pair i ‘ V ⊆ Sigma I W
      using ⟨V ⊆ W i⟩ ⟨i ∈ I⟩ by force
    qed (use ⟨x ∈ U⟩ ⟨U ⊆ V⟩ in auto)
  qed
qed

```

6.2.3 Locally connected spaces

definition *weakly_locally_connected_at*
where *weakly_locally_connected_at* $x X \equiv$ *neighbourhood_base_at* x (*connectedin* X) X

definition *locally_connected_at*
where *locally_connected_at* $x X \equiv$
neighbourhood_base_at x ($\lambda U. \text{openin } X U \wedge \text{connectedin } X U$) X

definition *locally_connected_space*
where *locally_connected_space* $X \equiv$ *neighbourhood_base_of* (*connectedin* X) X

lemma *locally_connected_A*: ($\forall U x. \text{openin } X U \wedge x \in U \longrightarrow \text{openin } X$ (*connected_component_of_set* (*subtopology* $X U$) x))

\implies *neighbourhood_base_of* ($\lambda U. \text{openin } X U \wedge \text{connectedin } X U$) X

unfolding *neighbourhood_base_of*

by (*metis* (*full_types*) *connected_component_of_refl* *connectedin_connected_component_of* *connectedin_subtopology* *mem_Collect_eq* *openin_subset* *topspace_subtopology_subset*)

lemma *locally_connected_B*: *locally_connected_space* $X \implies$
($\forall U x. \text{openin } X U \wedge x \in U \longrightarrow \text{openin } X$ (*connected_component_of_set* (*subtopology* $X U$) x))

unfolding *locally_connected_space_def* *neighbourhood_base_of*

apply (*erule* *all_forward*)

apply *clarify*

apply (*subst* *openin_subopen*)

by (*smt* (*verit*, *ccfv_threshold*) *Ball_Collect* *connected_component_of_def* *connected_component_of_equiv* *connectedin_subtopology* *in_mono* *mem_Collect_eq*)

lemma *locally_connected_C*: *neighbourhood_base_of* ($\lambda U. \text{openin } X U \wedge \text{connectedin } X U$) $X \implies$ *locally_connected_space* X

using *locally_connected_space_def* *neighbourhood_base_of_mono* **by** *auto*

lemma *locally_connected_space_alt*:

$locally_connected_space\ X \longleftrightarrow neighbourhood_base_of\ (\lambda U. openin\ X\ U \wedge connectedin\ X\ U)\ X$
using *locally_connected_A locally_connected_B locally_connected_C* **by** *blast*

lemma *locally_connected_space_eq_open_connected_component_of*:

$locally_connected_space\ X \longleftrightarrow$
 $(\forall U\ x. openin\ X\ U \wedge x \in U$
 $\longrightarrow openin\ X\ (connected_component_of_set\ (subtopology\ X\ U)\ x))$
by (*meson locally_connected_A locally_connected_B locally_connected_C*)

lemma *locally_connected_space*:

$locally_connected_space\ X \longleftrightarrow$
 $(\forall V\ x. openin\ X\ V \wedge x \in V \longrightarrow (\exists U. openin\ X\ U \wedge connectedin\ X\ U \wedge x \in U \wedge U \subseteq V))$
by (*simp add: locally_connected_space_alt open_neighbourhood_base_of*)

lemma *locally_path_connected_imp_locally_connected_space*:

$locally_path_connected_space\ X \implies locally_connected_space\ X$
by (*simp add: locally_connected_space_def locally_path_connected_space_def neighbourhood_base_of_mono_path_connectedin_imp_connectedin*)

lemma *locally_connected_space_open_connected_components*:

$locally_connected_space\ X \longleftrightarrow$
 $(\forall U\ C. openin\ X\ U \wedge C \in connected_components_of\ (subtopology\ X\ U) \longrightarrow openin\ X\ C)$
unfolding *connected_components_of_def locally_connected_space_eq_open_connected_component_of*
by (*smt (verit, best) image_iff openin_subset topspace_subtopology_subset*)

lemma *openin_connected_component_of_locally_connected_space*:

$locally_connected_space\ X \implies openin\ X\ (connected_component_of_set\ X\ x)$
by (*metis connected_component_of_eq_empty locally_connected_B openin_empty openin_topspace subtopology_topspace*)

lemma *openin_connected_components_of_locally_connected_space*:

$\llbracket locally_connected_space\ X; C \in connected_components_of\ X \rrbracket \implies openin\ X\ C$
by (*metis locally_connected_space_open_connected_components openin_topspace subtopology_topspace*)

lemma *weakly_locally_connected_at*:

$weakly_locally_connected_at\ x\ X \longleftrightarrow$
 $(\forall V. openin\ X\ V \wedge x \in V$
 $\longrightarrow (\exists U. openin\ X\ U \wedge x \in U \wedge U \subseteq V \wedge$
 $(\forall y \in U. \exists C. connectedin\ X\ C \wedge C \subseteq V \wedge x \in C \wedge y \in C)))$ (**is**
?lhs=?rhs)

proof

assume *?lhs* **then show** *?rhs*

```

    unfolding neighbourhood_base_at_def weakly_locally_connected_at_def
    by (meson subsetD subset_trans)
next
assume R: ?rhs
show ?lhs
    unfolding neighbourhood_base_at_def weakly_locally_connected_at_def
proof clarify
    fix V
    assume openin X V and x ∈ V
    then obtain U where openin X U x ∈ U U ⊆ V
        and U: ∀ y ∈ U. ∃ C. connectedin X C ∧ C ⊆ V ∧ x ∈ C ∧ y ∈ C
    using R by force
    show ∃ A B. openin X A ∧ connectedin X B ∧ x ∈ A ∧ A ⊆ B ∧ B ⊆ V
    proof (intro conjI exI)
        show connectedin X (connected_component_of_set (subtopology X V) x)
        by (meson connectedin_connected_component_of_connectedin_subtopology)
        show U ⊆ connected_component_of_set (subtopology X V) x
        using connected_component_of_maximal U
        by (simp add: connected_component_of_def connectedin_subtopology sub-
setI)
        show connected_component_of_set (subtopology X V) x ⊆ V
        using connected_component_of_subset_topspace by fastforce
    qed (auto simp: ⟨x ∈ U⟩ ⟨openin X U⟩)
qed
qed

```

lemma *locally_connected_space_iff_weak*:

locally_connected_space X \longleftrightarrow $(\forall x \in \text{topspace } X. \text{weakly_locally_connected_at } x X)$

by (simp add: locally_connected_space_def neighbourhood_base_of_def weakly_locally_connected_at

lemma *locally_connected_space_im_kleinen*:

locally_connected_space X \longleftrightarrow

$(\forall V x. \text{openin } X V \wedge x \in V$

$\longrightarrow (\exists U. \text{openin } X U \wedge x \in U \wedge U \subseteq V \wedge$

$(\forall y \in U. \exists C. \text{connectedin } X C \wedge C \subseteq V \wedge x \in C \wedge y \in C)))$

unfolding *locally_connected_space_iff_weak* *weakly_locally_connected_at*

using *openin_subset subsetD* by fastforce

lemma *locally_connected_space_open_subset*:

$\llbracket \text{locally_connected_space } X; \text{openin } X S \rrbracket \implies \text{locally_connected_space } (\text{subtopology } X S)$

unfolding *locally_connected_space_def* *neighbourhood_base_of*

by (smt (verit) *connectedin_subtopology openin_open_subtopology subset_trans*)

lemma *locally_connected_space_quotient_map_image*:

assumes *X*: *locally_connected_space X* and *f*: *quotient_map X Y f*

shows *locally_connected_space Y*

unfolding *locally_connected_space_open_connected_components*


```

proof clarify
  fix V C
  assume openin Y V and C: C ∈ connected_components_of (subtopology Y V)
  then have C ⊆ topspace Y
    using connected_components_of_subset by force
  have ope1: openin X {a ∈ topspace X. f a ∈ V}
    using ⟨openin Y V⟩ f openin_continuous_map_preimage quotient_imp_continuous_map
by blast
  define Vf where Vf ≡ {z ∈ topspace X. f z ∈ V}
  have openin X {x ∈ topspace X. f x ∈ C}
  proof (clarsimp simp: openin_subopen [where S = {x ∈ topspace X. f x ∈ C}])
    fix x
    assume x ∈ topspace X and f x ∈ C
    show ∃ T. openin X T ∧ x ∈ T ∧ T ⊆ {x ∈ topspace X. f x ∈ C}
    proof (intro exI conjI)
      show openin X (connected_component_of_set (subtopology X Vf) x)
      by (metis Vf_def X connected_component_of_eq_empty locally_connected_B
ope1 openin_empty
      openin_subset topspace_subtopology_subset)
      show x_in_conn: x ∈ connected_component_of_set (subtopology X Vf) x
      using C Vf_def ⟨f x ∈ C⟩ ⟨x ∈ topspace X⟩ connected_component_of_refl
connected_components_of_subset by fastforce
      have connected_component_of_set (subtopology X Vf) x ⊆ topspace X ∩ Vf
      using connected_component_of_subset_topspace by fastforce
      moreover
      have f ‘ connected_component_of_set (subtopology X Vf) x ⊆ C
      proof (rule connected_components_of_maximal [where X = subtopology Y
V])
        show C ∈ connected_components_of (subtopology Y V)
        by (simp add: C)
        have §: quotient_map (subtopology X Vf) (subtopology Y V) f
        by (simp add: Vf_def ⟨openin Y V⟩ f quotient_map_restriction)
        then show connectedin (subtopology Y V) (f ‘ connected_component_of_set
(subtopology X Vf) x)
        by (metis connectedin_connected_component_of connectedin_continuous_map_image
quotient_imp_continuous_map)
        show ¬ disjoint C (f ‘ connected_component_of_set (subtopology X Vf) x)
        using ⟨f x ∈ C⟩ x_in_conn by (auto simp: disjoint_iff)
      qed
      ultimately
      show connected_component_of_set (subtopology X Vf) x ⊆ {x ∈ topspace
X. f x ∈ C}
      by blast
    qed
  qed
  then show openin Y C
    using ⟨C ⊆ topspace Y⟩ f quotient_map_def by fastforce
qed

```

lemma *locally_connected_space_retraction_map_image*:

[[*retraction_map* $X\ Y\ r$; *locally_connected_space* X]]

\implies *locally_connected_space* Y

using *locally_connected_space_quotient_map_image* *retraction_imp_quotient_map*
by *blast*

lemma *homeomorphic_locally_connected_space*:

X *homeomorphic_space* $Y \implies$ *locally_connected_space* $X \longleftrightarrow$ *locally_connected_space* Y

by (*meson* *homeomorphic_map_def* *homeomorphic_space* *homeomorphic_space_sym* *locally_connected_space_quotient_map_image*)

lemma *locally_connected_space_euclideanreal*: *locally_connected_space* *euclideanreal*

by (*simp* *add*: *locally_path_connected_imp_locally_connected_space* *locally_path_connected_space_euclideanreal*)

lemma *locally_connected_is_realinterval*:

is_interval $S \implies$ *locally_connected_space*(*subtopology* *euclideanreal* S)

by (*simp* *add*: *locally_path_connected_imp_locally_connected_space* *locally_path_connected_is_realinterval*)

lemma *locally_connected_real_interval*:

locally_connected_space (*subtopology* *euclideanreal*{ $a..b$ })

locally_connected_space (*subtopology* *euclideanreal*{ $a<.. $<b$$ })

using *connected_Icc* *is_interval_connected_1* *locally_connected_is_realinterval*
by *auto*

lemma *locally_connected_space_discrete_topology*:

locally_connected_space (*discrete_topology* U)

by (*simp* *add*: *locally_path_connected_imp_locally_connected_space* *locally_path_connected_space_discrete_topology*)

lemma *locally_path_connected_imp_locally_connected_at*:

locally_path_connected_at $x\ X \implies$ *locally_connected_at* $x\ X$

by (*simp* *add*: *locally_connected_at_def* *locally_path_connected_at_def* *neighbourhood_base_at_mono* *path_connectedin_imp_connectedin*)

lemma *weakly_locally_path_connected_imp_weakly_locally_connected_at*:

weakly_locally_path_connected_at $x\ X \implies$ *weakly_locally_connected_at* $x\ X$

by (*metis* *path_connectedin_imp_connectedin* *weakly_locally_connected_at* *weakly_locally_path_connected_at*)

lemma *interior_of_locally_connected_subspace_component*:

assumes X : *locally_connected_space* X

and C : $C \in$ *connected_components_of* (*subtopology* $X\ S$)

shows X *interior_of* $C = C \cap X$ *interior_of* S

proof –

obtain C_{sub} : $C \subseteq$ *topspace* $X\ C \subseteq S$

by (*meson* C *connectedin_connected_components_of* *connectedin_subset_topspace* *connectedin_subtopology*)

```

show ?thesis
proof
  show  $X \text{ interior\_of } C \subseteq C \cap X \text{ interior\_of } S$ 
    by (simp add: Csub interior_of_mono interior_of_subset)
  have eq:  $X \text{ interior\_of } S = \bigcup (\text{connected\_components\_of } (\text{subtopology } X (X \text{ interior\_of } S)))$ 
    by (metis Union_connected_components_of_interior_of_subset_topspace
topspace_subtopology_subset)
  moreover have  $C \cap D \subseteq X \text{ interior\_of } C$ 
    if  $D \in \text{connected\_components\_of } (\text{subtopology } X (X \text{ interior\_of } S))$  for  $D$ 
  proof (cases  $C \cap D = \{\}$ )
    case False
      have  $D \subseteq X \text{ interior\_of } C$ 
      proof (rule interior_of_maximal)
        have  $\text{connectedin } (\text{subtopology } X S) D$ 
        by (meson connectedin_connected_components_of_connectedin_subtopology
interior_of_subset_subset_trans that)
      then show  $D \subseteq C$ 
        by (meson C False connected_components_of_maximal_disjnt_def)
      show  $\text{openin } X D$ 
      using  $X \text{ locally\_connected\_space\_open\_connected\_components\_openin\_interior\_of}$ 
that by blast
    qed
  then show ?thesis
    by blast
  qed auto
  ultimately show  $C \cap X \text{ interior\_of } S \subseteq X \text{ interior\_of } C$ 
    by blast
qed
qed

```

```

lemma frontier_of_locally_connected_subspace_component:
assumes  $X: \text{locally\_connected\_space } X$  and  $\text{closedin } X S$ 
and  $C: C \in \text{connected\_components\_of } (\text{subtopology } X S)$ 
shows  $X \text{ frontier\_of } C = C \cap X \text{ frontier\_of } S$ 
proof -
  obtain Csub:  $C \subseteq \text{topspace } X C \subseteq S$ 
    by (meson C connectedin_connected_components_of_connectedin_subset_topspace
connectedin_subtopology)
  then have  $X \text{ closure\_of } C - X \text{ interior\_of } C = C \cap X \text{ closure\_of } S - C \cap X$ 
interior_of S
    using assms
  apply (simp add: closure_of_closedin_flip: interior_of_locally_connected_subspace_component)
  by (metis closedin_connected_components_of_closedin_trans_full closure_of_eq
inf.orderE)
  then show ?thesis
    by (simp add: Diff_Int_distrib frontier_of_def)
qed

```

```

lemma locally_connected_space_prod_topology:
  locally_connected_space (prod_topology X Y)  $\longleftrightarrow$ 
    (prod_topology X Y) = trivial_topology  $\vee$ 
    locally_connected_space X  $\wedge$  locally_connected_space Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    using locally_connected_space_iff_weak by force
next
  case False
  then have ne: X  $\neq$  trivial_topology Y  $\neq$  trivial_topology
    by simp_all
  show ?thesis
  proof
    assume ?lhs then show ?rhs
    by (metis locally_connected_space_quotient_map_image ne quotient_map_fst
quotient_map_snd)
  next
    assume ?rhs
  with False have X: locally_connected_space X and Y: locally_connected_space
Y
    by auto
  show ?lhs
    unfolding locally_connected_space_def neighbourhood_base_of
  proof clarify
    fix UV x y
    assume UV: openin (prod_topology X Y) UV and (x,y)  $\in$  UV

    obtain A B where W12: openin X A  $\wedge$  openin Y B  $\wedge$  x  $\in$  A  $\wedge$  y  $\in$  B  $\wedge$  A
 $\times$  B  $\subseteq$  UV
    using X Y by (metis UV  $\langle$ (x,y)  $\in$  UV $\rangle$  openin_prod_topology_alt)
    then obtain C D K L
    where openin X C connectedin X K x  $\in$  C C  $\subseteq$  K K  $\subseteq$  A
    openin Y D connectedin Y L y  $\in$  D D  $\subseteq$  L L  $\subseteq$  B
    by (metis X Y locally_connected_space)
    with W12  $\langle$ openin X C $\rangle$   $\langle$ openin Y D $\rangle$ 
    show  $\exists$  U V. openin (prod_topology X Y) U  $\wedge$  connectedin (prod_topology X
Y) V  $\wedge$  (x, y)  $\in$  U  $\wedge$  U  $\subseteq$  V  $\wedge$  V  $\subseteq$  UV
    apply (rule_tac x=C  $\times$  D in exI)
    apply (rule_tac x=K  $\times$  L in exI)
    apply (auto simp: openin_prod_Times_iff connectedin_Times)
    done
  qed
qed
qed

```

```

lemma locally_connected_space_product_topology:
  locally_connected_space(product_topology X I)  $\longleftrightarrow$ 
    (product_topology X I) = trivial_topology  $\vee$ 
    finite {i. i  $\in$  I  $\wedge$   $\sim$ connected_space(X i)}  $\wedge$ 
    ( $\forall$  i  $\in$  I. locally_connected_space(X i)
    (is ?lhs  $\longleftrightarrow$  ?empty  $\vee$  ?rhs)
proof (cases ?empty)
  case True
    then show ?thesis
    by (simp add: locally_connected_space_def neighbourhood_base_of openin_closedin_eq)
next
  case False
    then obtain z where z: z  $\in$  ( $\prod_{E$  i  $\in$  I. topspace (X i))
    using discrete_topology_unique_derived_set by (fastforce iff: null_topspace_iff_trivial)
    have ?rhs if L: ?lhs
    proof -
      obtain U C where U: openin (product_topology X I) U
        and V: connectedin (product_topology X I) C
        and z  $\in$  U U  $\subseteq$  C and Csub: C  $\subseteq$  ( $\prod_{E$  i  $\in$  I. topspace (X i))
        using L apply (clarsimp simp add: locally_connected_space_def neighbourhood_base_of)
      by (metis openin_topspace_topspace_product_topology z)
      then obtain V where finV: finite {i  $\in$  I. V i  $\neq$  topspace (X i)}
        and XV:  $\bigwedge$  i. i  $\in$  I  $\implies$  openin (X i) (V i) and z  $\in$  PiE I V and subU: PiE I V  $\subseteq$  U
      by (force simp: openin_product_topology_alt)
      show ?thesis
      proof (intro conjI ballI)
        have connected_space (X i) if i  $\in$  I V i = topspace (X i) for i
        proof -
          have pc: connectedin (X i) (( $\lambda$ x. x i) ‘ C)
          by (metis V connectedin_continuous_map_image continuous_map_product_projection
            that(1))
          moreover have (( $\lambda$ x. x i) ‘ C) = topspace (X i)
          proof
            show ( $\lambda$ x. x i) ‘ C  $\subseteq$  topspace (X i)
              by (simp add: pc connectedin_subset_topspace)
            have V i  $\subseteq$  ( $\lambda$ x. x i) ‘ ( $\prod_{E$  i  $\in$  I. V i)
              by (metis  $\langle z \in \text{PiE } I V \rangle$  empty_iff image_projection_PiE order_refl
                that(1))
            also have ...  $\subseteq$  ( $\lambda$ x. x i) ‘ U
              using subU by blast
            finally show topspace (X i)  $\subseteq$  ( $\lambda$ x. x i) ‘ C
              using  $\langle U \subseteq C \rangle$  that by blast
          qed
          ultimately show ?thesis
          by (simp add: connectedin_topspace)
        qed
      then have {i  $\in$  I.  $\neg$  connected_space (X i)}  $\subseteq$  {i  $\in$  I. V i  $\neq$  topspace (X i)}

```

```

    by blast
    with finV show finite {i ∈ I. ¬ connected_space (X i)}
      using finite_subset by blast
  next
    show locally_connected_space (X i) if i ∈ I for i
      by (meson False L locally_connected_space_quotient_map_image_quo-
        tient_map_product_projection that)
    qed
  qed
  moreover have ?lhs if R: ?rhs
  proof (clarsimp simp add: locally_connected_space_def neighbourhood_base_of)
    fix F z
    assume openin (product_topology X I) F and z ∈ F
    then obtain W where finW: finite {i ∈ I. W i ≠ topspace (X i)}
      and opeW: ∧i. i ∈ I ⇒ openin (X i) (W i) and z ∈ Pi_E I W Pi_E I
    W ⊆ F
      by (auto simp: openin_product_topology_alt)
    have ∀i ∈ I. ∃ U C. openin (X i) U ∧ connectedin (X i) C ∧ z i ∈ U ∧ U ⊆
    C ∧ C ⊆ W i ∧
      (W i = topspace (X i) ∧
        connected_space (X i) → U = topspace (X i) ∧ C = topspace
    (X i))
      (is ∀i ∈ I. ?Φ i)
    proof
      fix i assume i ∈ I
      have locally_connected_space (X i)
        by (simp add: R ⟨i ∈ I⟩)
      moreover have *: openin (X i) (W i) z i ∈ W i
        using ⟨z ∈ Pi_E I W⟩ opeW ⟨i ∈ I⟩ by auto
      ultimately obtain U C where openin (X i) U connectedin (X i) C z i ∈ U
      U ⊆ C C ⊆ W i
        using ⟨i ∈ I⟩ by (force simp: locally_connected_space_def neighbour-
        hood_base_of)
      then show ?Φ i
        by (metis * connectedin_topspace openin_subset)
    qed
    then obtain U C where
      *: ∧i. i ∈ I ⇒ openin (X i) (U i) ∧ connectedin (X i) (C i) ∧ z i ∈ (U i)
      ∧ (U i) ⊆ (C i) ∧ (C i) ⊆ W i ∧
        (W i = topspace (X i) ∧ connected_space (X i)
          → (U i) = topspace (X i) ∧ (C i) = topspace (X i))
      by metis
    let ?A = {i ∈ I. ¬ connected_space (X i)} ∪ {i ∈ I. W i ≠ topspace (X i)}
    have {i ∈ I. U i ≠ topspace (X i)} ⊆ ?A
      by (clarsimp simp add: *)
    moreover have finite ?A
      by (simp add: that finW)
    ultimately have finite {i ∈ I. U i ≠ topspace (X i)}
      using finite_subset by auto

```

```

then have openin (product_topology X I) (PiE I U)
  using * by (simp add: openin_PiE_gen)
then show  $\exists U. \text{openin} (\text{product\_topology } X \ I) \ U \wedge$ 
  ( $\exists V. \text{connectedin} (\text{product\_topology } X \ I) \ V \wedge z \in U \wedge U \subseteq V \wedge V \subseteq$ 
F)
  using  $\langle z \in \text{PiE } I \ W \rangle \langle \text{PiE } I \ W \subseteq F \rangle$  *
  by (metis (no_types, opaque_lifting) PiE_iff PiE_mono connectedin_PiE
subset_iff)
qed
ultimately show ?thesis
using False by blast
qed

lemma locally_connected_space_sum_topology:
  locally_connected_space(sum_topology X I)  $\longleftrightarrow$ 
  ( $\forall i \in I. \text{locally\_connected\_space} (X \ i)$ ) (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
  by (smt (verit) homeomorphic_locally_connected_space locally_connected_space_open_subset
topological_property_of_sum_component)
next
  assume R: ?rhs
  show ?lhs
  proof (clarsimp simp add: locally_connected_space_def neighbourhood_base_of
forall_openin_sum_topology imp_conjL)
    fix W i x
    assume ope:  $\forall i \in I. \text{openin} (X \ i) (W \ i)$ 
    and  $i \in I$  and  $x \in W \ i$ 
    then obtain U V where  $U: \text{openin} (X \ i) \ U$  and  $V: \text{connectedin} (X \ i) \ V$ 
    and  $x \in U \ U \subseteq V \ V \subseteq W \ i$ 
    by (metis R  $\langle i \in I \rangle \langle x \in W \ i \rangle$  locally_connected_space)
    show  $\exists U. \text{openin} (\text{sum\_topology } X \ I) \ U \wedge (\exists V. \text{connectedin} (\text{sum\_topology}$ 
X I)  $V \wedge (i, x) \in U \wedge U \subseteq V \wedge V \subseteq \text{Sigma } I \ W)$ 
    proof (intro exI conjI)
      show openin (sum_topology X I) (Pair i ' U)
      by (meson U  $\langle i \in I \rangle$  open_map_component_injection open_map_def)
      show connectedin (sum_topology X I) (Pair i ' V)
      by (meson V  $\langle i \in I \rangle$  continuous_map_component_injection connecte-
din_continuous_map_image)
      show Pair i ' V  $\subseteq \text{Sigma } I \ W$ 
      using  $\langle V \subseteq W \ i \rangle \langle i \in I \rangle$  by force
    qed (use  $\langle x \in U \rangle \langle U \subseteq V \rangle$  in auto)
  qed
qed

```

6.2.4 Dimension of a topological space

Basic definition of the small inductive dimension relation. Works in any topological space.

inductive *dimension_le* :: [*'a topology, int*] \Rightarrow *bool* (**infix** *dim'_le* 50)
where $\llbracket -1 \leq n;$
 $\bigwedge V a. \llbracket \text{openin } X V; a \in V \rrbracket \Longrightarrow \exists U. a \in U \wedge U \subseteq V \wedge \text{openin } X U \wedge$
 $(\text{subtopology } X (X \text{ frontier_of } U)) \text{ dim_le } (n-1) \rrbracket$
 $\Longrightarrow X \text{ dim_le } (n::\text{int})$

lemma *dimension_le_neighbourhood_base*:
 $X \text{ dim_le } n \longleftrightarrow$
 $-1 \leq n \wedge \text{neighbourhood_base_of } (\lambda U. \text{openin } X U \wedge (\text{subtopology } X (X$
 $\text{frontier_of } U)) \text{ dim_le } (n-1)) X$
by (*smt* (*verit, best*) *dimension_le.simps open_neighbourhood_base_of*)

lemma *dimension_le_bound*: $X \text{ dim_le } n \Longrightarrow -1 \leq n$
using *dimension_le.simps* **by** *blast*

lemma *dimension_le_mono* [*rule_format*]:
assumes $X \text{ dim_le } m$
shows $m \leq n \longrightarrow X \text{ dim_le } n$
using *assms*
proof (*induction arbitrary: n rule: dimension_le.induct*)
qed (*smt* (*verit*) *dimension_le.simps*)

inductive_simps *dim_le_minus2* [*simp*]: $X \text{ dim_le } -2$

lemma *dimension_le_eq_empty* [*simp*]:
 $X \text{ dim_le } -1 \longleftrightarrow X = \text{trivial_topology}$
proof
show $X \text{ dim_le } (-1) \Longrightarrow X = \text{trivial_topology}$
by (*force intro: dimension_le.cases*)
next
show $X = \text{trivial_topology} \Longrightarrow X \text{ dim_le } (-1)$
using *dimension_le.simps openin_subset* **by** *fastforce*
qed

lemma *dimension_le_0_neighbourhood_base_of_clopen*:
 $X \text{ dim_le } 0 \longleftrightarrow \text{neighbourhood_base_of } (\lambda U. \text{closedin } X U \wedge \text{openin } X U) X$
proof -
have $(\text{subtopology } X (X \text{ frontier_of } U) \text{ dim_le } -1) = \text{closedin } X U$
if *openin* $X U$ **for** U
using *that clopenin_eq_frontier_of_openin_subset*
by (*fastforce simp add: subtopology_trivial_iff frontier_of_subset_topspace*
Int_absorb1)
then show *?thesis*
by (*smt* (*verit, del_insts*) *dimension_le.simps open_neighbourhood_base_of*)
qed

lemma *dimension_le_subtopology*:
 $X \text{ dim_le } n \Longrightarrow \text{subtopology } X S \text{ dim_le } n$
proof (*induction arbitrary: S rule: dimension_le.induct*)


```

case (1 n X)
show ?case
proof (intro dimension_le.intros)
  show - 1 ≤ n
    by (simp add: 1.hyps)
  fix U' a
  assume U': openin (subtopology X S) U' and a ∈ U'
  then obtain U where U: openin X U U' = U ∩ S
    by (meson openin_subtopology)
  then obtain V where a ∈ V V ⊆ U openin X V
    and subV: subtopology X (X frontier_of V) dim_le n-1
    and dimV: ∧T. subtopology X (X frontier_of V ∩ T) dim_le n-1
    by (metis 1.IH Int_iff ‹a ∈ U'› subtopology_subtopology)
  show ∃ W. a ∈ W ∧ W ⊆ U' ∧ openin (subtopology X S) W ∧ subtopology
(subtopology X S) (subtopology X S frontier_of W) dim_le n-1
  proof (intro exI conjI)
    show a ∈ S ∩ V S ∩ V ⊆ U'
      using ‹U' = U ∩ S› ‹a ∈ U'› ‹a ∈ V› ‹V ⊆ U› by blast+
    show openin (subtopology X S) (S ∩ V)
      by (simp add: ‹openin X V› openin_subtopology_Int2)
    have S ∩ subtopology X S frontier_of V ⊆ X frontier_of V
      by (simp add: frontier_of_subtopology_subset)
    then show subtopology (subtopology X S) (subtopology X S frontier_of (S ∩
V)) dim_le n-1
      by (metis dimV frontier_of_restrict inf.absorb_iff2 inf_left_idem subtopol-
ogy_subtopology tospace_subtopology)
    qed
  qed
qed

```

lemma dimension_le_subtopologies:

```

[[subtopology X T dim_le n; S ⊆ T]] ==> (subtopology X S) dim_le n
by (metis dimension_le_subtopology inf.absorb_iff2 subtopology_subtopology)

```

lemma dimension_le_eq_subtopology:

```

(subtopology X S) dim_le n ‹↔›
-1 ≤ n ∧
(∀ V a. openin X V ∧ a ∈ V ∧ a ∈ S
  → (∃ U. a ∈ U ∧ U ⊆ V ∧ openin X U ∧
    subtopology X (subtopology X S frontier_of (S ∩ U)) dim_le
(n-1)))

```

proof -

```

have *: (∃ T. a ∈ T ∧ T ∩ S ⊆ V ∩ S ∧ openin X T ∧ subtopology X (S ∩
(subtopology X S frontier_of (T ∩ S))) dim_le n-1)

```

```

  ‹↔› (∃ U. a ∈ U ∧ U ⊆ V ∧ openin X U ∧ subtopology X (subtopology X
S frontier_of (S ∩ U)) dim_le n-1)

```

if a ∈ V a ∈ S openin X V **for** a V

proof -

```

have ∃ U. a ∈ U ∧ U ⊆ V ∧ openin X U ∧ subtopology X (subtopology X S

```

```

frontier_of (S ∩ U)) dim_le n-1
  if a ∈ T and sub: T ∩ S ⊆ V ∩ S and openin X T
    and dim: subtopology X (S ∩ subtopology X S frontier_of (T ∩ S)) dim_le
n-1
  for T
    proof (intro exI conjI)
      show openin X (T ∩ V)
        using ⟨openin X V⟩ ⟨openin X T⟩ by blast
      show subtopology X (subtopology X S frontier_of (S ∩ (T ∩ V))) dim_le
n-1
        by (metis dim frontier_of_subset_subtopology inf.boundedE inf_absorb2
inf_assoc inf_commute sub)
      qed (use ⟨a ∈ V⟩ ⟨a ∈ T⟩ in auto)
      moreover have ∃ T. a ∈ T ∧ T ∩ S ⊆ V ∩ S ∧ openin X T ∧ subtopology
X (S ∩ subtopology X S frontier_of (T ∩ S)) dim_le n-1
        if a ∈ U and U ⊆ V and openin X U
          and dim: subtopology X (subtopology X S frontier_of (S ∩ U)) dim_le n-1
        for U
          by (metis that frontier_of_subset_subtopology inf_absorb2 inf_commute
inf_le1 le_inf_iff)
          ultimately show ?thesis
            by safe
          qed
        show ?thesis
          apply (simp add: dimension_le.simps [of _ n] subtopology_subtopology openin_subtopology
flip: *)
            by (safe; metis Int_iff inf_le2 le_inf_iff)
          qed
    qed

```

lemma *homeomorphic_space_dimension_le_aux:*

```

  assumes X homeomorphic_space Y X dim_le of_nat n - 1
  shows Y dim_le of_nat n - 1
  using assms
proof (induction n arbitrary: X Y)
  case 0
  then show ?case
    by (simp add: dimension_le_eq_empty homeomorphic_empty_space)
  next
  case (Suc n)
  then have X_dim_n: X dim_le n
    by simp
  show ?case
proof (clarsimp simp add: dimension_le.simps [of Y n])
  fix V b
  assume openin Y V and b ∈ V
  obtain f g where fg: homeomorphic_maps X Y f g
    using ⟨X homeomorphic_space Y⟩ homeomorphic_space_def by blast
  then have openin X (g ⁻¹ V)

```

```

    using ⟨openin Y V⟩ homeomorphic_map_openness_eq homeomorphic_maps_map
  by blast
  then obtain U where g b ∈ U openin X U and gim: U ⊆ g ‘ V and sub:
  subtopology X (X frontier_of U) dim_le int n - int 1
    using X_dim_n unfolding dimension_le.simps [of X n] by (metis ⟨b ∈ V⟩
  imageI_of_nat_eq_1_iff)
  show ∃ U. b ∈ U ∧ U ⊆ V ∧ openin Y U ∧ subtopology Y (Y frontier_of U)
  dim_le int n - 1
  proof (intro conjI exI)
    show b ∈ f ‘ U
      by (metis (no_types, lifting) ⟨b ∈ V⟩ ⟨g b ∈ U⟩ ⟨openin Y V⟩ fg homeo-
  morphic_maps_map image_iff openin_subset subsetD)
    show f ‘ U ⊆ V
      by (smt (verit, ccfv_threshold) ⟨openin Y V⟩ fg gim homeomorphic_maps_map
  image_iff openin_subset subset_iff)
    show openin Y (f ‘ U)
      using ⟨openin X U⟩ fg homeomorphic_map_openness_eq homeomor-
  phic_maps_map by blast
    show subtopology Y (Y frontier_of f ‘ U) dim_le int n-1
      proof (rule Suc.IH)
        have homeomorphic_maps (subtopology X (X frontier_of U)) (subtopology
  Y (Y frontier_of f ‘ U)) f g
          using ⟨openin X U⟩ fg
            by (metis frontier_of_subset_topspace homeomorphic_map_frontier_of
  homeomorphic_maps_map homeomorphic_maps_subtopologies openin_subset topspace_subtopology
  topspace_subtopology_subset)
        then show subtopology X (X frontier_of U) homeomorphic_space subtopology
  Y (Y frontier_of f ‘ U)
          using homeomorphic_space_def by blast
        show subtopology X (X frontier_of U) dim_le int n-1
          using sub by fastforce
      qed
    qed
  qed
  qed
  qed

lemma homeomorphic_space_dimension_le:
  assumes X homeomorphic_space Y
  shows X dim_le n ↔ Y dim_le n
proof (cases n ≥ -1)
  case True
  then show ?thesis
    using homeomorphic_space_dimension_le_aux [of _ _ nat(n+1)]
    by (smt (verit) assms homeomorphic_space_sym nat_eq_iff)
next
  case False
  then show ?thesis
    by (metis dimension_le_bound)
qed

```

```

lemma dimension_le_retraction_map_image:
   $\llbracket \text{retraction\_map } X \ Y \ r; \ X \ \text{dim\_le } n \rrbracket \implies Y \ \text{dim\_le } n$ 
  by (meson dimension_le_subtopology_homeomorphic_space_dimension_le_retraction_map_def retraction_maps_section_image2)

```

```

lemma dimension_le_discrete_topology [simp]: (discrete_topology  $U$ ) dim_le 0
  using dimension_le_simps dimension_le_eq_empty by fastforce

```

end

6.3 Some Uncountable Sets

```

theory Uncountable_Sets
  imports Path_Connected Continuum_Not_Denumerable
begin

```

```

lemma uncountable_closed_segment:
  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  assumes  $a \neq b$  shows uncountable (closed_segment  $a \ b$ )
unfolding path_image_linepath [symmetric] path_image_def
  using inj_on_linepath [OF assms] uncountable_closed_interval [of 0 1]
  countable_image_inj_on by auto

```

```

lemma uncountable_open_segment:
  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  assumes  $a \neq b$  shows uncountable (open_segment  $a \ b$ )
  by (simp add: assms open_segment_def uncountable_closed_segment uncountable_minus_countable)

```

```

lemma uncountable_convex:
  fixes  $a :: 'a::\text{real\_normed\_vector}$ 
  assumes convex  $S$   $a \in S$   $b \in S$   $a \neq b$ 
  shows uncountable  $S$ 
proof –
  have uncountable (closed_segment  $a \ b$ )
  by (simp add: uncountable_closed_segment assms)
  then show ?thesis
  by (meson assms convex_contains_segment countable_subset)
qed

```

```

lemma uncountable_ball:
  fixes  $a :: 'a::\text{euclidean\_space}$ 
  assumes  $r > 0$ 
  shows uncountable (ball  $a \ r$ )
proof –
  have uncountable (open_segment  $a \ (a + r *_{\mathbb{R}} (\text{SOME } i. i \in \text{Basis}))$ )
  by (metis Basis_zero SOME_Basis add_cancel_right_right assms less_le_scale_eq_0_iff uncountable_open_segment)

```

```

moreover have open_segment a (a + r *R (SOME i. i ∈ Basis)) ⊆ ball a r
using assms by (auto simp: in_segment algebra_simps dist_norm SOME_Basis)
ultimately show ?thesis
by (metis countable_subset)
qed

```

```

lemma ball_minus_countable_nonempty:
  assumes countable (A :: 'a :: euclidean_space set) r > 0
  shows ball z r - A ≠ {}
proof
  assume *: ball z r - A = {}
  have uncountable (ball z r - A)
    by (intro uncountable_minus_countable assms uncountable_ball)
  thus False by (subst (asm) *) auto
qed

```

```

lemma uncountable_cball:
  fixes a :: 'a::euclidean_space
  assumes r > 0
  shows uncountable (cball a r)
  using assms countable_subset uncountable_ball by auto

```

```

lemma pairwise_disjnt_countable:
  fixes  $\mathcal{N}$  :: nat set set
  assumes pairwise_disjnt  $\mathcal{N}$ 
  shows countable  $\mathcal{N}$ 
  by (simp add: assms countable_disjoint_open_subsets open_discrete)

```

```

lemma pairwise_disjnt_countable_Union:
  assumes countable ( $\bigcup \mathcal{N}$ ) and pwd: pairwise_disjnt  $\mathcal{N}$ 
  shows countable  $\mathcal{N}$ 
proof -
  obtain f ::  $\_ \Rightarrow$  nat where f: inj_on f ( $\bigcup \mathcal{N}$ )
  using assms by blast
  then have pairwise_disjnt ( $\bigcup X \in \mathcal{N}. \{f \text{ ` } X\}$ )
  using assms by (force simp: pairwise_def disjnt_inj_on_iff [OF f])
  then have countable ( $\bigcup X \in \mathcal{N}. \{f \text{ ` } X\}$ )
  using pairwise_disjnt_countable by blast
  then show ?thesis
  by (meson pwd countable_image_inj_on disjoint_image f inj_on_image pairwise_disjnt_countable)
qed

```

```

lemma connected_uncountable:
  fixes S :: 'a::metric_space set
  assumes connected S a ∈ S b ∈ S a ≠ b shows uncountable S
proof -
  have continuous_on S (dist a)
  by (intro continuous_intros)

```

```

then have connected (dist a ' S)
  by (metis connected_continuous_image <connected S>)
then have closed_segment 0 (dist a b) ⊆ (dist a ' S)
  by (simp add: assms closed_segment_subset is_interval_connected_1 is_interval_convex)
then have uncountable (dist a ' S)
  by (metis <a ≠ b> countable_subset dist_eq_0_iff uncountable_closed_segment)
then show ?thesis
  by blast
qed

```

```

lemma path_connected_uncountable:
  fixes S :: 'a::metric_space set
  assumes path_connected S a ∈ S b ∈ S a ≠ b shows uncountable S
  using path_connected_imp_connected assms connected_uncountable by metis

```

```

lemma simple_path_image_uncountable:
  fixes g :: real ⇒ 'a::metric_space
  assumes simple_path g
  shows uncountable (path_image g)
proof –
  have g 0 ∈ path_image g g (1/2) ∈ path_image g
    by (simp_all add: path_defs)
  moreover have g 0 ≠ g (1/2)
    using assms by (fastforce simp add: simple_path_def loop_free_def)
  ultimately have  $\forall a. \neg \text{path\_image } g \subseteq \{a\}$ 
    by blast
  then show ?thesis
    using assms connected_simple_path_image connected_uncountable by blast
qed

```

```

lemma arc_image_uncountable:
  fixes g :: real ⇒ 'a::metric_space
  assumes arc g
  shows uncountable (path_image g)
  by (simp add: arc_imp_simple_path assms simple_path_image_uncountable)

```

end

6.4 Homotopy of Maps

```

theory Homotopy
  imports Path_Connected Product_Topology Uncountable_Sets
begin

```

```

definition homotopic_with

```

```

where

```

```

homotopic_with P X Y f g ≡
  ( $\exists h. \text{continuous\_map } (\text{prod\_topology } (\text{top\_of\_set } \{0..1::\text{real}\}) X) Y h \wedge$ 
   ( $\forall x. h(0, x) = f x$ )  $\wedge$ 

```

$$(\forall x. h(1, x) = g x) \wedge \\ (\forall t \in \{0..1\}. P(\lambda x. h(t, x)))$$

p, q are functions $X \rightarrow Y$, and the property P restricts all intermediate maps. We often just want to require that P fixes some subset, but to include the case of a loop homotopy, it is convenient to have a general property P .

abbreviation *homotopic_with_canon* ::

$$[('a :: \text{topological_space} \Rightarrow 'b :: \text{topological_space}) \Rightarrow \text{bool}, 'a \text{ set}, 'b \text{ set}, 'a \Rightarrow 'b, 'a \Rightarrow 'b] \Rightarrow \text{bool}$$

where

$$\text{homotopic_with_canon } P \ S \ T \ p \ q \equiv \text{homotopic_with } P \ (\text{top_of_set } S) \ (\text{top_of_set } T) \ p \ q$$

lemma *split_01*: $\{0..1::\text{real}\} = \{0..1/2\} \cup \{1/2..1\}$
by *force*

lemma *split_01_prod*: $\{0..1::\text{real}\} \times X = (\{0..1/2\} \times X) \cup (\{1/2..1\} \times X)$
by *force*

lemma *image_Pair_const*: $(\lambda x. (x, c)) 'A = A \times \{c\}$
by *auto*

lemma *fst_o_paired [simp]*: $\text{fst} \circ (\lambda(x,y). (f x y, g x y)) = (\lambda(x,y). f x y)$
by *auto*

lemma *snd_o_paired [simp]*: $\text{snd} \circ (\lambda(x,y). (f x y, g x y)) = (\lambda(x,y). g x y)$
by *auto*

lemma *continuous_on_o_Pair*: $\llbracket \text{continuous_on } (T \times X) \ h; \ t \in T \rrbracket \Longrightarrow \text{continuous_on } X \ (h \circ \text{Pair } t)$
by (*fast intro: continuous_intros elim!: continuous_on_subset*)

lemma *continuous_map_o_Pair*:

assumes h : *continuous_map* $(\text{prod_topology } X \ Y) \ Z \ h$ **and** t : $t \in \text{topspace } X$

shows *continuous_map* $Y \ Z \ (h \circ \text{Pair } t)$

by (*intro continuous_map_compose [OF _ h] continuous_intros; simp add: t*)

6.4.1 Trivial properties

We often want to just localize the ending function equality or whatever.

proposition *homotopic_with*:

assumes $\bigwedge h \ k. (\bigwedge x. x \in \text{topspace } X \Longrightarrow h x = k x) \Longrightarrow (P h \longleftrightarrow P k)$

shows *homotopic_with* $P \ X \ Y \ p \ q \longleftrightarrow$

$(\exists h. \text{continuous_map } (\text{prod_topology } (\text{subtopology euclideanreal } \{0..1\}) \ X) \ Y \ h \wedge$

$$(\forall x \in \text{topspace } X. h(0, x) = p x) \wedge \\ (\forall x \in \text{topspace } X. h(1, x) = q x) \wedge \\ (\forall t \in \{0..1\}. P(\lambda x. h(t, x)))$$

```

unfolding homotopic_with_def
apply (rule iffI, blast, clarify)
apply (rule_tac x= $\lambda(u,v)$ . if  $v \in \text{topspace } X$  then  $h(u,v)$  else if  $u = 0$  then  $p$   $v$  else  $q$   $v$  in exI)
apply simp
by (smt (verit, best) SigmaE assms case_prod_conv continuous_map_eq topspace_prod_topology)

```

```

lemma homotopic_with_mono:
assumes hom: homotopic_with P X Y f g
and Q:  $\bigwedge h. [\text{continuous\_map } X Y h; P h] \implies Q h$ 
shows homotopic_with Q X Y f g
using hom unfolding homotopic_with_def
by (force simp: o_def dest: continuous_map_o_Pair intro: Q)

```

```

lemma homotopic_with_imp_continuous_maps:
assumes homotopic_with P X Y f g
shows continuous_map X Y f  $\wedge$  continuous_map X Y g
proof -
obtain h :: real  $\times$  'a  $\Rightarrow$  'b
where conth: continuous_map (prod_topology (top_of_set {0..1}) X) Y h
and h:  $\forall x. h(0, x) = f x \ \forall x. h(1, x) = g x$ 
using assms by (auto simp: homotopic_with_def)
have *:  $t \in \{0..1\} \implies \text{continuous\_map } X Y (h \circ (\lambda x. (t,x)))$  for  $t$ 
by (rule continuous_map_compose [OF _ conth] (simp add: o_def continuous_map_pairwise))
show ?thesis
using h *[of 0] *[of 1] by (simp add: continuous_map_eq)
qed

```

```

lemma homotopic_with_imp_continuous:
assumes homotopic_with_canon P X Y f g
shows continuous_on X f  $\wedge$  continuous_on X g
by (meson assms continuous_map_subtopology_eu homotopic_with_imp_continuous_maps)

```

```

lemma homotopic_with_imp_property:
assumes homotopic_with P X Y f g
shows P f  $\wedge$  P g
proof
obtain h where h:  $\bigwedge x. h(0, x) = f x \ \bigwedge x. h(1, x) = g x$  and P:  $\bigwedge t. t \in \{0..1::\text{real}\} \implies P(\lambda x. h(t,x))$ 
using assms by (force simp: homotopic_with_def)
show P f P g
using P [of 0] P [of 1] by (force simp: h)+
qed

```

```

lemma homotopic_with_equal:
assumes P f P g and conf: continuous_map X Y f and fg:  $\bigwedge x. x \in \text{topspace } X \implies f x = g x$ 
shows homotopic_with P X Y f g

```



```

unfolding homotopic_with_def
proof (intro exI conjI allI ballI)
  let ?h =  $\lambda(t::\text{real},x). \text{if } t = 1 \text{ then } g \ x \ \text{else } f \ x$ 
  show continuous_map (prod_topology (top_of_set {0..1}) X) Y ?h
proof (rule continuous_map_eq)
  show continuous_map (prod_topology (top_of_set {0..1}) X) Y (f  $\circ$  snd)
    by (simp add: contf_continuous_map_of_snd)
qed (auto simp: fg)
  show P ( $\lambda x. ?h \ (t, x)$ ) if  $t \in \{0..1\}$  for  $t$ 
    by (cases  $t = 1$ ) (simp_all add: assms)
qed auto

lemma homotopic_with_imp_subset1:
  homotopic_with_canon P X Y f g  $\implies f \ ' X \subseteq Y$ 
by (meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps)

lemma homotopic_with_imp_subset2:
  homotopic_with_canon P X Y f g  $\implies g \ ' X \subseteq Y$ 
by (meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps)

lemma homotopic_with_imp_funspace1:
  homotopic_with_canon P X Y f g  $\implies f \in X \rightarrow Y$ 
using homotopic_with_imp_subset1 by blast

lemma homotopic_with_imp_funspace2:
  homotopic_with_canon P X Y f g  $\implies g \in X \rightarrow Y$ 
using homotopic_with_imp_subset2 by blast

lemma homotopic_with_subset_left:
   $\llbracket \text{homotopic\_with\_canon } P \ X \ Y \ f \ g; Z \subseteq X \rrbracket \implies \text{homotopic\_with\_canon } P \ Z \ Y \ f \ g$ 
unfolding homotopic_with_def by (auto elim!: continuous_on_subset ex_forward)

lemma homotopic_with_subset_right:
   $\llbracket \text{homotopic\_with\_canon } P \ X \ Y \ f \ g; Y \subseteq Z \rrbracket \implies \text{homotopic\_with\_canon } P \ X \ Z \ f \ g$ 
unfolding homotopic_with_def by (auto elim!: continuous_on_subset ex_forward)

```

6.4.2 Homotopy with P is an equivalence relation

(on continuous functions mapping X into Y that satisfy P, though this only affects reflexivity)

```

lemma homotopic_with_refl [simp]: homotopic_with P X Y f f  $\longleftrightarrow$  continuous_map X Y f  $\wedge$  P f
  by (metis homotopic_with_equal homotopic_with_imp_continuous_maps homotopic_with_imp_property)

```

```

lemma homotopic_with_symD:
  assumes homotopic_with P X Y f g

```

```

      shows homotopic_with P X Y g f
    proof -
      let ?I01 = subtopology euclideanreal {0..1}
      let ?j =  $\lambda y. (1 - \text{fst } y, \text{snd } y)$ 
      have 1: continuous_map (prod_topology ?I01 X) (prod_topology euclideanreal X) ?j
      by (intro continuous_intros; simp add: continuous_map_subtopology_fst prod_topology_subtopology)
      have *: continuous_map (prod_topology ?I01 X) (prod_topology ?I01 X) ?j
      proof -
        have continuous_map (prod_topology ?I01 X) (subtopology (prod_topology euclideanreal X) ({0..1}  $\times$  topspace X)) ?j
        by (simp add: continuous_map_into_subtopology [OF 1] image_subset_iff flip: image_subset_iff_funcset)
        then show ?thesis
        by (simp add: prod_topology_subtopology(1))
      qed
    show ?thesis
    using assms
    apply (clarsimp simp: homotopic_with_def)
    subgoal for h
      by (rule_tac x=h  $\circ (\lambda y. (1 - \text{fst } y, \text{snd } y))$  in exI) (simp add: continuous_map_compose [OF *])
    done
  qed

```

lemma *homotopic_with_sym*:

```

  homotopic_with P X Y f g  $\longleftrightarrow$  homotopic_with P X Y g f
  by (metis homotopic_with_symD)

```

proposition *homotopic_with_trans*:

```

  assumes homotopic_with P X Y f g homotopic_with P X Y g h
  shows homotopic_with P X Y f h
  proof -
    let ?X01 = prod_topology (subtopology euclideanreal {0..1}) X
    obtain k1 k2
      where contk1: continuous_map ?X01 Y k1 and contk2: continuous_map ?X01 Y k2
      and k12:  $\forall x. k1 (1, x) = g x \ \forall x. k2 (0, x) = g x$ 
       $\forall x. k1 (0, x) = f x \ \forall x. k2 (1, x) = h x$ 
      and P:  $\forall t \in \{0..1\}. P (\lambda x. k1 (t, x)) \ \forall t \in \{0..1\}. P (\lambda x. k2 (t, x))$ 
    using assms by (auto simp: homotopic_with_def)
    define k where k  $\equiv \lambda y. \text{if } \text{fst } y \leq 1/2$ 
      then  $(k1 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x, \text{snd } x))) y$ 
      else  $(k2 \circ (\lambda x. (2 *_{\mathbb{R}} \text{fst } x - 1, \text{snd } x))) y$ 
    have keq:  $k1 (2 * u, v) = k2 (2 * u - 1, v)$  if  $u = 1/2$  for  $u v$ 
      by (simp add: k12 that)
    show ?thesis
      unfolding homotopic_with_def
    proof (intro exI conjI)

```

```

show continuous_map ?X01 Y k
  unfolding k_def
proof (rule continuous_map_cases_le)
  show fst: continuous_map ?X01 euclideanreal fst
    using continuous_map_fst continuous_map_in_subtopology by blast
  show continuous_map ?X01 euclideanreal ( $\lambda x. 1/2$ )
    by simp
  show continuous_map (subtopology ?X01 { $y \in \text{topspace } ?X01. \text{fst } y \leq 1/2$ })
Y
    ( $k1 \circ (\lambda x. (2 *_R \text{fst } x, \text{snd } x))$ )
    apply (intro fst continuous_map_compose [OF __ contk1] continuous_intros
continuous_map_into_subtopology continuous_map_from_subtopology | simp)+
    by (force simp: prod_topology_subtopology)
  show continuous_map (subtopology ?X01 { $y \in \text{topspace } ?X01. 1/2 \leq \text{fst } y$ })
Y
    ( $k2 \circ (\lambda x. (2 *_R \text{fst } x - 1, \text{snd } x))$ )
    apply (intro fst continuous_map_compose [OF __ contk2] continuous_intros
continuous_map_into_subtopology continuous_map_from_subtopology | simp)+
    by (force simp: prod_topology_subtopology)
  show ( $k1 \circ (\lambda x. (2 *_R \text{fst } x, \text{snd } x))$ )  $y = (k2 \circ (\lambda x. (2 *_R \text{fst } x - 1, \text{snd } x)))$   $y$ 
    if  $y \in \text{topspace } ?X01$  and  $\text{fst } y = 1/2$  for  $y$ 
    using that by (simp add: keq)
qed
show  $\forall x. k (0, x) = f x$ 
  by (simp add: k12 k_def)
show  $\forall x. k (1, x) = h x$ 
  by (simp add: k12 k_def)
show  $\forall t \in \{0..1\}. P (\lambda x. k (t, x))$ 
proof
  fix  $t$  show  $t \in \{0..1\} \implies P (\lambda x. k (t, x))$ 
    by (cases  $t \leq 1/2$ ) (auto simp: k_def P)
qed
qed
qed

```

lemma homotopic_with_id2:

```

( $\bigwedge x. x \in \text{topspace } X \implies g (f x) = x$ )  $\implies$  homotopic_with ( $\lambda x. \text{True}$ ) X X (g  $\circ$ 
f) id
by (metis comp_apply continuous_map_id eq_id_iff homotopic_with_equal ho-
motopic_with_symD)

```

6.4.3 Continuity lemmas

lemma homotopic_with_compose_continuous_map_left:

```

[[homotopic_with p X1 X2 f g; continuous_map X2 X3 h;  $\bigwedge j. p j \implies q(h \circ j)]]
\implies$  homotopic_with q X1 X3 (h  $\circ$  f) (h  $\circ$  g)
unfolding homotopic_with_def
apply clarify

```

subgoal for k
by (*rule_tac* $x=h \circ k$ **in** exI) (*rule conjI continuous_map_compose* | *simp add:*
 o_def)
done

lemma *homotopic_with_compose_continuous_map_right*:

assumes hom : *homotopic_with* p $X2$ $X3$ f g **and** $conh$: *continuous_map* $X1$ $X2$
 h

and q : $\bigwedge j. p\ j \implies q(j \circ h)$

shows *homotopic_with* q $X1$ $X3$ $(f \circ h)$ $(g \circ h)$

proof –

obtain k

where $contk$: *continuous_map* (*prod_topology* (*subtopology euclideanreal* $\{0..1\}$)
 $X2$) $X3$ k

and k : $\forall x. k(0, x) = f\ x \ \forall x. k(1, x) = g\ x$ **and** p : $\bigwedge t. t \in \{0..1\} \implies p(\lambda x.$
 $k(t, x))$

using hom **unfolding** *homotopic_with_def* **by** *blast*

have $hsnd$: *continuous_map* (*prod_topology* (*subtopology euclideanreal* $\{0..1\}$)
 $X1$) $X2$ $(h \circ snd)$

by (*rule continuous_map_compose* [*OF continuous_map_snd* $conh$])

let $?h = k \circ (\lambda(t,x). (t, h\ x))$

show $?thesis$

unfolding *homotopic_with_def*

proof (*intro exI conjI allI ballI*)

have *continuous_map* (*prod_topology* (*top_of_set* $\{0..1\}$) $X1$)

(*prod_topology* (*top_of_set* $\{0..1::real\}$) $X2$) $(\lambda(t, x). (t, h\ x))$

by (*metis* (*mono_tags*, *lifting*) *case_prod_beta'* *comp_def continuous_map_eq*
continuous_map_fst continuous_map_pairedI $hsnd$)

then show *continuous_map* (*prod_topology* (*subtopology euclideanreal* $\{0..1\}$)
 $X1$) $X3$ $?h$

by (*intro conjI continuous_map_compose* [*OF* $contk$])

show $q(\lambda x. ?h(t, x))$ **if** $t \in \{0..1\}$ **for** t

using q [*OF* p [*OF that*]] **by** (*simp add:* o_def)

qed (*auto simp:* k)

qed

corollary *homotopic_compose*:

assumes *homotopic_with* $(\lambda x. True)$ X Y f f' *homotopic_with* $(\lambda x. True)$ Y Z
 g g'

shows *homotopic_with* $(\lambda x. True)$ X Z $(g \circ f)$ $(g' \circ f')$

by (*metis* *assms homotopic_with_compose_continuous_map_left homotopic_with_compose_continuous*
homotopic_with_imp_continuous_maps homotopic_with_trans)

proposition *homotopic_with_compose_continuous_right*:

\llbracket *homotopic_with_canon* $(\lambda f. p(f \circ h))$ X Y f g ; *continuous_on* W h ; $h \in W$
 $\rightarrow X$ \rrbracket

\implies *homotopic_with_canon* p W Y $(f \circ h)$ $(g \circ h)$

by (*simp add:* *homotopic_with_compose_continuous_map_right image_subset_iff_funcset*)

proposition *homotopic_with_compose_continuous_left*:

$\llbracket \text{homotopic_with_canon } (\lambda f. p (h \circ f)) \ X \ Y \ f \ g; \text{continuous_on } Y \ h; h \in Y \rightarrow Z \rrbracket$

$\implies \text{homotopic_with_canon } p \ X \ Z \ (h \circ f) \ (h \circ g)$

by (*simp add: homotopic_with_compose_continuous_map_left image_subset_iff_funcset*)

lemma *homotopic_from_subtopology*:

$\text{homotopic_with } P \ X \ X' \ f \ g \implies \text{homotopic_with } P \ (\text{subtopology } X \ S) \ X' \ f \ g$

by (*metis continuous_map_id_subt homotopic_with_compose_continuous_map_right o_id*)

lemma *homotopic_on_emptyI*:

assumes $P \ f \ P \ g$

shows $\text{homotopic_with } P \ \text{trivial_topology } X \ f \ g$

by (*metis assms continuous_map_on_empty_empty_iff homotopic_with_equal topspace_discrete_topology*)

lemma *homotopic_on_empty*:

$(\text{homotopic_with } P \ \text{trivial_topology } X \ f \ g \longleftrightarrow P \ f \wedge P \ g)$

using *homotopic_on_emptyI homotopic_with_imp_property* **by** *metis*

lemma *homotopic_with_canon_on_empty*: $\text{homotopic_with_canon } (\lambda x. \text{True})$

$\{ \} \ t \ f \ g$

by (*auto intro: homotopic_with_equal*)

lemma *homotopic_constant_maps*:

$\text{homotopic_with } (\lambda x. \text{True}) \ X \ X' \ (\lambda x. a) \ (\lambda x. b) \longleftrightarrow$

$X = \text{trivial_topology} \vee \text{path_component_of } X' \ a \ b \ (\text{is } ?lhs = ?rhs)$

proof (*cases X = trivial_topology*)

case *False*

then obtain c **where** $c: c \in \text{topspace } X$

by *fastforce*

have $\exists g. \text{continuous_map } (\text{top_of_set } \{0..1::\text{real}\}) \ X' \ g \wedge g \ 0 = a \wedge g \ 1 = b$

if $x \in \text{topspace } X$ **and** $\text{hom}: \text{homotopic_with } (\lambda x. \text{True}) \ X \ X' \ (\lambda x. a) \ (\lambda x. b)$

for x

proof –

obtain $h :: \text{real} \times 'a \Rightarrow 'b$

where $\text{conth}: \text{continuous_map } (\text{prod_topology } (\text{top_of_set } \{0..1\}) \ X) \ X' \ h$

and $h: \bigwedge x. h \ 0, x = a \bigwedge x. h \ 1, x = b$

using hom **by** (*auto simp: homotopic_with_def*)

have $\text{cont}: \text{continuous_map } (\text{top_of_set } \{0..1\}) \ X' \ (h \circ (\lambda t. (t, c)))$

by (*rule continuous_map_compose [OF _ conth] continuous_intros | simp add: c*) $+$

then show *?thesis*

by (*force simp: h*)

qed

moreover have $\text{homotopic_with } (\lambda x. \text{True}) \ X \ X' \ (\lambda x. g \ 0) \ (\lambda x. g \ 1)$

if $x \in \text{topspace } X \ a = g \ 0 \ b = g \ 1 \ \text{continuous_map } (\text{top_of_set } \{0..1\}) \ X' \ g$

for x **and** $g :: \text{real} \Rightarrow 'b$

```

unfolding homotopic_with_def
by (force intro!: continuous_map_compose continuous_intros c that)
ultimately show ?thesis
using False
by (metis c path_component_of_set pathin_def)
qed (simp add: homotopic_on_empty)

```

```

proposition homotopic_with_eq:
assumes h: homotopic_with P X Y f g
and f':  $\bigwedge x. x \in \text{topspace } X \implies f' x = f x$ 
and g':  $\bigwedge x. x \in \text{topspace } X \implies g' x = g x$ 
and P:  $(\bigwedge h k. (\bigwedge x. x \in \text{topspace } X \implies h x = k x) \implies P h \longleftrightarrow P k)$ 
shows homotopic_with P X Y f' g'
by (smt (verit, ccfv_SIG) assms homotopic_with)

```

```

lemma homotopic_with_prod_topology:
assumes homotopic_with p X1 Y1 f f' and homotopic_with q X2 Y2 g g'
and r:  $\bigwedge i j. \llbracket p i; q j \rrbracket \implies r(\lambda(x,y). (i x, j y))$ 
shows homotopic_with r (prod_topology X1 X2) (prod_topology Y1 Y2)
 $(\lambda z. (f(\text{fst } z), g(\text{snd } z))) (\lambda z. (f'(\text{fst } z), g'(\text{snd } z)))$ 

```

proof –

```

obtain h
where h: continuous_map (prod_topology (subtopology euclideanreal {0..1})
X1) Y1 h
and h0:  $\bigwedge x. h (0, x) = f x$ 
and h1:  $\bigwedge x. h (1, x) = f' x$ 
and p:  $\bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket \implies p (\lambda x. h (t,x))$ 
using assms unfolding homotopic_with_def by auto
obtain k
where k: continuous_map (prod_topology (subtopology euclideanreal {0..1})
X2) Y2 k
and k0:  $\bigwedge x. k (0, x) = g x$ 
and k1:  $\bigwedge x. k (1, x) = g' x$ 
and q:  $\bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket \implies q (\lambda x. k (t,x))$ 
using assms unfolding homotopic_with_def by auto
let ?hk =  $\lambda(t,x,y). (h(t,x), k(t,y))$ 
show ?thesis
unfolding homotopic_with_def
proof (intro conjI allI exI)
show continuous_map (prod_topology (subtopology euclideanreal {0..1}) (prod_topology
X1 X2))
 $(\text{prod\_topology } Y1 \ Y2) \ ?hk$ 
unfolding continuous_map_pairwise case_prod_unfold
by (rule conjI continuous_map_pairedI continuous_intros continuous_map_id
 $[\text{unfolded id\_def}]$ 
 $\text{continuous\_map\_fst\_of } [\text{unfolded o\_def}] \ \text{continuous\_map\_snd\_of } [\text{unfolded}$ 
 $\text{o\_def}]$ 
 $\text{continuous\_map\_compose } [OF \_ h, \text{unfolded o\_def}]$ 
 $\text{continuous\_map\_compose } [OF \_ k, \text{unfolded o\_def}])+$ 

```

```

next
  fix x
  show ?hk (0, x) = (f (fst x), g (snd x)) ?hk (1, x) = (f' (fst x), g' (snd x))
    by (auto simp: case_prod_beta h0 k0 h1 k1)
  qed (auto simp: p q r)
qed

lemma homotopic_with_product_topology:
  assumes ht:  $\bigwedge i. i \in I \implies \text{homotopic\_with } (p\ i) (X\ i) (Y\ i) (f\ i) (g\ i)$ 
    and pq:  $\bigwedge h. (\bigwedge i. i \in I \implies p\ i (h\ i)) \implies q(\lambda x. (\lambda i \in I. h\ i (x\ i)))$ 
  shows  $\text{homotopic\_with } q (\text{product\_topology } X\ I) (\text{product\_topology } Y\ I)$ 
     $(\lambda z. (\lambda i \in I. (f\ i) (z\ i))) (\lambda z. (\lambda i \in I. (g\ i) (z\ i)))$ 

proof -
  obtain h
  where h:  $\bigwedge i. i \in I \implies \text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclidean-}$ 
 $\text{real } \{0..1\}) (X\ i)) (Y\ i) (h\ i)$ 
    and h0:  $\bigwedge i x. i \in I \implies h\ i (0, x) = f\ i\ x$ 
    and h1:  $\bigwedge i x. i \in I \implies h\ i (1, x) = g\ i\ x$ 
    and p:  $\bigwedge i t. \llbracket i \in I; t \in \{0..1\} \rrbracket \implies p\ i (\lambda x. h\ i (t, x))$ 
  using ht unfolding homotopic_with_def by metis
  show ?thesis
  unfolding homotopic_with_def
  proof (intro conjI allI exI)
    let ?h =  $\lambda (t, z). \lambda i \in I. h\ i (t, z\ i)$ 
    have  $\text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\}) (\text{product\_topology } X\ I))$ 
       $(Y\ i) (\lambda x. h\ i (fst\ x, snd\ x\ i))$  if  $i \in I$  for  $i$ 
    proof -
      have  $\S: \text{continuous\_map } (\text{prod\_topology } (\text{top\_of\_set } \{0..1\}) (\text{product\_topology } X\ I))$ 
         $(X\ i) (\lambda x. snd\ x\ i)$ 
      using  $\text{continuous\_map\_componentwise continuous\_map\_snd that}$  by fast-
      force
      show ?thesis
      unfolding  $\text{continuous\_map\_pairwise case\_prod\_unfold}$ 
      by (intro conjI that  $\S$   $\text{continuous\_intros continuous\_map\_compose [OF\_}$ 
 $h, \text{unfolded } o\_def]$ )
    qed
    then show  $\text{continuous\_map } (\text{prod\_topology } (\text{subtopology euclideanreal } \{0..1\})$ 
       $(\text{product\_topology } Y\ I)) ?h$ 
      by (auto simp:  $\text{continuous\_map\_componentwise case\_prod\_beta}$ )
    show  $?h (0, x) = (\lambda i \in I. f\ i (x\ i)) ?h (1, x) = (\lambda i \in I. g\ i (x\ i))$  for  $x$ 
      by (auto simp:  $\text{case\_prod\_beta h0 h1}$ )
    show  $\forall t \in \{0..1\}. q (\lambda x. ?h (t, x))$ 
      by (force intro: p pq)
  qed
qed

```

Homotopic triviality implicitly incorporates path-connectedness.

lemma *homotopic_triviality*:

shows $(\forall f g. \text{continuous_on } S f \wedge f \in S \rightarrow T \wedge$
 $\text{continuous_on } S g \wedge g \in S \rightarrow T$
 $\longrightarrow \text{homotopic_with_canon } (\lambda x. \text{True}) S T f g) \longleftrightarrow$
 $(S = \{\} \vee \text{path_connected } T) \wedge$
 $(\forall f. \text{continuous_on } S f \wedge f \in S \rightarrow T \longrightarrow (\exists c. \text{homotopic_with_canon}$
 $(\lambda x. \text{True}) S T f (\lambda x. c)))$
(is ?lhs = ?rhs)

proof *(cases S = {} ∨ T = {})*

case True then show ?thesis

by *(auto simp: homotopic_on_emptyI simp flip: image_subset_iff_funcset)*

next

case False show ?thesis

proof

assume *LHS [rule_format]: ?lhs*

have *pab: path_component T a b if a ∈ T b ∈ T for a b*

proof –

have *homotopic_with_canon (λx. True) S T (λx. a) (λx. b)*

by *(simp add: LHS image_subset_iff that)*

then show ?thesis

using *False homotopic_constant_maps [of top_of_set S top_of_set T a b]*

by *(metis path_component_of_canon_iff topspace_discrete_topology topspace_euclidean_subtopolo*

qed

moreover

have $\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S T f (\lambda x. c)$ **if** *continuous_on S f f ∈ S → T for f*

using *False LHS continuous_on_const that by blast*

ultimately show ?rhs

by *(simp add: path_connected_component)*

next

assume *RHS: ?rhs*

with *False have T: path_connected T*

by *blast*

show ?lhs

proof *clarify*

fix *f g*

assume *continuous_on S f f ∈ S → T continuous_on S g g ∈ S → T*

obtain *c d where c: homotopic_with_canon (λx. True) S T f (λx. c) and d: homotopic_with_canon (λx. True) S T g (λx. d)*

using *RHS ⟨continuous_on S f⟩ ⟨continuous_on S g⟩ ⟨f ∈ S → T⟩ ⟨g ∈ S → T⟩ by presburger*

with T have *path_component T c d*

by *(metis False ex_in_conv homotopic_with_imp_subset2 image_subset_iff path_connected_component)*

then have *homotopic_with_canon (λx. True) S T (λx. c) (λx. d)*

by *(simp add: homotopic_constant_maps)*

with c d show *homotopic_with_canon (λx. True) S T f g*

by *(meson homotopic_with_symD homotopic_with_trans)*

qed

qed
qed

6.4.4 Homotopy of paths, maintaining the same endpoints

definition *homotopic_paths* :: [*'a set, real \Rightarrow 'a, real \Rightarrow 'a::topological_space*] \Rightarrow *bool*

where

homotopic_paths *S p q* \equiv
homotopic_with_canon ($\lambda r. \text{pathstart } r = \text{pathstart } p \wedge \text{pathfinish } r = \text{pathfinish } p$) $\{0..1\}$ *S p q*

lemma *homotopic_paths*:

homotopic_paths *S p q* \longleftrightarrow
 $(\exists h. \text{continuous_on } (\{0..1\} \times \{0..1\}) h \wedge$
 $h \in (\{0..1\} \times \{0..1\}) \rightarrow S \wedge$
 $(\forall x \in \{0..1\}. h(0,x) = p \ x) \wedge$
 $(\forall x \in \{0..1\}. h(1,x) = q \ x) \wedge$
 $(\forall t \in \{0..1::\text{real}\}. \text{pathstart}(h \circ \text{Pair } t) = \text{pathstart } p \wedge$
 $\text{pathfinish}(h \circ \text{Pair } t) = \text{pathfinish } p))$

by (*auto simp: homotopic_paths_def homotopic_with_pathstart_def pathfinish_def*)

proposition *homotopic_paths_imp_pathstart*:

homotopic_paths *S p q* $\implies \text{pathstart } p = \text{pathstart } q$

by (*metis (mono_tags, lifting) homotopic_paths_def homotopic_with_imp_property*)

proposition *homotopic_paths_imp_pathfinish*:

homotopic_paths *S p q* $\implies \text{pathfinish } p = \text{pathfinish } q$

by (*metis (mono_tags, lifting) homotopic_paths_def homotopic_with_imp_property*)

lemma *homotopic_paths_imp_path*:

homotopic_paths *S p q* $\implies \text{path } p \wedge \text{path } q$

using *homotopic_paths_def homotopic_with_imp_continuous_maps path_def continuous_map_subtopology_eu* **by** *blast*

lemma *homotopic_paths_imp_subset*:

homotopic_paths *S p q* $\implies \text{path_image } p \subseteq S \wedge \text{path_image } q \subseteq S$

by (*metis (mono_tags) continuous_map_subtopology_eu homotopic_paths_def homotopic_with_imp_continuous_maps path_image_def*)

proposition *homotopic_paths_refl* [*simp*]: *homotopic_paths* *S p p* $\longleftrightarrow \text{path } p \wedge \text{path_image } p \subseteq S$

by (*simp add: homotopic_paths_def path_def path_image_def*)

proposition *homotopic_paths_sym*: *homotopic_paths* *S p q* $\implies \text{homotopic_paths } S q p$

by (*metis (mono_tags) homotopic_paths_def homotopic_paths_imp_pathfinish homotopic_paths_imp_pathstart homotopic_with_symD*)

proposition *homotopic_paths_sym_eq*: $\text{homotopic_paths } S p q \longleftrightarrow \text{homotopic_paths } S q p$

by (*metis homotopic_paths_sym*)

proposition *homotopic_paths_trans* [*trans*]:

assumes *homotopic_paths* $S p q$ *homotopic_paths* $S q r$

shows *homotopic_paths* $S p r$

using *assms homotopic_paths_imp_pathfinish homotopic_paths_imp_pathstart*

unfolding *homotopic_paths_def*

by (*smt (verit, ccfv_SIG) homotopic_with_mono homotopic_with_trans*)

proposition *homotopic_paths_eq*:

$\llbracket \text{path } p; \text{path_image } p \subseteq S; \bigwedge t. t \in \{0..1\} \implies p t = q t \rrbracket \implies \text{homotopic_paths } S p q$

by (*smt (verit, best) homotopic_paths homotopic_paths_refl*)

proposition *homotopic_paths_reparametrize*:

assumes *path* p

and *pips*: $\text{path_image } p \subseteq S$

and *contf*: *continuous_on* $\{0..1\}$ f

and *f01* : $f \in \{0..1\} \rightarrow \{0..1\}$

and [*simp*]: $f(0) = 0$ $f(1) = 1$

and q : $\bigwedge t. t \in \{0..1\} \implies q(t) = p(f t)$

shows *homotopic_paths* $S p q$

proof –

have *contp*: *continuous_on* $\{0..1\}$ p

by (*metis <path p> path_def*)

then have *continuous_on* $\{0..1\}$ $(p \circ f)$

by (*meson assms(4) contf continuous_on_compose continuous_on_subset image_subset_iff_funcset*)

then have *path* q

by (*simp add: path_def*) (*metis q continuous_on_cong*)

have *piqs*: $\text{path_image } q \subseteq S$

by (*smt (verit, ccfv_threshold) Pi_iff assms(2) assms(4) assms(7) image_subset_iff path_defs(4)*)

have *fb0*: $\bigwedge a b. \llbracket 0 \leq a; a \leq 1; 0 \leq b; b \leq 1 \rrbracket \implies 0 \leq (1 - a) * f b + a * b$

using *f01* **by** *force*

have *fb1*: $\llbracket 0 \leq a; a \leq 1; 0 \leq b; b \leq 1 \rrbracket \implies (1 - a) * f b + a * b \leq 1$ **for** $a b$

by (*intro convex_bound_le*) (*use f01 in auto*)

have *homotopic_paths* $S q p$

proof (*rule homotopic_paths_trans*)

show *homotopic_paths* $S q (p \circ f)$

using q **by** (*force intro: homotopic_paths_eq [OF <path q> piqs]*)

next

show *homotopic_paths* $S (p \circ f) p$

using *pips* [*unfolded path_image_def*]

apply (*simp add: homotopic_paths_def homotopic_with_def*)

apply (*rule_tac x=p \circ (\lambda y. (1 - (fst y)) *_R ((f \circ snd) y) + (fst y) *_R snd y) in exI*)

```

  apply (rule conjI contf continuous_intros continuous_on_subset [OF contp]
| simp)+
  by (auto simp: fb0 fb1 pathstart_def pathfinish_def)
qed
then show ?thesis
  by (simp add: homotopic_paths_sym)
qed

```

```

lemma homotopic_paths_subset:  $\llbracket \text{homotopic\_paths } S \ p \ q; S \subseteq t \rrbracket \implies \text{homotopic\_paths } t \ p \ q$ 
  unfolding homotopic_paths by fast

```

A slightly ad-hoc but useful lemma in constructing homotopies.

```

lemma continuous_on_homotopic_join_lemma:
  fixes q :: [real,real]  $\Rightarrow$  'a::topological_space
  assumes p: continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. p \ (fst \ y) \ (snd \ y))$  (is continuous_on ?A ?p)
  and q: continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. q \ (fst \ y) \ (snd \ y))$  (is continuous_on ?A ?q)
  and pf:  $\bigwedge t. t \in \{0..1\} \implies \text{pathfinish}(p \ t) = \text{pathstart}(q \ t)$ 
  shows continuous_on  $(\{0..1\} \times \{0..1\})$   $(\lambda y. (p(fst \ y) \ + \ + \ + \ q(fst \ y)) \ (snd \ y))$ 
proof -
  have §:  $(\lambda t. p \ (fst \ t) \ (2 * snd \ t)) = ?p \circ (\lambda y. (fst \ y, 2 * snd \ y))$ 
     $(\lambda t. q \ (fst \ t) \ (2 * snd \ t - 1)) = ?q \circ (\lambda y. (fst \ y, 2 * snd \ y - 1))$ 
  by force+
  show ?thesis
    unfolding joinpaths_def
  proof (rule continuous_on_cases_le)
    show continuous_on  $\{y \in ?A. snd \ y \leq 1/2\}$   $(\lambda t. p \ (fst \ t) \ (2 * snd \ t))$ 
      continuous_on  $\{y \in ?A. 1/2 \leq snd \ y\}$   $(\lambda t. q \ (fst \ t) \ (2 * snd \ t - 1))$ 
      continuous_on ?A snd
    unfolding §
  by (rule continuous_intros continuous_on_subset [OF p] continuous_on_subset [OF q] | force)+
  qed (use pf in  $\langle \text{auto simp: mult.commute pathstart_def pathfinish_def} \rangle$ )
qed

```

Congruence properties of homotopy w.r.t. path-combining operations.

```

lemma homotopic_paths_reversepath_D:
  assumes homotopic_paths S p q
  shows homotopic_paths S (reversepath p) (reversepath q)
  using assms
  apply (simp add: homotopic_paths_def homotopic_with_def, clarify)
  apply (rule_tac x=h  $\circ (\lambda x. (fst \ x, 1 - snd \ x))$  in exI)
  apply (rule conjI continuous_intros)+
  apply (auto simp: reversepath_def pathstart_def pathfinish_def elim!: continuous_on_subset)
  done

```

proposition *homotopic_paths_reversepath*:

$\llbracket \text{homotopic_paths } S \text{ (reversepath } p) \text{ (reversepath } q) \rrbracket \iff \text{homotopic_paths } S \text{ } p$
 q
using *homotopic_paths_reversepath_D* **by force**

proposition *homotopic_paths_join*:

$\llbracket \text{homotopic_paths } S \text{ } p \text{ } p'; \text{ homotopic_paths } S \text{ } q \text{ } q'; \text{ pathfinish } p = \text{pathstart } q \rrbracket$
 $\implies \text{homotopic_paths } S \text{ (} p \text{ +++ } q) \text{ (} p' \text{ +++ } q')$
apply (*clarsimp simp: homotopic_paths_def homotopic_with_def*)
apply (*rename_tac k1 k2*)
apply (*rule_tac x=($\lambda y. ((k1 \circ \text{Pair } (fst y)) \text{ +++ } (k2 \circ \text{Pair } (fst y)))$) (snd y)*)
in *exI*)
apply (*intro conjI continuous_intros continuous_on_homotopic_join_lemma;*
force simp: joinpaths_def pathstart_def pathfinish_def path_image_def)
done

proposition *homotopic_paths_continuous_image*:

$\llbracket \text{homotopic_paths } S \text{ } f \text{ } g; \text{ continuous_on } S \text{ } h; h \in S \rightarrow t \rrbracket \implies \text{homotopic_paths}$
 $t \text{ (} h \circ f) \text{ (} h \circ g)$
unfolding *homotopic_paths_def*
by (*simp add: homotopic_with_compose_continuous_map_left pathfinish_compose*
pathstart_compose image_subset_iff_funcset)

6.4.5 Group properties for homotopy of paths

So taking equivalence classes under homotopy would give the fundamental group

proposition *homotopic_paths_rid*:

assumes *path* p *path_image* $p \subseteq S$
shows *homotopic_paths* $S \text{ (} p \text{ +++ linepath (pathfinish } p) \text{ (pathfinish } p)) \text{ } p$

proof –

have \S : *continuous_on* $\{0..1\}$ ($\lambda t::\text{real. if } t \leq 1/2 \text{ then } 2 *_{\mathbb{R}} t \text{ else } 1$)
unfolding *split_01*
by (*rule continuous_on_cases continuous_intros | force simp: pathfinish_def*
joinpaths_def)
show *?thesis*
apply (*rule homotopic_paths_sym*)
using *assms unfolding pathfinish_def joinpaths_def*
by (*intro* \S *continuous_on_cases continuous_intros homotopic_paths_reparametrize*
 $[\text{where } f = \lambda t. \text{ if } t \leq 1/2 \text{ then } 2 *_{\mathbb{R}} t \text{ else } 1]; \text{ force}$)
qed

proposition *homotopic_paths_lid*:

$\llbracket \text{path } p; \text{ path_image } p \subseteq S \rrbracket \implies \text{homotopic_paths } S \text{ (linepath (pathstart } p)$
 $(\text{pathstart } p) \text{ +++ } p)$
using *homotopic_paths_rid* [*of reversepath* p S]
by (*metis homotopic_paths_reversepath path_image_reversepath path_reversepath*
pathfinish_linepath)

pathfinish_reversepath reversepath_joinpaths reversepath_linepath)

lemma *homotopic_paths_rid'*:

assumes *path p path_image p \subseteq s x = pathfinish p*
shows *homotopic_paths s (p +++ linepath x x) p*
using *homotopic_paths_rid[of p s] assms by simp*

lemma *homotopic_paths_lid'*:

$\llbracket \text{path } p; \text{path_image } p \subseteq s; x = \text{pathstart } p \rrbracket \implies \text{homotopic_paths } s (\text{linepath } x \text{ } x \text{ } +++ p) p$
using *homotopic_paths_lid[of p s] by simp*

proposition *homotopic_paths_assoc*:

$\llbracket \text{path } p; \text{path_image } p \subseteq S; \text{path } q; \text{path_image } q \subseteq S; \text{path } r; \text{path_image } r \subseteq S; \text{pathfinish } p = \text{pathstart } q; \text{pathfinish } q = \text{pathstart } r \rrbracket$
 $\implies \text{homotopic_paths } S (p \text{ } +++ (q \text{ } +++ r)) ((p \text{ } +++ q) \text{ } +++ r)$
apply (*subst homotopic_paths_sym*)
apply (*rule homotopic_paths_reparametrize*
 $\llbracket \text{where } f = \lambda t. \text{if } t \leq 1/2 \text{ then inverse } 2 *_{\mathbb{R}} t$
 $\text{else if } t \leq 3/4 \text{ then } t - (1/4)$
 $\text{else } 2 *_{\mathbb{R}} t - 1 \rrbracket$)
apply (*simp_all del: le_divide_eq_numeral1 add: subset_path_image_join*)
apply (*rule continuous_on_cases_1 continuous_intros | auto simp: joinpaths_def*)
done

proposition *homotopic_paths_rinv*:

assumes *path p path_image p \subseteq S*
shows *homotopic_paths S (p +++ reversepath p) (linepath (pathstart p) (pathstart p))*
proof –
have *p: continuous_on {0..1} p*
using *assms by (auto simp: path_def)*
let *?A = {0..1} \times {0..1}*
have *continuous_on ?A ($\lambda x. (\text{subpath } 0 (\text{fst } x) p \text{ } +++ \text{reversepath } (\text{subpath } 0 (\text{fst } x) p)) (\text{snd } x)$)*
unfolding *joinpaths_def subpath_def reversepath_def path_def add_0_right diff_0_right*
proof (*rule continuous_on_cases_le*)
show *continuous_on {x \in ?A. snd x \leq 1/2} ($\lambda t. p (\text{fst } t * (2 * \text{snd } t))$)*
 $\text{continuous_on } \{x \in ?A. 1/2 \leq \text{snd } x\} (\lambda t. p (\text{fst } t * (1 - (2 * \text{snd } t - 1))))$
 $\text{continuous_on } ?A \text{ snd}$
by (*intro continuous_on_compose2 [OF p] continuous_intros; auto simp: mult_le_one*)
qed (*auto simp: algebra_simps*)
then show *?thesis*
using *assms*
apply (*subst homotopic_paths_sym_eq*)

unfolding *homotopic_paths_def* *homotopic_with_def*
apply (*rule_tac* $x=(\lambda y. (\text{subpath } 0 (\text{fst } y) p \text{ +++ reversepath}(\text{subpath } 0 (\text{fst } y) p)) (\text{snd } y))$ **in** *exI*)
apply (*force simp: mult_le_one path_defs joinpaths_def subpath_def reversepath_def*)
done
qed

proposition *homotopic_paths_linv*:
assumes *path* p *path_image* $p \subseteq S$
shows *homotopic_paths* S (*reversepath* p +++ p) (*linepath* (*pathfinish* p))
(*pathfinish* p)
using *homotopic_paths_rinv* [*of reversepath* p S] **assms by** *simp*

6.4.6 Homotopy of loops without requiring preservation of endpoints

definition *homotopic_loops* :: '*a*::*topological_space* *set* \Rightarrow (*real* \Rightarrow '*a*) \Rightarrow (*real* \Rightarrow '*a*) \Rightarrow *bool* **where**
homotopic_loops S p $q \equiv$
homotopic_with_canon ($\lambda r. \text{pathfinish } r = \text{pathstart } r$) $\{0..1\}$ S p q

lemma *homotopic_loops*:
homotopic_loops S p $q \iff$
 $(\exists h. \text{continuous_on } (\{0..1::\text{real}\} \times \{0..1\}) h \wedge$
 $\text{image } h (\{0..1\} \times \{0..1\}) \subseteq S \wedge$
 $(\forall x \in \{0..1\}. h(0,x) = p \ x) \wedge$
 $(\forall x \in \{0..1\}. h(1,x) = q \ x) \wedge$
 $(\forall t \in \{0..1\}. \text{pathfinish}(h \circ \text{Pair } t) = \text{pathstart}(h \circ \text{Pair } t)))$
by (*simp add: homotopic_loops_def pathstart_def pathfinish_def homotopic_with*)

proposition *homotopic_loops_imp_loop*:
homotopic_loops S p $q \implies \text{pathfinish } p = \text{pathstart } p \wedge \text{pathfinish } q = \text{pathstart } q$
using *homotopic_with_imp_property* *homotopic_loops_def* **by** *blast*

proposition *homotopic_loops_imp_path*:
homotopic_loops S p $q \implies \text{path } p \wedge \text{path } q$
unfolding *homotopic_loops_def path_def*
using *homotopic_with_imp_continuous_maps continuous_map_subtopology_eu*
by *blast*

proposition *homotopic_loops_imp_subset*:
homotopic_loops S p $q \implies \text{path_image } p \subseteq S \wedge \text{path_image } q \subseteq S$
unfolding *homotopic_loops_def path_image_def*
by (*meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps*)

proposition *homotopic_loops_refl*:
homotopic_loops S p $p \iff$
 $\text{path } p \wedge \text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$

by (*simp add: homotopic_loops_def path_image_def path_def*)

proposition *homotopic_loops_sym*: $\text{homotopic_loops } S \ p \ q \implies \text{homotopic_loops } S \ q \ p$

by (*simp add: homotopic_loops_def homotopic_with_sym*)

proposition *homotopic_loops_sym_eq*: $\text{homotopic_loops } S \ p \ q \longleftrightarrow \text{homotopic_loops } S \ q \ p$

by (*metis homotopic_loops_sym*)

proposition *homotopic_loops_trans*:

$\llbracket \text{homotopic_loops } S \ p \ q; \text{homotopic_loops } S \ q \ r \rrbracket \implies \text{homotopic_loops } S \ p \ r$

unfolding *homotopic_loops_def* **by** (*blast intro: homotopic_with_trans*)

proposition *homotopic_loops_subset*:

$\llbracket \text{homotopic_loops } S \ p \ q; S \subseteq t \rrbracket \implies \text{homotopic_loops } t \ p \ q$

by (*fastforce simp: homotopic_loops*)

proposition *homotopic_loops_eq*:

$\llbracket \text{path } p; \text{path_image } p \subseteq S; \text{pathfinish } p = \text{pathstart } p; \bigwedge t. t \in \{0..1\} \implies p(t) = q(t) \rrbracket$

$\implies \text{homotopic_loops } S \ p \ q$

unfolding *homotopic_loops_def path_image_def path_def pathstart_def pathfinish_def*

by (*auto intro: homotopic_with_eq [OF homotopic_with_refl [where f = p, THEN iffD2]]*)

proposition *homotopic_loops_continuous_image*:

$\llbracket \text{homotopic_loops } S \ f \ g; \text{continuous_on } S \ h; h \in S \rightarrow t \rrbracket \implies \text{homotopic_loops } t \ (h \circ f) \ (h \circ g)$

unfolding *homotopic_loops_def*

by (*simp add: homotopic_with_compose_continuous_map_left pathfinish_def pathstart_def image_subset_iff_funcset*)

6.4.7 Relations between the two variants of homotopy

proposition *homotopic_paths_imp_homotopic_loops*:

$\llbracket \text{homotopic_paths } S \ p \ q; \text{pathfinish } p = \text{pathstart } p; \text{pathfinish } q = \text{pathstart } p \rrbracket \implies \text{homotopic_loops } S \ p \ q$

by (*auto simp: homotopic_with_def homotopic_paths_def homotopic_loops_def*)

proposition *homotopic_loops_imp_homotopic_paths_null*:

assumes *homotopic_loops* $S \ p$ (*linepath* $a \ a$)

shows *homotopic_paths* $S \ p$ (*linepath* (*pathstart* p)) (*pathstart* p)

proof –

have *path* p **by** (*metis assms homotopic_loops_imp_path*)

have *ploop*: *pathfinish* $p = \text{pathstart } p$ **by** (*metis assms homotopic_loops_imp_loop*)

have *pip*: *path_image* $p \subseteq S$ **by** (*metis assms homotopic_loops_imp_subset*)

let $?A = \{0..1::\text{real}\} \times \{0..1::\text{real}\}$

```

obtain  $h$  where  $conth$ :  $continuous\_on\ ?A\ h$ 
  and  $hs$ :  $h \in ?A \rightarrow S$ 
  and  $h0[simp]$ :  $\bigwedge x. x \in \{0..1\} \implies h(0,x) = p\ x$ 
  and  $h1[simp]$ :  $\bigwedge x. x \in \{0..1\} \implies h(1,x) = a$ 
  and  $ends$ :  $\bigwedge t. t \in \{0..1\} \implies pathfinish\ (h \circ Pair\ t) = pathstart\ (h \circ$ 
Pair  $t)$ 
  using  $assms$  by ( $auto\ simp$ :  $homotopic\_loops\ homotopic\_with\ image\_subset\_iff\_funcset$ )
  have  $conth0$ :  $path\ (\lambda u. h\ (u, 0))$ 
  unfolding  $path\_def$ 
  proof ( $rule\ continuous\_on\_compose\ [of\ \_ \_ h, unfolded\ o\_def]$ )
  show  $continuous\_on\ ((\lambda x. (x, 0))\ ' \{0..1\})\ h$ 
  by ( $force\ intro$ :  $continuous\_on\_subset\ [OF\ conth]$ )
qed ( $force\ intro$ :  $continuous\_intros$ )
  have  $pih0$ :  $path\_image\ (\lambda u. h\ (u, 0)) \subseteq S$ 
  using  $hs$  by ( $force\ simp$ :  $path\_image\_def$ )
  have  $c1$ :  $continuous\_on\ ?A\ (\lambda x. h\ (fst\ x * snd\ x, 0))$ 
  proof ( $rule\ continuous\_on\_compose\ [of\ \_ \_ h, unfolded\ o\_def]$ )
  show  $continuous\_on\ ((\lambda x. (fst\ x * snd\ x, 0))\ ' ?A)\ h$ 
  by ( $force\ simp$ :  $mult\_le\_one\ intro$ :  $continuous\_on\_subset\ [OF\ conth]$ )
qed ( $force\ intro$ :  $continuous\_intros$ )
  have  $c2$ :  $continuous\_on\ ?A\ (\lambda x. h\ (fst\ x - fst\ x * snd\ x, 0))$ 
  proof ( $rule\ continuous\_on\_compose\ [of\ \_ \_ h, unfolded\ o\_def]$ )
  show  $continuous\_on\ ((\lambda x. (fst\ x - fst\ x * snd\ x, 0))\ ' ?A)\ h$ 
  by ( $auto\ simp$ :  $algebra\_simps\ add\_increasing2\ mult\_left\_le\ intro$ :  $continu-$ 
ous_on_subset [OF conth])
qed ( $force\ intro$ :  $continuous\_intros$ )
  have  $[simp]$ :  $\bigwedge t. \llbracket 0 \leq t \wedge t \leq 1 \rrbracket \implies h\ (t, 1) = h\ (t, 0)$ 
  using  $ends$  by ( $simp\ add$ :  $pathfinish\_def\ pathstart\_def$ )
  have  $adhoc\_le$ :  $c * 4 \leq 1 + c * (d * 4)$  if  $\neg d * 4 \leq 3\ 0 \leq c\ c \leq 1$  for  $c$ 
 $d::real$ 
  proof -
  have  $c * 3 \leq c * (d * 4)$  using  $that\ less\_eq\_real\_def$  by  $auto$ 
  with  $\langle c \leq 1 \rangle$  show  $?thesis$  by  $fastforce$ 
qed
  have  $*$ :  $\bigwedge p\ x. \llbracket path\ p \wedge path(reversepath\ p);$ 
 $path\_image\ p \subseteq S \wedge path\_image(reversepath\ p) \subseteq S;$ 
 $pathfinish\ p = pathstart(linepath\ a\ a\ +++\ reversepath\ p) \wedge$ 
 $pathstart(reversepath\ p) = a \wedge pathstart\ p = x \rrbracket$ 
 $\implies homotopic\_paths\ S\ (p\ +++\ linepath\ a\ a\ +++\ reversepath\ p)$ 
( $linepath\ x\ x$ )
  by ( $metis\ homotopic\_paths\_lid\ homotopic\_paths\_join$ 
 $homotopic\_paths\_trans\ homotopic\_paths\_sym\ homotopic\_paths\_rinv$ )
  have  $1$ :  $homotopic\_paths\ S\ p\ (p\ +++\ linepath\ (pathfinish\ p)\ (pathfinish\ p))$ 
  using  $\langle path\ p \rangle$   $homotopic\_paths\_rid\ homotopic\_paths\_sym\ pip$  by  $blast$ 
  moreover have  $homotopic\_paths\ S\ (p\ +++\ linepath\ (pathfinish\ p)\ (pathfinish$ 
 $p))$ 
 $(linepath\ (pathstart\ p)\ (pathstart\ p)\ +++\ p\ +++$ 
 $linepath\ (pathfinish\ p)\ (pathfinish\ p))$ 
  using  $homotopic\_paths\_lid\ [of\ p\ +++\ linepath\ (pathfinish\ p)\ (pathfinish\ p)\ S]$ 

```



```

  by (metis 1 homotopic_paths_imp_path homotopic_paths_imp_subset homotopic_paths_sym pathstart_join)
  moreover
  have homotopic_paths S (linepath (pathstart p) (pathstart p) +++ p +++ linepath (pathfinish p) (pathfinish p))
    ((λu. h (u, 0)) +++ linepath a a +++ reversepath (λu. h (u, 0)))
  unfolding homotopic_paths_def homotopic_with_def
  proof (intro exI strip conjI)
    let ?h = λy. (subpath 0 (fst y) (λu. h (u, 0)) +++ (λu. h (Pair (fst y) u)) +++ subpath (fst y) 0 (λu. h (u, 0))) (snd y)
    have continuous_on ?A ?h
    by (intro continuous_on_homotopic_join_lemma; simp add: path_defs joinpaths_def subpath_def conth c1 c2)
    moreover have ?h ∈ ?A → S
    using hs
    unfolding joinpaths_def subpath_def
    by (force simp: algebra_simps mult_le_one mult_left_le intro: adhoc_le)
    ultimately show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set {0..1}))
      (top_of_set S) ?h
    by (simp add: subpath_reversepath_image_subset_iff_funcset)
  qed (use ploop in ⟨simp_all add: reversepath_def path_defs joinpaths_def o_def subpath_def conth c1 c2⟩)
  moreover have homotopic_paths S ((λu. h (u, 0)) +++ linepath a a +++ reversepath (λu. h (u, 0)))
    (linepath (pathstart p) (pathstart p))
  by (rule *; simp add: pih0 pathstart_def pathfinish_def conth0; simp add: reversepath_def joinpaths_def)
  ultimately show ?thesis
  by (blast intro: homotopic_paths_trans)
qed

```

proposition *homotopic_loops_conjugate:*

```

  fixes S :: 'a::real_normed_vector set
  assumes path p path q and pip: path_image p ⊆ S and piq: path_image q ⊆ S
    and pq: pathfinish p = pathstart q and qloop: pathfinish q = pathstart q
  shows homotopic_loops S (p +++ q +++ reversepath p) q
proof -
  have contp: continuous_on {0..1} p using ⟨path p⟩ [unfolded path_def] by blast
  have contq: continuous_on {0..1} q using ⟨path q⟩ [unfolded path_def] by blast
  let ?A = {0..1::real} × {0..1::real}
  have c1: continuous_on ?A (λx. p ((1 - fst x) * snd x + fst x))
  proof (rule continuous_on_compose [of _ _ p, unfolded o_def])
    show continuous_on ((λx. (1 - fst x) * snd x + fst x) ‘ ?A) p
    by (auto intro: continuous_on_subset [OF contp] simp: algebra_simps add_increasing2 mult_right_le_one_le sum_le_prod1)
  qed (force intro: continuous_intros)
  have c2: continuous_on ?A (λx. p ((fst x - 1) * snd x + 1))

```

```

proof (rule continuous_on_compose [of _ _ p, unfolded o_def])
  show continuous_on ((λx. (fst x - 1) * snd x + 1) ' ?A) p
  by (auto intro: continuous_on_subset [OF contp] simp: algebra_simps add_increasing2
mult_left_le_one_le)
  qed (force intro: continuous_intros)

have ps1: ∧ a b. [b * 2 ≤ 1; 0 ≤ b; 0 ≤ a; a ≤ 1] ⇒ p ((1 - a) * (2 * b) +
a) ∈ S
  using sum_le_prod1
  by (force simp: algebra_simps add_increasing2 mult_left_le intro: pip [unfolded
path_image_def, THEN subsetD])
  have ps2: ∧ a b. [¬ 4 * b ≤ 3; b ≤ 1; 0 ≤ a; a ≤ 1] ⇒ p ((a - 1) * (4 * b
- 3) + 1) ∈ S
  apply (rule pip [unfolded path_image_def, THEN subsetD])
  apply (rule image_eqI, blast)
  apply (simp add: algebra_simps)
  by (metis add_mono affine_ineq linear_mult commute mult.left_neutral mult_right_mono
add commute zero_le numeral)
have qs: ∧ a b. [4 * b ≤ 3; ¬ b * 2 ≤ 1] ⇒ q (4 * b - 2) ∈ S
  using path_image_def piq by fastforce
have homotopic_loops S (p +++ q +++ reversepath p)
  (linepath (pathstart q) (pathstart q) +++ q +++ linepath
(pathstart q) (pathstart q))
  unfolding homotopic_loops_def homotopic_with_def
proof (intro exI strip conjI)
  let ?h = (λy. (subpath (fst y) 1 p +++ q +++ subpath 1 (fst y) p) (snd y))
  have continuous_on ?A (λy. q (snd y))
  by (force simp: contq intro: continuous_on_compose [of _ _ q, unfolded
o_def] continuous_on_id continuous_on_snd)
  then have continuous_on ?A ?h
  using pq qloop
  by (intro continuous_on_homotopic_join_lemma) (auto simp: path_defs
joinpaths_def subpath_def c1 c2)
  then show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set
{0..1})) (top_of_set S) ?h
  by (auto simp: joinpaths_def subpath_def ps1 ps2 qs)
  show ?h (1,x) = (linepath (pathstart q) (pathstart q) +++ q +++ linepath
(pathstart q) (pathstart q)) x for x
  using pq by (simp add: pathfinish_def subpath_refl)
  qed (auto simp: subpath_reversepath)
moreover have homotopic_loops S (linepath (pathstart q) (pathstart q) +++ q
+++ linepath (pathstart q) (pathstart q)) q
proof -
  have homotopic_paths S (linepath (pathfinish q) (pathfinish q) +++ q) q
  using ⟨path q⟩ homotopic_paths_lid qloop piq by auto
  hence 1: ∧ f. homotopic_paths S f q ∨ ¬ homotopic_paths S f (linepath
(pathfinish q) (pathfinish q) +++ q)
  using homotopic_paths_trans by blast
  hence homotopic_paths S (linepath (pathfinish q) (pathfinish q) +++ q +++

```

```

linepath (pathfinish q) (pathfinish q)) q
  by (smt (verit, best) <path q> homotopic_paths_imp_path homotopic_paths_imp_subset
homotopic_paths_lid
      homotopic_paths_rid homotopic_paths_trans pathstart_join piq qloop)
  thus ?thesis
  by (metis (no_types) qloop homotopic_loops_sym homotopic_paths_imp_homotopic_loops
homotopic_paths_imp_pathfinish homotopic_paths_sym)
qed
ultimately show ?thesis
  by (blast intro: homotopic_loops_trans)
qed

```

lemma *homotopic_paths_loop_parts*:

```

  assumes loops: homotopic_loops S (p +++ reversepath q) (linepath a a) and
  path q
  shows homotopic_paths S p q
proof –
  have paths: homotopic_paths S (p +++ reversepath q) (linepath (pathstart p)
(pathstart p))
  using homotopic_loops_imp_homotopic_paths_null [OF loops] by simp
  then have path p
  using <path q> homotopic_loops_imp_path loops path_join path_join_path_ends
path_reversepath by blast
  show ?thesis
  proof (cases pathfinish p = pathfinish q)
  case True
  obtain pipq: path_image p  $\subseteq$  S path_image q  $\subseteq$  S
  by (metis Un_subset_iff paths <path p> <path q> homotopic_loops_imp_subset
homotopic_paths_imp_path loops
      path_image_join path_image_reversepath path_imp_reversepath path_join_eq)
  have homotopic_paths S p (p +++ (linepath (pathfinish p) (pathfinish p)))
  using <path p> <path_image p  $\subseteq$  S> homotopic_paths_rid homotopic_paths_sym
by blast
  moreover have homotopic_paths S (p +++ (linepath (pathfinish p) (pathfinish
p))) (p +++ (reversepath q +++ q))
  by (simp add: True <path p> <path q> pipq homotopic_paths_join homo-
topic_paths_linv homotopic_paths_sym)
  moreover have homotopic_paths S (p +++ (reversepath q +++ q)) ((p +++
reversepath q) +++ q)
  by (simp add: True <path p> <path q> homotopic_paths_assoc pipq)
  moreover have homotopic_paths S ((p +++ reversepath q) +++ q) (linepath
(pathstart p) (pathstart p) +++ q)
  by (simp add: <path q> homotopic_paths_join paths pipq)
  ultimately show ?thesis
  by (metis <path q> homotopic_paths_imp_path homotopic_paths_lid homo-
topic_paths_trans path_join_path_ends pathfinish_linepath pipq(2))
  next
  case False
  then show ?thesis

```

```

    using ⟨path q⟩ homotopic_loops_imp_path loops path_join_path_ends by
fastforce
qed
qed

```

6.4.8 Homotopy of "nearby" function, paths and loops

lemma *homotopic_with_linear*:

```

fixes f g :: _ ⇒ 'b::real_normed_vector
assumes contf: continuous_on S f
    and contg: continuous_on S g
    and sub:  $\bigwedge x. x \in S \implies \text{closed\_segment } (f x) (g x) \subseteq t$ 
shows homotopic_with_canon ( $\lambda z. \text{True}$ ) S t f g
unfolding homotopic_with_def
apply (rule_tac x= $\lambda y. ((1 - (fst y)) *_{\mathbb{R}} f(\text{snd } y) + (fst y) *_{\mathbb{R}} g(\text{snd } y))$  in exI)
using sub closed_segment_def
by (fastforce intro: continuous_intros continuous_on_subset [OF contf] continuous_on_compose2 [where g=f]
    continuous_on_subset [OF contg] continuous_on_compose2 [where g=g])

```

lemma *homotopic_paths_linear*:

```

fixes g h :: real ⇒ 'a::real_normed_vector
assumes path g path h pathstart h = pathstart g pathfinish h = pathfinish g
     $\bigwedge t. t \in \{0..1\} \implies \text{closed\_segment } (g t) (h t) \subseteq S$ 
shows homotopic_paths S g h
using assms
unfolding path_def
apply (simp add: closed_segment_def pathstart_def pathfinish_def homotopic_paths_def
    homotopic_with_def)
apply (rule_tac x= $\lambda y. ((1 - (fst y)) *_{\mathbb{R}} (g \circ \text{snd}) y + (fst y) *_{\mathbb{R}} (h \circ \text{snd}) y)$ 
in exI)
apply (intro conjI subsetI continuous_intros; force)
done

```

lemma *homotopic_loops_linear*:

```

fixes g h :: real ⇒ 'a::real_normed_vector
assumes path g path h pathfinish g = pathstart g pathfinish h = pathstart h
     $\bigwedge t x. t \in \{0..1\} \implies \text{closed\_segment } (g t) (h t) \subseteq S$ 
shows homotopic_loops S g h
using assms
unfolding path_defs homotopic_loops_def homotopic_with_def
apply (rule_tac x= $\lambda y. ((1 - (fst y)) *_{\mathbb{R}} g(\text{snd } y) + (fst y) *_{\mathbb{R}} h(\text{snd } y))$  in exI)
by (force simp: closed_segment_def intro!: continuous_intros intro: continuous_on_compose2 [where g=g] continuous_on_compose2 [where g=h])

```

lemma *homotopic_paths_nearby_explicit*:

```

assumes §: path g path h pathstart h = pathstart g pathfinish h = pathfinish g
    and no:  $\bigwedge t x. [t \in \{0..1\}; x \notin S] \implies \text{norm}(h t - g t) < \text{norm}(g t - x)$ 

```

shows *homotopic_paths* S g h
using *homotopic_paths_linear* [OF §] **by** (*metis linorder_not_le no_norm_minus_commute segment_bound1 subsetI*)

lemma *homotopic_loops_nearby_explicit*:

assumes §: *path* g *path* h *pathfinish* $g = \text{pathstart } g$ *pathfinish* $h = \text{pathstart } h$

and *no*: $\bigwedge t x. [t \in \{0..1\}; x \notin S] \implies \text{norm}(h\ t - g\ t) < \text{norm}(g\ t - x)$

shows *homotopic_loops* S g h

using *homotopic_loops_linear* [OF §] **by** (*metis linorder_not_le no_norm_minus_commute segment_bound1 subsetI*)

lemma *homotopic_nearby_paths*:

fixes $g\ h :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

assumes *path* g *open* S *path_image* $g \subseteq S$

shows $\exists e. 0 < e \wedge$

$(\forall h. \text{path } h \wedge$

$\text{pathstart } h = \text{pathstart } g \wedge \text{pathfinish } h = \text{pathfinish } g \wedge$

$(\forall t \in \{0..1\}. \text{norm}(h\ t - g\ t) < e) \longrightarrow \text{homotopic_paths } S\ g\ h)$

proof –

obtain e **where** $e > 0$ **and** $e: \bigwedge x\ y. x \in \text{path_image } g \implies y \in -S \implies e \leq \text{dist } x\ y$

using *separate_compact_closed* [of *path_image* $g - S$] **assms** **by** *force*

show *?thesis*

using e [*unfolded dist_norm*] $\langle e > 0 \rangle$

by (*fastforce simp: path_image_def intro!: homotopic_paths_nearby_explicit assms exI*)

qed

lemma *homotopic_nearby_loops*:

fixes $g\ h :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

assumes *path* g *open* S *path_image* $g \subseteq S$ *pathfinish* $g = \text{pathstart } g$

shows $\exists e. 0 < e \wedge$

$(\forall h. \text{path } h \wedge \text{pathfinish } h = \text{pathstart } h \wedge$

$(\forall t \in \{0..1\}. \text{norm}(h\ t - g\ t) < e) \longrightarrow \text{homotopic_loops } S\ g\ h)$

proof –

obtain e **where** $e > 0$ **and** $e: \bigwedge x\ y. x \in \text{path_image } g \implies y \in -S \implies e \leq \text{dist } x\ y$

using *separate_compact_closed* [of *path_image* $g - S$] **assms** **by** *force*

show *?thesis*

using e [*unfolded dist_norm*] $\langle e > 0 \rangle$

by (*fastforce simp: path_image_def intro!: homotopic_loops_nearby_explicit assms exI*)

qed

6.4.9 Homotopy and subpaths

lemma *homotopic_join_subpaths1*:

assumes *path* g **and** *pag: path_image* $g \subseteq S$

and $u: u \in \{0..1\}$ **and** $v: v \in \{0..1\}$ **and** $w: w \in \{0..1\}$ $u \leq v \leq w$

```

    shows homotopic_paths S (subpath u v g +++ subpath v w g) (subpath u w g)
  proof -
    have 1:  $t * 2 \leq 1 \implies u + t * (v * 2) \leq v + t * (u * 2)$  for t
      using affine_ineq ⟨u ≤ v⟩ by fastforce
    have 2:  $t * 2 > 1 \implies u + (2*t - 1) * v \leq v + (2*t - 1) * w$  for t
      by (metis add_mono_thms_linordered_semiring(1) diff_gt_0_iff_gt less_eq_real_def
      mult.commute mult_right_mono ⟨u ≤ v⟩ ⟨v ≤ w⟩)
    have t2:  $\bigwedge t::real. t*2 = 1 \implies t = 1/2$  by auto
    have homotopic_paths (path_image g) (subpath u v g +++ subpath v w g)
      (subpath u w g)
    proof (cases w = u)
      case True
      then show ?thesis
        by (metis ⟨path g⟩ homotopic_paths_rinv path_image_subpath_subset path_subpath
        pathstart_subpath reversepath_subpath subpath_refl u v)
      next
      case False
      let ?f =  $\lambda t. \text{if } t \leq 1/2 \text{ then } \text{inverse}((w - u)) *_R (2 * (v - u)) *_R t$ 
        else  $\text{inverse}((w - u)) *_R ((v - u) + (w - v)) *_R (2 *_R t$ 
        - 1))
      show ?thesis
      proof (rule homotopic_paths_sym [OF homotopic_paths_reparametrize [where
      f = ?f]])
        show path (subpath u w g)
          using assms(1) path_subpath u w(1) by blast
        show path_image (subpath u w g)  $\subseteq$  path_image g
          by (meson path_image_subpath_subset u w(1))
        show continuous_on {0..1} ?f
          unfolding split_01
          by (rule continuous_on_cases continuous_intros | force simp: pathfinish_def
          joinpaths_def dest!: t2)+
        show ?f  $\in$  {0..1}  $\rightarrow$  {0..1}
          using False assms
          by (force simp: field_simps not_le mult_left_mono affine_ineq dest!: 1 2)
        show (subpath u v g +++ subpath v w g) t = subpath u w g (?f t) if t  $\in$ 
          {0..1} for t
          using assms
          unfolding joinpaths_def subpath_def by (auto simp: divide_simps add.commute
          mult.commute mult.left_commute)
        qed (use False in auto)
      qed
    then show ?thesis
      by (rule homotopic_paths_subset [OF _ pag])
    qed

```

lemma homotopic_join_subpaths2:

```

  assumes homotopic_paths S (subpath u v g +++ subpath v w g) (subpath u w g)
  shows homotopic_paths S (subpath w v g +++ subpath v u g) (subpath w u g)
  by (metis assms homotopic_paths_reversepath_D pathfinish_subpath pathstart_subpath

```

reversepath_joinpaths reversepath_subpath)

lemma *homotopic_join_subpaths3*:

assumes *hom*: *homotopic_paths* S (*subpath* u v g +++ *subpath* v w g) (*subpath* u w g)
and *path* g **and** *pag*: *path_image* $g \subseteq S$
and $u \in \{0..1\}$ **and** $v \in \{0..1\}$ **and** $w \in \{0..1\}$
shows *homotopic_paths* S (*subpath* v w g +++ *subpath* w u g) (*subpath* v u g)
proof –
obtain *wvg*: *path* (*subpath* w v g) *path_image* (*subpath* w v g) $\subseteq S$
and *wug*: *path* (*subpath* w u g) *path_image* (*subpath* w u g) $\subseteq S$
and *vug*: *path* (*subpath* v u g) *path_image* (*subpath* v u g) $\subseteq S$
by (*meson* \langle *path* $g\rangle$ *pag* *path_image_subpath_subset_path_subpath_subset_trans* u v w)
have *homotopic_paths* S (*subpath* u w g +++ *subpath* w v g)
 $(($ *subpath* u v g +++ *subpath* v w g) +++ *subpath* w v g)
by (*simp* *add*: *hom* *homotopic_paths_join* *homotopic_paths_sym* *wvg*)
also have *homotopic_paths* S ... (*subpath* u v g +++ *subpath* v w g +++ *subpath* w v g)
using *wvg* *vug* \langle *path* $g\rangle$
by (*metis* *homotopic_paths_assoc* *homotopic_paths_sym* *path_image_subpath_commute* *path_subpath*
pathfinish_subpath_pathstart_subpath u v w)
also have *homotopic_paths* S ... (*subpath* u v g +++ *linepath* (*pathfinish* (*subpath* u v g)) (*pathfinish* (*subpath* u v g)))
using *wvg* *vug* \langle *path* $g\rangle$
by (*metis* *homotopic_paths_join* *homotopic_paths_linv* *homotopic_paths_refl* *path_image_subpath_commute*
path_subpath_pathfinish_subpath_pathstart_join_pathstart_subpath_reversepath_subpath u v)
also have *homotopic_paths* S ... (*subpath* u v g)
using *vug* \langle *path* $g\rangle$ **by** (*metis* *homotopic_paths_rid* *path_image_subpath_commute* *path_subpath* u v)
finally have *homotopic_paths* S (*subpath* u w g +++ *subpath* w v g) (*subpath* u v g) .
then show *?thesis*
using *homotopic_join_subpaths2* **by** *blast*
qed

proposition *homotopic_join_subpaths*:

\llbracket *path* g ; *path_image* $g \subseteq S$; $u \in \{0..1\}$; $v \in \{0..1\}$; $w \in \{0..1\}$ \rrbracket
 \implies *homotopic_paths* S (*subpath* u v g +++ *subpath* v w g) (*subpath* u w g)
by (*smt* (*verit*, *del_insts*) *homotopic_join_subpaths1* *homotopic_join_subpaths2* *homotopic_join_subpaths3*)

Relating homotopy of trivial loops to path-connectedness.

lemma *path_component_imp_homotopic_points*:

assumes *path_component* S a b
shows *homotopic_loops* S (*linepath* a a) (*linepath* b b)

proof –

```

obtain  $g :: \text{real} \Rightarrow 'a$  where  $g: \text{continuous\_on } \{0..1\} \ g \ g \in \{0..1\} \rightarrow S \ g \ 0 =$ 
 $a \ g \ 1 = b$ 
using assms by (auto simp: path_defs)
then have  $\text{continuous\_on } (\{0..1\} \times \{0..1\}) \ (g \circ \text{fst})$ 
by (fastforce intro!: continuous_intros)+
with  $g$  show ?thesis
by (auto simp: homotopic_loops_def homotopic_with_def path_defs Pi_iff)
qed

```

lemma *homotopic_loops_imp_path_component_value*:

```

 $\llbracket \text{homotopic\_loops } S \ p \ q; \ 0 \leq t; \ t \leq 1 \rrbracket \Longrightarrow \text{path\_component } S \ (p \ t) \ (q \ t)$ 
apply (clarsimp simp: homotopic_loops_def homotopic_with_def path_defs)
apply (rule_tac x=h \circ (\lambda u. (u, t)) in exI)
apply (fastforce elim!: continuous_on_subset intro!: continuous_intros)
done

```

lemma *homotopic_points_eq_path_component*:

```

 $\text{homotopic\_loops } S \ (\text{linepath } a \ a) \ (\text{linepath } b \ b) \longleftrightarrow \text{path\_component } S \ a \ b$ 
using homotopic_loops_imp_path_component_value path_component_imp_homotopic_points
by fastforce

```

lemma *path_connected_eq_homotopic_points*:

```

 $\text{path\_connected } S \longleftrightarrow$ 
 $(\forall a \ b. \ a \in S \wedge b \in S \longrightarrow \text{homotopic\_loops } S \ (\text{linepath } a \ a) \ (\text{linepath } b \ b))$ 
by (auto simp: path_connected_def path_component_def homotopic_points_eq_path_component)

```

6.4.10 Simply connected sets

defined as "all loops are homotopic (as loops)"

definition *simply_connected* **where**

```

 $\text{simply\_connected } S \equiv$ 
 $\forall p \ q. \ \text{path } p \wedge \text{pathfinish } p = \text{pathstart } p \wedge \text{path\_image } p \subseteq S \wedge$ 
 $\text{path } q \wedge \text{pathfinish } q = \text{pathstart } q \wedge \text{path\_image } q \subseteq S$ 
 $\longrightarrow \text{homotopic\_loops } S \ p \ q$ 

```

lemma *simply_connected_empty [iff]*: *simply_connected* $\{\}$

```

by (simp add: simply_connected_def)

```

lemma *simply_connected_imp_path_connected*:

```

fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
shows  $\text{simply\_connected } S \Longrightarrow \text{path\_connected } S$ 
by (simp add: simply_connected_def path_connected_eq_homotopic_points)

```

lemma *simply_connected_imp_connected*:

```

fixes  $S :: \_ :: \text{real\_normed\_vector\_set}$ 
shows  $\text{simply\_connected } S \Longrightarrow \text{connected } S$ 
by (simp add: path_connected_imp_connected simply_connected_imp_path_connected)

```


lemma *simply_connected_eq_contractible_loop_any*:

fixes $S :: _ :: \text{real_normed_vector_set}$

shows $\text{simply_connected } S \longleftrightarrow$

$(\forall p \ a. \text{path } p \wedge \text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p \wedge a \in S$
 $\longrightarrow \text{homotopic_loops } S \ p \ (\text{linepath } a \ a))$

(**is** $?lhs = ?rhs$)

proof

assume $?rhs$ **then show** $?lhs$

unfolding *simply_connected_def*

by (*metis pathfinish_in_path_image subsetD homotopic_loops_trans homotopic_loops_sym*)

qed (*force simp: simply_connected_def*)

lemma *simply_connected_eq_contractible_loop_some*:

fixes $S :: _ :: \text{real_normed_vector_set}$

shows $\text{simply_connected } S \longleftrightarrow$

$\text{path_connected } S \wedge$
 $(\forall p. \text{path } p \wedge \text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$
 $\longrightarrow (\exists a. a \in S \wedge \text{homotopic_loops } S \ p \ (\text{linepath } a \ a)))$

(**is** $?lhs = ?rhs$)

proof

assume $?lhs$

then show $?rhs$

using *simply_connected_eq_contractible_loop_any* **by** (*blast intro: simply_connected_imp_path_connected*)

next

assume $?rhs$

then show $?lhs$

by (*meson homotopic_loops_trans path_connected_eq_homotopic_points simply_connected_eq_contractible_loop_any*)

qed

lemma *simply_connected_eq_contractible_loop_all*:

fixes $S :: _ :: \text{real_normed_vector_set}$

shows $\text{simply_connected } S \longleftrightarrow$

$S = \{\}$ \vee
 $(\exists a \in S. \forall p. \text{path } p \wedge \text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$
 $\longrightarrow \text{homotopic_loops } S \ p \ (\text{linepath } a \ a))$

by (*meson ex_in_conv homotopic_loops_sym homotopic_loops_trans simply_connected_def simply_connected_eq_contractible_loop_any*)

lemma *simply_connected_eq_contractible_path*:

fixes $S :: _ :: \text{real_normed_vector_set}$

shows $\text{simply_connected } S \longleftrightarrow$

$\text{path_connected } S \wedge$
 $(\forall p. \text{path } p \wedge \text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p$
 $\longrightarrow \text{homotopic_paths } S \ p \ (\text{linepath } (\text{pathstart } p) \ (\text{pathstart } p)))$

(**is** $?lhs = ?rhs$)

proof

assume $?lhs$

1110

```

then show ?rhs
  unfolding simply_connected_imp_path_connected
  by (metis simply_connected_eq_contractible_loop_some homotopic_loops_imp_homotopic_paths_n)
next
  assume ?rhs
  then show ?lhs
    using homotopic_paths_imp_homotopic_loops simply_connected_eq_contractible_loop_some
by fastforce
qed

```

lemma simply_connected_eq_homotopic_paths:

fixes $S :: _::\text{real_normed_vector_set}$

shows simply_connected $S \iff$

$$\begin{aligned}
 & \text{path_connected } S \wedge \\
 & (\forall p \ q. \text{path } p \wedge \text{path_image } p \subseteq S \wedge \\
 & \quad \text{path } q \wedge \text{path_image } q \subseteq S \wedge \\
 & \quad \text{pathstart } q = \text{pathstart } p \wedge \text{pathfinish } q = \text{pathfinish } p \\
 & \quad \longrightarrow \text{homotopic_paths } S \ p \ q) \\
 & \text{(is } ?lhs = ?rhs)
 \end{aligned}$$

proof

assume ?lhs

then have pc: path_connected S

and *: $\bigwedge p. [\text{path } p; \text{path_image } p \subseteq S; \text{pathfinish } p = \text{pathstart } p] \implies \text{homotopic_paths } S \ p \ (\text{linepath } (\text{pathstart } p) \ (\text{pathstart } p))$

by (auto simp: simply_connected_eq_contractible_path)

have homotopic_paths $S \ p \ q$

if path p path_image $p \subseteq S$ path q
 path_image $q \subseteq S$ pathstart $q = \text{pathstart } p$
 pathfinish $q = \text{pathfinish } p$ **for** $p \ q$

proof -

have homotopic_paths $S \ p \ (p \ +++ \ \text{reversepath } q \ \+++ \ q)$

using that

by (smt (verit, best) homotopic_paths_join homotopic_paths_linv homotopic_paths_rid homotopic_paths_sym homotopic_paths_trans pathstart_linepath)

also have homotopic_paths $S \ \dots \ ((p \ +++ \ \text{reversepath } q) \ \+++ \ q)$

by (simp add: that homotopic_paths_assoc)

also have homotopic_paths $S \ \dots \ (\text{linepath } (\text{pathstart } q) \ (\text{pathstart } q) \ \+++ \ q)$

using * [of $p \ \+++ \ \text{reversepath } q$] that

by (simp add: homotopic_paths_assoc homotopic_paths_join path_image_join)

also have homotopic_paths $S \ \dots \ q$

using that homotopic_paths_lid **by** blast

finally show ?thesis .

qed

then show ?rhs

by (blast intro: pc *)

next

assume ?rhs

```

then show ?lhs
  by (force simp: simply_connected_eq_contractible_path)
qed

proposition simply_connected_Times:
  fixes  $S :: 'a::real\_normed\_vector\ set$  and  $T :: 'b::real\_normed\_vector\ set$ 
  assumes  $S$ : simply_connected  $S$  and  $T$ : simply_connected  $T$ 
  shows simply_connected( $S \times T$ )
proof -
  have homotopic_loops ( $S \times T$ )  $p$  (linepath ( $a, b$ ) ( $a, b$ ))
    if path  $p$  path_image  $p \subseteq S \times T$   $p\ 1 = p\ 0$   $a \in S$   $b \in T$ 
    for  $p\ a\ b$ 
  proof -
    have path (fst  $\circ p$ )
      by (simp add: continuous_on_fst Path_Connected.path_continuous_image
[OF  $\langle path\ p \rangle$ ])
    moreover have path_image (fst  $\circ p$ )  $\subseteq S$ 
      using that by (force simp: path_image_def)
    ultimately have  $p1$ : homotopic_loops  $S$  (fst  $\circ p$ ) (linepath  $a\ a$ )
      using  $S$  that
      by (simp add: simply_connected_eq_contractible_loop_any_pathfinish_def
pathstart_def)
    have path (snd  $\circ p$ )
      by (simp add: continuous_on_snd Path_Connected.path_continuous_image
[OF  $\langle path\ p \rangle$ ])
    moreover have path_image (snd  $\circ p$ )  $\subseteq T$ 
      using that by (force simp: path_image_def)
    ultimately have  $p2$ : homotopic_loops  $T$  (snd  $\circ p$ ) (linepath  $b\ b$ )
      using  $T$  that
      by (simp add: simply_connected_eq_contractible_loop_any_pathfinish_def
pathstart_def)
    show ?thesis
      using  $p1\ p2$  unfolding homotopic_loops
      apply clarify
      subgoal for  $h\ k$ 
        by (rule_tac  $x=\lambda z. (h\ z, k\ z)$  in exI) (force intro: continuous_intros simp:
path_defs)
      done
    qed
  with assms show ?thesis
  by (simp add: simply_connected_eq_contractible_loop_any_pathfinish_def path-
start_def)
qed

```

6.4.11 Contractible sets

definition contractible **where**

$contractible\ S \equiv \exists a. homotopic_with_canon\ (\lambda x. True)\ S\ S\ id\ (\lambda x. a)$

proposition *contractible_imp_simply_connected*:
fixes $S :: _ :: \text{real_normed_vector_set}$
assumes *contractible S* **shows** *simply_connected S*
proof (*cases S = {}*)
case *True* **then show** *?thesis* **by force**
next
case *False*
obtain a **where** $a: \text{homotopic_with_canon } (\lambda x. \text{True}) S S \text{ id } (\lambda x. a)$
using *assms* **by** (*force simp: contractible_def*)
then have $a \in S$
using *False homotopic_with_imp_funspace2* **by fastforce**
have $\forall p. \text{path } p \wedge$
 $\text{path_image } p \subseteq S \wedge \text{pathfinish } p = \text{pathstart } p \longrightarrow$
 $\text{homotopic_loops } S p (\text{linepath } a a)$
using a **apply** (*clarsimp simp: homotopic_loops_def homotopic_with_def*
path_defs)
apply (*rule_tac x=(h o ($\lambda y. (\text{fst } y, (p \circ \text{snd } y)))$) in exI*)
apply (*intro conjI continuous_on_compose continuous_intros; force elim: con-*
tinuous_on_subset)
done
with $\langle a \in S \rangle$ **show** *?thesis*
by (*auto simp: simply_connected_eq_contractible_loop_all False*)
qed

corollary *contractible_imp_connected*:
fixes $S :: _ :: \text{real_normed_vector_set}$
shows *contractible S* \implies *connected S*
by (*simp add: contractible_imp_simply_connected simply_connected_imp_connected*)

lemma *contractible_imp_path_connected*:
fixes $S :: _ :: \text{real_normed_vector_set}$
shows *contractible S* \implies *path_connected S*
by (*simp add: contractible_imp_simply_connected simply_connected_imp_path_connected*)

lemma *nullhomotopic_through_contractible*:
fixes $S :: _ :: \text{topological_space_set}$
assumes $f: \text{continuous_on } S f f \in S \rightarrow T$
and $g: \text{continuous_on } T g g \in T \rightarrow U$
and $T: \text{contractible } T$
obtains c **where** *homotopic_with_canon* $(\lambda h. \text{True}) S U (g \circ f) (\lambda x. c)$
proof –
obtain b **where** $b: \text{homotopic_with_canon } (\lambda x. \text{True}) T T \text{ id } (\lambda x. b)$
using *assms* **by** (*force simp: contractible_def*)
have *homotopic_with_canon* $(\lambda f. \text{True}) T U (g \circ \text{id}) (g \circ (\lambda x. b))$
by (*metis b continuous_map_subtopology_eu g homotopic_with_compose_continuous_map_left*
image_subset_iff_funcset)
then have *homotopic_with_canon* $(\lambda f. \text{True}) S U (g \circ \text{id} \circ f) (g \circ (\lambda x. b) \circ f)$
by (*simp add: f homotopic_with_compose_continuous_map_right image_subset_iff_funcset*)
then show *?thesis*

by (simp add: comp_def that)
qed

lemma *nullhomotopic_into_contractible*:
assumes *f*: continuous_on *S* *f* *f* ∈ *S* → *T*
and *T*: contractible *T*
obtains *c* **where** *homotopic_with_canon* (λh . True) *S* *T* *f* (λx . *c*)
by (rule *nullhomotopic_through_contractible* [OF *f*, of *id* *T*]) (use *assms* **in** *auto*)

lemma *nullhomotopic_from_contractible*:
assumes *f*: continuous_on *S* *f* *f* ∈ *S* → *T*
and *S*: contractible *S*
obtains *c* **where** *homotopic_with_canon* (λh . True) *S* *T* *f* (λx . *c*)
by (*auto* simp: comp_def intro: *nullhomotopic_through_contractible* [OF *continuous_on_id* *f* *S*])

lemma *homotopic_through_contractible*:
fixes *S* :: *real_normed_vector* set
assumes continuous_on *S* *f1* *f1* ∈ *S* → *T*
continuous_on *T* *g1* *g1* ∈ *T* → *U*
continuous_on *S* *f2* *f2* ∈ *S* → *T*
continuous_on *T* *g2* *g2* ∈ *T* → *U*
contractible *T* *path_connected* *U*
shows *homotopic_with_canon* (λh . True) *S* *U* (*g1* ∘ *f1*) (*g2* ∘ *f2*)
proof –
obtain *c1* **where** *c1*: *homotopic_with_canon* (λh . True) *S* *U* (*g1* ∘ *f1*) (λx . *c1*)
by (rule *nullhomotopic_through_contractible* [of *S* *f1* *T* *g1* *U*]) (use *assms* **in** *auto*)
obtain *c2* **where** *c2*: *homotopic_with_canon* (λh . True) *S* *U* (*g2* ∘ *f2*) (λx . *c2*)
by (rule *nullhomotopic_through_contractible* [of *S* *f2* *T* *g2* *U*]) (use *assms* **in** *auto*)
have *S* = {} ∨ (∃ *t*. *path_connected* *t* ∧ *t* ⊆ *U* ∧ *c2* ∈ *t* ∧ *c1* ∈ *t*)
proof (*cases* *S* = {})
case True **then show** ?*thesis* **by** force
next
case False
with *c1* *c2* **have** *c1* ∈ *U* *c2* ∈ *U*
using *homotopic_with_imp_continuous_maps*
by (*metis* *PiE* *equalsOI* *homotopic_with_imp_funspace2*) +
with ⟨*path_connected* *U*⟩ **show** ?*thesis* **by** blast
qed
then have *homotopic_with_canon* (λh . True) *S* *U* (λx . *c2*) (λx . *c1*)
by (*auto* simp: *path_component_homotopic_constant_maps*)
then show ?*thesis*
using *c1* *c2* *homotopic_with_symD* *homotopic_with_trans* **by** blast
qed

lemma *homotopic_into_contractible*:
fixes *S* :: '*a*::*real_normed_vector* set **and** *T*:: '*b*::*real_normed_vector* set

assumes f : *continuous_on* S f $f \in S \rightarrow T$
and g : *continuous_on* S g $g \in S \rightarrow T$
and T : *contractible* T
shows *homotopic_with_canon* $(\lambda h. \text{True})$ S T f g
using *homotopic_through_contractible* [*of* S f T *id* T g *id*]
by (*simp* *add*: *assms contractible_imp_path_connected*)

lemma *homotopic_from_contractible*:

fixes $S :: 'a::\text{real_normed_vector_set}$ **and** $T :: 'b::\text{real_normed_vector_set}$
assumes f : *continuous_on* S f $f \in S \rightarrow T$
and g : *continuous_on* S g $g \in S \rightarrow T$
and *contractible* S *path_connected* T
shows *homotopic_with_canon* $(\lambda h. \text{True})$ S T f g
using *homotopic_through_contractible* [*of* S *id* S f T *id* g]
by (*simp* *add*: *assms contractible_imp_path_connected*)

6.4.12 Starlike sets

definition *starlike* $S \longleftrightarrow (\exists a \in S. \forall x \in S. \text{closed_segment } a \ x \subseteq S)$

lemma *starlike_UNIV* [*simp*]: *starlike* $UNIV$
by (*simp* *add*: *starlike_def*)

lemma *convex_imp_starlike*:

convex $S \implies S \neq \{\}$ \implies *starlike* S
unfolding *convex_contains_segment* *starlike_def* **by** *auto*

lemma *starlike_convex_tweak_boundary_points*:

fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes *convex* S $S \neq \{\}$ **and** ST : *rel_interior* $S \subseteq T$ **and** TS : $T \subseteq \text{closure } S$
shows *starlike* T

proof –

have *rel_interior* $S \neq \{\}$
by (*simp* *add*: *assms rel_interior_eq_empty*)
with ST **obtain** a **where** $a \in \text{rel_interior } S$ **and** $a \in T$ **by** *blast*
have $\bigwedge x. x \in T \implies \text{open_segment } a \ x \subseteq \text{rel_interior } S$
by (*rule* *rel_interior_closure_convex_segment* [*OF* $\langle \text{convex } S \rangle$ a]) (*use* *assms*
in *auto*)
then **have** $\forall x \in T. a \in T \wedge \text{open_segment } a \ x \subseteq T$
using ST **by** (*blast* *intro*: $a \in T$ *rel_interior_closure_convex_segment* [*OF*
 $\langle \text{convex } S \rangle$ a])
then **show** *thesis*
unfolding *starlike_def* **using** *beqI* [*OF* $\langle a \in T \rangle$]
by (*simp* *add*: *closed_segment_eq_open*)

qed

lemma *starlike_imp_contractible_gen*:

fixes $S :: 'a::\text{real_normed_vector_set}$
assumes S : *starlike* S

and $P: \bigwedge a T. \llbracket a \in S; 0 \leq T; T \leq 1 \rrbracket \implies P(\lambda x. (1 - T) *_R x + T *_R a)$
obtains a **where** $\text{homotopic_with_canon } P \ S \ S \ (\lambda x. x) \ (\lambda x. a)$
proof –
obtain a **where** $a \in S$ **and** $a: \bigwedge x. x \in S \implies \text{closed_segment } a \ x \subseteq S$
using S **by** $(\text{auto simp: starlike_def})$
have $\bigwedge t b. 0 \leq t \wedge t \leq 1 \implies$
 $\quad \exists u. (1 - t) *_R b + t *_R a = (1 - u) *_R a + u *_R b \wedge 0 \leq u \wedge u \leq 1$
by $(\text{metis add_diff_cancel_right' diff_ge_0_iff_ge_le_add_diff_inverse pth_c(1)})$
then have $(\lambda y. (1 - \text{fst } y) *_R \text{snd } y + \text{fst } y *_R a) ' (\{0..1\} \times S) \subseteq S$
using a $[\text{unfolded closed_segment_def}]$ **by force**
then have $\text{homotopic_with_canon } P \ S \ S \ (\lambda x. x) \ (\lambda x. a)$
using $\langle a \in S \rangle$
unfolding $\text{homotopic_with_def}$
apply $(\text{rule_tac } x = \lambda y. (1 - (\text{fst } y)) *_R \text{snd } y + (\text{fst } y) *_R a$ **in** $\text{exI})$
apply $(\text{force simp: } P \ \text{intro: continuous_intros})$
done
then show $?thesis$
using $that$ **by blast**
qed

lemma $\text{starlike_imp_contractible}$:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{starlike } S \implies \text{contractible } S$
using $\text{starlike_imp_contractible_gen contractible_def}$ **by** $(\text{fastforce simp: id_def})$

lemma contractible_UNIV $[\text{simp}]$: $\text{contractible } (\text{UNIV} :: 'a::\text{real_normed_vector_set})$
by $(\text{simp add: starlike_imp_contractible})$

lemma $\text{starlike_imp_simply_connected}$:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{starlike } S \implies \text{simply_connected } S$
by $(\text{simp add: contractible_imp_simply_connected starlike_imp_contractible})$

lemma $\text{convex_imp_simply_connected}$:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{convex } S \implies \text{simply_connected } S$
using $\text{convex_imp_starlike starlike_imp_simply_connected}$ **by blast**

lemma $\text{starlike_imp_path_connected}$:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{starlike } S \implies \text{path_connected } S$
by $(\text{simp add: simply_connected_imp_path_connected starlike_imp_simply_connected})$

lemma $\text{starlike_imp_connected}$:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{starlike } S \implies \text{connected } S$
by $(\text{simp add: path_connected_imp_connected starlike_imp_path_connected})$

lemma *is_intervalSimplyConnected_1*:
fixes $S :: \text{real set}$
shows $\text{is_interval } S \longleftrightarrow \text{simply_connected } S$
by (*meson convex_imp_simply_connected is_interval_connected_1 is_interval_convex_1 simply_connected_imp_connected*)

lemma *contractible_empty* [*simp*]: *contractible* $\{\}$
by (*simp add: contractible_def homotopic_on_emptyI*)

lemma *contractible_convex_tweak_boundary_points*:
fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{convex } S$ **and** $TS: \text{rel_interior } S \subseteq T \ T \subseteq \text{closure } S$
shows *contractible* T
by (*metis assms closure_eq_empty contractible_empty empty_subsetI starlike_convex_tweak_boundary_points starlike_imp_contractible subset_antisym*)

lemma *convex_imp_contractible*:
fixes $S :: 'a::\text{real_normed_vector set}$
shows $\text{convex } S \implies \text{contractible } S$
using *contractible_empty convex_imp_starlike starlike_imp_contractible* **by** *blast*

lemma *contractible_sing* [*simp*]:
fixes $a :: 'a::\text{real_normed_vector}$
shows *contractible* $\{a\}$
by (*rule convex_imp_contractible [OF convex_singleton]*)

lemma *is_intervalContractible_1*:
fixes $S :: \text{real set}$
shows $\text{is_interval } S \longleftrightarrow \text{contractible } S$
using *contractible_imp_simply_connected convex_imp_contractible is_interval_convex_1 is_interval_simply_connected_1* **by** *auto*

lemma *contractible_Times*:
fixes $S :: 'a::\text{euclidean_space set}$ **and** $T :: 'b::\text{euclidean_space set}$
assumes $S: \text{contractible } S$ **and** $T: \text{contractible } T$
shows *contractible* $(S \times T)$

proof –

obtain $a \ h$ **where** $\text{conth: continuous_on } (\{0..1\} \times S) \ h$
and $\text{hsub: } h \in (\{0..1\} \times S) \rightarrow S$
and [*simp*]: $\bigwedge x. x \in S \implies h(0, x) = x$
and [*simp*]: $\bigwedge x. x \in S \implies h(1::\text{real}, x) = a$
using S **by** (*force simp: contractible_def homotopic_with*)
obtain $b \ k$ **where** $\text{contk: continuous_on } (\{0..1\} \times T) \ k$
and $\text{ksub: } k \in (\{0..1\} \times T) \rightarrow T$
and [*simp*]: $\bigwedge x. x \in T \implies k(0, x) = x$
and [*simp*]: $\bigwedge x. x \in T \implies k(1::\text{real}, x) = b$
using T **by** (*force simp: contractible_def homotopic_with*)
show *?thesis*
apply (*simp add: contractible_def homotopic_with*)


```

apply (rule exI [where x=a])
apply (rule exI [where x=b])
apply (rule exI [where x =  $\lambda z. (h (fst z, fst (snd z)), k (fst z, snd (snd z)))$ ])
using hsub ksub
apply (fastforce intro!: continuous_intros continuous_on_compose2 [OF conth]
continuous_on_compose2 [OF contk])
done
qed

```

6.4.13 Local versions of topological properties in general

definition *locally* :: ('a::topological_space set \Rightarrow bool) \Rightarrow 'a set \Rightarrow bool

where

```

locally P S  $\equiv$ 
   $\forall w x. \text{openin } (\text{top\_of\_set } S) w \wedge x \in w$ 
     $\longrightarrow (\exists U V. \text{openin } (\text{top\_of\_set } S) U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge V$ 
 $\subseteq w)$ 

```

lemma *locallyI*:

```

assumes  $\bigwedge w x. \llbracket \text{openin } (\text{top\_of\_set } S) w; x \in w \rrbracket$ 
==>  $\exists U V. \text{openin } (\text{top\_of\_set } S) U \wedge P V \wedge x \in U \wedge U \subseteq V \wedge$ 
 $V \subseteq w$ 
shows locally P S
using assms by (force simp: locally_def)

```

lemma *locallyE*:

```

assumes locally P S  $\text{openin } (\text{top\_of\_set } S) w x \in w$ 
obtains U V where  $\text{openin } (\text{top\_of\_set } S) U P V x \in U U \subseteq V V \subseteq w$ 
using assms unfolding locally_def by meson

```

lemma *locally_mono*:

```

assumes locally P S  $\bigwedge T. P T \Longrightarrow Q T$ 
shows locally Q S
by (metis assms locally_def)

```

lemma *locally_open_subset*:

```

assumes locally P S  $\text{openin } (\text{top\_of\_set } S) t$ 
shows locally P t
by (smt (verit, ccfv_SIG) assms order.trans locally_def openin_imp_subset
openin_subset_trans openin_trans)

```

lemma *locally_diff_closed*:

```

 $\llbracket \text{locally } P S; \text{closedin } (\text{top\_of\_set } S) t \rrbracket \Longrightarrow \text{locally } P (S - t)$ 
using locally_open_subset closedin_def by fastforce

```

lemma *locally_empty* [iff]: *locally* P {}

by (simp add: *locally_def* *openin_subtopology*)

lemma *locally_singleton* [iff]:

```

fixes  $a :: 'a::metric\_space$ 
shows  $locally\ P\ \{a\} \longleftrightarrow P\ \{a\}$ 
proof  $-$ 
  have  $\forall x::real. \neg 0 < x \implies P\ \{a\}$ 
    using  $zero\_less\_one$  by  $blast$ 
  then show  $?thesis$ 
    unfolding  $locally\_def$ 
    by  $(auto\ simp:\ openin\_euclidean\_subtopology\_iff\ subset\_singleton\_iff\ conj\_disj\_distribR)$ 
qed

```

```

lemma  $locally\_iff:$ 
   $locally\ P\ S \longleftrightarrow$ 
   $(\forall T\ x. open\ T \wedge x \in S \cap T \longrightarrow (\exists U. open\ U \wedge (\exists V. P\ V \wedge x \in S \cap U \wedge$ 
   $S \cap U \subseteq V \wedge V \subseteq S \cap T)))$ 
  by  $(smt\ (verit)\ locally\_def\ openin\_open)$ 

```

```

lemma  $locally\_Int:$ 
  assumes  $S:\ locally\ P\ S$  and  $T:\ locally\ P\ T$ 
  and  $P:\ \bigwedge S\ T. P\ S \wedge P\ T \implies P(S \cap T)$ 
  shows  $locally\ P\ (S \cap T)$ 
  unfolding  $locally\_iff$ 
proof  $clarify$ 
  fix  $A\ x$ 
  assume  $open\ A\ x \in A\ x \in S\ x \in T$ 
  then obtain  $U1\ V1\ U2\ V2$ 
    where  $open\ U1\ P\ V1\ x \in S \cap U1\ S \cap U1 \subseteq V1 \wedge V1 \subseteq S \cap A$ 
       $open\ U2\ P\ V2\ x \in T \cap U2\ T \cap U2 \subseteq V2 \wedge V2 \subseteq T \cap A$ 
    using  $S\ T$  unfolding  $locally\_iff$  by  $(meson\ IntI)$ 
  then have  $S \cap T \cap (U1 \cap U2) \subseteq V1 \cap V2\ V1 \cap V2 \subseteq S \cap T \cap A\ x \in S \cap$ 
   $T \cap (U1 \cap U2)$ 
    by  $blast+$ 
  moreover have  $P\ (V1 \cap V2)$ 
    by  $(simp\ add:\ P\ \langle P\ V1 \rangle\ \langle P\ V2 \rangle)$ 
  ultimately show  $\exists U. open\ U \wedge (\exists V. P\ V \wedge x \in S \cap T \cap U \wedge S \cap T \cap U \subseteq$ 
   $V \wedge V \subseteq S \cap T \cap A)$ 
    using  $\langle open\ U1 \rangle\ \langle open\ U2 \rangle$  by  $blast$ 
qed

```

```

lemma  $locally\_Times:$ 
  fixes  $S :: ('a::metric\_space)\ set$  and  $T :: ('b::metric\_space)\ set$ 
  assumes  $PS:\ locally\ P\ S$  and  $QT:\ locally\ Q\ T$  and  $R:\ \bigwedge S\ T. P\ S \wedge Q\ T \implies$ 
   $R(S \times T)$ 
  shows  $locally\ R\ (S \times T)$ 
  unfolding  $locally\_def$ 
proof  $(clarify)$ 
  fix  $W\ x\ y$ 
  assume  $W:\ openin\ (top\_of\_set\ (S \times T))\ W$  and  $xy:\ (x, y) \in W$ 
  then obtain  $U\ V$  where  $openin\ (top\_of\_set\ S)\ U\ x \in U$ 

```

```

      openin (top_of_set T) V y ∈ V U × V ⊆ W
    using Times_in_interior_subtopology by metis
  then obtain U1 U2 V1 V2
    where opeS: openin (top_of_set S) U1 ∧ P U2 ∧ x ∈ U1 ∧ U1 ⊆ U2 ∧
    U2 ⊆ U
      and opeT: openin (top_of_set T) V1 ∧ Q V2 ∧ y ∈ V1 ∧ V1 ⊆ V2 ∧
    V2 ⊆ V
    by (meson PS QT locallyE)
  then have openin (top_of_set (S × T)) (U1 × V1)
    by (simp add: openin_Times)
  moreover have R (U2 × V2)
    by (simp add: R opeS opeT)
  moreover have U1 × V1 ⊆ U2 × V2 ∧ U2 × V2 ⊆ W
    using opeS opeT ⟨U × V ⊆ W⟩ by auto
  ultimately show ∃ U V. openin (top_of_set (S × T)) U ∧ R V ∧ (x,y) ∈ U
  ∧ U ⊆ V ∧ V ⊆ W
    using opeS opeT by auto
qed

```

proposition *homeomorphism_locally_imp*:

```

  fixes S :: 'a::metric_space set and T :: 'b::t2_space set
  assumes S: locally_P S and hom: homeomorphism S T f g
    and Q: ∧ S S'. [P S; homeomorphism S S' f g] ⇒ Q S'
  shows locally_Q T
proof (clarsimp simp: locally_def)
  fix W y
  assume y ∈ W and openin (top_of_set T) W
  then obtain A where T: open A W = T ∩ A
    by (force simp: openin_open)
  then have W ⊆ T by auto
  have f: ∧ x. x ∈ S ⇒ g(f x) = x f ' S = T continuous_on S f
    and g: ∧ y. y ∈ T ⇒ f(g y) = y g ' T = S continuous_on T g
    using hom by (auto simp: homeomorphism_def)
  have gw: g ' W = S ∩ f - ' W
    using ⟨W ⊆ T⟩ g by force
  have openin (top_of_set S) (g ' W)
    using ⟨openin (top_of_set T) W⟩ continuous_on_open f gw by auto
  then obtain U V
    where osu: openin (top_of_set S) U and uv: P V g y ∈ U U ⊆ V V ⊆ g ' W
    by (metis S ⟨y ∈ W⟩ image_eqI locallyE)
  have V ⊆ S using uv by (simp add: gw)
  have fv: f ' V = T ∩ {x. g x ∈ V}
    using ⟨f ' S = T⟩ f ⟨V ⊆ S⟩ by auto
  have contuf: continuous_on V f
    using ⟨V ⊆ S⟩ continuous_on_subset f(3) by blast
  have openin (top_of_set (g ' T)) U
    using ⟨g ' T = S⟩ by (simp add: osu)
  then have openin (top_of_set T) (T ∩ g - ' U)

```

```

    using ‹continuous_on T g› continuous_on_open [THEN iffD1] by blast
  moreover have  $\exists V. Q V \wedge y \in (T \cap g^{-1} U) \wedge (T \cap g^{-1} U) \subseteq V \wedge V \subseteq W$ 
  proof (intro exI conjI)
    show  $f^{-1} V \subseteq W$ 
    using uv using Int_lower2 gw image_subsetI mem_Collect_eq subset_iff by
  auto
    then have contvg: continuous_on (f^{-1} V) g
    using ‹W ⊆ T› continuous_on_subset [OF g(3)] by blast
    have  $V \subseteq g^{-1} f^{-1} V$ 
    by (metis ‹V ⊆ S› hom_homeomorphism_def homeomorphism_of_subsets
  order_refl)
    then have homv: homeomorphism V (f^{-1} V) f g
    using ‹V ⊆ S› f by (auto simp: homeomorphism_def contvf contvg)
    show Q (f^{-1} V)
    using Q homv ‹P V› by blast
    show  $y \in T \cap g^{-1} U$ 
    using T(2) ‹y ∈ W› ‹g y ∈ U› by blast
    show  $T \cap g^{-1} U \subseteq f^{-1} V$ 
    using g(1) image_iff uv(3) by fastforce
  qed
  ultimately show  $\exists U. \text{openin } (\text{top\_of\_set } T) U \wedge (\exists v. Q v \wedge y \in U \wedge U \subseteq v$ 
 $\wedge v \subseteq W)$ 
  by meson
  qed

```

lemma *homeomorphism_locally*:

```

  fixes f: 'a::metric_space ⇒ 'b::metric_space
  assumes homeomorphism S T f g
    and  $\bigwedge S T. \text{homeomorphism } S T f g \implies (P S \longleftrightarrow Q T)$ 
  shows locally P S  $\longleftrightarrow$  locally Q T
  by (smt (verit) assms homeomorphism_locally_imp homeomorphism_symD)

```

lemma *homeomorphic_locally*:

```

  fixes S: 'a::metric_space set and T: 'b::metric_space set
  assumes hom: S homeomorphic T
    and iff:  $\bigwedge X Y. X \text{ homeomorphic } Y \implies (P X \longleftrightarrow Q Y)$ 
  shows locally P S  $\longleftrightarrow$  locally Q T
  by (smt (verit, ccfv_SIG) hom homeomorphic_def homeomorphism_locally home-
  omorphism_locally_imp iff)

```

lemma *homeomorphic_local_compactness*:

```

  fixes S: 'a::metric_space set and T: 'b::metric_space set
  shows S homeomorphic T  $\implies$  locally compact S  $\longleftrightarrow$  locally compact T
  by (simp add: homeomorphic_compactness homeomorphic_locally)

```

lemma *locally_translation*:

```

  fixes P :: 'a :: real_normed_vector set ⇒ bool
  shows  $(\bigwedge S. P ((+) a^{-1} S) = P S) \implies$  locally P  $((+) a^{-1} S) =$  locally P S
  using homeomorphism_locally [OF homeomorphism_translation]

```

by (metis (full_types) homeomorphism_image2)

lemma *locally_injective_linear_image*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes f : *linear* f *inj* f **and** *iff*: $\bigwedge S. P (f \text{ ` } S) \longleftrightarrow Q S$

shows *locally* $P (f \text{ ` } S) \longleftrightarrow$ *locally* $Q S$

by (smt (verit) *f* *homeomorphism_image2* *homeomorphism_locally_iff_linear_homeomorphism_image*)

lemma *locally_open_map_image*:

fixes $f :: 'a::real_normed_vector \Rightarrow 'b::real_normed_vector$

assumes P : *locally* $P S$

and f : *continuous_on* $S f$

and *oo*: $\bigwedge T. \text{openin } (top_of_set S) T \Longrightarrow \text{openin } (top_of_set (f \text{ ` } S)) (f \text{ ` } T)$

and Q : $\bigwedge T. \llbracket T \subseteq S; P T \rrbracket \Longrightarrow Q(f \text{ ` } T)$

shows *locally* $Q (f \text{ ` } S)$

proof (*clarsimp simp: locally_def*)

fix $W y$

assume *oiw*: *openin* $(top_of_set (f \text{ ` } S)) W$ **and** $y \in W$

then have $W \subseteq f \text{ ` } S$ **by** (*simp add: openin_euclidean_subtopology_iff*)

have *oivf*: *openin* $(top_of_set S) (S \cap f \text{ ` } W)$

by (*rule continuous_on_open [THEN iffD1, rule_format, OF f oiw]*)

then obtain x **where** $x \in S$ $f x = y$

using $\langle W \subseteq f \text{ ` } S \rangle \langle y \in W \rangle$ **by** *blast*

then obtain $U V$

where *openin* $(top_of_set S) U$ $P V$ $x \in U$ $U \subseteq V$ $V \subseteq S \cap f \text{ ` } W$

by (*metis IntI P \langle y \in W \rangle locallyE oivf vimageI*)

then have *openin* $(top_of_set (f \text{ ` } S)) (f \text{ ` } U)$

by (*simp add: oo*)

then show $\exists X. \text{openin } (top_of_set (f \text{ ` } S)) X \wedge (\exists Y. Q Y \wedge y \in X \wedge X \subseteq Y \wedge Y \subseteq W)$

using $Q \langle P V \rangle \langle U \subseteq V \rangle \langle V \subseteq S \cap f \text{ ` } W \rangle \langle f x = y \rangle \langle x \in U \rangle$ **by** *blast*

qed

6.4.14 An induction principle for connected sets

proposition *connected_induction*:

assumes *connected* S

and *opD*: $\bigwedge T a. \llbracket \text{openin } (top_of_set S) T; a \in T \rrbracket \Longrightarrow \exists z. z \in T \wedge P z$

and *opI*: $\bigwedge a. a \in S$

$\Longrightarrow \exists T. \text{openin } (top_of_set S) T \wedge a \in T \wedge$

$(\forall x \in T. \forall y \in T. P x \wedge P y \wedge Q x \longrightarrow Q y)$

and *etc*: $a \in S$ $b \in S$ $P a$ $P b$ $Q a$

shows $Q b$

proof –

let $?A = \{b. \exists T. \text{openin } (top_of_set S) T \wedge b \in T \wedge (\forall x \in T. P x \longrightarrow Q x)\}$

let $?B = \{b. \exists T. \text{openin } (top_of_set S) T \wedge b \in T \wedge (\forall x \in T. P x \longrightarrow \neg Q x)\}$

have $?A \cap ?B = \{\}$

by (*clarsimp simp: set_eq_iff*) (*metis (no_types, opaque_lifting) Int_iff opD*)

openin_Int)
moreover have $S \subseteq ?A \cup ?B$
by *clarsimp (meson opI)*
moreover have *openin (top_of_set S) ?A*
by *(subst openin_subopen, blast)*
moreover have *openin (top_of_set S) ?B*
by *(subst openin_subopen, blast)*
ultimately have $?A = \{\} \vee ?B = \{\}$
by *(metis (no_types, lifting) ‹connected S› connected_openin)*
then show *?thesis*
by *clarsimp (meson opI etc)*
qed

lemma *connected_equivalence_relation_gen*:
assumes *connected S*
and *etc: a ∈ S b ∈ S P a P b*
and *trans: $\bigwedge x y z. \llbracket R x y; R y z \rrbracket \implies R x z$*
and *opD: $\bigwedge T a. \llbracket \text{openin (top_of_set S) } T; a \in T \rrbracket \implies \exists z. z \in T \wedge P z$*
and *opI: $\bigwedge a. a \in S \implies \exists T. \text{openin (top_of_set S) } T \wedge a \in T \wedge (\forall x \in T. \forall y \in T. P x \wedge P y \longrightarrow R x y)$*
shows $R a b$
proof –
have $\bigwedge a b c. \llbracket a \in S; P a; b \in S; c \in S; P b; P c; R a b \rrbracket \implies R a c$
apply *(rule connected_induction [OF ‹connected S› opD], simp_all)*
by *(meson trans opI)*
then show *?thesis by (metis etc opI)*
qed

lemma *connected_induction_simple*:
assumes *connected S*
and *etc: a ∈ S b ∈ S P a*
and *opI: $\bigwedge a. a \in S \implies \exists T. \text{openin (top_of_set S) } T \wedge a \in T \wedge (\forall x \in T. \forall y \in T. P x \longrightarrow P y)$*
shows $P b$
by *(rule connected_induction [OF ‹connected S› __, where P = $\lambda x. \text{True}$])*
(use opI etc in auto)

lemma *connected_equivalence_relation*:
assumes *connected S*
and *etc: a ∈ S b ∈ S*
and *sym: $\bigwedge x y. \llbracket R x y; x \in S; y \in S \rrbracket \implies R y x$*
and *trans: $\bigwedge x y z. \llbracket R x y; R y z; x \in S; y \in S; z \in S \rrbracket \implies R x z$*
and *opI: $\bigwedge a. a \in S \implies \exists T. \text{openin (top_of_set S) } T \wedge a \in T \wedge (\forall x \in T. R a x)$*
shows $R a b$
proof –
have $\bigwedge a b c. \llbracket a \in S; b \in S; c \in S; R a b \rrbracket \implies R a c$

```

  by (smt (verit, ccfv_threshold) connected_induction_simple [OF ‹connected
S›]
      assms openin_imp_subset subset_eq)
  then show ?thesis by (metis etc opI)
qed

```

lemma *locally_constant_imp_constant*:

```

  assumes connected S
  and opI:  $\bigwedge a. a \in S \implies \exists T. \text{openin } (\text{top\_of\_set } S) T \wedge a \in T \wedge (\forall x \in T. f x = f a)$ 
  shows f_constant_on S
proof -
  have  $\bigwedge x y. x \in S \implies y \in S \implies f x = f y$ 
  apply (rule connected_equivalence_relation [OF ‹connected S›], simp_all)
  by (metis opI)
  then show ?thesis
  by (metis constant_on_def)
qed

```

lemma *locally_constant*:

```

  assumes connected S
  shows locally  $(\lambda U. f \text{ constant\_on } U) S \iff f \text{ constant\_on } S$  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
  by (smt (verit, del_insts) assms constant_on_def locally_constant_imp_constant
      locally_def openin_subtopology_self subset_iff)
next
  assume ?rhs then show ?lhs
  by (metis constant_on_subset locallyI openin_imp_subset order_refl)
qed

```

6.4.15 Basic properties of local compactness

proposition *locally_compact*:

```

  fixes S :: 'a :: metric_space set
  shows
    locally_compact S  $\iff$ 
     $(\forall x \in S. \exists u v. x \in u \wedge u \subseteq v \wedge v \subseteq S \wedge$ 
       $\text{openin } (\text{top\_of\_set } S) u \wedge \text{compact } v)$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
  by (meson locallyE openin_subtopology_self)
next
  assume r [rule_format]: ?rhs
  have *:  $\exists u v.$ 
     $\text{openin } (\text{top\_of\_set } S) u \wedge$ 

```

```

      compact v ∧ x ∈ u ∧ u ⊆ v ∧ v ⊆ S ∩ T
    if open T x ∈ S x ∈ T for x T
  proof -
    obtain U V where uv: x ∈ U U ⊆ V V ⊆ S compact V openin (top_of_set
S) U
    using r [OF ⟨x ∈ S⟩] by auto
    obtain e where e>0 and e: cball x e ⊆ T
    using open_contains_cball ⟨open T⟩ ⟨x ∈ T⟩ by blast
    show ?thesis
    apply (rule_tac x=(S ∩ ball x e) ∩ U in exI)
    apply (rule_tac x=cball x e ∩ V in exI)
    using that ⟨e > 0⟩ e uv
    apply auto
    done
  qed
  show ?lhs
  by (rule locallyI) (metis * Int_iff openin_open)
qed

lemma locally_compactE:
  fixes S :: 'a :: metric_space set
  assumes locally_compact S
  obtains u v where ∧x. x ∈ S ⇒ x ∈ u x ∧ u x ⊆ v x ∧ v x ⊆ S ∧
    openin (top_of_set S) (u x) ∧ compact (v x)
  using assms unfolding locally_compact by metis

lemma locally_compact_alt:
  fixes S :: 'a :: heine_borel set
  shows locally_compact S ⟷
    (∀x ∈ S. ∃U. x ∈ U ∧
      openin (top_of_set S) U ∧ compact(closure U) ∧ closure U ⊆ S)
  by (smt (verit, ccfv_threshold) bounded_subset closure_closed closure_mono
closure_subset
  compact_closure compact_imp_closed order.trans locally_compact)

lemma locally_compact_Int_cball:
  fixes S :: 'a :: heine_borel set
  shows locally_compact S ⟷ (∀x ∈ S. ∃e. 0 < e ∧ closed(cball x e ∩ S))
    (is ?lhs = ?rhs)
  proof
    assume L: ?lhs
    then have ∧x U V e. [U ⊆ V; V ⊆ S; compact V; 0 < e; cball x e ∩ S ⊆ U]
      ⇒ closed (cball x e ∩ S)
      by (metis compact_Int compact_cball compact_imp_closed inf.absorb_iff2
inf.assoc inf.orderE)
    with L show ?rhs
      by (meson locally_compactE openin_contains_cball)
  next
    assume R: ?rhs

```



```

show ?lhs unfolding locally_compact
proof
  fix x
  assume  $x \in S$ 
  then obtain e where  $e > 0$  and compact (cball x e  $\cap S$ )
    by (metis Int_commute compact_Int_closed compact_cball inf.right_idem
R)
  moreover have  $\forall y \in \text{ball } x \ e \ \cap S. \exists \varepsilon > 0. \text{cball } y \ \varepsilon \ \cap S \subseteq \text{ball } x \ e$ 
    by (meson Elementary_Metric_Spaces.open_ball IntD1 le_infI1 open_contains_cball_eq)
  moreover have openin (top_of_set S) (ball x e  $\cap S$ )
    by (simp add: inf_commute openin_open_Int)
  ultimately show  $\exists U \ V. x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S)$ 
 $U \wedge \text{compact } V$ 
    by (metis Int_iff  $\langle 0 < e \rangle \langle x \in S \rangle$  ball_subset_cball centre_in_ball inf_commute
inf_le1 inf_mono order_refl)
  qed
qed

lemma locally_compact_compact:
fixes S :: 'a :: heine_borel set
shows locally_compact S  $\longleftrightarrow$ 
  ( $\forall K. K \subseteq S \wedge \text{compact } K$ 
 $\longrightarrow (\exists U \ V. K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge$ 
 $\text{openin } (\text{top\_of\_set } S) \ U \wedge \text{compact } V)$ )
  (is ?lhs = ?rhs)

proof
assume ?lhs
then obtain u v where
   $uv: \bigwedge x. x \in S \implies x \in u \ x \wedge u \ x \subseteq v \ x \wedge v \ x \subseteq S \wedge$ 
 $\text{openin } (\text{top\_of\_set } S) \ (u \ x) \wedge \text{compact } (v \ x)$ 
by (metis locally_compactE)
have *:  $\exists U \ V. K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) \ U \wedge$ 
compact V
  if  $K \subseteq S$  compact K for K
proof –
have  $\bigwedge C. (\forall c \in C. \text{openin } (\text{top\_of\_set } K) \ c) \wedge K \subseteq \bigcup C \implies$ 
 $\exists D \subseteq C. \text{finite } D \wedge K \subseteq \bigcup D$ 
using that by (simp add: compact_eq_openin_cover)
moreover have  $\forall c \in (\lambda x. K \cap u \ x) \text{ ' } K. \text{openin } (\text{top\_of\_set } K) \ c$ 
using that by clarify (metis subsetD inf.absorb_iff2 openin_subset openin_subtopology_Int_subset
topspace_euclidean_subtopology uv)
moreover have  $K \subseteq \bigcup ((\lambda x. K \cap u \ x) \text{ ' } K)$ 
using that by clarsimp (meson subsetCE uv)
ultimately obtain D where  $D \subseteq (\lambda x. K \cap u \ x) \text{ ' } K$  finite D  $K \subseteq \bigcup D$ 
by metis
then obtain T where  $T: T \subseteq K$  finite T  $K \subseteq \bigcup ((\lambda x. K \cap u \ x) \text{ ' } T)$ 
by (metis finite_subset_image)
have  $Tuv: \bigcup (u \text{ ' } T) \subseteq \bigcup (v \text{ ' } T)$ 
using T that by (force dest!: uv)

```

```

moreover
have openin (top_of_set  $S$ ) ( $\bigcup (u \text{ ' } T)$ )
  using  $T$  that  $uv$  by fastforce
moreover
obtain compact ( $\bigcup (v \text{ ' } T) \bigcup (v \text{ ' } T) \subseteq S$ )
  by (metis  $T$  UN_subset_iff  $\langle K \subseteq S \rangle$  compact_UN_subset_iff  $uv$ )
ultimately show ?thesis
  using  $T$  by auto
qed
show ?rhs
  by (blast intro: *)
next
assume ?rhs
then show ?lhs
  apply (clarsimp simp: locally_compact)
  apply (drule_tac  $x=\{x\}$  in spec, simp)
  done
qed

lemma open_imp_locally_compact:
  fixes  $S :: 'a :: \text{heine\_borel set}$ 
  assumes open  $S$ 
  shows locally compact  $S$ 
proof -
  have  $*$ :  $\exists U V. x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) U \wedge \text{compact } V$ 
    if  $x \in S$  for  $x$ 
  proof -
    obtain  $e$  where  $e > 0$  and  $e: \text{cball } x e \subseteq S$ 
      using open_contains_cball assms  $\langle x \in S \rangle$  by blast
    have ope: openin (top_of_set  $S$ ) (ball  $x e$ )
      by (meson  $e$  open_ball ball_subset_cball dual_order.trans open_subset)
    show ?thesis
      by (meson  $\langle 0 < e \rangle$  ball_subset_cball centre_in_ball compact_cball  $e$  ope)
  qed
show ?thesis
  unfolding locally_compact by (blast intro: *)
qed

lemma closed_imp_locally_compact:
  fixes  $S :: 'a :: \text{heine\_borel set}$ 
  assumes closed  $S$ 
  shows locally compact  $S$ 
proof -
  have  $*$ :  $\exists U V. x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge \text{openin } (\text{top\_of\_set } S) U \wedge \text{compact } V$ 
    if  $x \in S$  for  $x$ 
  apply (rule_tac  $x = S \cap \text{ball } x 1$  in exI, rule_tac  $x = S \cap \text{cball } x 1$  in exI)
  using  $\langle x \in S \rangle$  assms by auto

```

```

show ?thesis
  unfolding locally_compact by (blast intro: *)
qed

lemma locally_compact_UNIV: locally_compact (UNIV :: 'a :: heine_borel set)
  by (simp add: closed_imp_locally_compact)

lemma locally_compact_Int:
  fixes S :: 'a :: t2_space set
  shows  $\llbracket$ locally_compact S; locally_compact T $\rrbracket \implies$  locally_compact (S  $\cap$  T)
  by (simp add: compact_Int locally_Int)

lemma locally_compact_closedin:
  fixes S :: 'a :: heine_borel set
  shows  $\llbracket$ closedin (top_of_set S) T; locally_compact S $\rrbracket$ 
     $\implies$  locally_compact T
  unfolding closedin_closed
  using closed_imp_locally_compact locally_compact_Int by blast

lemma locally_compact_delete:
  fixes S :: 'a :: t1_space set
  shows locally_compact S  $\implies$  locally_compact (S - {a})
  by (auto simp: openin_delete locally_open_subset)

lemma locally_closed:
  fixes S :: 'a :: heine_borel set
  shows locally_closed S  $\longleftrightarrow$  locally_compact S
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding locally_def
    apply (elim_all_forward_imp_forward asm_rl exE)
    apply (rename_tac U V)
    apply (rule_tac x = U  $\cap$  ball x 1 in exI)
    apply (rule_tac x = V  $\cap$  cball x 1 in exI)
    apply (force intro: openin_trans)
    done
  next
  assume ?rhs then show ?lhs
    using compact_eq_bounded_closed locally_mono by blast
qed

lemma locally_compact_openin_Un:
  fixes S :: 'a::euclidean_space set
  assumes LCS: locally_compact S and LCT: locally_compact T
    and opS: openin (top_of_set (S  $\cup$  T)) S
    and opT: openin (top_of_set (S  $\cup$  T)) T
  shows locally_compact (S  $\cup$  T)

```

```

proof –
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \ \cap \ (S \cup T)) \text{ if } x \in S \text{ for } x$ 
  proof –
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \ \cap \ S)$ 
      using  $LCS \ \langle x \in S \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by  $\text{blast}$ 
    moreover obtain  $e2$  where  $e2 > 0$  and  $e2: \text{cball } x \ e2 \ \cap \ (S \cup T) \subseteq S$ 
      by  $(\text{meson } \langle x \in S \rangle \ \text{opS } \text{openin\_contains\_cball})$ 
    then have  $\text{cball } x \ e2 \ \cap \ (S \cup T) = \text{cball } x \ e2 \ \cap \ S$ 
      by  $\text{force}$ 
    ultimately have  $\text{closed } (\text{cball } x \ (\text{min } e1 \ e2) \ \cap \ (S \cup T))$ 
      by  $(\text{metis } (\text{no\_types, lifting}) \ \text{cball\_min\_Int } \text{closed\_Int } \text{closed\_cball } \text{inf\_assoc } \text{inf\_commute})$ 
    then show  $?thesis$ 
      by  $(\text{metis } \langle 0 < e1 \rangle \ \langle 0 < e2 \rangle \ \text{min\_def})$ 
  qed
  moreover have  $\exists e > 0. \text{closed } (\text{cball } x \ e \ \cap \ (S \cup T)) \text{ if } x \in T \text{ for } x$ 
  proof –
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \ \cap \ T)$ 
      using  $LCT \ \langle x \in T \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by  $\text{blast}$ 
    moreover obtain  $e2$  where  $e2 > 0$  and  $e2: \text{cball } x \ e2 \ \cap \ (S \cup T) \subseteq T$ 
      by  $(\text{meson } \langle x \in T \rangle \ \text{opT } \text{openin\_contains\_cball})$ 
    then have  $\text{cball } x \ e2 \ \cap \ (S \cup T) = \text{cball } x \ e2 \ \cap \ T$ 
      by  $\text{force}$ 
    moreover have  $\text{closed } (\text{cball } x \ e1 \ \cap \ (\text{cball } x \ e2 \ \cap \ T))$ 
      by  $(\text{metis } \text{closed\_Int } \text{closed\_cball } e1 \ \text{inf\_left\_commute})$ 
    ultimately show  $?thesis$ 
      by  $(\text{rule\_tac } x = \text{min } e1 \ e2 \ \text{in } \text{exI}) \ (\text{simp } \text{add: } \langle 0 < e2 \rangle \ \text{cball\_min\_Int } \text{inf\_assoc})$ 
  qed
  ultimately show  $?thesis$ 
    by  $(\text{force } \text{simp: } \text{locally\_compact\_Int\_cball})$ 
qed

```

lemma $\text{locally_compact_closedin_Un}$:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes  $LCS: \text{locally\_compact } S$  and  $LCT: \text{locally\_compact } T$ 
  and  $clS: \text{closedin } (\text{top\_of\_set } (S \cup T)) \ S$ 
  and  $clT: \text{closedin } (\text{top\_of\_set } (S \cup T)) \ T$ 
shows  $\text{locally\_compact } (S \cup T)$ 

```

```

proof –
  have  $\exists e > 0. \text{closed } (\text{cball } x \ e \ \cap \ (S \cup T)) \text{ if } x \in S \ x \in T \text{ for } x$ 
  proof –
    obtain  $e1$  where  $e1 > 0$  and  $e1: \text{closed } (\text{cball } x \ e1 \ \cap \ S)$ 
      using  $LCS \ \langle x \in S \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by  $\text{blast}$ 
    moreover
    obtain  $e2$  where  $e2 > 0$  and  $e2: \text{closed } (\text{cball } x \ e2 \ \cap \ T)$ 
      using  $LCT \ \langle x \in T \rangle$  unfolding  $\text{locally\_compact\_Int\_cball}$  by  $\text{blast}$ 
    moreover have  $\text{closed } (\text{cball } x \ (\text{min } e1 \ e2) \ \cap \ (S \cup T))$ 
      by  $(\text{smt } (\text{verit}) \ \text{Int\_Un\_distrib2 } \text{Int\_commute } \text{cball\_min\_Int } \text{closed\_Int})$ 
  qed

```

```

closed_Un closed_cball e1 e2 inf_left_commute)
  ultimately show ?thesis
    by (rule_tac x=min e1 e2 in exI) linarith
qed
moreover
have  $\exists e>0. \text{closed } (\text{cball } x \ e \ \cap \ (S \cup T)) \text{ if } x: x \in S \ x \notin T \text{ for } x$ 
proof -
  obtain e1 where e1 > 0 and e1: closed (cball x e1  $\cap$  S)
    using LCS  $\langle x \in S \rangle$  unfolding locally_compact_Int_cball by blast
  moreover
  obtain e2 where e2>0 and cball x e2  $\cap$  (S  $\cup$  T)  $\subseteq$  S - T
    using clT x by (fastforce simp: openin_contains_cball closedin_def)
  then have closed (cball x e2  $\cap$  T)
  proof -
    have {} = T - (T - cball x e2)
      using Diff_subset Int_Diff  $\langle \text{cball } x \ e2 \ \cap \ (S \cup T) \subseteq S - T \rangle$  by auto
    then show ?thesis
      by (simp add: Diff_Diff_Int inf_commute)
  qed
  with e1 have closed ((cball x e1  $\cap$  cball x e2)  $\cap$  (S  $\cup$  T))
    apply (simp add: inf_commute inf_sup_distrib2)
    by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
  then have closed (cball x (min e1 e2)  $\cap$  (S  $\cup$  T))
    by (simp add: cball_min_Int inf_commute)
  ultimately show ?thesis
    using  $\langle 0 < e2 \rangle$  by (rule_tac x=min e1 e2 in exI) linarith
qed
moreover
have  $\exists e>0. \text{closed } (\text{cball } x \ e \ \cap \ (S \cup T)) \text{ if } x: x \notin S \ x \in T \text{ for } x$ 
proof -
  obtain e1 where e1 > 0 and e1: closed (cball x e1  $\cap$  T)
    using LCT  $\langle x \in T \rangle$  unfolding locally_compact_Int_cball by blast
  moreover
  obtain e2 where e2>0 and cball x e2  $\cap$  (S  $\cup$  T)  $\subseteq$  S  $\cup$  T - S
    using clS x by (fastforce simp: openin_contains_cball closedin_def)
  then have closed (cball x e2  $\cap$  S)
    by (metis Diff_disjoint Int_empty_right closed_empty inf.left_commute
inf.orderE inf_sup_absorb)
  with e1 have closed ((cball x e1  $\cap$  cball x e2)  $\cap$  (S  $\cup$  T))
    apply (simp add: inf_commute inf_sup_distrib2)
    by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
  then have closed (cball x (min e1 e2)  $\cap$  (S  $\cup$  T))
    by (auto simp: cball_min_Int)
  ultimately show ?thesis
    using  $\langle 0 < e2 \rangle$  by (rule_tac x=min e1 e2 in exI) linarith
qed
ultimately show ?thesis
  by (auto simp: locally_compact_Int_cball)
qed

```

lemma *locally_compact_Times*:

fixes $S :: 'a::\text{euclidean_space set}$ **and** $T :: 'b::\text{euclidean_space set}$
shows $\llbracket \text{locally compact } S; \text{ locally compact } T \rrbracket \implies \text{locally compact } (S \times T)$
by (*auto simp: compact_Times locally_Times*)

lemma *locally_compact_compact_subopen*:

fixes $S :: 'a :: \text{heine_borel set}$
shows
 $\text{locally compact } S \longleftrightarrow$
 $(\forall K T. K \subseteq S \wedge \text{compact } K \wedge \text{open } T \wedge K \subseteq T$
 $\longrightarrow (\exists U V. K \subseteq U \wedge U \subseteq V \wedge U \subseteq T \wedge V \subseteq S \wedge$
 $\text{openin } (\text{top_of_set } S) U \wedge \text{compact } V))$
(is ?lhs = ?rhs)

proof

assume $L: ?lhs$

show $?rhs$

proof *clarify*

fix $K :: 'a \text{ set}$ **and** $T :: 'a \text{ set}$

assume $K \subseteq S$ **and** $\text{compact } K$ **and** $\text{open } T$ **and** $K \subseteq T$

obtain $U V$ **where** $K \subseteq U$ $U \subseteq V$ $V \subseteq S$ $\text{compact } V$

and $\text{ope: openin } (\text{top_of_set } S) U$

using L **unfolding** *locally_compact_compact* **by** (*meson* $\langle K \subseteq S \rangle \langle \text{compact } K \rangle$)

show $\exists U V. K \subseteq U \wedge U \subseteq V \wedge U \subseteq T \wedge V \subseteq S \wedge$
 $\text{openin } (\text{top_of_set } S) U \wedge \text{compact } V$

proof (*intro exI conjI*)

show $K \subseteq U \cap T$

by (*simp add:* $\langle K \subseteq T \rangle \langle K \subseteq U \rangle$)

show $U \cap T \subseteq \text{closure}(U \cap T)$

by (*rule closure_subset*)

show $\text{closure}(U \cap T) \subseteq S$

by (*metis* $\langle U \subseteq V \rangle \langle V \subseteq S \rangle \langle \text{compact } V \rangle \text{closure_closed closure_mono}$
 $\text{compact_imp_closed inf.cobounded1 subset_trans}$)

show $\text{openin } (\text{top_of_set } S) (U \cap T)$

by (*simp add:* $\langle \text{open } T \rangle \text{ope openin_Int_open}$)

show $\text{compact } (\text{closure}(U \cap T))$

by (*meson* $\text{Int_lower1 } \langle U \subseteq V \rangle \langle \text{compact } V \rangle \text{bounded_subset compact_closure compact_eq_bounded_closed}$)

qed *auto*

qed

next

assume $?rhs$ **then show** $?lhs$

unfolding *locally_compact_compact*

by (*metis* $\text{open_openin openin_topspace subtopology_superset top.extremum}$
 $\text{topspace_euclidean_subtopology}$)

qed

6.4.16 Sura-Bura's results about compact components of sets

proposition *Sura_Bura_compact*:

fixes $S :: 'a::euclidean_space\ set$

assumes *compact S* and $C: C \in \text{components } S$

shows $C = \bigcap \{T. C \subseteq T \wedge \text{openin } (\text{top_of_set } S) T \wedge$
 $\text{closedin } (\text{top_of_set } S) T\}$

(is $C = \bigcap ?\mathcal{T}$)

proof

obtain x **where** $x: C = \text{connected_component_set } S\ x$ **and** $x \in S$

using C **by** (*auto simp: components_def*)

have $C \subseteq S$

by (*simp add: C in_components_subset*)

have $\bigcap ?\mathcal{T} \subseteq \text{connected_component_set } S\ x$

proof (*rule connected_component_maximal*)

have $x \in C$

by (*simp add: $\langle x \in S \rangle x$*)

then show $x \in \bigcap ?\mathcal{T}$

by *blast*

have $\text{clo}: \text{closed } (\bigcap ?\mathcal{T})$

by (*simp add: $\langle \text{compact } S \rangle \text{closed_Inter closedin_compact_eq compact_imp_closed}$*)

have *False*

if $K1: \text{closedin } (\text{top_of_set } (\bigcap ?\mathcal{T}))\ K1$ **and**

$K2: \text{closedin } (\text{top_of_set } (\bigcap ?\mathcal{T}))\ K2$ **and**

$K12_Int: K1 \cap K2 = \{\}$ **and** $K12_Un: K1 \cup K2 = \bigcap ?\mathcal{T}$ **and** $K1 \neq \{\}$

$K2 \neq \{\}$

for $K1\ K2$

proof –

have $\text{closed } K1\ \text{closed } K2$

using $\text{closedin_closed_trans } \text{clo } K1\ K2$ **by** *blast+*

then obtain $V1\ V2$ **where** $\text{open } V1\ \text{open } V2\ K1 \subseteq V1\ K2 \subseteq V2$ **and** $V12:$

$V1 \cap V2 = \{\}$

using $\text{separation_normal } \langle K1 \cap K2 = \{\} \rangle$ **by** *metis*

have $SV12_ne: (S - (V1 \cup V2)) \cap (\bigcap ?\mathcal{T}) \neq \{\}$

proof (*rule compact_imp_fip*)

show $\text{compact } (S - (V1 \cup V2))$

by (*simp add: $\langle \text{open } V1 \rangle \langle \text{open } V2 \rangle \langle \text{compact } S \rangle \text{compact_diff } \text{open_Un}$*)

show $\text{clo}\mathcal{T}: \text{closed } T$ **if** $T \in ?\mathcal{T}$ **for** T

using *that* $\langle \text{compact } S \rangle$

by (*force intro: closedin_closed_trans simp add: compact_imp_closed*)

show $(S - (V1 \cup V2)) \cap \bigcap \mathcal{F} \neq \{\}$ **if** *finite* \mathcal{F} **and** $\mathcal{F}: \mathcal{F} \subseteq ?\mathcal{T}$ **for** \mathcal{F}

proof

assume $djo: (S - (V1 \cup V2)) \cap \bigcap \mathcal{F} = \{\}$

obtain D **where** $\text{ope}D: \text{openin } (\text{top_of_set } S)\ D$

and $\text{clo}D: \text{closedin } (\text{top_of_set } S)\ D$

and $C \subseteq D$ **and** $DV12: D \subseteq V1 \cup V2$

proof (*cases* $\mathcal{F} = \{\}$)

case *True*

with $\langle C \subseteq S \rangle$ *djo* **that show** *?thesis*

by *force*

```

next
  case False show ?thesis
  proof
    show ope: openin (top_of_set S) ( $\bigcap \mathcal{F}$ )
      using openin_Inter ⟨finite  $\mathcal{F}$ ⟩ False  $\mathcal{F}$  by blast
    then show closedin (top_of_set S) ( $\bigcap \mathcal{F}$ )
      by (meson cloT  $\mathcal{F}$  closed_Inter closed_subset openin_imp_subset
subset_eq)
    show  $C \subseteq \bigcap \mathcal{F}$ 
      using  $\mathcal{F}$  by auto
    show  $\bigcap \mathcal{F} \subseteq V1 \cup V2$ 
      using ope djo openin_imp_subset by fastforce
  qed
  qed
  have connected C
    by (simp add: x)
  have closed D
    using ⟨compact S⟩ cloD closedin_closed_trans compact_imp_closed by
blast
  have cloV1: closedin (top_of_set D) (D  $\cap$  closure V1)
    and cloV2: closedin (top_of_set D) (D  $\cap$  closure V2)
    by (simp_all add: closedin_closed_Int)
  moreover have D  $\cap$  closure V1 = D  $\cap$  V1 D  $\cap$  closure V2 = D  $\cap$  V2
    using ⟨D  $\subseteq V1 \cup V2$ ⟩ ⟨open V1⟩ ⟨open V2⟩ V12
  by (auto simp: closure_subset [THEN subsetD] closure_iff_nhds_not_empty,
blast+)
  ultimately have cloDV1: closedin (top_of_set D) (D  $\cap$  V1)
    and cloDV2: closedin (top_of_set D) (D  $\cap$  V2)
    by metis+
  then obtain U1 U2 where closed U1 closed U2
    and D1: D  $\cap$  V1 = D  $\cap$  U1 and D2: D  $\cap$  V2 = D  $\cap$  U2
    by (auto simp: closedin_closed)
  have D  $\cap$  U1  $\cap$  C  $\neq$  {}
  proof
    assume D  $\cap$  U1  $\cap$  C = {}
    then have *: C  $\subseteq$  D  $\cap$  V2
      using D1 DV12 ⟨C  $\subseteq$  D⟩ by auto
    have 1: openin (top_of_set S) (D  $\cap$  V2)
      by (simp add: ⟨open V2⟩ opeD openin_Int_open)
    have 2: closedin (top_of_set S) (D  $\cap$  V2)
      using cloD cloDV2 closedin_trans by blast
    have  $\bigcap ?\mathcal{T} \subseteq D \cap V2$ 
      by (rule Inter_lower) (use * 1 2 in simp)
    then show False
      using K1 V12 ⟨K1  $\neq$  {}⟩ ⟨K1  $\subseteq$  V1⟩ closedin_imp_subset by blast
  qed
  moreover have D  $\cap$  U2  $\cap$  C  $\neq$  {}
  proof
    assume D  $\cap$  U2  $\cap$  C = {}

```



```

then have *:  $C \subseteq D \cap V1$ 
  using  $D2 DV12 \langle C \subseteq D \rangle$  by auto
have 1: openin (top_of_set  $S$ ) ( $D \cap V1$ )
  by (simp add:  $\langle \text{open } V1 \rangle \text{opeD openin\_Int\_open}$ )
have 2: closedin (top_of_set  $S$ ) ( $D \cap V1$ )
  using  $\text{cloD cloDV1 closedin\_trans}$  by blast
have  $\bigcap ?\mathcal{T} \subseteq D \cap V1$ 
  by (rule Inter_lower) (use * 1 2 in simp)
then show False
  using  $K2 V12 \langle K2 \neq \{\} \rangle \langle K2 \subseteq V2 \rangle \text{closedin\_imp\_subset}$  by blast
qed
ultimately show False
  using  $\langle \text{connected } C \rangle$  [unfolded connected_closed, simplified, rule_format,
of concl:  $D \cap U1 D \cap U2$ ]
  using  $\langle C \subseteq D \rangle D1 D2 V12 DV12 \langle \text{closed } U1 \rangle \langle \text{closed } U2 \rangle \langle \text{closed } D \rangle$ 
  by blast
qed
qed
show False
  by (metis (full_types) DiffE UnE Un_upper2 SV12_ne  $\langle K1 \subseteq V1 \rangle \langle K2 \subseteq$ 
 $V2 \rangle \text{disjoint\_iff\_not\_equal subsetCE sup\_ge1 K12\_Un}$ )
qed
then show connected ( $\bigcap ?\mathcal{T}$ )
  by (auto simp: connected_closedin_eq)
show  $\bigcap ?\mathcal{T} \subseteq S$ 
  by (fastforce simp: C in_components_subset)
qed
with  $x$  show  $\bigcap ?\mathcal{T} \subseteq C$  by simp
qed auto

corollary Sura_Bura_clopen_subset:
  fixes  $S :: 'a::\text{euclidean\_space set}$ 
  assumes  $S$ : locally compact S and  $C$ :  $C \in \text{components } S$  and compact C
  and  $U$ : open U C \subseteq U
  obtains  $K$  where openin (top_of_set  $S$ )  $K$  compact K C \subseteq K K \subseteq U
proof (rule ccontr)
  assume  $\neg$  thesis
  with that have neg:  $\nexists K. \text{openin} (\text{top\_of\_set } S) K \wedge \text{compact } K \wedge C \subseteq K \wedge K$ 
 $\subseteq U$ 
  by metis
  obtain  $V K$  where  $C \subseteq V V \subseteq U V \subseteq K K \subseteq S$  compact K
  and opeSV: openin (top_of_set  $S$ )  $V$ 
  using  $S U \langle \text{compact } C \rangle$  by (meson C in_components_subset locally_compact_compact_subopen)
  let  $?\mathcal{T} = \{T. C \subseteq T \wedge \text{openin} (\text{top\_of\_set } K) T \wedge \text{compact } T \wedge T \subseteq K\}$ 
  have  $CK$ :  $C \in \text{components } K$ 
  by (meson C  $\langle C \subseteq V \rangle \langle K \subseteq S \rangle \langle V \subseteq K \rangle \text{components\_intermediate\_subset}$ 
subset_trans)
  with  $\langle \text{compact } K \rangle$ 

```

```

have  $C = \bigcap \{T. C \subseteq T \wedge \text{openin } (\text{top\_of\_set } K) T \wedge \text{closedin } (\text{top\_of\_set } K) T\}$ 
  by (simp add: Sura_Bura_compact)
then have  $Ceq: C = \bigcap ?\mathcal{T}$ 
  by (simp add: closedin_compact_eq <compact K>)
obtain  $W$  where open  $W$  and  $W: V = S \cap W$ 
  using opeSV by (auto simp: openin_open)
have  $\neg(U \cap W) \cap \bigcap ?\mathcal{T} \neq \{\}$ 
proof (rule closed_imp_fip_compact)
  show  $\neg(U \cap W) \cap \bigcap \mathcal{F} \neq \{\}$ 
    if finite  $\mathcal{F}$  and  $\mathcal{F}: \mathcal{F} \subseteq ?\mathcal{T}$  for  $\mathcal{F}$ 
    proof (cases  $\mathcal{F} = \{\}$ )
      case True
      have False if  $U = UNIV$   $W = UNIV$ 
      proof  $\neg$ 
        have  $V = S$ 
          by (simp add: W <W = UNIV>)
        with neg show False
          using  $\langle C \subseteq V \rangle \langle K \subseteq S \rangle \langle V \subseteq K \rangle \langle V \subseteq U \rangle \langle \text{compact } K \rangle$  by auto
      qed
      with True show ?thesis
        by auto
    next
      case False
      show ?thesis
      proof
        assume  $\neg(U \cap W) \cap \bigcap \mathcal{F} = \{\}$ 
        then have  $FUW: \bigcap \mathcal{F} \subseteq U \cap W$ 
          by blast
        have  $C \subseteq \bigcap \mathcal{F}$ 
          using  $\mathcal{F}$  by auto
        moreover have  $\text{compact } (\bigcap \mathcal{F})$ 
          by (metis (no_types, lifting) compact_Inter False mem_Collect_eq subsetCE  $\mathcal{F}$ )
        moreover have  $\bigcap \mathcal{F} \subseteq K$ 
          using False that(2) by fastforce
        moreover have  $\text{opeKF}: \text{openin } (\text{top\_of\_set } K) (\bigcap \mathcal{F})$ 
          using False  $\mathcal{F}$   $\langle \text{finite } \mathcal{F} \rangle$  by blast
        then have  $\text{opeVF}: \text{openin } (\text{top\_of\_set } V) (\bigcap \mathcal{F})$ 
          using  $W \langle K \subseteq S \rangle \langle V \subseteq K \rangle \text{opeKF} \langle \bigcap \mathcal{F} \subseteq K \rangle FUW$  openin_subset_trans
by fastforce
        then have  $\text{openin } (\text{top\_of\_set } S) (\bigcap \mathcal{F})$ 
          by (metis opeSV openin_trans)
        moreover have  $\bigcap \mathcal{F} \subseteq U$ 
          by (meson  $\langle V \subseteq U \rangle \text{opeVF dual_order.trans openin_imp_subset}$ )
        ultimately show False
          using neg by blast
      qed
    qed

```

```

qed (use ⟨open W⟩ ⟨open U⟩ in auto)
with W Ceq ⟨C ⊆ V⟩ ⟨C ⊆ U⟩ show False
  by auto
qed

```

```

corollary Sura_Bura_clopen_subset_alt:
  fixes S :: 'a::euclidean_space set
  assumes S: locally_compact S and C: C ∈ components S and compact C
    and opeSU: openin (top_of_set S) U and C ⊆ U
  obtains K where openin (top_of_set S) K compact K C ⊆ K K ⊆ U
proof -
  obtain V where open V U = S ∩ V
    using opeSU by (auto simp: openin_open)
  with ⟨C ⊆ U⟩ have C ⊆ V
    by auto
  then show ?thesis
    using Sura_Bura_clopen_subset [OF S C ⟨compact C⟩ ⟨open V⟩]
    by (metis ⟨U = S ∩ V⟩ inf.bounded_iff openin_imp_subset that)
qed

```

```

corollary Sura_Bura:
  fixes S :: 'a::euclidean_space set
  assumes locally_compact S C ∈ components S compact C
  shows C = ⋂ {K. C ⊆ K ∧ compact K ∧ openin (top_of_set S) K}
    (is C = ?rhs)
proof
  show ?rhs ⊆ C
  proof (clarsimp, rule ccontr)
    fix x
    assume *: ∀ X. C ⊆ X ∧ compact X ∧ openin (top_of_set S) X ⟶ x ∈ X
    and x ∉ C
    obtain U V where open U open V {x} ⊆ U C ⊆ V U ∩ V = {}
      using separation_normal [of {x} C]
      by (metis Int_empty_left ⟨x ∉ C⟩ ⟨compact C⟩ closed_empty closed_insert
compact_imp_closed insert_disjoint(1))
    have x ∉ V
      using ⟨U ∩ V = {}⟩ ⟨{x} ⊆ U⟩ by blast
    then show False
      by (meson * Sura_Bura_clopen_subset ⟨C ⊆ V⟩ ⟨open V⟩ assms(1) assms(2)
assms(3) subsetCE)
  qed
qed blast

```

6.4.17 Special cases of local connectedness and path connectedness

```

lemma locally_connected_1:
  assumes

```

$\bigwedge V x. \llbracket \text{openin } (\text{top_of_set } S) \ V; x \in V \rrbracket \implies \exists U. \text{openin } (\text{top_of_set } S) \ U$
 $\wedge \text{connected } U \wedge x \in U \wedge U \subseteq V$

shows *locally connected S*
by (*metis assms locally_def*)

lemma *locally_connected_2*:

assumes *locally connected S*
 $\text{openin } (\text{top_of_set } S) \ t$
 $x \in t$

shows $\text{openin } (\text{top_of_set } S) \ (\text{connected_component_set } t \ x)$

proof –

{ **fix** $y :: 'a$

let $?SS = \text{top_of_set } S$

assume $1: \text{openin } ?SS \ t$

$\forall w \ x. \text{openin } ?SS \ w \wedge x \in w \longrightarrow (\exists u. \text{openin } ?SS \ u \wedge (\exists v. \text{connected}$
 $v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w))$

and $\text{connected_component } t \ x \ y$

then have $y \in t$ **and** $y: y \in \text{connected_component_set } t \ x$

using $\text{connected_component_subset}$ **by** *blast+*

obtain F **where**

$\forall x \ y. (\exists w. \text{openin } ?SS \ w \wedge (\exists u. \text{connected } u \wedge x \in w \wedge w \subseteq u \wedge u \subseteq y)) =$
 $(\text{openin } ?SS \ (F \ x \ y) \wedge (\exists u. \text{connected } u \wedge x \in F \ x \ y \wedge F \ x \ y \subseteq u \wedge u \subseteq y))$

by *moura*

then obtain G **where**

$\forall a \ A. (\exists U. \text{openin } ?SS \ U \wedge (\exists V. \text{connected } V \wedge a \in U \wedge U \subseteq V \wedge V \subseteq$
 $A)) = (\text{openin } ?SS \ (F \ a \ A) \wedge \text{connected } (G \ a \ A) \wedge a \in F \ a \ A \wedge F \ a \ A \subseteq G \ a \ A$
 $\wedge G \ a \ A \subseteq A)$

by *moura*

then have $*$: $\text{openin } ?SS \ (F \ y \ t) \wedge \text{connected } (G \ y \ t) \wedge y \in F \ y \ t \wedge F \ y \ t \subseteq$
 $G \ y \ t \wedge G \ y \ t \subseteq t$

using $1 \ \langle y \in t \rangle$ **by** *presburger*

have $G \ y \ t \subseteq \text{connected_component_set } t \ y$

by (*metis (no_types) * connected_component_eq self connected_component_mono*
contra_subsetD)

then have $\exists A. \text{openin } ?SS \ A \wedge y \in A \wedge A \subseteq \text{connected_component_set } t \ x$

by (*metis (no_types) * connected_component_eq dual_order.trans y*)

}

then show *?thesis*

using *assms openin_subopen* **by** (*force simp: locally_def*)

qed

lemma *locally_connected_3*:

assumes $\bigwedge t \ x. \llbracket \text{openin } (\text{top_of_set } S) \ t; x \in t \rrbracket$

$\implies \text{openin } (\text{top_of_set } S)$

$(\text{connected_component_set } t \ x)$

$\text{openin } (\text{top_of_set } S) \ v \ x \in v$

shows $\exists u. \text{openin } (\text{top_of_set } S) \ u \wedge \text{connected } u \wedge x \in u \wedge u \subseteq v$

using *assms connected_component_subset* **by** *fastforce*

lemma *locally_connected*:

locally_connected $S \longleftrightarrow$

$(\forall v x. \text{openin } (\text{top_of_set } S) v \wedge x \in v$

$\longrightarrow (\exists u. \text{openin } (\text{top_of_set } S) u \wedge \text{connected } u \wedge x \in u \wedge u \subseteq v))$

by (*metis* *locally_connected_1* *locally_connected_2* *locally_connected_3*)

lemma *locally_connected_open_connected_component*:

locally_connected $S \longleftrightarrow$

$(\forall t x. \text{openin } (\text{top_of_set } S) t \wedge x \in t$

$\longrightarrow \text{openin } (\text{top_of_set } S) (\text{connected_component_set } t x))$

by (*metis* *locally_connected_1* *locally_connected_2* *locally_connected_3*)

lemma *locally_path_connected_1*:

assumes

$\bigwedge v x. [\text{openin } (\text{top_of_set } S) v; x \in v]$

$\implies \exists u. \text{openin } (\text{top_of_set } S) u \wedge \text{path_connected } u \wedge x \in u \wedge u \subseteq v$

shows *locally_path_connected* S

by (*force simp: locally_def dest: assms*)

lemma *locally_path_connected_2*:

assumes *locally_path_connected* S

openin $(\text{top_of_set } S) t$

$x \in t$

shows *openin* $(\text{top_of_set } S) (\text{path_component_set } t x)$

proof –

{ **fix** $y :: 'a$

let $?SS = \text{top_of_set } S$

assume $1: \text{openin } ?SS t$

$\forall w x. \text{openin } ?SS w \wedge x \in w \longrightarrow (\exists u. \text{openin } ?SS u \wedge (\exists v. \text{path_connected } v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w))$

and *path_component* $t x y$

then have $y \in t$ **and** $y: y \in \text{path_component_set } t x$

using *path_component_mem(2)* **by** *blast+*

obtain F **where**

$\forall x y. (\exists w. \text{openin } ?SS w \wedge (\exists u. \text{path_connected } u \wedge x \in w \wedge w \subseteq u \wedge u \subseteq y)) = (\text{openin } ?SS (F x y) \wedge (\exists u. \text{path_connected } u \wedge x \in F x y \wedge F x y \subseteq u \wedge u \subseteq y))$

by *moura*

then obtain G **where**

$\forall a A. (\exists U. \text{openin } ?SS U \wedge (\exists V. \text{path_connected } V \wedge a \in U \wedge U \subseteq V \wedge V \subseteq A)) = (\text{openin } ?SS (F a A) \wedge \text{path_connected } (G a A) \wedge a \in F a A \wedge F a A \subseteq G a A \wedge G a A \subseteq A)$

by *moura*

then have $*$: *openin* $?SS (F y t) \wedge \text{path_connected } (G y t) \wedge y \in F y t \wedge F y t \subseteq G y t \wedge G y t \subseteq t$

using $1 \langle y \in t \rangle$ **by** *presburger*

have $G y t \subseteq \text{path_component_set } t y$

using $*$ *path_component_maximal_rev_subsetD* **by** *blast*

then have $\exists A. \text{openin } ?SS A \wedge y \in A \wedge A \subseteq \text{path_component_set } t x$

```

    by (metis * ⟨G y t ⊆ path_component_set t y⟩ dual_order.trans path_component_eq
y)
  }
  then show ?thesis
    using assms openin_subopen by (force simp: locally_def)
qed

```

```

lemma locally_path_connected_3:
  assumes  $\bigwedge t x. \llbracket \text{openin } (\text{top\_of\_set } S) t; x \in t \rrbracket$ 
     $\implies \text{openin } (\text{top\_of\_set } S) (\text{path\_component\_set } t x)$ 
     $\text{openin } (\text{top\_of\_set } S) v \ x \in v$ 
  shows  $\exists u. \text{openin } (\text{top\_of\_set } S) u \wedge \text{path\_connected } u \wedge x \in u \wedge u \subseteq v$ 
proof -
  have path_component v x x
    by (meson assms(3) path_component_refl)
  then show ?thesis
    by (metis assms mem_Collect_eq path_component_subset path_connected_path_component)
qed

```

```

proposition locally_path_connected:
  locally_path_connected S  $\iff$ 
    ( $\forall V x. \text{openin } (\text{top\_of\_set } S) V \wedge x \in V$ 
 $\implies (\exists U. \text{openin } (\text{top\_of\_set } S) U \wedge \text{path\_connected } U \wedge x \in U \wedge U \subseteq$ 
V))
  by (metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3)

```

```

proposition locally_path_connected_open_path_component:
  locally_path_connected S  $\iff$ 
    ( $\forall t x. \text{openin } (\text{top\_of\_set } S) t \wedge x \in t$ 
 $\implies \text{openin } (\text{top\_of\_set } S) (\text{path\_component\_set } t x)$ )
  by (metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3)

```

```

lemma locally_connected_open_component:
  locally_connected S  $\iff$ 
    ( $\forall t c. \text{openin } (\text{top\_of\_set } S) t \wedge c \in \text{components } t$ 
 $\implies \text{openin } (\text{top\_of\_set } S) c$ )
  by (metis components_iff locally_connected_open_connected_component)

```

```

proposition locally_connected_im_kleinen:
  locally_connected S  $\iff$ 
    ( $\forall v x. \text{openin } (\text{top\_of\_set } S) v \wedge x \in v$ 
 $\implies (\exists u. \text{openin } (\text{top\_of\_set } S) u \wedge$ 
 $x \in u \wedge u \subseteq v \wedge$ 
 $(\forall y. y \in u \implies (\exists c. \text{connected } c \wedge c \subseteq v \wedge x \in c \wedge y \in c))))$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (fastforce simp: locally_connected)

```

```

next
  assume ?rhs
  have *:  $\exists T. \text{openin } (\text{top\_of\_set } S) T \wedge x \in T \wedge T \subseteq c$ 
    if  $\text{openin } (\text{top\_of\_set } S) t$  and  $c: c \in \text{components } t$  and  $x \in c$  for  $t c x$ 
  proof -
    from that <?rhs> [rule_format, of t x]
    obtain u where u:
       $\text{openin } (\text{top\_of\_set } S) u \wedge x \in u \wedge u \subseteq t \wedge$ 
       $(\forall y. y \in u \longrightarrow (\exists c. \text{connected } c \wedge c \subseteq t \wedge x \in c \wedge y \in c))$ 
    using in_components_subset by auto
    obtain F :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
       $\forall x y. (\exists z. z \in x \wedge y = \text{connected\_component\_set } x z) = (F x y \in x \wedge y =$ 
       $\text{connected\_component\_set } x (F x y))$ 
    by moura
    then have F:  $F t c \in t \wedge c = \text{connected\_component\_set } t (F t c)$ 
      by (meson components_iff c)
    obtain G :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a where
       $G: \forall x y. (\exists z. z \in y \wedge z \notin x) = (G x y \in y \wedge G x y \notin x)$ 
    by moura
    have  $G c u \notin u \vee G c u \in c$ 
    using F by (metis (full_types) u connected_componentI connected_component_eq
    mem_Collect_eq that(3))
    then show ?thesis
      using G u by auto
  qed
  show ?lhs
    unfolding locally_connected_open_component by (meson * openin_subopen)
  qed

proposition locally_path_connected_in_kleinen:
  locally_path_connected S  $\longleftrightarrow$ 
   $(\forall v x. \text{openin } (\text{top\_of\_set } S) v \wedge x \in v$ 
   $\longrightarrow (\exists u. \text{openin } (\text{top\_of\_set } S) u \wedge$ 
   $x \in u \wedge u \subseteq v \wedge$ 
   $(\forall y. y \in u \longrightarrow (\exists p. \text{path } p \wedge \text{path\_image } p \subseteq v \wedge$ 
   $\text{pathstart } p = x \wedge \text{pathfinish } p = y))))$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (simp add: locally_path_connected_path_connected_def)
    apply (erule all_forward ex_forward imp_forward conjE | simp)+
    by (meson dual_order.trans)
next
  assume ?rhs
  have *:  $\exists T. \text{openin } (\text{top\_of\_set } S) T \wedge$ 
     $x \in T \wedge T \subseteq \text{path\_component\_set } u z$ 
    if  $\text{openin } (\text{top\_of\_set } S) u$  and  $z \in u$  and  $c: \text{path\_component } u z x$  for  $u z$ 
  x

```

```

proof –
  have  $x \in u$ 
    by (meson  $c$  path_component_mem(2))
  with that ‹?rhs› [rule_format, of  $u$   $x$ ]
  obtain  $U$  where  $U$ :
    openin (top_of_set  $S$ )  $U \wedge x \in U \wedge U \subseteq u \wedge$ 
    ( $\forall y. y \in U \longrightarrow (\exists p. \text{path } p \wedge \text{path\_image } p \subseteq u \wedge \text{pathstart } p = x \wedge$ 
    pathfinish  $p = y$ ))
    by blast
  show ?thesis
    by (metis  $U$   $c$  mem_Collect_eq path_component_def path_component_eq
    subsetI)
  qed
  show ?lhs
    unfolding locally_path_connected_open_path_component
    using * openin_subopen by fastforce
qed

```

```

lemma locally_path_connected_imp_locally_connected:
  locally_path_connected  $S \implies$  locally_connected  $S$ 
using locally_mono path_connected_imp_connected by blast

```

```

lemma locally_connected_components:
   $\llbracket$ locally_connected  $S$ ;  $c \in$  components  $S$  $\rrbracket \implies$  locally_connected  $c$ 
by (meson locally_connected_open_component locally_open_subset openin_subtopology_self)

```

```

lemma locally_path_connected_components:
   $\llbracket$ locally_path_connected  $S$ ;  $c \in$  components  $S$  $\rrbracket \implies$  locally_path_connected  $c$ 
by (meson locally_connected_open_component locally_open_subset locally_path_connected_imp_locally
    openin_subtopology_self)

```

```

lemma locally_path_connected_connected_component:
  locally_path_connected  $S \implies$  locally_path_connected (connected_component_set
   $S$   $x$ )
by (metis components_iff_connected_component_eq_empty locally_empty locally_path_connected_comp)

```

```

lemma open_imp_locally_path_connected:
  fixes  $S :: 'a :: \text{real\_normed\_vector\_set}$ 
  assumes open  $S$ 
  shows locally_path_connected  $S$ 
proof (rule locally_mono)
  show locally_convex  $S$ 
    using assms unfolding locally_def
    by (meson open_ball centre_in_ball convex_ball openE open_subset openin_imp_subset
    openin_open_trans subset_trans)
  show  $\bigwedge T :: 'a$  set. convex  $T \implies$  path_connected  $T$ 
    using convex_imp_path_connected by blast
qed

```


lemma *open_imp_locally_connected*:
fixes $S :: 'a :: \text{real_normed_vector_set}$
shows $\text{open } S \implies \text{locally_connected } S$
by (*simp add: locally_path_connected_imp_locally_connected open_imp_locally_path_connected*)

lemma *locally_path_connected_UNIV*: *locally_path_connected* ($\text{UNIV} :: 'a :: \text{real_normed_vector_set}$)
by (*simp add: open_imp_locally_path_connected*)

lemma *locally_connected_UNIV*: *locally_connected* ($\text{UNIV} :: 'a :: \text{real_normed_vector_set}$)
by (*simp add: open_imp_locally_connected*)

lemma *openin_connected_component_locally_connected*:
locally_connected S
 $\implies \text{openin } (\text{top_of_set } S) (\text{connected_component_set } S \ x)$
by (*metis connected_component_eq_empty locally_connected_2 openin_empty openin_subtopology_self*)

lemma *openin_components_locally_connected*:
 $\llbracket \text{locally_connected } S; c \in \text{components } S \rrbracket \implies \text{openin } (\text{top_of_set } S) \ c$
using *locally_connected_open_component openin_subtopology_self* **by** *blast*

lemma *openin_path_component_locally_path_connected*:
locally_path_connected S
 $\implies \text{openin } (\text{top_of_set } S) (\text{path_component_set } S \ x)$
by (*metis (no_types) empty_iff locally_path_connected_2 openin_subopen openin_subtopology_self path_component_eq_empty*)

lemma *closedin_path_component_locally_path_connected*:
assumes *locally_path_connected* S
shows $\text{closedin } (\text{top_of_set } S) (\text{path_component_set } S \ x)$
proof –
have $\text{openin } (\text{top_of_set } S) (\bigcup (\{\text{path_component_set } S \ y \mid y. y \in S\} - \{\text{path_component_set } S \ x\}))$
using *locally_path_connected_2 assms* **by** *fastforce*
then show *?thesis*
by (*simp add: closedin_def path_component_subset complement_path_component_Union*)
qed

lemma *convex_imp_locally_path_connected*:
fixes $S :: 'a :: \text{real_normed_vector_set}$
assumes *convex* S
shows *locally_path_connected* S
proof (*clarsimp simp: locally_path_connected*)
fix $V \ x$
assume $\text{openin } (\text{top_of_set } S) \ V$ **and** $x \in V$
then obtain $T \ e$ **where** $V = S \cap T \ x \in S \ 0 < e \ \text{ball } x \ e \subseteq T$
by (*metis Int_iff openE openin_open*)

```

then have openin (top_of_set S) (S  $\cap$  ball x e) path_connected (S  $\cap$  ball x e)
by (simp_all add: assms convex_Int convex_imp_path_connected openin_open_Int)
then show  $\exists U. \textit{openin} (\textit{top\_of\_set} S) U \wedge \textit{path\_connected} U \wedge x \in U \wedge U \subseteq$ 
V
using  $\langle 0 < e \rangle \langle V = S \cap T \rangle \langle \textit{ball} x e \subseteq T \rangle \langle x \in S \rangle$  by auto
qed

```

```

lemma convex_imp_locally_connected:
  fixes S :: 'a:: real_normed_vector set
  shows convex S  $\implies$  locally_connected S
by (simp add: locally_path_connected_imp_locally_connected convex_imp_locally_path_connected)

```

6.4.18 Relations between components and path components

```

lemma path_component_eq_connected_component:
  assumes locally_path_connected S
  shows (path_component S x = connected_component S x)
proof (cases x  $\in$  S)
  case True
  have openin (top_of_set (connected_component_set S x)) (path_component_set
S x)
  proof (rule openin_subset_trans)
    show openin (top_of_set S) (path_component_set S x)
    by (simp add: True assms locally_path_connected_2)
    show connected_component_set S x  $\subseteq$  S
    by (simp add: connected_component_subset)
  qed (simp add: path_component_subset_connected_component)
  moreover have closedin (top_of_set (connected_component_set S x)) (path_component_set
S x)
  proof (rule closedin_subset_trans [of S])
    show closedin (top_of_set S) (path_component_set S x)
    by (simp add: assms closedin_path_component_locally_path_connected)
    show connected_component_set S x  $\subseteq$  S
    by (simp add: connected_component_subset)
  qed (simp add: path_component_subset_connected_component)
  ultimately have *: path_component_set S x = connected_component_set S x
  by (metis connected_connected_component connected_clopen True path_component_eq_empty)
  then show ?thesis
  by blast
next
  case False then show ?thesis
  by (metis Collect_empty_eq_bot connected_component_eq_empty path_component_eq_empty)
qed

```

```

lemma path_component_eq_connected_component_set:
  locally_path_connected S  $\implies$  (path_component_set S x = connected_component_set
S x)
by (simp add: path_component_eq_connected_component)

```

lemma *locally_path_connected_path_component*:

locally_path_connected $S \implies$ *locally_path_connected* (*path_component_set* S x)

using *locally_path_connected_connected_component_path_component_eq_connected_component*
by *fastforce*

lemma *open_path_connected_component*:

fixes $S :: 'a :: \text{real_normed_vector_set}$

shows *open* $S \implies$ *path_component* S $x =$ *connected_component* S x

by (*simp add: path_component_eq_connected_component open_imp_locally_path_connected*)

lemma *open_path_connected_component_set*:

fixes $S :: 'a :: \text{real_normed_vector_set}$

shows *open* $S \implies$ *path_component_set* S $x =$ *connected_component_set* S x

by (*simp add: open_path_connected_component*)

proposition *locally_connected_quotient_image*:

assumes *lcS*: *locally_connected* S

and *oo*: $\bigwedge T. T \subseteq f^{-1} S$

\implies *openin* (*top_of_set* S) ($S \cap f^{-1} T$) \longleftrightarrow
openin (*top_of_set* ($f^{-1} S$)) T

shows *locally_connected* ($f^{-1} S$)

proof (*clarsimp simp: locally_connected_open_component*)

fix U C

assume *opefSU*: *openin* (*top_of_set* ($f^{-1} S$)) U **and** $C \in$ *components* U

then have $C \subseteq U$ $U \subseteq f^{-1} S$

by (*meson in_components_subset openin_imp_subset*)

then have *openin* (*top_of_set* ($f^{-1} S$)) $C \longleftrightarrow$

openin (*top_of_set* S) ($S \cap f^{-1} C$)

by (*auto simp: oo*)

moreover have *openin* (*top_of_set* S) ($S \cap f^{-1} C$)

proof (*subst openin_subopen, clarify*)

fix x

assume $x \in S$ $f x \in C$

show $\exists T. \text{openin} (\text{top_of_set } S) T \wedge x \in T \wedge T \subseteq (S \cap f^{-1} C)$

proof (*intro conjI exI*)

show *openin* (*top_of_set* S) (*connected_component_set* ($S \cap f^{-1} U$) x)

proof (*rule ccontr*)

assume **: \neg *openin* (*top_of_set* S) (*connected_component_set* ($S \cap f^{-1} U$) x)

then have $x \notin (S \cap f^{-1} U)$

using $\langle U \subseteq f^{-1} S \rangle$ *opefSU* *lcS* *locally_connected_2 oo* **by** *blast*

with ** **show** *False*

by (*metis (no_types) connected_component_eq_empty empty_iff openin_subopen*)

qed

next

show $x \in$ *connected_component_set* ($S \cap f^{-1} U$) x

using $\langle C \subseteq U \rangle \langle f x \in C \rangle \langle x \in S \rangle$ **by** *auto*

next

```

have contf: continuous_on S f
  by (simp add: continuous_on_open oo openin_imp_subset)
then have continuous_on (connected_component_set (S ∩ f -' U) x) f
by (meson connected_component_subset continuous_on_subset inf.boundedE)
then have connected (f ' connected_component_set (S ∩ f -' U) x)
by (rule connected_continuous_image [OF _ connected_connected_component])
moreover have f ' connected_component_set (S ∩ f -' U) x ⊆ U
  using connected_component_in by blast
moreover have C ∩ f ' connected_component_set (S ∩ f -' U) x ≠ {}
  using ⟨C ⊆ U⟩ ⟨f x ∈ C⟩ ⟨x ∈ S⟩ by fastforce
ultimately have fC: f ' (connected_component_set (S ∩ f -' U) x) ⊆ C
  by (rule components_maximal [OF ⟨C ∈ components U⟩])
have cUC: connected_component_set (S ∩ f -' U) x ⊆ (S ∩ f -' C)
  using connected_component_subset fC by blast
have connected_component_set (S ∩ f -' U) x ⊆ connected_component_set
(S ∩ f -' C) x
  proof -
    { assume x ∈ connected_component_set (S ∩ f -' U) x
      then have ?thesis
        using cUC connected_component_idemp connected_component_mono
      }
  by blast }
  then show ?thesis
    using connected_component_eq_empty by auto
qed
also have ... ⊆ (S ∩ f -' C)
  by (rule connected_component_subset)
finally show connected_component_set (S ∩ f -' U) x ⊆ (S ∩ f -' C) .
qed
qed
ultimately show openin (top_of_set (f ' S)) C
  by metis
qed

```

The proof resembles that above but is not identical!

proposition *locally_path_connected_quotient_image*:

assumes *lcS*: *locally_path_connected S*

and *oo*: $\bigwedge T. T \subseteq f ' S$

$\implies \text{openin } (\text{top_of_set } S) (S \cap f -' T) \iff \text{openin } (\text{top_of_set } (f ' S)) T$

shows *locally_path_connected (f ' S)*

proof (*clarsimp simp: locally_path_connected_open_path_component*)

fix *U y*

assume *opefSU*: *openin (top_of_set (f ' S)) U* **and** $y \in U$

then have *path_component_set U y* $\subseteq U$ $U \subseteq f ' S$

by (*meson path_component_subset openin_imp_subset*)**+**

then have *openin (top_of_set (f ' S)) (path_component_set U y)* \iff
openin (top_of_set S) (S ∩ f -' path_component_set U y)

proof -

have *path_component_set U y* $\subseteq f ' S$

```

    using  $\langle U \subseteq f^{-1} S \rangle \langle \text{path\_component\_set } U y \subseteq U \rangle$  by blast
  then show ?thesis
    using oo by blast
qed
moreover have openin (top_of_set S) (S  $\cap$  f -' path_component_set U y)
proof (subst openin_subopen, clarify)
  fix x
  assume  $x \in S$  and Uyfx: path_component U y (f x)
  then have f x  $\in U$ 
    using path_component_mem by blast
  show  $\exists T. \text{openin (top_of_set S)} T \wedge x \in T \wedge T \subseteq (S \cap f -' \text{path\_component\_set } U y)$ 
proof (intro conjI exI)
  show openin (top_of_set S) (path_component_set (S  $\cap$  f -' U) x)
  proof (rule ccontr)
    assume **:  $\neg \text{openin (top_of_set S) (path\_component\_set (S \cap f -' U) x)}$ 
    then have  $x \notin (S \cap f -' U)$ 
    by (metis (no_types, lifting)  $\langle U \subseteq f^{-1} S \rangle$  opefSU lcS oo locally_path_connected_open_path_component)
    then show False
    using **  $\langle \text{path\_component\_set } U y \subseteq U \rangle \langle x \in S \rangle \langle \text{path\_component } U y (f x) \rangle$  by blast
  qed
next
  show  $x \in \text{path\_component\_set (S \cap f -' U) x}$ 
  by (simp add:  $\langle f x \in U \rangle \langle x \in S \rangle \text{path\_component\_refl}$ )
next
  have contf: continuous_on S f
  by (simp add: continuous_on_open oo openin_imp_subset)
  then have continuous_on (path_component_set (S  $\cap$  f -' U) x) f
  by (meson Int_lower1 continuous_on_subset path_component_subset)
  then have path_connected (f^{-1} path_component_set (S  $\cap$  f -' U) x)
  by (simp add: path_connected_continuous_image)
  moreover have f^{-1} path_component_set (S  $\cap$  f -' U) x  $\subseteq U$ 
  using path_component_mem by fastforce
  moreover have f x  $\in f^{-1} \text{path\_component\_set (S \cap f -' U) x}$ 
  by (force simp:  $\langle x \in S \rangle \langle f x \in U \rangle \text{path\_component\_refl\_eq}$ )
  ultimately have f^{-1} (path_component_set (S  $\cap$  f -' U) x)  $\subseteq \text{path\_component\_set } U (f x)$ 
  by (meson path_component_maximal)
  also have ...  $\subseteq \text{path\_component\_set } U y$ 
  by (simp add: Uyfx path_component_maximal path_component_subset path_component_sym)
  finally have fC: f^{-1} (path_component_set (S  $\cap$  f -' U) x)  $\subseteq \text{path\_component\_set } U y$  .
  have cUC: path_component_set (S  $\cap$  f -' U) x  $\subseteq (S \cap f -' \text{path\_component\_set } U y)$ 
  using path_component_subset fC by blast
  have path_component_set (S  $\cap$  f -' U) x  $\subseteq \text{path\_component\_set (S \cap f -' U) x}$ 

```

```

path_component_set U y) x
  proof -
    have  $\bigwedge a. \text{path\_component\_set } (\text{path\_component\_set } (S \cap f -' U) x) a \subseteq$ 
    path_component_set (S  $\cap$  f -' path_component_set U y) a
      using cUC path_component_mono by blast
    then show ?thesis
      using path_component_path_component by blast
    qed
    also have  $\dots \subseteq (S \cap f -' \text{path\_component\_set } U y)$ 
      by (rule path_component_subset)
    finally show path_component_set (S  $\cap$  f -' U) x  $\subseteq (S \cap f -' \text{path\_component\_set } U y)$  .
    qed
  qed
  ultimately show openin (top_of_set (f ' S)) (path_component_set U y)
    by metis
qed

```

6.4.19 Components, continuity, openin, closedin

```

lemma continuous_on_components_gen:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
  assumes  $\bigwedge C. C \in \text{components } S \Longrightarrow$ 
    openin (top_of_set S) C  $\wedge$  continuous_on C f
  shows continuous_on S f
proof (clarsimp simp: continuous_openin_preimage_eq)
  fix t :: 'b set
  assume open t
  have *: S  $\cap$  f -' t =  $(\bigcup c \in \text{components } S. c \cap f -' t)$ 
    by auto
  show openin (top_of_set S) (S  $\cap$  f -' t)
    unfolding * using  $\langle$ open t $\rangle$  assms continuous_openin_preimage_gen openin_trans
    openin_Union by blast
qed

```

```

lemma continuous_on_components:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
  assumes locally_connected S  $\wedge$   $\bigwedge C. C \in \text{components } S \Longrightarrow$  continuous_on C f
  shows continuous_on S f
proof (rule continuous_on_components_gen)
  fix C
  assume C  $\in$  components S
  then show openin (top_of_set S) C  $\wedge$  continuous_on C f
    by (simp add: assms openin_components_locally_connected)
qed

```

```

lemma continuous_on_components_eq:
  locally_connected S
   $\Longrightarrow$  (continuous_on S f  $\longleftrightarrow$   $(\forall c \in \text{components } S. \text{continuous\_on } c f)$ )

```

by (meson continuous_on_components continuous_on_subset in_components_subset)

lemma continuous_on_components_open:

fixes $S :: 'a::real_normed_vector\ set$

assumes open S

$\bigwedge c. c \in \text{components } S \implies \text{continuous_on } c\ f$

shows continuous_on $S\ f$

using continuous_on_components open_imp_locally_connected assms by blast

lemma continuous_on_components_open_eq:

fixes $S :: 'a::real_normed_vector\ set$

shows open $S \implies (\text{continuous_on } S\ f \longleftrightarrow (\forall c \in \text{components } S. \text{continuous_on } c\ f))$

using continuous_on_subset in_components_subset

by (blast intro: continuous_on_components_open)

lemma closedin_union_complement_components:

assumes U : locally_connected U

and S : closedin (top_of_set U) S

and cuS : $c \subseteq \text{components}(U - S)$

shows closedin (top_of_set U) ($S \cup \bigcup c$)

proof -

have di : $(\bigwedge S\ T. S \in c \wedge T \in c' \implies \text{disjnt } S\ T) \implies \text{disjnt } (\bigcup c) (\bigcup c')$ for c'

by (simp add: disjnt_def) blast

have $S \subseteq U$

using S closedin_imp_subset by blast

moreover have $U - S = \bigcup c \cup \bigcup (\text{components } (U - S) - c)$

by (metis Diff_partition Union_components Union_Un_distrib assms(3))

moreover have $\text{disjnt } (\bigcup c) (\bigcup (\text{components } (U - S) - c))$

apply (rule di)

by (metis di DiffD1 DiffD2 assms(3) components_nonoverlap disjnt_def subsetCE)

ultimately have eq : $S \cup \bigcup c = U - (\bigcup (\text{components}(U - S) - c))$

by (auto simp: disjnt_def)

have $*$: openin (top_of_set U) $(\bigcup (\text{components } (U - S) - c))$

proof (rule openin_Union [OF openin_trans [of $U - S$]])

show openin (top_of_set $(U - S)$) T if $T \in \text{components } (U - S) - c$ for T

using that by (simp add: $U\ S$ locally_diff_closed openin_components_locally_connected)

show openin (top_of_set U) $(U - S)$ if $T \in \text{components } (U - S) - c$ for T

using that by (simp add: openin_diff S)

qed

have closedin (top_of_set U) $(U - \bigcup (\text{components } (U - S) - c))$

by (metis closedin_diff closedin_topospace topspace_euclidean_subtopology $*$)

then have openin (top_of_set U) $(U - (U - \bigcup (\text{components } (U - S) - c)))$

by (simp add: openin_diff)

then show ?thesis

by (force simp: eq closedin_def)

qed

lemma *closed_union_complement_components*:
fixes $S :: 'a::real_normed_vector\ set$
assumes $S: closed\ S$ **and** $c: c \subseteq components(-\ S)$
shows $closed(S \cup \bigcup c)$
proof –
have $closedin\ (top_of_set\ UNIV)\ (S \cup \bigcup c)$
by (*metis Compl_eq_Diff_UNIV S c closed_closedin closedin_union_complement_components*
locally_connected_UNIV subtopology_UNIV)
then show *?thesis* **by** *simp*
qed

lemma *closedin_Un_complement_component*:
fixes $S :: 'a::real_normed_vector\ set$
assumes $u: locally\ connected\ u$
and $S: closedin\ (top_of_set\ u)\ S$
and $c: c \in components(u - S)$
shows $closedin\ (top_of_set\ u)\ (S \cup c)$
proof –
have $closedin\ (top_of_set\ u)\ (S \cup \bigcup \{c\})$
using c **by** (*blast intro: closedin_union_complement_components [OF u S]*)
then show *?thesis*
by *simp*
qed

lemma *closed_Un_complement_component*:
fixes $S :: 'a::real_normed_vector\ set$
assumes $S: closed\ S$ **and** $c: c \in components(-S)$
shows $closed\ (S \cup c)$
by (*metis Compl_eq_Diff_UNIV S c closed_closedin closedin_Un_complement_component*
locally_connected_UNIV subtopology_UNIV)

6.4.20 Existence of isometry between subspaces of same dimension

lemma *isometry_subset_subspace*:
fixes $S :: 'a::euclidean_space\ set$
and $T :: 'b::euclidean_space\ set$
assumes $S: subspace\ S$
and $T: subspace\ T$
and $d: dim\ S \leq dim\ T$
obtains f **where** $linear\ f\ f \in S \rightarrow T \wedge x. x \in S \implies norm(f\ x) = norm\ x$
proof –
obtain B **where** $B \subseteq S$ **and** $Borth: pairwise\ orthogonal\ B$
and $B1: \bigwedge x. x \in B \implies norm\ x = 1$
and $independent\ B\ finite\ B\ card\ B = dim\ S\ span\ B = S$
by (*metis orthonormal_basis_subspace [OF S] independent_imp_finite*)
obtain C **where** $C \subseteq T$ **and** $Corth: pairwise\ orthogonal\ C$
and $C1: \bigwedge x. x \in C \implies norm\ x = 1$
and $independent\ C\ finite\ C\ card\ C = dim\ T\ span\ C = T$


```

  by (metis orthonormal_basis_subspace [OF T] independent_imp_finite)
obtain fb where fb ' B ⊆ C inj_on fb B
  by (metis ⟨card B = dim S⟩ ⟨card C = dim T⟩ ⟨finite B⟩ ⟨finite C⟩ card_le_inj
d)
then have pairwise_orth_fb: pairwise (λv j. orthogonal (fb v) (fb j)) B
  using Corth unfolding pairwise_def inj_on_def
  by (blast intro: orthogonal_clauses)
obtain f where linear f and ffb: ∧x. x ∈ B ⇒ f x = fb x
  using linear_independent_extend ⟨independent B⟩ by fastforce
have span (f ' B) ⊆ span C
  by (metis ⟨fb ' B ⊆ C⟩ ffb image_cong span_mono)
then have f ' S ⊆ T
  unfolding ⟨span B = S⟩ ⟨span C = T⟩ span_linear_image[OF ⟨linear f⟩] .
have [simp]: ∧x. x ∈ B ⇒ norm (fb x) = norm x
  using B1 C1 ⟨fb ' B ⊆ C⟩ by auto
have norm (f x) = norm x if x ∈ S for x
proof -
  interpret linear f by fact
  obtain a where x: x = (∑ v ∈ B. a v *R v)
  using ⟨finite B⟩ ⟨span B = S⟩ ⟨x ∈ S⟩ span_finite by fastforce
  have norm (f x) ^2 = norm (∑ v ∈ B. a v *R fb v) ^2 by (simp add: sum_scale
ffb x)
  also have ... = (∑ v ∈ B. norm ((a v *R fb v)) ^2)
  proof (rule norm_sum_Pythagorean [OF ⟨finite B⟩])
    show pairwise (λv j. orthogonal (a v *R fb v) (a j *R fb j)) B
    by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])
  qed
  also have ... = norm x ^2
  by (simp add: x pairwise_ortho_scaleR Borth norm_sum_Pythagorean [OF
⟨finite B⟩])
  finally show ?thesis
  by (simp add: norm_eq_sqrt_inner)
qed
then show ?thesis
  by (meson ⟨f ' S ⊆ T⟩ ⟨linear f⟩ image_subset_iff_funcset that)
qed

```

proposition *isometries_subspaces:*

fixes S :: 'a::euclidean_space set

and T :: 'b::euclidean_space set

assumes S: subspace S

and T: subspace T

and d: dim S = dim T

obtains f g **where** linear f linear g f ' S = T g ' T = S

$$\bigwedge x. x \in S \implies \text{norm}(f x) = \text{norm } x$$

$$\bigwedge x. x \in T \implies \text{norm}(g x) = \text{norm } x$$

$$\bigwedge x. x \in S \implies g(f x) = x$$

$$\bigwedge x. x \in T \implies f(g x) = x$$

proof -

```

obtain  $B$  where  $B \subseteq S$  and  $Borth$ : pairwise orthogonal  $B$ 
  and  $B1$ :  $\bigwedge x. x \in B \implies norm\ x = 1$ 
  and independent  $B$  finite  $B$  card  $B = dim\ S$  span  $B = S$ 
  by (metis orthonormal_basis_subspace [OF  $S$ ] independent_imp_finite)
obtain  $C$  where  $C \subseteq T$  and  $Corth$ : pairwise orthogonal  $C$ 
  and  $C1$ :  $\bigwedge x. x \in C \implies norm\ x = 1$ 
  and independent  $C$  finite  $C$  card  $C = dim\ T$  span  $C = T$ 
  by (metis orthonormal_basis_subspace [OF  $T$ ] independent_imp_finite)
obtain  $fb$  where  $bij\_betw\ fb\ B\ C$ 
  by (metis <finite  $B$ > <finite  $C$ >  $bij\_betw\_iff\_card$  <card  $B = dim\ S$ > <card  $C = dim\ T$ >  $d$ )
then have pairwise_orth_fb: pairwise ( $\lambda v\ j. orthogonal\ (fb\ v)\ (fb\ j)$ )  $B$ 
  using  $Corth$  unfolding pairwise_def inj_on_def  $bij\_betw\_def$ 
  by (blast intro: orthogonal_clauses)
obtain  $f$  where linear  $f$  and  $ffb$ :  $\bigwedge x. x \in B \implies f\ x = fb\ x$ 
  using linear_independent_extend <independent  $B$ > by fastforce
interpret  $f$ : linear  $f$  by fact
define  $gb$  where  $gb \equiv inv\_into\ B\ fb$ 
then have pairwise_orth_gb: pairwise ( $\lambda v\ j. orthogonal\ (gb\ v)\ (gb\ j)$ )  $C$ 
  using  $Borth$  < $bij\_betw\ fb\ B\ C$ > unfolding pairwise_def  $bij\_betw\_def$  by force
obtain  $g$  where linear  $g$  and  $ggb$ :  $\bigwedge x. x \in C \implies g\ x = gb\ x$ 
  using linear_independent_extend <independent  $C$ > by fastforce
interpret  $g$ : linear  $g$  by fact
have span ( $f\ 'B$ )  $\subseteq$  span  $C$ 
  by (metis < $bij\_betw\ fb\ B\ C$ >  $bij\_betw\_imp\_surj\_on\ eq\_iff\_ffb\ image\_cong$ )
then have  $f\ 'S \subseteq T$ 
  unfolding <span  $B = S$ > <span  $C = T$ > span_linear_image[OF <linear  $f$ >] .
have [ $simp$ ]:  $\bigwedge x. x \in B \implies norm\ (fb\ x) = norm\ x$ 
  using  $B1\ C1$  < $bij\_betw\ fb\ B\ C$ >  $bij\_betw\_imp\_surj\_on$  by fastforce
have  $f$  [ $simp$ ]:  $norm\ (f\ x) = norm\ x$   $g\ (f\ x) = x$  if  $x \in S$  for  $x$ 
proof -
  obtain  $a$  where  $x: x = (\sum v \in B. a\ v *_{\mathbb{R}} v)$ 
  using <finite  $B$ > <span  $B = S$ > < $x \in S$ > span_finite by fastforce
  have  $f\ x = (\sum v \in B. f\ (a\ v *_{\mathbb{R}} v))$ 
  using linear_sum [OF <linear  $f$ >]  $x$  by auto
  also have ... =  $(\sum v \in B. a\ v *_{\mathbb{R}} f\ v)$ 
  by (simp add:  $f.sum\ f.scale$ )
  also have ... =  $(\sum v \in B. a\ v *_{\mathbb{R}} fb\ v)$ 
  by (simp add:  $ffb.cong: sum.cong$ )
  finally have *:  $f\ x = (\sum v \in B. a\ v *_{\mathbb{R}} fb\ v)$  .
then have  $(norm\ (f\ x))^2 = (norm\ (\sum v \in B. a\ v *_{\mathbb{R}} fb\ v))^2$  by simp
  also have ... =  $(\sum v \in B. norm\ ((a\ v *_{\mathbb{R}} fb\ v))^2)$ 
proof (rule norm_sum_Pythagorean [OF <finite  $B$ >])
  show pairwise ( $\lambda v\ j. orthogonal\ (a\ v *_{\mathbb{R}} fb\ v)\ (a\ j *_{\mathbb{R}} fb\ j)$ )  $B$ 
  by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])
qed
also have ... =  $(norm\ x)^2$ 
  by (simp add:  $x\ pairwise\_ortho\_scaleR\ Borth\ norm\_sum\_Pythagorean$  [OF <finite  $B$ >])

```

```

finally show  $\text{norm } (f x) = \text{norm } x$ 
  by (simp add: norm_eq_sqrt_inner)
have  $g (f x) = g (\sum_{v \in B}. a v *_R fb v)$  by (simp add: *)
also have  $\dots = (\sum_{v \in B}. g (a v *_R fb v))$ 
  by (simp add: g.sum g.scale)
also have  $\dots = (\sum_{v \in B}. a v *_R g (fb v))$ 
  by (simp add: g.scale)
also have  $\dots = (\sum_{v \in B}. a v *_R v)$ 
proof (rule sum.cong [OF refl])
  show  $a x *_R g (fb x) = a x *_R x$  if  $x \in B$  for  $x$ 
    using that  $\langle \text{bij\_betw } fb \ B \ C \rangle$  bij_betwE bij_betw_inv_into_left gb_def ggb
by fastforce
qed
also have  $\dots = x$ 
  using  $x$  by blast
finally show  $g (f x) = x$  .
qed
have [simp]:  $\bigwedge x. x \in C \implies \text{norm } (gb x) = \text{norm } x$ 
by (metis B1 C1 \langle bij_betw fb B C \rangle bij_betw_imp_surj_on gb_def inv_into_into)
have  $g$  [simp]:  $f (g x) = x$  if  $x \in T$  for  $x$ 
proof -
  obtain  $a$  where  $x = (\sum_{v \in C}. a v *_R v)$ 
    using  $\langle \text{finite } C \rangle$   $\langle \text{span } C = T \rangle$   $\langle x \in T \rangle$  span_finite by fastforce
have  $g x = (\sum_{v \in C}. g (a v *_R v))$ 
  by (simp add: x.g.sum)
also have  $\dots = (\sum_{v \in C}. a v *_R g v)$ 
  by (simp add: g.scale)
also have  $\dots = (\sum_{v \in C}. a v *_R gb v)$ 
  by (simp add: ggb cong: sum.cong)
finally have  $f (g x) = f (\sum_{v \in C}. a v *_R gb v)$  by simp
also have  $\dots = (\sum_{v \in C}. f (a v *_R gb v))$ 
  by (simp add: f.scale f.sum)
also have  $\dots = (\sum_{v \in C}. a v *_R f (gb v))$ 
  by (simp add: f.scale f.sum)
also have  $\dots = (\sum_{v \in C}. a v *_R v)$ 
  using  $\langle \text{bij\_betw } fb \ B \ C \rangle$ 
by (simp add: bij_betw_def gb_def bij_betw_inv_into_right ffb inv_into_into)
also have  $\dots = x$ 
  using  $x$  by blast
finally show  $f (g x) = x$  .
qed
have gim:  $g \text{ ' } T = S$ 
  by (metis (full_types) S T \langle f \text{ ' } S \subseteq T \rangle d dim_eq_span dim_image_le f(2))
g.linear_axioms
  image_iff linear_subspace_image span_eq_iff subset_iff)
have fm:  $f \text{ ' } S = T$ 
  using  $\langle g \text{ ' } T = S \rangle$  image_iff by fastforce
have [simp]:  $\text{norm } (g x) = \text{norm } x$  if  $x \in T$  for  $x$ 
  using fm that by auto

```

1152

show ?thesis
 by (rule that [OF ⟨linear f⟩ ⟨linear g⟩]) (simp_all add: fim gim)
qed

corollary *isometry_subspaces*:
 fixes $S :: 'a::\text{euclidean_space set}$
 and $T :: 'b::\text{euclidean_space set}$
 assumes S : subspace S
 and T : subspace T
 and d : $\dim S = \dim T$
 obtains f **where** linear f $f ' S = T \wedge x. x \in S \implies \text{norm}(f x) = \text{norm } x$
using *isometries_subspaces* [OF *assms*]
by *metis*

corollary *isomorphisms_UNIV_UNIV*:
 assumes $\text{DIM}('M) = \text{DIM}('N)$
 obtains $f::'M::\text{euclidean_space} \Rightarrow 'N::\text{euclidean_space}$ **and** g
 where linear f linear g
 $\wedge x. \text{norm}(f x) = \text{norm } x \wedge y. \text{norm}(g y) = \text{norm } y$
 $\wedge x. g (f x) = x \wedge y. f(g y) = y$
 using *assms* **by** (auto intro: *isometries_subspaces* [of $\text{UNIV}::'M$ set $\text{UNIV}::'N$ set])

lemma *homeomorphic_subspaces*:
 fixes $S :: 'a::\text{euclidean_space set}$
 and $T :: 'b::\text{euclidean_space set}$
 assumes S : subspace S
 and T : subspace T
 and d : $\dim S = \dim T$
 shows S homeomorphic T
proof –
 obtain $f g$ **where** linear f linear g $f ' S = T$ $g ' T = S$
 $\wedge x. x \in S \implies g(f x) = x \wedge x. x \in T \implies f(g x) = x$
 by (blast intro: *isometries_subspaces* [OF *assms*])
 then show ?thesis
 unfolding *homeomorphic_def* *homeomorphism_def*
 apply (rule_tac $x=f$ **in** *exI*, rule_tac $x=g$ **in** *exI*)
 apply (auto simp: *linear_continuous_on* *linear_conv_bounded_linear*)
 done
qed

lemma *homeomorphic_affine_sets*:
 assumes affine S affine T $\text{aff_dim } S = \text{aff_dim } T$
 shows S homeomorphic T
proof (cases $S = \{\}$ $\vee T = \{\}$)
 case True **with** *assms* *aff_dim_empty* *homeomorphic_empty* **show** ?thesis
 by *metis*
next
 case False

```

then obtain  $a\ b$  where  $ab: a \in S\ b \in T$  by auto
then have  $ss: \text{subspace } ((+) (- a) ' S) \text{ subspace } ((+) (- b) ' T)$ 
  using affine_diffs_subspace assms by blast+
have  $dd: \text{dim } ((+) (- a) ' S) = \text{dim } ((+) (- b) ' T)$ 
  using  $assms\ ab$  by (simp add: aff_dim_eq_dim [OF hull_inc] image_def)
have  $S$  homeomorphic  $((+) (- a) ' S)$ 
  by (fact homeomorphic_translation)
also have  $\dots$  homeomorphic  $((+) (- b) ' T)$ 
  by (rule homeomorphic_subspaces [OF ss dd])
also have  $\dots$  homeomorphic  $T$ 
  using homeomorphic_translation [of T - b] by (simp add: homeomorphic_sym
[of T])
  finally show ?thesis .
qed

```

6.4.21 Retracts, in a general sense, preserve (co)homotopic triviality)

```

locale Retracts =
  fixes  $S\ h\ t\ k$ 
  assumes  $conth: \text{continuous\_on } S\ h$ 
    and  $imh: h ' S = t$ 
    and  $contk: \text{continuous\_on } t\ k$ 
    and  $imk: k \in t \rightarrow S$ 
    and  $idhk: \bigwedge y. y \in t \implies h(k\ y) = y$ 

```

begin

```

lemma homotopically_trivial_retraction_gen:
  assumes  $P: \bigwedge f. \llbracket \text{continuous\_on } U\ f; f \in U \rightarrow t; Q\ f \rrbracket \implies P(k \circ f)$ 
    and  $Q: \bigwedge f. \llbracket \text{continuous\_on } U\ f; f \in U \rightarrow S; P\ f \rrbracket \implies Q(h \circ f)$ 
    and  $Qeq: \bigwedge h\ k. (\bigwedge x. x \in U \implies h\ x = k\ x) \implies Q\ h = Q\ k$ 
    and  $hom: \bigwedge f\ g. \llbracket \text{continuous\_on } U\ f; f \in U \rightarrow S; P\ f; \text{continuous\_on } U\ g; g \in U \rightarrow S; P\ g \rrbracket \implies \text{homotopic\_with\_canon } P\ U\ S\ f\ g$ 
    and  $contf: \text{continuous\_on } U\ f$  and  $imf: f \in U \rightarrow t$  and  $Qf: Q\ f$ 
    and  $contg: \text{continuous\_on } U\ g$  and  $img: g \in U \rightarrow t$  and  $Qg: Q\ g$ 
  shows homotopic\_with\_canon  $Q\ U\ t\ f\ g$ 

```

proof –

```

  have  $\text{continuous\_on } U\ (k \circ f)$ 
    by (meson contf continuous_on_compose continuous_on_subset contk func-set_image imf)
  moreover have  $(k \circ f) ' U \subseteq S$ 
    using  $imf\ imk$  by fastforce
  moreover have  $P\ (k \circ f)$ 
    by (simp add: P Qf contf imf)
  moreover have  $\text{continuous\_on } U\ (k \circ g)$ 
    by (meson contg continuous_on_compose continuous_on_subset contk func-set_image img)

```

```

moreover have  $(k \circ g) \text{ ' } U \subseteq S$ 
  using img imk by fastforce
moreover have  $P (k \circ g)$ 
  by (simp add: P Qg contg img)
ultimately have homotopic_with_canon  $P U S (k \circ f) (k \circ g)$ 
  by (simp add: hom image_subset_iff)
then have homotopic_with_canon  $Q U t (h \circ (k \circ f)) (h \circ (k \circ g))$ 
apply (rule homotopic_with_compose_continuous_left [OF homotopic_with_mono])
  using Q conth imh by force+
then show ?thesis
proof (rule homotopic_with_eq; simp)
  show  $\bigwedge h k. (\bigwedge x. x \in U \implies h x = k x) \implies Q h = Q k$ 
    using Qeq topspace_euclidean_subtopology by blast
  show  $\bigwedge x. x \in U \implies f x = h (k (f x)) \wedge x. x \in U \implies g x = h (k (g x))$ 
    using idhk imf img by fastforce+
qed
qed

lemma homotopically_trivial_retraction_null_gen:
  assumes  $P: \bigwedge f. \llbracket \text{continuous\_on } U f; f \in U \rightarrow t; Q f \rrbracket \implies P(k \circ f)$ 
    and  $Q: \bigwedge f. \llbracket \text{continuous\_on } U f; f \in U \rightarrow S; P f \rrbracket \implies Q(h \circ f)$ 
    and  $Qeq: \bigwedge h k. (\bigwedge x. x \in U \implies h x = k x) \implies Q h = Q k$ 
    and  $hom: \bigwedge f. \llbracket \text{continuous\_on } U f; f \in U \rightarrow S; P f \rrbracket$ 
       $\implies \exists c. \text{homotopic\_with\_canon } P U S f (\lambda x. c)$ 
    and  $contf: \text{continuous\_on } U f$  and  $imf: f \in U \rightarrow t$  and  $Qf: Q f$ 
  obtains  $c$  where homotopic_with_canon  $Q U t f (\lambda x. c)$ 
proof -
  have  $feq: \bigwedge x. x \in U \implies (h \circ (k \circ f)) x = f x$  using idhk imf by auto
  have continuous_on  $U (k \circ f)$ 
    by (meson contf continuous_on_compose continuous_on_subset contk func-
set_image imf)
  moreover have  $(k \circ f) \in U \rightarrow S$ 
    using imf imk by fastforce
  moreover have  $P (k \circ f)$ 
    by (simp add: P Qf contf imf)
  ultimately obtain  $c$  where homotopic_with_canon  $P U S (k \circ f) (\lambda x. c)$ 
    by (metis hom)
  then have homotopic_with_canon  $Q U t (h \circ (k \circ f)) (h \circ (\lambda x. c))$ 
apply (rule homotopic_with_compose_continuous_left [OF homotopic_with_mono])
  using Q conth imh by force+
then have homotopic_with_canon  $Q U t f (\lambda x. h c)$ 
proof (rule homotopic_with_eq)
  show  $\bigwedge x. x \in \text{topspace } (\text{top\_of\_set } U) \implies f x = (h \circ (k \circ f)) x$ 
    using feq by auto
  show  $\bigwedge h k. (\bigwedge x. x \in \text{topspace } (\text{top\_of\_set } U) \implies h x = k x) \implies Q h = Q k$ 
    using Qeq topspace_euclidean_subtopology by blast
qed auto
then show ?thesis
  using that by blast

```

qed

lemma *cohomotopically_trivial_retraction_gen*:

assumes $P: \bigwedge f. \llbracket \text{continuous_on } t \ f; f \in t \rightarrow U; Q \ f \rrbracket \Longrightarrow P(f \circ h)$
 and $Q: \bigwedge f. \llbracket \text{continuous_on } S \ f; f \in S \rightarrow U; P \ f \rrbracket \Longrightarrow Q(f \circ k)$
 and $Qeq: \bigwedge h \ k. (\bigwedge x. x \in t \Longrightarrow h \ x = k \ x) \Longrightarrow Q \ h = Q \ k$
 and $hom: \bigwedge f \ g. \llbracket \text{continuous_on } S \ f; f \in S \rightarrow U; P \ f; \text{continuous_on } S \ g; g \in S \rightarrow U; P \ g \rrbracket \Longrightarrow \text{homotopic_with_canon } P \ S \ U \ f \ g$
 and $contf: \text{continuous_on } t \ f$ and $imf: f \in t \rightarrow U$ and $Qf: Q \ f$
 and $contg: \text{continuous_on } t \ g$ and $img: g \in t \rightarrow U$ and $Qg: Q \ g$
 shows *homotopic_with_canon* $Q \ t \ U \ f \ g$

proof –

have $feq: \bigwedge x. x \in t \Longrightarrow (f \circ h \circ k) \ x = f \ x$ using *idhk imf by auto*
 have $geq: \bigwedge x. x \in t \Longrightarrow (g \circ h \circ k) \ x = g \ x$ using *idhk img by auto*
 have *continuous_on* $S \ (f \circ h)$
 using *contf conth continuous_on_compose imh by blast*
 moreover have $(f \circ h) \in S \rightarrow U$
 using *imf imh by fastforce*
 moreover have $P \ (f \circ h)$
 by (*simp add: P Qf contf imf*)
 moreover have *continuous_on* $S \ (g \circ h)$
 using *contg continuous_on_compose continuous_on_subset conth imh by blast*
 moreover have $(g \circ h) \in S \rightarrow U$
 using *img imh by fastforce*
 moreover have $P \ (g \circ h)$
 by (*simp add: P Qg contg img*)
 ultimately have *homotopic_with_canon* $P \ S \ U \ (f \circ h) \ (g \circ h)$
 by (*simp add: hom*)
 then have *homotopic_with_canon* $Q \ t \ U \ (f \circ h \circ k) \ (g \circ h \circ k)$
 apply (*rule homotopic_with_compose_continuous_right [OF homotopic_with_mono]*)
 using $Q \ contk \ imk$ by *force+*
 then show *?thesis*
 proof (*rule homotopic_with_eq*)
 show $f \ x = (f \circ h \circ k) \ x \ g \ x = (g \circ h \circ k) \ x$
 if $x \in \text{topspace } (\text{top_of_set } t)$ for x
 using $feq \ geq$ that by *force+*
 qed (*use Qeq topspace_euclidean_subtopology in blast*)

qed

lemma *cohomotopically_trivial_retraction_null_gen*:

assumes $P: \bigwedge f. \llbracket \text{continuous_on } t \ f; f \in t \rightarrow U; Q \ f \rrbracket \Longrightarrow P(f \circ h)$
 and $Q: \bigwedge f. \llbracket \text{continuous_on } S \ f; f \in S \rightarrow U; P \ f \rrbracket \Longrightarrow Q(f \circ k)$
 and $Qeq: \bigwedge h \ k. (\bigwedge x. x \in t \Longrightarrow h \ x = k \ x) \Longrightarrow Q \ h = Q \ k$
 and $hom: \bigwedge f \ g. \llbracket \text{continuous_on } S \ f; f \in S \rightarrow U; P \ f \rrbracket \Longrightarrow \exists c. \text{homotopic_with_canon } P \ S \ U \ f \ (\lambda x. c)$
 and $contf: \text{continuous_on } t \ f$ and $imf: f \in t \rightarrow U$ and $Qf: Q \ f$
 obtains c where *homotopic_with_canon* $Q \ t \ U \ f \ (\lambda x. c)$

proof –

```

have feq:  $\bigwedge x. x \in t \implies (f \circ h \circ k) x = f x$  using idhk imf by auto
have continuous_on S (f o h)
  using contf conth continuous_on_compose imh by blast
moreover have (f o h)  $\in S \rightarrow U$ 
  using imf imh by fastforce
moreover have P (f o h)
  by (simp add: P Qf contf imf)
ultimately obtain c where homotopic_with_canon P S U (f o h) ( $\lambda x. c$ )
  by (metis hom)
then have §: homotopic_with_canon Q t U (f o h o k) (( $\lambda x. c$ ) o k)
proof (rule homotopic_with_compose_continuous_right [OF homotopic_with_mono])
  show  $\bigwedge h. \llbracket \text{continuous\_map } (\text{top\_of\_set } S) (\text{top\_of\_set } U) h; P h \rrbracket \implies Q (h$ 
o k)
  using Q by auto
qed (use contk imk in force)+
moreover have homotopic_with_canon Q t U f ( $\lambda x. c$ )
  using homotopic_with_eq [OF §] feq Qeq by fastforce
ultimately show ?thesis
  using that by blast
qed
end

```

lemma *simply_connected_retraction_gen*:

```

shows  $\llbracket \text{simply\_connected } S; \text{continuous\_on } S h; h \text{ ' } S = T;
\text{continuous\_on } T k; k \in T \rightarrow S; \bigwedge y. y \in T \implies h(k y) = y \rrbracket
\implies \text{simply\_connected } T$ 

```

apply (simp add: simply_connected_def path_def path_image_def homotopic_loops_def, clarify)

apply (rule Retracts.homotopically_trivial_retraction_gen

```

[ $\text{of } S h \_ k \_ \lambda p. \text{pathfinish } p = \text{pathstart } p \_ \lambda p. \text{pathfinish } p = \text{pathstart } p$ ])

```

apply (simp_all add: Retracts_def pathfinish_def pathstart_def image_subset_iff_funcset)
done

lemma *homeomorphic_simply_connected*:

```

 $\llbracket S \text{ homeomorphic } T; \text{simply\_connected } S \rrbracket \implies \text{simply\_connected } T$ 

```

by (auto simp: homeomorphic_def homeomorphism_def intro: simply_connected_retraction_gen)

lemma *homeomorphic_simply_connected_eq*:

```

 $S \text{ homeomorphic } T \implies (\text{simply\_connected } S \longleftrightarrow \text{simply\_connected } T)$ 

```

by (metis homeomorphic_simply_connected homeomorphic_sym)

6.4.22 Homotopy equivalence

6.4.23 Homotopy equivalence of topological spaces.

definition *homotopy_equivalent_space*

```

(infix homotopy'_equivalent'_space 50)

```

where $X \text{ homotopy_equivalent_space } Y \equiv$

```

( $\exists f g. \text{continuous\_map } X Y f \wedge$ 
```



```

    continuous_map Y X g  $\wedge$ 
    homotopic_with ( $\lambda x. \text{True}$ ) X X (g  $\circ$  f) id  $\wedge$ 
    homotopic_with ( $\lambda x. \text{True}$ ) Y Y (f  $\circ$  g) id

```

lemma *homeomorphic_imp_homotopy_equivalent_space*:

X homeomorphic_space *Y* \implies *X* homotopy_equivalent_space *Y*

unfolding homeomorphic_space_def homotopy_equivalent_space_def

apply (erule ex_forward)+

by (simp add: homotopic_with_equal homotopic_with_sym homeomorphic_maps_def)

lemma *homotopy_equivalent_space_refl*:

X homotopy_equivalent_space *X*

by (simp add: homeomorphic_imp_homotopy_equivalent_space homeomorphic_space_refl)

lemma *homotopy_equivalent_space_sym*:

X homotopy_equivalent_space *Y* \longleftrightarrow *Y* homotopy_equivalent_space *X*

by (meson homotopy_equivalent_space_def)

lemma *homotopy_eqv_trans* [trans]:

assumes 1: *X* homotopy_equivalent_space *Y* **and** 2: *Y* homotopy_equivalent_space *U*

shows *X* homotopy_equivalent_space *U*

proof –

obtain *f1 g1* **where** *f1*: continuous_map *X* *Y* *f1*

and *g1*: continuous_map *Y* *X* *g1*

and *hom1*: homotopic_with ($\lambda x. \text{True}$) *X* *X* (*g1* \circ *f1*) id
 homotopic_with ($\lambda x. \text{True}$) *Y* *Y* (*f1* \circ *g1*) id

using 1 **by** (auto simp: homotopy_equivalent_space_def)

obtain *f2 g2* **where** *f2*: continuous_map *Y* *U* *f2*

and *g2*: continuous_map *U* *Y* *g2*

and *hom2*: homotopic_with ($\lambda x. \text{True}$) *Y* *Y* (*g2* \circ *f2*) id
 homotopic_with ($\lambda x. \text{True}$) *U* *U* (*f2* \circ *g2*) id

using 2 **by** (auto simp: homotopy_equivalent_space_def)

have homotopic_with ($\lambda f. \text{True}$) *X* *Y* (*g2* \circ *f2* \circ *f1*) (id \circ *f1*)

using *f1* hom2(1) homotopic_with_compose_continuous_map_right **by** metis

then have homotopic_with ($\lambda f. \text{True}$) *X* *Y* (*g2* \circ (*f2* \circ *f1*)) (id \circ *f1*)

by (simp add: o_assoc)

then have homotopic_with ($\lambda x. \text{True}$) *X* *X*

(*g1* \circ (*g2* \circ (*f2* \circ *f1*))) (*g1* \circ (id \circ *f1*))

by (simp add: *g1* homotopic_with_compose_continuous_map_left)

moreover have homotopic_with ($\lambda x. \text{True}$) *X* *X* (*g1* \circ id \circ *f1*) id

using *hom1* **by** simp

ultimately have *SS*: homotopic_with ($\lambda x. \text{True}$) *X* *X* (*g1* \circ *g2* \circ (*f2* \circ *f1*)) id

by (metis comp_assoc homotopic_with_trans id_comp)

have homotopic_with ($\lambda f. \text{True}$) *U* *Y* (*f1* \circ *g1* \circ *g2*) (id \circ *g2*)

using *g2* hom1(2) homotopic_with_compose_continuous_map_right **by** fast-force

then have homotopic_with ($\lambda f. \text{True}$) *U* *Y* (*f1* \circ (*g1* \circ *g2*)) (id \circ *g2*)

by (simp add: o_assoc)

then have *homotopic_with* ($\lambda x. \text{True}$) $U U$
 $(f2 \circ (f1 \circ (g1 \circ g2))) (f2 \circ (id \circ g2))$
by (*simp add: f2 homotopic_with_compose_continuous_map_left*)
moreover have *homotopic_with* ($\lambda x. \text{True}$) $U U (f2 \circ id \circ g2) id$
using *hom2 by simp*
ultimately have $UU: \text{homotopic_with } (\lambda x. \text{True}) U U (f2 \circ f1 \circ (g1 \circ g2)) id$
by (*simp add: fun.map_comp hom2(2) homotopic_with_trans*)
show *?thesis*
unfolding *homotopy_equivalent_space_def*
by (*blast intro: f1 f2 g1 g2 continuous_map_compose SS UU*)
qed

lemma *deformation_retraction_imp_homotopy_equivalent_space*:
 $\llbracket \text{homotopic_with } (\lambda x. \text{True}) X X (S \circ r) id; \text{retraction_maps } X Y r S \rrbracket$
 $\implies X \text{ homotopy_equivalent_space } Y$
unfolding *homotopy_equivalent_space_def retraction_maps_def*
using *homotopic_with_id2 by fastforce*

lemma *deformation_retract_imp_homotopy_equivalent_space*:
 $\llbracket \text{homotopic_with } (\lambda x. \text{True}) X X r id; \text{retraction_maps } X Y r id \rrbracket$
 $\implies X \text{ homotopy_equivalent_space } Y$
using *deformation_retraction_imp_homotopy_equivalent_space by force*

lemma *deformation_retract_of_space*:
 $S \subseteq \text{topspace } X \wedge$
 $(\exists r. \text{homotopic_with } (\lambda x. \text{True}) X X id r \wedge \text{retraction_maps } X (\text{subtopology } X S) r id) \iff$
 $S \text{ retract_of_space } X \wedge (\exists f. \text{homotopic_with } (\lambda x. \text{True}) X X id f \wedge f' (\text{topspace } X) \subseteq S)$

proof (*cases S ⊆ topspace X*)

case *True*

moreover have $(\exists r. \text{homotopic_with } (\lambda x. \text{True}) X X id r \wedge \text{retraction_maps } X (\text{subtopology } X S) r id)$

$\iff (S \text{ retract_of_space } X \wedge (\exists f. \text{homotopic_with } (\lambda x. \text{True}) X X id f \wedge f' (\text{topspace } X) \subseteq S))$

unfolding *retract_of_space_def*

proof *safe*

fix $f r$

assume $f: \text{homotopic_with } (\lambda x. \text{True}) X X id f$

and $fS: f' \text{ topspace } X \subseteq S$

and $r: \text{continuous_map } X (\text{subtopology } X S) r$

and $\text{req}: \forall x \in S. r x = x$

show $\exists r. \text{homotopic_with } (\lambda x. \text{True}) X X id r \wedge \text{retraction_maps } X (\text{subtopology } X S) r id$

proof (*intro exI conjI*)

have *homotopic_with* ($\lambda x. \text{True}$) $X X f r$

proof (*rule homotopic_with_eq*)

show *homotopic_with* ($\lambda x. \text{True}$) $X X (r \circ f) (r \circ id)$

by (*metis continuous_map_into_fulltopology f homotopic_with_compose_continuous_map_left*)

```

homotopic_with_symD r)
  show f x = (r ∘ f) x if x ∈ topspace X for x
  using that fS req by auto
qed auto
then show homotopic_with (λx. True) X X id r
  by (rule homotopic_with_trans [OF f])
next
show retraction_maps X (subtopology X S) r id
  by (simp add: r req retraction_maps_def)
qed
qed (use True in ⟨auto simp: retraction_maps_def topspace_subtopology_subset
continuous_map_in_subtopology⟩)
ultimately show ?thesis by simp
qed (auto simp: retract_of_space_def retraction_maps_def)

```

6.4.24 Contractible spaces

The definition (which agrees with "contractible" on subsets of Euclidean space) is a little cryptic because we don't in fact assume that the constant "a" is in the space. This forces the convention that the empty space / set is contractible, avoiding some special cases.

definition *contractible_space where*

```
contractible_space X ≡ ∃ a. homotopic_with (λx. True) X X id (λx. a)
```

lemma *contractible_space_top_of_set [simp]:contractible_space (top_of_set S)*
 \longleftrightarrow *contractible S*

```
by (auto simp: contractible_space_def contractible_def)
```

lemma *contractible_space_empty [simp]:*

```

contractible_space trivial_topology
unfolding contractible_space_def homotopic_with_def
apply (rule_tac x=undefined in exI)
apply (rule_tac x=λ(t,x). if t = 0 then x else undefined in exI)
apply (auto simp: continuous_map_on_empty)
done

```

lemma *contractible_space_singleton [simp]:*

```

contractible_space (discrete_topology {a})
unfolding contractible_space_def homotopic_with_def
apply (rule_tac x=a in exI)
apply (rule_tac x=λ(t,x). if t = 0 then x else a in exI)
apply (auto intro: continuous_map_eq [where f = λz. a])
done

```

lemma *contractible_space_subset_singleton:*

```

topspace X ⊆ {a} ⇒ contractible_space X
by (metis contractible_space_empty contractible_space_singleton null_topspace_iff_trivial
subset_singletonD subtopology_eq_discrete_topology_sing)

```

lemma *contractible_space_subtopology_singleton* [simp]:

contractible_space (subtopology X $\{a\}$)

by (meson *contractible_space_subset_singleton insert_subset path_connectedin_singleton path_connectedin_subtopology subsetI*)

lemma *contractible_space*:

contractible_space $X \iff$

$X = \text{trivial_topology} \vee$

$(\exists a \in \text{topspace } X. \text{homotopic_with } (\lambda x. \text{True}) X X \text{id } (\lambda x. a))$

proof (cases $X = \text{trivial_topology}$)

case *False*

then show ?thesis

using *homotopic_with_imp_continuous_maps* **by** (fastforce simp: *contractible_space_def*)

qed (simp add: *contractible_space_empty*)

lemma *contractible_imp_path_connected_space*:

assumes *contractible_space* X **shows** *path_connected_space* X

proof (cases $X = \text{trivial_topology}$)

case *False*

have *: *path_connected_space* X

if $a \in \text{topspace } X$ **and** *conh*: *continuous_map* (prod_topology (top_of_set {0..1}) X) X h

and $h: \forall x. h(0, x) = x \wedge x. h(1, x) = a$

for a **and** $h :: \text{real} \times 'a \Rightarrow 'a$

proof -

have *path_component_of* X b **if** $b \in \text{topspace } X$ **for** b

unfolding *path_component_of_def*

proof (intro exI conjI)

let ? $g = h \circ (\lambda x. (x, b))$

show *pathin* X ? g

unfolding *pathin_def*

proof (rule *continuous_map_compose* [OF_ *conh*])

show *continuous_map* (top_of_set {0..1}) (prod_topology (top_of_set {0..1}) X) $(\lambda x. (x, b))$

using *that* **by** (auto intro!: *continuous_intros*)

qed

qed (use h in auto)

then show ?thesis

by (metis *path_component_of_equiv path_connected_space_iff_path_component*)

qed

show ?thesis

using *assms False* **by** (auto simp: *contractible_space homotopic_with_def* *)

qed (simp add: *path_connected_space_topspace_empty*)

lemma *contractible_imp_connected_space*:

contractible_space $X \implies \text{connected_space } X$

by (simp add: *contractible_imp_path_connected_space path_connected_imp_connected_space*)

lemma *contractible_space_alt*:

contractible_space $X \iff (\forall a \in \text{topspace } X. \text{homotopic_with } (\lambda x. \text{True}) X X \text{ id } (\lambda x. a))$ (**is** ?lhs = ?rhs)

proof

assume X : ?lhs

then obtain a **where** a : *homotopic_with* $(\lambda x. \text{True}) X X \text{ id } (\lambda x. a)$

by (*auto simp: contractible_space_def*)

show ?rhs

proof

show *homotopic_with* $(\lambda x. \text{True}) X X \text{ id } (\lambda x. b)$ **if** $b \in \text{topspace } X$ **for** b

proof (*rule homotopic_with_trans [OF a]*)

show *homotopic_with* $(\lambda x. \text{True}) X X (\lambda x. a) (\lambda x. b)$

using *homotopic_constant_maps path_connected_space_imp_path_component_of*

by (*metis X a contractible_imp_path_connected_space homotopic_with_sym*

homotopic_with_trans path_component_of_equiv *that*)

qed

qed

next

assume R : ?rhs

then show ?lhs

using *contractible_space_def* **by** *fastforce*

qed

lemma *compose_const [simp]*: $f \circ (\lambda x. a) = (\lambda x. f a) (\lambda x. a) \circ g = (\lambda x. a)$

by (*simp_all add: o_def*)

lemma *nullhomotopic_through_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** g : *continuous_map* $Y Z g$ **and** Y : *contractible_space* Y

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Z (g \circ f) (\lambda x. c)$

proof –

obtain b **where** b : *homotopic_with* $(\lambda x. \text{True}) Y Y \text{ id } (\lambda x. b)$

using Y **by** (*auto simp: contractible_space_def*)

show *thesis*

using *homotopic_with_compose_continuous_map_right*

[*OF homotopic_with_compose_continuous_map_left [OF b g] f*]

by (*force simp: that*)

qed

lemma *nullhomotopic_into_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** Y : *contractible_space* Y

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Y f (\lambda x. c)$

using *nullhomotopic_through_contractible_space [OF f _ Y]*

by (*metis continuous_map_id id_comp*)

lemma *nullhomotopic_from_contractible_space*:

assumes f : *continuous_map* $X Y f$ **and** X : *contractible_space* X

obtains c **where** *homotopic_with* $(\lambda h. \text{True}) X Y f (\lambda x. c)$

using *nullhomotopic_through_contractible_space* [*OF _ f X*]
by (*metis comp_id continuous_map_id*)

lemma *homotopy_dominated_contractibility*:

assumes *f*: *continuous_map* *X Y f* **and** *g*: *continuous_map* *Y X g*
and *hom*: *homotopic_with* ($\lambda x. \text{True}$) *Y Y* (*f* \circ *g*) *id* **and** *X*: *contractible_space*
X

shows *contractible_space* *Y*

proof –

obtain *c* **where** *c*: *homotopic_with* ($\lambda h. \text{True}$) *X Y f* ($\lambda x. c$)

using *nullhomotopic_from_contractible_space* [*OF f X*] .

have *homotopic_with* ($\lambda x. \text{True}$) *Y Y* (*f* \circ *g*) ($\lambda x. c$)

using *homotopic_with_compose_continuous_map_right* [*OF c g*] **by** *fastforce*

then have *homotopic_with* ($\lambda x. \text{True}$) *Y Y* *id* ($\lambda x. c$)

using *homotopic_with_trans* [*OF _ hom*] *homotopic_with_symD* **by** *blast*

then show *?thesis*

unfolding *contractible_space_def* ..

qed

lemma *homotopy_equivalent_space_contractibility*:

X *homotopy_equivalent_space* *Y* \implies (*contractible_space* *X* \longleftrightarrow *contractible_space*
Y)

unfolding *homotopy_equivalent_space_def*

by (*blast intro: homotopy_dominated_contractibility*)

lemma *homeomorphic_space_contractibility*:

X *homeomorphic_space* *Y*

\implies (*contractible_space* *X* \longleftrightarrow *contractible_space* *Y*)

by (*simp add: homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_contractibility*)

lemma *homotopic_through_contractible_space*:

continuous_map *X Y f* \wedge

continuous_map *X Y f'* \wedge

continuous_map *Y Z g* \wedge

continuous_map *Y Z g'* \wedge

contractible_space *Y* \wedge *path_connected_space* *Z*

\implies *homotopic_with* ($\lambda h. \text{True}$) *X Z* (*g* \circ *f*) (*g'* \circ *f'*)

using *nullhomotopic_through_contractible_space* [*of X Y f Z g*]

using *nullhomotopic_through_contractible_space* [*of X Y f' Z g'*]

by (*smt (verit) continuous_map_const homotopic_constant_maps homotopic_with_imp_continuous_map*
homotopic_with_symD homotopic_with_trans path_connected_space_imp_path_component_of)

lemma *homotopic_from_contractible_space*:

continuous_map *X Y f* \wedge *continuous_map* *X Y g* \wedge

contractible_space *X* \wedge *path_connected_space* *Y*

\implies *homotopic_with* ($\lambda x. \text{True}$) *X Y f g*

by (*metis comp_id continuous_map_id homotopic_through_contractible_space*)

lemma *homotopic_into_contractible_space*:

```

    continuous_map X Y f ∧ continuous_map X Y g ∧
      contractible_space Y
    ⇒ homotopic_with (λx. True) X Y f g
  by (metis continuous_map_id contractible_imp_path_connected_space homotopic_through_contractible_space id_comp)

lemma contractible_eq_homotopy_equivalent_singleton_subtopology:
  contractible_space X ↔
    X = trivial_topology ∨ (∃ a ∈ topspace X. X homotopy_equivalent_space
  (subtopology X {a}))(is ?lhs = ?rhs)
proof (cases X = trivial_topology)
  case False
  show ?thesis
  proof
    assume ?lhs
    then obtain a where a: homotopic_with (λx. True) X X id (λx. a)
    by (auto simp: contractible_space_def)
    then have a ∈ topspace X
    by (metis False continuous_map_const homotopic_with_imp_continuous_maps)
    then have homotopic_with (λx. True) (subtopology X {a}) (subtopology X
  {a}) id (λx. a)
    using connectedin_absolute connectedin_sing contractible_space_alt contractible_space_subtopology_singleton by fastforce
    then have X homotopy_equivalent_space subtopology X {a}
    unfolding homotopy_equivalent_space_def using ⟨a ∈ topspace X⟩
    by (metis (full_types) a comp_id continuous_map_const continuous_map_id_subt
  empty_subsetI homotopic_with_symD
    id_comp insertI1 insert_subset topspace_subtopology_subset)
    with ⟨a ∈ topspace X⟩ show ?rhs
    by blast
  next
    assume ?rhs
    then show ?lhs
    by (meson False contractible_space_subtopology_singleton homotopy_equivalent_space_contractibility)
  qed
qed (simp add: contractible_space_empty)

lemma contractible_space_retraction_map_image:
  assumes retraction_map X Y f and X: contractible_space X
  shows contractible_space Y
proof –
  obtain g where f: continuous_map X Y f and g: continuous_map Y X g and
  fg: ∀ y ∈ topspace Y. f(g y) = y
  using assms by (auto simp: retraction_map_def retraction_maps_def)
  obtain a where a: homotopic_with (λx. True) X X id (λx. a)
  using X by (auto simp: contractible_space_def)
  have homotopic_with (λx. True) Y Y id (λx. f a)
  proof (rule homotopic_with_eq)
    show homotopic_with (λx. True) Y Y (f ∘ id ∘ g) (f ∘ (λx. a) ∘ g)

```

```

    using fg a homotopic_with_compose_continuous_map_left homotopic_with_compose_continuous
  by metis
  qed (use fg in auto)
  then show ?thesis
    unfolding contractible_space_def by blast
  qed

```

```

lemma contractible_space_prod_topology:
  contractible_space (prod_topology X Y)  $\longleftrightarrow$ 
  X = trivial_topology  $\vee$  Y = trivial_topology  $\vee$  contractible_space X  $\wedge$  con-
  tractible_space Y
proof (cases X = trivial_topology  $\vee$  Y = trivial_topology)
  case True
  then have (prod_topology X Y) = trivial_topology
    by simp
  then show ?thesis
    by (auto simp: contractible_space_empty)
  next
  case False
  have contractible_space (prod_topology X Y)  $\longleftrightarrow$  contractible_space X  $\wedge$  con-
  tractible_space Y
  proof safe
    assume XY: contractible_space (prod_topology X Y)
    with False have retraction_map (prod_topology X Y) X fst
      by (auto simp: contractible_space False retraction_map_fst)
    then show contractible_space X
      by (rule contractible_space_retraction_map_image [OF _ XY])
    have retraction_map (prod_topology X Y) Y snd
      using False XY by (auto simp: contractible_space False retraction_map_snd)
    then show contractible_space Y
      by (rule contractible_space_retraction_map_image [OF _ XY])
  next
  assume contractible_space X and contractible_space Y
  with False obtain a b
    where a  $\in$  topspace X and a: homotopic_with ( $\lambda x.$  True) X X id ( $\lambda x.$  a)
      and b  $\in$  topspace Y and b: homotopic_with ( $\lambda x.$  True) Y Y id ( $\lambda x.$  b)
    by (auto simp: contractible_space)
  with False show contractible_space (prod_topology X Y)
    apply (simp add: contractible_space)
    apply (rule_tac x=a in bexI)
    apply (rule_tac x=b in bexI)
    using homotopic_with_prod_topology [OF a b]
    apply (metis (no_types, lifting) case_prod_Pair case_prod_beta' eq_id_iff)
    apply auto
  done
  qed
  with False show ?thesis
    by auto
  qed

```



```

lemma contractible_space_product_topology:
  contractible_space(product_topology X I)  $\longleftrightarrow$ 
    (product_topology X I) = trivial_topology  $\vee$  ( $\forall i \in I. \text{contractible\_space}(X i)$ )
proof (cases (product_topology X I) = trivial_topology)
  case False
  have 1: contractible_space (X i)
    if XI: contractible_space (product_topology X I) and  $i \in I$ 
    for i
  proof (rule contractible_space_retraction_map_image [OF XI])
    show retraction_map (product_topology X I) (X i) ( $\lambda x. x i$ )
      using False by (simp add: retraction_map_product_projection  $\langle i \in I \rangle$ )
  qed
  have 2: contractible_space (product_topology X I)
    if  $x \in \text{topspace}$  (product_topology X I) and cs:  $\forall i \in I. \text{contractible\_space}(X i)$ 
    for  $x :: 'a \Rightarrow 'b$ 
  proof -
    obtain f where  $f: \bigwedge i. i \in I \implies \text{homotopic\_with}(\lambda x. \text{True})(X i)(X i) \text{id}(\lambda x. f i)$ 
    using cs unfolding contractible_space_def by metis
    have homotopic_with ( $\lambda x. \text{True}$ )
      (product_topology X I) (product_topology X I) id ( $\lambda x. \text{restrict } f I$ )
    by (rule homotopic_with_eq [OF homotopic_with_product_topology [OF f]])
    (auto)
    then show ?thesis
      by (auto simp: contractible_space_def)
  qed
  show ?thesis
    using False 1 2 by (meson equals0I subtopology_eq_discrete_topology_empty)
qed auto

```

```

lemma contractible_space_subtopology_euclideanreal [simp]:
  contractible_space(subtopology euclideanreal S)  $\longleftrightarrow$  is_interval S
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have path_connectedin (subtopology euclideanreal S) S
    using contractible_imp_path_connected_space path_connectedin_topspace path_connectedin_absolute
    by (simp add: contractible_imp_path_connected)
  then show ?rhs
    by (simp add: is_interval_path_connected_1)
next
  assume ?rhs
  then have convex S
    by (simp add: is_interval_convex_1)
  show ?lhs

```

```

proof (cases S = {})
  case False
  then obtain z where z ∈ S
  by blast
  show ?thesis
  unfolding contractible_space_def homotopic_with_def
  proof (intro exI conjI allI)
  note § = convexD [OF ⟨convex S⟩, simplified]
  show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set S))
(top_of_set S)
      (λ(t,x). (1 - t) * x + t * z)
  using ⟨z ∈ S⟩
  by (auto simp: case_prod_unfold intro!: continuous_intros §)
qed auto
qed (simp add: contractible_space_empty)
qed

```

corollary *contractible_space_euclideanreal: contractible_space euclideanreal*
proof –

```

  have contractible_space (subtopology euclideanreal UNIV)
  using contractible_space_subtopology_euclideanreal by blast
  then show ?thesis
  by simp
qed

```

abbreviation *homotopy_eqv* :: 'a::topological_space set ⇒ 'b::topological_space set ⇒ bool

(**infix** homotopy'_eqv 50)

where S homotopy_eqv T ≡ top_of_set S homotopy_equivalent_space top_of_set T

lemma *homeomorphic_imp_homotopy_eqv: S homeomorphic T ⇒ S homotopy_eqv T*

```

unfolding homeomorphic_def homeomorphism_def homotopy_equivalent_space_def
by (metis continuous_map_subtopology_eu homotopic_with_id2 openin_imp_subset
openin_subtopology_self topspace_euclidean_subtopology)

```

lemma *homotopy_eqv_inj_linear_image:*

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space

assumes linear f inj f

shows (f ' S) homotopy_eqv S

by (metis assms homeomorphic_sym homeomorphic_imp_homotopy_eqv linear_homeomorphic_image)

lemma *homotopy_eqv_translation:*

fixes S :: 'a::real_normed_vector set

shows (+) a ' S homotopy_eqv S

using *homeomorphic_imp_homotopy_eqv homeomorphic_translation homeomorphic_sym* **by** *blast*

lemma *homotopy_eqv_homotopic_triviality_imp*:

fixes $S :: 'a::\text{real_normed_vector_set}$
and $T :: 'b::\text{real_normed_vector_set}$
and $U :: 'c::\text{real_normed_vector_set}$
assumes $S \text{ homotopy_eqv } T$
and $f: \text{continuous_on } U \ f \ f \in U \rightarrow T$
and $g: \text{continuous_on } U \ g \ g \in U \rightarrow T$
and $\text{homUS}: \bigwedge f \ g. \llbracket \text{continuous_on } U \ f; f \in U \rightarrow S; \text{continuous_on } U \ g; g \in U \rightarrow S \rrbracket$
 $\implies \text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ S \ f \ g$
shows $\text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ T \ f \ g$
proof –
obtain $h \ k$ **where** $h: \text{continuous_on } S \ h \ h \in S \rightarrow T$
and $k: \text{continuous_on } T \ k \ k \in T \rightarrow S$
and $\text{hom}: \text{homotopic_with_canon } (\lambda x. \text{True}) \ S \ S \ (k \circ h) \ \text{id}$
 $\text{homotopic_with_canon } (\lambda x. \text{True}) \ T \ T \ (h \circ k) \ \text{id}$
using *assms* **by** (*force simp: homotopy_equivalent_space_def image_subset_iff_funcset*)
have $\text{homotopic_with_canon } (\lambda f. \text{True}) \ U \ S \ (k \circ f) \ (k \circ g)$
proof (*rule homUS*)
show $\text{continuous_on } U \ (k \circ f) \ \text{continuous_on } U \ (k \circ g)$
using *continuous_on_compose continuous_on_subset f g k* **by** (*metis funcset_image*)
qed (*use f g k in <(force simp: o_def)>*)
then have $\text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ T \ (h \circ (k \circ f)) \ (h \circ (k \circ g))$
by (*simp add: h homotopic_with_compose_continuous_map_left image_subset_iff_funcset*)
moreover have $\text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ T \ (h \circ k \circ f) \ (\text{id} \circ f)$
by (*rule homotopic_with_compose_continuous_right [where X=T and Y=T]*);
simp add: hom f
moreover have $\text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ T \ (h \circ k \circ g) \ (\text{id} \circ g)$
by (*rule homotopic_with_compose_continuous_right [where X=T and Y=T]*);
simp add: hom g
ultimately show $\text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ T \ f \ g$
unfolding *o_assoc*
by (*metis homotopic_with_trans homotopic_with_sym id_comp*)
qed

lemma *homotopy_eqv_homotopic_triviality*:

fixes $S :: 'a::\text{real_normed_vector_set}$
and $T :: 'b::\text{real_normed_vector_set}$
and $U :: 'c::\text{real_normed_vector_set}$
assumes $S \text{ homotopy_eqv } T$
shows $(\forall f \ g. \text{continuous_on } U \ f \wedge f \in U \rightarrow S \wedge \text{continuous_on } U \ g \wedge g \in U \rightarrow S$
 $\implies \text{homotopic_with_canon } (\lambda x. \text{True}) \ U \ S \ f \ g) \longleftrightarrow$
 $(\forall f \ g. \text{continuous_on } U \ f \wedge f \in U \rightarrow T \wedge \text{continuous_on } U \ g \wedge g \in U \rightarrow T$

```

      → homotopic_with_canon (λx. True) U T f g)
    (is ?lhs = ?rhs)
  proof
    assume ?lhs
    then show ?rhs
      by (metis assms homotopy_eqv_homotopic_triviality_imp)
  next
    assume ?rhs
    moreover
    have T homotopy_eqv S
      using assms homotopy_equivalent_space_sym by blast
    ultimately show ?lhs
      by (blast intro: homotopy_eqv_homotopic_triviality_imp)
  qed

```

lemma *homotopy_eqv_cohomotopic_triviality_null_imp:*

```

  fixes S :: 'a::real_normed_vector set
    and T :: 'b::real_normed_vector set
    and U :: 'c::real_normed_vector set
  assumes S homotopy_eqv T
    and f: continuous_on T f f ∈ T → U
    and homSU: ∧f. [[continuous_on S f; f ∈ S → U]]
      ⇒ ∃ c. homotopic_with_canon (λx. True) S U f (λx. c)
  obtains c where homotopic_with_canon (λx. True) T U f (λx. c)
  proof -
    obtain h k where h: continuous_on S h h ∈ S → T
      and k: continuous_on T k k ∈ T → S
      and hom: homotopic_with_canon (λx. True) S S (k ∘ h) id
        homotopic_with_canon (λx. True) T T (h ∘ k) id
    using assms by (force simp: homotopy_equivalent_space_def image_subset_iff_funcset)
    obtain c where homotopic_with_canon (λx. True) S U (f ∘ h) (λx. c)
    proof (rule exE [OF homSU])
      show continuous_on S (f ∘ h)
        by (metis continuous_on_compose continuous_on_subset f h funcset_image)
    qed (use f h in force)
    then have homotopic_with_canon (λx. True) T U ((f ∘ h) ∘ k) ((λx. c) ∘ k)
      by (rule homotopic_with_compose_continuous_right [where X=S]) (use k in auto)
    moreover have homotopic_with_canon (λx. True) T U (f ∘ id) (f ∘ (h ∘ k))
      by (rule homotopic_with_compose_continuous_left [where Y=T])
        (use f in ‹auto simp: hom homotopic_with_symD›)
    ultimately show ?thesis
      using that homotopic_with_trans by (fastforce simp: o_def)
  qed

```

lemma *homotopy_eqv_cohomotopic_triviality_null:*

```

  fixes S :: 'a::real_normed_vector set
    and T :: 'b::real_normed_vector set

```

```

and U :: 'c::real_normed_vector set
assumes S homotopy_eqv T
shows ( $\forall f. \text{continuous\_on } S f \wedge f \in S \rightarrow U$ 
   $\rightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S U f (\lambda x. c))$ )  $\longleftrightarrow$ 
  ( $\forall f. \text{continuous\_on } T f \wedge f \in T \rightarrow U$ 
   $\rightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) T U f (\lambda x. c))$ )
by (rule iffI; metis assms homotopy_eqv_cohomotopic_triviality_null_imp_homotopy_equivalent_space_sym)

```

Similar to the proof above

```

lemma homotopy_eqv_homotopic_triviality_null_imp:
fixes S :: 'a::real_normed_vector set
and T :: 'b::real_normed_vector set
and U :: 'c::real_normed_vector set
assumes S homotopy_eqv T
and f: continuous_on U f f  $\in U \rightarrow T$ 
and homSU:  $\bigwedge f. \llbracket \text{continuous\_on } U f; f \in U \rightarrow S \rrbracket$ 
   $\implies \exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U S f (\lambda x. c)$ 
shows  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U T f (\lambda x. c)$ 
proof -
obtain h k where h: continuous_on S h h  $\in S \rightarrow T$ 
and k: continuous_on T k k  $\in T \rightarrow S$ 
and hom: homotopic_with_canon  $(\lambda x. \text{True}) S S (k \circ h)$  id
  homotopic_with_canon  $(\lambda x. \text{True}) T T (h \circ k)$  id
using assms by (force simp: homotopy_equivalent_space_def image_subset_iff_funcset)
obtain c::'a where homotopic_with_canon  $(\lambda x. \text{True}) U S (k \circ f)$   $(\lambda x. c)$ 
proof (rule exE [OF homSU [of k  $\circ$  f]])
show continuous_on U (k  $\circ$  f)
using continuous_on_compose continuous_on_subset f k by (metis funcset_image)
qed (use f k in force)
then have homotopic_with_canon  $(\lambda x. \text{True}) U T (h \circ (k \circ f)) (h \circ (\lambda x. c))$ 
by (rule homotopic_with_compose_continuous_left [where Y=S]) (use h in auto)
moreover have homotopic_with_canon  $(\lambda x. \text{True}) U T (id \circ f) ((h \circ k) \circ f)$ 
by (rule homotopic_with_compose_continuous_right [where X=T])
  (use f in  $\langle$ auto simp: hom_homotopic_with_symD $\rangle$ )
ultimately show ?thesis
using homotopic_with_trans by (fastforce simp: o_def)
qed

```

```

lemma homotopy_eqv_homotopic_triviality_null:
fixes S :: 'a::real_normed_vector set
and T :: 'b::real_normed_vector set
and U :: 'c::real_normed_vector set
assumes S homotopy_eqv T
shows ( $\forall f. \text{continuous\_on } U f \wedge f \in U \rightarrow S$ 
   $\rightarrow (\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) U S f (\lambda x. c))$ )  $\longleftrightarrow$ 
  ( $\forall f. \text{continuous\_on } U f \wedge f \in U \rightarrow T$ 

```

$\longrightarrow (\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) U T f (\lambda x. c))$

by (rule iffI; metis assms homotopy_eqv_homotopic_triviality_null_imp_homotopy_equivalent_space_sym)

lemma *homotopy_eqv_contractible_sets*:
fixes $S :: 'a::\text{real_normed_vector_set}$
and $T :: 'b::\text{real_normed_vector_set}$
assumes *contractible S contractible T* $S = \{\}$ \longleftrightarrow $T = \{\}$
shows $S \text{ homotopy_eqv } T$
proof (cases $S = \{\}$)
case *True* **with** *assms* **show** *?thesis*
using *homeomorphic_imp_homotopy_eqv* **by** *fastforce*
next
case *False*
with *assms* **obtain** $a b$ **where** $a \in S$ $b \in T$
by *auto*
then **show** *?thesis*
unfolding *homotopy_equivalent_space_def*
apply (rule_tac $x = \lambda x. b$ **in** *exI*, rule_tac $x = \lambda x. a$ **in** *exI*)
apply (*intro* *assms* *conjI* *continuous_on_id'* *homotopic_into_contractible*;
force)
done
qed

lemma *homotopy_eqv_empty1* [*simp*]:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $S \text{ homotopy_eqv } (\{\}::'b::\text{real_normed_vector_set}) \longleftrightarrow S = \{\}$ (**is** *?lhs = ?rhs*)
proof
assume *?lhs* **then** **show** *?rhs*
by (*metis* *continuous_map_subtopology_eu* *empty_iff_equalityI* *homotopy_equivalent_space_def* *image_subset_iff* *subsetI*)
qed (*use* *homeomorphic_imp_homotopy_eqv* **in** *force*)

lemma *homotopy_eqv_empty2* [*simp*]:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $(\{\}::'b::\text{real_normed_vector_set}) \text{ homotopy_eqv } S \longleftrightarrow S = \{\}$
using *homotopy_equivalent_space_sym* *homotopy_eqv_empty1* **by** *blast*

lemma *homotopy_eqv_contractibility*:
fixes $S :: 'a::\text{real_normed_vector_set}$ **and** $T :: 'b::\text{real_normed_vector_set}$
shows $S \text{ homotopy_eqv } T \implies (\text{contractible } S \longleftrightarrow \text{contractible } T)$
by (*meson* *contractible_space_top_of_set* *homotopy_equivalent_space_contractibility*)

lemma *homotopy_eqv_sing*:
fixes $S :: 'a::\text{real_normed_vector_set}$ **and** $a :: 'b::\text{real_normed_vector}$
shows $S \text{ homotopy_eqv } \{a\} \longleftrightarrow S \neq \{\} \wedge \text{contractible } S$
by (*metis* *contractible_sing_empty_not_insert* *homotopy_eqv_contractibility* *homotopy_eqv_contractible_sets* *homotopy_eqv_empty2*)

lemma *homeomorphic_contractible_eq*:
fixes $S :: 'a::real_normed_vector_set$ **and** $T :: 'b::real_normed_vector_set$
shows S *homeomorphic* $T \implies (contractible\ S \longleftrightarrow contractible\ T)$
by (*simp add: homeomorphic_imp_homotopy_eqv homotopy_eqv_contractibility*)

lemma *homeomorphic_contractible*:
fixes $S :: 'a::real_normed_vector_set$ **and** $T :: 'b::real_normed_vector_set$
shows $\llbracket contractible\ S; S$ *homeomorphic* $T \rrbracket \implies contractible\ T$
by (*metis homeomorphic_contractible_eq*)

6.4.25 Misc other results

lemma *bounded_connected_Compl_real*:
fixes $S :: real\ set$
assumes *bounded* S **and** *conn*: *connected*($- S$)
shows $S = \{\}$
proof –
obtain $a\ b$ **where** $S \subseteq box\ a\ b$
by (*meson assms bounded_subset_box_symmetric*)
then have $a \notin S\ b \notin S$
by *auto*
then have $\forall x. a \leq x \wedge x \leq b \longrightarrow x \in - S$
by (*meson Compl_iff_conn_connected_iff_interval*)
then show *?thesis*
using $\langle S \subseteq box\ a\ b \rangle$ **by** *auto*
qed

corollary *bounded_path_connected_Compl_real*:
fixes $S :: real\ set$
assumes *bounded* S *path_connected*($- S$) **shows** $S = \{\}$
by (*simp add: assms bounded_connected_Compl_real path_connected_imp_connected*)

lemma *bounded_connected_Compl_1*:
fixes $S :: 'a::\{euclidean_space\}\ set$
assumes *bounded* S **and** *conn*: *connected*($- S$) **and** $1: DIM('a) = 1$
shows $S = \{\}$
proof –
have $DIM('a) = DIM(real)$
by (*simp add: 1*)
then obtain $f::'a \Rightarrow real$ **and** g
where *linear* $f \wedge x. norm(f\ x) = norm\ x$ **and** $fg: \wedge x. g(f\ x) = x \wedge y. f(g\ y) = y$
by (*rule isomorphisms_UNIV_UNIV*) *blast*
with $\langle bounded\ S \rangle$ **have** *bounded* $(f\ ' S)$
using *bounded_linear_image linear_linear* **by** *blast*
have *bij* f **by** (*metis fg bijI'*)
have *connected* $(f\ ' (-S))$
using *connected_linear_image assms* $\langle linear\ f \rangle$ **by** *blast*
moreover have $f\ ' (-S) = - (f\ ' S)$

```

    by (simp add: ⟨bij f⟩ bij_image_Compl_eq)
  finally have connected (− (f ‘ S))
    by simp
  then have f ‘ S = {}
    using ⟨bounded (f ‘ S)⟩ bounded_connected_Compl_real by blast
  then show ?thesis
    by blast
qed

```

```

lemma connected_card_eq_iff_nontrivial:
  fixes S :: 'a::metric_space set
  shows connected S  $\implies$  uncountable S  $\longleftrightarrow$   $\neg(\exists a. S \subseteq \{a\})$ 
  by (metis connected_uncountable finite.emptyI finite.insertI rev_finite_subset
  singleton_iff subsetI uncountable_infinite)

```

```

lemma connected_finite_iff_singleton:
  fixes S :: 'a::metric_space set
  assumes connected S
  shows finite S  $\longleftrightarrow$  S = {}  $\vee$   $(\exists a. S = \{a\})$ 
  using assms connected_uncountable countable_finite by blast

```

6.4.26 Some simple positive connection theorems

```

proposition path_connected_convex_diff_countable:
  fixes U :: 'a::euclidean_space set
  assumes convex U  $\wedge$  collinear U countable S
  shows path_connected(U − S)
proof (clarsimp simp: path_connected_def)
  fix a b
  assume a  $\in$  U a  $\notin$  S b  $\in$  U b  $\notin$  S
  let ?m = midpoint a b
  show  $\exists g. \text{path } g \wedge \text{path\_image } g \subseteq U - S \wedge \text{pathstart } g = a \wedge \text{pathfinish } g = b$ 
  proof (cases a = b)
    case True
    then show ?thesis
      by (metis DiffI ⟨a  $\in$  U⟩ ⟨a  $\notin$  S⟩ path_component_def path_component_refl)
    next
    case False
    then have a  $\neq$  ?m b  $\neq$  ?m
      using midpoint_eq_endpoint by fastforce+
    have ?m  $\in$  U
      using ⟨a  $\in$  U⟩ ⟨b  $\in$  U⟩ ⟨convex U⟩ convex_contains_segment by force
    obtain c where c  $\in$  U and nc_abc:  $\neg$  collinear {a,b,c}
      by (metis False ⟨a  $\in$  U⟩ ⟨b  $\in$  U⟩ ⟨ $\neg$  collinear U⟩ collinear_triples insert_absorb)
    have ncoll_mca:  $\neg$  collinear {?m,c,a}
      by (metis (full_types) ⟨a  $\neq$  ?m⟩ collinear_3_trans collinear_midpoint insert_commute nc_abc)
    have ncoll_mcb:  $\neg$  collinear {?m,c,b}

```



```

    by (metis (full_types) ‹b ≠ ?m› collinear_3_trans collinear_midpoint insert_commute nc_abc)
  have c ≠ ?m
  by (metis collinear_midpoint insert_commute nc_abc)
  then have closed_segment ?m c ⊆ U
  by (simp add: ‹c ∈ U› ‹?m ∈ U› ‹convex U› closed_segment_subset)
  then obtain z where z: z ∈ closed_segment ?m c
    and disjS: (closed_segment a z ∪ closed_segment z b) ∩ S = {}
  proof -
    have False if closed_segment ?m c ⊆ {z. (closed_segment a z ∪ closed_segment z b) ∩ S ≠ {}}
  proof -
    have closb: closed_segment ?m c ⊆
      {z ∈ closed_segment ?m c. closed_segment a z ∩ S ≠ {}} ∪ {z ∈ closed_segment ?m c. closed_segment z b ∩ S ≠ {}}
    using that by blast
    have *: countable {z ∈ closed_segment ?m c. closed_segment z u ∩ S ≠ {}}
      if u ∈ U u ∉ S and ncoll: ¬ collinear {?m, c, u} for u
    proof -
      have **: False if x1: x1 ∈ closed_segment ?m c and x2: x2 ∈ closed_segment ?m c
        and x1 ≠ x2 x1 ≠ u
        and w: w ∈ closed_segment x1 u w ∈ closed_segment x2 u
        and w ∈ S for x1 x2 w
      proof -
        have x1 ∈ affine_hull {?m,c} x2 ∈ affine_hull {?m,c}
        using segment_as_ball x1 x2 by auto
        then have coll_x1: collinear {x1, ?m, c} and coll_x2: collinear {?m, c, x2}
      by (simp_all add: affine_hull_3_imp_collinear) (metis affine_hull_3_imp_collinear insert_commute)
      have ¬ collinear {x1, u, x2}
      proof
        assume collinear {x1, u, x2}
        then have collinear {?m, c, u}
          by (metis (full_types) ‹c ≠ ?m› coll_x1 coll_x2 collinear_3_trans insert_commute ncoll ‹x1 ≠ x2›)
        with ncoll show False ..
      qed
      then have closed_segment x1 u ∩ closed_segment u x2 = {u}
        by (blast intro!: Int_closed_segment)
      then have w = u
        using closed_segment_commute w by auto
      show ?thesis
        using ‹u ∉ S› ‹w = u› that(7) by auto
    qed
    then have disj: disjoint ((⋃ z ∈ closed_segment ?m c. {closed_segment z u ∩ S}))
      by (fastforce simp: pairwise_def disjnt_def)

```

```

      have cou: countable (( $\bigcup z \in \text{closed\_segment } ?m \ c. \{\text{closed\_segment } z \ u \cap S\}$ ) -  $\{\{\}\}$ )
    apply (rule pairwise_disjnt_countable_Union [OF _ pairwise_subset [OF disj]])
    apply (rule countable_subset [OF _  $\langle \text{countable } S \rangle$ ], auto)
    done
    define f where f  $\equiv \lambda X. (\text{THE } z. z \in \text{closed\_segment } ?m \ c \wedge X = \text{closed\_segment } z \ u \cap S)$ 
    show ?thesis
    proof (rule countable_subset [OF _ countable_image [OF cou, where f=f]], clarify)
      fix x
      assume x:  $x \in \text{closed\_segment } ?m \ c \ \text{closed\_segment } x \ u \cap S \neq \{\}$ 
      show  $x \in f \ ' (\bigcup z \in \text{closed\_segment } ?m \ c. \{\text{closed\_segment } z \ u \cap S\}) - \{\{\}\}$ 
    proof (rule_tac x=closed_segment x u  $\cap S$  in image_eqI)
      show  $x = f (\text{closed\_segment } x \ u \cap S)$ 
    unfolding f_def
    by (rule the_equality [symmetric]) (use x in  $\langle \text{auto dest: **} \rangle$ )
    qed (use x in auto)
  qed
qed
have uncountable (closed_segment ?m c)
by (metis  $\langle c \neq ?m \rangle$  uncountable_closed_segment)
then show False
using closb * [OF  $\langle a \in U \rangle \langle a \notin S \rangle$  ncoll_mca] * [OF  $\langle b \in U \rangle \langle b \notin S \rangle$  ncoll_mcb]
by (simp add: closed_segment_commute countable_subset)
qed
then show ?thesis
by (force intro: that)
qed
show ?thesis
proof (intro exI conjI)
  have path_image (linepath a z +++ linepath z b)  $\subseteq U$ 
  by (metis  $\langle a \in U \rangle \langle b \in U \rangle \langle \text{closed\_segment } ?m \ c \subseteq U \rangle z \langle \text{convex } U \rangle$ 
closed_segment_subset contra_subsetD path_image_linepath subset_path_image_join)
  with disjS show path_image (linepath a z +++ linepath z b)  $\subseteq U - S$ 
  by (force simp: path_image_join)
qed auto
qed
qed

```

corollary *connected_convex_diff_countable*:

fixes $U :: 'a::\text{euclidean_space}$ set

assumes $\text{convex } U \neg \text{collinear } U$ countable S

shows $\text{connected}(U - S)$

by (simp add: *assms path_connected_convex_diff_countable path_connected_imp_connected*)

```

lemma path_connected_punctured_convex:
  assumes convex S and aff: aff_dim S  $\neq$  1
  shows path_connected(S - {a})
proof -
  consider aff_dim S = -1 | aff_dim S = 0 | aff_dim S  $\geq$  2
  using assms aff_dim_geq [of S] by linarith
  then show ?thesis
  proof cases
    assume aff_dim S = -1
    then show ?thesis
      by (metis aff_dim_empty empty_Diff path_connected_empty)
  next
    assume aff_dim S = 0
    then show ?thesis
      by (metis aff_dim_eq_0 Diff_cancel Diff_empty Diff_insert0 convex_empty
convex_imp_path_connected path_connected_singleton singletonD)
  next
    assume ge2: aff_dim S  $\geq$  2
    then have  $\neg$  collinear S
    proof (clarsimp simp: collinear_affine_hull)
      fix u v
      assume S  $\subseteq$  affine_hull {u, v}
      then have aff_dim S  $\leq$  aff_dim {u, v}
        by (metis (no_types) aff_dim_affine_hull aff_dim_subset)
      with ge2 show False
    by (metis (no_types) aff_dim_2 antisym aff_not_numerical_le_zero one_le_numerical
order_trans)
  qed
  moreover have countable {a}
    by simp
  ultimately show ?thesis
    by (metis path_connected_convex_diff_countable [OF  $\langle$ convex S $\rangle$ ])
  qed
qed

```

```

lemma connected_punctured_convex:
  shows  $\llbracket$ convex S; aff_dim S  $\neq$  1 $\rrbracket \implies$  connected(S - {a})
  using path_connected_imp_connected path_connected_punctured_convex by blast

```

```

lemma path_connected_complement_countable:
  fixes S :: 'a::euclidean_space set
  assumes 2  $\leq$  DIM('a) countable S
  shows path_connected(- S)
proof -
  have  $\neg$  collinear (UNIV::'a set)
    using assms by (auto simp: collinear_aff_dim [of UNIV :: 'a set])
  then have path_connected(UNIV - S)
    by (simp add:  $\langle$ countable S $\rangle$  path_connected_convex_diff_countable)

```

then show *?thesis*
by (*simp add: Compl_eq_Diff_UNIV*)
qed

proposition *path_connected_openin_diff_countable:*

fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes *connected S and ope: openin (top_of_set (affine hull S)) S*
and $\neg \text{collinear } S$ *countable T*
shows *path_connected(S - T)*
proof (*clarsimp simp: path_connected_component*)
fix $x\ y$
assume $xy: x \in S\ x \notin T\ y \in S\ y \notin T$
show *path_component (S - T) x y*
proof (*rule connected_equivalence_relation_gen [OF <connected S>, where P = $\lambda x. x \notin T$]*)
show $\exists z. z \in U \wedge z \notin T$ **if** *opeU: openin (top_of_set S) U and $x \in U$ for U*
proof –
have *openin (top_of_set (affine hull S)) U*
using *opeU ope openin_trans* **by** *blast*
with $\langle x \in U \rangle$ **obtain** r **where** $U_{\text{sub}}: U \subseteq \text{affine hull } S$ **and** $r > 0$
and $\text{sub}U: \text{ball } x\ r \cap \text{affine hull } S \subseteq U$
by (*auto simp: openin_contains_ball*)
with $\langle x \in U \rangle$ **have** $x \in \text{ball } x\ r \cap \text{affine hull } S$
by *auto*
have $\neg S \subseteq \{x\}$
using $\langle \neg \text{collinear } S \rangle$ *collinear_subset* **by** *blast*
then obtain x' **where** $x' \neq x\ x' \in S$
by *blast*
obtain y **where** $y \neq x\ y \in \text{ball } x\ r \cap \text{affine hull } S$
proof
show $x + (r / 2 / \text{norm}(x' - x)) *_{\mathbb{R}} (x' - x) \neq x$
using $\langle x' \neq x \rangle \langle r > 0 \rangle$ **by** *auto*
show $x + (r / 2 / \text{norm}(x' - x)) *_{\mathbb{R}} (x' - x) \in \text{ball } x\ r \cap \text{affine hull } S$
using $\langle x' \neq x \rangle \langle r > 0 \rangle \langle x' \in S \rangle$ x
by (*simp add: dist_norm mem_affine_3_minus_hull_inc*)
qed
have *convex (ball x r \cap affine hull S)*
by (*simp add: affine_imp_convex convex_Int*)
with $x\ y\ \text{sub}U$ **have** *uncountable U*
by (*meson countable_subset uncountable_convex*)
then have $\neg U \subseteq T$
using $\langle \text{countable } T \rangle$ *countable_subset* **by** *blast*
then show *?thesis* **by** *blast*
qed
show $\exists U. \text{openin (top_of_set S)}\ U \wedge x \in U \wedge$
 $(\forall x \in U. \forall y \in U. x \notin T \wedge y \notin T \longrightarrow \text{path_component (S - T) } x\ y)$
if $x \in S$ **for** x
proof –

```

obtain  $r$  where  $S_{\text{sub}}: S \subseteq \text{affine hull } S$  and  $r > 0$ 
  and  $\text{sub}S: \text{ball } x \ r \cap \text{affine hull } S \subseteq S$ 
  using  $\text{ope } \langle x \in S \rangle$  by (auto simp: openin_contains_ball)
then have  $\text{conv}: \text{convex } (\text{ball } x \ r \cap \text{affine hull } S)$ 
  by (simp add: affine_imp_convex convex_Int)
have  $\neg \text{aff\_dim } (\text{affine hull } S) \leq 1$ 
  using  $\langle \neg \text{collinear } S \rangle$  collinear_aff_dim by auto
then have  $\neg \text{aff\_dim } (\text{ball } x \ r \cap \text{affine hull } S) \leq 1$ 
  by (metis (no_types, opaque_lifting) aff_dim_convex_Int_open IntI open_ball
 $\langle 0 < r \rangle$  aff_dim_affine_hull affine_affine_hull affine_imp_convex centre_in_ball
empty_iff_hull_subset inf_commute subsetCE that)
then have  $\neg \text{collinear } (\text{ball } x \ r \cap \text{affine hull } S)$ 
  by (simp add: collinear_aff_dim)
then have  $*$ :  $\text{path\_connected } ((\text{ball } x \ r \cap \text{affine hull } S) - T)$ 
  by (rule path_connected_convex_diff_countable [OF conv _ <countable T>])
have  $ST: \text{ball } x \ r \cap \text{affine hull } S - T \subseteq S - T$ 
  using  $\text{sub}S$  by auto
show ?thesis
proof (intro exI conjI)
  show  $x \in \text{ball } x \ r \cap \text{affine hull } S$ 
    using  $\langle x \in S \rangle$   $\langle r > 0 \rangle$  by (simp add: hull_inc)
  have  $\text{openin } (\text{top\_of\_set } (\text{affine hull } S)) (\text{ball } x \ r \cap \text{affine hull } S)$ 
    by (subst inf_commute) (simp add: openin_Int_open)
  then show  $\text{openin } (\text{top\_of\_set } S) (\text{ball } x \ r \cap \text{affine hull } S)$ 
    by (rule openin_subset_trans [OF _ subS Ssub])
  qed (use * path_component_trans in <auto simp: path_connected_component
path_component_of_subset [OF ST]>)
  qed
  qed (use xy path_component_trans in auto)
qed

```

corollary *connected_openin_diff_countable:*

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes  $\text{connected } S$  and  $\text{ope}: \text{openin } (\text{top\_of\_set } (\text{affine hull } S)) \ S$ 
and  $\neg \text{collinear } S$  countable T
shows  $\text{connected}(S - T)$ 
by (metis path_connected_imp_connected path_connected_openin_diff_countable
[OF assms])

```

corollary *path_connected_open_diff_countable:*

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes  $2 \leq \text{DIM}(a)$   $\text{open } S$   $\text{connected } S$  countable T
shows  $\text{path\_connected}(S - T)$ 
proof (cases S = {})
  case True
    then show ?thesis
      by (simp)
  next
    case False

```

```

show ?thesis
proof (rule path_connected_openin_diff_countable)
  show openin (top_of_set (affine hull S)) S
  by (simp add: assms hull_subset open_subset)
  show ¬ collinear S
  using assms False by (simp add: collinear_aff_dim aff_dim_open)
qed (simp_all add: assms)
qed

```

```

corollary connected_open_diff_countable:
  fixes S :: 'a::euclidean_space set
  assumes 2 ≤ DIM('a) open S connected S countable T
  shows connected(S - T)
by (simp add: assms path_connected_imp_connected path_connected_open_diff_countable)

```

6.4.27 Self-homeomorphisms shuffling points about

The theorem *homeomorphism_moving_points_exists*

```

lemma homeomorphism_moving_point_1:
  fixes a :: 'a::euclidean_space
  assumes affine T a ∈ T and u: u ∈ ball a r ∩ T
  obtains f g where homeomorphism (cball a r ∩ T) (cball a r ∩ T) f g
    f a = u ∧ x. x ∈ sphere a r ⇒ f x = x
proof -
  have nou: norm (u - a) < r and u ∈ T
  using u by (auto simp: dist_norm norm_minus_commute)
  then have 0 < r
  by (metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u)
  define f where f ≡ λx. (1 - norm(x - a) / r) *R (u - a) + x
  have *: False if eq: x + (norm y / r) *R u = y + (norm x / r) *R u
    and nou: norm u < r and yx: norm y < norm x for x y and u::'a
  proof -
  have x = y + (norm x / r - (norm y / r)) *R u
  using eq by (simp add: algebra_simps)
  then have norm x = norm (y + ((norm x - norm y) / r) *R u)
  by (metis diff_divide_distrib)
  also have ... ≤ norm y + norm(((norm x - norm y) / r) *R u)
  using norm_triangle_ineq by blast
  also have ... = norm y + (norm x - norm y) * (norm u / r)
  using yx ⟨r > 0⟩
  by (simp add: field_split_simps)
  also have ... < norm y + (norm x - norm y) * 1
  proof (subst add_less_cancel_left)
  show (norm x - norm y) * (norm u / r) < (norm x - norm y) * 1
  proof (rule mult_strict_left_mono)
  show norm u / r < 1
  using ⟨0 < r⟩ divide_less_eq_1_pos nou by blast
  qed (simp add: yx)
qed

```

```

    also have ... = norm x
      by simp
    finally show False by simp
  qed
  have inj f
    unfolding f_def
  proof (clarsimp simp: inj_on_def)
    fix x y
    assume (1 - norm (x - a) / r) *R (u - a) + x =
      (1 - norm (y - a) / r) *R (u - a) + y
    then have eq: (x - a) + (norm (y - a) / r) *R (u - a) = (y - a) + (norm
(x - a) / r) *R (u - a)
      by (auto simp: algebra_simps)
    show x=y
  proof (cases norm (x - a) = norm (y - a))
    case True
      then show ?thesis
        using eq by auto
    next
      case False
        then consider norm (x - a) < norm (y - a) | norm (x - a) > norm (y -
a)
          by linarith
        then have False
          proof cases
            case 1 show False
              using * [OF _ nou 1] eq by simp
            next
              case 2 with * [OF eq nou] show False
                by auto
          qed
        then show x=y ..
    qed
  qed
  then have inj_onf: inj_on f (cball a r ∩ T)
    using inj_on_Int by fastforce
  have contf: continuous_on (cball a r ∩ T) f
    unfolding f_def using ‹0 < r› by (intro continuous_intros) blast
  have fim: f ‘ (cball a r ∩ T) = cball a r ∩ T
  proof
    have *: norm (y + (1 - norm y / r) *R u) ≤ r if norm y ≤ r norm u < r
  for y u::'a
    proof -
      have norm (y + (1 - norm y / r) *R u) ≤ norm y + norm((1 - norm y /
r) *R u)
        using norm_triangle_ineq by blast
      also have ... = norm y + abs(1 - norm y / r) * norm u
        by simp
      also have ... ≤ r

```

```

proof –
  have  $(r - \text{norm } u) * (r - \text{norm } y) \geq 0$ 
    using that by auto
  then have  $r * \text{norm } u + r * \text{norm } y \leq r * r + \text{norm } u * \text{norm } y$ 
    by (simp add: algebra_simps)
  then show ?thesis
    using that  $\langle 0 < r \rangle$  by (simp add: abs_if field_simps)
qed
finally show ?thesis .
qed
have  $f' (cball\ a\ r) \subseteq cball\ a\ r$ 
  using * nou
  apply (clarsimp simp: dist_norm norm_minus_commute f_def)
  by (metis diff_add_eq diff_diff_add diff_diff_eq2 norm_minus_commute)
moreover have  $f' T \subseteq T$ 
  unfolding f_def using  $\langle \text{affine } T \rangle \langle a \in T \rangle \langle u \in T \rangle$ 
  by (force simp: add_commute mem_affine_3_minus)
ultimately show  $f' (cball\ a\ r \cap T) \subseteq cball\ a\ r \cap T$ 
  by blast
next
show  $cball\ a\ r \cap T \subseteq f' (cball\ a\ r \cap T)$ 
proof (clarsimp simp: dist_norm norm_minus_commute)
  fix x
  assume  $x: \text{norm } (x - a) \leq r$  and  $x \in T$ 
  have  $\exists v \in \{0..1\}. ((1 - v) * r - \text{norm } ((x - a) - v *_{\mathbb{R}} (u - a))) \cdot 1 = 0$ 
    by (rule ivt_decreasing_component_on_1) (auto simp: x_continuous_intros)
  then obtain v where  $0 \leq v \leq 1$ 
    and  $v: (1 - v) * r = \text{norm } ((x - a) - v *_{\mathbb{R}} (u - a))$ 
    by auto
  then have  $n: \text{norm } (a - (x - v *_{\mathbb{R}} (u - a))) = r - r * v$ 
    by (simp add: field_simps norm_minus_commute)
  show  $x \in f' (cball\ a\ r \cap T)$ 
  proof (rule image_eqI)
    show  $x = f (x - v *_{\mathbb{R}} (u - a))$ 
      using  $\langle r > 0 \rangle v$  by (simp add: f_def) (simp add: field_simps)
    have  $x - v *_{\mathbb{R}} (u - a) \in cball\ a\ r$ 
      using  $\langle r > 0 \rangle \langle 0 \leq v \rangle$ 
      by (simp add: dist_norm n)
    moreover have  $x - v *_{\mathbb{R}} (u - a) \in T$ 
      by (simp add: f_def  $\langle u \in T \rangle \langle x \in T \rangle$  assms mem_affine_3_minus2)
    ultimately show  $x - v *_{\mathbb{R}} (u - a) \in cball\ a\ r \cap T$ 
      by blast
  qed
qed
qed
have compact  $(cball\ a\ r \cap T)$ 
  by (simp add: affine_closed compact_Int_closed  $\langle \text{affine } T \rangle$ )
then obtain g where homeomorphism  $(cball\ a\ r \cap T) (cball\ a\ r \cap T)$  f g
  by (metis homeomorphism_compact [OF _ contf fim inj_onf])

```


then show *thesis*
apply (*rule_tac f=f in that*)
using $\langle r > 0 \rangle$ **by** (*simp_all add: f_def dist_norm norm_minus_commute*)
qed

corollary *homeomorphism_moving_point_2:*

fixes $a :: 'a::\text{euclidean_space}$
assumes *affine* T $a \in T$ **and** $u: u \in \text{ball } a \ r \cap T$ **and** $v: v \in \text{ball } a \ r \cap T$
obtains $f \ g$ **where** *homeomorphism* $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) f \ g$
 $f \ u = v \wedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies f \ x = x$
proof –
have $0 < r$
by (*metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u*)
obtain $f1 \ g1$ **where** *hom1: homeomorphism* $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) f1$
 $g1$
and $f1 \ a = u$ **and** $f1: \wedge x. x \in \text{sphere } a \ r \implies f1 \ x = x$
using *homeomorphism_moving_point_1* [*OF* $\langle \text{affine } T \rangle \langle a \in T \rangle u$] **by** *blast*
obtain $f2 \ g2$ **where** *hom2: homeomorphism* $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) f2$
 $g2$
and $f2 \ a = v$ **and** $f2: \wedge x. x \in \text{sphere } a \ r \implies f2 \ x = x$
using *homeomorphism_moving_point_1* [*OF* $\langle \text{affine } T \rangle \langle a \in T \rangle v$] **by** *blast*
show *?thesis*
proof
show *homeomorphism* $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) (f2 \circ g1) (f1 \circ g2)$
by (*metis homeomorphism_compose homeomorphism_symD hom1 hom2*)
have $g1 \ u = a$
using $\langle 0 < r \rangle \langle f1 \ a = u \rangle$ *assms hom1 homeomorphism_apply1* **by** *fastforce*
then show $(f2 \circ g1) \ u = v$
by (*simp add:* $\langle f2 \ a = v \rangle$)
show $\wedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies (f2 \circ g1) \ x = x$
using $f1 \ f2 \ hom1$ *homeomorphism_apply1* **by** *fastforce*
qed
qed

corollary *homeomorphism_moving_point_3:*

fixes $a :: 'a::\text{euclidean_space}$
assumes *affine* T $a \in T$ **and** $ST: \text{ball } a \ r \cap T \subseteq S \subseteq T$
and $u: u \in \text{ball } a \ r \cap T$ **and** $v: v \in \text{ball } a \ r \cap T$
obtains $f \ g$ **where** *homeomorphism* $S \ S \ f \ g$
 $f \ u = v \{x. \neg (f \ x = x \wedge g \ x = x)\} \subseteq \text{ball } a \ r \cap T$
proof –
obtain $f \ g$ **where** *hom: homeomorphism* $(\text{cball } a \ r \cap T) (\text{cball } a \ r \cap T) f \ g$
and $f \ u = v$ **and** $fid: \wedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies f \ x = x$
using *homeomorphism_moving_point_2* [*OF* $\langle \text{affine } T \rangle \langle a \in T \rangle u$] **by** *blast*
have $gid: \wedge x. \llbracket x \in \text{sphere } a \ r; x \in T \rrbracket \implies g \ x = x$
using $fid \ hom$ *homeomorphism_apply1* **by** *fastforce*
define ff **where** $ff \equiv \lambda x. \text{if } x \in \text{ball } a \ r \cap T \text{ then } f \ x \text{ else } x$
define gg **where** $gg \equiv \lambda x. \text{if } x \in \text{ball } a \ r \cap T \text{ then } g \ x \text{ else } x$

```

show ?thesis
proof
  show homeomorphism S S ff gg
  proof (rule homeomorphismI)
    have continuous_on ((cball a r ∩ T) ∪ (T - ball a r)) ff
      unfolding ff_def
      using homeomorphism_cont1 [OF hom]
      by (intro continuous_on_cases) (auto simp: affine_closed ‹affine T› fid)
    then show continuous_on S ff
      by (rule continuous_on_subset) (use ST in auto)
    have continuous_on ((cball a r ∩ T) ∪ (T - ball a r)) gg
      unfolding gg_def
      using homeomorphism_cont2 [OF hom]
      by (intro continuous_on_cases) (auto simp: affine_closed ‹affine T› gid)
    then show continuous_on S gg
      by (rule continuous_on_subset) (use ST in auto)
  show ff ' S ⊆ S
  proof (clarsimp simp: ff_def)
    fix x
    assume x ∈ S and x: dist a x < r and x ∈ T
    then have f x ∈ cball a r ∩ T
      using homeomorphism_image1 [OF hom] by force
    then show f x ∈ S
      using ST(1) ‹x ∈ T› gid hom homeomorphism_def x by fastforce
  qed
  show gg ' S ⊆ S
  proof (clarsimp simp: gg_def)
    fix x
    assume x ∈ S and x: dist a x < r and x ∈ T
    then have g x ∈ cball a r ∩ T
      using homeomorphism_image2 [OF hom] by force
    then have g x ∈ ball a r
      using homeomorphism_apply2 [OF hom]
      by (metis Diff_Diff_Int Diff_iff ‹x ∈ T› cball_def fid le_less
mem_Collect_eq mem_ball mem_sphere x)
    then show g x ∈ S
      using ST(1) ‹g x ∈ cball a r ∩ T› by force
  qed
  show ∧x. x ∈ S ⇒ gg (ff x) = x
  unfolding ff_def gg_def
  using homeomorphism_apply1 [OF hom] homeomorphism_image1 [OF
hom]
  by simp (metis Int_iff homeomorphism_apply1 [OF hom] fid image_eqI
less_eq_real_def mem_cball mem_sphere)
  show ∧x. x ∈ S ⇒ ff (gg x) = x
  unfolding ff_def gg_def
  using homeomorphism_apply2 [OF hom] homeomorphism_image2 [OF
hom]
  by simp (metis Int_iff fid image_eqI less_eq_real_def mem_cball mem_sphere)

```

```

qed
show ff u = v
  using u by (auto simp: ff_def ‹f u = v›)
show {x. ¬ (ff x = x ∧ gg x = x)} ⊆ ball a r ∩ T
  by (auto simp: ff_def gg_def)
qed
qed

```

proposition *homeomorphism_moving_point*:

```

fixes a :: 'a::euclidean_space
assumes ope: openin (top_of_set (affine hull S)) S
  and S ⊆ T
  and TS: T ⊆ affine hull S
  and S: connected S a ∈ S b ∈ S
obtains f g where homeomorphism T T f g f a = b
  {x. ¬ (f x = x ∧ g x = x)} ⊆ S
  bounded {x. ¬ (f x = x ∧ g x = x)}
proof -
have 1: ∃ h k. homeomorphism T T h k ∧ h (f d) = d ∧
  {x. ¬ (h x = x ∧ k x = x)} ⊆ S ∧ bounded {x. ¬ (h x = x ∧ k x = x)}
  if d ∈ S f d ∈ S and homfg: homeomorphism T T f g
  and S: {x. ¬ (f x = x ∧ g x = x)} ⊆ S
  and bo: bounded {x. ¬ (f x = x ∧ g x = x)} for d f g
proof (intro exI conjI)
show homgf: homeomorphism T T g f
  by (metis homeomorphism_symD homfg)
then show g (f d) = d
  by (meson ‹S ⊆ T› homeomorphism_def subsetD ‹d ∈ S›)
show {x. ¬ (g x = x ∧ f x = x)} ⊆ S
  using S by blast
show bounded {x. ¬ (g x = x ∧ f x = x)}
  using bo by (simp add: conj_commute)
qed
have 2: ∃ f g. homeomorphism T T f g ∧ f x = f2 (f1 x) ∧
  {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x =
x)}
  if x ∈ S f1 x ∈ S f2 (f1 x) ∈ S
  and hom: homeomorphism T T f1 g1 homeomorphism T T f2 g2
  and sub: {x. ¬ (f1 x = x ∧ g1 x = x)} ⊆ S {x. ¬ (f2 x = x ∧ g2 x
= x)} ⊆ S
  and bo: bounded {x. ¬ (f1 x = x ∧ g1 x = x)} bounded {x. ¬ (f2 x
= x ∧ g2 x = x)}
  for x f1 f2 g1 g2
proof (intro exI conjI)
show homgf: homeomorphism T T (f2 ∘ f1) (g1 ∘ g2)
  by (metis homeomorphism_compose hom)
then show (f2 ∘ f1) x = f2 (f1 x)
  by force

```

```

show  $\{x. \neg ((f2 \circ f1) x = x \wedge (g1 \circ g2) x = x)\} \subseteq S$ 
using sub by force
have bounded  $(\{x. \neg (f1 x = x \wedge g1 x = x)\} \cup \{x. \neg (f2 x = x \wedge g2 x = x)\})$ 
using bo by simp
then show bounded  $\{x. \neg ((f2 \circ f1) x = x \wedge (g1 \circ g2) x = x)\}$ 
by (rule bounded_subset) auto
qed
have  $\exists U. \text{openin } (\text{top\_of\_set } S) U \wedge$ 
 $d \in U \wedge$ 
 $(\forall x \in U.$ 
 $\exists f g. \text{homeomorphism } T T f g \wedge f d = x \wedge$ 
 $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S \wedge$ 
 $\text{bounded } \{x. \neg (f x = x \wedge g x = x)\})$ 
if  $d \in S$  for  $d$ 
proof –
obtain  $r$  where  $r > 0$  and  $r: \text{ball } d r \cap \text{affine hull } S \subseteq S$ 
by (metis  $\langle d \in S \rangle$  ope openin_contains_ball)
have  $*$ :  $\exists f g. \text{homeomorphism } T T f g \wedge f d = e \wedge$ 
 $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S \wedge$ 
 $\text{bounded } \{x. \neg (f x = x \wedge g x = x)\}$  if  $e \in S$   $e \in \text{ball } d r$  for  $e$ 
apply (rule homeomorphism_moving_point_3 [of affine hull S d r T d e])
using  $r \langle S \subseteq T \rangle$  TS that
apply (auto simp:  $\langle d \in S \rangle \langle 0 < r \rangle$  hull_inc)
using bounded_subset by blast
show ?thesis
by (rule_tac  $x=S \cap \text{ball } d r$  in exI) (fastforce simp: openin_open_Int  $\langle 0 < r \rangle$  that intro: *)
qed
have  $\exists f g. \text{homeomorphism } T T f g \wedge f a = b \wedge$ 
 $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S \wedge \text{bounded } \{x. \neg (f x = x \wedge g x = x)\}$ 
by (rule connected_equivalence_relation [OF S]; blast intro: 1 2 3)
then show ?thesis
using that by auto
qed

```

```

lemma homeomorphism_moving_points_exists_gen:
assumes  $K: \text{finite } K \wedge i. i \in K \implies x i \in S \wedge y i \in S$ 
 $\text{pairwise } (\lambda i j. (x i \neq x j) \wedge (y i \neq y j)) K$ 
and  $2 \leq \text{aff\_dim } S$ 
and ope: openin (top_of_set (affine hull S))  $S$ 
and  $S \subseteq T \subseteq \text{affine hull } S$  connected S
shows  $\exists f g. \text{homeomorphism } T T f g \wedge (\forall i \in K. f(x i) = y i) \wedge$ 
 $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S \wedge \text{bounded } \{x. \neg (f x = x \wedge g x = x)\}$ 
using assms
proof (induction K)
case empty
then show ?case
by (force simp: homeomorphism_ident)

```

```

next
case (insert i K)
then have xney:  $\bigwedge j. [j \in K; j \neq i] \implies x i \neq x j \wedge y i \neq y j$ 
  and pw: pairwise  $(\lambda i j. x i \neq x j \wedge y i \neq y j)$  K
  and xi ∈ S yi ∈ S
  and xyS:  $\bigwedge i. i \in K \implies x i \in S \wedge y i \in S$ 
  by (simp_all add: pairwise_insert)
obtain fg where homfg: homeomorphism T T fg and feq:  $\bigwedge i. i \in K \implies f(x$ 
i) = y i
  and fg_sub:  $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S$ 
  and bo_fg: bounded  $\{x. \neg (f x = x \wedge g x = x)\}$ 
  using insert.IH [OF xyS pw] insert.prem by (blast intro: that)
then have  $\exists f g. \text{homeomorphism } T T f g \wedge (\forall i \in K. f(x i) = y i) \wedge$ 
 $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S \wedge \text{bounded } \{x. \neg (f x = x \wedge g x =$ 
x)\}
  using insert by blast
have aff_eq: affine hull (S - y ' K) = affine hull S
proof (rule affine_hull_Diff [OF ope])
  show finite (y ' K)
  by (simp add: insert.hyps(1))
  show y ' K  $\subseteq$  S
  using  $\langle y i \in S \rangle$  insert.hyps(2) xney xyS by fastforce
qed
have f_in_S: f x ∈ S if x ∈ S for x
  using homfg fg_sub homeomorphism_apply1  $\langle S \subseteq T \rangle$ 
proof -
  have  $(f (f x) \neq f x \vee g (f x) \neq f x) \vee f x \in S$ 
  by (metis  $\langle S \subseteq T \rangle$  homfg subsetD homeomorphism_apply1 that)
  then show ?thesis
  using fg_sub by force
qed
obtain h k where homhk: homeomorphism T T h k and heq: h (f (x i)) = y i
  and hk_sub:  $\{x. \neg (h x = x \wedge k x = x)\} \subseteq S - y ' K$ 
  and bo_hk: bounded  $\{x. \neg (h x = x \wedge k x = x)\}$ 
proof (rule homeomorphism_moving_point [of S - y'K T f(x i) y i])
  show openin (top_of_set (affine hull (S - y ' K))) (S - y ' K)
  by (simp add: aff_eq openin_diff finite_imp_closedin image_subset_iff
hull_inc insert xyS)
  show S - y ' K  $\subseteq$  T
  using  $\langle S \subseteq T \rangle$  by auto
  show T  $\subseteq$  affine hull (S - y ' K)
  using insert by (simp add: aff_eq)
  show connected (S - y ' K)
proof (rule connected_openin_diff_countable [OF  $\langle \text{connected } S \rangle$  ope])
  show  $\neg$  collinear S
  using collinear_aff_dim  $\langle 2 \leq \text{aff\_dim } S \rangle$  by force
  show countable (y ' K)
  using countable_finite insert.hyps(1) by blast
qed

```

```

    have  $\bigwedge k. \llbracket f(x\ i) = y\ k; k \in K \rrbracket \implies \text{False}$ 
      by (metis feq homfg  $\langle x\ i \in S \rangle$  homeomorphism_def  $\langle S \subseteq T \rangle \langle i \notin K \rangle$ 
subsetCE xney xyS)
    then show  $f(x\ i) \in S - y\ 'K$ 
      by (auto simp: f_in_S  $\langle x\ i \in S \rangle$ )
    show  $y\ i \in S - y\ 'K$ 
      using insert.hyps xney by (auto simp:  $\langle y\ i \in S \rangle$ )
  qed blast
  show ?case
  proof (intro exI conjI)
    show homeomorphism T T (h o f) (g o k)
      using homfg homhk homeomorphism_compose by blast
    show  $\forall i \in \text{insert } i\ K. (h \circ f)(x\ i) = y\ i$ 
      using feq hk_sub by (auto simp: heq)
    show  $\{x. \neg((h \circ f)\ x = x \wedge (g \circ k)\ x = x)\} \subseteq S$ 
      using fg_sub hk_sub by force
    have bounded ( $\{x. \neg(f\ x = x \wedge g\ x = x)\} \cup \{x. \neg(h\ x = x \wedge k\ x = x)\}$ )
      using bo_fg bo_hk bounded_Un by blast
    then show bounded  $\{x. \neg((h \circ f)\ x = x \wedge (g \circ k)\ x = x)\}$ 
      by (rule bounded_subset) auto
  qed
qed

proposition homeomorphism_moving_points_exists:
  fixes S :: 'a::euclidean_space set
  assumes 2:  $2 \leq \text{DIM}('a)$  open S connected S  $S \subseteq T$  finite K
    and KS:  $\bigwedge i. i \in K \implies x\ i \in S \wedge y\ i \in S$ 
    and pw: pairwise ( $\lambda i\ j. (x\ i \neq x\ j) \wedge (y\ i \neq y\ j)$ ) K
    and S:  $S \subseteq T$   $T \subseteq \text{affine hull } S$  connected S
  obtains f g where homeomorphism T T f g  $\bigwedge i. i \in K \implies f(x\ i) = y\ i$ 
     $\{x. \neg(f\ x = x \wedge g\ x = x)\} \subseteq S$  bounded  $\{x. (\neg(f\ x = x \wedge g\ x =$ 
x))\}
  proof (cases S = {})
    case True
      then show ?thesis
        using KS homeomorphism_ident that by fastforce
    next
      case False
        then have affS: affine hull S = UNIV
          by (simp add: affine_hull_open  $\langle \text{open } S \rangle$ )
        then have ope: openin (top_of_set (affine hull S)) S
          using  $\langle \text{open } S \rangle$  open_openin by auto
        have  $2 \leq \text{DIM}('a)$  by (rule 2)
        also have ... = aff_dim (UNIV :: 'a set)
          by simp
        also have ...  $\leq \text{aff\_dim } S$ 
          by (metis aff_dim_UNIV aff_dim_affine_hull aff_dim_le_DIM affS)
        finally have  $2 \leq \text{aff\_dim } S$ 
          by linarith
  
```

```

then show ?thesis
  using homeomorphism_moving_points_exists_gen [OF ‹finite K› KS pw _
  ope S] that by fastforce
qed

```

The theorem *homeomorphism_grouping_points_exists*

lemma *homeomorphism_grouping_point_1*:

```

fixes a::real and c::real
assumes a < b < c < d
obtains f g where homeomorphism (cbox a b) (cbox c d) f g f a = c f b = d
proof -
  define f where f ≡ λx. ((d - c) / (b - a)) * x + (c - a * ((d - c) / (b -
  a)))
  have ∃g. homeomorphism (cbox a b) (cbox c d) f g
  proof (rule homeomorphism_compact)
    show continuous_on (cbox a b) f
      unfolding f_def by (intro continuous_intros)
    have f ‘ {a..b} = {c..d}
      unfolding f_def image_affinity_atLeastAtMost
      using assms sum_sqs_eq by (auto simp: field_split_simps)
    then show f ‘ cbox a b = cbox c d
      by auto
    show inj_on f (cbox a b)
      unfolding f_def inj_on_def using assms by auto
  qed auto
  then obtain g where homeomorphism (cbox a b) (cbox c d) f g ..
  then show ?thesis
  proof
    show f a = c
      by (simp add: f_def)
    show f b = d
      using assms sum_sqs_eq [of a b] by (auto simp: f_def field_split_simps)
  qed
qed

```

lemma *homeomorphism_grouping_point_2*:

```

fixes a::real and w::real
assumes hom_ab: homeomorphism (cbox a b) (cbox u v) f1 g1
  and hom_bc: homeomorphism (cbox b c) (cbox v w) f2 g2
  and b ∈ cbox a c v ∈ cbox u w
  and eq: f1 a = u f1 b = v f2 b = v f2 c = w
obtains f g where homeomorphism (cbox a c) (cbox u w) f g f a = u f c = w
  ∧x. x ∈ cbox a b ⇒ f x = f1 x ∧x. x ∈ cbox b c ⇒ f x = f2 x
proof -
  have le: a ≤ b b ≤ c u ≤ v v ≤ w
    using assms by simp_all
  then have ac: cbox a c = cbox a b ∪ cbox b c and uw: cbox u w = cbox u v ∪
  cbox v w

```

```

    by auto
  define f where f  $\equiv \lambda x. \text{if } x \leq b \text{ then } f1 \ x \text{ else } f2 \ x$ 
  have  $\exists g. \text{homeomorphism } (cbox \ a \ c) \ (cbox \ u \ w) \ f \ g$ 
  proof (rule homeomorphism_compact)
    have cf1: continuous_on (cbox a b) f1
      using hom_ab homeomorphism_cont1 by blast
    have cf2: continuous_on (cbox b c) f2
      using hom_bc homeomorphism_cont1 by blast
    show continuous_on (cbox a c) f
      unfolding f_def using le eq
      by (force intro: continuous_on_cases_le [OF continuous_on_subset [OF cf1]
        continuous_on_subset [OF cf2]])
    have f ' cbox a b = f1 ' cbox a b f ' cbox b c = f2 ' cbox b c
      unfolding f_def using eq by force+
    then show f ' cbox a c = cbox u w
      unfolding ac uw image_Un by (metis hom_ab hom_bc homeomorphism_def)
    have neq12: f1 x  $\neq$  f2 y if x: a  $\leq$  x  $\leq$  b and y: b < y  $\leq$  c for x y
    proof -
      have f1 x  $\in$  cbox u v
        by (metis hom_ab homeomorphism_def image_eqI mem_box_real(2) x)
      moreover have f2 y  $\in$  cbox v w
        by (metis (full_types) hom_bc homeomorphism_def image_subset_iff
          mem_box_real(2) not_le not_less_iff_gr_or_eq order_refl y)
      moreover have f2 y  $\neq$  f2 b
        by (metis cancel_comm_monoid_add_class.diff_cancel diff_gt_0_iff_gt
          hom_bc homeomorphism_def le(2) less_imp_le less_numeral_extra(3) mem_box_real(2)
          order_refl y)
      ultimately show ?thesis
        using le eq by simp
    qed
  have inj_on f1 (cbox a b)
    by (metis (full_types) hom_ab homeomorphism_def inj_onI)
  moreover have inj_on f2 (cbox b c)
    by (metis (full_types) hom_bc homeomorphism_def inj_onI)
  ultimately show inj_on f (cbox a c)
    apply (simp (no_asm) add: inj_on_def)
    apply (simp add: f_def inj_on_eq_iff)
    using neq12 by force
  qed auto
  then obtain g where homeomorphism (cbox a c) (cbox u w) f g ..
  then show ?thesis
    using eq f_def le that by force
  qed

```

lemma homeomorphism_grouping_point_3:

fixes a::real

assumes cbox_sub: cbox c d \subseteq box a b cbox u v \subseteq box a b

and box_ne: box c d \neq {} box u v \neq {}

obtains f g where homeomorphism (cbox a b) (cbox a b) f g f a = a f b = b

$$\bigwedge x. x \in \text{cbox } c \ d \implies f \ x \in \text{cbox } u \ v$$

proof –

have *less*: $a < c \ a < u \ d < b \ v < b \ c < d \ u < v \ \text{cbox } c \ d \neq \{\}$
using *assms*
by (*simp_all add: cbox_sub subset_eq*)
obtain $f1 \ g1$ **where** $1: \text{homeomorphism } (\text{cbox } a \ c) \ (\text{cbox } a \ u) \ f1 \ g1$
and $f1_eq: f1 \ a = a \ f1 \ c = u$
using *homeomorphism_grouping_point_1 [OF ‹a < c› ‹a < u›]*.
obtain $f2 \ g2$ **where** $2: \text{homeomorphism } (\text{cbox } c \ d) \ (\text{cbox } u \ v) \ f2 \ g2$
and $f2_eq: f2 \ c = u \ f2 \ d = v$
using *homeomorphism_grouping_point_1 [OF ‹c < d› ‹u < v›]*.
obtain $f3 \ g3$ **where** $3: \text{homeomorphism } (\text{cbox } d \ b) \ (\text{cbox } v \ b) \ f3 \ g3$
and $f3_eq: f3 \ d = v \ f3 \ b = b$
using *homeomorphism_grouping_point_1 [OF ‹d < b› ‹v < b›]*.
obtain $f4 \ g4$ **where** $4: \text{homeomorphism } (\text{cbox } a \ d) \ (\text{cbox } a \ v) \ f4 \ g4$ **and** $f4 \ a = a \ f4 \ d = v$
and $f4_eq: \bigwedge x. x \in \text{cbox } a \ c \implies f4 \ x = f1 \ x \ \bigwedge x. x \in \text{cbox } c \ d \implies f4 \ x = f2 \ x$
using *homeomorphism_grouping_point_2 [OF 1 2] less* **by** (*auto simp: f1_eq f2_eq*)
obtain $f \ g$ **where** $fg: \text{homeomorphism } (\text{cbox } a \ b) \ (\text{cbox } a \ b) \ f \ g \ f \ a = a \ f \ b = b$
and $f_eq: \bigwedge x. x \in \text{cbox } a \ d \implies f \ x = f4 \ x \ \bigwedge x. x \in \text{cbox } d \ b \implies f \ x = f3 \ x$
using *homeomorphism_grouping_point_2 [OF 4 3] less* **by** (*auto simp: f4_eq f3_eq f2_eq f1_eq*)
show *?thesis*
proof (*rule that [OF fg]*)
show $f \ x \in \text{cbox } u \ v$ **if** $x \in \text{cbox } c \ d$ **for** x
using *that f4_eq f_eq homeomorphism_image1 [OF 2]*
by (*metis atLeastAtMost_iff box_real(2) image_eqI less(1) less_eq_real_def order_trans*)
qed
qed

lemma *homeomorphism_grouping_point_4*:

fixes $T :: \text{real set}$

assumes *open U open S connected S U ≠ {} finite K K ⊆ S U ⊆ S S ⊆ T*

obtains $f \ g$ **where** *homeomorphism T T f g*

$$\bigwedge x. x \in K \implies f \ x \in U \ \{x. (\neg (f \ x = x \wedge g \ x = x))\} \subseteq S$$

$$\text{bounded } \{x. (\neg (f \ x = x \wedge g \ x = x))\}$$

proof –

obtain $c \ d$ **where** $\text{box } c \ d \neq \{\}$ $\text{cbox } c \ d \subseteq U$

proof –

obtain u **where** $u \in U$

using $\langle U \neq \{\} \rangle$ **by** *blast*

then obtain e **where** $e > 0 \ \text{cball } u \ e \subseteq U$

using $\langle \text{open } U \rangle$ *open_contains_cball* **by** *blast*

then show *?thesis*

```

    by (rule_tac c=u and d=u+e in that) (auto simp: dist_norm subset_iff)
qed
have compact K
  by (simp add: ⟨finite K⟩ finite_imp_compact)
obtain a b where box a b ≠ {} K ⊆ cbox a b cbox a b ⊆ S
proof (cases K = {})
  case True then show ?thesis
    using ⟨box c d ≠ {}⟩ ⟨cbox c d ⊆ U⟩ ⟨U ⊆ S⟩ that by blast
  next
  case False
  then obtain a b where a ∈ K b ∈ K
    and a:  $\bigwedge x. x \in K \implies a \leq x$  and b:  $\bigwedge x. x \in K \implies x \leq b$ 
    using compact_attains_inf compact_attains_sup by (metis ⟨compact K⟩)+
  obtain e where e > 0 cball b e ⊆ S
    using ⟨open S⟩ open_contains_cball
    by (metis ⟨b ∈ K⟩ ⟨K ⊆ S⟩ subsetD)
  show ?thesis
  proof
    show box a (b + e) ≠ {}
      using ⟨0 < e⟩ ⟨b ∈ K⟩ a by force
    show K ⊆ cbox a (b + e)
      using ⟨0 < e⟩ a b by fastforce
    have a ∈ S
      using ⟨a ∈ K⟩ assms(6) by blast
    have b + e ∈ S
      using ⟨0 < e⟩ ⟨cball b e ⊆ S⟩ by (force simp: dist_norm)
    show cbox a (b + e) ⊆ S
      using ⟨a ∈ S⟩ ⟨b + e ∈ S⟩ ⟨connected S⟩ connected_contains_Icc by auto
  qed
qed
obtain w z where cbox w z ⊆ S and sub_wz: cbox a b ∪ cbox c d ⊆ box w z
proof -
  have a ∈ S b ∈ S
    using ⟨box a b ≠ {}⟩ ⟨cbox a b ⊆ S⟩ by auto
  moreover have c ∈ S d ∈ S
    using ⟨box c d ≠ {}⟩ ⟨cbox c d ⊆ U⟩ ⟨U ⊆ S⟩ by force+
  ultimately have min a c ∈ S max b d ∈ S
    by linarith+
  then obtain e1 e2 where e1 > 0 cball (min a c) e1 ⊆ S e2 > 0 cball (max
b d) e2 ⊆ S
    using ⟨open S⟩ open_contains_cball by metis
  then have *: min a c - e1 ∈ S max b d + e2 ∈ S
    by (auto simp: dist_norm)
  show ?thesis
  proof
    show cbox (min a c - e1) (max b d + e2) ⊆ S
      using * ⟨connected S⟩ connected_contains_Icc by auto
    show cbox a b ∪ cbox c d ⊆ box (min a c - e1) (max b d + e2)
      using ⟨0 < e1⟩ ⟨0 < e2⟩ by auto
  qed

```

```

qed
qed
then
obtain f g where hom: homeomorphism (cbox w z) (cbox w z) f g
  and f w = w f z = z
  and fin:  $\bigwedge x. x \in \text{cbox } a \ b \implies f x \in \text{cbox } c \ d$ 
  using homeomorphism_grouping_point_3 [of a b w z c d]
  using  $\langle \text{cbox } a \ b \neq \{\} \rangle \langle \text{cbox } c \ d \neq \{\} \rangle$  by blast
have contfg: continuous_on (cbox w z) f continuous_on (cbox w z) g
  using hom homeomorphism_def by blast+
define f' where f'  $\equiv \lambda x. \text{if } x \in \text{cbox } w \ z \ \text{then } f \ x \ \text{else } x$ 
define g' where g'  $\equiv \lambda x. \text{if } x \in \text{cbox } w \ z \ \text{then } g \ x \ \text{else } x$ 
show ?thesis
proof
  have T:  $\text{cbox } w \ z \cup (T - \text{cbox } w \ z) = T$ 
    using  $\langle \text{cbox } w \ z \subseteq S \rangle \langle S \subseteq T \rangle$  by auto
  show homeomorphism T T f' g'
  proof
    have clo: closedin (top_of_set (cbox w z  $\cup$  (T - cbox w z))) (T - cbox w z)
      by (metis Diff_Diff_Int Diff_subset T closedin_def open_box openin_open_Int
topspace_euclidean_subtopology)
    have  $\bigwedge x. \llbracket w \leq x \wedge x \leq z; w < x \longrightarrow \neg x < z \rrbracket \implies f x = x$ 
      using  $\langle f w = w \rangle \langle f z = z \rangle$  by auto
    moreover have  $\bigwedge x. \llbracket w \leq x \wedge x \leq z; w < x \longrightarrow \neg x < z \rrbracket \implies g x = x$ 
      using  $\langle f w = w \rangle \langle f z = z \rangle$  hom homeomorphism_apply1 by fastforce
    ultimately
    have continuous_on (cbox w z  $\cup$  (T - cbox w z)) f' continuous_on (cbox w z
 $\cup$  (T - cbox w z)) g'
      unfolding f'_def g'_def
      by (intro continuous_on_cases_local contfg continuous_on_id clo; auto
simp: closed_subset)+
    then show continuous_on T f' continuous_on T g'
      by (simp_all only: T)
    show f' ' $T \subseteq T$ '
      unfolding f'_def
      by clarsimp (metis  $\langle \text{cbox } w \ z \subseteq S \rangle \langle S \subseteq T \rangle$  subsetD hom homeomorphism_def
imageI mem_box_real(2))
    show g' ' $T \subseteq T$ '
      unfolding g'_def
      by clarsimp (metis  $\langle \text{cbox } w \ z \subseteq S \rangle \langle S \subseteq T \rangle$  subsetD hom homeomorphism_def
imageI mem_box_real(2))
    show  $\bigwedge x. x \in T \implies g' (f' x) = x$ 
      unfolding f'_def g'_def
      using homeomorphism_apply1 [OF hom] homeomorphism_image1 [OF
hom] by fastforce
    show  $\bigwedge y. y \in T \implies f' (g' y) = y$ 
      unfolding f'_def g'_def
      using homeomorphism_apply2 [OF hom] homeomorphism_image2 [OF
hom] by fastforce

```

```

qed
show  $\bigwedge x. x \in K \implies f' x \in U$ 
  using fin sub_wz  $\langle K \subseteq \text{cbox } a \ b \rangle \langle \text{cbox } c \ d \subseteq U \rangle$  by (force simp: f'_def)
show  $\{x. \neg (f' x = x \wedge g' x = x)\} \subseteq S$ 
  using  $\langle \text{cbox } w \ z \subseteq S \rangle$  by (auto simp: f'_def g'_def)
show bounded  $\{x. \neg (f' x = x \wedge g' x = x)\}$ 
proof (rule bounded_subset [of cbox w z])
  show bounded (cbox w z)
    using bounded_cbox by blast
  show  $\{x. \neg (f' x = x \wedge g' x = x)\} \subseteq \text{cbox } w \ z$ 
    by (auto simp: f'_def g'_def)
qed
qed
qed

```

proposition *homeomorphism_grouping_points_exists*:

```

fixes S :: 'a::euclidean_space set
assumes open U open S connected S U  $\neq \{\}$  finite K  $K \subseteq S$   $U \subseteq S$   $S \subseteq T$ 
obtains f g where homeomorphism T T f g  $\{x. (\neg (f x = x \wedge g x = x))\} \subseteq S$ 
  bounded  $\{x. (\neg (f x = x \wedge g x = x))\} \bigwedge x. x \in K \implies f x \in U$ 
proof (cases 2 ≤ DIM('a))
  case True
  have TS:  $T \subseteq \text{affine hull } S$ 
    using affine_hull_open assms by blast
  have infinite U
    using  $\langle \text{open } U \rangle \langle U \neq \{\} \rangle$  finite_imp_not_open by blast
  then obtain P where  $P \subseteq U$  finite P  $\text{card } K = \text{card } P$ 
    using infinite_arbitrarily_large by metis
  then obtain  $\gamma$  where  $\gamma$ : bij_betw  $\gamma$  K P
    using  $\langle \text{finite } K \rangle$  finite_same_card_bij by blast
  obtain f g where homeomorphism T T f g  $\bigwedge i. i \in K \implies f (\text{id } i) = \gamma i$   $\{x. \neg (f x = x \wedge g x = x)\} \subseteq S$  bounded  $\{x. \neg (f x = x \wedge g x = x)\}$ 
  proof (rule homeomorphism_moving_points_exists [OF True  $\langle \text{open } S \rangle \langle \text{connected } S \rangle \langle S \subseteq T \rangle \langle \text{finite } K \rangle$ ])
    show  $\bigwedge i. i \in K \implies \text{id } i \in S \wedge \gamma i \in S$ 
      using  $\langle P \subseteq U \rangle \langle \text{bij\_betw } \gamma \ K \ P \rangle \langle K \subseteq S \rangle \langle U \subseteq S \rangle$  bij_betwE by blast
    show pairwise  $(\lambda i \ j. \text{id } i \neq \text{id } j \wedge \gamma i \neq \gamma j) \ K$ 
      using  $\gamma$  by (auto simp: pairwise_def bij_betw_def inj_on_def)
  qed (use affine_hull_open assms that in auto)
  then show ?thesis
    using  $\gamma$   $\langle P \subseteq U \rangle$  bij_betwE by (fastforce simp: intro!: that)
next
case False
with DIM_positive have  $\text{DIM}('a) = 1$ 
  by (simp add: dual_order.antisym)
then obtain h::'a  $\Rightarrow$  real and j
where linear h linear j
  and noh:  $\bigwedge x. \text{norm}(h \ x) = \text{norm } x$  and noj:  $\bigwedge y. \text{norm}(j \ y) = \text{norm } y$ 
  and hj:  $\bigwedge x. j(h \ x) = x \ \bigwedge y. h(j \ y) = y$ 

```

```

and ranh: surj h
using isomorphisms_UNIV_UNIV
by (metis (mono_tags, opaque_lifting) DIM_real UNIV_eq_I range_eqI)
obtain f g where hom: homeomorphism (h ' T) (h ' T) f g
      and f:  $\bigwedge x. x \in h ' K \implies f x \in h ' U$ 
      and sub:  $\{x. \neg (f x = x \wedge g x = x)\} \subseteq h ' S$ 
      and bou: bounded  $\{x. \neg (f x = x \wedge g x = x)\}$ 
apply (rule homeomorphism_grouping_point_4 [of h ' U h ' S h ' K h ' T])
by (simp_all add: assms image_mono <linear h> open_surjective_linear_image
connected_linear_image ranh)
have jf:  $j (f (h x)) = x \iff f (h x) = h x$  for x
      by (metis hj)
have jg:  $j (g (h x)) = x \iff g (h x) = h x$  for x
      by (metis hj)
have cont_hj: continuous_on X h continuous_on Y j for X Y
      by (simp_all add: <linear h> <linear j> linear_linear linear_continuous_on)
show ?thesis
proof
  show homeomorphism T T (j o f o h) (j o g o h)
  proof
    show continuous_on T (j o f o h) continuous_on T (j o g o h)
    using hom homeomorphism_def
    by (blast intro: continuous_on_compose cont_hj)+
    show (j o f o h) ' T  $\subseteq$  T (j o g o h) ' T  $\subseteq$  T
    by auto (metis (mono_tags, opaque_lifting) hj(1) hom homeomorphism_def
imageE imageI)+
    show  $\bigwedge x. x \in T \implies (j o g o h) ((j o f o h) x) = x$ 
    using hj hom homeomorphism_apply1 by fastforce
    show  $\bigwedge y. y \in T \implies (j o f o h) ((j o g o h) y) = y$ 
    using hj hom homeomorphism_apply2 by fastforce
  qed
  show  $\{x. \neg ((j o f o h) x = x \wedge (j o g o h) x = x)\} \subseteq S$ 
  proof (clarsimp simp: jf jg hj)
    show  $f (h x) = h x \implies g (h x) \neq h x \implies x \in S$  for x
    using sub [THEN subsetD, of h x] hj by simp (metis imageE)
  qed
  have bounded (j '  $\{x. (\neg (f x = x \wedge g x = x))\}$ )
  by (rule bounded_linear_image [OF bou]) (use <linear j> linear_conv_bounded_linear
in auto)
  moreover
  have *:  $\{x. \neg ((j o f o h) x = x \wedge (j o g o h) x = x)\} = j ' \{x. (\neg (f x = x \wedge
g x = x))\}$ 
  using hj by (auto simp: jf jg image_iff, metis+)
  ultimately show bounded  $\{x. \neg ((j o f o h) x = x \wedge (j o g o h) x = x)\}$ 
  by metis
  show  $\bigwedge x. x \in K \implies (j o f o h) x \in U$ 
  using f hj by fastforce
  qed
qed

```

proposition *homeomorphism_grouping_points_exists_gen*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes $\text{ope}U: \text{openin}(\text{top_of_set } S) U$
and $\text{ope}S: \text{openin}(\text{top_of_set}(\text{affine hull } S)) S$
and $U \neq \{\}$ *finite* $K \subseteq S$ **and** $S: S \subseteq T \ T \subseteq \text{affine hull } S$ *connected* S
obtains $f g$ **where** *homeomorphism* $T \ T \ f \ g \ \{x. (\neg (f \ x = x \wedge g \ x = x))\} \subseteq S$
 $\text{bounded } \{x. (\neg (f \ x = x \wedge g \ x = x))\} \wedge x. x \in K \implies f \ x \in U$
proof (*cases* $2 \leq \text{aff_dim } S$)
case *True*
have $\text{ope}U': \text{openin}(\text{top_of_set}(\text{affine hull } S)) U$
using $\text{ope}S \ \text{ope}U \ \text{openin_trans}$ **by** *blast*
obtain u **where** $u \in U \ u \in S$
using $\langle U \neq \{\} \rangle \ \text{ope}U \ \text{openin_imp_subset}$ **by** *fastforce+*
have *infinite* U
proof (*rule* *infinite_openin* [*OF* $\text{ope}U \ \langle u \in U \rangle$])
show u *islimpt* S
using *True* $\langle u \in S \rangle \ \text{assms}(8) \ \text{connected_imp_perfect_aff_dim}$ **by** *fastforce*
qed
then obtain P **where** $P \subseteq U$ *finite* $P \ \text{card } K = \text{card } P$
using *infinite_arbitrarily_large* **by** *metis*
then obtain γ **where** $\gamma: \text{bij_betw } \gamma \ K \ P$
using $\langle \text{finite } K \rangle \ \text{finite_same_card_bij}$ **by** *blast*
have $\exists f \ g. \ \text{homeomorphism } T \ T \ f \ g \wedge (\forall i \in K. f(\text{id } i) = \gamma \ i) \wedge$
 $\{x. \neg (f \ x = x \wedge g \ x = x)\} \subseteq S \wedge \text{bounded } \{x. \neg (f \ x = x \wedge g \ x = x)\}$
proof (*rule* *homeomorphism_moving_points_exists_gen* [*OF* $\langle \text{finite } K \rangle \ _ _$
 $\text{True } \text{ope}S \ S$])
show $\bigwedge i. i \in K \implies \text{id } i \in S \wedge \gamma \ i \in S$
by (*metis* *id_apply* $\text{ope}U \ \text{openin_contains_cball}$ *subsetCE* $\langle P \subseteq U \rangle \ \langle \text{bij_betw}$
 $\gamma \ K \ P \rangle \ \langle K \subseteq S \rangle \ \text{bij_betwE}$)
show *pairwise* $(\lambda i \ j. \ \text{id } i \neq \text{id } j \wedge \gamma \ i \neq \gamma \ j) \ K$
using γ **by** (*auto* *simp: pairwise_def* *bij_betw_def* *inj_on_def*)
qed
then show *?thesis*
using $\gamma \ \langle P \subseteq U \rangle \ \text{bij_betwE}$ **by** (*fastforce* *simp: intro!*: *that*)
next
case *False*
with *aff_dim_geq* [*of* S] **consider** $\text{aff_dim } S = -1 \mid \text{aff_dim } S = 0 \mid \text{aff_dim}$
 $S = 1$ **by** *linarith*
then show *?thesis*
proof *cases*
assume $\text{aff_dim } S = -1$
then have $S = \{\}$
using *aff_dim_empty* **by** *blast*
then have *False*
using $\langle U \neq \{\} \rangle \ \langle K \subseteq S \rangle \ \text{openin_imp_subset}$ [*OF* $\text{ope}U$] **by** *blast*
then show *?thesis* ..
next

```

assume aff_dim S = 0
then obtain a where S = {a}
  using aff_dim_eq_0 by blast
then have K ⊆ U
  using ⟨U ≠ {}⟩ ⟨K ⊆ S⟩ openin_imp_subset [OF opeU] by blast
show ?thesis
  using ⟨K ⊆ U⟩ by (intro that [of id id]) (auto intro: homeomorphismI)
next
assume aff_dim S = 1
then have affine_hull_S_homeomorphic (UNIV :: real set)
  by (auto simp: homeomorphic_affine_sets)
then obtain h::'a⇒real and j where homhj: homeomorphism (affine_hull S)
UNIV h j
  using homeomorphic_def by blast
  then have h: ∧x. x ∈ affine_hull S ⇒ j(h(x)) = x and j: ∧y. j y ∈ affine_hull S ∧ h(j y) = y
    by (auto simp: homeomorphic_def)
  have connh: connected (h ' S)
    by (meson Topological_Spaces.connected_continuous_image ⟨connected S⟩
homeomorphism_cont1 homeomorphism_of_subsets homhj_hull_subset_top_greatest)
  have hUS: h ' U ⊆ h ' S
    by (meson homeomorphism_imp_open_map homeomorphism_of_subsets
homhj_hull_subset_opeS_opeU_open_UNIV_openin_open_eq)
  have opn: openin (top_of_set (affine_hull S)) U ⇒ open (h ' U) for U
    using homeomorphism_imp_open_map [OF homhj] by simp
  have open (h ' U) open (h ' S)
    by (auto intro: opeS_opeU_openin_trans opn)
  then obtain f g where hom: homeomorphism (h ' T) (h ' T) f g
    and f: ∧x. x ∈ h ' K ⇒ f x ∈ h ' U
    and sub: {x. ¬ (f x = x ∧ g x = x)} ⊆ h ' S
    and bou: bounded {x. ¬ (f x = x ∧ g x = x)}
  apply (rule homeomorphism_grouping_points_exists [of h ' U h ' S h ' K h
' T])
  using assms by (auto simp: connh hUS)
  have jf: ∧x. x ∈ affine_hull S ⇒ j (f (h x)) = x ⟷ f (h x) = h x
    by (metis h j)
  have jg: ∧x. x ∈ affine_hull S ⇒ j (g (h x)) = x ⟷ g (h x) = h x
    by (metis h j)
  have cont_hj: continuous_on T h continuous_on Y j for Y
proof (rule continuous_on_subset [OF _ ⟨T ⊆ affine_hull S⟩])
  show continuous_on (affine_hull S) h
    using homeomorphic_def homhj by blast
qed (meson continuous_on_subset homeomorphic_def homhj_top_greatest)
define f' where f' ≡ λx. if x ∈ affine_hull S then (j ∘ f ∘ h) x else x
define g' where g' ≡ λx. if x ∈ affine_hull S then (j ∘ g ∘ h) x else x
show ?thesis
proof
  show homeomorphism T T f' g'
proof

```

```

    have continuous_on T (j ∘ f ∘ h)
      using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
    then show continuous_on T f'
      apply (rule continuous_on_eq)
      using ⟨T ⊆ affine hull S⟩ f'_def by auto
    have continuous_on T (j ∘ g ∘ h)
      using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
    then show continuous_on T g'
      apply (rule continuous_on_eq)
      using ⟨T ⊆ affine hull S⟩ g'_def by auto
    show f' ' T ⊆ T
    proof (clarsimp simp: f'_def)
      fix x assume x ∈ T
      then have f (h x) ∈ h ' T
        by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
      then show j (f (h x)) ∈ T
        using ⟨T ⊆ affine hull S⟩ h by auto
    qed
    show g' ' T ⊆ T
    proof (clarsimp simp: g'_def)
      fix x assume x ∈ T
      then have g (h x) ∈ h ' T
        by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
      then show j (g (h x)) ∈ T
        using ⟨T ⊆ affine hull S⟩ h by auto
    qed
    show ∧x. x ∈ T ⇒ g' (f' x) = x
      using h j hom homeomorphism_apply1 by (fastforce simp: f'_def g'_def)
    show ∧y. y ∈ T ⇒ f' (g' y) = y
      using h j hom homeomorphism_apply2 by (fastforce simp: f'_def g'_def)
    qed
  next
    have §: ∧x y. [x ∈ affine hull S; h x = h y; y ∈ S] ⇒ x ∈ S
      by (metis h hull_inc)
    show {x. ¬ (f' x = x ∧ g' x = x)} ⊆ S
      using sub by (simp add: f'_def g'_def jf jg) (force elim: §)
  next
    have compact (j ' closure {x. ¬ (f x = x ∧ g x = x)})
      using bou by (auto simp: compact_continuous_image cont_hj)
    then have bounded (j ' {x. ¬ (f x = x ∧ g x = x)})
      by (rule bounded_closure_image [OF compact_imp_bounded])
    moreover
    have *: {x ∈ affine hull S. j (f (h x)) ≠ x ∨ j (g (h x)) ≠ x} = j ' {x. ¬ (f
x = x ∧ g x = x)}
      using h j by (auto simp: image_iff; metis)

```



```

ultimately have bounded {x ∈ affine hull S. j (f (h x)) ≠ x ∨ j (g (h x)) ≠
x}
  by metis
then show bounded {x. ¬ (f' x = x ∧ g' x = x)}
  by (simp add: f'_def g'_def Collect_mono bounded_subset)
next
show f' x ∈ U if x ∈ K for x
proof -
  have U ⊆ S
    using opeU openin_imp_subset by blast
  then have j (f (h x)) ∈ U
    using f h hull_subset that by fastforce
  then show f' x ∈ U
    using ⟨K ⊆ S⟩ S f'_def that by auto
qed
qed
qed
qed

```

6.4.28 Nullhomotopic mappings

A mapping out of a sphere is nullhomotopic iff it extends to the ball. This even works out in the degenerate cases when the radius is ≤ 0 , and we also don't need to explicitly assume continuity since it's already implicit in both sides of the equivalence.

lemma *nullhomotopic_from_lemma*:

```

assumes contg: continuous_on (cball a r - {a}) g
  and fa:  $\bigwedge e. 0 < e$ 
            $\implies \exists d. 0 < d \wedge (\forall x. x \neq a \wedge \text{norm}(x - a) < d \implies \text{norm}(g x - f$ 
a) < e)
```

and $r: \bigwedge x. x \in \text{cball } a \ r \wedge x \neq a \implies f x = g x$

shows *continuous_on* (cball a r) f

proof (clarsimp simp: continuous_on_eq_continuous_within Ball_def)

fix x

assume $x: \text{dist } a \ x \leq r$

show *continuous* (at x within cball a r) f

proof (cases $x=a$)

case True

then show ?thesis

by (metis continuous_within_eps_delta fa dist_norm dist_self r)

next

case False

show ?thesis

proof (rule continuous_transform_within [where $f=g$ and $d = \text{norm}(x-a)$])

have $\exists d > 0. \forall x' \in \text{cball } a \ r.$

$\text{dist } x' \ x < d \implies \text{dist } (g \ x') \ (g \ x) < e$ **if** $e > 0$ **for** e

proof -

obtain d **where** $d > 0$

```

    and d:  $\bigwedge x'. \llbracket \text{dist } x' a \leq r; x' \neq a; \text{dist } x' x < d \rrbracket \implies$ 
               $\text{dist } (g x') (g x) < e$ 
    using contg False x <e>0>
    unfolding continuous_on_iff by (fastforce simp: dist_commute intro:
that)
    show ?thesis
      using <d > 0> <x ≠ a>
      by (rule_tac x=min d (norm(x - a)) in exI)
        (auto simp: dist_commute dist_norm [symmetric] intro!: d)
    qed
    then show continuous (at x within cball a r) g
      using contg False by (auto simp: continuous_within_eps_delta)
    show 0 < norm (x - a)
      using False by force
    show x ∈ cball a r
      by (simp add: x)
    show  $\bigwedge x'. \llbracket x' \in \text{cball } a r; \text{dist } x' x < \text{norm } (x - a) \rrbracket$ 
       $\implies g x' = f x'$ 
      by (metis dist_commute dist_norm less_le r)
    qed
  qed
qed

proposition nullhomotopic_from_sphere_extension:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  shows  $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{sphere } a r) S f (\lambda x. c)) \iff$ 
     $(\exists g. \text{continuous\_on } (\text{cball } a r) g \wedge g \text{ ' } (\text{cball } a r) \subseteq S \wedge$ 
       $(\forall x \in \text{sphere } a r. g x = f x))$ 
    (is ?lhs = ?rhs)
proof (cases r 0::real rule: linorder_cases)
  case less
    then show ?thesis
      by (simp add: homotopic_on_emptyI)
  next
    case equal
      show ?thesis
        proof
          assume L: ?lhs
          with equal have [simp]: f a ∈ S
            using homotopic_with_imp_subset1 by fastforce
          obtain h:: real × 'M  $\Rightarrow$  'a
            where h: continuous_on  $(\{0..1\} \times \{a\})$  h h '  $(\{0..1\} \times \{a\}) \subseteq S$  h (0, a)
            = f a
            using L equal by (auto simp: homotopic_with)
          then have continuous_on (cball a r)  $(\lambda x. h (0, a)) (\lambda x. h (0, a)) \text{ ' cball } a r$ 
             $\subseteq S$ 
            by (auto simp: equal)
          then show ?rhs
            using h(3) local.equal by force
        qed

```

```

next
  assume ?rhs
  then show ?lhs
    using equal_continuous_on_const by (force simp: homotopic_with)
qed
next
  case greater
  let ?P = continuous_on {x. norm(x - a) = r} f ∧ f ' {x. norm(x - a) = r}
  ⊆ S
  have ?P if ?lhs using that
  proof
    fix c
    assume c: homotopic_with_canon (λx. True) (sphere a r) S f (λx. c)
    then have contf: continuous_on (sphere a r) f
      by (metis homotopic_with_imp_continuous)
    moreover have fim: f ' sphere a r ⊆ S
      by (meson continuous_map_subtopology_eu c homotopic_with_imp_continuous_maps)
    show ?P
      using contf fim by (auto simp: sphere_def dist_norm norm_minus_commute)
  qed
  moreover have ?P if ?rhs using that
  proof
    fix g
    assume g: continuous_on (cball a r) g ∧ g ' cball a r ⊆ S ∧ (∀ xa ∈ sphere a r.
  g xa = f xa)
    then have f ' {x. norm(x - a) = r} ⊆ S
      using sphere_cball [of a r] unfolding image_subset_iff sphere_def
      by (metis dist_commute dist_norm mem_Collect_eq subset_eq)
    with g show ?P
      by (auto simp: dist_norm norm_minus_commute elim!: continuous_on_eq
  [OF continuous_on_subset])
  qed
  moreover have ?thesis if ?P
  proof
    assume ?lhs
    then obtain c where homotopic_with_canon (λx. True) (sphere a r) S (λx.
  c) f
      using homotopic_with_sym by blast
    then obtain h where conth: continuous_on ({0..1::real} × sphere a r) h
      and him: h ' ({0..1} × sphere a r) ⊆ S
      and h: ∧x. h(0, x) = c ∧x. h(1, x) = f x
      by (auto simp: homotopic_with_def)
    obtain b1::'M where b1 ∈ Basis
      using SOME_Basis by auto
    have c ∈ h ' ({0..1} × sphere a r)
    proof
      show c = h (0, a + r *_R b1)
        by (simp add: h)
      show (0, a + r *_R b1) ∈ {0..1::real} × sphere a r

```

```

    using greater ‹b1 ∈ Basis› by (auto simp: dist_norm)
  qed
  then have c ∈ S
    using him by blast
  have ucont: uniformly_continuous_on ({0..1::real} × (sphere a r)) h
    by (force intro: compact_Times cont_h compact_uniformly_continuous)
  let ?g = λx. h (norm (x - a)/r,
    a + (if x = a then r *R b1 else (r / norm(x - a)) *R (x - a)))
  let ?g' = λx. h (norm (x - a)/r, a + (r / norm(x - a)) *R (x - a))
  show ?rhs
  proof (intro exI conjI)
    have continuous_on (cball a r - {a}) ?g'
      using greater
    by (force simp: dist_norm norm_minus_commute intro: continuous_on_compose2
      [OF cont_h] continuous_intros)
    then show continuous_on (cball a r) ?g
    proof (rule nullhomotopic_from_lemma)
      show ∃ d > 0. ∀ x. x ≠ a ∧ norm (x - a) < d ⟶ norm (?g' x - ?g a) < e
    if 0 < e for e
      proof -
        obtain d where 0 < d
          and d: ∧ x x'. [x ∈ {0..1} × sphere a r; x' ∈ {0..1} × sphere a r; norm
            (x' - x) < d]
            ⟹ norm (h x' - h x) < e
          using uniformly_continuous_onE [OF ucont ‹0 < e›] by (auto simp:
            dist_norm)
          have *: norm (h (norm (x - a) / r,
            a + (r / norm (x - a)) *R (x - a)) - h (0, a + r *R b1)) <
            e (is norm (?ha - ?hb) < e)
            if x ≠ a norm (x - a) < r norm (x - a) < d * r for x
          proof -
            have norm (?ha - ?hb) = norm (?ha - h (0, a + (r / norm (x - a))
              *R (x - a)))
              by (simp add: h)
            also have ... < e
              using greater ‹0 < d› ‹b1 ∈ Basis› that
            by (intro d) (simp_all add: dist_norm, simp add: field_simps)
            finally show ?thesis .
          qed
          show ?thesis
            using greater ‹0 < d›
            by (rule_tac x = min r (d * r) in exI) (auto simp: *)
        qed
      show ∧ x. x ∈ cball a r ∧ x ≠ a ⟹ ?g x = ?g' x
        by auto
    qed
  next
  show ?g ' cball a r ⊆ S
    using greater him ‹c ∈ S›

```

```

    by (force simp: h dist_norm norm_minus_commute)
  next
    show  $\forall x \in \text{sphere } a \ r. \ ?g \ x = f \ x$ 
      using greater by (auto simp: h dist_norm norm_minus_commute)
  qed
next
assume ?rhs
then obtain  $g$  where  $\text{contg: continuous\_on } (\text{cball } a \ r) \ g$ 
  and  $\text{gim: } g \ ' \ \text{cball } a \ r \subseteq S$ 
  and  $\text{gf: } \forall x \in \text{sphere } a \ r. \ g \ x = f \ x$ 
  by auto
let ?h =  $\lambda y. \ g \ (a + (\text{fst } y) *_R (\text{snd } y - a))$ 
have continuous_on  $(\{0..1\} \times \text{sphere } a \ r) \ ?h$ 
proof (rule continuous_on_compose2 [OF contg])
  show continuous_on  $(\{0..1\} \times \text{sphere } a \ r) \ (\lambda x. \ a + \text{fst } x *_R (\text{snd } x - a))$ 
    by (intro continuous_intros)
  qed (auto simp: dist_norm norm_minus_commute mult_left_le_one_le)
moreover
have ?h  $\ ' \ (\{0..1\} \times \text{sphere } a \ r) \subseteq S$ 
  by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gim
[THEN subsetD])
moreover
have  $\forall x \in \text{sphere } a \ r. \ ?h \ (0, x) = g \ a \ \forall x \in \text{sphere } a \ r. \ ?h \ (1, x) = f \ x$ 
  by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gf)
ultimately have homotopic_with_canon  $(\lambda x. \ \text{True}) \ (\text{sphere } a \ r) \ S \ (\lambda x. \ g \ a) \ f$ 
  by (auto simp: homotopic_with)
then show ?lhs
  using homotopic_with_symD by blast
qed
ultimately
show ?thesis by meson
qed
end

```

6.5 Euclidean space and n-spheres, as subtopologies of n-dimensional space

```

theory Abstract_Euclidean_Space
imports Homotopy Locally
begin

```

6.5.1 Euclidean spaces as abstract topologies

```

definition Euclidean_space ::  $\text{nat} \Rightarrow (\text{nat} \Rightarrow \text{real}) \ \text{topology}$ 
  where  $\text{Euclidean\_space } n \equiv \text{subtopology } (\text{powertop\_real } \text{UNIV}) \ \{x. \ \forall i \geq n. \ x \ i = 0\}$ 

```

1202

lemma *topspace_Euclidean_space*:
 $\text{topspace}(\text{Euclidean_space } n) = \{x. \forall i \geq n. x\ i = 0\}$
 by (*simp add: Euclidean_space_def*)

lemma *nontrivial_Euclidean_space*: $\text{Euclidean_space } n \neq \text{trivial_topology}$
 using *topspace_Euclidean_space* [of n] **by** *force*

lemma *subset_Euclidean_space* [*simp*]:
 $\text{topspace}(\text{Euclidean_space } m) \subseteq \text{topspace}(\text{Euclidean_space } n) \iff m \leq n$
 apply (*simp add: topspace_Euclidean_space subset_iff, safe*)
 apply (*drule_tac x=($\lambda i. \text{if } i < m \text{ then } 1 \text{ else } 0$) in spec*)
 apply *auto*
 using *not_less* **by** *fastforce*

lemma *topspace_Euclidean_space_alt*:
 $\text{topspace}(\text{Euclidean_space } n) = (\bigcap i \in \{n..\}. \{x. x \in \text{topspace}(\text{powertop_real UNIV}) \wedge x\ i \in \{0\}\})$
 by (*auto simp: topspace_Euclidean_space*)

lemma *closedin_Euclidean_space*:
 $\text{closedin}(\text{powertop_real UNIV}) (\text{topspace}(\text{Euclidean_space } n))$
proof –
 have $\text{closedin}(\text{powertop_real UNIV}) \{x. x\ i = 0\}$ **if** $n \leq i$ **for** i
 proof –
 have $\text{closedin}(\text{powertop_real UNIV}) \{x \in \text{topspace}(\text{powertop_real UNIV}). x\ i \in \{0\}\}$
 proof (*rule closedin_continuous_map_preimage*)
 show $\text{continuous_map}(\text{powertop_real UNIV}) \text{euclideanreal } (\lambda x. x\ i)$
 by (*metis UNIV_I continuous_map_product_coordinates*)
 show $\text{closedin euclideanreal } \{0\}$
 by *simp*
 qed
 then show *?thesis*
 by *auto*
 qed
 then show *?thesis*
 unfolding *topspace_Euclidean_space_alt*
 by *force*
qed

lemma *closedin_Euclidean_imp_closed*: $\text{closedin}(\text{Euclidean_space } m) S \implies \text{closed } S$
 by (*metis Euclidean_space_def closed_closedin closedin_Euclidean_space closedin_closed_subtopology euclidean_product_topology topspace_Euclidean_space*)

lemma *closedin_Euclidean_space_iff*:
 $\text{closedin}(\text{Euclidean_space } m) S \iff \text{closed } S \wedge S \subseteq \text{topspace}(\text{Euclidean_space } m)$
 (is *?lhs* \iff *?rhs***)**

proof

show $?lhs \implies ?rhs$
using `closedin_closed_subtopology topspace_Euclidean_space`
by (`fastforce simp: topspace_Euclidean_space_alt closedin_Euclidean_imp_closed`)
show $?rhs \implies ?lhs$
apply (`simp add: closedin_subtopology Euclidean_space_def`)
by (`metis (no_types) closed_closedin euclidean_product_topology inf.orderE`)
qed

lemma `continuous_map_componentwise_Euclidean_space`:

`continuous_map X (Euclidean_space n) ($\lambda x i. \text{if } i < n \text{ then } f x i \text{ else } 0$)` \longleftrightarrow
 $(\forall i < n. \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x i))$

proof –

have $*$: `continuous_map X euclideanreal ($\lambda x. \text{if } k < n \text{ then } f x k \text{ else } 0$)`
if $\bigwedge i. i < n \implies \text{continuous_map } X \text{ euclideanreal } (\lambda x. f x i)$ **for** k
by (`intro continuous_intros that`)
show $?thesis$
unfolding `Euclidean_space_def continuous_map_in_subtopology`
by (`fastforce simp: continuous_map_componentwise_UNIV * elim: continuous_map_eq`)
qed

lemma `continuous_map_Euclidean_space_add` [`continuous_intros`]:

$\llbracket \text{continuous_map } X \text{ (Euclidean_space } n) f; \text{continuous_map } X \text{ (Euclidean_space } n) g \rrbracket$
 $\implies \text{continuous_map } X \text{ (Euclidean_space } n) (\lambda x i. f x i + g x i)$
unfolding `Euclidean_space_def continuous_map_in_subtopology`
by (`fastforce simp add: continuous_map_componentwise_UNIV continuous_map_add`)

lemma `continuous_map_Euclidean_space_diff` [`continuous_intros`]:

$\llbracket \text{continuous_map } X \text{ (Euclidean_space } n) f; \text{continuous_map } X \text{ (Euclidean_space } n) g \rrbracket$
 $\implies \text{continuous_map } X \text{ (Euclidean_space } n) (\lambda x i. f x i - g x i)$
unfolding `Euclidean_space_def continuous_map_in_subtopology`
by (`fastforce simp add: continuous_map_componentwise_UNIV continuous_map_diff`)

lemma `continuous_map_Euclidean_space_iff`:

`continuous_map (Euclidean_space m) euclidean g`
 $= \text{continuous_on (topspace (Euclidean_space m)) } g$

proof

assume `continuous_map (Euclidean_space m) euclidean g`
then have `continuous_map (top_of_set {f. $\forall n \geq m. f n = 0$ }) euclidean g`
by (`simp add: Euclidean_space_def euclidean_product_topology`)
then show `continuous_on (topspace (Euclidean_space m)) g`
by (`metis continuous_map_subtopology_eu subtopology_topspace topspace_Euclidean_space`)
next

assume `continuous_on (topspace (Euclidean_space m)) g`
then have `continuous_map (top_of_set {f. $\forall n \geq m. f n = 0$ }) euclidean g`
by (`metis (lifting) continuous_map_into_fulltopology continuous_map_subtopology_eu`)

```

order_refl topspace Euclidean_space)
  then show continuous_map (Euclidean_space m) euclidean g
    by (simp add: Euclidean_space_def euclidean_product_topology)
qed

```

lemma *cm_Euclidean_space_iff_continuous_on*:

```

continuous_map (subtopology (Euclidean_space m) S) (Euclidean_space n) f
   $\longleftrightarrow$  continuous_on (topspace (subtopology (Euclidean_space m) S)) f  $\wedge$ 
     $f \in (\text{topspace (subtopology (Euclidean\_space } m) S)) \rightarrow \text{topspace (Euclidean\_space } n)$ 
(is ?P  $\longleftrightarrow$  ?Q  $\wedge$  ?R)

```

proof –

have ?Q if ?P

proof –

have $\bigwedge n. \text{Euclidean_space } n = \text{top_of_set } \{f. \forall m \geq n. f m = 0\}$

by (simp add: Euclidean_space_def euclidean_product_topology)

with that show ?thesis

by (simp add: subtopology_subtopology)

qed

moreover

have ?R if ?P

using that by (simp add: image_subset_iff continuous_map_def)

moreover

have ?P if ?Q ?R

proof –

```

have continuous_map (top_of_set (topspace (subtopology (subtopology (powertop_real
UNIV) {f.  $\forall n \geq m. f n = 0$ }) S))) (top_of_set (topspace (subtopology (powertop_real
UNIV) {f.  $\forall na \geq n. f na = 0$ }))) f

```

using Euclidean_space_def that by auto

then show ?thesis

```

  by (simp add: Euclidean_space_def euclidean_product_topology subtopology_subtopology)

```

qed

ultimately show ?thesis

by auto

qed

lemma *homeomorphic_Euclidean_space_product_topology*:

```

Euclidean_space n homeomorphic_space product_topology ( $\lambda i. \text{euclideanreal } \{..<n\}$ )

```

proof –

have *cm*: continuous_map (product_topology ($\lambda i. \text{euclideanreal } \{..<n\}$))

euclideanreal ($\lambda x. \text{if } k < n \text{ then } x k \text{ else } 0$) **for** *k*

by (auto intro: continuous_map_if_continuous_map_product_projection)

show ?thesis

unfolding homeomorphic_space_def homeomorphic_maps_def

apply (rule_tac $x = \lambda f. \text{restrict } f \{..<n\}$ **in** *exI*)

apply (rule_tac $x = \lambda f i. \text{if } i < n \text{ then } f i \text{ else } 0$ **in** *exI*)

apply (simp add: Euclidean_space_def continuous_map_in_subtopology)


```

apply (intro conjI continuous_map_from_subtopology)
apply (force simp: continuous_map_componentwise cm intro: continuous_map_product_projection)+
done
qed

```

```

lemma contractible_Euclidean_space [simp]: contractible_space (Euclidean_space n)
using homeomorphic_Euclidean_space_product_topology contractible_space_euclideanreal
contractible_space_product_topology homeomorphic_space_contractibility by
blast

```

```

lemma path_connected_Euclidean_space: path_connected_space (Euclidean_space n)
by (simp add: contractible_imp_path_connected_space)

```

```

lemma connected_Euclidean_space: connected_space (Euclidean_space n)
by (simp add: contractible_imp_connected_space)

```

```

lemma locally_path_connected_Euclidean_space:
  locally_path_connected_space (Euclidean_space n)
apply (simp add: homeomorphic_locally_path_connected_space [OF homeomorphic_Euclidean_space_product_topology [of n]]
  locally_path_connected_space_product_topology)
using locally_path_connected_space_euclideanreal by auto

```

```

lemma compact_Euclidean_space [simp]:
  compact_space (Euclidean_space n)  $\longleftrightarrow$  n = 0
using homeomorphic_compact_space [OF homeomorphic_Euclidean_space_product_topology]
by (auto simp: product_topology_trivial_iff compact_space_product_topology)

```

6.5.2 n-dimensional spheres

definition nsphere where

$nsphere\ n \equiv\ subtopology\ (Euclidean_space\ (Suc\ n))\ \{x.\ (\sum\ i \leq n.\ x\ i^2) = 1\}$

lemma nsphere:

$nsphere\ n = subtopology\ (powertop_real\ UNIV)\ \{x.\ (\sum\ i \leq n.\ x\ i^2) = 1 \wedge (\forall\ i > n.\ x\ i = 0)\}$

by (simp add: nsphere_def Euclidean_space_def subtopology_subtopology Suc_le_eq Collect_conj_eq Int_commute)

lemma continuous_map_nsphere_projection: continuous_map (nsphere n) euclideanreal $(\lambda x.\ x\ k)$

unfolding nsphere

by (blast intro: continuous_map_from_subtopology [OF continuous_map_product_projection])

lemma in_topospace_nsphere: $(\lambda n.\ if\ n = 0\ then\ 1\ else\ 0) \in\ topspace\ (nsphere\ n)$

by (simp add: nsphere_def topspace_Euclidean_space power2_eq_square if_distrib
[where f = $\lambda x. x * _$] cong: if_cong)

lemma nonempty_nsphere [simp]: (nsphere n) \neq trivial_topology
by (metis discrete_topology_unique empty_iff in_topspace_nsphere)

lemma subtopology_nsphere_equator:

subtopology (nsphere (Suc n)) {x. x(Suc n) = 0} = nsphere n

proof –

have ($\{x. (\sum_{i \leq n. (x i)^2} + (x (Suc n))^2 = 1 \wedge (\forall i > Suc n. x i = 0))\} \cap \{x. x$
(Suc n) = 0})

= $\{x. (\sum_{i \leq n. (x i)^2} = 1 \wedge (\forall i > n. x i = (0::real))\}$

using Suc_lessI [of n] **by** (fastforce simp: set_eq_iff)

then show ?thesis

by (simp add: nsphere subtopology_subtopology)

qed

lemma topspace_nsphere_minus1:

assumes x: $x \in$ topspace (nsphere n) **and** x n = 0

shows $x \in$ topspace (nsphere (n - Suc 0))

proof (cases n = 0)

case True

then show ?thesis

using x **by** auto

next

case False

have subt_eq: nsphere (n - Suc 0) = subtopology (nsphere n) {x. x n = 0}

by (metis False Suc_pred le_zero_eq not_le subtopology_nsphere_equator)

with x **show** ?thesis

by (simp add: assms)

qed

lemma continuous_map_nsphere_reflection:

continuous_map (nsphere n) (nsphere n) ($\lambda x i. \text{if } i = k \text{ then } -x i \text{ else } x i$)

proof –

have cm: continuous_map (powertop_real UNIV) euclideanreal ($\lambda x. \text{if } j = k$
then $-x j$ else $x j$) **for** j

proof (cases j=k)

case True

then show ?thesis

by simp (metis UNIV_I continuous_map_product_projection)

next

case False

then show ?thesis

by (auto intro: continuous_map_product_projection)

qed

have eq: ($\text{if } i = k \text{ then } x k * x k \text{ else } x i * x i$) = $x i * x i$ **for** i **and** $x :: \text{nat} \Rightarrow$
real

by simp

```

show ?thesis
apply (simp add: nsphere continuous_map_in_subtopology continuous_map_componentwise_UNIV
        continuous_map_from_subtopology cm)
apply (intro conjI allI impI continuous_intros continuous_map_from_subtopology
        continuous_map_product_projection)
apply (auto simp: power2_eq_square if_distrib [where f =  $\lambda x. x * \_$ ] eq
        cong: if_cong)
done
qed

```

proposition *contractible_space_upper_hemisphere:*

```

assumes  $k \leq n$ 
shows contractible_space(subtopology (nsphere n) { $x. x k \geq 0$ })
proof -
define  $p:: nat \Rightarrow real$  where  $p \equiv \lambda i. \text{if } i = k \text{ then } 1 \text{ else } 0$ 
have  $p \in \text{topspace}(nsphere\ n)$ 
using assms
by (simp add: nsphere p_def power2_eq_square if_distrib [where f =  $\lambda x. x * \_$ ]
    cong: if_cong)
let ?g =  $\lambda x\ i. x\ i / \text{sqrt}(\sum_{j \leq n. x\ j^2})$ 
let ?h =  $\lambda(t,q)\ i. (1 - t) * q\ i + t * p\ i$ 
let ?Y = subtopology (Euclidean_space (Suc n)) { $x. 0 \leq x k \wedge (\exists i \leq n. x\ i \neq 0)$ }
have continuous_map (prod_topology (top_of_set {0..1}) (subtopology (nsphere
n) { $x. 0 \leq x k$ }))
    (subtopology (nsphere n) { $x. 0 \leq x k$ }) (?g  $\circ$  ?h)
proof (rule continuous_map_compose)
have *:  $\llbracket 0 \leq b\ k; (\sum_{i \leq n. (b\ i)^2} = 1; \forall i > n. b\ i = 0; 0 \leq a; a \leq 1 \rrbracket$ 
     $\implies \exists i. (i = k \longrightarrow (1 - a) * b\ k + a \neq 0) \wedge$ 
     $(i \neq k \longrightarrow i \leq n \wedge a \neq 1 \wedge b\ i \neq 0)$  for  $a::real$  and  $b$ 
apply (cases  $a \neq 1 \wedge b\ k = 0$ ; simp)
apply (metis (no_types, lifting) atMost_iff sum_neutral zero_power2)
by (metis add_commute add_le_same_cancel2 diff_ge_0_iff_ge diff_zero
    less_eq_real_def mult_eq_0_iff mult_nonneg_nonneg not_le numeral_One zero_neq_numeral)
show continuous_map (prod_topology (top_of_set {0..1}) (subtopology (nsphere
n) { $x. 0 \leq x k$ })) ?Y ?h
using assms
apply (auto simp: * nsphere continuous_map_componentwise_UNIV
    prod_topology_subtopology subtopology_subtopology case_prod_unfold
    continuous_map_in_subtopology Euclidean_space_def p_def if_distrib
    [where f =  $\lambda x. \_ * x$ ] cong: if_cong)
apply (intro continuous_map_prod_snd continuous_intros continuous_map_from_subtopology)
apply auto
done
next
have 1:  $\bigwedge x\ i. \llbracket i \leq n; x\ i \neq 0 \rrbracket \implies (\sum_{i \leq n. (x\ i / \text{sqrt}(\sum_{j \leq n. (x\ j)^2}))^2} = 1$ 
by (force simp: sum_nonneg sum_nonneg_eq_0_iff field_split_simps simp
    flip: sum_divide_distrib)

```

```

have cm: continuous_map ?Y (nsphere n) ( $\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
unfolding Euclidean_space_def nsphere subtopology_subtopology continuous_map_in_subtopology
proof (intro continuous_intros conjI)
show continuous_map
  (subtopology (powertop_real UNIV) ({x.  $\forall i \geq \text{Suc } n. x i = 0$ }  $\cap$  {x.  $0 \leq x k \wedge (\exists i \leq n. x i \neq 0)$ }))
  (powertop_real UNIV) ( $\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
unfolding continuous_map_componentwise
by (intro continuous_intros conjI ballI) (auto simp: sum_nonneg_eq_0_iff)
qed (auto simp: 1)
show continuous_map ?Y (subtopology (nsphere n) {x.  $0 \leq x k$ }) ( $\lambda x i. x i / \text{sqrt} (\sum j \leq n. (x j)^2)$ )
by (force simp: cm sum_nonneg continuous_map_in_subtopology if_distrib
[where f =  $\lambda x. \_ * x$ ] cong: if_cong)
qed
moreover have (?g  $\circ$  ?h) (0, x) = x
if x  $\in$  topspace (subtopology (nsphere n) {x.  $0 \leq x k$ }) for x
using that
by (simp add: assms nsphere)
moreover
have (?g  $\circ$  ?h) (1, x) = p
if x  $\in$  topspace (subtopology (nsphere n) {x.  $0 \leq x k$ }) for x
by (force simp: assms p_def power2_eq_square if_distrib [where f =  $\lambda x. x *$ 
_] cong: if_cong)
ultimately
show ?thesis
apply (simp add: contractible_space_def homotopic_with)
apply (rule_tac x=p in exI)
apply (rule_tac x=?g  $\circ$  ?h in exI, force)
done
qed

```

corollary *contractible_space_lower_hemisphere*:

```

assumes k  $\leq$  n
shows contractible_space (subtopology (nsphere n) {x.  $x k \leq 0$ })
proof –
have contractible_space (subtopology (nsphere n) {x.  $0 \leq x k$ }) = ?thesis
proof (rule homeomorphic_space_contractibility)
show subtopology (nsphere n) {x.  $0 \leq x k$ } homeomorphic_space subtopology
(nsphere n) {x.  $x k \leq 0$ }
unfolding homeomorphic_space_def homeomorphic_maps_def
apply (rule_tac x= $\lambda x i. \text{if } i = k \text{ then } -(x i) \text{ else } x i$  in exI) +
apply (auto simp: continuous_map_in_subtopology continuous_map_from_subtopology
continuous_map_nsphere_reflection)
done
qed
then show ?thesis

```

using contractible_space_upper_hemisphere [OF assms] by metis
qed

proposition nullhomotopic_nonsurjective_sphere_map:

assumes f : continuous_map (nsphere p) (nsphere p) f

and fim : $f \text{ ' } (topspace(nsphere\ p)) \neq topspace(nsphere\ p)$

obtains a where homotopic_with $(\lambda x. True)$ (nsphere p) (nsphere p) f $(\lambda x. a)$

proof –

obtain a where a : $a \in topspace(nsphere\ p)$ $a \notin f \text{ ' } (topspace(nsphere\ p))$

using fim continuous_map_image_subset_topspace f by blast

then have $a1$: $(\sum i \leq p. (a\ i)^2) = 1$ and $a0$: $\bigwedge i. i > p \implies a\ i = 0$

by (simp_all add: nsphere)

have $f1$: $(\sum j \leq p. (f\ x\ j)^2) = 1$ if $x \in topspace\ (nsphere\ p)$ for x

proof –

have $f\ x \in topspace\ (nsphere\ p)$

using continuous_map_image_subset_topspace f that by blast

then show *thesis*

by (simp add: nsphere)

qed

show *thesis*

proof

let $?g = \lambda x\ i. x\ i / \text{sqrt}(\sum j \leq p. x\ j^2)$

let $?h = \lambda(t,x)\ i. (1 - t) * f\ x\ i - t * a\ i$

let $?Y = \text{subtopology}\ (\text{Euclidean_space}\ (\text{Suc}\ p))\ (-\ \{\lambda i. 0\})$

let $?T01 = \text{top_of_set}\ \{0..1::\text{real}\}$

have 1 : continuous_map (prod_topology ?T01 (nsphere p)) (nsphere p) $(?g \circ ?h)$

proof (rule continuous_map_compose)

have continuous_map (prod_topology ?T01 (nsphere p)) euclideanreal $((\lambda x. f\ x\ k) \circ \text{snd})$ for k

unfolding nsphere

apply (simp add: continuous_map_of_snd flip: null_topspace_iff_trivial)

apply (rule continuous_map_compose [of _ nsphere p f , unfolded o_def])

using f apply (simp add: nsphere)

by (simp add: continuous_map_nsphere_projection)

then have continuous_map (prod_topology ?T01 (nsphere p)) euclideanreal $(\lambda r. ?h\ r\ k)$

for k

unfolding case_prod_unfold o_def

by (intro continuous_map_into_fulltopology [OF continuous_mapfst] continuous_intros) auto

moreover have $?h \text{ ' } (\{0..1\} \times topspace\ (nsphere\ p)) \subseteq \{x. \forall i \geq \text{Suc}\ p. x\ i = 0\}$

using continuous_map_image_subset_topspace [OF f]

by (auto simp: nsphere_image_subset_iff a0)

moreover have $(\lambda i. 0) \notin ?h \text{ ' } (\{0..1\} \times topspace\ (nsphere\ p))$

proof clarify

fix $t\ b$

```

    assume eq:  $(\lambda i. 0) = (\lambda i. (1 - t) * f b i - t * a i)$  and  $t \in \{0..1\}$  and
  b:  $b \in \text{topspace } (\text{nsphere } p)$ 
  have  $(1 - t)^2 = (\sum i \leq p. ((1 - t) * f b i)^2)$ 
    using f1 [OF b] by (simp add: power_mult_distrib flip: sum_distrib_left)
  also have  $\dots = (\sum i \leq p. (t * a i)^2)$ 
    using eq by (simp add: fun_eq_iff)
  also have  $\dots = t^2$ 
    using a1 by (simp add: power_mult_distrib flip: sum_distrib_left)
  finally have  $1 - t = t$ 
    by (simp add: power2_eq_iff)
  then have  $*: t = 1/2$ 
    by simp
  have fba:  $f b \neq a$ 
    using a(2) b by blast
  then show False
    using eq unfolding * by (simp add: fun_eq_iff)
qed
ultimately show continuous_map (prod_topology ?T01 (nsphere p)) ?Y ?h
  by (simp add: Euclidean_space_def continuous_map_in_subtopology continuous_map_componentwise_UNIV)
next
  have *:  $\llbracket \forall i \geq \text{Suc } p. x i = 0; x \neq (\lambda i. 0) \rrbracket \implies (\sum j \leq p. (x j)^2) \neq 0$  for  $x :: \text{nat} \Rightarrow \text{real}$ 
    by (force simp: fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff)
  show continuous_map ?Y (nsphere p) ?g
    apply (simp add: Euclidean_space_def continuous_map_in_subtopology continuous_map_componentwise_UNIV
      nsphere continuous_map_componentwise_subtopology_subtopology)
    apply (intro conjI allI continuous_intros continuous_map_from_subtopology [OF continuous_map_product_projection])
    apply (simp_all add: *)
    apply (force simp: sum_nonneg fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff power_divide simp flip: sum_divide_distrib)
  done
qed
have 2:  $(?g \circ ?h) (0, x) = f x$  if  $x \in \text{topspace } (\text{nsphere } p)$  for  $x$ 
  using that f1 by simp
have 3:  $(?g \circ ?h) (1, x) = (\lambda i. - a i)$  for  $x$ 
  using a by (force simp: field_split_simps nsphere)
then show homotopic_with  $(\lambda x. \text{True})$  (nsphere p) (nsphere p) f  $(\lambda x. (\lambda i. - a i))$ 
  by (force simp: homotopic_with intro: 1 2 3)
qed
qed

```

lemma Hausdorff_Euclidean_space:

Hausdorff_space (Euclidean_space n)

unfolding Euclidean_space_def

by (rule Hausdorff_space_subtopology) (metis Hausdorff_space_euclidean Haus-

dorff_space_product_topology)

end

6.6 Various Forms of Topological Spaces

theory *Abstract_Topological_Spaces*

imports *Lindelof_Spaces Locally Abstract_Euclidean_Space Sum_Topology FSigma*
begin

6.6.1 Connected topological spaces

lemma *connected_space_eq_frontier_eq_empty:*

$connected_space\ X \longleftrightarrow (\forall S. S \subseteq topspace\ X \wedge X\ frontier_of\ S = \{\} \longrightarrow S = \{\} \vee S = topspace\ X)$

by (*meson clopenin_eq_frontier_of_connected_space_clopen_in*)

lemma *connected_space_frontier_eq_empty:*

$connected_space\ X \wedge S \subseteq topspace\ X$

$\implies (X\ frontier_of\ S = \{\} \longleftrightarrow S = \{\} \vee S = topspace\ X)$

by (*meson connected_space_eq_frontier_eq_empty frontier_of_empty_frontier_of_topspace*)

lemma *connectedin_eq_subset_separated_union:*

$connectedin\ X\ C \longleftrightarrow$

$C \subseteq topspace\ X \wedge (\forall S\ T. separatedin\ X\ S\ T \wedge C \subseteq S \cup T \longrightarrow C \subseteq S \vee C \subseteq T)$ (**is** *?lhs=?rhs*)

proof

assume *?lhs* **then show** *?rhs*

using *connectedin_subset_topspace connectedin_subset_separated_union* **by** *blast*
next

assume *?rhs*

then show *?lhs*

by (*metis closure_of_subset connectedin_separation_dual_order_eq_iff_inf_orderE separatedin_def sup.boundedE*)

qed

lemma *connectedin_clopen_cases:*

$\llbracket connectedin\ X\ C; closedin\ X\ T; openin\ X\ T \rrbracket \implies C \subseteq T \vee disjnt\ C\ T$

by (*metis Diff_eq_empty_iff Int_empty_right clopenin_eq_frontier_of_connectedin_Int_frontier_of_disjnt_def*)

lemma *connected_space_retraction_map_image:*

$\llbracket retraction_map\ X\ X'\ r; connected_space\ X \rrbracket \implies connected_space\ X'$

using *connected_space_quotient_map_image retraction_imp_quotient_map* **by** *blast*

lemma *connectedin_imp_perfect_gen:*

assumes *X: t1_space X* **and** *S: connectedin X S* **and** *nontriv: $\nexists a. S = \{a\}$*

shows $S \subseteq X$ *derived_set_of* S
unfolding *derived_set_of_def*
proof (*intro subsetI CollectI conjI strip*)
show $X S$: $x \in \text{topspace } X$ **if** $x \in S$ **for** x
using *that S connectedin by fastforce*
show $\exists y. y \neq x \wedge y \in S \wedge y \in T$
if $x \in S$ **and** $x \in T \wedge \text{openin } X T$ **for** $x T$
proof –
have *opeXx: openin X (topspace X - {x})*
by (*meson X openin_topspace t1_space_openin_delete_alt*)
moreover
have $S \subseteq T \cup (\text{topspace } X - \{x\})$
using *XS that(2) by auto*
moreover have $(\text{topspace } X - \{x\}) \cap S \neq \{\}$
by (*metis Diff_triv S connectedin double_diff empty_subsetI inf_commute*
insert_subsetI nontriv that(1))
ultimately show *?thesis*
using *that connectedinD [OF S, of T topspace X - {x}]*
by *blast*
qed
qed

lemma *connectedin_imp_perfect*:
 $\llbracket \text{Hausdorff_space } X; \text{connectedin } X S; \nexists a. S = \{a\} \rrbracket \implies S \subseteq X$ *derived_set_of* S
by (*simp add: Hausdorff_imp_t1_space connectedin_imp_perfect_gen*)

6.6.2 The notion of "separated between" (complement of "connected between")

definition *separated_between*
where *separated_between* $X S T \equiv$
 $\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge U \cup V = \text{topspace } X \wedge \text{disjnt } U V \wedge$
 $S \subseteq U \wedge T \subseteq V$

lemma *separated_between_alt*:
separated_between $X S T \longleftrightarrow$
 $(\exists U V. \text{closedin } X U \wedge \text{closedin } X V \wedge U \cup V = \text{topspace } X \wedge \text{disjnt } U V$
 $\wedge S \subseteq U \wedge T \subseteq V)$
unfolding *separated_between_def*
by (*metis separatedin_open_sets separation_closedin_Un_gen subtopology_topspace*
separatedin_closed_sets separation_openin_Un_gen)

lemma *separated_between*:
separated_between $X S T \longleftrightarrow$
 $(\exists U. \text{closedin } X U \wedge \text{openin } X U \wedge S \subseteq U \wedge T \subseteq \text{topspace } X - U)$
unfolding *separated_between_def closedin_def disjnt_def*
by (*smt (verit, del_insts) Diff_cancel Diff_disjoint Diff_partition Un_Diff Un_Diff_Int*)

openin_subset)

lemma *separated_between_mono*:

$\llbracket \text{separated_between } X \ S \ T; S' \subseteq S; T' \subseteq T \rrbracket \implies \text{separated_between } X \ S' \ T'$
by (*meson order.trans separated_between*)

lemma *separated_between_refl*:

$\text{separated_between } X \ S \ S \longleftrightarrow S = \{\}$
unfolding *separated_between_def*
by (*metis Un_empty_right disjnt_def disjnt_empty2 disjnt_subset2 disjnt_sym le_iff_inf openin_empty openin_topspace*)

lemma *separated_between_sym*:

$\text{separated_between } X \ S \ T \longleftrightarrow \text{separated_between } X \ T \ S$
by (*metis disjnt_sym separated_between_alt sup_commute*)

lemma *separated_between_imp_subset*:

$\text{separated_between } X \ S \ T \implies S \subseteq \text{topspace } X \wedge T \subseteq \text{topspace } X$
by (*metis le_supI1 le_supI2 separated_between_def*)

lemma *separated_between_empty*:

$(\text{separated_between } X \ \{\} \ S \longleftrightarrow S \subseteq \text{topspace } X) \wedge (\text{separated_between } X \ S \ \{\} \longleftrightarrow S \subseteq \text{topspace } X)$
by (*metis Diff_empty bot.extremum closedin_empty openin_empty separated_between separated_between_imp_subset separated_between_sym*)

lemma *separated_between_Un*:

$\text{separated_between } X \ S \ (T \cup U) \longleftrightarrow \text{separated_between } X \ S \ T \wedge \text{separated_between } X \ S \ U$
by (*auto simp: separated_between*)

lemma *separated_between_Un'*:

$\text{separated_between } X \ (S \cup T) \ U \longleftrightarrow \text{separated_between } X \ S \ U \wedge \text{separated_between } X \ T \ U$
by (*simp add: separated_between_Un separated_between_sym*)

lemma *separated_between_imp_disjoint*:

$\text{separated_between } X \ S \ T \implies \text{disjnt } S \ T$
by (*meson disjnt_iff separated_between_def subsetD*)

lemma *separated_between_imp_separatedin*:

$\text{separated_between } X \ S \ T \implies \text{separatedin } X \ S \ T$
by (*meson separated_between_def separatedin_mono separatedin_open_sets*)

lemma *separated_between_full*:

assumes $S \cup T = \text{topspace } X$
shows $\text{separated_between } X \ S \ T \longleftrightarrow \text{disjnt } S \ T \wedge \text{closedin } X \ S \wedge \text{openin } X \ S \wedge \text{closedin } X \ T \wedge \text{openin } X \ T$

proof –

have $\text{separated_between } X S T \longrightarrow \text{separatedin } X S T$
by (*simp add: separated_between_imp_separatedin*)
then show *?thesis*
unfolding *separated_between_def*
by (*metis assms separation_closedin_Un_gen separation_openin_Un_gen subset_refl subtopology_topspace*)
qed

lemma *separated_between_eq_separatedin*:

$S \cup T = \text{topspace } X \implies (\text{separated_between } X S T \longleftrightarrow \text{separatedin } X S T)$
by (*simp add: separated_between_full separatedin_full*)

lemma *separated_between_pointwise_left*:

assumes *compactin X S*

shows $\text{separated_between } X S T \longleftrightarrow$

$(S = \{\} \longrightarrow T \subseteq \text{topspace } X) \wedge (\forall x \in S. \text{separated_between } X \{x\} T)$

(**is** *?lhs=?rhs*)

proof

assume *?lhs* **then show** *?rhs*

using *separated_between_imp_subset separated_between_mono* **by** *fastforce*

next

assume *R: ?rhs*

then have $T \subseteq \text{topspace } X$

by (*meson equals0I separated_between_imp_subset*)

show *?lhs*

proof –

obtain *U* **where** $U: \forall x \in S. \text{openin } X (U x)$

$\forall x \in S. \exists V. \text{openin } X V \wedge U x \cup V = \text{topspace } X \wedge \text{disjnt } (U x) V \wedge \{x\} \subseteq U x \wedge T \subseteq V$

using *R* **unfolding** *separated_between_def* **by** *metis*

then have $S \subseteq \bigcup (U \text{ ` } S)$

by *blast*

then obtain *K* **where** *finite K K ⊆ S* **and** $K: S \subseteq (\bigcup_{i \in K}. U i)$

using *assms U* **unfolding** *compactin_def* **by** (*smt (verit) finite_subset_image_imageE*)

show *?thesis*

unfolding *separated_between*

proof (*intro conjI exI*)

have $\bigwedge x. x \in K \implies \text{closedin } X (U x)$

by (*smt (verit) ⟨K ⊆ S⟩ Diff_cancel U(2) Un_Diff Un_Diff_Int disjnt_def openin_closedin_eq subsetD*)

then show $\text{closedin } X (\bigcup (U \text{ ` } K))$

by (*metis (mono_tags, lifting) ⟨finite K⟩ closedin_Union finite_imageI image_iff*)

show $\text{openin } X (\bigcup (U \text{ ` } K))$

using *U(1) ⟨K ⊆ S⟩* **by** *blast*

show $S \subseteq \bigcup (U \text{ ` } K)$

by (*simp add: K*)

```

have  $\bigwedge x i. \llbracket x \in T; i \in K; x \in U i \rrbracket \implies \text{False}$ 
by (meson U(2)  $\langle K \subseteq S \rangle \text{disjnt\_iff\_subsetD}$ )
then show  $T \subseteq \text{topspace } X - \bigcup (U \text{ ` } K)$ 
using  $\langle T \subseteq \text{topspace } X \rangle$  by auto
qed
qed
qed

```

lemma *separated_between_pointwise_right*:

```

compactin X T
 $\implies \text{separated\_between } X S T \longleftrightarrow (T = \{\}) \longrightarrow S \subseteq \text{topspace } X) \wedge (\forall y \in$ 
 $T. \text{separated\_between } X S \{y\})$ 
by (meson separated_between_pointwise_left separated_between_sym)

```

lemma *separated_between_closure_of*:

```

 $S \subseteq \text{topspace } X \implies \text{separated\_between } X (X \text{ closure\_of } S) T \longleftrightarrow \text{separated\_between}$ 
 $X S T$ 
by (meson closure_of_minimal_eq separated_between_alt)

```

lemma *separated_between_closure_of'*:

```

 $T \subseteq \text{topspace } X \implies \text{separated\_between } X S (X \text{ closure\_of } T) \longleftrightarrow \text{separated\_between}$ 
 $X S T$ 
by (meson separated_between_closure_of separated_between_sym)

```

lemma *separated_between_closure_of_eq*:

```

separated_between X S T  $\longleftrightarrow S \subseteq \text{topspace } X \wedge \text{separated\_between } X (X \text{ closure\_of } S) T$ 
by (metis separated_between_closure_of separated_between_imp_subset)

```

lemma *separated_between_closure_of_eq'*:

```

separated_between X S T  $\longleftrightarrow T \subseteq \text{topspace } X \wedge \text{separated\_between } X S (X \text{ closure\_of } T)$ 
by (metis separated_between_closure_of' separated_between_imp_subset)

```

lemma *separated_between_frontier_of_eq'*:

```

separated_between X S T  $\longleftrightarrow$ 
 $T \subseteq \text{topspace } X \wedge \text{disjnt } S T \wedge \text{separated\_between } X S (X \text{ frontier\_of } T)$  (is
?lhs=?rhs)

```

proof

assume ?lhs **then show** ?rhs

```

by (metis interior_of_union_frontier_of separated_between_Un separated_between_closure_of_eq'
separated_between_imp_disjoint)

```

next

assume R: ?rhs

```

then obtain U where U: closedin X U openin X U  $S \subseteq U$   $X \text{ closure\_of } T -$ 
 $X \text{ interior\_of } T \subseteq \text{topspace } X - U$ 

```

```

by (metis frontier_of_def separated_between)

```

```

show ?lhs
proof (rule separated_between_mono [of _ S X closure_of T])
  have separated_between X S T
    unfolding separated_between
  proof (intro conjI exI)
    show  $S \subseteq U - T$   $T \subseteq \text{topspace } X - (U - T)$ 
      using R U(3) by (force simp: disjnt_iff)+
    have  $T \subseteq X \text{ closure\_of } T$ 
      by (simp add: R closure_of_subset)
    then have *:  $U - T = U - X \text{ interior\_of } T$ 
      using U(4) interior_of_subset by fastforce
    then show closedin X (U - T)
      by (simp add: U(1) closedin_diff)
    have  $U \cap X \text{ frontier\_of } T = \{\}$ 
      using U(4) frontier_of_def by fastforce
    then show openin X (U - T)
      by (metis * Diff_Un U(2) Un_Diff_Int Un_Int_eq(1) closedin_closure_of
interior_of_union_frontier_of openin_diff sup_bot_right)
    qed
  then show separated_between X S (X closure_of T)
    by (simp add: R separated_between_closure_of')
  qed (auto simp: R closure_of_subset)
qed

lemma separated_between_frontier_of_eq:
  separated_between X S T  $\longleftrightarrow$   $S \subseteq \text{topspace } X \wedge \text{disjnt } S T \wedge \text{separated\_between}$ 
  X (X frontier_of S) T
  by (metis disjnt_sym separated_between_frontier_of_eq' separated_between_sym)

lemma separated_between_frontier_of:
   $\llbracket S \subseteq \text{topspace } X; \text{disjnt } S T \rrbracket$ 
   $\implies$  (separated_between X (X frontier_of S) T  $\longleftrightarrow$  separated_between X S T)
  using separated_between_frontier_of_eq by blast

lemma separated_between_frontier_of':
   $\llbracket T \subseteq \text{topspace } X; \text{disjnt } S T \rrbracket$ 
   $\implies$  (separated_between X S (X frontier_of T)  $\longleftrightarrow$  separated_between X S T)
  using separated_between_frontier_of_eq' by auto

lemma connected_space_separated_between:
  connected_space X  $\longleftrightarrow$   $(\forall S T. \text{separated\_between } X S T \longrightarrow S = \{\} \vee T = \{\})$ 
  (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis Diff_cancel connected_space_clopen_in separated_between_subset_empty)
  next
  assume ?rhs then show ?lhs
    by (meson connected_space_eq_not_separated separated_between_eq_separatedin)
qed

```

lemma *connected_space_imp_separated_between_trivial*:
connected_space X
 \implies (*separated_between* X S $T \longleftrightarrow S = \{\}$ $\wedge T \subseteq \text{topspace } X \vee S \subseteq \text{topspace } X \wedge T = \{\}$)
by (*metis connected_space_separated_between_separated_between_empty*)

6.6.3 Connected components

lemma *connected_component_of_subtopology_eq*:
connected_component_of (*subtopology* X U) $a = \text{connected_component_of } X$ a
 \longleftrightarrow
connected_component_of_set X $a \subseteq U$
by (*force simp: connected_component_of_set_connected_in_subtopology_connected_component_of_def fun_eq_iff subset_iff*)

lemma *connected_components_of_subtopology*:
assumes $C \in \text{connected_components_of } X$ $C \subseteq U$
shows $C \in \text{connected_components_of} (\text{subtopology } X$ $U)$
proof –
obtain a **where** a : *connected_component_of_set* X $a \subseteq U$ **and** $a \in \text{topspace } X$
and Ceq : $C = \text{connected_component_of_set } X$ a
using *assms* **by** (*force simp: connected_components_of_def*)
then have $a \in U$
by (*simp add: connected_component_of_refl_in_mono*)
then have *connected_component_of_set* X $a = \text{connected_component_of_set}$
(*subtopology* X U) a
by (*metis a connected_component_of_subtopology_eq*)
then show *?thesis*
by (*simp add: Ceq <a ∈ U> <a ∈ topspace X> connected_component_in_connected_components_of*)
qed

lemma *open_in_finite_connected_components*:
assumes *finite*(*connected_components_of* X) $C \in \text{connected_components_of } X$
shows *open_in* X C

proof –
have *closed_in* X (*topspace* $X - C$)
by (*metis DiffD1 assms closed_in_Union closed_in_connected_components_of complement_connected_components_of_Union finite_Diff*)
then show *?thesis*
by (*simp add: assms connected_components_of_subset open_in_closed_in*)
qed

thm *connected_component_of_eq_overlap*

lemma *connected_components_of_disjoint*:
assumes $C \in \text{connected_components_of } X$ $C' \in \text{connected_components_of } X$
shows (*disjnt* C $C' \longleftrightarrow (C \neq C')$)
using *assms nonempty_connected_components_of_pairwiseD pairwise_disjoint_connected_components_of*
by *fastforce*

lemma *connected_components_of_overlap*:

$\llbracket C \in \text{connected_components_of } X; C' \in \text{connected_components_of } X \rrbracket \implies C \cap C' \neq \{\} \iff C = C'$
by (*meson connected_components_of_disjoint disjnt_def*)

lemma *pairwise_separated_connected_components_of*:

pairwise (separatedin X) (connected_components_of X)
by (*simp add: closedin_connected_components_of connected_components_of_disjoint pairwiseI separatedin_closed_sets*)

lemma *finite_connected_components_of_finite*:

finite(tospace X) \implies finite(connected_components_of X)
by (*simp add: Union_connected_components_of finite_UnionD*)

lemma *connected_component_of_unique*:

$\llbracket x \in C; \text{connectedin } X C; \bigwedge C'. x \in C' \wedge \text{connectedin } X C' \rrbracket \implies C' \subseteq C$
 $\implies \text{connected_component_of_set } X x = C$
by (*meson connected_component_of_maximal connectedin_connected_component_of_subsetD subset_antisym*)

lemma *closedin_connected_component_of_subtopology*:

$\llbracket C \in \text{connected_components_of (subtopology } X s); X \text{ closure_of } C \subseteq s \rrbracket \implies \text{closedin } X C$
by (*metis closedin_Int_closure_of closedin_connected_components_of_closure_of_eq inf.absorb_iff2*)

lemma *connected_component_of_discrete_topology*:

connected_component_of_set (discrete_topology U) x = (if x \in U then {x} else {})
by (*simp add: locally_path_connected_space_discrete_topology flip: path_component_eq_connected_c*)

lemma *connected_components_of_discrete_topology*:

connected_components_of (discrete_topology U) = ($\lambda x. \{x\}$) ' U
by (*simp add: connected_component_of_discrete_topology connected_components_of_def*)

lemma *connected_component_of_continuous_image*:

$\llbracket \text{continuous_map } X Y f; \text{connected_component_of } X x y \rrbracket$
 $\implies \text{connected_component_of } Y (f x) (f y)$
by (*meson connected_component_of_def connectedin_continuous_map_image image_eqI*)

lemma *homeomorphic_map_connected_component_of*:

assumes *homeomorphic_map X Y f* **and** *x: x \in tospace X*
shows *connected_component_of_set Y (f x) = f ' (connected_component_of_set X x)*

proof –

obtain *g* **where** *g: continuous_map X Y f*
 $\text{continuous_map } Y X g \wedge x. x \in \text{topspace } X \implies g (f x) = x$

```

   $\bigwedge y. y \in \text{topspace } Y \implies f (g y) = y$ 
  using assms(1) homeomorphic_map_maps homeomorphic_maps_def by fastforce
  show ?thesis
  using connected_component_in_topspace [of Y] x g
        connected_component_of_continuous_image [of X Y f]
        connected_component_of_continuous_image [of Y X g]
  by force
qed

lemma homeomorphic_map_connected_components_of:
  assumes homeomorphic_map X Y f
  shows connected_components_of Y = (image f) ' (connected_components_of
X)
proof -
  have topspace Y = f ' topspace X
  by (metis assms homeomorphic_imp_surjective_map)
  with homeomorphic_map_connected_component_of [OF assms] show ?thesis
  by (auto simp: connected_components_of_def image_iff)
qed

lemma connected_component_of_pair:
  connected_component_of_set (prod_topology X Y) (x,y) =
    connected_component_of_set X x  $\times$  connected_component_of_set Y y
proof (cases x  $\in$  topspace X  $\wedge$  y  $\in$  topspace Y)
  case True
  show ?thesis
  proof (rule connected_component_of_unique)
    show (x, y)  $\in$  connected_component_of_set X x  $\times$  connected_component_of_set
Y y
    using True by (simp add: connected_component_of_refl)
    show connectedin (prod_topology X Y) (connected_component_of_set X x  $\times$ 
connected_component_of_set Y y)
    by (metis connectedin_Times connectedin_connected_component_of)
    show C  $\subseteq$  connected_component_of_set X x  $\times$  connected_component_of_set
Y y
    if (x, y)  $\in$  C  $\wedge$  connectedin (prod_topology X Y) C for C
    using that unfolding connected_component_of_def
    apply clarsimp
    by (metis (no_types) connectedin_continuous_map_image continuous_map_fst
continuous_map_snd fst_conv imageI snd_conv)
  qed
  next
  case False then show ?thesis
  by (metis Sigma_empty1 Sigma_empty2 connected_component_of_eq_empty
mem_Sigma_iff topspace_prod_topology)
qed

```

```

lemma connected_components_of_prod_topology:

```

1220

$connected_components_of (prod_topology X Y) =$
 $\{C \times D \mid C D. C \in connected_components_of X \wedge D \in connected_components_of Y\}$ (is ?lhs=?rhs)

proof

show ?lhs \subseteq ?rhs

apply (clarsimp simp: connected_components_of_def)

by (metis (no_types) connected_component_of_pair imageI)

next

show ?rhs \subseteq ?lhs

using connected_component_of_pair

by (fastforce simp: connected_components_of_def)

qed

lemma connected_component_of_product_topology:

$connected_component_of_set (product_topology X I) x =$

(if $x \in extensional I$ then $PiE I (\lambda i. connected_component_of_set (X i) (x i))$

else $\{\}$)

(is ?lhs = If _ ?R _)

proof (cases $x \in topspace(product_topology X I)$)

case True

have ?lhs = $(\Pi_E i \in I. connected_component_of_set (X i) (x i))$

if $xX: \bigwedge i. i \in I \implies x i \in topspace (X i)$ **and** $ext: x \in extensional I$

proof (rule connected_component_of_unique)

show $x \in ?R$

by (simp add: PiE_iff connected_component_of_refl local.ext xX)

show $connectedin (product_topology X I) ?R$

by (simp add: connectedin_PiE connectedin_connected_component_of)

show $C \subseteq ?R$

if $x \in C \wedge connectedin (product_topology X I) C$ **for** C

proof -

have $C \subseteq extensional I$

using PiE_def connectedin_subset_tospace **that** **by** fastforce

have $\bigwedge y. y \in C \implies y \in (\Pi i \in I. connected_component_of_set (X i) (x i))$

apply (simp add: connected_component_of_def Pi_def)

by (metis connectedin_continuous_map_image continuous_map_product_projection imageI that)

then show ?thesis

using PiE_def $\langle C \subseteq extensional I \rangle$ **by** fastforce

qed

qed

with True **show** ?thesis

by (simp add: PiE_iff)

next

case False

then show ?thesis

by (smt (verit, best) PiE_eq_empty_iff PiE_iff connected_component_of_eq_empty topspace_product_topology)

qed

lemma *connected_components_of_product_topology*:
 $connected_components_of (product_topology X I) =$
 $\{PiE I B \mid B. \forall i \in I. B i \in connected_components_of(X i)\}$ (is ?lhs=?rhs)

proof
show ?lhs \subseteq ?rhs
by (auto simp: connected_components_of_def connected_component_of_product_topology PiE_iff)
show ?rhs \subseteq ?lhs
proof
fix F
assume F \in ?rhs
then obtain B **where** Feq: F = PiE I B **and**
 $\forall i \in I. \exists x \in topspace (X i). B i = connected_component_of_set (X i) x$
by (force simp: connected_components_of_def connected_component_of_product_topology image_iff)
then obtain f **where**
 $f: \bigwedge i. i \in I \implies f i \in topspace (X i) \wedge B i = connected_component_of_set (X i) (f i)$
by metis
then have $(\lambda i \in I. f i) \in ((\Pi_E i \in I. topspace (X i)) \cap extensional I)$
by simp
with f **show** F \in ?lhs
unfolding Feq *connected_components_of_def connected_component_of_product_topology image_iff*
by (smt (verit, del_insts) PiE_cong restrict_PiE_iff restrict_apply' restrict_extensional topspace_product_topology)
qed
qed

6.6.4 Monotone maps (in the general topological sense)

definition *monotone_map*
where *monotone_map* X Y f ==
 $f ' (topspace X) \subseteq topspace Y \wedge$
 $(\forall y \in topspace Y. connectedin X \{x \in topspace X. f x = y\})$

lemma *monotone_map*:
 $monotone_map X Y f \longleftrightarrow$
 $f ' (topspace X) \subseteq topspace Y \wedge (\forall y. connectedin X \{x \in topspace X. f x = y\})$
apply (simp add: monotone_map_def)
by (metis (mono_tags, lifting) connectedin_empty [of X] Collect_empty_eq image_subset_iff)

lemma *monotone_map_in_subtopology*:
 $monotone_map X (subtopology Y S) f \longleftrightarrow monotone_map X Y f \wedge f ' (topspace X) \subseteq S$

by (smt (verit, del_insts) le_inf_iff monotone_map topspace_subtopology)

lemma *monotone_map_from_subtopology*:

assumes *monotone_map* $X Y f$

$\bigwedge x y. \llbracket x \in \text{topspace } X; y \in \text{topspace } X; x \in S; f x = f y \rrbracket \implies y \in S$

shows *monotone_map* (subtopology $X S$) $Y f$

proof –

have $\bigwedge y. y \in \text{topspace } Y \implies \text{connectedin } X \{x \in \text{topspace } X. x \in S \wedge f x = y\}$

by (smt (verit) Collect_cong assms connectedin_empty empty_def monotone_map_def)

then show ?thesis

using assms by (auto simp: monotone_map_def connectedin_subtopology)

qed

lemma *monotone_map_restriction*:

monotone_map $X Y f \wedge \{x \in \text{topspace } X. f x \in v\} = u$

$\implies \text{monotone_map}$ (subtopology $X u$) (subtopology $Y v$) f

by (smt (verit, best) IntI Int_Collect image_subset_iff mem_Collect_eq monotone_map monotone_map_from_subtopology topspace_subtopology)

lemma *injective_imp_monotone_map*:

assumes $f' \text{topspace } X \subseteq \text{topspace } Y$ *inj_on* f (topspace X)

shows *monotone_map* $X Y f$

unfolding *monotone_map_def*

proof (intro conjI assms strip)

fix y

assume $y \in \text{topspace } Y$

then have $\{x \in \text{topspace } X. f x = y\} = \{\} \vee (\exists a \in \text{topspace } X. \{x \in \text{topspace } X. f x = y\} = \{a\})$

using assms(2) unfolding *inj_on_def* by blast

then show *connectedin* $X \{x \in \text{topspace } X. f x = y\}$

by (metis (no_types, lifting) connectedin_empty connectedin_sing)

qed

lemma *embedding_imp_monotone_map*:

embedding_map $X Y f \implies \text{monotone_map}$ $X Y f$

by (metis (no_types) embedding_map_def homeomorphic_eq_everything_map inf.absorb_iff2 injective_imp_monotone_map topspace_subtopology)

lemma *section_imp_monotone_map*:

section_map $X Y f \implies \text{monotone_map}$ $X Y f$

by (simp add: embedding_imp_monotone_map section_imp_embedding_map)

lemma *homeomorphic_imp_monotone_map*:

homeomorphic_map $X Y f \implies \text{monotone_map}$ $X Y f$

by (meson section_and_retraction_eq_homeomorphic_map section_imp_monotone_map)

proposition *connected_space_monotone_quotient_map_preimage*:

assumes f : *monotone_map* $X Y f$ *quotient_map* $X Y f$ **and** *connected_space* Y

```

shows connected_space X
proof (rule ccontr)
  assume  $\neg$  connected_space X
  then obtain U V where openin X U openin X V  $U \cap V = \{\}$ 
    U  $\neq \{\}$  V  $\neq \{\}$  and topUV: topspace X  $\subseteq U \cup V$ 
    by (auto simp: connected_space_def)
  then have UVsub: U  $\subseteq$  topspace X V  $\subseteq$  topspace X
    by (auto simp: openin_subset)
  have  $\neg$  connected_space Y
    unfolding connected_space_def not_not
  proof (intro exI conjI)
    show topspace Y  $\subseteq f'U \cup f'V$ 
      by (metis f(2) image_Un quotient_imp_surjective_map subset_Un_eq
topUV)
    show  $f'U \neq \{\}$ 
      by (simp add:  $\langle U \neq \{\} \rangle$ )
    show  $(f'V) \neq \{\}$ 
      by (simp add:  $\langle V \neq \{\} \rangle$ )
    have *:  $y \notin f'V$  if  $y \in f'U$  for y
    proof -
      have  $\S$ : connectedin X  $\{x \in \text{topspace } X. f x = y\}$ 
        using f(1) monotone_map by fastforce
      show ?thesis
        using connectedinD [OF  $\S$   $\langle \text{openin } X U \rangle \langle \text{openin } X V \rangle$ ] UVsub topUV  $\langle U$ 
 $\cap V = \{\} \rangle$  that
        by (force simp: disjoint_iff)
    qed
    then show  $f'U \cap f'V = \{\}$ 
      by blast
    show openin Y  $(f'U)$ 
      using f  $\langle \text{openin } X U \rangle$  topUV * unfolding quotient_map_saturated_open
by force
    show openin Y  $(f'V)$ 
      using f  $\langle \text{openin } X V \rangle$  topUV * unfolding quotient_map_saturated_open
by force
    qed
    then show False
      by (simp add: assms)
  qed

```

lemma connectedin_monotone_quotient_map_preimage:

assumes monotone_map X Y f quotient_map X Y f connectedin Y C openin Y C \vee closedin Y C

shows connectedin X $\{x \in \text{topspace } X. f x \in C\}$

proof -

have connected_space (subtopology X $\{x \in \text{topspace } X. f x \in C\})$

proof -

have connected_space (subtopology Y C)

using $\langle \text{connectedin } Y C \rangle$ connectedin_def by blast

```

    moreover have quotient_map (subtopology X {a ∈ topspace X. f a ∈ C})
(subtopology Y C) f
    by (simp add: assms quotient_map_restriction)
    ultimately show ?thesis
    using ‹monotone_map X Y f› connected_space_monotone_quotient_map_preimage
monotone_map_restriction by blast
    qed
    then show ?thesis
    by (simp add: connectedin_def)
  qed

```

lemma *monotone_open_map*:

assumes *continuous_map X Y f open_map X Y f and fim*: $f \text{ ' (topspace X) = topspace Y}$

shows $\text{monotone_map X Y f} \longleftrightarrow (\forall C. \text{connectedin Y C} \longrightarrow \text{connectedin X } \{x \in \text{topspace X. } f x \in C\})$
(is ?lhs=?rhs)

proof

assume *L*: ?lhs

show ?rhs

unfolding *connectedin_def*

proof (*intro strip conjI*)

fix *C*

assume *C*: $C \subseteq \text{topspace Y} \wedge \text{connected_space (subtopology Y C)}$

show $\text{connected_space (subtopology X } \{x \in \text{topspace X. } f x \in C\})$

proof (*rule connected_space_monotone_quotient_map_preimage*)

show $\text{monotone_map (subtopology X } \{x \in \text{topspace X. } f x \in C\}) \text{ (subtopology Y C) f}$

by (*simp add: L monotone_map_restriction*)

show $\text{quotient_map (subtopology X } \{x \in \text{topspace X. } f x \in C\}) \text{ (subtopology Y C) f}$

proof (*rule continuous_open_imp_quotient_map*)

show $\text{continuous_map (subtopology X } \{x \in \text{topspace X. } f x \in C\}) \text{ (subtopology Y C) f}$

using *assms continuous_map_from_subtopology continuous_map_in_subtopology*
by *fastforce*

qed (*use open_map_restriction assms in fastforce*)+

qed (*simp add: C*)

qed *auto*

next

assume ?rhs

then have $\forall y. \text{connectedin Y } \{y\} \longrightarrow \text{connectedin X } \{x \in \text{topspace X. } f x = y\}$

by (*smt (verit) Collect_cong singletonD singletonI*)

then show ?lhs

by (*simp add: fim monotone_map_def*)

qed

lemma *monotone_closed_map*:

assumes *continuous_map X Y f closed_map X Y f and fim*: $f \text{ ' (topspace X) =$

```

topspace Y
  shows monotone_map X Y f  $\longleftrightarrow$  ( $\forall C. \text{connectedin } Y C \longrightarrow \text{connectedin } X \{x \in \text{topspace } X. f x \in C\}$ )
  (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
    unfolding connectedin_def
  proof (intro strip conjI)
    fix C
    assume C:  $C \subseteq \text{topspace } Y \wedge \text{connected\_space } (\text{subtopology } Y C)$ 
    show  $\text{connected\_space } (\text{subtopology } X \{x \in \text{topspace } X. f x \in C\})$ 
    proof (rule connected_space_monotone_quotient_map_preimage)
      show  $\text{monotone\_map } (\text{subtopology } X \{x \in \text{topspace } X. f x \in C\}) (\text{subtopology } Y C) f$ 
        by (simp add: L monotone_map_restriction)
      show  $\text{quotient\_map } (\text{subtopology } X \{x \in \text{topspace } X. f x \in C\}) (\text{subtopology } Y C) f$ 
        proof (rule continuous_closed_imp_quotient_map)
          show  $\text{continuous\_map } (\text{subtopology } X \{x \in \text{topspace } X. f x \in C\}) (\text{subtopology } Y C) f$ 
            using assms continuous_map_from_subtopology continuous_map_in_subtopology
          by fastforce
          qed (use closed_map_restriction assms in fastforce)+
          qed (simp add: C)
          qed auto
    next
    assume ?rhs
    then have  $\forall y. \text{connectedin } Y \{y\} \longrightarrow \text{connectedin } X \{x \in \text{topspace } X. f x = y\}$ 
      by (smt (verit) Collect_cong singletonD singletonI)
    then show ?lhs
      by (simp add: fim_monotone_map_def)
  qed

```

6.6.5 Other countability properties

definition *second_countable*

where $\text{second_countable } X \equiv$
 $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } X V) \wedge$
 $(\forall U x. \text{openin } X U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U))$

definition *first_countable*

where $\text{first_countable } X \equiv$
 $\forall x \in \text{topspace } X.$
 $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } X V) \wedge$
 $(\forall U. \text{openin } X U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U))$

definition *separable_space*

where $\text{separable_space } X \equiv$

$\exists C. \text{countable } C \wedge C \subseteq \text{topspace } X \wedge X \text{ closure_of } C = \text{topspace } X$

lemma *second_countable*:

second_countable $X \longleftrightarrow$

$(\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge \text{openin } X = \text{arbitrary_union_of } (\lambda x. x \in \mathcal{B}))$

by (*smt* (*verit*) *openin_topology_base_unique* *second_countable_def*)

lemma *second_countable_subtopology*:

assumes *second_countable* X

shows *second_countable* (*subtopology* $X S$)

proof –

obtain \mathcal{B} **where** $\mathcal{B}: \text{countable } \mathcal{B} \wedge V. V \in \mathcal{B} \implies \text{openin } X V$

$\wedge U x. \text{openin } X U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$

using *assms* **by** (*auto simp: second_countable_def*)

show *?thesis*

unfolding *second_countable_def*

proof (*intro exI conjI*)

show $\forall V \in ((\cap)S) \text{ ' } \mathcal{B}. \text{openin } (\text{subtopology } X S) V$

using *openin_subtopology_Int2* \mathcal{B} **by** *blast*

show $\forall U x. \text{openin } (\text{subtopology } X S) U \wedge x \in U \longrightarrow (\exists V \in ((\cap)S) \text{ ' } \mathcal{B}. x \in V \wedge V \subseteq U)$

using \mathcal{B} **unfolding** *openin_subtopology*

by (*smt* (*verit*, *del_insts*) *IntI image_iff inf_commute inf_le1 subset_iff*)

qed (*use* \mathcal{B} **in** *auto*)

qed

lemma *second_countable_discrete_topology*:

second_countable(*discrete_topology* U) $\longleftrightarrow \text{countable } U$ (**is** *?lhs=?rhs*)

proof

assume $L: ?lhs$

then

obtain \mathcal{B} **where** $\mathcal{B}: \text{countable } \mathcal{B} \wedge V. V \in \mathcal{B} \implies V \subseteq U$

$\wedge W x. W \subseteq U \wedge x \in W \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq W)$

by (*auto simp: second_countable_def*)

then have $\{x\} \in \mathcal{B}$ **if** $x \in U$ **for** x

by (*metis empty_subsetI insertCI insert_subset subset_antisym that*)

then show *?rhs*

by (*smt* (*verit*) *countable_subset image_subsetI* $\langle \text{countable } \mathcal{B} \rangle$ *countable_image_inj_on* [*OF inj_singleton*])

next

assume *?rhs*

then show *?lhs*

unfolding *second_countable_def*

by (*rule_tac* $x=(\lambda x. \{x\})$ $\text{ ' } U$ **in** *exI*) *auto*

qed

lemma *second_countable_open_map_image*:

assumes *continuous_map* $X Y f$ *open_map* $X Y f$

```

  and fm:  $f^{-1}(\text{topspace } X) = \text{topspace } Y$  and second_countable X
  shows second_countable Y
  proof -
    have openXYf:  $\bigwedge U. \text{openin } X \ U \longrightarrow \text{openin } Y \ (f^{-1} \ U)$ 
      using assms by (auto simp: open_map_def)
    obtain  $\mathcal{B}$  where  $\mathcal{B}$ : countable  $\mathcal{B} \wedge V. V \in \mathcal{B} \implies \text{openin } X \ V$ 
      and  $*$ :  $\bigwedge U \ x. \text{openin } X \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$ 
      using assms by (auto simp: second_countable_def)
    show ?thesis
      unfolding second_countable_def
    proof (intro exI conjI strip)
      fix V y
      assume  $V: \text{openin } Y \ V \wedge y \in V$ 
      then obtain x where  $x \in \text{topspace } X$  and  $x: f \ x = y$ 
        by (metis fm image_iff openin_subset subsetD)

      then obtain W where  $W \in \mathcal{B} \ x \in W \ W \subseteq \{x \in \text{topspace } X. f \ x \in V\}$ 
        using  $*$  [of  $\{x \in \text{topspace } X. f \ x \in V\} \ x$ ] V assms openin_continuous_map_preimage

      by force
      then show  $\exists W \in (\text{image } f)^{-1} \ \mathcal{B}. y \in W \wedge W \subseteq V$ 
        using x by auto
    qed (use  $\mathcal{B}$  openXYf in auto)
  qed

  lemma homeomorphic_space_second_countability:
     $X \text{ homeomorphic\_space } Y \implies (\text{second\_countable } X \longleftrightarrow \text{second\_countable } Y)$ 
    by (meson homeomorphic_eq_everything_map homeomorphic_space homeomorphic_space_sym second_countable_open_map_image)

  lemma second_countable_retraction_map_image:
     $[\text{retraction\_map } X \ Y \ r; \text{second\_countable } X] \implies \text{second\_countable } Y$ 
    using hereditary_imp_retractive_property homeomorphic_space_second_countability second_countable_subtopology by blast

  lemma second_countable_imp_first_countable:
     $\text{second\_countable } X \implies \text{first\_countable } X$ 
    by (metis first_countable_def second_countable_def)

  lemma first_countable_subtopology:
    assumes first_countable X
    shows first_countable (subtopology X S)
    unfolding first_countable_def
  proof
    fix x
    assume  $x \in \text{topspace } (\text{subtopology } X \ S)$ 
    then obtain  $\mathcal{B}$  where countable  $\mathcal{B}$  and  $\mathcal{B}: \bigwedge V. V \in \mathcal{B} \implies \text{openin } X \ V$ 
       $\bigwedge U. \text{openin } X \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$ 
      using assms first_countable_def by force
  
```

```

show  $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } (\text{subtopology } X \ S) \ V) \wedge (\forall U. \text{openin } (\text{subtopology } X \ S) \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U))$ 
proof (intro exI conjI strip)
  show  $\text{countable } (((\cap)S) \ ' \ \mathcal{B})$ 
    using  $\langle \text{countable } \mathcal{B} \rangle$  by blast
  show  $\text{openin } (\text{subtopology } X \ S) \ V$  if  $V \in ((\cap)S) \ ' \ \mathcal{B}$  for  $V$ 
    using  $\mathcal{B} \ \text{openin\_subtopology\_Int2}$  that by fastforce
  show  $\exists V \in ((\cap)S) \ ' \ \mathcal{B}. x \in V \wedge V \subseteq U$ 
    if  $\text{openin } (\text{subtopology } X \ S) \ U \wedge x \in U$  for  $U$ 
    using that  $\mathcal{B}(2)$  by (clarsimp simp: openin_subtopology) (meson le_infI2)
qed
qed

```

```

lemma first_countable_discrete_topology:
  first_countable (discrete_topology U)
  unfolding first_countable_def topspace_discrete_topology openin_discrete_topology
proof
  fix  $x$  assume  $x \in U$ 
  show  $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. V \subseteq U) \wedge (\forall Ua. Ua \subseteq U \wedge x \in Ua \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq Ua))$ 
    using  $\langle x \in U \rangle$  by (rule_tac  $x = \{x\}$  in exI) auto
qed

```

```

lemma first_countable_open_map_image:
  assumes continuous_map X Y f open_map X Y f
  and  $\text{fim: } f \ ' \ (\text{topspace } X) = \text{topspace } Y$  and first_countable X
shows first_countable Y
  unfolding first_countable_def
proof
  fix  $y$ 
  assume  $y \in \text{topspace } Y$ 
  have  $\text{openXYf: } \bigwedge U. \text{openin } X \ U \longrightarrow \text{openin } Y \ (f \ ' \ U)$ 
    using  $\text{assms}$  by (auto simp: open_map_def)
  then obtain  $x$  where  $x: x \in \text{topspace } X \ \text{f } x = y$ 
    by (metis  $\langle y \in \text{topspace } Y \rangle \ \text{fim} \ \text{imageE}$ )
  obtain  $\mathcal{B}$  where  $\mathcal{B}: \text{countable } \mathcal{B} \wedge \forall V. V \in \mathcal{B} \implies \text{openin } X \ V$ 
    and  $*$ :  $\bigwedge U. \text{openin } X \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$ 
    using  $\text{assms } x$  first_countable_def by force
  show  $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } Y \ V) \wedge (\forall U. \text{openin } Y \ U \wedge y \in U \longrightarrow (\exists V \in \mathcal{B}. y \in V \wedge V \subseteq U))$ 
proof (intro exI conjI strip)
  fix  $V$  assume  $\text{openin } Y \ V \wedge y \in V$ 
  then have  $\exists W \in \mathcal{B}. x \in W \wedge W \subseteq \{x \in \text{topspace } X. \text{f } x \in V\}$ 
    using  $*$  [of  $\{x \in \text{topspace } X. \text{f } x \in V\}$ ]  $\text{assms}$  openin_continuous_map_preimage
  by fastforce
  then show  $\exists V' \in (\text{image } f) \ ' \ \mathcal{B}. y \in V' \wedge V' \subseteq V$ 
    using image_mono  $x$  by auto

```


qed (use \mathcal{B} openXYf in force)+
qed

lemma *homeomorphic_space_first_countability:*

X homeomorphic_space $Y \implies$ first_countable $X \longleftrightarrow$ first_countable Y
by (meson first_countable_open_map_image homeomorphic_eq_everything_map
homeomorphic_space homeomorphic_space_sym)

lemma *first_countable_retraction_map_image:*

\llbracket retraction_map X Y r ; first_countable X $\rrbracket \implies$ first_countable Y
using first_countable_subtopology hereditary_imp_retractive_property homeo-
morphic_space_first_countability **by** blast

lemma *separable_space_open_subset:*

assumes separable_space X openin X S
shows separable_space (subtopology X S)

proof –

obtain C **where** C : countable C $C \subseteq$ topspace X X closure_of $C =$ topspace X

by (meson assms separable_space_def)

then have $\bigwedge x T. \llbracket x \in$ topspace X ; $x \in T$; openin (subtopology X S) T \rrbracket

$\implies \exists y. y \in S \wedge y \in C \wedge y \in T$

by (smt (verit) \langle openin X S \rangle in_closure_of openin_open_subtopology subsetD)

with C \langle openin X S \rangle **show** ?thesis

unfolding separable_space_def

by (rule_tac $x=S \cap C$ **in** exI) (force simp: in_closure_of)

qed

lemma *separable_space_continuous_map_image:*

assumes separable_space X continuous_map X Y f

and fim: $f ' ($ topspace X) = topspace Y

shows separable_space Y

proof –

have cont: $\bigwedge S. f ' (X$ closure_of $S) \subseteq Y$ closure_of $f ' S$

by (simp add: assms continuous_map_image_closure_subset)

obtain C **where** C : countable C $C \subseteq$ topspace X X closure_of $C =$ topspace X

by (meson assms separable_space_def)

then show ?thesis

unfolding separable_space_def

by (metis cont fim closure_of_subset_topspace countable_image image_mono
subset_antisym)

qed

lemma *separable_space_quotient_map_image:*

\llbracket quotient_map X Y q ; separable_space X $\rrbracket \implies$ separable_space Y

by (meson quotient_imp_continuous_map quotient_imp_surjective_map separa-
ble_space_continuous_map_image)

lemma *separable_space_retraction_map_image:*

\llbracket retraction_map X Y r ; separable_space X $\rrbracket \implies$ separable_space Y

using *retraction_imp_quotient_map separable_space_quotient_map_image* **by**
blast

lemma *homeomorphic_separable_space*:

X *homeomorphic_space* $Y \implies$ (*separable_space* $X \longleftrightarrow$ *separable_space* Y)
by (*meson homeomorphic_eq_everything_map homeomorphic_maps_map homeomorphic_space_def separable_space_continuous_map_image*)

lemma *separable_space_discrete_topology*:

separable_space(*discrete_topology* U) \longleftrightarrow *countable* U
by (*metis countable_Int2 discrete_topology_closure_of_dual_order.refl inf.orderE separable_space_def topspace_discrete_topology*)

lemma *second_countable_imp_separable_space*:

assumes *second_countable* X
shows *separable_space* X

proof –

obtain \mathcal{B} **where** \mathcal{B} : *countable* $\mathcal{B} \wedge V. V \in \mathcal{B} \implies$ *openin* $X V$

and $*$: $\bigwedge U x. \text{openin } X U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$

using *assms* **by** (*auto simp: second_countable_def*)

obtain c **where** c : $\bigwedge V. \llbracket V \in \mathcal{B}; V \neq \{\} \rrbracket \implies c V \in V$

by (*metis all_not_in_conv*)

then have $*$: $\bigwedge x. x \in \text{topspace } X \implies x \in X \text{ closure_of } c \text{ ' } (\mathcal{B} - \{\{\}\})$

using $*$ **by** (*force simp: closure_of_def*)

show *?thesis*

unfolding *separable_space_def*

proof (*intro exI conjI*)

show *countable* ($c \text{ ' } (\mathcal{B} - \{\{\}\})$)

using $\mathcal{B}(1)$ **by** *blast*

show ($c \text{ ' } (\mathcal{B} - \{\{\}\}) \subseteq \text{topspace } X$)

using $\mathcal{B}(2)$ c *openin_subset* **by** *fastforce*

show $X \text{ closure_of } (c \text{ ' } (\mathcal{B} - \{\{\}\})) = \text{topspace } X$

by (*meson ** closure_of_subset_topspace subsetI subset_antisym*)

qed

qed

lemma *second_countable_imp_Lindelof_space*:

assumes *second_countable* X
shows *Lindelof_space* X

unfolding *Lindelof_space_def*

proof *clarify*

fix \mathcal{U}

assume $\forall U \in \mathcal{U}. \text{openin } X U$ **and** $UU: \bigcup \mathcal{U} = \text{topspace } X$

obtain \mathcal{B} **where** \mathcal{B} : *countable* $\mathcal{B} \wedge V. V \in \mathcal{B} \implies$ *openin* $X V$

and $*$: $\bigwedge U x. \text{openin } X U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$

using *assms* **by** (*auto simp: second_countable_def*)

define \mathcal{B}' **where** $\mathcal{B}' = \{B \in \mathcal{B}. \exists U. U \in \mathcal{U} \wedge B \subseteq U\}$

have \mathcal{B}' : *countable* $\mathcal{B}' \cup \mathcal{B}' = \bigcup \mathcal{U}$

using \mathcal{B} **using** $*$ $\langle \forall U \in \mathcal{U}. \text{openin } X U \rangle$ **by** (*fastforce simp: \mathcal{B}'_def*) $+$

```

have  $\bigwedge b. \exists U. b \in \mathcal{B}' \longrightarrow U \in \mathcal{U} \wedge b \subseteq U$ 
  by (simp add:  $\mathcal{B}'\_def$ )
then obtain  $G$  where  $G: \bigwedge b. b \in \mathcal{B}' \longrightarrow G b \in \mathcal{U} \wedge b \subseteq G b$ 
  by metis
with  $\mathcal{B}' UU$  show  $\exists \mathcal{V}. \text{countable } \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup \mathcal{V} = \text{topspace } X$ 
  by (rule_tac x=G '  $\mathcal{B}'$  in exI) fastforce
qed

```

6.6.6 Neighbourhood bases EXTRAS

Neighbourhood bases: useful for "local" properties of various kinds

lemma *openin_topology_neighbourhood_base_unique:*

openin X = arbitrary_union_of P \longleftrightarrow

($\forall u. P u \longrightarrow \text{openin } X u$) \wedge neighbourhood_base_of P X

by (*smt (verit, best) open_neighbourhood_base_of openin_topology_base_unique*)

lemma *neighbourhood_base_at_topology_base:*

openin X = arbitrary_union_of b

\implies (neighbourhood_base_at x P X \longleftrightarrow

($\forall w. b w \wedge x \in w \longrightarrow (\exists u v. \text{openin } X u \wedge P v \wedge x \in u \wedge u \subseteq v \wedge v$

$\subseteq w$)))

apply (*simp add: neighbourhood_base_at_def*)

by (*smt (verit, del_insts) openin_topology_base_unique subset_trans*)

lemma *neighbourhood_base_of_unlocalized:*

assumes $\bigwedge S t. P S \wedge \text{openin } X t \wedge (t \neq \{\}) \wedge t \subseteq S \implies P t$

shows *neighbourhood_base_of P X \longleftrightarrow*

($\forall x \in \text{topspace } X. \exists u v. \text{openin } X u \wedge P v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq \text{topspace}$

X)

apply (*simp add: neighbourhood_base_of_def*)

by (*smt (verit, ccfv_SIG) assms_empty_iff neighbourhood_base_at_unlocalized*)

lemma *neighbourhood_base_at_discrete_topology:*

neighbourhood_base_at x P (discrete_topology u) \longleftrightarrow $x \in u \implies P \{x\}$

apply (*simp add: neighbourhood_base_at_def*)

by (*smt (verit) empty_iff empty_subsetI insert_subset singletonI subsetD subset_singletonD*)

lemma *neighbourhood_base_of_discrete_topology:*

neighbourhood_base_of P (discrete_topology u) \longleftrightarrow ($\forall x \in u. P \{x\}$)

apply (*simp add: neighbourhood_base_of_def*)

using *neighbourhood_base_at_discrete_topology[of _ P u]*

by (*metis empty_subsetI insert_subset neighbourhood_base_at_def openin_discrete_topology singletonI*)

lemma *second_countable_neighbourhood_base_alt:*

second_countable X \longleftrightarrow

($\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } X V) \wedge \text{neighbourhood_base_of } (\lambda A. A \in \mathcal{B})$

X)

by (metis (full_types) openin_topology_neighbourhood_base_unique second_countable)

lemma first_countable_neighbourhood_base_alt:

first_countable $X \longleftrightarrow$
 $(\forall x \in \text{topspace } X. \exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } X V) \wedge \text{neighbourhood_base_at } x (\lambda V. V \in \mathcal{B}) X)$
unfolding first_countable_def
apply (intro ball_cong refl ex_cong conj_cong)
by (metis (mono_tags, lifting) open_neighbourhood_base_at)

lemma second_countable_neighbourhood_base:

second_countable $X \longleftrightarrow$
 $(\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge \text{neighbourhood_base_of } (\lambda V. V \in \mathcal{B}) X)$ (is ?lhs=?rhs)
proof
assume ?lhs
then show ?rhs
using second_countable_neighbourhood_base_alt **by** blast
next
assume ?rhs
then obtain \mathcal{B} **where** countable \mathcal{B}
and $\mathcal{B}: \bigwedge W x. \text{openin } X W \wedge x \in W \longrightarrow (\exists U. \text{openin } X U \wedge (\exists V. V \in \mathcal{B} \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$
by (metis neighbourhood_base_of)
then show ?lhs
unfolding second_countable_neighbourhood_base_alt neighbourhood_base_of
apply (rule_tac $x=(\lambda u. X \text{interior_of } u)$ ' \mathcal{B} in exI)
by (smt (verit, best) interior_of_eq interior_of_mono countable_image image_iff openin_interior_of)
qed

lemma first_countable_neighbourhood_base:

first_countable $X \longleftrightarrow$
 $(\forall x \in \text{topspace } X. \exists \mathcal{B}. \text{countable } \mathcal{B} \wedge \text{neighbourhood_base_at } x (\lambda V. V \in \mathcal{B}) X)$ (is ?lhs=?rhs)
proof
assume ?lhs
then show ?rhs
by (metis first_countable_neighbourhood_base_alt)
next
assume $R: ?rhs$
show ?lhs
unfolding first_countable_neighbourhood_base_alt
proof
fix x
assume $x \in \text{topspace } X$
with R **obtain** \mathcal{B} **where** countable \mathcal{B} **and** $\mathcal{B}: \text{neighbourhood_base_at } x (\lambda V. V \in \mathcal{B}) X$
by blast
then

```

show  $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge \text{Ball } \mathcal{B} (\text{openin } X) \wedge \text{neighbourhood\_base\_at } x (\lambda V. V \in \mathcal{B}) X$ 
  unfolding neighbourhood_base_at_def
  apply (rule_tac  $x = (\lambda u. X \text{interior\_of } u)$  ‘ $\mathcal{B}$  in exI)
  by (smt (verit, best) countable_image_image_iff_interior_of_eq_interior_of_mono
openin_interior_of)
qed
qed

```

6.6.7 T_0 spaces and the Kolmogorov quotient

definition *t0_space* where

```

t0_space  $X \equiv$ 
   $\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. x \neq y \longrightarrow (\exists U. \text{openin } X U \wedge (x \notin U \longleftrightarrow y \in U))$ 

```

lemma *t0_space_expansive*:

```

 $\llbracket \text{topspace } Y = \text{topspace } X; \bigwedge U. \text{openin } X U \implies \text{openin } Y U \rrbracket \implies \text{t0\_space } X \implies \text{t0\_space } Y$ 
by (metis t0_space_def)

```

lemma *t1_imp_t0_space*: $t1_space X \implies t0_space X$

by (*metis* *t0_space_def* *t1_space_def*)

lemma *t1_eq_symmetric_t0_space_alt*:

```

t1_space  $X \longleftrightarrow$ 
   $t0\_space X \wedge$ 
   $(\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. x \in X \text{closure\_of } \{y\} \longleftrightarrow y \in X \text{closure\_of } \{x\})$ 
apply (simp add: t0_space_def t1_space_def closure_of_def)
by (smt (verit, best) openin_topspace)

```

lemma *t1_eq_symmetric_t0_space*:

```

t1_space  $X \longleftrightarrow t0\_space X \wedge (\forall x y. x \in X \text{closure\_of } \{y\} \longleftrightarrow y \in X \text{closure\_of } \{x\})$ 
by (auto simp: t1_eq_symmetric_t0_space_alt_in_closure_of)

```

lemma *Hausdorff_imp_t0_space*:

```

Hausdorff_space  $X \implies t0\_space X$ 
by (simp add: Hausdorff_imp_t1_space t1_imp_t0_space)

```

lemma *t0_space*:

```

t0_space  $X \longleftrightarrow$ 
   $(\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. x \neq y \longrightarrow (\exists C. \text{closedin } X C \wedge (x \notin C \longleftrightarrow y \in C)))$ 
unfolding t0_space_def by (metis Diff_iff_closedin_def openin_closedin_eq)

```

lemma *homeomorphic_t0_space*:

assumes X *homeomorphic_space* Y

1234

shows $t0_space\ X \longleftrightarrow t0_space\ Y$
proof –
 obtain f **where** $f: homeomorphic_map\ X\ Y\ f$ **and** $F: inj_on\ f\ (topspace\ X)$
and $topspace\ Y = f\ 'topspace\ X$
 by ($metis\ assms\ homeomorphic_imp_injective_map\ homeomorphic_imp_surjective_map\ homeomorphic_space$)
 with $inj_on_image_mem_iff\ [OF\ F]$
 show $?thesis$
 apply ($simp\ add: t0_space_def\ homeomorphic_eq_everything_map\ continuous_map_def\ open_map_def\ inj_on_def$)
 by ($smt\ (verit)\ mem_Collect_eq\ openin_subset$)
qed

lemma $t0_space_closure_of_sing$:
 $t0_space\ X \longleftrightarrow$
 $(\forall x \in topspace\ X. \forall y \in topspace\ X. X\ closure_of\ \{x\} = X\ closure_of\ \{y\}$
 $\longrightarrow x = y)$
 by ($simp\ add: t0_space_def\ closure_of_def\ set_eq_iff$) ($smt\ (verit)$)

lemma $t0_space_discrete_topology$: $t0_space\ (discrete_topology\ S)$
 by ($simp\ add: Hausdorff_imp_t0_space$)

lemma $t0_space_subtopology$: $t0_space\ X \implies t0_space\ (subtopology\ X\ U)$
 by ($simp\ add: t0_space_def\ openin_subtopology$) ($metis\ Int_iff$)

lemma $t0_space_retraction_map_image$:
 $\llbracket retraction_map\ X\ Y\ r; t0_space\ X \rrbracket \implies t0_space\ Y$
 using $hereditary_imp_retractive_property\ homeomorphic_t0_space\ t0_space_subtopology$
by $blast$

lemma $t0_space_prod_topologyI$: $\llbracket t0_space\ X; t0_space\ Y \rrbracket \implies t0_space\ (prod_topology\ X\ Y)$
 by ($simp\ add: t0_space_closure_of_sing\ closure_of_Times\ closure_of_eq_empty_gen\ times_eq_iff\ flip: sing_Times_sing\ insert_Times_insert$)

lemma $t0_space_prod_topology_iff$:
 $t0_space\ (prod_topology\ X\ Y) \longleftrightarrow prod_topology\ X\ Y = trivial_topology \vee$
 $t0_space\ X \wedge t0_space\ Y$ (**is** $?lhs=?rhs$)
proof
 assume $?lhs$
 then show $?rhs$
 by ($metis\ prod_topology_trivial_iff\ retraction_map_fst\ retraction_map_snd\ t0_space_retraction_map_image$)
qed ($metis\ t0_space_discrete_topology\ t0_space_prod_topologyI$)

proposition $t0_space_product_topology$:
 $t0_space\ (product_topology\ X\ I) \longleftrightarrow product_topology\ X\ I = trivial_topology$
 $\vee (\forall i \in I. t0_space\ (X\ i))$

```

  (is ?lhs=?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (meson retraction_map_product_projection t0_space_retraction_map_image)
next
  assume R: ?rhs
  show ?lhs
  proof (cases product_topology X I = trivial_topology)
    case True
    then show ?thesis
      by (simp add: t0_space_def)
  next
    case False
    show ?thesis
      unfolding t0_space
      proof (intro strip)
        fix x y
        assume x: x ∈ topspace (product_topology X I)
          and y: y ∈ topspace (product_topology X I)
          and x ≠ y
        then obtain i where i ∈ I x i ≠ y i
          by (metis PiE_ext topspace_product_topology)
        then have t0_space (X i)
          using False R by blast
        then obtain U where closedin (X i) U (x i ∉ U ⟷ y i ∈ U)
          by (metis t0_space PiE_mem ⟨i ∈ I⟩ ⟨x i ≠ y i⟩ topspace_product_topology
x y)
        with ⟨i ∈ I⟩ x y show ∃ U. closedin (product_topology X I) U ∧ (x ∉ U) =
(y ∈ U)
          by (rule_tac x=PiE I (λj. if j = i then U else topspace(X j)) in exI)
            (simp add: closedin_product_topology PiE_iff)
      qed
    qed
  qed

```

6.6.8 Kolmogorov quotients

definition *Kolmogorov_quotient*

where $Kolmogorov_quotient\ X \equiv \lambda x. @y. \forall U. openin\ X\ U \longrightarrow (y \in U \longleftrightarrow x \in U)$

lemma *Kolmogorov_quotient_in_open*:

$openin\ X\ U \implies (Kolmogorov_quotient\ X\ x \in U \longleftrightarrow x \in U)$
 by (smt (verit, ccfv_SIG) Kolmogorov_quotient_def someI_ex)

lemma *Kolmogorov_quotient_in_topspace*:

$Kolmogorov_quotient\ X\ x \in topspace\ X \longleftrightarrow x \in topspace\ X$
 by (simp add: Kolmogorov_quotient_in_open)

lemma *Kolmogorov_quotient_in_closed*:

closedin X C \implies (*Kolmogorov_quotient X* $x \in C \iff x \in C$)

unfolding *closedin_def*

by (*meson DiffD2 DiffI Kolmogorov_quotient_in_open Kolmogorov_quotient_in_topspace in_mono*)

lemma *continuous_map_Kolmogorov_quotient*:

continuous_map X X (*Kolmogorov_quotient X*)

using *Kolmogorov_quotient_in_open openin_subopen openin_subset*

by (*fastforce simp: continuous_map_def Kolmogorov_quotient_in_topspace*)

lemma *open_map_Kolmogorov_quotient_explicit*:

openin X U \implies *Kolmogorov_quotient X* ' $U = \text{Kolmogorov_quotient } X \text{ ' } \text{topspace } X \cap U$

using *Kolmogorov_quotient_in_open openin_subset* **by** *fastforce*

lemma *open_map_Kolmogorov_quotient_gen*:

open_map (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X* ' S))
(*Kolmogorov_quotient X*)

proof (*clarsimp simp: open_map_def openin_subtopology_alt image_iff*)

fix U

assume *openin X U*

then have *Kolmogorov_quotient X* ' ($S \cap U$) = *Kolmogorov_quotient X* ' $S \cap U$

using *Kolmogorov_quotient_in_open* [*of X U*] **by** *auto*

then show $\exists V. \text{openin } X \ V \wedge \text{Kolmogorov_quotient } X \text{ ' } (S \cap U) = \text{Kolmogorov_quotient } X \text{ ' } S \cap V$

using $\langle \text{openin } X \ U \rangle$ **by** *blast*

qed

lemma *open_map_Kolmogorov_quotient*:

open_map X (*subtopology X* (*Kolmogorov_quotient X* ' *topspace X*))

(*Kolmogorov_quotient X*)

by (*metis open_map_Kolmogorov_quotient_gen subtopology_topspace*)

lemma *closed_map_Kolmogorov_quotient_explicit*:

closedin X U \implies *Kolmogorov_quotient X* ' $U = \text{Kolmogorov_quotient } X \text{ ' } \text{topspace } X \cap U$

using *closedin_subset* **by** (*fastforce simp: Kolmogorov_quotient_in_closed*)

lemma *closed_map_Kolmogorov_quotient_gen*:

closed_map (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X* ' S))

(*Kolmogorov_quotient X*)

using *Kolmogorov_quotient_in_closed* **by** (*force simp: closed_map_def closedin_subtopology_alt image_iff*)

lemma *closed_map_Kolmogorov_quotient*:


```

  closed_map X (subtopology X (Kolmogorov_quotient X ' topspace X))
    (Kolmogorov_quotient X)
  by (metis closed_map_Kolmogorov_quotient_gen subtopology_topspace)

lemma quotient_map_Kolmogorov_quotient_gen:
  quotient_map (subtopology X S) (subtopology X (Kolmogorov_quotient X ' S))
    (Kolmogorov_quotient X)
proof (intro continuous_open_imp_quotient_map)
  show continuous_map (subtopology X S) (subtopology X (Kolmogorov_quotient
    X ' S)) (Kolmogorov_quotient X)
  by (simp add: continuous_map_Kolmogorov_quotient continuous_map_from_subtopology
    continuous_map_in_subtopology image_mono)
  show open_map (subtopology X S) (subtopology X (Kolmogorov_quotient X ' S))
    (Kolmogorov_quotient X)
  using open_map_Kolmogorov_quotient_gen by blast
  show Kolmogorov_quotient X ' topspace (subtopology X S) = topspace (subtopology
    X (Kolmogorov_quotient X ' S))
  by (force simp: Kolmogorov_quotient_in_open)
qed

lemma quotient_map_Kolmogorov_quotient:
  quotient_map X (subtopology X (Kolmogorov_quotient X ' topspace X)) (Kolmogorov_quotient
    X)
  by (metis quotient_map_Kolmogorov_quotient_gen subtopology_topspace)

lemma Kolmogorov_quotient_eq:
  Kolmogorov_quotient X x = Kolmogorov_quotient X y  $\longleftrightarrow$ 
    ( $\forall U. \text{openin } X \ U \longrightarrow (x \in U \longleftrightarrow y \in U)$ ) (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
  by (metis Kolmogorov_quotient_in_open)
next
  assume ?rhs then show ?lhs
  by (simp add: Kolmogorov_quotient_def)
qed

lemma Kolmogorov_quotient_eq_alt:
  Kolmogorov_quotient X x = Kolmogorov_quotient X y  $\longleftrightarrow$ 
    ( $\forall U. \text{closedin } X \ U \longrightarrow (x \in U \longleftrightarrow y \in U)$ ) (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
  by (metis Kolmogorov_quotient_in_closed)
next
  assume ?rhs then show ?lhs
  by (smt (verit) Diff_iff Kolmogorov_quotient_eq closedin_topspace in_mono
    openin_closedin_eq)
qed

lemma Kolmogorov_quotient_continuous_map:

```

1238

assumes *continuous_map* $X\ Y\ f\ t0_space\ Y$ **and** $x: x \in topspace\ X$
shows $f\ (Kolmogorov_quotient\ X\ x) = f\ x$
using *assms* **unfolding** *continuous_map_def\ t0_space_def*
by (*smt* (*verit*, *ccfv_threshold*) *Kolmogorov_quotient_in_open\ Kolmogorov_quotient_in_topspace\ PiE\ mem_Collect_eq*)

lemma *t0_space_Kolmogorov_quotient*:
t0_space (*subtopology* $X\ (Kolmogorov_quotient\ X\ 'topspace\ X)$)
apply (*clarsimp\ simp: t0_space_def*)
by (*smt* (*verit*, *best*) *Kolmogorov_quotient_eq\ imageE\ image_eqI\ open_map_Kolmogorov_quotient\ open_map_def*)

lemma *Kolmogorov_quotient_id*:
t0_space $X \implies x \in topspace\ X \implies Kolmogorov_quotient\ X\ x = x$
by (*metis\ Kolmogorov_quotient_in_open\ Kolmogorov_quotient_in_topspace\ t0_space_def*)

lemma *Kolmogorov_quotient_idemp*:
Kolmogorov_quotient $X\ (Kolmogorov_quotient\ X\ x) = Kolmogorov_quotient\ X\ x$
by (*simp\ add: Kolmogorov_quotient_eq\ Kolmogorov_quotient_in_open*)

lemma *retraction_maps_Kolmogorov_quotient*:
retraction_maps X
(*subtopology* $X\ (Kolmogorov_quotient\ X\ 'topspace\ X)$)
(*Kolmogorov_quotient* X) *id*
unfolding *retraction_maps_def\ continuous_map_in_subtopology*
using *Kolmogorov_quotient_idemp\ continuous_map_Kolmogorov_quotient* **by**
force

lemma *retraction_map_Kolmogorov_quotient*:
retraction_map X
(*subtopology* $X\ (Kolmogorov_quotient\ X\ 'topspace\ X)$)
(*Kolmogorov_quotient* X)
using *retraction_map_def\ retraction_maps_Kolmogorov_quotient* **by** *blast*

lemma *retract_of_space_Kolmogorov_quotient_image*:
Kolmogorov_quotient $X\ 'topspace\ X\ retract_of_space\ X$
proof –
have *continuous_map* $X\ X\ (Kolmogorov_quotient\ X)$
by (*simp\ add: continuous_map_Kolmogorov_quotient*)
then have *Kolmogorov_quotient* $X\ 'topspace\ X \subseteq topspace\ X$
by (*simp\ add: continuous_map_image_subset_topspace*)
then show *?thesis*
by (*meson\ retract_of_space_retraction_maps\ retraction_maps_Kolmogorov_quotient*)
qed

lemma *Kolmogorov_quotient_lift_exists*:
assumes $S \subseteq topspace\ X\ t0_space\ Y$ **and** $f: continuous_map\ (subtopology\ X\ S)\ Y\ f$

```

obtains g where continuous_map (subtopology X (Kolmogorov_quotient X ' S))
  Y g
       $\bigwedge x. x \in S \implies g(\text{Kolmogorov\_quotient } X \ x) = f \ x$ 
proof –
  have  $\bigwedge x \ y. \llbracket x \in S; y \in S; \text{Kolmogorov\_quotient } X \ x = \text{Kolmogorov\_quotient } X \ y \rrbracket \implies f \ x = f \ y$ 
    using assms
    apply (simp add: Kolmogorov_quotient_eq t0_space_def continuous_map_def
      Int_absorb1 openin_subtopology)
    by (smt (verit, del_insts) Int_iff mem_Collect_eq Pi_iff)
    then obtain g where g: continuous_map (subtopology X (Kolmogorov_quotient
      X ' S)) Y g
      g ' (topspace X  $\cap$  Kolmogorov_quotient X ' S) = f ' S
       $\bigwedge x. x \in S \implies g(\text{Kolmogorov\_quotient } X \ x) = f \ x$ 
    using quotient_map_lift_exists [OF quotient_map_Kolmogorov_quotient_gen
      [of X S] f]
    by (metis assms(1) topspace_subtopology topspace_subtopology_subset)
    show ?thesis
    proof qed (use g in auto)
qed

```

6.6.9 Closed diagonals and graphs

lemma *Hausdorff_space_closedin_diagonal*:

Hausdorff_space X \longleftrightarrow *closedin* (*prod_topology X X*) (($\lambda x. (x,x)$) ' *topspace X*)

proof –

have $\S: ((\lambda x. (x, x)) ' \text{topspace } X) \subseteq \text{topspace } X \times \text{topspace } X$

by *auto*

show *?thesis*

apply (*simp add: closedin_def openin_prod_topology_alt Hausdorff_space_def*
disjnt_iff §)

apply (*intro all_cong1 imp_cong ex_cong1 conj_cong refl*)

by (*force dest!: openin_subset*)**+**

qed

lemma *closed_map_diag_eq*:

closed_map X (*prod_topology X X*) ($\lambda x. (x,x)$) \longleftrightarrow *Hausdorff_space X*

proof –

have *section_map X* (*prod_topology X X*) ($\lambda x. (x, x)$)

unfolding *section_map_def retraction_maps_def*

by (*smt (verit) continuous_map_fst continuous_map_of_fst continuous_map_on_empty*
continuous_map_pairwise fst_conv fst_diag_fst snd_diag_fst)

then have *embedding_map X* (*prod_topology X X*) ($\lambda x. (x, x)$)

by (*rule section_imp_embedding_map*)

then show *?thesis*

using *Hausdorff_space_closedin_diagonal embedding_imp_closed_map_eq* **by**
blast

qed

lemma *proper_map_diag_eq* [*simp*]:
 $\text{proper_map } X (\text{prod_topology } X X) (\lambda x. (x,x)) \longleftrightarrow \text{Hausdorff_space } X$
by (*simp add: closed_map_diag_eq inj_on_convolve_ident injective_imp_proper_eq closed_map*)

lemma *closedin_continuous_maps_eq*:
assumes *Hausdorff_space* *Y* **and** *f: continuous_map* *X Y* *f* **and** *g: continuous_map* *X Y* *g*
shows *closedin* *X* $\{x \in \text{topspace } X. f x = g x\}$
proof –
have $\S: \{x \in \text{topspace } X. f x = g x\} = \{x \in \text{topspace } X. (f x, g x) \in ((\lambda y. (y, y)) \text{ ` } \text{topspace } Y)\}$
using *f continuous_map_image_subset_topspace* **by** *fastforce*
show *?thesis*
unfolding \S
proof (*intro closedin_continuous_map_preimage*)
show *continuous_map* *X* (*prod_topology* *Y Y*) $(\lambda x. (f x, g x))$
by (*simp add: continuous_map_pairedI f g*)
show *closedin* (*prod_topology* *Y Y*) $((\lambda y. (y, y)) \text{ ` } \text{topspace } Y)$
using *Hausdorff_space_closedin_diagonal* *assms* **by** *blast*
qed
qed

lemma *forall_in_closure_of*:
assumes $x \in X$ *closure_of* *S* $\wedge x. x \in S \implies P x$
and *closedin* *X* $\{x \in \text{topspace } X. P x\}$
shows *P x*
by (*smt (verit, ccfv_threshold) Diff_iff assms closedin_def in_closure_of mem_Collect_eq*)

lemma *forall_in_closure_of_eq*:
assumes $x: x \in X$ *closure_of* *S*
and *Y: Hausdorff_space* *Y*
and *f: continuous_map* *X Y* *f* **and** *g: continuous_map* *X Y* *g*
and *fg: $\wedge x. x \in S \implies f x = g x$*
shows $f x = g x$
proof –
have *closedin* *X* $\{x \in \text{topspace } X. f x = g x\}$
using *Y closedin_continuous_maps_eq f g* **by** *blast*
then show *?thesis*
using *forall_in_closure_of [OF x fg]*
by *fastforce*
qed

lemma *retract_of_space_imp_closedin*:
assumes *Hausdorff_space* *X* **and** *S: S retract_of_space* *X*
shows *closedin* *X* *S*
proof –
obtain *r* **where** *r: continuous_map* *X* (*subtopology* *X S*) $r \forall x \in S. r x = x$
using *assms* **by** (*meson retract_of_space_def*)
then have $\S: S = \{x \in \text{topspace } X. r x = x\}$

```

using S retract_of_space_imp_subset by (force simp: continuous_map_def
Pi_iff)
show ?thesis
unfolding §
using r continuous_map_into_fulltopology assms
by (force intro: closedin_continuous_maps_eq)
qed

lemma homeomorphic_maps_graph:
  homeomorphic_maps X (subtopology (prod_topology X Y) ((λx. (x, f x)) ‘
(topspace X)))
  (λx. (x, f x)) fst  $\longleftrightarrow$  continuous_map X Y f
  (is ?lhs=?rhs)
proof
  assume ?lhs
  then
  have h: homeomorphic_map X (subtopology (prod_topology X Y) ((λx. (x, f x))
  ‘topspace X)) (λx. (x, f x))
  by (auto simp: homeomorphic_maps_map)
  have f = snd ∘ (λx. (x, f x))
  by force
  then show ?rhs
  by (metis (no_types, lifting) h continuous_map_in_subtopology continuous_map_snd_of
homeomorphic_eq_everything_map)
  next
  assume ?rhs
  then show ?lhs
  unfolding homeomorphic_maps_def
  by (smt (verit, del_insts) continuous_map_eq continuous_map_subtopology_fst
embedding_map_def
  embedding_map_graph homeomorphic_eq_everything_map image_cong
image_iff prod.sel(1))
qed

```

6.6.10 KC spaces, those where all compact sets are closed.

definition *kc_space*

where *kc_space X* $\equiv \forall S. \text{compactin } X S \longrightarrow \text{closedin } X S$

lemma *kc_space_euclidean*: *kc_space (euclidean :: 'a::metric_space topology)*

by (*simp add: compact_imp_closed kc_space_def*)

lemma *kc_space_expansive*:

$\llbracket \text{kc_space } X; \text{topspace } Y = \text{topspace } X; \bigwedge U. \text{openin } X U \implies \text{openin } Y U \rrbracket$
 $\implies \text{kc_space } Y$

by (*meson compactin_contractive kc_space_def topology_finer_closedin*)

lemma *compactin_imp_closedin_gen*:

$\llbracket \text{kc_space } X; \text{compactin } X S \rrbracket \implies \text{closedin } X S$

1242

using *kc_space_def* **by** *blast*

lemma *Hausdorff_imp_kc_space*: *Hausdorff_space X* \implies *kc_space X*
by (*simp add: compactin_imp_closedin kc_space_def*)

lemma *kc_imp_t1_space*: *kc_space X* \implies *t1_space X*
by (*simp add: finite_imp_compactin kc_space_def t1_space_closedin_finite*)

lemma *kc_space_subtopology*:
kc_space X \implies *kc_space(subtopology X S)*
by (*metis closedin_Int_closure_of_closure_of_eq compactin_subtopology inf.absorb2 kc_space_def*)

lemma *kc_space_discrete_topology*:
kc_space(discrete_topology U)
using *Hausdorff_space_discrete_topology compactin_imp_closedin kc_space_def*
by *blast*

lemma *kc_space_continuous_injective_map_preimage*:
assumes *kc_space Y continuous_map X Y f* **and** *inj: inj_on f (topspace X)*
shows *kc_space X*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *S: compactin X S*
have $S = \{x \in \text{topspace } X. f\ x \in f\ 'S\}$
using *S compactin_subset_topspace inj_onD [OF inj]* **by** *blast*
with *assms S* **show** *closedin X S*
by (*metis (no_types, lifting) Collect_cong closedin_continuous_map_preimage compactin_imp_closedin_gen image_compactin*)
qed

lemma *kc_space_retraction_map_image*:
assumes *retraction_map X Y r kc_space X*
shows *kc_space Y*
proof –
obtain *s* **where** *s: continuous_map X Y r continuous_map Y X s* $\wedge x. x \in \text{topspace } Y \implies r\ (s\ x) = x$
using *assms* **by** (*force simp: retraction_map_def retraction_maps_def*)
then **have** *inj: inj_on s (topspace Y)*
by (*metis inj_on_def*)
show *?thesis*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *S: compactin Y S*
have $S = \{x \in \text{topspace } Y. s\ x \in s\ 'S\}$
using *S compactin_subset_topspace inj_onD [OF inj]* **by** *blast*
with *assms S* **show** *closedin Y S*

```

  by (meson compactin_imp_closedin_gen inj_kc_space_continuous_injective_map_preimage
s(2))
  qed
qed

```

lemma *homeomorphic_kc_space:*

X *homeomorphic_space* $Y \implies kc_space\ X \longleftrightarrow kc_space\ Y$

```

  by (meson homeomorphic_eq_everything_map homeomorphic_space homeomor-
phic_space_sym kc_space_continuous_injective_map_preimage)

```

lemma *compact_kc_eq_maximal_compact_space:*

assumes *compact_space* X

shows $kc_space\ X \longleftrightarrow$

$(\forall Y. topspace\ Y = topspace\ X \wedge (\forall S. openin\ X\ S \longrightarrow openin\ Y\ S) \wedge$
compact_space $Y \longrightarrow Y = X)$ (**is** *?lhs=?rhs*)

proof

assume *?lhs*

then show *?rhs*

```

  by (metis closedin_compact_space compactin_contractive kc_space_def topol-
ogy_eq_topology_finer_closedin)

```

next

assume $R: ?rhs$

show *?lhs*

unfolding *kc_space_def*

proof (*intro strip*)

fix S

assume $S: compactin\ X\ S$

define Y **where**

$Y \equiv topology\ (arbitrary_union_of\ (finite_intersection_of\ (\lambda A. A = topspace$
 $X - S \vee openin\ X\ A)$
relative_to (*topspace* X)))

have $topspace\ Y = topspace\ X$

by (*auto simp: Y_def*)

have $openin\ X\ T \longrightarrow openin\ Y\ T$ **for** T

```

  by (simp add: Y_def arbitrary_union_of_inc finite_intersection_of_inc
openin_subbase openin_subset relative_to_subset_inc)

```

have *compact_space* Y

proof (*rule Alexander_subbase_alt*)

show $\exists \mathcal{F}'. finite\ \mathcal{F}' \wedge \mathcal{F}' \subseteq \mathcal{C} \wedge topspace\ X \subseteq \bigcup \mathcal{F}'$

if $\mathcal{C}: \mathcal{C} \subseteq insert\ (topspace\ X - S)\ (Collect\ (openin\ X))$ **and** *sub: topspace*
 $X \subseteq \bigcup \mathcal{C}$ **for** \mathcal{C}

proof -

consider $\mathcal{C} \subseteq Collect\ (openin\ X) \mid \mathcal{V}$ **where** $\mathcal{C} = insert\ (topspace\ X - S)$
 $\mathcal{V} \subseteq Collect\ (openin\ X)$

using \mathcal{C} **by** (*metis insert_Diff_subset_insert_iff*)

then show *?thesis*

proof *cases*

case 1

then show *?thesis*

```

    by (metis assms compact_space_alt mem_Collect_eq subsetD that(2))
  next
  case 2
  then have  $S \subseteq \bigcup \mathcal{V}$ 
    using  $S$  sub_compactin_subset_topspace by blast
  with 2 obtain  $\mathcal{F}$  where finite  $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{V} \wedge S \subseteq \bigcup \mathcal{F}$ 
    using  $S$  unfolding compactin_def by (metis Ball_Collect)
  with 2 show ?thesis
    by (rule_tac x=insert (topspace X - S)  $\mathcal{F}$  in exI) blast
  qed
  qed
  qed (auto simp:  $Y$ _def)
  have  $Y = X$ 
    using  $R \langle \bigwedge S. \text{openin } X S \longrightarrow \text{openin } Y S \rangle \langle \text{compact\_space } Y \rangle \langle \text{topspace } Y = \text{topspace } X \rangle$  by blast
  moreover have openin  $Y$  (topspace  $X - S$ )
    by (simp add:  $Y$ _def arbitrary_union_of_inc finite_intersection_of_inc openin_subbase_relative_to_subset_inc)
  ultimately show closedin  $X S$ 
    unfolding closedin_def using  $S$  compactin_subset_topspace by blast
  qed
  qed

```

lemma continuous_imp_closed_map_gen:

```

[[compact_space  $X$ ; kc_space  $Y$ ; continuous_map  $X Y f$ ]]  $\implies$  closed_map  $X Y f$ 
by (meson closed_map_def closedin_compact_space compactin_imp_closedin_gen image_compactin)

```

lemma kc_space_compact_subtopologies:

```

kc_space  $X \longleftrightarrow (\forall K. \text{compactin } X K \longrightarrow \text{kc\_space}(\text{subtopology } X K))$  (is ?lhs=?rhs)

```

proof

```

  assume ?lhs

```

```

  then show ?rhs

```

```

    by (auto simp: kc_space_def closedin_closed_subtopology compactin_subtopology)
  next

```

```

  assume  $R$ : ?rhs

```

```

  show ?lhs

```

```

    unfolding kc_space_def

```

```

  proof (intro strip)

```

```

    fix  $K$ 

```

```

    assume  $K$ : compactin  $X K$ 

```

```

    then have  $K \subseteq \text{topspace } X$ 

```

```

      by (simp add: compactin_subset_topspace)

```

```

    moreover have  $X$  closure_of  $K \subseteq K$ 

```

```

  proof

```

```

    fix  $x$ 

```

```

    assume  $x$ :  $x \in X$  closure_of  $K$ 

```



```

have kc_space (subtopology X K)
  by (simp add: R <compactin X K>)
have compactin X (insert x K)
  by (metis K x compactin_Un compactin_sing in_closure_of insert_is_Un)
then show  $x \in K$ 
by (metis R x K Int_insert_left_if1 closedin_Int_closure_of compact_imp_compactin_subtopology

      insertCI kc_space_def subset_insertI)
qed
ultimately show closedin X K
  using closure_of_subset_eq by blast
qed
qed

lemma kc_space_compact_prod_topology:
  assumes compact_space X
  shows  $kc\_space(prod\_topology\ X\ X) \longleftrightarrow Hausdorff\_space\ X$  (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
    unfolding closed_map_diag_eq [symmetric]
    proof (intro continuous_imp_closed_map_gen)
      show continuous_map X (prod_topology X X) ( $\lambda x. (x, x)$ )
        by (intro continuous_intros)
    qed (use L assms in auto)
next
  assume ?rhs then show ?lhs
    by (simp add: Hausdorff_imp_kc_space Hausdorff_space_prod_topology)
qed

lemma kc_space_prod_topology:
   $kc\_space(prod\_topology\ X\ X) \longleftrightarrow (\forall K. compactin\ X\ K \longrightarrow Hausdorff\_space(subtopology\ X\ K))$  (is ?lhs=?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (metis compactin_subspace kc_space_compact_prod_topology kc_space_subtopology
        subtopology_Times)
next
  assume R: ?rhs
  have kc_space (subtopology (prod_topology X X) L) if compactin (prod_topology X X) L for L
  proof –
    define K where  $K \equiv fst\ 'L \cup snd\ 'L$ 
    have  $L \subseteq K \times K$ 
      by (force simp: K_def)
    have compactin X K
      by (metis K_def compactin_Un continuous_map_fst continuous_map_snd
          image_compactin that)
  qed

```

```

then have Hausdorff_space (subtopology X K)
  by (simp add: R)
then have kc_space (prod_topology (subtopology X K) (subtopology X K))
  by (simp add: ⟨compactin X K⟩ compact_space_subtopology kc_space_compact_prod_topology)
then have kc_space (subtopology (prod_topology (subtopology X K) (subtopology
X K)) L)
  using kc_space_subtopology by blast
then show ?thesis
  using ⟨L ⊆ K × K⟩ subtopology_Times subtopology_subtopology
  by (metis (no_types, lifting) Sigma_cong inf.absorb_iff2)
qed
then show ?lhs
  using kc_space_compact_subtopologies by blast
qed

```

```

lemma kc_space_prod_topology_alt:
  kc_space(prod_topology X X) ↔
    kc_space X ∧
    (∀ K. compactin X K → Hausdorff_space(subtopology X K))
using Hausdorff_imp_kc_space kc_space_compact_subtopologies kc_space_prod_topology
by blast

```

```

proposition kc_space_prod_topology_left:
assumes X: kc_space X and Y: Hausdorff_space Y
shows kc_space (prod_topology X Y)
unfolding kc_space_def
proof (intro strip)
  fix K
  assume K: compactin (prod_topology X Y) K
  then have K ⊆ topspace X × topspace Y
    using compactin_subset_topspace topspace_prod_topology by blast
  moreover have ∃ T. openin (prod_topology X Y) T ∧ (a,b) ∈ T ∧ T ⊆ (topspace
X × topspace Y) − K
    if ab: (a,b) ∈ (topspace X × topspace Y) − K for a b
  proof −
    have compactin Y {b}
      using that by force
    moreover
    have compactin Y {y ∈ topspace Y. (a,y) ∈ K}
    proof −
      have compactin (prod_topology X Y) (K ∩ {a} × topspace Y)
        using that compact_Int_closedin [OF K]
        by (simp add: X closedin_prod_Times_iff compactin_imp_closedin_gen)
      moreover have subtopology (prod_topology X Y) (K ∩ {a} × topspace Y)
homeomorphic_space
        subtopology Y {y ∈ topspace Y. (a, y) ∈ K}
      unfolding homeomorphic_space_def homeomorphic_maps_def
      using that
      apply (rule_tac x=snd in exI)

```

```

    apply (rule_tac x=Pair a in exI)
  by (force simp: continuous_map_in_subtopology continuous_map_from_subtopology
      continuous_map_subtopology_snd continuous_map_paired)
  ultimately show ?thesis
  by (simp add: compactin_subspace homeomorphic_compact_space)
qed
moreover have disjnt {b} {y ∈ topspace Y. (a,y) ∈ K}
  using ab by force
ultimately obtain V U where VU: openin Y V openin Y U {b} ⊆ V {y ∈
topspace Y. (a,y) ∈ K} ⊆ U disjnt V U
  using Hausdorff_space_compact_separation [OF Y] by blast
define V' where V' ≡ topspace Y - U
have W: closedin Y V' {y ∈ topspace Y. (a,y) ∈ K} ⊆ topspace Y - V' disjnt
V (topspace Y - V')
  using VU by (auto simp: V'_def disjnt_iff)
with VU obtain V ⊆ topspace Y V' ⊆ topspace Y
  by (meson closedin_subset openin_closedin_eq)
then obtain b ∈ V disjnt {y ∈ topspace Y. (a,y) ∈ K} V' V ⊆ V'
  using VU unfolding disjnt_iff V'_def by force
define C where C ≡ image fst (K ∩ {z ∈ topspace(prod_topology X Y). snd
z ∈ V'})
  have closedin (prod_topology X Y) {z ∈ topspace (prod_topology X Y). snd z
∈ V'}
  using closedin_continuous_map_preimage ⟨closedin Y V'⟩ continuous_map_snd
by blast
  then have compactin X C
  unfolding C_def by (meson K compact_Int_closedin continuous_map_fst
image_compactin)
  then have closedin X C
  using assms by (auto simp: kc_space_def)
show ?thesis
proof (intro exI conjI)
show openin (prod_topology X Y) ((topspace X - C) × V)
  by (simp add: VU ⟨closedin X C⟩ openin_diff openin_prod_Times_iff)
have a ∉ C
  using VU by (auto simp: C_def V'_def)
then show (a, b) ∈ (topspace X - C) × V
  using ⟨a ∉ C⟩ ⟨b ∈ V⟩ that by blast
show (topspace X - C) × V ⊆ topspace X × topspace Y - K
  using ⟨V ⊆ V'⟩ ⟨V ⊆ topspace Y⟩
  apply (simp add: C_def)
  by (smt (verit, ccfv_threshold) DiffE DiffI IntI SigmaE SigmaI image_eqI
mem_Collect_eq prod.sel(1) snd_conv subset_iff)
qed
qed
ultimately show closedin (prod_topology X Y) K
  by (metis surj_pair closedin_def openin_subopen topspace_prod_topology)
qed

```

```

lemma kc_space_prod_topology_right:
  [[Hausdorff_space X; kc_space Y]]  $\implies$  kc_space (prod_topology X Y)
  using kc_space_prod_topology_left homeomorphic_kc_space homeomorphic_space_prod_topology subspace
  by blast

```

6.6.11 Technical results about proper maps, perfect maps, etc

```

lemma compact_imp_proper_map_gen:
  assumes  $Y: \bigwedge S. [S \subseteq \text{topspace } Y; \bigwedge K. \text{compactin } Y K \implies \text{compactin } Y (S \cap K)]$ 
     $\implies \text{closedin } Y S$ 
  and fm:  $f' (\text{topspace } X) \subseteq \text{topspace } Y$ 
  and f:  $\text{continuous\_map } X Y f \vee \text{kc\_space } X$ 
  and  $YX: \bigwedge K. \text{compactin } Y K \implies \text{compactin } X \{x \in \text{topspace } X. f x \in K\}$ 
  shows proper_map  $X Y f$ 
  unfolding proper_map_alt closed_map_def
  proof (intro conjI strip)
    fix C
    assume C:  $\text{closedin } X C$ 
    show  $\text{closedin } Y (f' C)$ 
    proof (intro Y)
      show  $f' C \subseteq \text{topspace } Y$ 
      using C closedin_subset_fm by blast
    fix K
    assume K:  $\text{compactin } Y K$ 
    define A where  $A \equiv \{x \in \text{topspace } X. f x \in K\}$ 
    have  $\text{eq}: f' C \cap K = f' (\{x \in \text{topspace } X. f x \in K\} \cap C)$ 
      using C closedin_subset by auto
    show  $\text{compactin } Y (f' C \cap K)$ 
      unfolding eq
    proof (rule image_compactin)
      show  $\text{compactin } (\text{subtopology } X A) (\{x \in \text{topspace } X. f x \in K\} \cap C)$ 
      proof (rule closedin_compact_space)
        show  $\text{compact\_space } (\text{subtopology } X A)$ 
          by (simp add: A_def K YX compact_space_subtopology)
        show  $\text{closedin } (\text{subtopology } X A) (\{x \in \text{topspace } X. f x \in K\} \cap C)$ 
          using A_def C closedin_subtopology by blast
      qed
    have  $\text{continuous\_map } (\text{subtopology } X A) (\text{subtopology } Y K) f$  if kc_space X
      unfolding continuous_map_closedin
    proof (intro conjI strip)
      show  $f \in \text{topspace } (\text{subtopology } X A) \rightarrow \text{topspace } (\text{subtopology } Y K)$ 
      using A_def K compactin_subset_topospace by fastforce
    next
      fix C
      assume C:  $\text{closedin } (\text{subtopology } Y K) C$ 
      show  $\text{closedin } (\text{subtopology } X A) \{x \in \text{topspace } (\text{subtopology } X A). f x \in C\}$ 
      proof (rule compactin_imp_closedin_gen)

```

```

show kc_space (subtopology X A)
  by (simp add: kc_space_subtopology that)
have [simp]:  $\{x \in \text{topspace } X. f x \in K \wedge f x \in C\} = \{x \in \text{topspace } X. f$ 
 $x \in C\}$ 
  using C closedin_imp_subset by auto
have compactin (subtopology Y K) C
  by (simp add: C K closedin_compact_space compact_space_subtopology)
then have compactin X  $\{x \in \text{topspace } X. x \in A \wedge f x \in C\}$ 
  by (auto simp: A_def compactin_subtopology dest: YX)
then show compactin (subtopology X A)  $\{x \in \text{topspace } (\text{subtopology } X A).$ 
 $f x \in C\}$ 
  by (auto simp add: compactin_subtopology)
qed
qed
with f show continuous_map (subtopology X A) Y f
  using continuous_map_from_subtopology continuous_map_in_subtopology
by blast
qed
qed
qed (simp add: YX)

```

lemma *tube_lemma_left*:

```

assumes W: openin (prod_topology X Y) W and C: compactin X C
  and y: y \in topspace Y and subW: C \times \{y\} \subseteq W
shows  $\exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge C \subseteq U \wedge y \in V \wedge U \times V \subseteq W$ 
proof (cases C = \{\})
  case True
    with y show ?thesis by auto
  next
    case False
have  $\exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge x \in U \wedge y \in V \wedge U \times V \subseteq W$ 
  if x \in C for x
    using W openin_prod_topology_alt subW subsetD that by fastforce
then obtain U V where UV: \bigwedge x. x \in C \implies openin X (U x) \wedge openin Y (V
 $x) \wedge x \in U x \wedge y \in V x \wedge U x \times V x \subseteq W$ 
    by metis
then obtain D where D: finite D D \subseteq C C \subseteq \bigcup (U ` D)
    using compactinD [OF C, of U`C]
    by (smt (verit) UN_I finite_subset_image imageE subsetI)
show ?thesis
proof (intro exI conjI)
  show openin X  $(\bigcup (U ` D))$  openin Y  $(\bigcap (V ` D))$ 
    using D False UV by blast+
  show  $y \in \bigcap (V ` D) C \subseteq \bigcup (U ` D) \cup (U ` D) \times \bigcap (V ` D) \subseteq W$ 
    using D UV by force+
qed
qed

```

lemma *Wallace_theorem_prod_topology*:

```

assumes compactin X K compactin Y L
and W: openin (prod_topology X Y) W and subW: K × L ⊆ W
obtains U V where openin X U openin Y V K ⊆ U L ⊆ V U × V ⊆ W
proof –
have  $\bigwedge y. y \in L \implies \exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge K \subseteq U \wedge y \in V \wedge U$ 
 $\times V \subseteq W$ 
proof (intro tube_lemma_left assms)
fix y assume y ∈ L
show y ∈ topspace Y
using assms ⟨y ∈ L⟩ compactin_subset_topspace by blast
show K × {y} ⊆ W
using ⟨y ∈ L⟩ subW by force
qed
then obtain U V where UV:
 $\bigwedge y. y \in L \implies \text{openin } X (U y) \wedge \text{openin } Y (V y) \wedge K \subseteq U y \wedge y \in V y$ 
 $\wedge U y \times V y \subseteq W$ 
by metis
then obtain M where finite M M ⊆ L and M: L ⊆  $\bigcup (V' M)$ 
using ⟨compactin Y L⟩ unfolding compactin_def
by (smt (verit) UN_iff finite_subset_image imageE subset_iff)
show thesis
proof (cases M={})
case True
with M have L={ }
by blast
then show ?thesis
using ⟨compactin X K⟩ compactin_subset_topspace that by fastforce
next
case False
show ?thesis
proof
show openin X ( $\bigcap (U' M)$ )
using False UV ⟨M ⊆ L⟩ ⟨finite M⟩ by blast
show openin Y ( $\bigcup (V' M)$ )
using UV ⟨M ⊆ L⟩ by blast
show K ⊆  $\bigcap (U' M)$ 
by (meson INF_greatest UV ⟨M ⊆ L⟩ subsetD)
show L ⊆  $\bigcup (V' M)$ 
by (simp add: M)
show  $\bigcap (U' M) \times \bigcup (V' M) \subseteq W$ 
using UV ⟨M ⊆ L⟩ by fastforce
qed
qed
qed

```

lemma proper_map_prod:

```

proper_map (prod_topology X Y) (prod_topology X' Y') (λ(x,y). (f x, g y))  $\longleftrightarrow$ 
(prod_topology X Y) = trivial_topology  $\vee$  proper_map X X' f  $\wedge$  proper_map
Y Y' g

```

```

  (is ?lhs  $\longleftrightarrow$  _  $\vee$  ?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True then show ?thesis by auto
next
  case False
  then have ne: topspace X  $\neq$  {} topspace Y  $\neq$  {}
    by auto
  define h where h  $\equiv$   $\lambda(x,y). (f x, g y)$ 
  have proper_map X X' f proper_map Y Y' g if ?lhs
  proof -
  have cm: closed_map X X' f closed_map Y Y' g
    using that False closed_map_prod proper_imp_closed_map by blast+
  show proper_map X X' f
  proof (clarsimp simp add: proper_map_def cm)
  fix y
  assume y: y  $\in$  topspace X'
  obtain z where z: z  $\in$  topspace Y
    using ne by blast
  then have eq: {x  $\in$  topspace X. f x = y} =
    fst ' {u  $\in$  topspace X  $\times$  topspace Y. h u = (y,g z)}
  by (force simp: h_def)
  show compactin X {x  $\in$  topspace X. f x = y}
  unfolding eq
  proof (intro image_compactin)
  have g z  $\in$  topspace Y'
    by (meson closed_map_def closedin_subset closedin_topspace cm image_subset_iff z)
  with y show compactin (prod_topology X Y) {u  $\in$  topspace X  $\times$  topspace Y. (h u) = (y, g z)}
    using that by (simp add: h_def proper_map_def)
  show continuous_map (prod_topology X Y) X fst
    by (simp add: continuous_map_fst)
  qed
  qed
  show proper_map Y Y' g
  proof (clarsimp simp add: proper_map_def cm)
  fix y
  assume y: y  $\in$  topspace Y'
  obtain z where z: z  $\in$  topspace X
    using ne by blast
  then have eq: {x  $\in$  topspace Y. g x = y} =
    snd ' {u  $\in$  topspace X  $\times$  topspace Y. h u = (f z,y)}
  by (force simp: h_def)
  show compactin Y {x  $\in$  topspace Y. g x = y}
  unfolding eq
  proof (intro image_compactin)
  have f z  $\in$  topspace X'
    by (meson closed_map_def closedin_subset closedin_topspace cm image_subset_iff z)

```

```

with  $y$  show compactin (prod_topology X Y) { $u \in \text{topspace } X \times \text{topspace } Y$ . ( $h\ u = (f\ z, y)$ )}
using that by (simp add: proper_map_def h_def)
show continuous_map (prod_topology X Y) Y snd
by (simp add: continuous_map_snd)
qed
qed
moreover
{ assume  $R$ : ?rhs
then have fgim:  $f \in \text{topspace } X \rightarrow \text{topspace } X' \ g \in \text{topspace } Y \rightarrow \text{topspace } Y'$ 
and cm: closed_map X X'  $f$  closed_map Y Y'  $g$ 
by (auto simp: proper_map_def closed_map_imp_subset_topspace)
have closed_map (prod_topology X Y) (prod_topology X' Y')  $h$ 
unfolding closed_map_fibre_neighbourhood_imp_conjL
proof (intro conjI strip)
show  $h \in \text{topspace } (prod\_topology\ X\ Y) \rightarrow \text{topspace } (prod\_topology\ X'\ Y')$ 
unfolding  $h\_def$  using fgim by auto
fix  $W\ w$ 
assume  $W$ : openin (prod_topology X Y)  $W$ 
and  $w$ :  $w \in \text{topspace } (prod\_topology\ X'\ Y')$ 
and subW: { $x \in \text{topspace } (prod\_topology\ X\ Y)$ .  $h\ x = w$ }  $\subseteq W$ 
then obtain  $x'\ y'$  where weq:  $w = (x',y')$   $x' \in \text{topspace } X' \ y' \in \text{topspace } Y'$ 
by auto
have eq: { $u \in \text{topspace } X \times \text{topspace } Y$ .  $h\ u = (x',y')$ } = { $x \in \text{topspace } X$ .  $f\ x = x'$ }  $\times$  { $y \in \text{topspace } Y$ .  $g\ y = y'$ }
by (auto simp:  $h\_def$ )
obtain  $U\ V$  where openin X  $U$  openin Y  $V$   $U \times V \subseteq W$ 
and  $U$ : { $x \in \text{topspace } X$ .  $f\ x = x'$ }  $\subseteq U$ 
and  $V$ : { $x \in \text{topspace } Y$ .  $g\ x = y'$ }  $\subseteq V$ 
proof (rule Wallace_theorem_prod_topology)
show compactin X { $x \in \text{topspace } X$ .  $f\ x = x'$ } compactin Y { $x \in \text{topspace } Y$ .  $g\ x = y'$ }
using  $R$  weq unfolding proper_map_def closed_map_fibre_neighbourhood
by fastforce+
show { $x \in \text{topspace } X$ .  $f\ x = x'$ }  $\times$  { $x \in \text{topspace } Y$ .  $g\ x = y'$ }  $\subseteq W$ 
using weq subW by (auto simp:  $h\_def$ )
qed (use  $W$  in auto)
obtain  $U'$  where openin X'  $U'$   $x' \in U'$  and  $U'$ : { $x \in \text{topspace } X$ .  $f\ x \in U'$ }  $\subseteq U$ 
using cm U  $\langle$ openin X  $U$  $\rangle$  weq unfolding closed_map_fibre_neighbourhood
by meson
obtain  $V'$  where openin Y'  $V'$   $y' \in V'$  and  $V'$ : { $x \in \text{topspace } Y$ .  $g\ x \in V'$ }  $\subseteq V$ 
using cm V  $\langle$ openin Y  $V$  $\rangle$  weq unfolding closed_map_fibre_neighbourhood
by meson
show  $\exists V$ . openin (prod_topology X' Y')  $V$   $\wedge w \in V$   $\wedge$  { $x \in \text{topspace } (prod\_topology\ X\ Y)$ .  $h\ x \in V$ }  $\subseteq W$ 

```



```

proof (intro conjI exI)
  show openin (prod_topology X' Y') (U' × V')
    by (simp add: ⟨openin X' U'⟩ ⟨openin Y' V'⟩ openin_prod_Times_iff)
  show  $w \in U' \times V'$ 
    using ⟨ $x' \in U'$ ⟩ ⟨ $y' \in V'$ ⟩ weq by blast
  show  $\{x \in \text{topspace } (\text{prod\_topology } X Y). h\ x \in U' \times V'\} \subseteq W$ 
    using ⟨ $U \times V \subseteq W$ ⟩ U' V' h_def by auto
qed
qed
moreover
have compactin (prod_topology X Y) { $u \in \text{topspace } X \times \text{topspace } Y. h\ u =$ 
( $w, z$ )}
  if  $w \in \text{topspace } X'$  and  $z \in \text{topspace } Y'$  for  $w\ z$ 
proof -
  have eq:  $\{u \in \text{topspace } X \times \text{topspace } Y. h\ u = (w, z)\} =$ 
 $\{u \in \text{topspace } X. f\ u = w\} \times \{y. y \in \text{topspace } Y \wedge g\ y = z\}$ 
    by (auto simp: h_def)
  show ?thesis
    using R that by (simp add: eq_compactin_Times_proper_map_def)
qed
ultimately have ?lhs
  by (auto simp: h_def_proper_map_def)
}
ultimately show ?thesis using False by metis
qed

lemma proper_map_paired:
assumes Hausdorff_space X ∧ proper_map X Y f ∧ proper_map X Z g ∨
Hausdorff_space Y ∧ continuous_map X Y f ∧ proper_map X Z g ∨
Hausdorff_space Z ∧ proper_map X Y f ∧ continuous_map X Z g
shows proper_map X (prod_topology Y Z) (λx. (f x, g x))
using assms
proof (elim disjE conjE)
assume §: Hausdorff_space X proper_map X Y f proper_map X Z g
have eq: (λx. (f x, g x)) = (λ(x, y). (f x, g y)) ∘ (λx. (x, x))
  by auto
show proper_map X (prod_topology Y Z) (λx. (f x, g x))
  unfolding eq
proof (rule proper_map_compose)
show proper_map X (prod_topology X X) (λx. (x, x))
  by (simp add: §)
show proper_map (prod_topology X X) (prod_topology Y Z) (λ(x, y). (f x, g
y))
  by (simp add: § proper_map_prod)
qed
next
assume §: Hausdorff_space Y continuous_map X Y f proper_map X Z g
have eq: (λx. (f x, g x)) = (λ(x, y). (x, g y)) ∘ (λx. (f x, x))
  by auto

```

```

show proper_map X (prod_topology Y Z) ( $\lambda x. (f x, g x)$ )
  unfolding eq
proof (rule proper_map_compose)
  show proper_map X (prod_topology Y X) ( $\lambda x. (f x, x)$ )
    by (simp add: § proper_map_paired_continuous_map_left)
  show proper_map (prod_topology Y X) (prod_topology Y Z) ( $\lambda(x,y). (x, g y)$ )
    by (simp add: § proper_map_prod proper_map_id [unfolded id_def])
qed
next
assume §: Hausdorff_space Z proper_map X Y f continuous_map X Z g
have eq: ( $\lambda x. (f x, g x) = (\lambda(x,y). (f x, y)) \circ (\lambda x. (x, g x))$ )
  by auto
show proper_map X (prod_topology Y Z) ( $\lambda x. (f x, g x)$ )
  unfolding eq
proof (rule proper_map_compose)
  show proper_map X (prod_topology X Z) ( $\lambda x. (x, g x)$ )
    using § proper_map_paired_continuous_map_right by auto
  show proper_map (prod_topology X Z) (prod_topology Y Z) ( $\lambda(x,y). (f x, y)$ )
    by (simp add: § proper_map_prod proper_map_id [unfolded id_def])
qed
qed

lemma proper_map_pairwise:
  assumes
    Hausdorff_space X  $\wedge$  proper_map X Y (fst  $\circ$  f)  $\wedge$  proper_map X Z (snd  $\circ$  f)
   $\vee$ 
    Hausdorff_space Y  $\wedge$  continuous_map X Y (fst  $\circ$  f)  $\wedge$  proper_map X Z (snd
 $\circ$  f)  $\vee$ 
    Hausdorff_space Z  $\wedge$  proper_map X Y (fst  $\circ$  f)  $\wedge$  continuous_map X Z (snd
 $\circ$  f)
  shows proper_map X (prod_topology Y Z) f
  using proper_map_paired [OF assms] by (simp add: o_def)

lemma proper_map_from_composition_right:
  assumes Hausdorff_space Y proper_map X Z (g  $\circ$  f) and contf: continuous_map X Y f
  and contg: continuous_map Y Z g
  shows proper_map X Y f
proof –
  define YZ where YZ  $\equiv$  subtopology (prod_topology Y Z) (( $\lambda x. (x, g x)$ ) ‘
topspace Y)
  have proper_map X Y (fst  $\circ$  ( $\lambda x. (f x, (g \circ f) x)$ ))
  proof (rule proper_map_compose)
  have [simp]:  $x \in \text{topspace } X \implies f x \in \text{topspace } Y$  for x
    using contf continuous_map_preimage_topspace by auto
  show proper_map X YZ ( $\lambda x. (f x, (g \circ f) x)$ )
  unfolding YZ_def
  using assms
  by (force intro!: proper_map_into_subtopology proper_map_paired simp:

```

```

o_def image_iff)+
  show proper_map YZ Y fst
    using contg
  by (simp flip: homeomorphic_maps_graph add: YZ_def homeomorphic_maps_map
homeomorphic_imp_proper_map)
qed
moreover have fst  $\circ$  ( $\lambda x. (f x, (g \circ f) x)$ ) = f
  by auto
ultimately show ?thesis
  by auto
qed

```

```

lemma perfect_map_from_composition_right:
  [[Hausdorff_space Y; perfect_map X Z (g  $\circ$  f);
  continuous_map X Y f; continuous_map Y Z g; f ' topspace X = topspace Y]]
   $\implies$  perfect_map X Y f
  by (meson perfect_map_def proper_map_from_composition_right)

```

```

lemma perfect_map_from_composition_right_inj:
  [[perfect_map X Z (g  $\circ$  f); f ' topspace X = topspace Y;
  continuous_map X Y f; continuous_map Y Z g; inj_on g (topspace Y)]]
   $\implies$  perfect_map X Y f
  by (meson continuous_map_openin_preimage_eq perfect_map_def proper_map_from_composition_right_inj)

```

6.6.12 Regular spaces

Regular spaces are *not* a priori assumed to be Hausdorff or T_1

```

definition regular_space
  where regular_space X  $\equiv$ 
     $\forall C a. \text{closedin } X C \wedge a \in \text{topspace } X - C$ 
       $\longrightarrow$  ( $\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge a \in U \wedge C \subseteq V \wedge \text{disjnt } U V$ )

```

```

lemma homeomorphic_regular_space_aux:
  assumes hom: X homeomorphic_space Y and X: regular_space X
  shows regular_space Y
proof -
  obtain f g where hmf: homeomorphic_map X Y f and hmg: homeomorphic_map
Y X g
  and fg: ( $\forall x \in \text{topspace } X. g(f x) = x$ )  $\wedge$  ( $\forall y \in \text{topspace } Y. f(g y) = y$ )
  using assms X homeomorphic_maps_map homeomorphic_space_def by fast-
force
  show ?thesis
  unfolding regular_space_def
proof clarify
  fix C a
  assume Y: closedin Y C a  $\in$  topspace Y and a  $\notin$  C
  then obtain closedin X (g ' C) g a  $\in$  topspace X g a  $\notin$  g ' C
  using <closedin Y C> hmg homeomorphic_map_closedness_eq

```

by (*smt* (*verit*, *ccfv_SIG*) *fg* *homeomorphic_imp_surjective_map_image_iff* *in_mono*)
then obtain $S T$ **where** ST : *openin* $X S$ *openin* $X T$ $g a \in S$ $g'C \subseteq T$ *disjnt* $S T$
using X **unfolding** *regular_space_def* **by** (*metis* *DiffI*)
then have *openin* $Y (f'S)$ *openin* $Y (f'T)$
by (*meson* *hmf* *homeomorphic_map_openness_eq*) +
moreover have $a \in f'S \wedge C \subseteq f'T$
using ST **by** (*smt* (*verit*, *best*) Y *closedin_subset* *fg* *image_eqI* *subset_iff*)
moreover have *disjnt* $(f'S)$ $(f'T)$
using ST **by** (*smt* (*verit*, *ccfv_SIG*) *disjnt_iff* *fg* *image_iff* *openin_subset* *subsetD*)
ultimately show $\exists U V. \text{openin } Y U \wedge \text{openin } Y V \wedge a \in U \wedge C \subseteq V \wedge$
disjnt $U V$
by *metis*
qed
qed

lemma *homeomorphic_regular_space*:

X *homeomorphic_space* Y

\implies (*regular_space* $X \iff$ *regular_space* Y)

by (*meson* *homeomorphic_regular_space_aux* *homeomorphic_space_sym*)

lemma *regular_space*:

regular_space $X \iff$

$(\forall C a. \text{closedin } X C \wedge a \in \text{topspace } X - C$

$\longrightarrow (\exists U. \text{openin } X U \wedge a \in U \wedge \text{disjnt } C (\text{X closure_of } U)))$)

unfolding *regular_space_def*

proof (*intro* *all_cong1* *imp_cong* *refl* *ex_cong1*)

fix $C a U$

assume C : *closedin* $X C \wedge a \in \text{topspace } X - C$

show $(\exists V. \text{openin } X U \wedge \text{openin } X V \wedge a \in U \wedge C \subseteq V \wedge \text{disjnt } U V)$

$\iff (\text{openin } X U \wedge a \in U \wedge \text{disjnt } C (\text{X closure_of } U))$ (**is** *?lhs=?rhs*)

proof

assume *?lhs*

then show *?rhs*

by (*smt* (*verit*, *best*) *disjnt_iff* *in_closure_of* *subsetD*)

next

assume R : *?rhs*

then have *disjnt* U (*topspace* $X - X$ *closure_of* U)

by (*meson* *DiffD2* *closure_of_subset* *disjnt_iff* *openin_subset* *subsetD*)

moreover have $C \subseteq \text{topspace } X - X$ *closure_of* U

by (*meson* C *DiffI* R *closedin_subset* *disjnt_iff* *subset_eq*)

ultimately show *?lhs*

using R **by** (*rule_tac* $x = \text{topspace } X - X$ *closure_of* U **in** *exI*) *auto*

qed

qed

lemma *neighbourhood_base_of_closedin*:

```

  neighbourhood_base_of (closedin X) X  $\longleftrightarrow$  regular_space X (is ?lhs=?rhs)
proof -
  have ?lhs  $\longleftrightarrow$  ( $\forall W x$ . openin X W  $\wedge$   $x \in W \longrightarrow$ 
    ( $\exists U$ . openin X U  $\wedge$  ( $\exists V$ . closedin X V  $\wedge$   $x \in U \wedge U \subseteq V \wedge V \subseteq$ 
    W)))
    by (simp add: neighbourhood_base_of)
  also have ...  $\longleftrightarrow$  ( $\forall W x$ . closedin X W  $\wedge$   $x \in \text{topspace } X - W \longrightarrow$ 
    ( $\exists U$ . openin X U  $\wedge$  ( $\exists V$ . closedin X V  $\wedge$   $x \in U \wedge U \subseteq V \wedge V$ 
     $\subseteq \text{topspace } X - W$ )))
    by (smt (verit) Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq)
  also have ...  $\longleftrightarrow$  ?rhs
  proof -
  have  $\S$ : ( $\exists V$ . closedin X V  $\wedge$   $x \in U \wedge U \subseteq V \wedge V \subseteq \text{topspace } X - W$ )
     $\longleftrightarrow$  ( $\exists V$ . openin X V  $\wedge$   $x \in U \wedge W \subseteq V \wedge \text{disjnt } U V$ ) (is ?lhs=?rhs)
    if openin X U closedin X W  $x \in \text{topspace } X$   $x \notin W$  for W U x
  proof
    assume ?lhs with  $\langle$ closedin X W $\rangle$  show ?rhs
    unfolding closedin_def by (smt (verit) Diff_mono disjnt_Diff1 double_diff
    subset_eq)
  next
    assume ?rhs with  $\langle$ openin X U $\rangle$  show ?lhs
    unfolding openin_closedin_eq disjnt_def
    by (smt (verit) Diff_Diff_Int Diff_disjoint Diff_eq_empty_iff Int_Diff
    inf.orderE)
  qed
  show ?thesis
    unfolding regular_space_def
    by (intro all_cong1 ex_cong1 imp_cong refl) (metis  $\S$  DiffE)
  qed
finally show ?thesis .
qed

```

```

lemma regular_space_discrete_topology [simp]:
  regular_space (discrete_topology S)
using neighbourhood_base_of_closedin neighbourhood_base_of_discrete_topology
by fastforce

```

```

lemma regular_space_subtopology:
  regular_space X  $\implies$  regular_space (subtopology X S)
  unfolding regular_space_def openin_subtopology_alt closedin_subtopology_alt
  disjnt_iff
  by clarsimp (smt (verit, best) inf.orderE inf_le1 le_inf_iff)

```

```

lemma regular_space_retraction_map_image:
   $\llbracket$ retraction_map X Y r; regular_space X $\rrbracket \implies$  regular_space Y
  using hereditary_imp_retractive_property homeomorphnic_regular_space regular
  space_subtopology by blast

```

lemma *regular_t0_imp_Hausdorff_space*:

$\llbracket \text{regular_space } X; \text{t0_space } X \rrbracket \implies \text{Hausdorff_space } X$

apply (*clarsimp simp: regular_space_def t0_space Hausdorff_space_def*)

by (*metis disjnt_sym subsetD*)

lemma *regular_t0_eq_Hausdorff_space*:

$\text{regular_space } X \implies (\text{t0_space } X \longleftrightarrow \text{Hausdorff_space } X)$

using *Hausdorff_imp_t0_space regular_t0_imp_Hausdorff_space* **by** *blast*

lemma *regular_t1_imp_Hausdorff_space*:

$\llbracket \text{regular_space } X; \text{t1_space } X \rrbracket \implies \text{Hausdorff_space } X$

by (*simp add: regular_t0_imp_Hausdorff_space t1_imp_t0_space*)

lemma *regular_t1_eq_Hausdorff_space*:

$\text{regular_space } X \implies \text{t1_space } X \longleftrightarrow \text{Hausdorff_space } X$

using *regular_t0_imp_Hausdorff_space t1_imp_t0_space t1_or_Hausdorff_space*
by *blast*

lemma *compact_Hausdorff_imp_regular_space*:

assumes *compact_space X Hausdorff_space X*

shows *regular_space X*

unfolding *regular_space_def*

proof *clarify*

fix *S a*

assume *closedin X S* **and** *a ∈ topspace X* **and** *a ∉ S*

then show $\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge a \in U \wedge S \subseteq V \wedge \text{disjnt } U V$

using *assms unfolding Hausdorff_space_compact_sets*

by (*metis closedin_compact_space compactin_sing disjnt_empty1 insert_subset disjnt_insert1*)

qed

lemma *neighbourhood_base_of_closed_Hausdorff_space*:

$\text{regular_space } X \wedge \text{Hausdorff_space } X \longleftrightarrow$

$\text{neighbourhood_base_of } (\lambda C. \text{closedin } X C \wedge \text{Hausdorff_space}(\text{subtopology } X C)) X$ **(is ?lhs=?rhs)**

proof

assume *?lhs* **then show** *?rhs*

by (*simp add: Hausdorff_space_subtopology neighbourhood_base_of_closedin*)

next

assume *?rhs* **then show** *?lhs*

by (*metis (mono_tags, lifting) Hausdorff_space_closed_neighbourhood neighbourhood_base_of_neighbourhood_base_of_closedin openin_topspace*)

qed

lemma *locally_compact_imp_kc_eq_Hausdorff_space*:

$\text{neighbourhood_base_of } (\text{compactin } X) X \implies \text{kc_space } X \longleftrightarrow \text{Hausdorff_space } X$

by (*metis Hausdorff_imp_kc_space kc_imp_t1_space kc_space_def neighbourhood_base_of_closedin neighbourhood_base_of_mono regular_t1_imp_Hausdorff_space*)

```

lemma regular_space_compact_closed_separation:
  assumes  $X$ : regular_space  $X$ 
    and  $S$ : compactin  $X$   $S$ 
    and  $T$ : closedin  $X$   $T$ 
    and disjnt  $S$   $T$ 
  shows  $\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U V$ 
proof (cases  $S = \{\}$ )
  case True
    then show ?thesis
    by (meson T closedin_def disjnt_empty1 empty_subsetI openin_empty openin_topspace)
  next
    case False
    then have  $\exists U V. x \in S \longrightarrow \text{openin } X U \wedge \text{openin } X V \wedge x \in U \wedge T \subseteq V \wedge$ 
disjnt  $U V$  for  $x$ 
      using assms unfolding regular_space_def
      by (smt (verit) Diff_iff compactin_subset_topspace disjnt_iff subsetD)
    then obtain  $U V$  where  $UV: \bigwedge x. x \in S \implies \text{openin } X (U x) \wedge \text{openin } X (V x) \wedge x \in (U x) \wedge T \subseteq (V x) \wedge \text{disjnt } (U x) (V x)$ 
      by metis
    then obtain  $\mathcal{F}$  where finite  $\mathcal{F}$   $\mathcal{F} \subseteq U' S S \subseteq \bigcup \mathcal{F}$ 
      using  $S$  unfolding compactin_def by (smt (verit) UN_iff image_iff subsetI)
    then obtain  $K$  where finite  $K$   $K \subseteq S$  and  $K: S \subseteq \bigcup (U' K)$ 
      by (metis finite_subset_image)
    show ?thesis
    proof (intro exI conjI)
      show openin  $X$   $(\bigcup (U' K))$ 
        using  $\langle K \subseteq S \rangle UV$  by blast
      show openin  $X$   $(\bigcap (V' K))$ 
        using False  $K UV \langle K \subseteq S \rangle \langle \text{finite } K \rangle$  by blast
      show  $S \subseteq \bigcup (U' K)$ 
        by (simp add:  $K$ )
      show  $T \subseteq \bigcap (V' K)$ 
        using  $UV \langle K \subseteq S \rangle$  by blast
      show disjnt  $(\bigcup (U' K)) (\bigcap (V' K))$ 
        by (smt (verit) Inter_iff UN_E  $UV \langle K \subseteq S \rangle$  disjnt_iff image_eqI subset_iff)
    qed
  qed

```

```

lemma regular_space_compact_closed_sets:
  regular_space  $X \longleftrightarrow$ 
     $(\forall S T. \text{compactin } X S \wedge \text{closedin } X T \wedge \text{disjnt } S T$ 
       $\longrightarrow (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$ 
         $V))$  (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
    using regular_space_compact_closed_separation by fastforce
next
  assume  $R: ?rhs$ 

```

```

show ?lhs
  unfolding regular_space
proof clarify
  fix S x
  assume closedin X S and  $x \in \text{topspace } X$  and  $x \notin S$ 
  then obtain U V where openin X U  $\wedge$  openin X V  $\wedge$   $\{x\} \subseteq U \wedge S \subseteq V \wedge$ 
disjnt U V
  by (metis R compactin_sing disjnt_empty1 disjnt_insert1)
  then show  $\exists U. \text{openin } X U \wedge x \in U \wedge \text{disjnt } S (X \text{ closure\_of } U)$ 
  by (smt (verit, best) disjnt_iff in_closure_of insert_subset subsetD)
qed
qed

```

lemma regular_space_prod_topology:

```

regular_space (prod_topology X Y)  $\longleftrightarrow$ 
  X = trivial_topology  $\vee$  Y = trivial_topology  $\vee$  regular_space X  $\wedge$  regular_
lar_space Y (is ?lhs=?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
  by (metis regular_space_retraction_map_image retraction_map_fst retraction_
tion_map_snd)
next
  assume R: ?rhs
  show ?lhs
  proof (cases X = trivial_topology  $\vee$  Y = trivial_topology)
  case True then show ?thesis by auto
  next
  case False
  then have regular_space X regular_space Y
  using R by auto
  show ?thesis
  unfolding neighbourhood_base_of_closedin [symmetric] neighbourhood_base_of
proof clarify
  fix W x y
  assume W: openin (prod_topology X Y) W and  $(x,y) \in W$ 
  then obtain U V where U: openin X U  $x \in U$  and V: openin Y V  $y \in V$ 
and  $U \times V \subseteq W$ 
  by (metis openin_prod_topology_alt)
  obtain D1 C1 where 1: openin X D1 closedin X C1  $x \in D1$   $D1 \subseteq C1$   $C1 \subseteq$ 
U
  by (metis  $\langle$ regular_space X $\rangle$  U neighbourhood_base_of neighbourhood_base_of_closedin)
  obtain D2 C2 where 2: openin Y D2 closedin Y C2  $y \in D2$   $D2 \subseteq C2$   $C2 \subseteq$ 
V
  by (metis  $\langle$ regular_space Y $\rangle$  V neighbourhood_base_of neighbourhood_base_of_closedin)
  show  $\exists U V. \text{openin } (prod\_topology \ X \ Y) \ U \wedge \text{closedin } (prod\_topology \ X \ Y)$ 
V  $\wedge$ 
   $(x,y) \in U \wedge U \subseteq V \wedge V \subseteq W$ 

```



```

proof (intro conjI exI)
  show openin (prod_topology X Y) (D1 × D2)
    by (simp add: 1 2 openin_prod_Times_iff)
  show closedin (prod_topology X Y) (C1 × C2)
    by (simp add: 1 2 closedin_prod_Times_iff)
qed (use 1 2 ⟨U × V ⊆ W⟩ in auto)
qed
qed
qed

lemma regular_space_product_topology:
  regular_space (product_topology X I) ↔
  (product_topology X I) = trivial_topology ∨ (∀ i ∈ I. regular_space (X i)) (is
  ?lhs=?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (meson regular_space_retraction_map_image retraction_map_product_projection)
next
  assume R: ?rhs
  show ?lhs
  proof (cases product_topology X I = trivial_topology)
    case True
      then show ?thesis
        by auto
    next
      case False
        then obtain x where x: x ∈ topspace (product_topology X I)
          by (meson ex_in_conv_null_topspace_iff_trivial)
        define  $\mathcal{F}$  where  $\mathcal{F} \equiv \{PiE\ I\ U \mid U.\ finite\ \{i \in I.\ U\ i \neq\ topspace\ (X\ i)\} \wedge (\forall i \in I.\ openin\ (X\ i)\ (U\ i))\}$ 
        have oo: openin (product_topology X I) = arbitrary_union_of (λW. W ∈  $\mathcal{F}$ )
          by (simp add:  $\mathcal{F}$ _def openin_product_topology product_topology_base_alt)
        have ∃ U V. openin (product_topology X I) U ∧ closedin (product_topology X I) V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆ W
          if fin: finite {i ∈ I. W i ≠ topspace (X i)}
            and opeW: ∧k. k ∈ I ⇒ openin (X k) (W k) and x: x ∈ PiE I W for W
          x
        proof –
          have ∧i. i ∈ I ⇒ ∃ U V. openin (X i) U ∧ closedin (X i) V ∧ x i ∈ U ∧ U ⊆ V ∧ V ⊆ W i
            by (metis False PiE_iff R neighbourhood_base_of_neighbourhood_base_of_closedin_opeW x)
          then obtain U C where UC:
            ∧i. i ∈ I ⇒ openin (X i) (U i) ∧ closedin (X i) (C i) ∧ x i ∈ U i ∧ U i ⊆ C i ∧ C i ⊆ W i
            by metis
          define PI where PI ≡ λV. PiE I (λi. if W i = topspace(X i) then topspace(X i) else V i)

```

```

show ?thesis
proof (intro conjI exI)
  have  $\forall i \in I. W\ i \neq \text{topspace } (X\ i) \longrightarrow \text{openin } (X\ i)\ (U\ i)$ 
    using UC by force
  with fin show openin (product_topology X I) (PI U)
by (simp add: Collect_mono_iff PI_def openin_PiE_gen rev_finite_subset)
show closedin (product_topology X I) (PI C)
  by (simp add: UC closedin_product_topology PI_def)
show  $x \in PI\ U$ 
  using UC  $x$  by (fastforce simp: PI_def)
show  $PI\ U \subseteq PI\ C$ 
  by (smt (verit) UC Orderings.order_eq_iff PiE_mono PI_def)
show  $PI\ C \subseteq Pi_E\ I\ W$ 
  by (simp add: UC PI_def subset_PiE)
qed
qed
then have neighbourhood_base_of (closedin (product_topology X I)) (product_topology
X I)
  unfolding neighbourhood_base_of_topology_base [OF oo] by (force simp:
F_def)
then show ?thesis
  by (simp add: neighbourhood_base_of_closedin)
qed
qed

lemma closed_map_paired_gen_aux:
  assumes regular_space Y and  $f: \text{closed\_map } Z\ X$  and  $g: \text{closed\_map } Z\ Y$   $g$ 
  and  $\text{clo: } \bigwedge y. y \in \text{topspace } X \implies \text{closedin } Z\ \{x \in \text{topspace } Z. f\ x = y\}$ 
  and contg: continuous_map Z Y  $g$ 
  shows closed_map Z (prod_topology X Y)  $(\lambda x. (f\ x, g\ x))$ 
  unfolding closed_map_def
proof (intro strip)
  fix C assume closedin Z C
  then have  $C \subseteq \text{topspace } Z$ 
  by (simp add: closedin_subset)
  have  $f \in \text{topspace } Z \rightarrow \text{topspace } X$   $g \in \text{topspace } Z \rightarrow \text{topspace } Y$ 
  by (simp_all add: assms closed_map_imp_subset_topospace)
  show closedin (prod_topology X Y)  $((\lambda x. (f\ x, g\ x))\ 'C)$ 
  unfolding closedin_def topspace_prod_topology
proof (intro conjI)
  have closedin Y  $(g\ 'C)$ 
  using  $\langle \text{closedin } Z\ C \rangle$  assms(3) closed_map_def by blast
  with assms show  $(\lambda x. (f\ x, g\ x))\ 'C \subseteq \text{topspace } X \times \text{topspace } Y$ 
  by (smt (verit) SigmaI  $\langle \text{closedin } Z\ C \rangle$  closed_map_def closedin_subset im-
age_subset_iff)
  have *:  $\exists T. \text{openin } (\text{prod\_topology } X\ Y)\ T \wedge (y1, y2) \in T \wedge T \subseteq \text{topspace } X$ 
 $\times \text{topspace } Y - (\lambda x. (f\ x, g\ x))\ 'C$ 
  if  $(y1, y2) \notin (\lambda x. (f\ x, g\ x))\ 'C$  and  $y1: y1 \in \text{topspace } X$  and  $y2: y2 \in$ 
 $\text{topspace } Y$ 

```

```

    for y1 y2
  proof -
    define A where A ≡ topspace Z - (C ∩ {x ∈ topspace Z. f x = y1})
    have A: openin Z A {x ∈ topspace Z. g x = y2} ⊆ A
      using that ⟨closedin Z C⟩ clo that(2) by (auto simp: A_def)
    obtain V0 where openin Y V0 ∧ y2 ∈ V0 and UA: {x ∈ topspace Z. g x
    ∈ V0} ⊆ A
      using g A y2 unfolding closed_map_fibre_neighbourhood by blast
    then obtain V V' where VV: openin Y V ∧ closedin Y V' ∧ y2 ∈ V ∧ V
    ⊆ V' and V' ⊆ V0
      by (metis (no_types, lifting) ⟨regular_space Y⟩ neighbourhood_base_of
    neighbourhood_base_of_closedin)
    with UA have subA: {x ∈ topspace Z. g x ∈ V'} ⊆ A
      by blast
    show ?thesis
    proof -
      define B where B ≡ topspace Z - (C ∩ {x ∈ topspace Z. g x ∈ V'})
      have openin Z B
        using VV ⟨closedin Z C⟩ contg by (fastforce simp: B_def continu-
    ous_map_closedin)
      have {x ∈ topspace Z. f x = y1} ⊆ B
        using A_def subA by (auto simp: A_def B_def)
      then obtain U where openin X U y1 ∈ U and subB: {x ∈ topspace Z. f
    x ∈ U} ⊆ B
        using ⟨openin Z B⟩ y1 f unfolding closed_map_fibre_neighbourhood by
    meson
      show ?thesis
    proof (intro conjI exI)
      show openin (prod_topology X Y) (U × V)
        by (metis VV ⟨openin X U⟩ openin_prod_Times_iff)
      show (y1, y2) ∈ U × V
        by (simp add: VV ⟨y1 ∈ U⟩)
      show U × V ⊆ topspace X × topspace Y - (λx. (f x, g x)) ` C
        using VV ⟨C ⊆ topspace Z⟩ ⟨openin X U⟩ subB
        by (force simp: image_iff B_def subset_iff dest: openin_subset)
      qed
    qed
    qed
    then show openin (prod_topology X Y) (topspace X × topspace Y - (λx. (f
    x, g x)) ` C)
      by (smt (verit, ccfv_threshold) Diff_iff SigmaE openin_subopen)
    qed
  qed

```

lemma *closed_map_paired_gen:*

```

  assumes f: closed_map Z X f and g: closed_map Z Y g
  and D: (regular_space X ∧ continuous_map Z X f ∧ (∀ z ∈ topspace Y. closedin
  Z {x ∈ topspace Z. g x = z}))

```

$\vee \text{regular_space } Y \wedge \text{continuous_map } Z Y g \wedge (\forall y \in \text{topspace } X. \text{closedin } Z \{x \in \text{topspace } Z. f x = y\})$
 (is ?RSX \vee ?RSY)
 shows closed_map Z (prod_topology X Y) ($\lambda x. (f x, g x)$)
 using D
proof
 assume RSX: ?RSX
 have eq: ($\lambda x. (f x, g x) = (\lambda(x,y). (y,x)) \circ (\lambda x. (g x, f x))$)
 by auto
 show ?thesis
 unfolding eq
proof (rule closed_map_compose)
 show closed_map Z (prod_topology Y X) ($\lambda x. (g x, f x)$)
 using RSX closed_map_paired_gen_aux f g by fastforce
 show closed_map (prod_topology Y X) (prod_topology X Y) ($\lambda(x, y). (y, x)$)
 using homeomorphic_imp_closed_map homeomorphic_map_swap by blast
qed
qed (blast intro: assms closed_map_paired_gen_aux)

lemma closed_map_paired:

assumes closed_map Z X f and contf: continuous_map Z X f
 closed_map Z Y g and contg: continuous_map Z Y g
 and D: t1_space X \wedge regular_space Y \vee regular_space X \wedge t1_space Y
 shows closed_map Z (prod_topology X Y) ($\lambda x. (f x, g x)$)
proof (rule closed_map_paired_gen)
 show regular_space X \wedge continuous_map Z X f \wedge ($\forall z \in \text{topspace } Y. \text{closedin } Z \{x \in \text{topspace } Z. g x = z\}$) \vee regular_space Y \wedge continuous_map Z Y g \wedge ($\forall y \in \text{topspace } X. \text{closedin } Z \{x \in \text{topspace } Z. f x = y\}$)
 using D contf contg
 by (smt (verit, del_insts) Collect_cong closedin_continuous_map_preimage t1_space_closedin_singleton_singleton_iff)
qed (use assms in auto)

lemma closed_map_pairwise:

assumes closed_map Z X (fst \circ f) continuous_map Z X (fst \circ f)
 closed_map Z Y (snd \circ f) continuous_map Z Y (snd \circ f)
 t1_space X \wedge regular_space Y \vee regular_space X \wedge t1_space Y
 shows closed_map Z (prod_topology X Y) f
proof –
 have closed_map Z (prod_topology X Y) ($\lambda a. ((fst \circ f) a, (snd \circ f) a)$)
 using assms closed_map_paired by blast
 then show ?thesis
 by auto
qed

lemma continuous_imp_proper_map:

$\llbracket \text{compact_space } X; \text{kc_space } Y; \text{continuous_map } X Y f \rrbracket \implies \text{proper_map } X Y$
 f
 by (simp add: continuous_closed_imp_proper_map continuous_imp_closed_map_gen)

kc_imp_t1_space)

lemma *tube_lemma_right*:

assumes W : *openin* (*prod_topology* X Y) W **and** C : *compactin* Y C
and x : $x \in \text{topspace } X$ **and** $\text{sub}W$: $\{x\} \times C \subseteq W$
shows $\exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge x \in U \wedge C \subseteq V \wedge U \times V \subseteq W$
proof (*cases* $C = \{\}$)
case *True*
with x **show** *?thesis* **by** *auto*
next
case *False*
have $\exists U V. \text{openin } X U \wedge \text{openin } Y V \wedge x \in U \wedge y \in V \wedge U \times V \subseteq W$
if $y \in C$ **for** y
using W *openin_prod_topology_alt* $\text{sub}W$ *subsetD* **that** **by** *fastforce*
then obtain $U V$ **where** UV : $\bigwedge y. y \in C \implies \text{openin } X (U y) \wedge \text{openin } Y (V y) \wedge x \in U y \wedge y \in V y \wedge U y \times V y \subseteq W$
by *metis*
then obtain D **where** D : *finite* D $D \subseteq C$ $C \subseteq \bigcup (V \text{ ' } D)$
using *compactinD* [*OF* C , *of* $V \text{ ' } C$]
by (*smt* (*verit*) *UN_I* *finite_subset_image* *imageE* *subsetI*)
show *?thesis*
proof (*intro* *exI* *conjI*)
show *openin* X ($\bigcap (U \text{ ' } D)$) *openin* Y ($\bigcup (V \text{ ' } D)$)
using D *False* UV **by** *blast+*
show $x \in \bigcap (U \text{ ' } D)$ $C \subseteq \bigcup (V \text{ ' } D) \cap (\bigcap (U \text{ ' } D)) \times \bigcup (V \text{ ' } D) \subseteq W$
using D UV **by** *force+*
qed
qed

lemma *closed_map_fst*:

assumes *compact_space* Y
shows *closed_map* (*prod_topology* X Y) X *fst*
proof –
have $*$: $\{x \in \text{topspace } X \times \text{topspace } Y. \text{fst } x \in U\} = U \times \text{topspace } Y$
if $U \subseteq \text{topspace } X$ **for** U
using *that* **by** *force*
have $**$: $\bigwedge U y. \llbracket \text{openin } (\text{prod_topology } X Y) U; y \in \text{topspace } X; \{x \in \text{topspace } X \times \text{topspace } Y. \text{fst } x = y\} \subseteq U \rrbracket$
 $\implies \exists V. \text{openin } X V \wedge y \in V \wedge V \times \text{topspace } Y \subseteq U$
using *tube_lemma_right*[*of* X Y $_$ *topspace* Y] *assms* **by** (*fastforce* *simp*:
compact_space_def)
show *?thesis*
unfolding *closed_map_fibre_neighbourhood*
by (*force* *simp*: $*$ *openin_subset* *cong*: *conj_cong* *intro*: $**$)
qed

lemma *closed_map_snd*:

```

assumes compact_space X
shows closed_map (prod_topology X Y) Y snd
proof –
  have snd = fst o prod.swap
    by force
  moreover have closed_map (prod_topology X Y) Y (fst o prod.swap)
  proof (rule closed_map_compose)
    show closed_map (prod_topology X Y) (prod_topology Y X) prod.swap
      by (metis (no_types, lifting) homeomorphic_imp_closed_map homeomor-
        phic_map_eq homeomorphic_map_swap prod.swap_def split_beta)
    show closed_map (prod_topology Y X) Y fst
      by (simp add: closed_map_fst assms)
  qed
  ultimately show ?thesis
    by metis
qed

```

```

lemma closed_map_paired_closed_map_right:
  [[closed_map X Y f; regular_space X;
     $\bigwedge y. y \in \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x = y\}$ ]
   $\implies$  closed_map X (prod_topology X Y) ( $\lambda x. (x, f x)$ )
  by (rule closed_map_paired_gen [OF closed_map_id, unfolded id_def]) auto

```

```

lemma closed_map_paired_closed_map_left:
assumes closed_map X Y f regular_space X
   $\bigwedge y. y \in \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x = y\}$ 
shows closed_map X (prod_topology Y X) ( $\lambda x. (f x, x)$ )
proof –
  have eq: ( $\lambda x. (f x, x)$ ) = ( $\lambda(x,y). (y,x)$ ) o ( $\lambda x. (x, f x)$ )
    by auto
  show ?thesis
    unfolding eq
  proof (rule closed_map_compose)
    show closed_map X (prod_topology X Y) ( $\lambda x. (x, f x)$ )
      by (simp add: assms closed_map_paired_closed_map_right)
    show closed_map (prod_topology X Y) (prod_topology Y X) ( $\lambda(x, y). (y, x)$ )
      using homeomorphic_imp_closed_map homeomorphic_map_swap by blast
  qed
qed

```

```

lemma closed_map_imp_closed_graph:
assumes closed_map X Y f regular_space X
   $\bigwedge y. y \in \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x = y\}$ 
shows closedin (prod_topology X Y) (( $\lambda x. (x, f x)$ ) ' topspace X)
using assms closed_map_def closed_map_paired_closed_map_right by blast

```

```

lemma proper_map_paired_closed_map_right:
assumes closed_map X Y f regular_space X
   $\bigwedge y. y \in \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x = y\}$ 

```

shows *proper_map* X (*prod_topology* X Y) ($\lambda x. (x, f x)$)
by (*simp add: assms closed_injective_imp_proper_map inj_on_def closed_map_paired_closed_map_right*)

lemma *proper_map_paired_closed_map_left*:

assumes *closed_map* X Y f *regular_space* X

$\bigwedge y. y \in \text{topspace } Y \implies \text{closedin } X \{x \in \text{topspace } X. f x = y\}$

shows *proper_map* X (*prod_topology* Y X) ($\lambda x. (f x, x)$)

by (*simp add: assms closed_injective_imp_proper_map inj_on_def closed_map_paired_closed_map_left*)

proposition *regular_space_continuous_proper_map_image*:

assumes *regular_space* X **and** *contf*: *continuous_map* X Y f **and** *pmapf*:

proper_map X Y f

and *fm*: $f' (\text{topspace } X) = \text{topspace } Y$

shows *regular_space* Y

unfolding *regular_space_def*

proof *clarify*

fix C y

assume *closedin* Y C **and** $y \in \text{topspace } Y$ **and** $y \notin C$

have *closed_map* X Y f ($\forall y \in \text{topspace } Y. \text{compactin } X \{x \in \text{topspace } X. f x = y\}$)

using *pmapf proper_map_def* **by** *force+*

moreover **have** *closedin* X $\{z \in \text{topspace } X. f z \in C\}$

using $\langle \text{closedin } Y C \rangle$ *contf continuous_map_closedin* **by** *fastforce*

moreover **have** *disjnt* $\{z \in \text{topspace } X. f z = y\}$ $\{z \in \text{topspace } X. f z \in C\}$

using $\langle y \notin C \rangle$ *disjnt_iff* **by** *blast*

ultimately

obtain U V **where** UV : *openin* X U *openin* X V $\{z \in \text{topspace } X. f z = y\} \subseteq$

$U \{z \in \text{topspace } X. f z \in C\} \subseteq V$

and dUV : *disjnt* U V

using $\langle y \in \text{topspace } Y \rangle$ $\langle \text{regular_space } X \rangle$ **unfolding** *regular_space_compact_closed_sets*
by *meson*

have $*$: $\bigwedge U$ $T. \text{openin } X$ $U \wedge T \subseteq \text{topspace } Y \wedge \{x \in \text{topspace } X. f x \in T\} \subseteq$
 $U \longrightarrow$

$(\exists V. \text{openin } Y$ $V \wedge T \subseteq V \wedge \{x \in \text{topspace } X. f x \in V\} \subseteq U)$

using $\langle \text{closed_map } X$ Y $f \rangle$ **unfolding** *closed_map_preimage_neighbourhood*

by *blast*

obtain $V1$ **where** $V1$: *openin* Y $V1 \wedge y \in V1$ **and** *sub1*: $\{x \in \text{topspace } X. f x \in V1\} \subseteq U$

using $*$ [*of* U $\{y\}$] UV $\langle y \in \text{topspace } Y \rangle$ **by** *auto*

moreover

obtain $V2$ **where** *openin* Y $V2 \wedge C \subseteq V2$ **and** *sub2*: $\{x \in \text{topspace } X. f x \in V2\} \subseteq V$

by (*smt* (*verit*, *ccfv_SIG*) $*$ UV $\langle \text{closedin } Y$ $C \rangle$ *closedin_subset mem_Collect_eq subset_iff*)

moreover **have** *disjnt* $V1$ $V2$

proof $-$

have $\bigwedge x. [\forall x. x \in U \longrightarrow x \notin V; x \in V1; x \in V2] \implies \text{False}$

by (*smt* (*verit*) $V1$ *fm image_iff mem_Collect_eq openin_subset sub1 sub2*)

```

subsetD)
  with dUV show ?thesis by (auto simp: disjnt_iff)
qed
ultimately show  $\exists U V. \text{openin } Y U \wedge \text{openin } Y V \wedge y \in U \wedge C \subseteq V \wedge \text{disjnt } U V$ 
  by meson
qed

```

```

lemma regular_space_perfect_map_image:
   $\llbracket \text{regular\_space } X; \text{perfect\_map } X Y f \rrbracket \implies \text{regular\_space } Y$ 
  by (meson perfect_map_def regular_space_continuous_proper_map_image)

```

```

proposition regular_space_perfect_map_image_eq:
  assumes Hausdorff_space X and perf: perfect_map X Y f
  shows regular_space X  $\longleftrightarrow$  regular_space Y (is ?lhs=?rhs)
proof
  assume ?lhs
  then show ?rhs
    using perf regular_space_perfect_map_image by blast
next
  assume R: ?rhs
  have continuous_map X Y f proper_map X Y f and fim:  $f^{-1}(\text{topspace } X) = \text{topspace } Y$ 
  using perf by (auto simp: perfect_map_def)
  then have closed_map X Y f and preYf:  $(\forall y \in \text{topspace } Y. \text{compactin } X \{x \in \text{topspace } X. f x = y\})$ 
  by (simp_all add: proper_map_def)
  show ?lhs
    unfolding regular_space_def
  proof clarify
    fix C x
    assume closedin X C and  $x \in \text{topspace } X$  and  $x \notin C$ 
    obtain U1 U2 where openin X U1 openin X U2  $\{x\} \subseteq U1$  and disjnt U1 U2
      and subV:  $C \cap \{z \in \text{topspace } X. f z = f x\} \subseteq U2$ 
    proof (rule Hausdorff_space_compact_separation [of X {x} C  $\cap \{z \in \text{topspace } X. f z = f x\}$ , OF  $\langle \text{Hausdorff\_space } X \rangle$ ])
      show compactin X {x}
        by (simp add:  $\langle x \in \text{topspace } X \rangle$ )
      show compactin X  $(C \cap \{z \in \text{topspace } X. f z = f x\})$ 
        using  $\langle \text{closedin } X C \rangle$  fim  $\langle x \in \text{topspace } X \rangle$  closed_Int_compactin preYf by
      fastforce
      show disjnt {x}  $(C \cap \{z \in \text{topspace } X. f z = f x\})$ 
        using  $\langle x \notin C \rangle$  by force
    qed
    have closedin Y  $(f^{-1}(C - U2))$ 
      using  $\langle \text{closed\_map } X Y f \rangle$   $\langle \text{closedin } X C \rangle$   $\langle \text{openin } X U2 \rangle$  closed_map_def
  by blast
  moreover
  have  $f x \in \text{topspace } Y - f^{-1}(C - U2)$ 

```



```

    using ⟨closedin X C⟩ ⟨continuous_map X Y f⟩ ⟨x ∈ topspace X⟩ closedin_subset
    continuous_map_def subV
    by (fastforce simp: Pi_iff)
  ultimately
  obtain V1 V2 where VV: openin Y V1 openin Y V2 f x ∈ V1
    and subV2: f ' (C - U2) ⊆ V2 and disjnt V1 V2
    by (meson R regular_space_def)
  show ∃ U U'. openin X U ∧ openin X U' ∧ x ∈ U ∧ C ⊆ U' ∧ disjnt U U'
  proof (intro exI conjI)
    show openin X (U1 ∩ {x ∈ topspace X. f x ∈ V1})
    using VV(1) ⟨continuous_map X Y f⟩ ⟨openin X U1⟩ continuous_map by
  fastforce
    show openin X (U2 ∪ {x ∈ topspace X. f x ∈ V2})
    using VV(2) ⟨continuous_map X Y f⟩ ⟨openin X U2⟩ continuous_map by
  fastforce
    show x ∈ U1 ∩ {x ∈ topspace X. f x ∈ V1}
    using VV(3) ⟨x ∈ topspace X⟩ ⟨{x} ⊆ U1⟩ by auto
    show C ⊆ U2 ∪ {x ∈ topspace X. f x ∈ V2}
    using ⟨closedin X C⟩ closedin_subset subV2 by auto
    show disjnt (U1 ∩ {x ∈ topspace X. f x ∈ V1}) (U2 ∪ {x ∈ topspace X. f x
  ∈ V2})
    using ⟨disjnt U1 U2⟩ ⟨disjnt V1 V2⟩ by (auto simp: disjnt_iff)
  qed
  qed
  qed

```

6.6.13 Locally compact spaces

definition *locally_compact_space*

where *locally_compact_space* X ≡

$\forall x \in \text{topspace } X. \exists U K. \text{openin } X U \wedge \text{compactin } X K \wedge x \in U \wedge U \subseteq K$

lemma *homeomorphic_locally_compact_spaceD*:

assumes X: *locally_compact_space* X and X *homeomorphic_space* Y

shows *locally_compact_space* Y

proof –

obtain f where hmf: *homeomorphic_map* X Y f

using assms *homeomorphic_space* by blast

then have eq: *topspace* Y = f ' (*topspace* X)

by (simp add: *homeomorphic_imp_surjective_map*)

have $\exists V K. \text{openin } Y V \wedge \text{compactin } Y K \wedge f x \in V \wedge V \subseteq K$

if $x \in \text{topspace } X \text{ openin } X U \text{ compactin } X K x \in U U \subseteq K$ for $x U K$

using that

by (meson hmf *homeomorphic_map_compactness_eq* *homeomorphic_map_openness_eq* *image_mono* *image_eqI*)

with X show ?thesis

by (smt (verit) eq *image_iff* *locally_compact_space_def*)

qed

lemma *homeomorphic_locally_compact_space*:
assumes *X homeomorphic_space Y*
shows *locally_compact_space X* \longleftrightarrow *locally_compact_space Y*
by (*meson assms homeomorphic_locally_compact_spaceD homeomorphic_space_sym*)

lemma *locally_compact_space_retraction_map_image*:
assumes *retraction_map X Y r* **and** *X: locally_compact_space X*
shows *locally_compact_space Y*
proof –
obtain *s* **where** *s: retraction_maps X Y r s*
using *assms retraction_map_def* **by** *blast*
obtain *T* **where** *T retract_of_space X* **and** *Teq: T = s ' topspace Y*
using *retraction_maps_section_image1 s* **by** *blast*
then obtain *r* **where** *r: continuous_map X (subtopology X T) r* $\forall x \in T. r x =$
x
by (*meson retract_of_space_def*)
have *locally_compact_space (subtopology X T)*
unfolding *locally_compact_space_def openin_subtopology_alt*
proof *clarisimp*
fix *x*
assume *x \in topspace X x \in T*
obtain *U K* **where** *UK: openin X U \wedge compactin X K \wedge x \in U \wedge U \subseteq K*
by (*meson X \langle x \in topspace X \rangle locally_compact_space_def*)
then have *compactin (subtopology X T) (r ' K) \wedge T \cap U \subseteq r ' K*
by (*smt (verit) IntD1 image_compactin_image_iff inf_le2 r_subset_iff*)
then show $\exists U. openin X U \wedge (\exists K. compactin (subtopology X T) K \wedge x \in U$
 $\wedge T \cap U \subseteq K)$
using *UK* **by** *auto*
qed
with *Teq* **show** *?thesis*
using *homeomorphic_locally_compact_space retraction_maps_section_image2*
s **by** *blast*
qed

lemma *compact_imp_locally_compact_space*:
compact_space X \implies *locally_compact_space X*
using *compact_space_def locally_compact_space_def* **by** *blast*

lemma *neighbourhood_base_imp_locally_compact_space*:
neighbourhood_base_of (compactin X) X \implies *locally_compact_space X*
by (*metis locally_compact_space_def neighbourhood_base_of openin_tospace*)

lemma *locally_compact_imp_neighbourhood_base*:
assumes *loc: locally_compact_space X* **and** *reg: regular_space X*
shows *neighbourhood_base_of (compactin X) X*
unfolding *neighbourhood_base_of*
proof *clarify*
fix *W x*
assume *openin X W* **and** *x \in W*

then obtain $U K$ **where** $\text{openin } X U \text{ compactin } X K x \in U U \subseteq K$
by (*metis* *loc* *locally_compact_space_def* *openin_subset* *subsetD*)
moreover have $\text{openin } X (U \cap W) \wedge x \in U \cap W$
using $\langle \text{openin } X W \rangle \langle x \in W \rangle \langle \text{openin } X U \rangle \langle x \in U \rangle$ **by** *blast*
then have $\exists u' v. \text{openin } X u' \wedge \text{closedin } X v \wedge x \in u' \wedge u' \subseteq v \wedge v \subseteq U \wedge v \subseteq W$
using *reg*
by (*metis* *le_infE* *neighbourhood_base_of_neighbourhood_base_of_closedin*)
then show $\exists U V. \text{openin } X U \wedge \text{compactin } X V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W$
by (*meson* $\langle U \subseteq K \rangle \langle \text{compactin } X K \rangle \text{closed_compactin}$ *subset_trans*)
qed

lemma *Hausdorff_regular*: $\llbracket \text{Hausdorff_space } X; \text{neighbourhood_base_of } (\text{compactin } X) X \rrbracket \implies \text{regular_space } X$

by (*metis* *compactin_imp_closedin_neighbourhood_base_of_closedin_neighbourhood_base_of_mono*)

lemma *locally_compact_Hausdorff_imp_regular_space*:

assumes *loc*: *locally_compact_space* X **and** *Hausdorff_space* X

shows *regular_space* X

unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*

proof *clarify*

fix $W x$

assume $\text{openin } X W$ **and** $x \in W$

then have $x \in \text{topspace } X$

using *openin_subset* **by** *blast*

then obtain $U K$ **where** $\text{openin } X U \text{ compactin } X K$ **and** $UK: x \in U U \subseteq K$

by (*meson* *loc* *locally_compact_space_def*)

with $\langle \text{Hausdorff_space } X \rangle$ **have** *regular_space* (*subtopology* $X K$)

using *Hausdorff_space_subtopology* *compact_Hausdorff_imp_regular_space* *compact_space_subtopology* **by** *blast*

then have $\exists U' V'. \text{openin } (\text{subtopology } X K) U' \wedge \text{closedin } (\text{subtopology } X K) V' \wedge x \in U' \wedge U' \subseteq V' \wedge V' \subseteq K \cap W$

unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*

by (*meson* *IntI* $\langle U \subseteq K \rangle \langle \text{openin } X W \rangle \langle x \in U \rangle \langle x \in W \rangle \text{openin_subtopology_Int2}$ *subsetD*)

then obtain $V C$ **where** $\text{openin } X V \text{ closedin } X C$ **and** $VC: x \in K \cap V K \cap V \subseteq K \cap C K \cap C \subseteq K \cap W$

by (*metis* *Int_commute* *closedin_subtopology* *openin_subtopology*)

show $\exists U V. \text{openin } X U \wedge \text{closedin } X V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W$

proof (*intro* *conjI* *exI*)

show $\text{openin } X (U \cap V)$

using $\langle \text{openin } X U \rangle \langle \text{openin } X V \rangle$ **by** *blast*

show $\text{closedin } X (K \cap C)$

using $\langle \text{closedin } X C \rangle \langle \text{compactin } X K \rangle \text{compactin_imp_closedin}$ $\langle \text{Hausdorff_space } X \rangle$ **by** *blast*

qed (*use* $UK VC$ **in** *auto*)

qed

lemma *locally_compact_space_neighbourhood_base*:

Hausdorff_space X \vee *regular_space X*

\implies *locally_compact_space X* \longleftrightarrow *neighbourhood_base_of (compactin X)*

X

by (*metis locally_compact_imp_neighbourhood_base locally_compact_Hausdorff_imp_regular_space neighbourhood_base_imp_locally_compact_space*)

lemma *locally_compact_Hausdorff_or_regular*:

locally_compact_space X \wedge (*Hausdorff_space X* \vee *regular_space X*) \longleftrightarrow *locally_compact_space X* \wedge *regular_space X*

using *locally_compact_Hausdorff_imp_regular_space* **by** *blast*

lemma *locally_compact_space_compact_closedin*:

assumes *Hausdorff_space X* \vee *regular_space X*

shows *locally_compact_space X* \longleftrightarrow

$(\forall x \in \text{topspace } X. \exists U K. \text{openin } X U \wedge \text{compactin } X K \wedge \text{closedin } X K$

$\wedge x \in U \wedge U \subseteq K)$

using *locally_compact_Hausdorff_or_regular* **unfolding** *locally_compact_space_def*

by (*metis assms closed_compactin inf.absorb_iff2 le_infE neighbourhood_base_of neighbourhood_base_of_closedin*)

lemma *locally_compact_space_compact_closure_of*:

assumes *Hausdorff_space X* \vee *regular_space X*

shows *locally_compact_space X* \longleftrightarrow

$(\forall x \in \text{topspace } X. \exists U. \text{openin } X U \wedge \text{compactin } X (X \text{ closure_of } U) \wedge x$

$\in U)$ (**is** *?lhs=?rhs*)

proof

assume *?lhs* **then show** *?rhs*

by (*metis assms closed_compactin closedin_closure_of closure_of_eq closure_of_mono locally_compact_space_compact_closedin*)

next

assume *?rhs* **then show** *?lhs*

by (*meson closure_of_subset locally_compact_space_def openin_subset*)

qed

lemma *locally_compact_space_neighbourhood_base_closedin*:

assumes *Hausdorff_space X* \vee *regular_space X*

shows *locally_compact_space X* \longleftrightarrow *neighbourhood_base_of* $(\lambda C. \text{compactin } X C \wedge \text{closedin } X C)$ *X* (**is** *?lhs=?rhs*)

proof

assume *L*: *?lhs*

then have *regular_space X*

using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*

with *L* **have** *neighbourhood_base_of (compactin X) X*

by (*simp add: locally_compact_imp_neighbourhood_base*)

with $\langle \text{regular_space } X \rangle$ **show** *?rhs*

by (*smt (verit, ccfv_threshold) closed_compactin neighbourhood_base_of_subset_trans neighbourhood_base_of_closedin*)

```

next
  assume ?rhs then show ?lhs
    using neighbourhood_base_imp_locally_compact_space neighbourhood_base_of_mono
  by blast
qed

```

```

lemma locally_compact_space_neighbourhood_base_closure_of:
  assumes Hausdorff_space X  $\vee$  regular_space X
  shows locally_compact_space X  $\longleftrightarrow$  neighbourhood_base_of ( $\lambda T$ . compactin X
(X closure_of T)) X
  (is ?lhs=?rhs)

```

```

proof
  assume L: ?lhs
  then have regular_space X
    using assms locally_compact_Hausdorff_imp_regular_space by blast
  with L have neighbourhood_base_of ( $\lambda A$ . compactin X  $A \wedge$  closedin X A) X
    using locally_compact_space_neighbourhood_base_closedin by blast
  then show ?rhs
    by (simp add: closure_of_closedin neighbourhood_base_of_mono)
next
  assume ?rhs then show ?lhs
    unfolding locally_compact_space_def neighbourhood_base_of
  by (meson closure_of_subset openin_topspace subset_trans)
qed

```

```

lemma locally_compact_space_neighbourhood_base_open_closure_of:
  assumes Hausdorff_space X  $\vee$  regular_space X
  shows locally_compact_space X  $\longleftrightarrow$ 
    neighbourhood_base_of ( $\lambda U$ . openin X U  $\wedge$  compactin X (X closure_of
U)) X
  (is ?lhs=?rhs)

```

```

proof
  assume L: ?lhs
  then have regular_space X
    using assms locally_compact_Hausdorff_imp_regular_space by blast
  then have neighbourhood_base_of ( $\lambda T$ . compactin X (X closure_of T)) X
    using L locally_compact_space_neighbourhood_base_closure_of by auto
  with L show ?rhs
    unfolding neighbourhood_base_of
  by (meson closed_compactin closure_of_closure_of closure_of_eq closure_of_mono
subset_trans)
next
  assume ?rhs then show ?lhs
    unfolding locally_compact_space_def neighbourhood_base_of
  by (meson closure_of_subset openin_topspace subset_trans)
qed

```

```

lemma locally_compact_space_compact_closed_compact:
  assumes Hausdorff_space X  $\vee$  regular_space X

```

shows *locally_compact_space* $X \longleftrightarrow$
 $(\forall K. \text{compactin } X \ K$
 $\longrightarrow (\exists U \ L. \text{openin } X \ U \wedge \text{compactin } X \ L \wedge \text{closedin } X \ L \wedge K \subseteq U \wedge$
 $U \subseteq L))$
(is ?lhs=?rhs)

proof

assume $L: ?lhs$
then obtain $U \ L$ **where** $UL: \forall x \in \text{topspace } X. \text{openin } X \ (U \ x) \wedge \text{compactin } X$
 $(L \ x) \wedge \text{closedin } X \ (L \ x) \wedge x \in U \ x \wedge U \ x \subseteq L \ x$
unfolding *locally_compact_space_compact_closedin* [*OF assms*]
by *metis*
show $?rhs$
proof clarify
fix K
assume *compactin* $X \ K$
then have $K \subseteq \text{topspace } X$
by (*simp add: compactin_subset_topspace*)
then have $*$: $(\forall U \in U \text{ ' } K. \text{openin } X \ U) \wedge K \subseteq \bigcup (U \text{ ' } K)$
using UL **by** *blast*
with $\langle \text{compactin } X \ K \rangle$ **obtain** KF **where** $KF: \text{finite } KF \ K \subseteq K \ K \subseteq \bigcup (U$
 $\text{ ' } KF)$
by (*metis compactinD finite_subset_image*)
show $\exists U \ L. \text{openin } X \ U \wedge \text{compactin } X \ L \wedge \text{closedin } X \ L \wedge K \subseteq U \wedge U \subseteq L$
proof (*intro conjI exI*)
show *openin* $X \ (\bigcup (U \text{ ' } KF))$
using $*$ $\langle KF \subseteq K \rangle$ **by** *fastforce*
show *compactin* $X \ (\bigcup (L \text{ ' } KF))$
by (*smt (verit) UL* $\langle K \subseteq \text{topspace } X \rangle$ KF *compactin_Union finite_imageI*
imageE subset_iff)
show *closedin* $X \ (\bigcup (L \text{ ' } KF))$
by (*smt (verit) UL* $\langle K \subseteq \text{topspace } X \rangle$ KF *closedin_Union finite_imageI*
imageE subsetD)
qed (*use UL* $\langle K \subseteq \text{topspace } X \rangle$ KF **in** *auto*)
qed
next
assume $?rhs$ **then show** $?lhs$
by (*metis compactin_sing insert_subset locally_compact_space_def*)
qed

lemma *locally_compact_regular_space_neighbourhood_base*:

locally_compact_space $X \wedge \text{regular_space } X \longleftrightarrow$
 $\text{neighbourhood_base_of } (\lambda C. \text{compactin } X \ C \wedge \text{closedin } X \ C) \ X$
using *locally_compact_space_neighbourhood_base_closedin* *neighbourhood_base_of_closedin*
neighbourhood_base_of_mono **by** *blast*

lemma *locally_compact_kc_space*:

neighbourhood_base_of (*compactin* X) $X \wedge \text{kc_space } X \longleftrightarrow$
 $\text{locally_compact_space } X \wedge \text{Hausdorff_space } X$
using *Hausdorff_imp_kc_space* *locally_compact_imp_kc_eq_Hausdorff_space*

locally_compact_space_neighbourhood_base **by** *blast*

lemma *locally_compact_kc_space_alt*:

neighbourhood_base_of (*compactin* X) $X \wedge kc_space\ X \longleftrightarrow$
locally_compact_space $X \wedge Hausdorff_space\ X \wedge regular_space\ X$
using *Hausdorff_regular_locally_compact_kc_space* **by** *blast*

lemma *locally_compact_kc_imp_regular_space*:

\llbracket *neighbourhood_base_of* (*compactin* X) $X; kc_space\ X \rrbracket \implies regular_space\ X$
using *Hausdorff_regular_locally_compact_imp_kc_eq_Hausdorff_space* **by** *blast*

lemma *kc_locally_compact_space*:

kc_space X
 \implies *neighbourhood_base_of* (*compactin* X) $X \longleftrightarrow$ *locally_compact_space* $X \wedge$
Hausdorff_space $X \wedge regular_space\ X$
using *Hausdorff_regular_locally_compact_kc_space* **by** *blast*

lemma *locally_compact_space_closed_subset*:

assumes *loc*: *locally_compact_space* X **and** *closedin* $X\ S$
shows *locally_compact_space* (*subtopology* $X\ S$)
proof (*clarsimp simp: locally_compact_space_def*)
fix x **assume** $x \in topspace\ X\ x \in S$
then obtain $U\ K$ **where** UK : *openin* $X\ U \wedge compactin\ X\ K \wedge x \in U \wedge U \subseteq K$
by (*meson loc locally_compact_space_def*)
show $\exists U. openin\ (subtopology\ X\ S)\ U \wedge$
 $(\exists K. compactin\ (subtopology\ X\ S)\ K \wedge x \in U \wedge U \subseteq K)$
proof (*intro conjI exI*)
show *openin* (*subtopology* $X\ S$) ($S \cap U$)
by (*simp add: UK openin_subtopology_Int2*)
show *compactin* (*subtopology* $X\ S$) ($S \cap K$)
by (*simp add: UK assms(2) closed_Int_compactin_compactin_subtopology*)
qed (*use UK x in auto*)
qed

lemma *locally_compact_space_open_subset*:

assumes X : *Hausdorff_space* $X \vee regular_space\ X$ **and** *loc*: *locally_compact_space* X **and** *openin* $X\ S$
shows *locally_compact_space* (*subtopology* $X\ S$)
proof (*clarsimp simp: locally_compact_space_def*)
fix x **assume** $x \in topspace\ X\ x \in S$
then obtain $U\ K$ **where** UK : *openin* $X\ U \wedge compactin\ X\ K \wedge x \in U \wedge U \subseteq K$
by (*meson loc locally_compact_space_def*)
moreover have *reg*: *regular_space* X
using X *loc locally_compact_Hausdorff_imp_regular_space* **by** *blast*
moreover have *openin* $X\ (U \cap S)$
by (*simp add: UK <openin X S> openin_Int*)
ultimately obtain $V\ C$
where VC : *openin* $X\ V \wedge closedin\ X\ C \wedge x \in V \wedge V \subseteq C \wedge C \subseteq U \wedge C \subseteq S$

```

by (metis ⟨ $x \in S$ ⟩ IntI le_inf_iff neighbourhood_base_of_neighbourhood_base_of_closedin)
show  $\exists U. \text{openin } (\text{subtopology } X \ S) \ U \wedge$ 
      ( $\exists K. \text{compactin } (\text{subtopology } X \ S) \ K \wedge x \in U \wedge U \subseteq K$ )
proof (intro conjI exI)
  show  $\text{openin } (\text{subtopology } X \ S) \ V$ 
    using VC by (meson ⟨ $\text{openin } X \ S$ ⟩ openin_open_subtopology_order_trans)
  show  $\text{compactin } (\text{subtopology } X \ S) \ (C \cap K)$ 
    using UK VC closed_Int_compactin compactin_subtopology by fastforce
qed (use UK VC x in auto)
qed

```

lemma *locally_compact_space_discrete_topology:*

locally_compact_space (discrete_topology U)

by (simp add: neighbourhood_base_imp_locally_compact_space neighbourhood_base_of_discrete_topo)

lemma *locally_compact_space_continuous_open_map_image:*

$\llbracket \text{continuous_map } X \ X' \ f; \text{open_map } X \ X' \ f;$

$f' \text{ topspace } X = \text{topspace } X'; \text{locally_compact_space } X \rrbracket \implies \text{locally_compact_space } X'$

unfolding *locally_compact_space_def open_map_def*

by (smt (verit, ccfv_SIG) image_compactin_image_iff_image_mono)

lemma *locally_compact_subspace_openin_closure_of:*

assumes Hausdorff_space X **and** $S: S \subseteq \text{topspace } X$

and loc: *locally_compact_space (subtopology X S)*

shows $\text{openin } (\text{subtopology } X \ (X \ \text{closure_of } S)) \ S$

unfolding *openin_subopen [where S=S]*

proof *clarify*

fix a **assume** $a \in S$

then obtain T K **where** *: $\text{openin } X \ T \ \text{compactin } X \ K \ K \subseteq S \ a \in S \ a \in T \ S$

$\cap T \subseteq K$

using loc **unfolding** *locally_compact_space_def*

by (metis IntE S compactin_subtopology_inf_commute openin_subtopology_topspace_subtopology_subset)

have $T \cap X \ \text{closure_of } S \subseteq X \ \text{closure_of } (T \cap S)$

by (simp add: *(1) openin_Int_closure_of_subset)

also have $\dots \subseteq S$

using * ⟨Hausdorff_space X⟩ **by** (metis closure_of_minimal compactin_imp_closedin order.trans_inf_commute)

finally have $T \cap X \ \text{closure_of } S \subseteq T \cap S$ **by** simp

then have $\text{openin } (\text{subtopology } X \ (X \ \text{closure_of } S)) \ (T \cap S)$

unfolding *openin_subtopology using ⟨openin X T⟩ S closure_of_subset by fastforce*

with * **show** $\exists T. \text{openin } (\text{subtopology } X \ (X \ \text{closure_of } S)) \ T \wedge a \in T \wedge T \subseteq S$

by blast

qed

lemma *locally_compact_subspace_closed_Int_openin:*

$\llbracket \text{Hausdorff_space } X \wedge S \subseteq \text{topspace } X \wedge \text{locally_compact_space}(\text{subtopology } X$

$S)$]

 $\implies \exists C U. \text{closedin } X C \wedge \text{openin } X U \wedge C \cap U = S$

by (*metis* *closedin_closure_of_inf_commute* *locally_compact_subspace_openin_closure_of_openin_subtopology*)

lemma *locally_compact_subspace_open_in_closure_of_eq*:

assumes *Hausdorff_space* X **and** *loc*: *locally_compact_space* X

shows $\text{openin } (\text{subtopology } X (X \text{ closure_of } S)) S \longleftrightarrow S \subseteq \text{topspace } X \wedge \text{locally_compact_space}(\text{subtopology } X S)$ (**is** $?lhs=?rhs$)

proof

assume L : $?lhs$

then obtain $S \subseteq \text{topspace } X$ *regular_space* X

using *assms* *locally_compact_Hausdorff_imp_regular_space* *openin_subset* **by**

fastforce

then have *locally_compact_space* (*subtopology* (*subtopology* $X (X \text{ closure_of } S)$)

 S)

by (*simp* *add*: L *loc* *locally_compact_space_closed_subset* *locally_compact_space_open_subset*

regular_space_subtopology)

then show $?rhs$

by (*metis* L *inf.orderE* *inf_commute* *le_inf_iff* *openin_subset* *subtopology_subtopology*

topspace_subtopology)

next

assume $?rhs$ **then show** $?lhs$

using *assms* *locally_compact_subspace_openin_closure_of* **by** *blast*

qed

lemma *locally_compact_subspace_closed_Int_openin_eq*:

assumes *Hausdorff_space* X **and** *loc*: *locally_compact_space* X

shows $(\exists C U. \text{closedin } X C \wedge \text{openin } X U \wedge C \cap U = S) \longleftrightarrow S \subseteq \text{topspace } X$

 $\wedge \text{locally_compact_space}(\text{subtopology } X S)$ (**is** $?lhs=?rhs$)

proof

assume L : $?lhs$

then obtain $C U$ **where** $\text{closedin } X C$ $\text{openin } X U$ **and** *Seq*: $S = C \cap U$

by *blast*

then have $C \subseteq \text{topspace } X$

by (*simp* *add*: *closedin_subset*)

have *locally_compact_space* (*subtopology* (*subtopology* $X C$) (*topspace* (*subtopology*

 $X C$) $\cap U$))

proof (*rule* *locally_compact_space_open_subset*)

show *locally_compact_space* (*subtopology* $X C$)

by (*simp* *add*: $\langle \text{closedin } X C \rangle$ *loc* *locally_compact_space_closed_subset*)

show $\text{openin } (\text{subtopology } X C) (\text{topspace } (\text{subtopology } X C) \cap U)$

by (*simp* *add*: $\langle \text{openin } X U \rangle$ *Int_left_commute* *inf_commute* *openin_Int*

openin_subtopology_Int2)

qed (*simp* *add*: *Hausdorff_space_subtopology* $\langle \text{Hausdorff_space } X \rangle$)

then show $?rhs$

by (*metis* *Seq* $\langle C \subseteq \text{topspace } X \rangle$ *inf.coboundedI1* *subtopology_subtopology*

subtopology_topspace)

next

```

assume ?rhs then show ?lhs
  using assms locally_compact_subspace_closed_Int_openin by blast
qed

lemma dense_locally_compact_openin_Hausdorff_space:
   $\llbracket \text{Hausdorff\_space } X; S \subseteq \text{topspace } X; X \text{ closure\_of } S = \text{topspace } X; \text{ locally\_compact\_space } (\text{subtopology } X S) \rrbracket \implies \text{openin } X S$ 
by (metis locally_compact_subspace_openin_closure_of_subtopology_topspace)

lemma locally_compact_space_prod_topology:
   $\text{locally\_compact\_space } (\text{prod\_topology } X Y) \longleftrightarrow$ 
   $(\text{prod\_topology } X Y) = \text{trivial\_topology} \vee$ 
   $\text{locally\_compact\_space } X \wedge \text{locally\_compact\_space } Y$  (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    using locally_compact_space_discrete_topology by force
  next
  case False
  then obtain w z where wz: w ∈ topspace X z ∈ topspace Y
  by fastforce
  show ?thesis
  proof
    assume L: ?lhs then show ?rhs
    by (metis locally_compact_space_retraction_map_image_prod_topology_trivial_iff
retraction_map_fst retraction_map_snd)
  next
    assume R: ?rhs
    show ?lhs
    unfolding locally_compact_space_def
    proof clarsimp
      fix x y
      assume x ∈ topspace X and y ∈ topspace Y
      obtain U C where openin X U compactin X C x ∈ U U ⊆ C
      by (meson False R  $\langle x \in \text{topspace } X \rangle$  locally_compact_space_def)
      obtain V D where openin Y V compactin Y D y ∈ V V ⊆ D
      by (meson False R  $\langle y \in \text{topspace } Y \rangle$  locally_compact_space_def)
      show  $\exists U. \text{openin } (\text{prod\_topology } X Y) U \wedge (\exists K. \text{compactin } (\text{prod\_topology } X Y) K \wedge (x, y) \in U \wedge U \subseteq K)$ 
      proof (intro exI conjI)
        show  $\text{openin } (\text{prod\_topology } X Y) (U \times V)$ 
        by (simp add:  $\langle \text{openin } X U \rangle \langle \text{openin } Y V \rangle$  openin_prod_Times_iff)
        show  $\text{compactin } (\text{prod\_topology } X Y) (C \times D)$ 
        by (simp add:  $\langle \text{compactin } X C \rangle \langle \text{compactin } Y D \rangle$  compactin_Times)
        show  $(x, y) \in U \times V$ 
        by (simp add:  $\langle x \in U \rangle \langle y \in V \rangle$ )
        show  $U \times V \subseteq C \times D$ 
        by (simp add: Sigma_mono  $\langle U \subseteq C \rangle \langle V \subseteq D \rangle$ )
      qed
    qed

```

```

qed
qed
qed

lemma locally_compact_space_product_topology:
  locally_compact_space(product_topology X I)  $\longleftrightarrow$ 
    product_topology X I = trivial_topology  $\vee$ 
    finite {i  $\in$  I.  $\neg$  compact_space(X i)}  $\wedge$  ( $\forall$  i  $\in$  I. locally_compact_space(X
i)) (is ?lhs=?rhs)
proof (cases (product_topology X I) = trivial_topology)
  case True
  then show ?thesis
    by (simp add: locally_compact_space_def)
next
  case False
  show ?thesis
  proof
    assume L: ?lhs
    obtain z where z: z  $\in$  topspace (product_topology X I)
      using False
      by (meson ex_in_conv null_topspace_iff_trivial)
    with L z obtain U C where openin (product_topology X I) U compactin
      (product_topology X I) C z  $\in$  U U  $\subseteq$  C
      by (meson locally_compact_space_def)
    then obtain V where finV: finite {i  $\in$  I. V i  $\neq$  topspace (X i)} and  $\forall$  i  $\in$  I.
      openin (X i) (V i)
      and z  $\in$  PiE I V PiE I V  $\subseteq$  U
      by (auto simp: openin_product_topology_alt)
    have compact_space (X i) if i  $\in$  I V i = topspace (X i) for i
    proof -
      have compactin (X i) (( $\lambda$ x. x i) ' C)
        using  $\langle$ compactin (product_topology X I) C $\rangle$  image_compactin
        by (metis continuous_map_product_projection  $\langle$ i  $\in$  I $\rangle$ )
      moreover have V i  $\subseteq$  ( $\lambda$ x. x i) ' C
      proof -
        have V i  $\subseteq$  ( $\lambda$ x. x i) ' PiE I V
          by (metis  $\langle$ z  $\in$  PiE I V $\rangle$  empty_iff_image_projection_PiE order_refl  $\langle$ i
 $\in$  I $\rangle$ )
        also have ...  $\subseteq$  ( $\lambda$ x. x i) ' C
          using  $\langle$ U  $\subseteq$  C $\rangle$   $\langle$ PiE I V  $\subseteq$  U $\rangle$  by blast
      finally show ?thesis .
    qed
    ultimately show ?thesis
      by (metis closed_compactin closedin_topspace compact_space_def that(2))
  qed
  with finV have finite {i  $\in$  I.  $\neg$  compact_space (X i)}
    by (metis (mono_tags, lifting) mem_Collect_eq finite_subset subsetI)
  moreover have locally_compact_space (X i) if i  $\in$  I for i
    by (meson False L locally_compact_space_retraction_map_image retrac-

```

```

tion_map_product_projection that)
  ultimately show ?rhs by metis
next
  assume R: ?rhs
  show ?lhs
    unfolding locally_compact_space_def
  proof clarsimp
    fix z
    assume z: z ∈ (ΠE i∈I. topspace (X i))
    have ∃ U C. openin (X i) U ∧ compactin (X i) C ∧ z i ∈ U ∧ U ⊆ C ∧
      (compact_space(X i) → U = topspace(X i) ∧ C = topspace(X
i))
      if i ∈ I for i
        using that R z unfolding locally_compact_space_def compact_space_def
      by (metis (no_types, lifting) False PiE_mem openin_topspace set_eq_subset)
    then obtain U C where UC: ∧i. i ∈ I ⇒
      openin (X i) (U i) ∧ compactin (X i) (C i) ∧ z i ∈ U i ∧ U i ⊆ C i ∧
      (compact_space(X i) → U i = topspace(X i) ∧ C i = topspace(X
i))
      by metis
    show ∃ U. openin (product_topology X I) U ∧ (∃ K. compactin (product_topology
X I) K ∧ z ∈ U ∧ U ⊆ K)
    proof (intro exI conjI)
      show openin (product_topology X I) (PiE I U)
      by (smt (verit) Collect_cong False R UC compactin_subspace openin_PiE_gen
subset_antisym subtopology_topspace)
      show compactin (product_topology X I) (PiE I C)
      by (simp add: UC compactin_PiE)
    qed (use UC z in blast)+
  qed
qed
qed

```

lemma *locally_compact_space_sum_topology:*

locally_compact_space (sum_topology X I) ↔ (∀ i ∈ I. locally_compact_space (X i)) (is ?lhs=?rhs)

proof

assume ?lhs then show ?rhs

by (metis closed_map_component_injection embedding_map_imp_homeomorphic_space embedding_map_component_injection

embedding_imp_closed_map_eq_homeomorphic_locally_compact_space locally_compact_space_closed_subset)

next

assume R: ?rhs

show ?lhs

unfolding locally_compact_space_def

proof clarsimp

fix i y

assume i ∈ I and y: y ∈ topspace (X i)

```

then obtain  $U K$  where  $UK$ :  $\text{openin } (X \ i) \ U \ \text{compactin } (X \ i) \ K \ y \in U \ U \subseteq K$ 
using  $R$  by (fastforce simp: locally_compact_space_def)
then show  $\exists U. \text{openin } (\text{sum\_topology } X \ I) \ U \wedge (\exists K. \text{compactin } (\text{sum\_topology } X \ I) \ K \wedge (i, y) \in U \wedge U \subseteq K)$ 
by (metis <i> \in I> continuous_map_component_injection image_compactin image_mono imageI open_map_component_injection open_map_def)
qed
qed

```

```

lemma locally_compact_space_euclidean:
  locally_compact_space (euclidean::'a::heine_borel topology)
unfolding locally_compact_space_def
proof (intro strip)
  fix  $x::'a$ 
  assume  $x \in \text{topspace euclidean}$ 
  have  $\text{ball } x \ 1 \subseteq \text{cball } x \ 1$ 
  by auto
  then show  $\exists U K. \text{openin euclidean } U \wedge \text{compactin euclidean } K \wedge x \in U \wedge U \subseteq K$ 
  by (metis Elementary_Metric_Spaces.open_ball centre_in_ball compact_cball compactin_euclidean_iff open_openin zero_less_one)
qed

```

```

lemma locally_compact_Euclidean_space:
  locally_compact_space (Euclidean_space n)
using homeomorphic_locally_compact_space [OF homeomorphic_Euclidean_space_product_topology]
using locally_compact_space_product_topology locally_compact_space_euclidean
by fastforce

```

```

proposition quotient_map_prod_right:
assumes loc: locally_compact_space Z
and reg: Hausdorff_space Z \vee regular_space Z
and f: quotient_map X Y f
shows quotient_map (prod_topology Z X) (prod_topology Z Y) (\lambda(x,y). (x,f y))
proof -
  define  $h$  where  $h \equiv (\lambda(x::'a,y). (x,f y))$ 
  have continuous_map (prod_topology Z X) Y (f o snd)
  by (simp add: continuous_map_of_snd f quotient_imp_continuous_map)
  then have cmh: continuous_map (prod_topology Z X) (prod_topology Z Y) h
  by (simp add: h_def continuous_map_paired split_def continuous_mapfst o_def)
  have fm: f 'topspace X = topspace Y
  by (simp add: f quotient_imp_surjective_map)
moreover
  have openin (prod_topology Z X) {u \in topspace Z \times topspace X. h u \in W}
   $\longleftrightarrow$  openin (prod_topology Z Y) W (is ?lhs=?rhs)

```

```

if  $W: W \subseteq \text{topspace } Z \times \text{topspace } Y$  for  $W$ 
proof
  define  $S$  where  $S \equiv \{u \in \text{topspace } Z \times \text{topspace } X. h \ u \in W\}$ 
  assume  $?lhs$ 
  then have  $L: \text{openin } (\text{prod\_topology } Z \ X) \ S$ 
    using  $S\_def$  by  $blast$ 
  have  $\exists T. \text{openin } (\text{prod\_topology } Z \ Y) \ T \wedge (x0, z0) \in T \wedge T \subseteq W$ 
    if  $\S: (x0, z0) \in W$  for  $x0 \ z0$ 
  proof  $-$ 
    have  $x0: x0 \in \text{topspace } Z$ 
      using  $W$  that by  $blast$ 
    obtain  $y0$  where  $y0: y0 \in \text{topspace } X \ f \ y0 = z0$ 
      by  $(metis \ W \ fim \ imageE \ insert\_absorb \ insert\_subset \ mem\_Sigma\_iff \ \S)$ 
    then have  $(x0, y0) \in S$ 
      by  $(simp \ add: \ S\_def \ h\_def \ \text{that } x0)$ 
    have  $\text{continuous\_map } Z \ (\text{prod\_topology } Z \ X) \ (\lambda x. (x, y0))$ 
      by  $(simp \ add: \ \text{continuous\_map\_paired } y0)$ 
    with  $\text{openin\_continuous\_map\_preimage } [OF \ L]$ 
    have  $\text{ope\_ZS}: \text{openin } Z \ \{x \in \text{topspace } Z. (x, y0) \in S\}$ 
      by  $blast$ 
    obtain  $U \ U'$  where  $\text{openin } Z \ U \ \text{compactin } Z \ U' \ \text{closedin } Z \ U'$ 
       $x0 \in U \ U \subseteq U' \ U' \subseteq \{x \in \text{topspace } Z. (x, y0) \in S\}$ 
      using  $\text{loc } \text{ope\_ZS } x0 \ \langle (x0, y0) \in S \rangle$ 
      by  $(\text{force } \text{simp: } \text{locally\_compact\_space\_neighbourhood\_base\_closedin } [OF$ 
 $\text{reg}]$ 
         $\text{neighbourhood\_base\_of})$ 
    then have  $D: U' \times \{y0\} \subseteq S$ 
      by  $(\text{auto } \text{simp: } )$ 
    define  $V$  where  $V \equiv \{z \in \text{topspace } Y. U' \times \{y \in \text{topspace } X. f \ y = z\} \subseteq$ 
 $S\}$ 
    have  $z0 \in V$ 
      using  $D \ y0 \ \text{Int\_Collect } \text{fim}$  by  $(\text{fastforce } \text{simp: } h\_def \ V\_def \ S\_def)$ 
    have  $\text{openin } X \ \{x \in \text{topspace } X. f \ x \in V\} \implies \text{openin } Y \ V$ 
      using  $f \ \text{unfolding } V\_def \ \text{quotient\_map\_def } \text{subset\_iff}$ 
      by  $(\text{smt } (\text{verit}, \ \text{del\_insts}) \ \text{Collect\_cong } \text{mem\_Collect\_eq})$ 
    moreover have  $\text{openin } X \ \{x \in \text{topspace } X. f \ x \in V\}$ 
    proof  $-$ 
      let  $?Z = \text{subtopology } Z \ U'$ 
      have  $*: \{x \in \text{topspace } X. f \ x \in V\} = \text{topspace } X - \text{snd } '(U' \times \text{topspace } X - S)$ 
        by  $(\text{force } \text{simp: } V\_def \ S\_def \ h\_def \ \text{simp } \text{flip: } \text{fim})$ 
      have  $\text{compact\_space } ?Z$ 
        using  $\langle \text{compactin } Z \ U' \rangle \ \text{compactin\_subspace}$  by  $\text{auto}$ 
      moreover have  $\text{closedin } (\text{prod\_topology } ?Z \ X) \ (U' \times \text{topspace } X - S)$ 
        by  $(\text{simp } \text{add: } L \ \langle \text{closedin } Z \ U' \rangle \ \text{closedin\_closed\_subtopology } \text{closedin\_diff}$ 
 $\text{closedin\_prod\_Times\_iff}$ 
         $\text{prod\_topology\_subtopology}(1))$ 
      ultimately show  $?thesis$ 
        using  $* \ \text{closed\_map\_snd } \text{closed\_map\_def}$  by  $\text{fastforce}$ 

```

```

qed
ultimately have openin Y V
  by metis
show ?thesis
proof (intro conjI exI)
  show openin (prod_topology Z Y) (U × V)
    by (simp add: openin_prod_Times_iff ‹openin Z U› ‹openin Y V›)
  show (x0, z0) ∈ U × V
    by (simp add: ‹x0 ∈ U› ‹z0 ∈ V›)
  show U × V ⊆ W
    using ‹U ⊆ U'› by (force simp: V_def S_def h_def simp flip: fm)
qed
qed
with openin_subopen show ?rhs by force
next
assume ?rhs then show ?lhs
  using openin_continuous_map_preimage cmh by fastforce
qed
ultimately show ?thesis
  by (fastforce simp: image_iff quotient_map_def h_def)
qed

lemma quotient_map_prod_left:
  assumes loc: locally_compact_space Z
    and reg: Hausdorff_space Z ∨ regular_space Z
    and f: quotient_map X Y f
  shows quotient_map (prod_topology X Z) (prod_topology Y Z) ( $\lambda(x,y). (f x,y)$ )
proof -
  have ( $\lambda(x,y). (f x,y)$ ) = prod.swap ∘ ( $\lambda(x,y). (x,f y)$ ) ∘ prod.swap
    by force
  then
  show ?thesis
    apply (rule ssubst)
  proof (intro quotient_map_compose)
    show quotient_map (prod_topology X Z) (prod_topology Z X) prod.swap
      quotient_map (prod_topology Z Y) (prod_topology Y Z) prod.swap
      using homeomorphic_map_def homeomorphic_map_swap quotient_map_eq
  by fastforce+
    show quotient_map (prod_topology Z X) (prod_topology Z Y) ( $\lambda(x, y). (x, f$ 
y))
      by (simp add: f loc quotient_map_prod_right reg)
  qed
qed
qed

lemma locally_compact_space_perfect_map_preimage:
  assumes locally_compact_space X' and f: perfect_map X X' f
  shows locally_compact_space X
  unfolding locally_compact_space_def
proof (intro strip)

```

```

fix x
assume x: x ∈ topspace X
then obtain U K where openin X' U compactin X' K f x ∈ U U ⊆ K
  using assms unfolding locally_compact_space_def perfect_map_def
  by (metis (no_types, lifting) continuous_map_closedin Pi_iff)
show ∃ U K. openin X U ∧ compactin X K ∧ x ∈ U ∧ U ⊆ K
proof (intro exI conjI)
  have continuous_map X X' f
    using f perfect_map_def by blast
  then show openin X {x ∈ topspace X. f x ∈ U}
    by (simp add: ⟨openin X' U⟩ continuous_map)
  show compactin X {x ∈ topspace X. f x ∈ K}
    using ⟨compactin X' K⟩ f perfect_imp_proper_map proper_map_alt by blast
qed (use x ⟨f x ∈ U⟩ ⟨U ⊆ K⟩ in auto)
qed

```

6.6.14 Special characterizations of classes of functions into and out of R

```

lemma monotone_map_into_euclideanreal_alt:
  assumes continuous_map X euclideanreal f
  shows (∀ k. is_interval k ⟶ connectedin X {x ∈ topspace X. f x ∈ k}) ⟷
    connected_space X ∧ monotone_map X euclideanreal f (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof
    show connected_space X
      using L connected_space_subconnected by blast
    have connectedin X {x ∈ topspace X. f x ∈ {y}} for y
      by (metis L is_interval_1 nle_le singletonD)
    then show monotone_map X euclideanreal f
      by (simp add: monotone_map)
  qed
next
  assume R: ?rhs
  then
  have *: False
    if a < b closedin X U closedin X V U ≠ {} V ≠ {} disjnt U V
      and UV: {x ∈ topspace X. f x ∈ {a..b}} = U ∪ V
      and dis: disjnt U {x ∈ topspace X. f x = b} disjnt V {x ∈ topspace X. f x
= a}
    for a b U V
  proof –
    define E1 where E1 ≡ U ∪ {x ∈ topspace X. f x ∈ {c. c ≤ a}}
    define E2 where E2 ≡ V ∪ {x ∈ topspace X. f x ∈ {c. b ≤ c}}
    have closedin X {x ∈ topspace X. f x ≤ a} closedin X {x ∈ topspace X. b ≤ f
x}
      using assms continuous_map_upper_lower_semicontinuous_le by blast+

```



```

then have closedin X E1 closedin X E2
  unfolding E1_def E2_def using that by auto
moreover
have E1 ∩ E2 = {}
  unfolding E1_def E2_def using ‹a < b› ‹disjnt U V› dis UV
  by (simp add: disjnt_def set_eq_iff) (smt (verit))
have topspace X ⊆ E1 ∪ E2
  unfolding E1_def E2_def using UV by fastforce
have E1 = {} ∨ E2 = {}
  using R connected_space_closedin
  using ‹E1 ∩ E2 = {}› ‹closedin X E1› ‹closedin X E2› ‹topspace X ⊆ E1 ∪
E2› by blast
then show False
  using E1_def E2_def ‹U ≠ {}› ‹V ≠ {}› by fastforce
qed
show ?lhs
proof (intro strip)
  fix K :: real set
  assume is_interval K
  have False
    if a ∈ K b ∈ K and clo: closedin X U closedin X V
    and UV: {x. x ∈ topspace X ∧ f x ∈ K} ⊆ U ∪ V
    U ∩ V ∩ {x. x ∈ topspace X ∧ f x ∈ K} = {}
    and nondis: ¬ disjnt U {x. x ∈ topspace X ∧ f x = a}
    ¬ disjnt V {x. x ∈ topspace X ∧ f x = b}
  for a b U V
proof -
  have ∀ y. connectedin X {x. x ∈ topspace X ∧ f x = y}
    using R monotone_map by fastforce
  then have **: False if p ∈ U ∧ q ∈ V ∧ f p = f q ∧ f q ∈ K for p q
    unfolding connectedin_closedin
    using ‹a ∈ K› ‹b ∈ K› UV clo that
    by (smt (verit, ccfv_threshold) closedin_subset disjoint_iff mem_Collect_eq
subset_iff)
  consider a < b | a = b | b < a
  by linarith
  then show ?thesis
proof cases
  case 1
  define W where W ≡ {x ∈ topspace X. f x ∈ {a..b}}
  have closedin X W
    unfolding W_def
    by (metis (no_types) assms closed_real_atLeastAtMost closed_closedin
continuous_map_closedin)
  show ?thesis
proof (rule * [OF 1, of U ∩ W V ∩ W])
  show closedin X (U ∩ W) closedin X (V ∩ W)
    using ‹closedin X W› clo by auto
  show U ∩ W ≠ {} V ∩ W ≠ {}

```

```

    using nondis 1 by (auto simp: disjnt_iff W_def)
  show disjnt (U ∩ W) (V ∩ W)
    using ‹is_interval K› unfolding is_interval_1 disjnt_iff W_def
      by (metis (mono_tags, lifting) ‹a ∈ K› ‹b ∈ K› ** Int_Collect
atLeastAtMost_iff)
    have ∧x. [x ∈ topspace X; a ≤ f x; f x ≤ b] ⇒ x ∈ U ∨ x ∈ V
      using ‹a ∈ K› ‹b ∈ K› ‹is_interval K› UV unfolding is_interval_1
disjnt_iff
      by blast
    then show {x ∈ topspace X. f x ∈ {a..b}} = U ∩ W ∪ V ∩ W
      by (auto simp: W_def)
    show disjnt (U ∩ W) {x ∈ topspace X. f x = b} disjnt (V ∩ W) {x ∈
topspace X. f x = a}
      using ** ‹a ∈ K› ‹b ∈ K› nondis by (force simp: disjnt_iff)+
    qed
  next
  case 2
  then show ?thesis
    using ** nondis ‹b ∈ K› by (force simp add: disjnt_iff)
  next
  case 3
  define W where W ≡ {x ∈ topspace X. f x ∈ {b..a}}
  have closedin X W
    unfolding W_def
    by (metis (no_types) assms closed_real_atLeastAtMost closed_closedin
continuous_map_closedin)
  show ?thesis
  proof (rule * [OF 3, of V ∩ W U ∩ W])
    show closedin X (U ∩ W) closedin X (V ∩ W)
      using ‹closedin X W› clo by auto
    show U ∩ W ≠ {} V ∩ W ≠ {}
      using nondis 3 by (auto simp: disjnt_iff W_def)
    show disjnt (V ∩ W) (U ∩ W)
      using ‹is_interval K› unfolding is_interval_1 disjnt_iff W_def
      by (metis (mono_tags, lifting) ‹a ∈ K› ‹b ∈ K› ** Int_Collect
atLeastAtMost_iff)
    have ∧x. [x ∈ topspace X; b ≤ f x; f x ≤ a] ⇒ x ∈ U ∨ x ∈ V
      using ‹a ∈ K› ‹b ∈ K› ‹is_interval K› UV unfolding is_interval_1
disjnt_iff
      by blast
    then show {x ∈ topspace X. f x ∈ {b..a}} = V ∩ W ∪ U ∩ W
      by (auto simp: W_def)
    show disjnt (V ∩ W) {x ∈ topspace X. f x = a} disjnt (U ∩ W) {x ∈
topspace X. f x = b}
      using ** ‹a ∈ K› ‹b ∈ K› nondis by (force simp: disjnt_iff)+
    qed
  qed
  then show connectedin X {x ∈ topspace X. f x ∈ K}

```

```

    unfolding connectedin_closedin_disjnt_iff by blast
  qed
qed

lemma monotone_map_into_euclideanreal:
  [[connected_space  $X$ ; continuous_map  $X$  euclideanreal  $f$ ]]
   $\implies$  monotone_map  $X$  euclideanreal  $f$   $\longleftrightarrow$ 
    ( $\forall k. \text{is\_interval } k \longrightarrow \text{connectedin } X \{x \in \text{topspace } X. f x \in k\}$ )
  by (simp add: monotone_map_into_euclideanreal_alt)

lemma monotone_map_euclideanreal_alt:
  ( $\forall I::\text{real set. is\_interval } I \longrightarrow \text{is\_interval } \{x::\text{real. } x \in S \wedge f x \in I\}$ )  $\longleftrightarrow$ 
    is\_interval  $S \wedge (\text{mono\_on } S f \vee \text{antimono\_on } S f)$  (is ?lhs=?rhs)
proof
  assume  $L$  [rule_format]: ?lhs
  show ?rhs
  proof
    show is\_interval  $S$ 
    using  $L$  is\_interval_1 by auto
    have False if  $a \in S$   $b \in S$   $c \in S$   $a < b$   $b < c$  and  $d: f a < f b \wedge f c < f b \vee f a > f b \wedge f c > f b$  for  $a$   $b$   $c$ 
    using  $d$ 
    proof
      assume  $f a < f b \wedge f c < f b$ 
      then show False
      using  $L$  [of  $\{y. y < f b\}$ ] unfolding is\_interval_1
      by (smt (verit, best) mem_Collect_eq that)
    next
      assume  $f b < f a \wedge f b < f c$ 
      then show False
      using  $L$  [of  $\{y. y > f b\}$ ] unfolding is\_interval_1
      by (smt (verit, best) mem_Collect_eq that)
    qed
    then show mono\_on  $S f \vee \text{monotone\_on } S (\leq) (\geq) f$ 
    unfolding monotone_on_def by (smt (verit))
  qed
next
  assume ?rhs then show ?lhs
  unfolding is\_interval_1 monotone_on_def by (simp meson)
qed

```



```

lemma monotone_map_euclideanreal:
  fixes  $S :: \text{real set}$ 
  shows
    [[is\_interval  $S$ ; continuous_on  $S$   $f$ ]]  $\implies$ 
      monotone_map (top_of_set  $S$ ) euclideanreal  $f$   $\longleftrightarrow$  (mono\_on  $S f \vee \text{monotone\_on } S (\leq) (\geq) f$ )
    using monotone_map_euclideanreal_alt

```

by (*simp add: monotone_map_into_euclideanreal connectedin_subtopology is_interval_connected_1*)

lemma *injective_eq_monotone_map*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes *is_interval S continuous_on S f*

shows $\text{inj_on } f \ S \longleftrightarrow \text{strict_mono_on } S \ f \vee \text{strict_antimono_on } S \ f$

by (*metis assms injective_imp_monotone_map monotone_map_euclideanreal strict_antimono_iff_antimono*

strict_mono_iff_mono top_greatest topspace_euclidean topspace_euclidean_subtopology)

6.6.15 Normal spaces

definition *normal_space*

where *normal_space X* \equiv

$\forall S \ T. \text{closedin } X \ S \wedge \text{closedin } X \ T \wedge \text{disjnt } S \ T$

$\longrightarrow (\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$

$V)$

lemma *normal_space_retraction_map_image*:

assumes *r: retraction_map X Y r* **and** *X: normal_space X*

shows *normal_space Y*

unfolding *normal_space_def*

proof *clarify*

fix $S \ T$

assume *closedin Y S* **and** *closedin Y T* **and** *disjnt S T*

obtain r' **where** $r': \text{retraction_maps } X \ Y \ r \ r'$

using *r retraction_map_def* **by** *blast*

have *closedin X {x ∈ topspace X. r x ∈ S}* *closedin X {x ∈ topspace X. r x ∈ T}*

using *closedin_continuous_map_preimage* $\langle \text{closedin } Y \ S \rangle \langle \text{closedin } Y \ T \rangle r'$

by (*auto simp: retraction_maps_def*)

moreover

have *disjnt {x ∈ topspace X. r x ∈ S} {x ∈ topspace X. r x ∈ T}*

using $\langle \text{disjnt } S \ T \rangle$ **by** (*auto simp: disjnt_def*)

ultimately

obtain $U \ V$ **where** $UV: \text{openin } X \ U \wedge \text{openin } X \ V \wedge \{x \in \text{topspace } X. r \ x \in S\} \subseteq U \wedge \{x \in \text{topspace } X. r \ x \in T\} \subseteq V \text{ disjnt } U \ V$

by (*meson X normal_space_def*)

show $\exists U \ V. \text{openin } Y \ U \wedge \text{openin } Y \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U \ V$

proof (*intro exI conjI*)

show *openin Y {x ∈ topspace Y. r' x ∈ U}* *openin Y {x ∈ topspace Y. r' x ∈ V}*

using *openin_continuous_map_preimage* $UV \ r'$

by (*auto simp: retraction_maps_def*)

show $S \subseteq \{x \in \text{topspace } Y. r' \ x \in U\}$ $T \subseteq \{x \in \text{topspace } Y. r' \ x \in V\}$

using *openin_continuous_map_preimage* $UV \ r' \langle \text{closedin } Y \ S \rangle \langle \text{closedin } Y$

$T \rangle$

by (*auto simp add: closedin_def continuous_map_closedin retraction_maps_def subset_iff Pi_iff*)

```

show  $\text{disjnt } \{x \in \text{topspace } Y. r' x \in U\} \{x \in \text{topspace } Y. r' x \in V\}$ 
  using  $\langle \text{disjnt } U V \rangle$  by (auto simp: disjnt_def)
qed
qed

lemma homeomorphic_normal_space:
   $X \text{ homeomorphic\_space } Y \implies \text{normal\_space } X \longleftrightarrow \text{normal\_space } Y$ 
  unfolding homeomorphic_space_def
  by (meson homeomorphic_imp_retraction_maps homeomorphic_maps_sym normal_space_retraction_map_image retraction_map_def)

lemma normal_space:
   $\text{normal\_space } X \longleftrightarrow$ 
   $(\forall S T. \text{closedin } X S \wedge \text{closedin } X T \wedge \text{disjnt } S T$ 
     $\longrightarrow (\exists U. \text{openin } X U \wedge S \subseteq U \wedge \text{disjnt } T (X \text{ closure\_of } U)))$ 
proof –
  have  $(\exists V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U V) \longleftrightarrow$ 
   $\text{openin } X U \wedge S \subseteq U \wedge \text{disjnt } T (X \text{ closure\_of } U)$ 
  (is ?lhs=?rhs)
  if  $\text{closedin } X S \text{ closedin } X T \text{ disjnt } S T$  for  $S T U$ 
proof
  show  $?lhs \implies ?rhs$ 
  by (smt (verit, best) disjnt_iff_in_closure_of subsetD)
  assume  $R: ?rhs$ 
  then have  $(U \cup S) \cap (\text{topspace } X - X \text{ closure\_of } U) = \{\}$ 
  by (metis Diff_eq_empty_iff Int_Diff Int_Un_eq(4) closure_of_subset inf.orderE openin_subset)
  moreover have  $T \subseteq \text{topspace } X - X \text{ closure\_of } U$ 
  by (meson DiffI R closedin_subset disjnt_iff subsetD subsetI that(2))
  ultimately show  $?lhs$ 
  by (metis  $R \text{ closedin\_closure\_of closedin\_def disjnt\_def sup.orderE}$ )
qed
then show  $?thesis$ 
  unfolding normal_space_def by meson
qed

lemma normal_space_alt:
   $\text{normal\_space } X \longleftrightarrow$ 
   $(\forall S U. \text{closedin } X S \wedge \text{openin } X U \wedge S \subseteq U \longrightarrow (\exists V. \text{openin } X V \wedge S \subseteq V$ 
   $\wedge X \text{ closure\_of } V \subseteq U))$ 
proof –
  have  $\exists V. \text{openin } X V \wedge S \subseteq V \wedge X \text{ closure\_of } V \subseteq U$ 
  if  $\bigwedge T. \text{closedin } X T \longrightarrow \text{disjnt } S T \longrightarrow (\exists U. \text{openin } X U \wedge S \subseteq U \wedge \text{disjnt}$ 
   $T (X \text{ closure\_of } U))$ 
   $\text{closedin } X S \text{ openin } X U S \subseteq U$ 
  for  $S U$ 
  using that
  by (smt (verit) Diff_eq_empty_iff Int_Diff closure_of_subset_topspace disjnt_def inf.orderE inf_commute openin_closedin_eq)

```

moreover have $\exists U. \text{openin } X \ U \wedge S \subseteq U \wedge \text{disjnt } T \ (X \ \text{closure_of } U)$
if $\bigwedge U. \text{openin } X \ U \wedge S \subseteq U \longrightarrow (\exists V. \text{openin } X \ V \wedge S \subseteq V \wedge X \ \text{closure_of } V \subseteq U)$
and $\text{closedin } X \ S \ \text{closedin } X \ T \ \text{disjnt } S \ T$
for $S \ T$
using that
by (*smt (verit) Diff_Diff_Int Diff_eq_empty_iff Int_Diff closedin_def disjnt_def inf.absorb_iff2 inf.orderE*)
ultimately show *?thesis*
by (*fastforce simp: normal_space*)
qed

lemma *normal_space_closures:*

$\text{normal_space } X \longleftrightarrow$
 $(\forall S \ T. S \subseteq \text{topspace } X \wedge T \subseteq \text{topspace } X \wedge$
 $\text{disjnt } (X \ \text{closure_of } S) \ (X \ \text{closure_of } T)$
 $\longrightarrow (\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$
 $V))$
(is ?lhs=?rhs)

proof

show *?lhs \implies ?rhs*
by (*meson closedin_closure_of_closure_of_subset normal_space_def order.trans*)
show *?rhs \implies ?lhs*
by (*metis closedin_subset_closure_of_eq normal_space_def*)

qed

lemma *normal_space_disjoint_closures:*

$\text{normal_space } X \longleftrightarrow$
 $(\forall S \ T. \text{closedin } X \ S \wedge \text{closedin } X \ T \wedge \text{disjnt } S \ T$
 $\longrightarrow (\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge$
 $\text{disjnt } (X \ \text{closure_of } U) \ (X \ \text{closure_of } V)))$
(is ?lhs=?rhs)

proof

show *?lhs \implies ?rhs*
by (*metis closedin_closure_of normal_space*)
show *?rhs \implies ?lhs*
by (*smt (verit) closure_of_subset_disjnt_iff normal_space openin_subset subset_eq*)

qed

lemma *normal_space_dual:*

$\text{normal_space } X \longleftrightarrow$
 $(\forall U \ V. \text{openin } X \ U \longrightarrow \text{openin } X \ V \wedge U \cup V = \text{topspace } X$
 $\longrightarrow (\exists S \ T. \text{closedin } X \ S \wedge \text{closedin } X \ T \wedge S \subseteq U \wedge T \subseteq V \wedge S \cup T =$
 $\text{topspace } X))$
(is _ = ?rhs)

proof –

have $\text{normal_space } X \longleftrightarrow$
 $(\forall U \ V. \text{closedin } X \ U \longrightarrow \text{closedin } X \ V \longrightarrow \text{disjnt } U \ V \longrightarrow$

```

      (∃ S T. ¬ (openin X S ∧ openin X T →
        ¬ (U ⊆ S ∧ V ⊆ T ∧ disjnt S T))))
    unfolding normal_space_def by meson
  also have ... ↔ (∀ U V. openin X U → openin X V ∧ disjnt (topspace X -
    U) (topspace X - V) →
    (∃ S T. ¬ (openin X S ∧ openin X T →
      ¬ (topspace X - U ⊆ S ∧ topspace X - V ⊆ T ∧ disjnt S
    T))))
    by (auto simp: all_closedin)
  also have ... ↔ ?rhs
  proof -
    have *: disjnt (topspace X - U) (topspace X - V) ↔ U ∪ V = topspace X
      if U ⊆ topspace X V ⊆ topspace X for U V
      using that by (auto simp: disjnt_iff)
    show ?thesis
      using ex_closedin *
      apply (simp add: ex_closedin * [OF openin_subset openin_subset] cong:
        conj_cong)
      apply (intro all_cong1 ex_cong1 imp_cong refl)
      by (smt (verit, best) * Diff_Diff_Int Diff_subset Diff_subset_conv inf.orderE
        inf_commute openin_subset sup_commute)
    qed
  finally show ?thesis .
  qed

```

lemma *normal_t1_imp_Hausdorff_space:*

```

  assumes normal_space X t1_space X
  shows Hausdorff_space X
  unfolding Hausdorff_space_def
  proof clarify
    fix x y
    assume xy: x ∈ topspace X y ∈ topspace X x ≠ y
    then have disjnt {x} {y}
      by (auto simp: disjnt_iff)
    then show ∃ U V. openin X U ∧ openin X V ∧ x ∈ U ∧ y ∈ V ∧ disjnt U V
      using assms xy closedin_t1_singleton normal_space_def
      by (metis singletonI subsetD)
  qed

```

lemma *normal_t1_eq_Hausdorff_space:*

```

  normal_space X ⇒ t1_space X ↔ Hausdorff_space X
  using normal_t1_imp_Hausdorff_space t1_or_Hausdorff_space by blast

```

lemma *normal_t1_imp_regular_space:*

```

  [[normal_space X; t1_space X]] ⇒ regular_space X
  by (metis compactin_imp_closedin normal_space_def normal_t1_eq_Hausdorff_space
    regular_space_compact_closed_sets)

```

lemma *compact_Hausdorff_or_regular_imp_normal_space*:

$\llbracket \text{compact_space } X; \text{Hausdorff_space } X \vee \text{regular_space } X \rrbracket$
 $\implies \text{normal_space } X$

by (*metis Hausdorff_space_compact_sets closedin_compact_space normal_space_def regular_space_compact_closed_sets*)

lemma *normal_space_discrete_topology*:

$\text{normal_space}(\text{discrete_topology } U)$

by (*metis discrete_topology_closure_of_inf_le2 normal_space_alt*)

lemma *normal_space_fsigmas*:

$\text{normal_space } X \iff$

$(\forall S T. \text{fsigma_in } X S \wedge \text{fsigma_in } X T \wedge \text{separatedin } X S T$

$\longrightarrow (\exists U B. \text{openin } X U \wedge \text{openin } X B \wedge S \subseteq U \wedge T \subseteq B \wedge \text{disjnt } U$

$B))$ (**is** *?lhs=?rhs*)

proof

assume *L*: *?lhs*

show *?rhs*

proof *clarify*

fix *S T*

assume *fsigma_in X S*

then obtain *C* **where** $C: \bigwedge n. \text{closedin } X (C n) \wedge \bigwedge n. C n \subseteq C (Suc n) \cup$
 $(\text{range } C) = S$

by (*meson fsigma_in_ascending*)

assume *fsigma_in X T*

then obtain *D* **where** $D: \bigwedge n. \text{closedin } X (D n) \wedge \bigwedge n. D n \subseteq D (Suc n) \cup$
 $(\text{range } D) = T$

by (*meson fsigma_in_ascending*)

assume *separatedin X S T*

have $\bigwedge n. \text{disjnt } (D n) (X \text{ closure_of } S)$

by (*metis D(3) <separatedin X S T> disjnt_Union1 disjnt_def rangeI separatedin_def*)

then have $\bigwedge n. \exists V V'. \text{openin } X V \wedge \text{openin } X V' \wedge D n \subseteq V \wedge X \text{ closure_of } S \subseteq V' \wedge \text{disjnt } V V'$

by (*metis D(1) L closedin_closure_of_normal_space_def*)

then obtain *V V'* **where** $V: \bigwedge n. \text{openin } X (V n)$ **and** $\bigwedge n. \text{openin } X (V' n)$
 $\bigwedge n. \text{disjnt } (V n) (V' n)$

and $DV: \bigwedge n. D n \subseteq V n$

and $\text{sub}V': \bigwedge n. X \text{ closure_of } S \subseteq V' n$

by *metis*

then have $VV: V' n \cap X \text{ closure_of } V n = \{\}$ **for** *n*

using *openin_Int_closure_of_eq_empty* [*of X V' n V n*] **by** (*simp add: Int_commute disjnt_def*)

have $\bigwedge n. \text{disjnt } (C n) (X \text{ closure_of } T)$

by (*metis C(3) <separatedin X S T> disjnt_Union1 disjnt_def rangeI separatedin_def*)

then have $\bigwedge n. \exists U U'. \text{openin } X U \wedge \text{openin } X U' \wedge C n \subseteq U \wedge X \text{ closure_of } T \subseteq U' \wedge \text{disjnt } U U'$

by (*metis C(1) L closedin_closure_of_normal_space_def*)


```

then obtain  $U U'$  where  $U: \bigwedge n. \text{openin } X (U n)$  and  $\bigwedge n. \text{openin } X (U' n)$ 
 $\bigwedge n. \text{disjnt } (U n) (U' n)$ 
  and  $CU: \bigwedge n. C n \subseteq U n$ 
  and  $\text{sub}U': \bigwedge n. X \text{ closure\_of } T \subseteq U' n$ 
  by metis
then have  $UU: U' n \cap X \text{ closure\_of } U n = \{\}$  for  $n$ 
  using openin_Int_closure_of_eq_empty [of X U' n U n] by (simp add:
Int_commute disjnt_def)
show  $\exists U B. \text{openin } X U \wedge \text{openin } X B \wedge S \subseteq U \wedge T \subseteq B \wedge \text{disjnt } U B$ 
proof (intro conjI exI)
  have  $\bigwedge S n. \text{closedin } X (\bigcup_{m \leq n}. X \text{ closure\_of } V m)$ 
    by (force intro: closedin_Union)
  then show  $\text{openin } X (\bigcup n. U n - (\bigcup_{m \leq n}. X \text{ closure\_of } V m))$ 
    using  $U$  by blast
  have  $\bigwedge S n. \text{closedin } X (\bigcup_{m \leq n}. X \text{ closure\_of } U m)$ 
    by (force intro: closedin_Union)
  then show  $\text{openin } X (\bigcup n. V n - (\bigcup_{m \leq n}. X \text{ closure\_of } U m))$ 
    using  $V$  by blast
  have  $S \subseteq \text{topspace } X$ 
    by (simp add: fsigma_in X S fsigma_in_subset)
  then show  $S \subseteq (\bigcup n. U n - (\bigcup_{m \leq n}. X \text{ closure\_of } V m))$ 
    apply (clarsimp simp: Ball_def)
    by (metis VV C(3) CU IntI UN_E closure_of_subset empty_iff subV'
subsetD)
  have  $T \subseteq \text{topspace } X$ 
    by (simp add: fsigma_in X T fsigma_in_subset)
  then show  $T \subseteq (\bigcup n. V n - (\bigcup_{m \leq n}. X \text{ closure\_of } U m))$ 
    apply (clarsimp simp: Ball_def)
    by (metis UU D(3) DV IntI UN_E closure_of_subset empty_iff subU'
subsetD)
  have  $\bigwedge x m n. [x \in U n; x \in V m; \forall k \leq m. x \notin X \text{ closure\_of } U k] \implies \exists k \leq n. x \in X \text{ closure\_of } V k$ 
    by (meson U V closure_of_subset nat_le_linear openin_subset subsetD)
  then show  $\text{disjnt } (\bigcup n. U n - (\bigcup_{m \leq n}. X \text{ closure\_of } V m)) (\bigcup n. V n - (\bigcup_{m \leq n}. X \text{ closure\_of } U m))$ 
    by (force simp: disjnt_iff)
  qed
qed
next
  show  $?rhs \implies ?lhs$ 
    by (simp add: closed_imp_fsigma_in normal_space_def separatedin_closed_sets)
qed

lemma normal_space_fsigma_subtopology:
assumes normal_space X fsigma_in X S
shows normal_space (subtopology X S)
unfolding normal_space_fsigmas
proof clarify
  fix  $T U$ 

```

assume $f\sigma$ *in* (subtopology X S) T
and $f\sigma$ *in* (subtopology X S) U
and TU : *separatedin* (subtopology X S) T U
then obtain A B **where** $\text{openin } X$ $A \wedge \text{openin } X$ $B \wedge T \subseteq A \wedge U \subseteq B \wedge \text{disjnt } A$ B
by (*metis assms fsigma_in_fsigma_subtopology normal_space_fsigmas separatedin_subtopology*)
then
show $\exists A$ B . $\text{openin } (subtopology$ X $S)$ $A \wedge \text{openin } (subtopology$ X $S)$ $B \wedge T \subseteq A \wedge U \subseteq B \wedge \text{disjnt } A$ B
using TU
by (*force simp add: separatedin_subtopology openin_subtopology_alt disjnt_iff*)
qed

lemma *normal_space_closed_subtopology*:
assumes *normal_space* X *closedin* X S
shows *normal_space* (subtopology X S)
by (*simp add: assms closed_imp_fsigma_in normal_space_fsigma_subtopology*)

lemma *normal_space_continuous_closed_map_image*:
assumes *normal_space* X **and** *contf*: *continuous_map* X Y f
and *clof*: *closed_map* X Y f **and** *fm*: $f' \text{topspace } X = \text{topspace } Y$
shows *normal_space* Y
unfolding *normal_space_def*
proof *clarify*
fix S T
assume *closedin* Y S **and** *closedin* Y T **and** *disjnt* S T
have *closedin* X $\{x \in \text{topspace } X. f\ x \in S\}$ *closedin* X $\{x \in \text{topspace } X. f\ x \in T\}$
using $\langle \text{closedin } Y\ S \rangle \langle \text{closedin } Y\ T \rangle$ *closedin_continuous_map_preimage contf*
by *auto*
moreover
have *disjnt* $\{x \in \text{topspace } X. f\ x \in S\}$ $\{x \in \text{topspace } X. f\ x \in T\}$
using $\langle \text{disjnt } S\ T \rangle$ **by** (*auto simp: disjnt_iff*)
ultimately
obtain U V **where** *closedin* X U *closedin* X V
and *subXU*: $\{x \in \text{topspace } X. f\ x \in S\} \subseteq \text{topspace } X - U$
and *subXV*: $\{x \in \text{topspace } X. f\ x \in T\} \subseteq \text{topspace } X - V$
and *dis*: *disjnt* ($\text{topspace } X - U$) ($\text{topspace } X - V$)
using $\langle \text{normal_space } X \rangle$ **by** (*force simp add: normal_space_def ex_openin*)
have *closedin* Y ($f' U$) *closedin* Y ($f' V$)
using $\langle \text{closedin } X\ U \rangle \langle \text{closedin } X\ V \rangle$ *clof closed_map_def* **by** *blast+*
moreover **have** $S \subseteq \text{topspace } Y - f' U$
using $\langle \text{closedin } Y\ S \rangle \langle \text{closedin } X\ U \rangle$ *subXU* **by** (*force dest: closedin_subset*)
moreover **have** $T \subseteq \text{topspace } Y - f' V$
using $\langle \text{closedin } Y\ T \rangle \langle \text{closedin } X\ V \rangle$ *subXV* **by** (*force dest: closedin_subset*)
moreover **have** *disjnt* ($\text{topspace } Y - f' U$) ($\text{topspace } Y - f' V$)
using *fm dis* **by** (*force simp add: disjnt_iff*)

ultimately show $\exists U V. \text{openin } Y U \wedge \text{openin } Y V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U V$
 by (force simp add: ex_openin)
 qed

6.6.16 Hereditary topological properties

definition *hereditarily*

where *hereditarily* $P X \equiv$
 $\forall S. S \subseteq \text{topspace } X \longrightarrow P(\text{subtopology } X S)$

lemma *hereditarily*:

hereditarily $P X \longleftrightarrow (\forall S. P(\text{subtopology } X S))$
 by (metis Int_lower1 hereditarily_def subtopology_restrict)

lemma *hereditarily_mono*:

$\llbracket \text{hereditarily } P X; \bigwedge x. P x \implies Q x \rrbracket \implies \text{hereditarily } Q X$
 by (simp add: hereditarily)

lemma *hereditarily_inc*:

hereditarily $P X \implies P X$
 by (metis hereditarily_subtopology_topspace)

lemma *hereditarily_subtopology*:

hereditarily $P X \implies \text{hereditarily } P (\text{subtopology } X S)$
 by (simp add: hereditarily_subtopology_subtopology)

lemma *hereditarily_normal_space_continuous_closed_map_image*:

assumes $X: \text{hereditarily_normal_space } X$ **and** $\text{contf}: \text{continuous_map } X Y f$
and $\text{clof}: \text{closed_map } X Y f$ **and** $\text{fim}: f'(\text{topspace } X) = \text{topspace } Y$

shows *hereditarily normal space* Y

unfolding *hereditarily_def*

proof (intro strip)

fix T

assume $T \subseteq \text{topspace } Y$

then have $\text{nx}: \text{normal_space } (\text{subtopology } X \{x \in \text{topspace } X. f x \in T\})$

by (meson X hereditarily)

moreover have $\text{continuous_map } (\text{subtopology } X \{x \in \text{topspace } X. f x \in T\})$
 $(\text{subtopology } Y T) f$

by (simp add: contf continuous_map_from_subtopology continuous_map_in_subtopology
 image_subset_iff)

moreover have $\text{closed_map } (\text{subtopology } X \{x \in \text{topspace } X. f x \in T\}) (\text{subtopology } Y T) f$

by (simp add: clof closed_map_restriction)

ultimately show *normal space* $(\text{subtopology } Y T)$

using fim normal_space_continuous_closed_map_image by fastforce

qed

lemma *homeomorphic_hereditarily_normal_space*:

X *homeomorphic_space* Y
 \implies (*hereditarily_normal_space* $X \longleftrightarrow$ *hereditarily_normal_space* Y)
by (*meson hereditarily_normal_space_continuous_closed_map_image homeomorphic_eq_everything_map*
homeomorphic_space homeomorphic_space_sym)

lemma *hereditarily_normal_space_retraction_map_image*:
 \llbracket *retraction_map* $X Y r$; *hereditarily_normal_space* X $\rrbracket \implies$ *hereditarily_normal_space* Y
by (*smt (verit) hereditarily_subtopology hereditary_imp_retractive_property homeomorphic_hereditarily_normal_space*)

6.6.17 Limits in a topological space

lemma *limitin_const_iff*:
assumes *t1_space* $X \neg$ *trivial_limit* F
shows *limitin* $X (\lambda k. a) l F \longleftrightarrow l \in$ *topspace* $X \wedge a = l$ (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
using *assms unfolding limitin_def t1_space_def* **by** (*metis eventually_const openin_topospace*)
next
assume *?rhs* **then show** *?lhs*
using *assms* **by** (*auto simp: limitin_def t1_space_def*)
qed

lemma *compactin_sequence_with_limit*:
assumes *lim: limitin* $X \sigma l$ *sequentially* **and** $S \subseteq$ *range* σ **and** $SX: S \subseteq$ *topspace* X
shows *compactin* X (*insert* $l S$)
unfolding *compactin_def*
proof (*intro conjI strip*)
show *insert* $l S \subseteq$ *topspace* X
by (*meson SX insert_subset lim limitin_topospace*)
fix \mathcal{U}
assume $\S: \text{Ball } \mathcal{U} \text{ (openin } X) \wedge \text{insert } l S \subseteq \bigcup \mathcal{U}$
have $\exists V. \text{finite } V \wedge V \subseteq \mathcal{U} \wedge (\exists t \in V. l \in t) \wedge S \subseteq \bigcup V$
if $*$: $\forall x \in S. \exists T \in \mathcal{U}. x \in T$ **and** $T \in \mathcal{U} l \in T$ **for** T
proof –
obtain V **where** $V: \bigwedge x. x \in S \implies V x \in \mathcal{U} \wedge x \in V x$
using $*$ **by** *metis*
obtain N **where** $N: \bigwedge n. N \leq n \implies \sigma n \in T$
by (*meson* $\S \langle T \in \mathcal{U} \rangle \langle l \in T \rangle \text{lim limitin_sequentially}$)
show *?thesis*
proof (*intro conjI exI*)
have $x \in T$
if $x \in S$ **and** $\forall A. (\forall x \in S. (\forall n \leq N. x \neq \sigma n) \vee A \neq V x) \vee x \notin A$ **for** x
by (*metis (no_types) N V that assms(2) imageE nle_le subsetD*)
then show $S \subseteq \bigcup (\text{insert } T (V \text{ ' } (S \cap \sigma \text{ ' } \{0..N\})))$

```

    by force
  qed (use V that in auto)
qed
then show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \text{insert } l \ S \subseteq \bigcup \mathcal{F}$ 
  by (smt (verit, best) Union_iff § insert_subset subsetD)
qed

```

lemma *limitin_Hausdorff_unique*:

```

  assumes limitin X f l1 F limitin X f l2 F  $\neg$  trivial_limit F Hausdorff_space X
  shows  $l1 = l2$ 
proof (rule ccontr)
  assume  $l1 \neq l2$ 
  with assms obtain U V where openin X U openin X V  $l1 \in U$   $l2 \in V$  disjnt
  U V
  by (metis Hausdorff_space_def limitin_topspace)
  then have eventually  $(\lambda x. f\ x \in U)$  F eventually  $(\lambda x. f\ x \in V)$  F
  using assms by (fastforce simp: limitin_def)+
  then have  $\exists x. f\ x \in U \wedge f\ x \in V$ 
  using assms eventually_elim2 filter_eq_iff by fastforce
  with assms  $\langle \text{disjnt } U\ V \rangle$  show False
  by (meson disjnt_iff)
qed

```

lemma *limitin_kc_unique*:

```

  assumes kc_space X and lim1: limitin X f l1 sequentially and lim2: limitin X
  f l2 sequentially
  shows  $l1 = l2$ 
proof (rule ccontr)
  assume  $l1 \neq l2$ 
  define A where  $A \equiv \text{insert } l1\ (\text{range } f - \{l2\})$ 
  have  $l1 \in \text{topspace } X$ 
  using lim1 limitin_def by fastforce
  moreover have compactin X  $(\text{insert } l1\ (\text{topspace } X \cap (\text{range } f - \{l2\})))$ 
  by (meson Diff_subset compactin_sequence_with_limit_inf_le1_inf_le2 lim1
  subset_trans)
  ultimately have compactin X  $(\text{topspace } X \cap A)$ 
  by (simp add: A_def)
  then have OXA: openin X  $(\text{topspace } X - A)$ 
  by (metis Diff_Diff_Int Diff_subset  $\langle \text{kc\_space } X \rangle$  kc_space_def openin_closedin_eq)
  have  $l2 \in \text{topspace } X - A$ 
  using  $\langle l1 \neq l2 \rangle$  A_def lim2 limitin_topspace by fastforce
  then have  $\forall_F x$  in sequentially.  $f\ x = l2$ 
  using limitinD [OF lim2 OXA] by (auto simp: A_def eventually_conj_iff)
  then show False
  using limitin_transform_eventually [OF lim1]
  limitin_const_iff [OF kc_imp_t1_space trivial_limit_sequentially]
  using  $\langle l1 \neq l2 \rangle$   $\langle \text{kc\_space } X \rangle$  by fastforce
qed

```

lemma *limitin_closedin*:
assumes *lim*: *limitin* X f l F
and *closedin* X S **and** *ev*: *eventually* $(\lambda x. f\ x \in S)$ $F \neg$ *trivial_limit* F
shows $l \in S$
proof (*rule ccontr*)
assume $l \notin S$
have $\forall_F x$ *in* F . $f\ x \in$ *topspace* $X - S$
by (*metis Diff_iff* $\langle l \notin S \rangle$ \langle *closedin* X $S \rangle$ *closedin_def* *lim* *limitin_def*)
with *ev* *eventually_elim2* *trivial_limit_def* **show** *False*
by *force*
qed

6.6.18 Quasi-components

definition *quasi_component_of* :: $'a$ *topology* \Rightarrow $'a \Rightarrow 'a \Rightarrow$ *bool*
where
quasi_component_of X x $y \equiv$
 $x \in$ *topspace* $X \wedge y \in$ *topspace* $X \wedge$
 $(\forall T. \text{closedin } X\ T \wedge \text{openin } X\ T \longrightarrow (x \in T \longleftrightarrow y \in T))$

abbreviation *quasi_component_of_set* S $x \equiv$ *Collect* (*quasi_component_of* S x)

definition *quasi_components_of* :: $'a$ *topology* \Rightarrow $('a$ *set*) *set*
where
quasi_components_of $X =$ *quasi_component_of_set* X $'$ *topspace* X

lemma *quasi_component_in_topspace*:
quasi_component_of X x $y \Longrightarrow x \in$ *topspace* $X \wedge y \in$ *topspace* X
by (*simp* *add*: *quasi_component_of_def*)

lemma *quasi_component_of_refl* [*simp*]:
quasi_component_of X x $x \longleftrightarrow x \in$ *topspace* X
by (*simp* *add*: *quasi_component_of_def*)

lemma *quasi_component_of_sym*:
quasi_component_of X x $y \longleftrightarrow$ *quasi_component_of* X y x
by (*meson* *quasi_component_of_def*)

lemma *quasi_component_of_trans*:
 $[[$ *quasi_component_of* X x y ; *quasi_component_of* X y $z] \Longrightarrow$ *quasi_component_of*
 X x z
by (*simp* *add*: *quasi_component_of_def*)

lemma *quasi_component_of_subset_topspace*:
quasi_component_of_set X $x \subseteq$ *topspace* X
using *quasi_component_of_def* **by** *fastforce*

lemma *quasi_component_of_eq_empty*:
quasi_component_of_set X $x = \{\}$ $\longleftrightarrow (x \notin$ *topspace* $X)$

using *quasi_component_of_def* by *fastforce*

lemma *quasi_component_of*:

quasi_component_of X x y \longleftrightarrow
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge (\forall T. x \in T \wedge \text{closedin } X T \wedge \text{openin } X T \longrightarrow y \in T)$
unfolding *quasi_component_of_def* by (*metis Diff_iff closedin_def openin_closedin_eq*)

lemma *quasi_component_of_alt*:

quasi_component_of X x y \longleftrightarrow
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
 $\neg (\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge U \cup V = \text{topspace } X \wedge \text{disjnt } U V$
 $\wedge x \in U \wedge y \in V)$
(is ?lhs = ?rhs)

proof

show *?lhs* \implies *?rhs*

unfolding *quasi_component_of_def*

by (*metis disjnt_iff separatedin_full separatedin_open_sets*)

show *?rhs* \implies *?lhs*

unfolding *quasi_component_of_def*

by (*metis Diff_disjoint Diff_iff Un_Diff_cancel closedin_def disjnt_def inf_commute sup.orderE sup_commute*)

qed

lemma *quasi_components_lepoll_topspace*: *quasi_components_of* X \lesssim *topspace* X

by (*simp add: image_lepoll quasi_components_of_def*)

lemma *quasi_component_of_separated*:

quasi_component_of X x y \longleftrightarrow
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge$
 $\neg (\exists U V. \text{separatedin } X U V \wedge U \cup V = \text{topspace } X \wedge x \in U \wedge y \in V)$
by (*meson quasi_component_of_alt separatedin_full separatedin_open_sets*)

lemma *quasi_component_of_subtopology*:

quasi_component_of (*subtopology* X s) x y \implies *quasi_component_of* X x y

unfolding *quasi_component_of_def*

by (*simp add: closedin_subtopology*) (*metis Int_iff inf_commute openin_subtopology_Int2*)

lemma *quasi_component_of_mono*:

quasi_component_of (*subtopology* X S) x y $\wedge S \subseteq T$
 \implies *quasi_component_of* (*subtopology* X T) x y

by (*metis inf.absorb_iff2 quasi_component_of_subtopology subtopology_subtopology*)

lemma *quasi_component_of_equiv*:

quasi_component_of X x y \longleftrightarrow
 $x \in \text{topspace } X \wedge y \in \text{topspace } X \wedge \text{quasi_component_of } X x = \text{quasi_component_of } X y$

1300

using *quasi_component_of_def* **by** *fastforce*

lemma *quasi_component_of_disjoint* [*simp*]:

$disjnt (quasi_component_of_set\ X\ x) (quasi_component_of_set\ X\ y) \longleftrightarrow \neg (quasi_component_of\ X\ x\ y)$

by (*metis disjnt_iff quasi_component_of_equiv mem_Collect_eq*)

lemma *quasi_component_of_eq*:

$quasi_component_of\ X\ x = quasi_component_of\ X\ y \longleftrightarrow (x \notin topspace\ X \wedge y \notin topspace\ X)$

$\vee x \in topspace\ X \wedge y \in topspace\ X \wedge quasi_component_of\ X\ x\ y$

by (*metis Collect_empty_eq_bot quasi_component_of_eq_empty quasi_component_of_equiv*)

lemma *topspace_imp_quasi_components_of*:

assumes $x \in topspace\ X$

obtains C **where** $C \in quasi_components_of\ X$ $x \in C$

by (*metis assms imageI mem_Collect_eq quasi_component_of_refl quasi_components_of_def*)

lemma *Union_quasi_components_of*: $\bigcup (quasi_components_of\ X) = topspace\ X$

by (*auto simp: quasi_components_of_def quasi_component_of_def*)

lemma *pairwise_disjoint_quasi_components_of*:

$pairwise\ disjnt (quasi_components_of\ X)$

by (*auto simp: quasi_components_of_def quasi_component_of_def disjoint_def*)

lemma *complement_quasi_components_of_Union*:

assumes $C \in quasi_components_of\ X$

shows $topspace\ X - C = \bigcup (quasi_components_of\ X - \{C\})$ (**is** $?lhs = ?rhs$)

proof

show $?lhs \subseteq ?rhs$

using *Union_quasi_components_of* **by** *fastforce*

show $?rhs \subseteq ?lhs$

using *assms*

using *quasi_component_of_equiv* **by** (*fastforce simp add: quasi_components_of_def image_iff subset_iff*)

qed

lemma *nonempty_quasi_components_of*:

$C \in quasi_components_of\ X \implies C \neq \{\}$

by (*metis imageE quasi_component_of_eq_empty quasi_components_of_def*)

lemma *quasi_components_of_subset*:

$C \in quasi_components_of\ X \implies C \subseteq topspace\ X$

using *Union_quasi_components_of* **by** *force*

lemma *quasi_component_in_quasi_components_of*:

$quasi_component_of_set\ X\ a \in quasi_components_of\ X \longleftrightarrow a \in topspace\ X$

by (*metis (no_types, lifting) image_iff quasi_component_of_eq_empty quasi_components_of_def*)

lemma *quasi_components_of_eq_empty* [simp]:
 $quasi_components_of\ X = \{\} \longleftrightarrow X = trivial_topology$
by (simp add: quasi_components_of_def)

lemma *quasi_components_of_empty_space* [simp]:
 $quasi_components_of\ trivial_topology = \{\}$
by simp

lemma *quasi_component_of_set*:
 $quasi_component_of_set\ X\ x =$
 (if $x \in topspace\ X$
 then $\bigcap \{t. closedin\ X\ t \wedge openin\ X\ t \wedge x \in t\}$
 else $\{\}$)
by (auto simp: quasi_component_of)

lemma *closedin_quasi_component_of*: $closedin\ X\ (quasi_component_of_set\ X\ x)$
by (auto simp: quasi_component_of_set)

lemma *closedin_quasi_components_of*:
 $C \in quasi_components_of\ X \implies closedin\ X\ C$
by (auto simp: quasi_components_of_def closedin_quasi_component_of)

lemma *openin_finite_quasi_components*:
 $\llbracket finite(quasi_components_of\ X); C \in quasi_components_of\ X \rrbracket \implies openin\ X\ C$
apply (simp add: openin_closedin_eq quasi_components_of_subset complement_quasi_components_of_Union)
by (meson DiffD1 closedin_Union closedin_quasi_components_of_finite_Diff)

lemma *quasi_component_of_eq_overlap*:
 $quasi_component_of\ X\ x = quasi_component_of\ X\ y \longleftrightarrow$
 $(x \notin topspace\ X \wedge y \notin topspace\ X) \vee$
 $\neg (quasi_component_of_set\ X\ x \cap quasi_component_of_set\ X\ y = \{\})$
using quasi_component_of_equiv **by** fastforce

lemma *quasi_component_of_nonoverlap*:
 $quasi_component_of_set\ X\ x \cap quasi_component_of_set\ X\ y = \{\} \longleftrightarrow$
 $(x \notin topspace\ X) \vee (y \notin topspace\ X) \vee$
 $\neg (quasi_component_of\ X\ x = quasi_component_of\ X\ y)$
by (metis inf.idem quasi_component_of_eq_empty quasi_component_of_eq_overlap)

lemma *quasi_component_of_overlap*:
 $\neg (quasi_component_of_set\ X\ x \cap quasi_component_of_set\ X\ y = \{\}) \longleftrightarrow$
 $x \in topspace\ X \wedge y \in topspace\ X \wedge quasi_component_of\ X\ x = quasi_component_of\ X\ y$
by (meson quasi_component_of_nonoverlap)

lemma *quasi_components_of_disjoint*:
 $\llbracket C \in quasi_components_of\ X; D \in quasi_components_of\ X \rrbracket \implies disjoint\ C\ D$

1302

$\longleftrightarrow C \neq D$

by (*metis disjnt_self_iff_empty nonempty_quasi_components_of pairwiseD pairwise_disjoint_quasi_components_of*)

lemma *quasi_components_of_overlap*:

$\llbracket C \in \text{quasi_components_of } X; D \in \text{quasi_components_of } X \rrbracket \implies \neg (C \cap D = \{\}) \longleftrightarrow C = D$

by (*metis disjnt_def quasi_components_of_disjoint*)

lemma *pairwise_separated_quasi_components_of*:

pairwise (separatedin X) (quasi_components_of X)

by (*metis closedin_quasi_components_of pairwise_def pairwise_disjoint_quasi_components_of separatedin_closed_sets*)

lemma *finite_quasi_components_of_finite*:

finite (topspace X) \implies finite (quasi_components_of X)

by (*simp add: Union_quasi_components_of_finite_UnionD*)

lemma *connected_imp_quasi_component_of*:

assumes *connected_component_of X x y*

shows *quasi_component_of X x y*

proof –

have $x \in \text{topspace } X \ y \in \text{topspace } X$

by (*meson assms connected_component_of_equiv*)+

with *assms show ?thesis*

apply (*clarsimp simp add: quasi_component_of_connected_component_of_def*)

by (*meson connectedin_clopen_cases disjnt_iff subsetD*)

qed

lemma *connected_component_subset_quasi_component_of*:

connected_component_of_set X x \subseteq quasi_component_of_set X x

using *connected_imp_quasi_component_of* **by** *force*

lemma *quasi_component_as_connected_component_Union*:

quasi_component_of_set X x =

$\bigcup (\text{connected_component_of_set } X \text{ ` } \text{quasi_component_of_set } X \text{ } x)$

(*is ?lhs = ?rhs*)

proof

show $?lhs \subseteq ?rhs$

using *connected_component_of_refl quasi_component_of* **by** *fastforce*

show $?rhs \subseteq ?lhs$

apply (*rule SUP_least*)

by (*simp add: connected_component_subset_quasi_component_of quasi_component_of_equiv*)

qed

lemma *quasi_components_as_connected_components_Union*:

assumes $C \in \text{quasi_components_of } X$

obtains \mathcal{T} **where** $\mathcal{T} \subseteq \text{connected_components_of } X \ \bigcup \mathcal{T} = C$

proof –

```

obtain  $x$  where  $x \in \text{topspace } X$  and  $Ceq: C = \text{quasi\_component\_of\_set } X \ x$ 
  by (metis assms imageE quasi\_components\_of\_def)
define  $\mathcal{T}$  where  $\mathcal{T} \equiv \text{connected\_component\_of\_set } X \ ' \text{quasi\_component\_of\_set}$ 
 $X \ x$ 
show thesis
proof
  show  $\mathcal{T} \subseteq \text{connected\_components\_of } X$ 
  by (simp add:  $\mathcal{T}$ _def connected\_components\_of\_def image_mono quasi\_component\_of\_subset\_topspace)
  show  $\bigcup \mathcal{T} = C$ 
  by (metis  $\mathcal{T}$ _def Ceq quasi\_component\_as\_connected\_component\_Union)
qed
qed

```

```

lemma path_imp_quasi_component_of:
  path_component_of  $X \ x \ y \implies \text{quasi\_component\_of } X \ x \ y$ 
by (simp add: connected_imp_quasi_component_of path_imp_connected_component_of)

```

```

lemma path_component_subset_quasi_component_of:
  path_component_of_set  $X \ x \subseteq \text{quasi\_component\_of\_set } X \ x$ 
by (simp add: Collect_mono path_imp_quasi_component_of)

```

```

lemma connected_space_iff_quasi_component:
  connected_space  $X \longleftrightarrow (\forall x \in \text{topspace } X. \forall y \in \text{topspace } X. \text{quasi\_component\_of}$ 
 $X \ x \ y)$ 
unfolding connected_space_clopen_in_closedin_def quasi_component_of
by blast

```

```

lemma connected_space_imp_quasi_component_of:
   $\llbracket \text{connected\_space } X; a \in \text{topspace } X; b \in \text{topspace } X \rrbracket \implies \text{quasi\_component\_of}$ 
 $X \ a \ b$ 
by (simp add: connected_space_iff_quasi_component)

```

```

lemma connected_space_quasi_component_set:
  connected_space  $X \longleftrightarrow (\forall x \in \text{topspace } X. \text{quasi\_component\_of\_set } X \ x =$ 
 $\text{topspace } X)$ 
by (metis Ball_Collect connected_space_iff_quasi_component quasi_component_of_subset_topspace
subset_antisym)

```

```

lemma connected_space_iff_quasi_components_eq:
  connected_space  $X \longleftrightarrow$ 
   $(\forall C \in \text{quasi\_components\_of } X. \forall D \in \text{quasi\_components\_of } X. C = D)$ 
apply (simp add: quasi_components_of_def)
by (metis connected_space_iff_quasi_component mem_Collect_eq quasi_component_of_equiv)

```

```

lemma quasi_components_of_subset_sing:
  quasi_components_of  $X \subseteq \{S\} \longleftrightarrow \text{connected\_space } X \wedge (X = \text{trivial\_topology}$ 
 $\vee \text{topspace } X = S)$ 
proof (cases quasi_components_of  $X = \{\}$ )
  case True

```

```

then show ?thesis
  by (simp add: subset_singleton_iff)
next
  case False
  then show ?thesis
  apply (simp add: connected_space_iff_quasi_components_eq_subset_iff Ball_def)
  by (metis False Union_quasi_components_of_ccpo_Sup_singleton insert_iff
is_singletonE is_singletonI)
qed

```

```

lemma connected_space_iff_quasi_components_subset_sing:
  connected_space X  $\longleftrightarrow$  ( $\exists a$ . quasi_components_of X  $\subseteq$  {a})
  by (simp add: quasi_components_of_subset_sing)

```

```

lemma quasi_components_of_eq_singleton:
  quasi_components_of X = {S}  $\longleftrightarrow$ 
  connected_space X  $\wedge$   $\neg$  (X = trivial_topology)  $\wedge$  S = topspace X
  by (metis empty_not_insert quasi_components_of_eq_empty quasi_components_of_subset_sing
subset_singleton_iff)

```

```

lemma quasi_components_of_connected_space:
  connected_space X
   $\implies$  quasi_components_of X = (if X = trivial_topology then {} else {topspace X})
  by (simp add: quasi_components_of_eq_singleton)

```

```

lemma separated_between_singletons:
  separated_between X {x} {y}  $\longleftrightarrow$ 
  x  $\in$  topspace X  $\wedge$  y  $\in$  topspace X  $\wedge$   $\neg$  (quasi_component_of X x y)
proof (cases x  $\in$  topspace X  $\wedge$  y  $\in$  topspace X)
  case True
  then show ?thesis
  by (auto simp add: separated_between_def quasi_component_of_alt)
qed (use separated_between_imp_subset in blast)

```

```

lemma quasi_component_nonseparated:
  quasi_component_of X x y  $\longleftrightarrow$  x  $\in$  topspace X  $\wedge$  y  $\in$  topspace X  $\wedge$   $\neg$  (separated_between
X {x} {y})
  by (metis quasi_component_of_equiv separated_between_singletons)

```

```

lemma separated_between_quasi_component_pointwise_left:
  assumes C  $\in$  quasi_components_of X
  shows separated_between X C S  $\longleftrightarrow$  ( $\exists x \in C$ . separated_between X {x} S) (is
?lhs = ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
  using assms quasi_components_of_disjoint separated_between_mono by fast-
force
next

```

```

assume ?rhs
then obtain  $y$  where separated_between  $X \{y\} S$  and  $y \in C$ 
by metis
with assms show ?lhs
by (force simp add: separated_between_quasi_components_of_def quasi_component_of_def)
qed

```

```

lemma separated_between_quasi_component_pointwise_right:
 $C \in \text{quasi\_components\_of } X \implies \text{separated\_between } X S C \longleftrightarrow (\exists x \in C. \text{separated\_between } X S \{x\})$ 
by (simp add: separated_between_quasi_component_pointwise_left separated_between_sym)

```

```

lemma separated_between_quasi_component_point:
assumes  $C \in \text{quasi\_components\_of } X$ 
shows separated_between  $X C \{x\} \longleftrightarrow x \in \text{topspace } X - C$  (is ?lhs = ?rhs)
proof
show ?lhs  $\implies$  ?rhs
by (meson DiffI disjnt_insert2 insert_subset separated_between_imp_disjoint separated_between_imp_subset)
next
assume ?rhs
with assms show ?lhs
unfolding quasi_components_of_def image_iff Diff_iff separated_between_quasi_component_pointwise_left
[OF assms]
by (metis mem_Collect_eq quasi_component_of_refl separated_between_singletons)
qed

```

```

lemma separated_between_point_quasi_component:
 $C \in \text{quasi\_components\_of } X \implies \text{separated\_between } X \{x\} C \longleftrightarrow x \in \text{topspace } X - C$ 
by (simp add: separated_between_quasi_component_point separated_between_sym)

```

```

lemma separated_between_quasi_component_compact:
 $\llbracket C \in \text{quasi\_components\_of } X; \text{compactin } X K \rrbracket \implies (\text{separated\_between } X C K \longleftrightarrow \text{disjnt } C K)$ 
unfolding disjnt_iff
using compactin_subset_topspace quasi_components_of_subset separated_between_pointwise_right separated_between_quasi_component_point by fastforce

```

```

lemma separated_between_compact_quasi_component:
 $\llbracket \text{compactin } X K; C \in \text{quasi\_components\_of } X \rrbracket \implies \text{separated\_between } X K C \longleftrightarrow \text{disjnt } K C$ 
using disjnt_sym separated_between_quasi_component_compact separated_between_sym
by blast

```

```

lemma separated_between_quasi_components:
assumes  $C: C \in \text{quasi\_components\_of } X$  and  $D: D \in \text{quasi\_components\_of } X$ 
shows separated_between  $X C D \longleftrightarrow \text{disjnt } C D$  (is ?lhs = ?rhs)
proof

```

```

show ?lhs  $\implies$  ?rhs
  by (simp add: separated_between_imp_disjoint)
next
assume ?rhs
obtain x y where x: C = quasi_component_of_set X x and x  $\in$  C
  and y: D = quasi_component_of_set X y and y  $\in$  D
  using assms by (auto simp: quasi_components_of_def)
then have separated_between X {x} {y}
  using <disjnt C D> separated_between_singletons by fastforce
with <x  $\in$  C> <y  $\in$  D> show ?lhs
  by (auto simp: assms separated_between_quasi_component_pointwise_left_separated_between_quasi_component_pointwise_right)
qed

```

```

lemma quasi_eq_connected_component_of_eq:
  quasi_component_of X x = connected_component_of X x  $\longleftrightarrow$ 
  connectedin X (quasi_component_of_set X x) (is ?lhs = ?rhs)
proof (cases x  $\in$  topspace X)
case True
show ?thesis
proof
show ?lhs  $\implies$  ?rhs
  by (simp add: connectedin_connected_component_of)
next
assume ?rhs
then have  $\bigwedge y$ . quasi_component_of X x y = connected_component_of X x y
  by (metis connected_component_of_def connected_imp_quasi_component_of
  mem_Collect_eq quasi_component_of_equiv)
then show ?lhs
  by force
qed
next
case False
then show ?thesis
  by (metis Collect_empty_eq_bot connected_component_of_eq_empty connectedin_empty_quasi_component_of_eq_empty)
qed

```

```

lemma connected_quasi_component_of:
assumes C  $\in$  quasi_components_of X
shows C  $\in$  connected_components_of X  $\longleftrightarrow$  connectedin X C (is ?lhs = ?rhs)
proof
show ?lhs  $\implies$  ?rhs
  using assms
  by (simp add: connectedin_connected_components_of)
next
assume ?rhs
with assms show ?lhs
  unfolding quasi_components_of_def connected_components_of_def image_iff

```

by (metis quasi_eq_connected_component_of_eq)
qed

lemma quasi_component_of_clopen_cases:

$\llbracket C \in \text{quasi_components_of } X; \text{closedin } X T; \text{openin } X T \rrbracket \implies C \subseteq T \vee \text{disjnt } C T$

by (smt (verit) disjnt_iff_image_iff mem_Collect_eq quasi_component_of_def quasi_components_of_def subset_iff)

lemma quasi_components_of_set:

assumes $C \in \text{quasi_components_of } X$

shows $\bigcap \{T. \text{closedin } X T \wedge \text{openin } X T \wedge C \subseteq T\} = C$ (is ?lhs = ?rhs)

proof

have $x \in C$ if $x \in \bigcap \{T. \text{closedin } X T \wedge \text{openin } X T \wedge C \subseteq T\}$ for x

proof (rule ccontr)

assume $x \notin C$

have $x \in \text{topspace } X$

using assms quasi_components_of_subset that by force

then have separated_between $X C \{x\}$

by (simp add: $\langle x \notin C \rangle$ assms separated_between_quasi_component_point)

with that show False

by (auto simp: separated_between)

qed

then show ?lhs \subseteq ?rhs

by auto

qed blast

lemma open_quasi_eq_connected_components_of:

assumes $\text{openin } X C$

shows $C \in \text{quasi_components_of } X \iff C \in \text{connected_components_of } X$ (is ?lhs = ?rhs)

proof (cases closedin $X C$)

case True

show ?thesis

proof

assume $L: ?lhs$

have $T = \{\} \vee T = \text{topspace } X \cap C$

if $\text{openin } (\text{subtopology } X C) T \text{ closedin } (\text{subtopology } X C) T$ for T

proof –

have $C \subseteq T \vee \text{disjnt } C T$

by (meson L True assms closedin_trans_full openin_trans_full quasi_component_of_clopen_cases that)

with that show ?thesis

by (metis Int_absorb2 True closedin_imp_subset closure_of_subset_eq disjnt_def inf_absorb2)

qed

with L assms show ?rhs

by (simp add: connected_quasi_component_of_connected_space_clopen_in connectedin_def openin_subset)

```

next
  assume ?rhs
  then obtain x where x ∈ topspace X and x: C = connected_component_of_set
X x
    by (metis connected_components_of_def imageE)
    have C = quasi_component_of_set X x
    using True assms connected_component_of_refl connected_imp_quasi_component_of
quasi_component_of_def x by fastforce
    then show ?lhs
      using ⟨x ∈ topspace X⟩ quasi_components_of_def by fastforce
    qed
next
  case False
  then show ?thesis
    using closedin_connected_components_of closedin_quasi_components_of by
blast
  qed

```

```

lemma quasi_component_of_continuous_image:
  assumes f: continuous_map X Y f and qc: quasi_component_of X x y
  shows quasi_component_of Y (f x) (f y)
  unfolding quasi_component_of_def
proof (intro strip conjI)
  show f x ∈ topspace Y f y ∈ topspace Y
    using assms by (simp_all add: continuous_map_def quasi_component_of_def
Pi_iff)
  fix T
  assume closedin Y T ∧ openin Y T
  with assms show (f x ∈ T) = (f y ∈ T)
    by (smt (verit) continuous_map_closedin continuous_map_def mem_Collect_eq
quasi_component_of_def)
  qed

```

```

lemma quasi_component_of_discrete_topology:
  quasi_component_of_set (discrete_topology U) x = (if x ∈ U then {x} else {})
proof -
  have quasi_component_of_set (discrete_topology U) y = {y} if y ∈ U for y
  using that
  apply (simp add: set_eq_iff quasi_component_of_def)
  by (metis Set.set_insert insertE subset_insertI)
  then show ?thesis
    by (simp add: quasi_component_of)
  qed

```

```

lemma quasi_components_of_discrete_topology:
  quasi_components_of (discrete_topology U) = (λx. {x}) ‘ U
  by (auto simp add: quasi_components_of_def quasi_component_of_discrete_topology)

```

```

lemma homeomorphic_map_quasi_component_of:

```



```

assumes hmf: homeomorphic_map X Y f and  $x \in \text{topspace } X$ 
shows quasi_component_of_set Y (f x) = f ' (quasi_component_of_set X x)
proof –
  obtain g where hmg: homeomorphic_map Y X g
    and contf: continuous_map X Y f and contg: continuous_map Y X g
    and fg: ( $\forall x \in \text{topspace } X. g(f\ x) = x$ )  $\wedge$  ( $\forall y \in \text{topspace } Y. f(g\ y) = y$ )
    by (smt (verit, best) hmf homeomorphic_map_maps homeomorphic_maps_def)
  show ?thesis
proof
  show quasi_component_of_set Y (f x)  $\subseteq$  f ' quasi_component_of_set X x
    using quasi_component_of_continuous_image [OF contg]
     $\langle x \in \text{topspace } X \rangle$  fg image_iff quasi_component_of_subset_topspace by
fastforce
  show f ' quasi_component_of_set X x  $\subseteq$  quasi_component_of_set Y (f x)
    using quasi_component_of_continuous_image [OF contf] by blast
qed
qed

```

lemma *homeomorphic_map_quasi_components_of*:

```

assumes homeomorphic_map X Y f
shows quasi_components_of Y = image (image f) (quasi_components_of X)
using assms
proof –
  have  $\exists x \in \text{topspace } X. \text{quasi\_component\_of\_set } Y\ y = f\ ' \text{quasi\_component\_of\_set } X\ x$ 
    if  $y \in \text{topspace } Y$  for y
    by (metis that assms homeomorphic_imp_surjective_map homeomorphic_map_quasi_component_of_image_iff)
  moreover have  $\exists x \in \text{topspace } Y. f\ ' \text{quasi\_component\_of\_set } X\ u = \text{quasi\_component\_of\_set } Y\ x$ 
    if  $u \in \text{topspace } X$  for u
    by (metis that assms homeomorphic_imp_surjective_map homeomorphic_map_quasi_component_of_imageI)
  ultimately show ?thesis
    by (auto simp: quasi_components_of_def image_iff)
qed

```

lemma *openin_quasi_component_of_locally_connected_space*:

```

assumes locally_connected_space X
shows openin X (quasi_component_of_set X x)
proof –
  have *: openin X (connected_component_of_set X x)
    by (simp add: assms openin_connected_component_of_locally_connected_space)
  moreover have connected_component_of_set X x = quasi_component_of_set X x
    using * closedin_connected_component_of connected_component_of_refl connected_imp_quasi_component_of
    quasi_component_of_def by fastforce

```

1310

ultimately show *?thesis*
by *simp*
qed

lemma *openin_quasi_components_of_locally_connected_space:*
locally_connected_space X \wedge *c* \in *quasi_components_of X*
 \implies *openin X c*
by (*smt* (*verit*, *best*) *image_iff_openin_quasi_component_of_locally_connected_space*
quasi_components_of_def)

lemma *quasi_eq_connected_components_of_alt:*
quasi_components_of X = *connected_components_of X* \longleftrightarrow ($\forall C \in$ *quasi_components_of*
X. connectedin X C)
(**is** *?lhs* = *?rhs*)

proof
assume *R: ?rhs*
moreover have *connected_components_of X* \subseteq *quasi_components_of X*
using *R unfolding quasi_components_of_def connected_components_of_def*
by (*force simp flip: quasi_eq_connected_component_of_eq*)
ultimately show *?lhs*
using *connected_quasi_component_of* **by** *blast*
qed (*use connected_quasi_component_of in blast*)

lemma *connected_subset_quasi_components_of_pointwise:*
connected_components_of X \subseteq *quasi_components_of X* \longleftrightarrow
($\forall x \in$ *topspace X. quasi_component_of X x* = *connected_component_of X x*)
(**is** *?lhs* = *?rhs*)

proof
assume *L: ?lhs*
have *connectedin X (quasi_component_of_set X x)* **if** *x* \in *topspace X* **for** *x*
proof –
have $\exists y \in$ *topspace X. connected_component_of_set X x* = *quasi_component_of_set*
X y
using *L that by (force simp: quasi_components_of_def connected_components_of_def*
image_subset_iff)
then show *?thesis*
by (*metis connected_component_of_equiv connectedin_connected_component_of*
mem_Collect_eq quasi_component_of_eq)
qed
then show *?rhs*
by (*simp add: quasi_eq_connected_component_of_eq*)
qed (*simp add: connected_components_of_def quasi_components_of_def*)

lemma *quasi_subset_connected_components_of_pointwise:*
quasi_components_of X \subseteq *connected_components_of X* \longleftrightarrow
($\forall x \in$ *topspace X. quasi_component_of X x* = *connected_component_of X x*)
by (*simp add: connected_quasi_component_of_image_subset_iff quasi_components_of_def*
quasi_eq_connected_component_of_eq)

lemma *quasi_eq_connected_components_of_pointwise*:

quasi_components_of X = connected_components_of X \longleftrightarrow

($\forall x \in \text{topspace } X. \text{quasi_component_of } X \ x = \text{connected_component_of } X \ x$)

using *connected_subset_quasi_components_of_pointwise* *quasi_subset_connected_components_of_pointwise*
by *fastforce*

lemma *quasi_eq_connected_components_of_pointwise_alt*:

quasi_components_of X = connected_components_of X \longleftrightarrow

($\forall x. \text{quasi_component_of } X \ x = \text{connected_component_of } X \ x$)

unfolding *quasi_eq_connected_components_of_pointwise*

by (*metis* *connectedin_empty* *quasi_component_of_eq_empty* *quasi_eq_connected_component_of_eq*)

lemma *quasi_eq_connected_components_of_inclusion*:

quasi_components_of X = connected_components_of X \longleftrightarrow

connected_components_of X \subseteq *quasi_components_of X* \vee

quasi_components_of X \subseteq *connected_components_of X*

by (*simp* *add*: *connected_subset_quasi_components_of_pointwise* *dual_order.eq_iff* *quasi_subset_connected_components_of_pointwise*)

lemma *quasi_eq_connected_components_of*:

finite(connected_components_of X) \vee

finite(quasi_components_of X) \vee

locally_connected_space X \vee

compact_space X \wedge (*Hausdorff_space X* \vee *regular_space X* \vee *normal_space*

X)

\implies *quasi_components_of X = connected_components_of X*

proof (*elim* *disjE*)

show *quasi_components_of X = connected_components_of X*

if *finite(connected_components_of X)*

unfolding *quasi_eq_connected_components_of_inclusion*

using *that* *open_in_finite_connected_components* *open_quasi_eq_connected_components_of*

by *blast*

show *quasi_components_of X = connected_components_of X*

if *finite(quasi_components_of X)*

unfolding *quasi_eq_connected_components_of_inclusion*

using *that* *open_quasi_eq_connected_components_of* *openin_finite_quasi_components*

by *blast*

show *quasi_components_of X = connected_components_of X*

if *locally_connected_space X*

unfolding *quasi_eq_connected_components_of_inclusion*

using *that* *open_quasi_eq_connected_components_of* *openin_quasi_components_of_locally_connected_space*

by *auto*

show *quasi_components_of X = connected_components_of X*

if *compact_space X* \wedge (*Hausdorff_space X* \vee *regular_space X* \vee *normal_space*

X)

proof –

show *?thesis*

unfolding *quasi_eq_connected_components_of_alt*

```

proof (intro strip)
  fix C
  assume C: C ∈ quasi_components_of X
  then have cloC: closedin X C
    by (simp add: closedin_quasi_components_of)
  have normal_space X
    using that compact_Hausdorff_or_regular_imp_normal_space by blast
  show connectedin X C
    proof (clarsimp simp add: connectedin_def connected_space_closedin_eq
closedin_closed_subtopology cloC closedin_subset [OF cloC])
      fix S T
      assume S ⊆ C and closedin X S and S ∩ T = {} and SUT: S ∪ T =
topspace X ∩ C
      and T: T ⊆ C T ≠ {} and closedin X T
      with ⟨normal_space X⟩ obtain U V where UV: openin X U openin X V
S ⊆ U T ⊆ V disjnt U V
      by (meson disjnt_def normal_space_def)
      moreover have compactin X (topspace X - (U ∪ V))
      using UV that by (intro closedin_compact_space closedin_diff openin_Un)
    auto
      ultimately have separated_between X C (topspace X - (U ∪ V)) ↔
disjnt C (topspace X - (U ∪ V))
      by (simp add: ⟨C ∈ quasi_components_of X⟩ separated_between_quasi_component_compact)
      moreover have disjnt C (topspace X - (U ∪ V))
        using UV SUT disjnt_def by fastforce
      ultimately have separated_between X C (topspace X - (U ∪ V))
        by simp
      then obtain A B where openin X A openin X B A ∪ B = topspace X
disjnt A B C ⊆ A
        and subB: topspace X - (U ∪ V) ⊆ B
        by (meson separated_between_def)
      have B ∪ U = topspace X - (A ∩ V)
      proof
        show B ∪ U ⊆ topspace X - A ∩ V
          using ⟨openin X U⟩ ⟨disjnt U V⟩ ⟨disjnt A B⟩ ⟨openin X B⟩ disjnt_iff
openin_closedin_eq by fastforce
        show topspace X - A ∩ V ⊆ B ∪ U
          using ⟨A ∪ B = topspace X⟩ subB by fastforce
      qed
      then have closedin X (B ∪ U)
        using ⟨openin X V⟩ ⟨openin X A⟩ by auto
      then have C ⊆ B ∪ U ∨ disjnt C (B ∪ U)
        using quasi_component_of_clopen_cases [OF C] ⟨openin X U⟩ ⟨openin
X B⟩ by blast
      with UV show S = {}
        by (metis UnE ⟨C ⊆ A⟩ ⟨S ⊆ C⟩ T ⟨disjnt A B⟩ all_not_in_conv
disjnt_Un2 disjnt_iff subset_eq)
      qed
    qed

```

qed
qed

lemma *quasi_eq_connected_component_of*:

finite(connected_components_of X) \vee
finite(quasi_components_of X) \vee
locally_connected_space X \vee
compact_space X \wedge (Hausdorff_space X \vee regular_space X \vee normal_space
X)
 \implies *quasi_component_of X x = connected_component_of X x*
by (*metis quasi_eq_connected_components_of quasi_eq_connected_components_of_pointwise_alt*)

6.6.19 Additional quasicomponent and continuum properties like Boundary Bumping

lemma *cut_wire_fence_theorem_gen*:

assumes *compact_space X* **and** *X: Hausdorff_space X \vee regular_space X \vee normal_space X*
and *S: compactin X S* **and** *T: closedin X T*
and *dis: $\bigwedge C. connectedin X C \implies disjnt C S \vee disjnt C T$*
shows *separated_between X S T*
proof –
have *x \in topspace X* **if** *x \in S* **and** *T = {}* **for** *x*
using *that S compactin_subset_topspace* **by** *auto*
moreover **have** *separated_between X {x} {y}* **if** *x \in S* **and** *y \in T* **for** *x y*
proof (*cases x \in topspace X \wedge y \in topspace X*)
case *True*
then **have** \neg *connected_component_of X x y*
by (*meson dis connected_component_of_def disjnt_iff that*)
with *True X \langle compact_space X \rangle* **show** *?thesis*
by (*metis quasi_component_nonseparated quasi_eq_connected_component_of*)
next
case *False*
then **show** *?thesis*
using *S T compactin_subset_topspace closedin_subset that* **by** *blast*
qed
ultimately **show** *?thesis*
using *assms*
by (*simp add: separated_between_pointwise_left separated_between_pointwise_right*
closedin_compact_space closedin_subset)
qed

lemma *cut_wire_fence_theorem*:

\llbracket *compact_space X; Hausdorff_space X; closedin X S; closedin X T;*
 $\bigwedge C. connectedin X C \implies disjnt C S \vee disjnt C T$
 \implies *separated_between X S T*
by (*simp add: closedin_compact_space cut_wire_fence_theorem_gen*)

lemma *separated_between_from_closed_subtopology*:

assumes *XC*: *separated_between* (subtopology *X C*) *S* (*X frontier_of C*)

and *ST*: *separated_between* (subtopology *X C*) *S T*

shows *separated_between X S T*

proof –

obtain *U* **where** *clo*: *closedin* (subtopology *X C*) *U* **and** *ope*: *openin* (subtopology *X C*) *U*

and $S \subseteq U$ **and** *sub*: $X \text{ frontier_of } C \cup T \subseteq \text{topspace (subtopology } X C) - U$

by (*meson assms separated_between separated_between_Un*)

then have $X \text{ frontier_of } C \cup T \subseteq \text{topspace } X \cap C - U$

by *auto*

have *closedin X (topspace X \cap C)*

by (*metis XC frontier_of_restrict frontier_of_subset_eq inf_le1 separated_between_imp_subset topspace_subtopology*)

then have *closedin X U*

by (*metis clo closedin_closed_subtopology_subtopology_restrict*)

moreover have $\text{openin (subtopology } X C) U \longleftrightarrow \text{openin } X U \wedge U \subseteq C$

using *disjnt_iff sub* **by** (*force intro!: openin_subset_topspace_eq*)

with *ope* **have** *openin X U*

by *blast*

moreover have $T \subseteq \text{topspace } X - U$

using *ope openin_closedin_eq sub* **by** *auto*

ultimately show *?thesis*

using $\langle S \subseteq U \rangle$ *separated_between* **by** *blast*

qed

lemma *separated_between_from_closed_subtopology_frontier*:

separated_between (subtopology *X T*) *S* (*X frontier_of T*)

$\implies \text{separated_between } X S (X \text{ frontier_of } T)$

using *separated_between_from_closed_subtopology* **by** *blast*

lemma *separated_between_from_frontier_of_closed_subtopology*:

assumes *separated_between* (subtopology *X T*) *S* (*X frontier_of T*)

shows *separated_between X S (topspace X - T)*

proof –

have *disjnt S (topspace X - T)*

using *assms disjnt_iff separated_between_imp_subset* **by** *fastforce*

then show *?thesis*

by (*metis Diff_subset assms frontier_of_complement separated_between_from_closed_subtopology separated_between_frontier_of_eq'*)

qed

lemma *separated_between_compact_connected_component*:

assumes *locally_compact_space X Hausdorff_space X*

and *C*: $C \in \text{connected_components_of } X$

and *compactin X C closedin X T disjnt C T*

shows *separated_between X C T*

```

proof –
  have  $C_{sub}: C \subseteq \text{topspace } X$ 
    by (simp add: assms(4) compactin_subset_topspace)
  have Hausdorff_space (subtopology  $X$  (topspace  $X - T$ ))
    using Hausdorff_space_subtopology assms(2) by blast
  moreover have compactin (subtopology  $X$  (topspace  $X - T$ ))  $C$ 
    using assms  $C_{sub}$  by (metis Diff_Int_distrib Diff_empty compact_imp_compactin_subtopology
disjnt_def le_iff_inf)
  moreover have locally_compact_space (subtopology  $X$  (topspace  $X - T$ ))
    by (meson assms closedin_def locally_compact_Hausdorff_imp_regular_space
locally_compact_space_open_subset)
  ultimately
  obtain  $N L$  where openin  $X N$  compactin  $X L$  closedin  $X L$   $C \subseteq N$   $N \subseteq L$ 
    and  $L_{sub}: L \subseteq \text{topspace } X - T$ 
    using  $\langle \text{Hausdorff\_space } X \rangle \langle \text{closedin } X T \rangle$ 
  apply (simp add: locally_compact_space_compact_closed_compact compactin_subtopology)
    by (meson closedin_def compactin_imp_closedin openin_trans_full)
  then have disC: disjnt  $C$  (topspace  $X - L$ )
    by (meson DiffD2 disjnt_iff_subset_iff)
  have separated_between (subtopology  $X L$ )  $C$  ( $X$  frontier_of  $L$ )
  proof (rule cut_wire_fence_theorem)
    show compact_space (subtopology  $X L$ )
      by (simp add: compactin_X_L) compact_space_subtopology)
    show Hausdorff_space (subtopology  $X L$ )
      by (simp add: Hausdorff_space_subtopology  $\langle \text{Hausdorff\_space } X \rangle$ )
    show closedin (subtopology  $X L$ )  $C$ 
      by (meson  $\langle C \subseteq N \rangle \langle N \subseteq L \rangle \langle \text{Hausdorff\_space } X \rangle \langle \text{compactin } X C \rangle$ 
closedin_subset_topspace compactin_imp_closedin subset_trans)
    show closedin (subtopology  $X L$ ) ( $X$  frontier_of  $L$ )
      by (simp add: closedin_X_L) closedin_frontier_of_closedin_subset_topspace
frontier_of_subset_closedin)
    show disjnt  $D C \vee \text{disjnt } D$  ( $X$  frontier_of  $L$ )
      if connectedin (subtopology  $X L$ )  $D$  for  $D$ 
    proof (rule ccontr)
      assume  $\neg (\text{disjnt } D C \vee \text{disjnt } D$  ( $X$  frontier_of  $L$ ))
      moreover have connectedin  $X D$ 
        using connectedin_subtopology that by blast
      ultimately show False
        using that connected_components_of_maximal [of  $C X D$ ]  $C$ 
        apply (simp add: disjnt_iff)
        by (metis Diff_eq_empty_iff  $\langle C \subseteq N \rangle \langle N \subseteq L \rangle \langle \text{openin } X N \rangle$  disjoint_iff
frontier_of_openin_straddle_Int(2) subsetD)
    qed
  qed
  then have separated_between  $X$  ( $X$  frontier_of  $C$ ) (topspace  $X - L$ )
    using separated_between_from_frontier_of_closed_subtopology separated_between_frontier_of_eq
by blast
  with  $\langle \text{closedin } X T \rangle$ 
    separated_between_frontier_of [OF  $C_{sub}$  disC]

```

1316

show *?thesis*
unfolding *separated_between* **by** (*smt (verit) Diff_iff Lsub closedin_subset subset_iff*)
qed

lemma *wilder_locally_compact_component_thm*:
assumes *locally_compact_space X Hausdorff_space X*
and $C \in \text{connected_components_of } X$ *compactin X C openin X W* $C \subseteq W$
obtains $U V$ **where** *openin X U openin X V disjnt U V* $U \cup V = \text{topspace } X$
 $C \subseteq U$ $U \subseteq W$
proof –
have *closedin X (topspace X - W)*
using $\langle \text{openin } X \ W \rangle$ **by** *blast*
moreover have *disjnt C (topspace X - W)*
using $\langle C \subseteq W \rangle$ *disjnt_def* **by** *fastforce*
ultimately have *separated_between X C (topspace X - W)*
using *separated_between_compact_connected_component* *assms* **by** *blast*
then show *thesis*
by (*smt (verit, del_insts) DiffI disjnt_iff openin_subset separated_between_def subset_iff that*)
qed

lemma *compact_quasi_eq_connected_components_of*:
assumes *locally_compact_space X Hausdorff_space X compactin X C*
shows $C \in \text{quasi_components_of } X \longleftrightarrow C \in \text{connected_components_of } X$
proof –
have *compactin X (connected_component_of_set X x)*
if $x \in \text{topspace } X$ *compactin X (quasi_component_of_set X x)* **for** x
proof (*rule closed_compactin*)
show *compactin X (quasi_component_of_set X x)*
by (*simp add: that*)
show *connected_component_of_set X x* \subseteq *quasi_component_of_set X x*
by (*simp add: connected_component_subset_quasi_component_of*)
show *closedin X (connected_component_of_set X x)*
by (*simp add: closedin_connected_component_of*)
qed
moreover have *connected_component_of X x = quasi_component_of X x*
if $\S: x \in \text{topspace } X$ *compactin X (connected_component_of_set X x)* **for** x
proof –
have $\bigwedge y. \text{connected_component_of } X \ x \ y \implies \text{quasi_component_of } X \ x \ y$
by (*simp add: connected_imp_quasi_component_of*)
moreover have *False* **if** *non:* $\neg \text{connected_component_of } X \ x \ y$ **and** *quasi:*
quasi_component_of X x y **for** y
proof –
have $y \in \text{topspace } X$
by (*meson quasi_component_of_equiv that*)
then have *closedin X {y}*
by (*simp add: Hausdorff_space X compactin_imp_closedin*)
moreover have *disjnt (connected_component_of_set X x) {y}*


```

    by (simp add: non)
  moreover have  $\neg$  separated_between  $X$  (connected_component_of_set  $X$   $x$ )
  { $y$ }
    using  $\S$  quasi_separated_between_pointwise_left
  by (fastforce simp: quasi_component_nonseparated connected_component_of_refl)
  ultimately show False
  using assms by (metis  $\S$  connected_component_in_connected_components_of
separated_between_compact_connected_component)
  qed
  ultimately show ?thesis
  by blast
  qed
  ultimately show ?thesis
  using  $\langle$ compactin  $X$   $C$  $\rangle$  unfolding connected_components_of_def image_iff
quasi_components_of_def by metis
  qed

```

lemma boundary_bumping_theorem_closed_gen:

```

  assumes connected_space  $X$  locally_compact_space  $X$  Hausdorff_space  $X$  closedin
 $X$   $S$ 
   $S \neq$  topspace  $X$  and  $C$ : compactin  $X$   $C$   $C \in$  connected_components_of (subtopology
 $X$   $S$ )
  shows  $C \cap X$  frontier_of  $S \neq \{\}$ 
  proof
    assume  $\S$ :  $C \cap X$  frontier_of  $S = \{\}$ 
    consider  $C \neq \{\}$   $X$  frontier_of  $S \subseteq$  topspace  $X$  |  $C \subseteq$  topspace  $X$   $S = \{\}$ 
    using  $C$  by (metis frontier_of_subset_topspace nonempty_connected_components_of)
    then show False
  proof cases
    case 1
    have separated_between (subtopology  $X$   $S$ )  $C$  ( $X$  frontier_of  $S$ )
    proof (rule separated_between_compact_connected_component)
      show compactin (subtopology  $X$   $S$ )  $C$ 
      using  $C$  compact_imp_compactin_subtopology connected_components_of_subset
    by fastforce
    show closedin (subtopology  $X$   $S$ ) ( $X$  frontier_of  $S$ )
    by (simp add:  $\langle$ closedin  $X$   $S$  $\rangle$  closedin_frontier_of closedin_subset_topspace
frontier_of_subset_closedin)
    show disjnt  $C$  ( $X$  frontier_of  $S$ )
    using  $\S$  by (simp add: disjnt_def)
  qed (use assms Hausdorff_space_subtopology_locally_compact_space_closed_subset
in auto)
  then have separated_between  $X$   $C$  ( $X$  frontier_of  $S$ )
  using separated_between_from_closed_subtopology by auto
  then have  $X$  frontier_of  $S = \{\}$ 
  using  $\langle$  $C \neq \{\}$  $\rangle$   $\langle$ connected_space  $X$  $\rangle$  connected_space_separated_between by
blast
  moreover have  $C \subseteq S$ 

```

1318

```
      using C connected_components_of_subset by fastforce
    ultimately show False
      using 1 assms by (metis closedin_subset connected_space_eq_frontier_eq_empty
subset_empty)
    next
      case 2
      then show False
        using C connected_components_of_eq_empty by fastforce
    qed
  qed
```

lemma *boundary_bumping_theorem_closed*:

```
  assumes connected_space X compact_space X Hausdorff_space X closedin X S
    S ≠ topspace X C ∈ connected_components_of(subtopology X S)
  shows C ∩ X frontier_of S ≠ {}
  by (meson assms boundary_bumping_theorem_closed_gen closedin_compact_space
closedin_connected_components_of
closedin_trans_full compact_imp_locally_compact_space)
```

lemma *intermediate_continuum_exists*:

```
  assumes connected_space X locally_compact_space X Hausdorff_space X
    and C: compactin X C connectedin X C C ≠ {} C ≠ topspace X
    and U: openin X U C ⊆ U
  obtains D where compactin X D connectedin X D C ⊂ D D ⊂ U
proof –
  have C ⊆ topspace X
    by (simp add: C compactin_subset_topspace)
  with C obtain a where a: a ∈ topspace X a ∉ C
    by blast
  moreover have compactin (subtopology X (U - {a})) C
    by (simp add: C U a compact_imp_compactin_subtopology_subset_Diff_insert)
  moreover have Hausdorff_space (subtopology X (U - {a}))
    using Hausdorff_space_subtopology assms(3) by blast
  moreover
  have locally_compact_space (subtopology X (U - {a}))
    by (rule locally_compact_space_open_subset)
    (auto simp: locally_compact_Hausdorff_imp_regular_space open_in_Hausdorff_delete
assms)
  ultimately obtain V K where V: openin X V a ∉ V V ⊆ U and K: compactin
X K a ∉ K K ⊆ U
    and cloK: closedin (subtopology X (U - {a})) K and C ⊆ V V ⊆ K
    using locally_compact_space_compact_closed_compact [of subtopology X (U
- {a})] assms
    by (smt (verit, del_insts) Diff_empty compactin_subtopology open_in_Hausdorff_delete
openin_open_subtopology_subset_Diff_insert)
  then obtain D where D: D ∈ connected_components_of (subtopology X K)
and C ⊆ D
    using C
```

```

by (metis compactin_subset_topospace connected_component_in_connected_components_of
      connected_component_of_maximal_connectedin_subtopology subset_empty
      subset_eq topospace_subtopology_subset)
show thesis
proof
  have cloD: closedin (subtopology X K) D
  by (simp add: D closedin_connected_components_of)
  then have XKD: compactin (subtopology X K) D
  by (simp add: K closedin_compact_space compact_space_subtopology)
  then show compactin X D
  using compactin_subtopology_imp_compact by blast
  show connectedin X D
  using D connectedin_connected_components_of_connectedin_subtopology by
blast
  have K ≠ topospace X
  using K a by blast
  moreover have V ⊆ X interior_of K
  by (simp add: ⟨openin X V⟩ ⟨V ⊆ K⟩ interior_of_maximal)
  ultimately have C ≠ D
  using boundary_bumping_theorem_closed_gen [of X K C] D ⟨C ⊆ V⟩
  by (auto simp add: assms K compactin_imp_closedin frontier_of_def)
  then show C ⊂ D
  using ⟨C ⊆ D⟩ by blast
  have D ⊆ U
  using K(β) ⟨closedin (subtopology X K) D⟩ closedin_imp_subset by blast
  moreover have D ≠ U
  using K XKD ⟨C ⊂ D⟩ assms
  by (metis ⟨K ≠ topospace X⟩ cloD closedin_imp_subset compactin_imp_closedin
      connected_space_clopen_in
      inf_bot_left inf_le2 subset_antisym)
  ultimately
  show D ⊂ U by blast
qed
qed

```

lemma boundary_bumping_theorem_gen:

```

assumes X: connected_space X locally_compact_space X Hausdorff_space X
and S ⊂ topospace X and C: C ∈ connected_components_of (subtopology X S)
and compC: compactin X (X closure_of C)
shows X frontier_of C ∩ X frontier_of S ≠ {}
proof -
  have Csub: C ⊆ topospace X C ⊆ S and connectedin X C
  using C connectedin_connected_components_of_connectedin_subset_topospace
      connectedin_subtopology
  by fastforce+
  have C ≠ {}
  using C nonempty_connected_components_of by blast
  obtain X interior_of C ⊆ X interior_of S X closure_of C ⊆ X closure_of S

```

by (simp add: Csub closure_of_mono interior_of_mono)
 moreover have False if $X \text{ closure_of } C \subseteq X \text{ interior_of } S$
 proof –
 have $X \text{ closure_of } C = C$
 by (meson C closedin_connected_component_of_subtopology closure_of_eq
 interior_of_subset order_trans that)
 with that have $C \subseteq X \text{ interior_of } S$
 by simp
 then obtain D where $\text{compactin } X D$ and $\text{connectedin } X D$ and $C \subset D$ and
 $D \subset X \text{ interior_of } S$
 using intermediate_continuum_exists assms $\langle X \text{ closure_of } C = C \rangle \text{ compC}$
 C_{sub}
 by (metis $\langle C \neq \{\} \rangle \langle \text{connectedin } X C \rangle \text{ openin_interior_of psubsetE}$)
 then have $D \subseteq C$
 by (metis $C \langle C \neq \{\} \rangle \text{ connected_components_of_maximal connectedin_subtopology}$
 $\text{disjnt_def inf.orderE interior_of_subset order_trans psubsetE}$)
 then show False
 using $\langle C \subset D \rangle$ by blast
 qed
 ultimately show ?thesis
 by (smt (verit, ccfv_SIG) DiffI disjoint_iff_not_equal frontier_of_def sub-
 set_eq)
 qed

lemma *boundary_bumping_theorem*:

$\llbracket \text{connected_space } X; \text{ compact_space } X; \text{ Hausdorff_space } X; S \subset \text{topspace } X;$
 $C \in \text{connected_components_of}(\text{subtopology } X S) \rrbracket$
 $\implies X \text{ frontier_of } C \cap X \text{ frontier_of } S \neq \{\}$
 by (simp add: boundary_bumping_theorem_gen closedin_compact_space com-
 pact_imp_locally_compact_space)

6.6.20 Compactly generated spaces (k-spaces)

These don't have to be Hausdorff

definition *k_space* where

$k_space X \equiv$
 $\forall S. S \subseteq \text{topspace } X \implies$
 $(\text{closedin } X S \iff (\forall K. \text{compactin } X K \implies \text{closedin } (\text{subtopology } X K) (K$
 $\cap S)))$

lemma *k_space*:

$k_space X \iff$
 $(\forall S. S \subseteq \text{topspace } X \wedge$
 $(\forall K. \text{compactin } X K \implies \text{closedin } (\text{subtopology } X K) (K \cap S)) \implies \text{closedin}$
 $X S)$
 by (metis closedin_subtopology inf_commute k_space_def)

lemma *k_space_open*:

$k_space X \iff$

$(\forall S. S \subseteq \text{topspace } X \wedge$
 $(\forall K. \text{compactin } X K \longrightarrow \text{openin } (\text{subtopology } X K) (K \cap S)) \longrightarrow \text{openin } X S)$
proof –
have $\text{openin } X S$
if $k_space X S \subseteq \text{topspace } X$
and $\forall K. \text{compactin } X K \longrightarrow \text{openin } (\text{subtopology } X K) (K \cap S)$ **for** S
using that unfolding $k_space \text{openin_closedin_eq}$
by $(\text{metis } \text{Diff_Int_distrib2 } \text{Diff_subset } \text{inf_commute } \text{topspace_subtopology})$
moreover have $k_space X$
if $\forall S. S \subseteq \text{topspace } X \wedge (\forall K. \text{compactin } X K \longrightarrow \text{openin } (\text{subtopology } X K) (K \cap S)) \longrightarrow \text{openin } X S$
unfolding $k_space \text{openin_closedin_eq}$
by $(\text{simp add: } \text{Diff_Int_distrib } \text{closedin_def } \text{inf_commute } \text{that})$
ultimately show $?thesis$
by blast
qed

lemma k_space_alt :
 $k_space X \longleftrightarrow$
 $(\forall S. S \subseteq \text{topspace } X$
 $\longrightarrow (\text{openin } X S \longleftrightarrow (\forall K. \text{compactin } X K \longrightarrow \text{openin } (\text{subtopology } X K) (K \cap S))))$
by $(\text{meson } k_space_open \text{openin_subtopology_Int2})$

lemma $k_space_quotient_map_image$:
assumes $q: \text{quotient_map } X Y q$ **and** $X: k_space X$
shows $k_space Y$
unfolding k_space
proof clarify
fix S
assume $S \subseteq \text{topspace } Y$ **and** $S: \forall K. \text{compactin } Y K \longrightarrow \text{closedin } (\text{subtopology } Y K) (K \cap S)$
then have iff: $\text{closedin } X \{x \in \text{topspace } X. q x \in S\} \longleftrightarrow \text{closedin } Y S$
using $q \text{quotient_map_closedin}$ **by fastforce**
have $\text{closedin } (\text{subtopology } X K) (K \cap \{x \in \text{topspace } X. q x \in S\})$ **if** $\text{compactin } X K$ **for** K
proof –
have $\{x \in \text{topspace } X. q x \in q ' K\} \cap K = K$
using $\text{compactin_subset_topspace}$ **that by blast**
then have $*$: $\text{subtopology } X K = \text{subtopology } (\text{subtopology } X \{x \in \text{topspace } X. q x \in q ' K\}) K$
by $(\text{simp add: } \text{subtopology_subtopology})$
have $**$: $K \cap \{x \in \text{topspace } X. q x \in S\} =$
 $K \cap \{x \in \text{topspace } (\text{subtopology } X \{x \in \text{topspace } X. q x \in q ' K\}). q x \in q ' K \cap S\}$
by auto
have $K \subseteq \text{topspace } X$
by $(\text{simp add: } \text{compactin_subset_topspace } \text{that})$

```

show ?thesis
unfolding * **
proof (intro closedin_continuous_map_preimage closedin_subtopology_Int_closed)
  show continuous_map (subtopology X {x ∈ topspace X. q x ∈ q ‘ K})
(subtopology Y (q ‘ K)) q
  by (auto simp add: continuous_map_in_subtopology continuous_map_from_subtopology
q quotient_imp_continuous_map)
  show closedin (subtopology Y (q ‘ K)) (q ‘ K ∩ S)
  by (meson S image_compactin q quotient_imp_continuous_map that)
qed
qed
then have closedin X {x ∈ topspace X. q x ∈ S}
  by (metis (no_types, lifting) X k_space mem_Collect_eq subsetI)
with iff show closedin Y S by simp
qed

```

```

lemma k_space_retraction_map_image:
  [[retraction_map X Y r; k_space X]] ⇒ k_space Y
using k_space_quotient_map_image retraction_imp_quotient_map by blast

```

```

lemma homeomorphic_k_space:
  X homeomorphic_space Y ⇒ k_space X ↔ k_space Y
by (meson homeomorphic_map_def homeomorphic_space homeomorphic_space_sym
k_space_quotient_map_image)

```

```

lemma k_space_perfect_map_image:
  [[k_space X; perfect_map X Y f]] ⇒ k_space Y
using k_space_quotient_map_image perfect_imp_quotient_map by blast

```

```

lemma locally_compact_imp_k_space:
assumes locally_compact_space X
shows k_space X
unfolding k_space
proof clarify
  fix S
  assume S ⊆ topspace X and S: ∀ K. compactin X K → closedin (subtopology
X K) (K ∩ S)
  have False if non: ¬ (X closure_of S ⊆ S)
  proof –
    obtain x where x ∈ X closure_of S x ∉ S
    using non by blast
    then have x ∈ topspace X
      by (simp add: in_closure_of)
    then obtain K U where openin X U compactin X K x ∈ U U ⊆ K
      by (meson assms locally_compact_space_def)
    then show False
      using ⟨x ∈ X closure_of S⟩ openin_Int_closure_of_eq [OF ⟨openin X U⟩]
      by (smt (verit, ccfv_threshold) Int_iff S ⟨x ∉ S⟩ closedin_Int_closure_of
inf.orderE inf_assoc)
  qed

```

```

qed
then show closedin X S
  using S ‹S ⊆ topspace X› closure_of_subset_eq by blast
qed

```

```

lemma compact_imp_k_space:
  compact_space X ⟹ k_space X
  by (simp add: compact_imp_locally_compact_space locally_compact_imp_k_space)

```

```

lemma k_space_discrete_topology: k_space(discrete_topology U)
  by (simp add: k_space_open)

```

```

lemma k_space_closed_subtopology:
  assumes k_space X closedin X C
  shows k_space (subtopology X C)
unfolding k_space_compactin_subtopology
proof clarsimp

```

```

  fix S
  assume Ssub: S ⊆ topspace X S ⊆ C
  and S: ∀K. compactin X K ∧ K ⊆ C ⟶ closedin (subtopology (subtopology
X C) K) (K ∩ S)
  have closedin (subtopology X K) (K ∩ S) if compactin X K for K
  proof -
    have closedin (subtopology (subtopology X C) (K ∩ C)) ((K ∩ C) ∩ S)
    by (simp add: S ‹closedin X C› compact_Int_closedin that)
    then show ?thesis
    using ‹closedin X C› Ssub by (auto simp add: closedin_subtopology)
  qed
  then show closedin (subtopology X C) S
  by (metis Ssub ‹k_space X› closedin_subset_topspace k_space_def)
qed

```

```

lemma k_space_subtopology:
  assumes 1: ∧T. [T ⊆ topspace X; T ⊆ S;
    ∧K. compactin X K ⟹ closedin (subtopology X (K ∩ S)) (K ∩
T)] ⟹ closedin (subtopology X S) T
  assumes 2: ∧K. compactin X K ⟹ k_space(subtopology X (K ∩ S))
  shows k_space (subtopology X S)
  unfolding k_space
proof (intro conjI strip)
  fix U
  assume §: U ⊆ topspace (subtopology X S) ∧ (∀K. compactin (subtopology X S)
K ⟶ closedin (subtopology (subtopology X S) K) (K ∩ U))
  have closedin (subtopology X (K ∩ S)) (K ∩ U) if compactin X K for K
  proof -
    have K ∩ U ⊆ topspace (subtopology X (K ∩ S))
    using § by auto
  moreover
  have ∧K'. compactin (subtopology X (K ∩ S)) K' ⟹ closedin (subtopology

```

```

(subtopology X (K ∩ S)) K') (K' ∩ K ∩ U)
  by (metis § compactin_subtopology_inf.orderE inf_commute subtopology_subtopology)
  ultimately show ?thesis
    by (metis (no_types, opaque_lifting) 2 inf.assoc k_space_def that)
qed
then show closedin (subtopology X S) U
  using 1 § by auto
qed

```

lemma *k_space_subtopology_open*:

```

assumes 1:  $\bigwedge T. \llbracket T \subseteq \text{topspace } X; T \subseteq S; \bigwedge K. \text{compactin } X K \implies \text{openin } (\text{subtopology } X (K \cap S)) (K \cap T) \rrbracket \implies \text{openin } (\text{subtopology } X S) T$ 
assumes 2:  $\bigwedge K. \text{compactin } X K \implies \text{k\_space}(\text{subtopology } X (K \cap S))$ 
shows k_space (subtopology X S)
  unfolding k_space_open
proof (intro conjI strip)
  fix U
  assume §:  $U \subseteq \text{topspace } (\text{subtopology } X S) \wedge (\forall K. \text{compactin } (\text{subtopology } X S) K \longrightarrow \text{openin } (\text{subtopology } (\text{subtopology } X S) K) (K \cap U))$ 
  have openin (subtopology X (K ∩ S)) (K ∩ U) if compactin X K for K
  proof -
    have  $K \cap U \subseteq \text{topspace } (\text{subtopology } X (K \cap S))$ 
    using § by auto
  moreover
    have  $\bigwedge K'. \text{compactin } (\text{subtopology } X (K \cap S)) K' \implies \text{openin } (\text{subtopology } (\text{subtopology } X (K \cap S)) K') (K' \cap K \cap U)$ 
    by (metis § compactin_subtopology_inf.orderE inf_commute subtopology_subtopology)
    ultimately show ?thesis
      by (metis (no_types, opaque_lifting) 2 inf.assoc k_space_open that)
  qed
  then show openin (subtopology X S) U
    using 1 § by auto
qed

```

lemma *k_space_open_subtopology_aux*:

```

assumes kc_space X compact_space X openin X V
shows k_space (subtopology X V)
proof (clarsimp simp: k_space_subtopology_subtopology compactin_subtopology Int_absorb1)
  fix S
  assume S ⊆ topspace X
  and S ⊆ V
  and S:  $\forall K. \text{compactin } X K \wedge K \subseteq V \longrightarrow \text{closedin } (\text{subtopology } X K) (K \cap S)$ 
  then have V ⊆ topspace X
    using assms openin_subset by blast
  have S = V ∩ ((topspace X - V) ∪ S)
    using ⟨S ⊆ V⟩ by auto

```



```

moreover have closedin (subtopology  $X$   $V$ ) ( $V \cap ((\text{topspace } X - V) \cup S)$ )
proof (intro closedin_subtopology_Int_closed compactin_imp_closedin_gen  $\langle k_c\_space$ 
 $X \rangle$ )
  show compactin  $X$  ( $\text{topspace } X - V \cup S$ )
    unfolding compactin_def
  proof (intro conjI strip)
    show  $\text{topspace } X - V \cup S \subseteq \text{topspace } X$ 
      by (simp add:  $\langle S \subseteq \text{topspace } X \rangle$ )
    fix  $U$ 
    assume  $U$ : Ball  $U$  (openin  $X$ )  $\wedge$   $\text{topspace } X - V \cup S \subseteq \bigcup U$ 
    moreover
    have compactin  $X$  ( $\text{topspace } X - V$ )
      using assms closedin_compact_space by blast
    ultimately obtain  $\mathcal{G}$  where finite  $\mathcal{G}$   $\mathcal{G} \subseteq U$  and  $\mathcal{G}$ :  $\text{topspace } X - V \subseteq \bigcup \mathcal{G}$ 
      unfolding compactin_def using  $\langle V \subseteq \text{topspace } X \rangle$  by (metis le_sup_iff)
    then have  $\text{topspace } X - \bigcup \mathcal{G} \subseteq V$ 
      by blast
    then have closedin (subtopology  $X$  ( $\text{topspace } X - \bigcup \mathcal{G}$ )) ( $(\text{topspace } X - \bigcup \mathcal{G})$ 
 $\cap S$ )
      by (meson  $S$   $U$   $\langle \mathcal{G} \subseteq U \rangle$  compact_space  $X$ ) closedin_compact_space
openin_Union openin_closedin_eq subset_iff)
    then have compactin  $X$  ( $(\text{topspace } X - \bigcup \mathcal{G}) \cap S$ )
      by (meson  $U$   $\langle \mathcal{G} \subseteq U \rangle$  compact_space  $X$ ) closedin_compact_space closedin_trans_full
openin_Union openin_closedin_eq subset_iff)
    then obtain  $\mathcal{H}$  where finite  $\mathcal{H}$   $\mathcal{H} \subseteq U$   $(\text{topspace } X - \bigcup \mathcal{G}) \cap S \subseteq \bigcup \mathcal{H}$ 
      unfolding compactin_def by (smt (verit, best)  $U$  inf_le2 subset_trans
sup.boundedE)
    with  $\mathcal{G}$  have  $\text{topspace } X - V \cup S \subseteq \bigcup (\mathcal{G} \cup \mathcal{H})$ 
      using  $\langle S \subseteq \text{topspace } X \rangle$  by auto
    then show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq U \wedge \text{topspace } X - V \cup S \subseteq \bigcup \mathcal{F}$ 
      by (metis  $\langle \mathcal{G} \subseteq U \rangle$   $\langle \mathcal{H} \subseteq U \rangle$   $\langle \text{finite } \mathcal{G} \rangle$   $\langle \text{finite } \mathcal{H} \rangle$  finite_Un le_sup_iff)
    qed
  qed
ultimately show closedin (subtopology  $X$   $V$ )  $S$ 
  by metis
qed

```

lemma *k_space_open_subtopology*:

assumes X : *k_space* $X \vee$ *Hausdorff_space* $X \vee$ *regular_space* X **and** *k_space*
 X *openin* X S

shows *k_space*(subtopology X S)

proof (*rule k_space_subtopology_open*)

fix T

assume $T \subseteq \text{topspace } X$

and $T \subseteq S$

and T : $\bigwedge K. \text{compactin } X K \implies \text{openin (subtopology } X (K \cap S)) (K \cap T)$

have *openin* (subtopology X K) ($K \cap T$) **if** *compactin* X K **for** K

by (*smt* (*verit, ccfv_threshold*) T *assms*(3) *inf_assoc inf_commute openin_Int*)

```

openin_subtopology that)
  then show openin (subtopology X S) T
    by (metis ‹T ⊆ S› ‹T ⊆ topspace X› assms(2) k_space_alt subset_openin_subtopology)
next
fix K
assume compactin X K
then have KS: openin (subtopology X K) (K ∩ S)
  by (simp add: ‹openin X S› openin_subtopology_Int2)
have XK: compact_space (subtopology X K)
  by (simp add: ‹compactin X K› compact_space_subtopology)
show k_space (subtopology X (K ∩ S))
  using X
proof (rule disjE)
  assume kc_space X
  then show k_space (subtopology X (K ∩ S))
    using k_space_open_subtopology_aux [of subtopology X K K ∩ S]
    by (simp add: KS XK kc_space_subtopology subtopology_subtopology)
  next
  assume Hausdorff_space X ∨ regular_space X
  then have locally_compact_space (subtopology (subtopology X K) (K ∩ S))
    using locally_compact_space_open_subset Hausdorff_space_subtopology KS
  XK
    compact_imp_locally_compact_space regular_space_subtopology by blast
  then show k_space (subtopology X (K ∩ S))
    by (simp add: locally_compact_imp_k_space subtopology_subtopology)
qed
qed

```

lemma *k_kc_space_subtopology*:

```

[[k_space X; kc_space X; openin X S ∨ closedin X S]] ==> k_space(subtopology
X S) ∧ kc_space(subtopology X S)
  by (metis k_space_closed_subtopology k_space_open_subtopology kc_space_subtopology)

```

lemma *k_space_as_quotient_explicit*:

```

k_space X ‹⟷› quotient_map (sum_topology (subtopology X) {K. compactin X
K}) X snd

```

proof –

```

  have [simp]: {x ∈ topspace X. x ∈ K ∧ x ∈ U} = K ∩ U if U ⊆ topspace X
for K U

```

```

  using that by blast

```

```

  show ?thesis

```

```

  apply (simp add: quotient_map_def openin_sum_topology snd_image_Sigma
k_space_alt)

```

```

  by (smt (verit, del_insts) Union_iff compactin_sing inf.orderE mem_Collect_eq
singletonI subsetI)

```

qed

lemma *k_space_as_quotient*:

```

fixes X :: 'a topology
shows k_space X  $\longleftrightarrow$  ( $\exists q. \exists Y:: ('a \text{ set } * 'a) \text{ topology. locally\_compact\_space}$ 
 $Y \wedge \text{quotient\_map } Y X q$ )
  (is ?lhs=?rhs)
proof
  show k_space X if ?rhs
    using that k_space_quotient_map_image locally_compact_imp_k_space by
blast
  next
    assume k_space X
    show ?rhs
    proof (intro exI conjI)
      show locally_compact_space (sum_topology (subtopology X) {K. compactin X
K})
        by (simp add: compact_imp_locally_compact_space compact_space_subtopology
locally_compact_space_sum_topology)
      show quotient_map (sum_topology (subtopology X) {K. compactin X K}) X
snd
        using <k_space X> k_space_as_quotient_explicit by blast
    qed
  qed

lemma k_space_prod_topology_left:
  assumes X: locally_compact_space X Hausdorff_space X  $\vee$  regular_space X
and k_space Y
  shows k_space (prod_topology X Y)
proof -
  obtain q and Z :: ('b set * 'b) topology where locally_compact_space Z and q:
quotient_map Z Y q
    using <k_space Y> k_space_as_quotient by blast
  then show ?thesis
    using quotient_map_prod_right [OF X q] X k_space_quotient_map_image
locally_compact_imp_k_space
    locally_compact_space_prod_topology by blast
qed

lemma k_space_prod_topology_right:
  assumes k_space X and Y: locally_compact_space Y Hausdorff_space Y  $\vee$ 
regular_space Y
  shows k_space (prod_topology X Y)
  using assms homeomorphic_k_space homeomorphic_space_prod_topology_swap
k_space_prod_topology_left by blast

lemma continuous_map_from_k_space:
  assumes k_space X and f:  $\bigwedge K. \text{compactin } X K \implies \text{continuous\_map}(\text{subtopology}$ 
 $X K) Y f$ 
  shows continuous_map X Y f
proof -

```

```

have  $\bigwedge x. x \in \text{topspace } X \implies f x \in \text{topspace } Y$ 
  by (metis compactin_absolute compactin_sing f image_compactin image_empty
image_insert)
moreover have closedin X  $\{x \in \text{topspace } X. f x \in C\}$  if closedin Y C for C
proof -
  have  $\{x \in \text{topspace } X. f x \in C\} \subseteq \text{topspace } X$ 
    by fastforce
  moreover
  have eq:  $K \cap \{x \in \text{topspace } X. f x \in C\} = \{x. x \in \text{topspace}(\text{subtopology } X K) \wedge f x \in (f ' K \cap C)\}$  for K
    by auto
  have closedin (subtopology X K)  $(K \cap \{x \in \text{topspace } X. f x \in C\})$  if compactin
  X K for K
    unfolding eq
  proof (rule closedin_continuous_map_preimage)
    show continuous_map (subtopology X K) (subtopology Y (f'K)) f
      by (simp add: continuous_map_in_subtopology f image_mono that)
    show closedin (subtopology Y (f'K))  $(f ' K \cap C)$ 
      using  $\langle \text{closedin } Y C \rangle$  closedin_subtopology by blast
  qed
  ultimately show ?thesis
    using  $\langle k\_space X \rangle k\_space$  by blast
qed
ultimately show ?thesis
  by (simp add: continuous_map_closedin)
qed

lemma closed_map_into_k_space:
  assumes  $k\_space Y$  and  $fim: f \in (\text{topspace } X) \rightarrow \text{topspace } Y$ 
  and  $f: \bigwedge K. compactin Y K$ 
   $\implies \text{closed\_map}(\text{subtopology } X \{x \in \text{topspace } X. f x \in K\}) (\text{subtopology } Y K) f$ 
  shows closed_map X Y f
  unfolding closed_map_def
proof (intro strip)
  fix C
  assume closedin X C
  have closedin (subtopology Y K)  $(K \cap f ' C)$ 
    if compactin Y K for K
  proof -
    have eq:  $K \cap f ' C = f ' (\{x \in \text{topspace } X. f x \in K\} \cap C)$ 
      using  $\langle \text{closedin } X C \rangle$  closedin_subset by auto
    show ?thesis
      unfolding eq
      by (metis (no_types, lifting)  $\langle \text{closedin } X C \rangle$  closed_map_def closedin_subtopology
f inf_commute that)
  qed
  then show closedin Y  $(f ' C)$ 
    using  $\langle k\_space Y \rangle$  unfolding k_space

```

by (meson <closedin X C> closedin_subset order.trans fim funcset_image subset_image_iff)
qed

Essentially the same proof

lemma *open_map_into_k_space:*

assumes *k_space Y* **and** *fim: f ∈ (topspace X) → topspace Y*

and *f: ∧K. compactin Y K*

\implies *open_map (subtopology X {x ∈ topspace X. f x ∈ K}) (subtopology*

Y K) f

shows *open_map X Y f*

unfolding *open_map_def*

proof (*intro strip*)

fix *C*

assume *openin X C*

have *openin (subtopology Y K) (K ∩ f ' C)*

if *compactin Y K* **for** *K*

proof –

have *eq: K ∩ f ' C = f ' ({x ∈ topspace X. f x ∈ K} ∩ C)*

using <*openin X C*> *openin_subset* **by** *auto*

show *?thesis*

unfolding *eq*

by (*metis (no_types, lifting) <openin X C> open_map_def openin_subtopology f inf_commute that*)

qed

then show *openin Y (f ' C)*

using <*k_space Y*> **unfolding** *k_space_open*

by (*meson <openin X C> openin_subset order.trans fim funcset_image subset_image_iff*)

qed

lemma *quotient_map_into_k_space:*

fixes *f :: 'a ⇒ 'b*

assumes *k_space Y* **and** *cmf: continuous_map X Y f*

and *fim: f ' (topspace X) = topspace Y*

and *f: ∧k. compactin Y k*

\implies *quotient_map (subtopology X {x ∈ topspace X. f x ∈ k})*
(subtopology Y k) f

shows *quotient_map X Y f*

proof –

have *closedin Y C*

if $C \subseteq$ *topspace Y* **and** *K: closedin X {x ∈ topspace X. f x ∈ C}* **for** *C*

proof –

have *closedin (subtopology Y K) (K ∩ C)* **if** *compactin Y K* **for** *K*

proof –

define *Kf* **where** $Kf \equiv \{x \in \text{topspace } X. f x \in K\}$

have $*$: $K \cap C \subseteq$ *topspace Y* \wedge $K \cap C \subseteq K$

using <*C ⊆ topspace Y*> **by** *blast*

then have *eq: closedin (subtopology X Kf) (Kf ∩ {x ∈ topspace X. f x ∈ C})*

```

=
      closedin (subtopology Y K) (K ∩ C)
    using f [OF that] * unfolding quotient_map_closedin Kf_def
    by (smt (verit, ccfv_SIG) Collect_cong Int_def compactin_subset_topospace
mem_Collect_eq that topospace_subtopology topospace_subtopology_subset)
    have dd: {x ∈ topospace X ∩ Kf. f x ∈ K ∩ C} = Kf ∩ {x ∈ topospace X. f x
∈ C}
    by (auto simp add: Kf_def)
    have closedin (subtopology X Kf) {x ∈ topospace X. x ∈ Kf ∧ f x ∈ K ∧ f x
∈ C}
    using K closedin_subtopology by (fastforce simp add: Kf_def)
    with K closedin_subtopology_Int_closed eq show ?thesis
    by blast
  qed
  then show ?thesis
  using ⟨k_space Y⟩ that unfolding k_space by blast
  qed
  moreover have closedin X {x ∈ topospace X. f x ∈ K}
  if K ⊆ topospace Y closedin Y K for K
  using that cmf_continuous_map_closedin by fastforce
  ultimately show ?thesis
  unfolding quotient_map_closedin using fim by blast
  qed

```

lemma *quotient_map_into_k_space_eq*:

assumes *k_space Y kc_space Y*

shows *quotient_map X Y f* \longleftrightarrow

continuous_map X Y f \wedge *f* ' (*topospace X*) = *topospace Y* \wedge

($\forall K$. *compactin Y K*

\longrightarrow *quotient_map (subtopology X {x ∈ topospace X. f x ∈ K}) (subtopology*

Y K) f)

using *assms*

by (*auto simp: kc_space_def intro: quotient_map_into_k_space quotient_map_restriction*

dest: quotient_imp_continuous_map quotient_imp_surjective_map)

lemma *open_map_into_k_space_eq*:

assumes *k_space Y*

shows *open_map X Y f* \longleftrightarrow

f ∈ (*topospace X*) \rightarrow *topospace Y* \wedge

($\forall k$. *compactin Y k*

\longrightarrow *open_map (subtopology X {x ∈ topospace X. f x ∈ k}) (subtopology*

Y k) f)

using *assms open_map_imp_subset_topospace open_map_into_k_space open_map_restriction*

by *fastforce*

lemma *closed_map_into_k_space_eq*:

assumes *k_space Y*

shows *closed_map X Y f* \longleftrightarrow

f ∈ (*topospace X*) \rightarrow *topospace Y* \wedge

```

      ( $\forall k. \text{compactin } Y \ k$ 
         $\longrightarrow \text{closed\_map } (\text{subtopology } X \ \{x \in \text{topspace } X. f \ x \in k\}) \ (\text{subtopology } Y \ k) \ f)$ 
      (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (simp add: closed_map_imp_subset_topspace closed_map_restriction)
  show ?rhs  $\implies$  ?lhs
    by (simp add: assms closed_map_into_k_space)
qed

```

```

lemma proper_map_into_k_space:
  assumes k_space Y and fim:  $f \in (\text{topspace } X) \rightarrow \text{topspace } Y$ 
  and f:  $\bigwedge K. \text{compactin } Y \ K$ 
     $\implies \text{proper\_map } (\text{subtopology } X \ \{x \in \text{topspace } X. f \ x \in K\})$ 
      ( $\text{subtopology } Y \ K) \ f$ 
  shows proper_map X Y f
proof -
  have closed_map X Y f
    by (meson assms closed_map_into_k_space fim proper_map_def)
  with f topspace_subtopology_subset show ?thesis
    apply (simp add: proper_map_alt)
    by (smt (verit, best) Collect_cong compactin_absolute)
qed

```

```

lemma proper_map_into_k_space_eq:
  assumes k_space Y
  shows proper_map X Y f  $\longleftrightarrow$ 
     $f \in (\text{topspace } X) \rightarrow \text{topspace } Y \wedge$ 
    ( $\forall K. \text{compactin } Y \ K$ 
       $\longrightarrow \text{proper\_map } (\text{subtopology } X \ \{x \in \text{topspace } X. f \ x \in K\}) \ (\text{subtopology } Y \ K) \ f)$ 
    (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (simp add: proper_map_imp_subset_topspace proper_map_restriction)
  show ?rhs  $\implies$  ?lhs
    by (simp add: assms funcset_image proper_map_into_k_space)
qed

```

```

lemma compact_imp_proper_map:
  assumes k_space Y kc_space Y and fim:  $f \in (\text{topspace } X) \rightarrow \text{topspace } Y$ 
  and f: continuous_map X Y f  $\vee$  kc_space X
  and comp:  $\bigwedge K. \text{compactin } Y \ K \implies \text{compactin } X \ \{x \in \text{topspace } X. f \ x \in K\}$ 
  shows proper_map X Y f
proof (rule compact_imp_proper_map_gen)
  fix S
  assume S  $\subseteq$  topspace Y
    and  $\bigwedge K. \text{compactin } Y \ K \implies \text{compactin } Y \ (S \cap K)$ 

```

1332

```
with assms show closedin Y S
  by (simp add: closedin_subset_topspace inf_commute k_space kc_space_def)
qed (use assms in auto)
```

```
lemma proper_eq_compact_map:
  assumes k_space Y kc_space Y
  and f: continuous_map X Y f  $\vee$  kc_space X
  shows proper_map X Y f  $\longleftrightarrow$ 
    f  $\in$  (topspace X)  $\rightarrow$  topspace Y  $\wedge$ 
    ( $\forall$  K. compactin Y K  $\longrightarrow$  compactin X {x  $\in$  topspace X. f x  $\in$  K})
  (is ?lhs  $\longleftrightarrow$  ?rhs)
```

```
proof
  show ?lhs  $\implies$  ?rhs
  using  $\langle$ k_space Y $\rangle$  compactin_proper_map_preimage proper_map_into_k_space_eq
  by blast
qed (use assms compact_imp_proper_map in auto)
```

```
lemma compact_imp_perfect_map:
  assumes k_space Y kc_space Y and f ' (topspace X) = topspace Y
  and continuous_map X Y f
  and  $\bigwedge$  K. compactin Y K  $\implies$  compactin X {x  $\in$  topspace X. f x  $\in$  K}
  shows perfect_map X Y f
  by (simp add: assms compact_imp_proper_map perfect_map_def flip: image_subset_iff_funcset)
```

end

6.7 Abstract Metric Spaces

```
theory Abstract_Metric_Spaces
  imports Elementary_Metric_Spaces Abstract_Limits Abstract_Topological_Spaces
begin
```

```
locale Metric_space =
  fixes M :: 'a set and d :: 'a  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes nonneg [simp]:  $\bigwedge$  x y. 0  $\leq$  d x y
  assumes commute:  $\bigwedge$  x y. d x y = d y x
  assumes zero [simp]:  $\bigwedge$  x y.  $\llbracket$ x  $\in$  M; y  $\in$  M $\rrbracket \implies$  d x y = 0  $\longleftrightarrow$  x=y
  assumes triangle:  $\bigwedge$  x y z.  $\llbracket$ x  $\in$  M; y  $\in$  M; z  $\in$  M $\rrbracket \implies$  d x z  $\leq$  d x y + d y z
```

Link with the type class version

```
interpretation Met_TC: Metric_space UNIV dist
  by (simp add: dist_commute dist_triangle Metric_space.intro)
```

```
context Metric_space
begin
```

```
lemma subspace: M'  $\subseteq$  M  $\implies$  Metric_space M' d
  by (simp add: commute_in_mono Metric_space.intro triangle)
```


lemma *abs_mdistsimp*: $|d\ x\ y| = d\ x\ y$
by *simp*

lemma *mdist_pos_less*: $\llbracket x \neq y; x \in M; y \in M \rrbracket \implies 0 < d\ x\ y$
by (*metis less_eq_real_def nonneg zero*)

lemma *mdist_zero* [*simp*]: $x \in M \implies d\ x\ x = 0$
by *simp*

lemma *mdist_pos_eq* [*simp*]: $\llbracket x \in M; y \in M \rrbracket \implies 0 < d\ x\ y \longleftrightarrow x \neq y$
using *mdist_pos_less zero* **by** *fastforce*

lemma *triangle'*: $\llbracket x \in M; y \in M; z \in M \rrbracket \implies d\ x\ z \leq d\ x\ y + d\ z\ y$
by (*simp add: commute triangle*)

lemma *triangle''*: $\llbracket x \in M; y \in M; z \in M \rrbracket \implies d\ x\ z \leq d\ y\ x + d\ y\ z$
by (*simp add: commute triangle*)

lemma *mdist_reverse_triangle*: $\llbracket x \in M; y \in M; z \in M \rrbracket \implies |d\ x\ y - d\ y\ z| \leq d\ x\ z$
by (*smt (verit) commute triangle*)

Open and closed balls

definition *mball* **where** $mball\ x\ r \equiv \{y. x \in M \wedge y \in M \wedge d\ x\ y < r\}$

definition *mcball* **where** $mcball\ x\ r \equiv \{y. x \in M \wedge y \in M \wedge d\ x\ y \leq r\}$

lemma *in_mball* [*simp*]: $y \in mball\ x\ r \longleftrightarrow x \in M \wedge y \in M \wedge d\ x\ y < r$
by (*simp add: mball_def*)

lemma *centre_in_mball_iff* [*iff*]: $x \in mball\ x\ r \longleftrightarrow x \in M \wedge 0 < r$
using *in_mball mdist_zero* **by** *force*

lemma *mball_subset_mspace*: $mball\ x\ r \subseteq M$
by *auto*

lemma *mball_eq_empty*: $mball\ x\ r = \{\} \longleftrightarrow (x \notin M) \vee r \leq 0$
by (*smt (verit, best) Collect_empty_eq centre_in_mball_iff mball_def nonneg*)

lemma *mball_subset*: $\llbracket d\ x\ y + a \leq b; y \in M \rrbracket \implies mball\ x\ a \subseteq mball\ y\ b$
by (*smt (verit) commute in_mball subsetI triangle*)

lemma *disjoint_mball*: $r + r' \leq d\ x\ x' \implies disjnt\ (mball\ x\ r)\ (mball\ x'\ r')$
by (*smt (verit) commute disjnt_iff in_mball triangle*)

lemma *mball_subset_concentric*: $r \leq s \implies mball\ x\ r \subseteq mball\ x\ s$
by *auto*

lemma *in_mcball* [*simp*]: $y \in mcball\ x\ r \longleftrightarrow x \in M \wedge y \in M \wedge d\ x\ y \leq r$

1334

by (simp add: mcball_def)

lemma *centre_in_mcball_iff* [iff]: $x \in \text{mcball } x \ r \longleftrightarrow x \in M \wedge 0 \leq r$
using *mdist_zero* by force

lemma *mcball_eq_empty*: $\text{mcball } x \ r = \{\} \longleftrightarrow (x \notin M) \vee r < 0$
by (smt (verit, best) *Collect_empty_eq centre_in_mcball_iff empty_iff mcball_def nonneg*)

lemma *mcball_subset_mspace*: $\text{mcball } x \ r \subseteq M$
by auto

lemma *mball_subset_mcball*: $\text{mball } x \ r \subseteq \text{mcball } x \ r$
by auto

lemma *mcball_subset*: $\llbracket d \ x \ y + a \leq b; y \in M \rrbracket \implies \text{mcball } x \ a \subseteq \text{mcball } y \ b$
by (smt (verit) *in_mcball mdist_reverse_triangle subsetI*)

lemma *mcball_subset_concentric*: $r \leq s \implies \text{mcball } x \ r \subseteq \text{mcball } x \ s$
by force

lemma *mcball_subset_mball*: $\llbracket d \ x \ y + a < b; y \in M \rrbracket \implies \text{mcball } x \ a \subseteq \text{mball } y \ b$
by (smt (verit) *commute_in_mball in_mcball subsetI triangle*)

lemma *mcball_subset_mball_concentric*: $a < b \implies \text{mcball } x \ a \subseteq \text{mball } x \ b$
by force

end

6.7.1 Metric topology

context *Metric_space*
begin

definition *mopen where*
 $\text{mopen } U \equiv U \subseteq M \wedge (\forall x. x \in U \longrightarrow (\exists r > 0. \text{mball } x \ r \subseteq U))$

definition *mtopology :: 'a topology where*
 $\text{mtopology} \equiv \text{topology } \text{mopen}$

lemma *is_topology_metric_topology* [iff]: *istopology mopen*

proof –

have $\bigwedge S \ T. \llbracket \text{mopen } S; \text{mopen } T \rrbracket \implies \text{mopen } (S \cap T)$
by (smt (verit, del_insts) *Int_iff in_mball mopen_def subset_eq*)

moreover have $\bigwedge \mathcal{K}. (\forall K \in \mathcal{K}. \text{mopen } K) \longrightarrow \text{mopen } (\bigcup \mathcal{K})$

using *mopen_def* by fastforce

ultimately show *?thesis*

by (simp add: *istopology_def*)

qed

lemma *openin_mtopology*: $openin\ mtopology\ U \longleftrightarrow U \subseteq M \wedge (\forall x. x \in U \longrightarrow (\exists r>0. mball\ x\ r \subseteq U))$

by (*simp add: mopen_def mtopology_def*)

lemma *topspace_mtopology* [*simp*]: $topspace\ mtopology = M$

by (*meson order.refl mball_subset_mspace openin_mtopology openin_subset openin_topspace subset_antisym zero_less_one*)

lemma *subtopology_mspace* [*simp*]: $subtopology\ mtopology\ M = mtopology$

by (*metis subtopology_topspace topspace_mtopology*)

lemma *open_in_mspace* [*iff*]: $openin\ mtopology\ M$

by (*metis openin_topspace topspace_mtopology*)

lemma *closedin_mspace* [*iff*]: $closedin\ mtopology\ M$

by (*metis closedin_topspace topspace_mtopology*)

lemma *openin_mball* [*iff*]: $openin\ mtopology\ (mball\ x\ r)$

proof –

have $\bigwedge y. [x \in M; d\ x\ y < r] \Longrightarrow \exists s>0. mball\ y\ s \subseteq mball\ x\ r$

by (*metis add_diff_cancel_left' add_diff_eq commute less_add_same_cancel1 mball_subset order_refl*)

then show *?thesis*

by (*auto simp: openin_mtopology*)

qed

lemma *mtopology_base*:

$mtopology = topology(arbitrary\ union_of\ (\lambda U. \exists x \in M. \exists r>0. U = mball\ x\ r))$

proof –

have $\bigwedge S. \exists x\ r. x \in M \wedge 0 < r \wedge S = mball\ x\ r \Longrightarrow openin\ mtopology\ S$

using *openin_mball* **by** *blast*

moreover have $\bigwedge U\ x. [openin\ mtopology\ U; x \in U] \Longrightarrow \exists B. (\exists x\ r. x \in M \wedge 0 < r \wedge B = mball\ x\ r) \wedge x \in B \wedge B \subseteq U$

by (*metis centre_in_mball_iff_in_mono openin_mtopology*)

ultimately show *?thesis*

by (*smt (verit) topology_base_unique*)

qed

lemma *closedin_metric*:

$closedin\ mtopology\ C \longleftrightarrow C \subseteq M \wedge (\forall x. x \in M - C \longrightarrow (\exists r>0. disjnt\ C\ (mball\ x\ r)))$ (*is ?lhs = ?rhs*)

proof

show *?lhs* \Longrightarrow *?rhs*

unfolding *closedin_def openin_mtopology*

by (*metis Diff_disjoint disjnt_def disjnt_subset2 topspace_mtopology*)

show *?rhs* \Longrightarrow *?lhs*

unfolding *closedin_def openin_mtopology disjnt_def*

1336

by (metis Diff_subset Diff_triv Int_Diff Int_commute inf.absorb_iff2 mball_subset_mspace
topspace_mtopology)

qed

lemma closedin_mcball [iff]: closedin_mtopology (mcball x r)

proof –

have $\exists r > 0. \text{disjnt } (\text{mcball } x \ r) \ (\text{mball } y \ r)$ if $x \notin M$ for y

by (metis disjnt_empty1_gt_ex mcball_eq_empty that)

moreover have $\text{disjnt } (\text{mcball } x \ r) \ (\text{mball } y \ (d \ x \ y - r))$ if $y \in M$ and $d \ x \ y > r$ for

y

using that disjnt_iff_in_mball_in_mcball mdist_reverse_triangle by force

ultimately show ?thesis

using closedin_metric mball_subset_mspace by fastforce

qed

lemma mball_iff_mcball: $(\exists r > 0. \text{mball } x \ r \subseteq U) = (\exists r > 0. \text{mcball } x \ r \subseteq U)$

by (meson dense mball_subset_mcball mcball_subset_mball_concentric order_trans)

lemma openin_mtopology_mcball:

$\text{openin_mtopology } U \iff U \subseteq M \wedge (\forall x. x \in U \implies (\exists r. 0 < r \wedge \text{mcball } x \ r \subseteq U))$

by (simp add: mball_iff_mcball openin_mtopology)

lemma metric_derived_set_of:

$\text{mtopology_derived_set_of } S = \{x \in M. \forall r > 0. \exists y \in S. y \neq x \wedge y \in \text{mball } x \ r\}$ (is
?lhs=?rhs)

proof

show ?lhs \subseteq ?rhs

unfolding openin_mtopology_derived_set_of_def

by clarsimp (metis in_mball openin_mball openin_mtopology zero)

show ?rhs \subseteq ?lhs

unfolding openin_mtopology_derived_set_of_def

by clarify (metis subsetD topspace_mtopology)

qed

lemma metric_closure_of:

$\text{mtopology_closure_of } S = \{x \in M. \forall r > 0. \exists y \in S. y \in \text{mball } x \ r\}$

proof –

have $\bigwedge x \ r. [0 < r; x \in \text{mtopology_closure_of } S] \implies \exists y \in S. y \in \text{mball } x \ r$

by (metis centre_in_mball_iff_in_closure_of openin_mball topspace_mtopology)

moreover have $\bigwedge x \ T. [x \in M; \forall r > 0. \exists y \in S. y \in \text{mball } x \ r] \implies x \in \text{mtopology_closure_of } S$

by (smt (verit) in_closure_of_in_mball openin_mtopology subsetD topspace_mtopology)

ultimately show ?thesis

by (auto simp: in_closure_of)

qed

lemma metric_closure_of_alt:

$\text{mtopology_closure_of } S = \{x \in M. \forall r > 0. \exists y \in S. y \in \text{mcball } x \ r\}$

proof –
have $\bigwedge x r. [\forall r>0. x \in M \wedge (\exists y \in S. y \in \text{mcball } x r); 0 < r] \implies \exists y \in S. y \in M \wedge d x y < r$
by (*meson dense in_mcball le_less_trans*)
then show *?thesis*
by (*fastforce simp: metric_closure_of in_closure_of*)
qed

lemma *metric_interior_of*:
 $\text{mtopology interior_of } S = \{x \in M. \exists \varepsilon > 0. \text{mball } x \ \varepsilon \subseteq S\}$ (**is** *?lhs=?rhs*)
proof
show *?lhs* \subseteq *?rhs*
using *interior_of_maximal_eq openin_mtopology* **by** *fastforce*
show *?rhs* \subseteq *?lhs*
using *interior_of_def openin_mball* **by** *fastforce*
qed

lemma *metric_interior_of_alt*:
 $\text{mtopology interior_of } S = \{x \in M. \exists \varepsilon > 0. \text{mcball } x \ \varepsilon \subseteq S\}$
by (*fastforce simp: mball_iff_mcball metric_interior_of*)

lemma *in_interior_of_mball*:
 $x \in \text{mtopology interior_of } S \iff x \in M \wedge (\exists \varepsilon > 0. \text{mball } x \ \varepsilon \subseteq S)$
using *metric_interior_of* **by** *force*

lemma *in_interior_of_mcball*:
 $x \in \text{mtopology interior_of } S \iff x \in M \wedge (\exists \varepsilon > 0. \text{mcball } x \ \varepsilon \subseteq S)$
using *metric_interior_of_alt* **by** *force*

lemma *Hausdorff_space_mtopology*: *Hausdorff_space mtopology*
unfolding *Hausdorff_space_def*

proof *clarify*
fix *x y*
assume *x*: $x \in \text{topspace mtopology}$ **and** *y*: $y \in \text{topspace mtopology}$ **and** $x \neq y$
then have *gt0*: $d x y / 2 > 0$
by *auto*
have *disjnt* $(\text{mball } x (d x y / 2)) (\text{mball } y (d x y / 2))$
by (*simp add: disjoint_mball*)
then show $\exists U V. \text{openin mtopology } U \wedge \text{openin mtopology } V \wedge x \in U \wedge y \in V \wedge \text{disjnt } U V$
by (*metis centre_in_mball_iff gt0 openin_mball topspace_mtopology x y*)
qed

6.7.2 Bounded sets

definition *mbounded where* $\text{mbounded } S \iff (\exists x B. S \subseteq \text{mcball } x B)$

lemma *mbounded_pos*: $\text{mbounded } S \iff (\exists x B. 0 < B \wedge S \subseteq \text{mcball } x B)$

proof –

1338

have $\exists x' r'. 0 < r' \wedge S \subseteq \text{mcball } x' r'$ **if** $S \subseteq \text{mcball } x r$ **for** $x r$
by (*metis gt_ex less_eq_real_def linorder_not_le mcball_subset_concentric order_trans that*)
then show *?thesis*
by (*auto simp: mbounded_def*)
qed

lemma *mbounded_alt*:
 $\text{mbounded } S \longleftrightarrow S \subseteq M \wedge (\exists B. \forall x \in S. \forall y \in S. d \ x \ y \leq B)$
proof –
have $\bigwedge x B. S \subseteq \text{mcball } x B \implies \forall x \in S. \forall y \in S. d \ x \ y \leq 2 * B$
by (*smt (verit, best) commute in_mcball subsetD triangle*)
then show *?thesis*
apply (*auto simp: mbounded_def subset_iff*)
apply *blast+*
done
qed

lemma *mbounded_alt_pos*:
 $\text{mbounded } S \longleftrightarrow S \subseteq M \wedge (\exists B > 0. \forall x \in S. \forall y \in S. d \ x \ y \leq B)$
by (*smt (verit, del_insts) gt_ex mbounded_alt*)

lemma *mbounded_subset*: $\llbracket \text{mbounded } T; S \subseteq T \rrbracket \implies \text{mbounded } S$
by (*meson mbounded_def order_trans*)

lemma *mbounded_subset_mspace*: $\text{mbounded } S \implies S \subseteq M$
by (*simp add: mbounded_alt*)

lemma *mbounded*:
 $\text{mbounded } S \longleftrightarrow S = \{\} \vee (\forall x \in S. x \in M) \wedge (\exists y B. y \in M \wedge (\forall x \in S. d \ y \ x \leq B))$
by (*meson all_not_in_conv in_mcball mbounded_def subset_iff*)

lemma *mbounded_empty [iff]*: $\text{mbounded } \{\}$
by (*simp add: mbounded*)

lemma *mbounded_mcball*: $\text{mbounded } (\text{mcball } x r)$
using *mbounded_def* **by** *auto*

lemma *mbounded_mball [iff]*: $\text{mbounded } (\text{mball } x r)$
by (*meson mball_subset_mcball mbounded_def*)

lemma *mbounded_insert*: $\text{mbounded } (\text{insert } a \ S) \longleftrightarrow a \in M \wedge \text{mbounded } S$

proof –
have $\bigwedge y B. \llbracket y \in M; \forall x \in S. d \ y \ x \leq B \rrbracket$
 $\implies \exists y. y \in M \wedge (\exists B \geq d \ y \ a. \forall x \in S. d \ y \ x \leq B)$
by (*metis order.trans nle_le*)
then show *?thesis*

by (auto simp: mbounded)
qed

lemma *mbounded_Int*: $mbounded\ S \implies mbounded\ (S \cap T)$
by (meson inf_le1 mbounded_subset)

lemma *mbounded_Un*: $mbounded\ (S \cup T) \longleftrightarrow mbounded\ S \wedge mbounded\ T$ (is
?lhs=?rhs)

proof

assume *R*: *?rhs*

show *?lhs*

proof (cases $S = \{\} \vee T = \{\}$)

case True then show *?thesis*

using *R* by auto

next

case False

obtain $x\ y\ B\ C$ **where** $S \subseteq mball\ x\ B$ $T \subseteq mball\ y\ C$ $B > 0$ $C > 0$ $x \in M$
 $y \in M$

using *R* *mbounded_pos*

by (metis False mball_eq_empty subset_empty)

then have $S \cup T \subseteq mball\ x\ (B + C + d\ x\ y)$

by (smt (verit) commute_dual_order.trans le_supI mball_subset mdist_pos_eq)

then show *?thesis*

using *mbounded_def* by blast

qed

next

show *?lhs* \implies *?rhs*

using *mbounded_def* by auto

qed

lemma *mbounded_Union*:

$\llbracket \text{finite } \mathcal{F}; \bigwedge X. X \in \mathcal{F} \implies mbounded\ X \rrbracket \implies mbounded\ (\bigcup \mathcal{F})$

by (induction \mathcal{F} rule: finite_induct) (auto simp: mbounded_Un)

lemma *mbounded_closure_of*:

$mbounded\ S \implies mbounded\ (mtopology\ closure_of\ S)$

by (meson closedin_mball closure_of_minimal mbounded_def)

lemma *mbounded_closure_of_eq*:

$S \subseteq M \implies (mbounded\ (mtopology\ closure_of\ S) \longleftrightarrow mbounded\ S)$

by (metis closure_of_subset mbounded_closure_of mbounded_subset topology_mtopology)

lemma *maxdist_thm*:

assumes *mbounded* *S*

and $x \in S$

and $y \in S$

shows $d\ x\ y = (\text{SUP } z \in S. |d\ x\ z - d\ z\ y|)$

proof –

1340

```
have |d x z - d z y| ≤ d x y if z ∈ S for z
  by (metis all_not_in_conv assms mbounded mdist_reverse_triangle that)
moreover have d x y ≤ r
  if  $\bigwedge z. z \in S \implies |d x z - d z y| \leq r$  for r :: real
  using that assms mbounded_subset_mspace mdist_zero by fastforce
ultimately show ?thesis
  by (intro cSup_eq [symmetric]) auto
qed
```

```
lemma metric_eq_thm:  $\llbracket S \subseteq M; x \in S; y \in S \rrbracket \implies (x = y) = (\forall z \in S. d x z = d y z)$ 
  by (metis commute subset_iff zero)
```

```
lemma compactin_imp_mbounded:
  assumes compactin_mtopology S
  shows mbounded S
proof -
  have  $S \subseteq M$ 
  and com:  $\bigwedge \mathcal{U}. \llbracket \forall U \in \mathcal{U}. \text{openin\_mtopology } U; S \subseteq \bigcup \mathcal{U} \rrbracket \implies \exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$ 
  using assms by (auto simp: compactin_def mbounded_def)
  show ?thesis
  proof (cases  $S = \{\}$ )
  case False
  with  $\langle S \subseteq M \rangle$  obtain a where  $a \in S \wedge a \in M$ 
  by blast
  with  $\langle S \subseteq M \rangle$  gt_ex have  $S \subseteq \bigcup (\text{range } (\text{mball } a))$ 
  by force
  then obtain  $\mathcal{F}$  where finite  $\mathcal{F} \wedge \mathcal{F} \subseteq \text{range } (\text{mball } a) \wedge S \subseteq \bigcup \mathcal{F}$ 
  by (metis (no_types, opaque_lifting) com imageE openin_mball)
  then show ?thesis
  using mbounded_Union mbounded_subset by fastforce
qed auto
qed
```

end

```
lemma mcball_eq_cball [simp]:  $\text{Met\_TC.mcball} = \text{cball}$ 
  by force
```

```
lemma mball_eq_ball [simp]:  $\text{Met\_TC.mball} = \text{ball}$ 
  by force
```

```
lemma mopen_eq_open [simp]:  $\text{Met\_TC.mopen} = \text{open}$ 
  by (force simp: open_contains_ball Met_TC.mopen_def)
```

```
lemma limitin_iff_tendsto [iff]:  $\text{limitin } \text{Met\_TC.mtopology } \sigma \ x \ F = \text{tendsto } \sigma \ x$ 
```


F

by (*simp add: Met_TC.mtopology_def*)

lemma *mtopology_is_euclidean* [*simp*]: *Met_TC.mtopology = euclidean*
by (*simp add: Met_TC.mtopology_def*)

lemma *mbounded_iff_bounded* [*iff*]: *Met_TC.mbounded A \longleftrightarrow bounded A*
by (*metis Met_TC.mbounded UNIV_I all_not_in_conv bounded_def*)

6.7.3 Subspace of a metric space

locale *Submetric = Metric_space +*
fixes *A*
assumes *subset: A \subseteq M*

sublocale *Submetric \subseteq sub: Metric_space A d*
by (*simp add: subset subspace*)

context *Submetric*
begin

lemma *mball_submetric_eq*: *sub.mball a r = (if a \in A then A \cap mball a r else {})*
and *mcball_submetric_eq*: *sub.mcball a r = (if a \in A then A \cap mcball a r else {})*
using *subset by force+*

lemma *mtopology_submetric*: *sub.mtopology = subtopology mtopology A*
unfolding *topology_eq*

proof (*intro allI iffI*)

fix *S*

assume *openin sub.mtopology S*

then have $\exists T. \text{openin } (\text{subtopology } mtopology \ A) \ T \wedge x \in T \wedge T \subseteq S$ **if** $x \in S$ **for** x

by (*metis mball_submetric_eq openin_mball openin_subtopology_Int2 subcentre_in_mball_iff sub.openin_mtopology_subsetD that*)

then show *openin (subtopology mtopology A) S*

by (*meson openin_subopen*)

next

fix *S*

assume *openin (subtopology mtopology A) S*

then obtain *T* **where** *openin mtopology T S = T \cap A*

by (*meson openin_subtopology*)

then have *mopen T*

by (*simp add: mopen_def openin_mtopology*)

then have *sub.mopen (T \cap A)*

unfolding *sub.mopen_def mopen_def*

by (*metis inf.coboundedI2 mball_submetric_eq Int_iff $\langle S = T \cap A \rangle$ inf.bounded_iff subsetI*)

1342

```
    then show openin sub.mtopology S
      using ⟨S = T ∩ A⟩ sub.mopen_def sub.openin_mtopology by force
    qed
```

```
lemma mbounded_submetric: sub.mbounded T  $\longleftrightarrow$  mbounded T  $\wedge$  T  $\subseteq$  A
  by (meson mbounded_alt sub.mbounded_alt subset subset_trans)
```

end

```
lemma (in Metric_space) submetric_empty [iff]: Submetric M d {}
proof qed auto
```

6.7.4 Abstract type of metric spaces

```
typedef 'a metric = {(M::'a set,d). Metric_space M d}
morphisms dest_metric metric
proof -
  have Metric_space {} ( $\lambda x y. 0$ )
  by (auto simp: Metric_space_def)
  then show ?thesis
  by blast
qed
```

```
definition mspace where mspace m  $\equiv$  fst (dest_metric m)
```

```
definition mdist where mdist m  $\equiv$  snd (dest_metric m)
```

```
lemma Metric_space_mspace_mdist [iff]: Metric_space (mspace m) (mdist m)
  by (metis Product_Type.Collect_case_prodD dest_metric mdist_def mspace_def)
```

```
lemma mdist_nonneg [simp]:  $\bigwedge x y. 0 \leq \text{mdist } m \ x \ y$ 
  by (metis Metric_space_def Metric_space_mspace_mdist)
```

```
lemma mdist_commute:  $\bigwedge x y. \text{mdist } m \ x \ y = \text{mdist } m \ y \ x$ 
  by (metis Metric_space_def Metric_space_mspace_mdist)
```

```
lemma mdist_zero [simp]:  $\bigwedge x y. \llbracket x \in \text{mspace } m; y \in \text{mspace } m \rrbracket \implies \text{mdist } m \ x \ y = 0 \longleftrightarrow x=y$ 
  by (meson Metric_space.zero Metric_space_mspace_mdist)
```

```
lemma mdist_triangle:  $\bigwedge x y z. \llbracket x \in \text{mspace } m; y \in \text{mspace } m; z \in \text{mspace } m \rrbracket \implies \text{mdist } m \ x \ z \leq \text{mdist } m \ x \ y + \text{mdist } m \ y \ z$ 
  by (meson Metric_space.triangle Metric_space_mspace_mdist)
```

```
lemma (in Metric_space) mspace_metric[simp]:
  mspace (metric (M,d)) = M
  by (simp add: metric_inverse mspace_def subspace)
```

```
lemma (in Metric_space) mdist_metric[simp]:
```

$mdist (metric (M,d)) = d$
by (*simp add: mdist_def metric_inverse subspace*)

lemma *metric_collapse* [*simp*]: $metric (mspace\ m, mdist\ m) = m$
by (*simp add: dest_metric_inverse mdist_def mspace_def*)

definition *mtopology_of* :: 'a *metric* \Rightarrow 'a *topology*
where *mtopology_of* $\equiv \lambda m. Metric_space.mtopology (mspace\ m) (mdist\ m)$

lemma *topspace_mtopology_of* [*simp*]: $topspace (mtopology_of\ m) = mspace\ m$
by (*simp add: Metric_space.topspace_mtopology Metric_space_mspace_mdistspace_mtopology_of_def*)

lemma (**in** *Metric_space*) *mtopology_of* [*simp*]:
 $mtopology_of (metric (M,d)) = mtopology$
by (*simp add: mtopology_of_def*)

definition *mball_of* $\equiv \lambda m. Metric_space.mball (mspace\ m) (mdist\ m)$

lemma *in_mball_of* [*simp*]: $y \in mball_of\ m\ x\ r \longleftrightarrow x \in mspace\ m \wedge y \in mspace\ m \wedge mdist\ m\ x\ y < r$
by (*simp add: Metric_space.in_mball mball_of_def*)

lemma (**in** *Metric_space*) *mball_of* [*simp*]:
 $mball_of (metric (M,d)) = mball$
by (*simp add: mball_of_def*)

definition *mcball_of* $\equiv \lambda m. Metric_space.mcball (mspace\ m) (mdist\ m)$

lemma *in_mcball_of* [*simp*]: $y \in mcball_of\ m\ x\ r \longleftrightarrow x \in mspace\ m \wedge y \in mspace\ m \wedge mdist\ m\ x\ y \leq r$
by (*simp add: Metric_space.in_mcball mcball_of_def*)

lemma (**in** *Metric_space*) *mcball_of* [*simp*]:
 $mcball_of (metric (M,d)) = mcball$
by (*simp add: mcball_of_def*)

definition *euclidean_metric* $\equiv metric (UNIV, dist)$

lemma *mspace_euclidean_metric* [*simp*]: $mspace\ euclidean_metric = UNIV$
by (*simp add: euclidean_metric_def*)

lemma *mdist_euclidean_metric* [*simp*]: $mdist\ euclidean_metric = dist$
by (*simp add: euclidean_metric_def*)

lemma *mtopology_of_euclidean* [*simp*]: $mtopology_of\ euclidean_metric = euclidean$
by (*simp add: Met_TC.mtopology_def mtopology_of_def*)

Allows reference to the current metric space within the locale as a value

definition (in *Metric_space*) *Self* \equiv *metric* (*M*,*d*)

lemma (in *Metric_space*) *mspace_Self* [*simp*]: *mspace Self* = *M*
by (*simp add: Self_def*)

lemma (in *Metric_space*) *mdist_Self* [*simp*]: *mdist Self* = *d*
by (*simp add: Self_def*)

Subspace of a metric space

definition *submetric where*
submetric \equiv $\lambda m S. \text{metric } (S \cap \text{mspace } m, \text{mdist } m)$

lemma *mspace_submetric* [*simp*]: *mspace (submetric m S)* = *S* \cap *mspace m*
unfolding *submetric_def*
by (*meson Metric_space.subspace inf_le2 Metric_space.mspace_mdists Metric_space.mspace_metric*)

lemma *mdist_submetric* [*simp*]: *mdist (submetric m S)* = *mdist m*
unfolding *submetric_def*
by (*meson Metric_space.subspace inf_le2 Metric_space.mdists_metric Metric_space.mspace_mdists*)

lemma *submetric_UNIV* [*simp*]: *submetric m UNIV* = *m*
by (*simp add: submetric_def dest_metric_inverse mdist_def mspace_def*)

lemma *submetric_submetric* [*simp*]:
submetric (submetric m S) T = *submetric m (S* \cap *T)*
by (*metis submetric_def Int_assoc inf_commute mdist_submetric mspace_submetric*)

lemma *submetric_mspace* [*simp*]:
submetric m (mspace m) = *m*
by (*simp add: submetric_def dest_metric_inverse mdist_def mspace_def*)

lemma *submetric_restrict*:
submetric m S = *submetric m (mspace m* \cap *S)*
by (*metis submetric_mspace submetric_submetric*)

lemma *mtopology_of_submetric*: *mtopology_of (submetric m A)* = *subtopology*
(*mtopology_of m*) *A*

proof –

interpret *Submetric mspace m mdist m A* \cap *mspace m*
using *Metric_space_mspace_mdists Submetric.intro Submetric_axioms.intro*
inf_le2 **by** *blast*
have *sub.mtopology* = *subtopology (mtopology_of m) A*
by (*metis inf_commute mtopology_of_def mtopology_submetric subtopology_mspace*
subtopology_subtopology)
then show *?thesis*
by (*simp add: submetric_def*)

qed

6.7.5 The discrete metric

locale *discrete_metric* =
 fixes $M :: 'a \text{ set}$

definition (in *discrete_metric*) $dd :: 'a \Rightarrow 'a \Rightarrow \text{real}$
 where $dd \equiv \lambda x y :: 'a. \text{if } x=y \text{ then } 0 \text{ else } 1$

lemma *metric_M_dd*: *Metric_space* M *discrete_metric.dd*
 by (*simp add: discrete_metric.dd_def Metric_space.intro*)

sublocale *discrete_metric* \subseteq *disc*: *Metric_space* M *dd*
 by (*simp add: metric_M_dd*)

lemma (in *discrete_metric*) *mopen_singleton*:
 assumes $x \in M$ **shows** *disc.mopen* $\{x\}$

proof –

have $\text{disc.mball } x \ (1/2) \subseteq \{x\}$
 by (*smt (verit) dd_def disc.in_mball less_divide_eq_1_pos singleton_iff subsetI*)

with *assms* **show** *?thesis*

using *disc.mopen_def half_gt_zero_iff zero_less_one* **by** *blast*

qed

lemma (in *discrete_metric*) *mtopology_discrete_metric*:
 $\text{disc.mtopology} = \text{discrete_topology } M$

proof –

have $\bigwedge x. x \in M \implies \text{openin } \text{disc.mtopology } \{x\}$

by (*simp add: disc.mtopology_def mopen_singleton*)

then **show** *?thesis*

by (*metis disc.topspace_mtopology discrete_topology_unique*)

qed

lemma (in *discrete_metric*) *discrete_ultrametric*:

$dd \ x \ z \leq \max (dd \ x \ y) (dd \ y \ z)$

by (*simp add: dd_def*)

lemma (in *discrete_metric*) *dd_le1*: $dd \ x \ y \leq 1$

by (*simp add: dd_def*)

lemma (in *discrete_metric*) *mbounded_discrete_metric*: $\text{disc.mbounded } S \longleftrightarrow S \subseteq M$

by (*meson dd_le1 disc.mbounded_alt*)

6.7.6 Metrizable spaces

definition *metrizable_space* **where**

$\text{metrizable_space } X \equiv \exists M \ d. \text{Metric_space } M \ d \wedge X = \text{Metric_space.mtopology}$

1346

M d

lemma (in *Metric_space*) *metrizable_space_mtopology*: *metrizable_space mtopology*
oggy
using *local.Metric_space_axioms metrizable_space_def* by *blast*

lemma (in *Metric_space*) *first_countable_mtopology*: *first_countable mtopology*
proof (*clarsimp simp add: first_countable_def*)
fix *x*
assume $x \in M$
define \mathcal{B} where $\mathcal{B} \equiv \text{mball } x \text{ ' } \{r \in \mathbb{Q}. 0 < r\}$
show $\exists \mathcal{B}. \text{countable } \mathcal{B} \wedge (\forall V \in \mathcal{B}. \text{openin } mtopology \ V) \wedge (\forall U. \text{openin } mtopology \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U))$
proof (*intro exI conjI ballI*)
show *countable* \mathcal{B}
by (*simp add: B_def countable_rat*)
show $\forall U. \text{openin } mtopology \ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U)$
proof *clarify*
fix *U*
assume *openin mtopology U* and $x \in U$
then obtain *r* where $r > 0$ and $r: \text{mball } x \ r \subseteq U$
by (*meson openin_mtopology*)
then obtain *q* where $q \in \text{Rats } 0 < q < r$
using *Rats_dense_in_real* by *blast*
then show $\exists V \in \mathcal{B}. x \in V \wedge V \subseteq U$
unfolding \mathcal{B}_def using $\langle x \in M \rangle r$ by *fastforce*
qed
qed (*auto simp: B_def*)
qed

lemma *metrizable_imp_first_countable*:
metrizable_space X \implies *first_countable X*
by (*force simp add: metrizable_space_def Metric_space.first_countable_mtopology*)

lemma *openin_mtopology_eq_open* [*simp*]: *openin Met_TC.mtopology = open*
by (*simp add: Met_TC.mtopology_def*)

lemma *closedin_mtopology_eq_closed* [*simp*]: *closedin Met_TC.mtopology = closed*
proof –
have (*euclidean::'a topology*) = *Met_TC.mtopology*
by (*simp add: Met_TC.mtopology_def*)
then show *?thesis*
using *closed_closedin* by *fastforce*
qed

lemma *compactin_mtopology_eq_compact* [*simp*]: *compactin Met_TC.mtopology = compact*
by (*simp add: compactin_def compact_eq_Heine_Borel_fun_eq_iff*) *meson*

lemma *metrizable_space_discrete_topology* [*simp*]:
metrizable_space(*discrete_topology* *U*)
by (*metis* *discrete_metric.mtopology_discrete_metric* *metric_M_dd* *metrizable_space_def*)

lemma *empty_metrizable_space*: *metrizable_space* *trivial_topology*
by *simp*

lemma *metrizable_space_subtopology*:
assumes *metrizable_space* *X*
shows *metrizable_space*(*subtopology* *X* *S*)
proof –
obtain *M d* **where** *Metric_space* *M d* **and** *X*: *X* = *Metric_space.mtopology* *M*
d
using *assms* *metrizable_space_def* **by** *blast*
then interpret *Submetric* *M d* *M* \cap *S*
by (*simp* *add*: *Submetric.intro* *Submetric_axioms_def*)
show *?thesis*
unfolding *metrizable_space_def*
by (*metis* *X* *mtopology_submetric* *sub.Metric_space_axioms* *subtopology_restrict*
topspace_mtopology)
qed

lemma *homeomorphic_metrizable_space_aux*:
assumes *X* *homeomorphic_space* *Y* *metrizable_space* *X*
shows *metrizable_space* *Y*
proof –
obtain *M d* **where** *Metric_space* *M d* **and** *X*: *X* = *Metric_space.mtopology* *M*
d
using *assms* **by** (*auto* *simp*: *metrizable_space_def*)
then interpret *m*: *Metric_space* *M d*
by *simp*
obtain *f g* **where** *hmf*: *homeomorphic_map* *X* *Y* *f* **and** *hmg*: *homeomorphic_map*
Y *X* *g*
and *fg*: $(\forall x \in M. g(f x) = x) \wedge (\forall y \in \text{topspace } Y. f(g y) = y)$
using *assms* *X* *homeomorphic_maps_map* *homeomorphic_space_def* **by** *fast-*
force
define *d'* **where** $d' x y \equiv d (g x) (g y)$ **for** *x y*
interpret *m'*: *Metric_space* *topspace* *Y* *d'*
unfolding *d'_def*
proof
show $(d (g x) (g y) = 0) = (x = y)$ **if** $x \in \text{topspace } Y$ $y \in \text{topspace } Y$ **for** *x y*
by (*metis* *fg* *X* *hmg* *homeomorphic_imp_surjective_map* *imageI* *m.topspace_mtopology*
m.zero *that*)
show $d (g x) (g z) \leq d (g x) (g y) + d (g y) (g z)$
if $x \in \text{topspace } Y$ **and** $y \in \text{topspace } Y$ **and** $z \in \text{topspace } Y$ **for** *x y z*
by (*metis* *X* *that* *hmg* *homeomorphic_eq_everything_map* *imageI* *m.topspace_mtopology*
m.triangle)
qed (*auto* *simp*: *m.nonneg* *m.commute*)
have *Y* = *Metric_space.mtopology* (*topspace* *Y*) *d'*

```

    unfolding topology_eq
  proof (intro allI)
    fix S
    have openin m'.mtopology S if S: S ⊆ topspace Y and openin X (g ' S)
      unfolding m'.openin_mtopology
    proof (intro conjI that strip)
      fix y
      assume y ∈ S
      then obtain r where r > 0 and r: m.mball (g y) r ⊆ g ' S
        using X ⟨openin X (g ' S)⟩ m.openin_mtopology using ⟨y ∈ S⟩ by auto
      then have g ' m'.mball y r ⊆ m.mball (g y) r
        using X d'_def hmg homeomorphic_imp_surjective_map by fastforce
      with S fg have m'.mball y r ⊆ S
        by (smt (verit, del_insts) image_iff m'.in_mball r subset_iff)
      then show ∃ r > 0. m'.mball y r ⊆ S
        using ⟨0 < r⟩ by blast
    qed
  moreover have openin X (g ' S) if ope': openin m'.mtopology S
  proof -
    have ∃ r > 0. m.mball (g y) r ⊆ g ' S if y ∈ S for y
    proof -
      have y: y ∈ topspace Y
        using m'.openin_mtopology ope' that by blast
      obtain r where r > 0 and r: m'.mball y r ⊆ S
        using ope' by (meson ⟨y ∈ S⟩ m'.openin_mtopology)
      moreover have ∧ x. [x ∈ M; d (g y) x < r] ⇒ ∃ u. u ∈ topspace Y ∧ d'
        y u < r ∧ x = g u
        using fg X d'_def hmf homeomorphic_imp_surjective_map by fastforce
      ultimately have m.mball (g y) r ⊆ g ' m'.mball y r
        using y by (force simp: m'.openin_mtopology)
      then show ?thesis
        using ⟨0 < r⟩ r by blast
    qed
    then show ?thesis
      using X hmg homeomorphic_imp_surjective_map m.openin_mtopology ope'
      openin_subset by fastforce
  qed
  ultimately have (S ⊆ topspace Y ∧ openin X (g ' S)) = openin m'.mtopology
  S
    using m'.topspace_mtopology openin_subset by blast
  then show openin Y S = openin m'.mtopology S
    by (simp add: m'.mopen_def homeomorphic_map_openness_eq [OF hmg])
  qed
  then show ?thesis
    using m'.metrizable_space_mtopology by force
  qed

```

lemma homeomorphic_metrizable_space:
 assumes X homeomorphic_space Y

shows $\text{metrizable_space } X \longleftrightarrow \text{metrizable_space } Y$
using *assms* $\text{homeomorphic_metrizable_space_aux}$ $\text{homeomorphic_space_sym}$
by *metis*

lemma *metrizable_space_retraction_map_image*:
 $\text{retraction_map } X Y r \wedge \text{metrizable_space } X$
 $\implies \text{metrizable_space } Y$
using *hereditary_imp_retractive_property* $\text{metrizable_space_subtopology}$ *homeomorphic_metrizable_space*
by *blast*

lemma *metrizable_imp_Hausdorff_space*:
 $\text{metrizable_space } X \implies \text{Hausdorff_space } X$
by (*metis* $\text{Metric_space.Hausdorff_space_mtopology}$ $\text{metrizable_space_def}$)

lemma *metrizable_imp_t1_space*:
 $\text{metrizable_space } X \implies \text{t1_space } X$
by (*simp* *add*: $\text{Hausdorff_imp_t1_space}$ $\text{metrizable_imp_Hausdorff_space}$)

lemma *closed_imp_gdelta_in*:
assumes X : $\text{metrizable_space } X$ **and** S : $\text{closedin } X S$
shows $\text{gdelta_in } X S$
proof –
obtain $M d$ **where** $\text{Metric_space } M d$ **and** $X \text{eq}$: $X = \text{Metric_space.mtopology } M d$
using X $\text{metrizable_space_def}$ **by** *blast*
then interpret M : $\text{Metric_space } M d$
by *blast*
have $S \subseteq M$
using $M.\text{closedin_metric } \langle X = M.\text{mtopology} \rangle S$ **by** *blast*
show *?thesis*
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
by *simp*
next
case *False*
have $\exists y \in S. d x y < \text{inverse } (1 + \text{real } n)$ **if** $x \in S$ **for** $x n$
using $\langle S \subseteq M \rangle M.\text{mdist_zero } [\text{of } x]$ **that** **by** *force*
moreover
have $x \in S$ **if** $x \in M$ **and** \S : $\bigwedge n. \exists y \in S. d x y < \text{inverse}(\text{Suc } n)$ **for** x
proof –
have $*$: $\exists y \in S. d x y < \varepsilon$ **if** $\varepsilon > 0$ **for** ε
by (*metis* \S *that* $\text{not0_implies_Suc_order_less_le}$ $\text{order_less_le_trans}$ real_arch_inverse)
have $\text{closedin } M.\text{mtopology } S$

```

    using S by (simp add: Xeq)
  then show ?thesis
    apply (simp add: M.closedin_metric)
    by (metis * ⟨x ∈ M⟩ M.in_mball disjnt_insert1 insert_absorb subsetD)
  qed
  ultimately have Seq: S = ⋂(range (λn. {x∈M. ∃y∈S. d x y < inverse(Suc
n)}))
    using ⟨S ⊆ M⟩ by force
  have openin M.mtopology {xa ∈ M. ∃y∈S. d xa y < inverse (1 + real n)} for
n
  proof (clarsimp simp: M.openin_mtopology)
    fix x y
    assume x ∈ M y ∈ S and dxy: d x y < inverse (1 + real n)
    then have ∧z. [z ∈ M; d x z < inverse (1 + real n) - d x y] ⇒ ∃y∈S. d
z y < inverse (1 + real n)
      by (smt (verit) M.commute M.triangle ⟨S ⊆ M⟩ in_mono)
    with dxy show ∃r>0. M.mball x r ⊆ {z ∈ M. ∃y∈S. d z y < inverse (1 +
real n)}
      by (rule_tac x=inverse(Suc n) - d x y in exI) auto
  qed
  then show ?thesis
    apply (subst Seq)
    apply (force simp: Xeq intro: gdelta_in_Inter open_imp_gdelta_in)
  done
  qed
qed

```

```

lemma open_imp_fsigma_in:
  [metrizable_space X; openin X S] ⇒ fsigma_in X S
  by (meson closed_imp_gdelta_in fsigma_in_gdelta_in openin_closedin openin_subset)

```

```

lemma metrizable_space_euclidean:
  metrizable_space (euclidean :: 'a::metric_space topology)
  using Met_TC.metrizable_space_mtopology by auto

```

```

lemma (in Metric_space) regular_space_mtopology:
  regular_space mtopology
  unfolding regular_space_def
  proof clarify
    fix C a
    assume C: closedin mtopology C and a: a ∈ topspace mtopology and a ∉ C
    have openin mtopology (topspace mtopology - C)
      by (simp add: C openin_diff)
    then obtain r where r>0 and r: mball a r ⊆ topspace mtopology - C
      unfolding openin_mtopology using ⟨a ∉ C⟩ a by auto
    show ∃U V. openin mtopology U ∧ openin mtopology V ∧ a ∈ U ∧ C ⊆ V ∧
disjnt U V
      proof (intro exI conjI)
        show a ∈ mball a (r/2)

```

```

    using <0 < r> a by force
    show C ⊆ topspace mtopology - mball a (r/2)
    using C <0 < r> r by (fastforce simp: closedin_metric)
  qed (auto simp: openin_mball closedin_mball openin_diff disjnt_iff)
qed

```

```

lemma metrizable_imp_regular_space:
  metrizable_space X ⇒ regular_space X
  by (metis Metric_space.regular_space_mtopology metrizable_space_def)

```

```

lemma regular_space_euclidean:
  regular_space (euclidean :: 'a::metric_space topology)
  by (simp add: metrizable_imp_regular_space metrizable_space_euclidean)

```

6.7.7 Limits at a point in a topological space

```

lemma (in Metric_space) eventually_atin_metric:
  eventually P (atin mtopology a) ⇔
    (a ∈ M → (∃ δ > 0. ∀ x. x ∈ M ∧ 0 < d x a ∧ d x a < δ → P x)) (is
  ?lhs=?rhs)
proof (cases a ∈ M)
  case True
  show ?thesis
  proof
    assume L: ?lhs
    with True obtain U where openin mtopology U a ∈ U and U: ∀ x ∈ U - {a}.
  P x
    by (auto simp: eventually_atin)
    then obtain r where r > 0 and mball a r ⊆ U
    by (meson openin_mtopology)
    with U show ?rhs
    by (smt (verit, ccfv_SIG) commute_in_mball insert_Diff_single insert_iff
  subset_iff)
  next
    assume ?rhs
    then obtain δ where δ > 0 and δ: ∀ x. x ∈ M ∧ 0 < d x a ∧ d x a < δ →
  P x
    using True by blast
    then have ∀ x ∈ mball a δ - {a}. P x
    by (simp add: commute)
    then show ?lhs
    unfolding eventually_atin openin_mtopology
    by (metis True <0 < δ> centre_in_mball_iff openin_mball openin_mtopology)
  qed
qed
qed auto

```

6.7.8 Normal spaces and metric spaces

```

lemma (in Metric_space) normal_space_mtopology:

```

```

    normal_space mtopology
  unfolding normal_space_def
proof clarify
  fix S T
  assume closedin mtopology S
  then have  $\bigwedge x. x \in M - S \implies (\exists r > 0. \text{mball } x \ r \subseteq M - S)$ 
    by (simp add: closedin_def openin_mtopology)
  then obtain  $\delta$  where  $d0: \bigwedge x. x \in M - S \implies \delta \ x > 0 \wedge \text{mball } x \ (\delta \ x) \subseteq M - S$ 
    by metis
  assume closedin mtopology T
  then have  $\bigwedge x. x \in M - T \implies (\exists r > 0. \text{mball } x \ r \subseteq M - T)$ 
    by (simp add: closedin_def openin_mtopology)
  then obtain  $\varepsilon$  where  $e: \bigwedge x. x \in M - T \implies \varepsilon \ x > 0 \wedge \text{mball } x \ (\varepsilon \ x) \subseteq M - T$ 
    by metis
  assume disjnt S T
  have  $S \subseteq M \ T \subseteq M$ 
  using  $\langle \text{closedin mtopology } S \rangle \langle \text{closedin mtopology } T \rangle \text{closedin\_metric}$  by blast+
  have  $\delta: \bigwedge x. x \in T \implies \delta \ x > 0 \wedge \text{mball } x \ (\delta \ x) \subseteq M - S$ 
    by (meson DiffI  $\langle T \subseteq M \rangle \langle \text{disjnt } S \ T \rangle d0 \text{disjnt\_iff subsetD}$ )
  have  $\varepsilon: \bigwedge x. x \in S \implies \varepsilon \ x > 0 \wedge \text{mball } x \ (\varepsilon \ x) \subseteq M - T$ 
    by (meson Diff\_iff  $\langle S \subseteq M \rangle \langle \text{disjnt } S \ T \rangle \text{disjnt\_iff } e \text{subsetD}$ )
  show  $\exists U \ V. \text{openin mtopology } U \wedge \text{openin mtopology } V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U \ V$ 
  proof (intro exI conjI)
    show  $\text{openin mtopology } (\bigcup_{x \in S. \text{mball } x \ (\varepsilon \ x / 2)}) \text{openin mtopology } (\bigcup_{x \in T. \text{mball } x \ (\delta \ x / 2)})$ 
      by force+
    show  $S \subseteq (\bigcup_{x \in S. \text{mball } x \ (\varepsilon \ x / 2)})$ 
      using  $\varepsilon \langle S \subseteq M \rangle$  by force
    show  $T \subseteq (\bigcup_{x \in T. \text{mball } x \ (\delta \ x / 2)})$ 
      using  $\delta \langle T \subseteq M \rangle$  by force
    show  $\text{disjnt } (\bigcup_{x \in S. \text{mball } x \ (\varepsilon \ x / 2)}) (\bigcup_{x \in T. \text{mball } x \ (\delta \ x / 2)})$ 
      using  $\varepsilon \ \delta$ 
      apply (clarsimp simp: disjnt\_iff subset\_iff)
      by (smt (verit, ccfv\_SIG) field\_sum\_of\_halves\_triangle')
  qed
qed

```

```

lemma metrizable_imp_normal_space:
  metrizable_space X  $\implies$  normal_space X
  by (metis Metric_space.normal_space_mtopology metrizable_space_def)

```

6.7.9 Topological limit in metric spaces

```

lemma (in Metric_space) limitin_mspace:
  limitin mtopology f l F  $\implies$  l  $\in$  M
  using limitin_topospace by fastforce

```

lemma (in *Metric_space*) *limitin_metric_unique*:

$\llbracket \text{limitin_mtopology } f \text{ } l1 \text{ } F; \text{ limitin_mtopology } f \text{ } l2 \text{ } F; F \neq \text{bot} \rrbracket \implies l1 = l2$
by (*meson Hausdorff_space_mtopology limitin_Hausdorff_unique*)

lemma (in *Metric_space*) *limitin_metric*:

$\text{limitin_mtopology } f \text{ } l \text{ } F \longleftrightarrow l \in M \wedge (\forall \varepsilon > 0. \text{eventually } (\lambda x. f \text{ } x \in M \wedge d \text{ } (f \text{ } x) \text{ } l < \varepsilon) \text{ } F)$
(is ?lhs=?rhs)

proof

assume *L*: ?lhs

show ?rhs

unfolding *limitin_def*

proof (*intro conjI strip*)

show $l \in M$

using *L limitin_mspace* **by** *blast*

fix $\varepsilon :: \text{real}$

assume $\varepsilon > 0$

then have $\forall_F x \text{ in } F. f \text{ } x \in \text{mball } l \ \varepsilon$

using *L openin_mball* **by** (*fastforce simp: limitin_def*)

then show $\forall_F x \text{ in } F. f \text{ } x \in M \wedge d \text{ } (f \text{ } x) \text{ } l < \varepsilon$

using *commute eventually_mono* **by** *fastforce*

qed

next

assume *R*: ?rhs

then show ?lhs

by (*force simp: limitin_def commute openin_mtopology subset_eq elim: eventually_mono*)

qed

lemma (in *Metric_space*) *limit_metric_sequentially*:

$\text{limitin_mtopology } f \text{ } l \text{ } \text{sequentially} \longleftrightarrow$

$l \in M \wedge (\forall \varepsilon > 0. \exists N. \forall n \geq N. f \text{ } n \in M \wedge d \text{ } (f \text{ } n) \text{ } l < \varepsilon)$

by (*auto simp: limitin_metric eventually_sequentially*)

lemma (in *Submetric*) *limitin_submetric_iff*:

$\text{limitin_sub.mtopology } f \text{ } l \text{ } F \longleftrightarrow$

$l \in A \wedge \text{eventually } (\lambda x. f \text{ } x \in A) \text{ } F \wedge \text{limitin_mtopology } f \text{ } l \text{ } F$ **(is ?lhs=?rhs)**

by (*simp add: limitin_subtopology mtopology_submetric*)

lemma (in *Metric_space*) *metric_closedin_iff_sequentially_closed*:

$\text{closedin_mtopology } S \longleftrightarrow$

$S \subseteq M \wedge (\forall \sigma \text{ l. range } \sigma \subseteq S \wedge \text{limitin_mtopology } \sigma \text{ } l \text{ } \text{sequentially} \longrightarrow l \in S)$

(is ?lhs=?rhs)

proof

assume ?lhs **then show** ?rhs

by (*force simp: closedin_metric limitin_closedin range_subsetD*)

next

assume *R*: ?rhs

```

show ?lhs
  unfolding closedin_metric
proof (intro conjI strip)
  show  $S \subseteq M$ 
    using R by blast
  fix x
  assume  $x \in M - S$ 
  have False if  $\forall r > 0. \exists y. y \in M \wedge y \in S \wedge d x y < r$ 
  proof -
    have  $\forall n. \exists y. y \in M \wedge y \in S \wedge d x y < \text{inverse}(\text{Suc } n)$ 
      using that by auto
    then obtain  $\sigma$  where  $\sigma: \bigwedge n. \sigma n \in M \wedge \sigma n \in S \wedge d x (\sigma n) < \text{inverse}(\text{Suc } n)$ 
      by metis
    then have range  $\sigma \subseteq M$ 
      by blast
    have  $\exists N. \forall n \geq N. d x (\sigma n) < \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
    proof -
      have real (Suc (nat [inverse  $\varepsilon$ ]))  $\geq$  inverse  $\varepsilon$ 
        by linarith
      then have  $\forall n \geq \text{nat } [\text{inverse } \varepsilon]. d x (\sigma n) < \varepsilon$ 
        by (metis  $\sigma$  inverse_inverse_eq inverse_le_imp_le nat_ceiling_le_eq
          le_le_not_less_eq_eq order.strict_trans2 that)
      then show ?thesis ..
    qed
    with  $\sigma$  have limitin_mtopology  $\sigma$  x sequentially
      using  $\langle x \in M - S \rangle$  commute_limit_metric_sequentially by auto
    then show ?thesis
      by (metis R DiffD2  $\sigma$  image_subset_iff  $\langle x \in M - S \rangle$ )
    qed
  then show  $\exists r > 0. \text{disjnt } S (\text{mball } x r)$ 
    by (meson disjnt_iff_in_mball)
  qed
qed

```

lemma (in Metric_space) limit_atin_metric:

$$\text{limitin } X f y (\text{atin_mtopology } x) \longleftrightarrow$$

$$y \in \text{topspace } X \wedge$$

$$(x \in M$$

$$\longrightarrow (\forall V. \text{openin } X V \wedge y \in V$$

$$\longrightarrow (\exists \delta > 0. \forall x'. x' \in M \wedge 0 < d x' x \wedge d x' x < \delta \longrightarrow f x' \in V)))$$

by (force simp: limitin_def eventually_atin_metric)

lemma (in Metric_space) limitin_metric_dist_null:

$$\text{limitin_mtopology } f l F \longleftrightarrow l \in M \wedge \text{eventually } (\lambda x. f x \in M) F \wedge ((\lambda x. d (f x) l) \longrightarrow 0) F$$

by (simp add: limitin_metric tendsto_iff eventually_conj_iff all_conj_distrib imp_conjR gt_ex)

6.7.10 Cauchy sequences and complete metric spaces

context *Metric_space*
begin

definition *MCauchy* :: (nat \Rightarrow 'a) \Rightarrow bool
where *MCauchy* $\sigma \equiv \text{range } \sigma \subseteq M \wedge (\forall \varepsilon > 0. \exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma n) (\sigma n') < \varepsilon)$

definition *mcomplete*
where *mcomplete* $\equiv (\forall \sigma. \text{MCauchy } \sigma \longrightarrow (\exists x. \text{limitin mtopology } \sigma x \text{ sequentially}))$

lemma *mcomplete_empty [iff]*: *Metric_space.mcomplete* {} *d*
by (*simp add: Metric_space.MCauchy_def Metric_space.mcomplete_def subspace*)

lemma *MCauchy_imp_MCauchy_suffix*: *MCauchy* $\sigma \implies \text{MCauchy } (\sigma \circ (+)n)$
unfolding *MCauchy_def image_subset_iff comp_apply*
by (*metis UNIV_I add.commute trans_le_add1*)

lemma *mcomplete*:
mcomplete \longleftrightarrow
 $(\forall \sigma. (\forall_F n \text{ in sequentially. } \sigma n \in M) \wedge$
 $(\forall \varepsilon > 0. \exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma n) (\sigma n') < \varepsilon) \longrightarrow$
 $(\exists x. \text{limitin mtopology } \sigma x \text{ sequentially}))$ (**is** ?lhs=?rhs)

proof

assume *L*: ?lhs

show ?rhs

proof *clarify*

fix σ

assume $\forall_F n \text{ in sequentially. } \sigma n \in M$

and $\sigma: \forall \varepsilon > 0. \exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma n) (\sigma n') < \varepsilon$

then obtain *N* **where** $\bigwedge n. n \geq N \implies \sigma n \in M$

by (*auto simp: eventually_sequentially*)

with σ **have** *MCauchy* $(\sigma \circ (+)N)$

unfolding *MCauchy_def image_subset_iff comp_apply* **by** (*meson le_add1 trans_le_add2*)

then obtain *x* **where** *limitin mtopology* $(\sigma \circ (+)N) x$ *sequentially*

using *L* *MCauchy_imp_MCauchy_suffix mcomplete_def* **by** *blast*

then have *limitin mtopology* σx *sequentially*

unfolding *o_def* **by** (*auto simp: add.commute limitin_sequentially_offset_rev*)

then show $\exists x. \text{limitin mtopology } \sigma x \text{ sequentially}$..

qed

qed (*simp add: mcomplete_def MCauchy_def image_subset_iff*)

lemma *mcomplete_empty_mspace*: $M = \{\}$ \implies *mcomplete*
using *MCauchy_def mcomplete_def* **by** *blast*

lemma *MCauchy_const [simp]*: *MCauchy* $(\lambda n. a) \longleftrightarrow a \in M$

using *MCauchy_def mdist_zero* by *auto*

lemma *convergent_imp_MCauchy*:

assumes *range* $\sigma \subseteq M$ **and** *lim*: *limitin* *mtopology* σ *l* *sequentially*

shows *MCauchy* σ

unfolding *MCauchy_def image_subset_iff*

proof (*intro conjI strip*)

fix $\varepsilon::\text{real}$

assume $\varepsilon > 0$

then have $\forall_F n$ *in sequentially*. $\sigma n \in M \wedge d(\sigma n) l < \varepsilon/2$

using *half_gt_zero lim limitin_metric* **by** *blast*

then obtain N **where** $\bigwedge n. n \geq N \implies \sigma n \in M \wedge d(\sigma n) l < \varepsilon/2$

by (*force simp: eventually_sequentially*)

then show $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d(\sigma n) (\sigma n') < \varepsilon$

by (*smt (verit) limitin_mspace mdist_reverse_triangle_field_sum_of_halves*
lim)

qed (*use assms in blast*)

lemma *mcomplete_alt*:

mcomplete $\longleftrightarrow (\forall \sigma. \text{MCauchy } \sigma \longleftrightarrow \text{range } \sigma \subseteq M \wedge (\exists x. \text{limitin } \text{mtopology } \sigma$
x sequentially))

using *MCauchy_def convergent_imp_MCauchy mcomplete_def* **by** *blast*

lemma *MCauchy_subsequence*:

assumes *strict_mono* r *MCauchy* σ

shows *MCauchy* $(\sigma \circ r)$

proof –

have $d(\sigma(r n)) (\sigma(r n')) < \varepsilon$

if $N \leq n \wedge N \leq n'$ *strict_mono* $r \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d(\sigma n) (\sigma n')$
 $< \varepsilon$

for $\varepsilon N n n'$

using *that* **by** (*meson le_trans strict_mono_imp_increasing*)

moreover have *range* $(\lambda x. \sigma(r x)) \subseteq M$

using *MCauchy_def assms* **by** *blast*

ultimately show *?thesis*

using *assms* **by** (*simp add: MCauchy_def*) *metis*

qed

lemma *MCauchy_offset*:

assumes *cau*: *MCauchy* $(\sigma \circ (+)k)$ **and** $\sigma: \bigwedge n. n < k \implies \sigma n \in M$

shows *MCauchy* σ

unfolding *MCauchy_def image_subset_iff*

proof (*intro conjI strip*)

fix n

show $\sigma n \in M$

using *assms*

unfolding *MCauchy_def image_subset_iff*

by (*metis UNIV_I comp_apply le_iff_add linorder_not_le*)


```

next
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  obtain  $N$  where  $\forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d ((\sigma \circ (+)k) n) ((\sigma \circ (+)k) n') < \varepsilon$ 
  using cau  $\langle \varepsilon > 0 \rangle$  by (fastforce simp: MCauchy_def)
  then show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma n) (\sigma n') < \varepsilon$ 
  unfolding o_def
  apply (rule_tac x=k+N in exI)
  by (smt (verit, del_insts) add.assoc le_add1 less_eqE)
qed

```

lemma *MCauchy_convergent_subsequence:*

```

  assumes cau: MCauchy  $\sigma$  and strict_mono  $r$ 
  and lim: limitin mtopology  $(\sigma \circ r)$  a sequentially
  shows limitin mtopology  $\sigma$  a sequentially
  unfolding limitin_metric
proof (intro conjI strip)
  show  $a \in M$ 
  by (meson assms limitin_mspace)
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  then obtain  $N1$  where  $N1: \bigwedge n n'. \llbracket n \geq N1; n' \geq N1 \rrbracket \implies d (\sigma n) (\sigma n') < \varepsilon/2$ 
  using cau unfolding MCauchy_def by (meson half_gt_zero)
  obtain  $N2$  where  $N2: \bigwedge n. n \geq N2 \implies (\sigma \circ r) n \in M \wedge d ((\sigma \circ r) n) a < \varepsilon/2$ 
  by (metis (no_types, lifting) lim  $\langle \varepsilon > 0 \rangle$  half_gt_zero limit_metric_sequentially)
  have  $\sigma n \in M \wedge d (\sigma n) a < \varepsilon$  if  $n \geq \max N1 N2$  for  $n$ 
  proof (intro conjI)
    show  $\sigma n \in M$ 
    using MCauchy_def cau by blast
    have  $N1 \leq r n$ 
    by (meson  $\langle \text{strict\_mono } r \rangle$  le_trans max.cobounded1 strict_mono_imp_increasing that)
    then show  $d (\sigma n) a < \varepsilon$ 
    using  $N1$  [of n r n]  $N2$  [of n]  $\langle \sigma n \in M \rangle \langle a \in M \rangle$  triangle that by fastforce
  qed
  then show  $\forall_F n$  in sequentially.  $\sigma n \in M \wedge d (\sigma n) a < \varepsilon$ 
  using eventually_sequentially by blast
qed

```

lemma *MCauchy_interleaving_gen:*

```

  MCauchy  $(\lambda n. \text{if even } n \text{ then } x(n \text{ div } 2) \text{ else } y(n \text{ div } 2)) \longleftrightarrow$ 
  (MCauchy  $x \wedge$  MCauchy  $y \wedge (\lambda n. d (x n) (y n)) \longrightarrow 0$ ) (is ?lhs=?rhs)
proof
  assume L: ?lhs
  have evens: strict_mono  $(\lambda n::\text{nat}. 2 * n)$  and odds: strict_mono  $(\lambda n::\text{nat}. \text{Suc } (2 * n))$ 
  by (auto simp: strict_mono_def)
  show ?rhs

```

```

proof (intro conjI)
  show MCauchy x MCauchy y
    using MCauchy_subsequence [OF evens L] MCauchy_subsequence [OF odds
L] by (auto simp: o_def)
  show ( $\lambda n. d (x n) (y n)$ )  $\longrightarrow 0$ 
    unfolding LIMSEQ_iff
  proof (intro strip)
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    then obtain N where N:
       $\bigwedge n n'. \llbracket n \geq N; n' \geq N \rrbracket \implies d (\text{if even } n \text{ then } x (n \text{ div } 2) \text{ else } y (n \text{ div } 2))$ 
         $(\text{if even } n' \text{ then } x (n' \text{ div } 2) \text{ else } y (n' \text{ div } 2)) < \varepsilon$ 
    using L MCauchy_def by fastforce
    have  $d (x n) (y n) < \varepsilon$  if  $n \geq N$  for n
      using N [of 2*n Suc(2*n)] that by auto
    then show  $\exists N. \forall n \geq N. \text{norm } (d (x n) (y n) - 0) < \varepsilon$ 
      by auto
  qed
qed
next
  assume R: ?rhs
  show ?lhs
    unfolding MCauchy_def
  proof (intro conjI strip)
    show range ( $\lambda n. \text{if even } n \text{ then } x (n \text{ div } 2) \text{ else } y (n \text{ div } 2)$ )  $\subseteq M$ 
      using R by (auto simp: MCauchy_def)
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    obtain Nx where Nx:  $\bigwedge n n'. \llbracket n \geq Nx; n' \geq Nx \rrbracket \implies d (x n) (x n') < \varepsilon/2$ 
      by (meson half_gt_zero MCauchy_def R  $\langle \varepsilon > 0 \rangle$ )
    obtain Ny where Ny:  $\bigwedge n n'. \llbracket n \geq Ny; n' \geq Ny \rrbracket \implies d (y n) (y n') < \varepsilon/2$ 
      by (meson half_gt_zero MCauchy_def R  $\langle \varepsilon > 0 \rangle$ )
    obtain Nxy where Nxy:  $\bigwedge n. n \geq Nxy \implies d (x n) (y n) < \varepsilon/2$ 
      using R  $\langle \varepsilon > 0 \rangle$  half_gt_zero unfolding LIMSEQ_iff
      by (metis abs_mdif diff_zero real_norm_def)
    define N where  $N \equiv 2 * \text{Max}\{Nx, Ny, Nxy\}$ 
    show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\text{if even } n \text{ then } x (n \text{ div } 2) \text{ else } y$ 
       $(n \text{ div } 2)) (\text{if even } n' \text{ then } x (n' \text{ div } 2) \text{ else } y (n' \text{ div } 2)) < \varepsilon$ 
    proof (intro exI strip)
      fix n n'
      assume  $N \leq n$  and  $N \leq n'$ 
      then have  $n \text{ div } 2 \geq Nx \text{ n div } 2 \geq Ny \text{ n div } 2 \geq Nxy \text{ n' div } 2 \geq Nx \text{ n' div}$ 
 $2 \geq Ny$ 
      by (auto simp: N_def)
      then have  $d (x (n \text{ div } 2)) (y (n \text{ div } 2)) < \varepsilon/2$ 
        and  $d (x (n \text{ div } 2)) (x (n' \text{ div } 2)) < \varepsilon/2$ 
        and  $d (y (n \text{ div } 2)) (y (n' \text{ div } 2)) < \varepsilon/2$ 
      using Nx Ny Nxy by blast+
      have inM:  $x (n \text{ div } 2) \in M \ x (n' \text{ div } 2) \in M \ y (n \text{ div } 2) \in M \ y (n' \text{ div } 2) \in$ 

```

```

M
  using MCauchy_def R by blast+
  show d (if even n then x (n div 2) else y (n div 2)) (if even n' then x (n' div
2) else y (n' div 2)) < ε
  proof (cases even n)
    case nt: True
    show ?thesis
    proof (cases even n')
      case True
      with ⟨ε > 0⟩ nt dxnn' show ?thesis by auto
    next
      case False
      with nt dxyn dynn' inM triangle show ?thesis
      by fastforce
    qed
  next
  case nf: False
  show ?thesis
  proof (cases even n')
    case True
    then show ?thesis
    by (smt (verit) ⟨ε > 0⟩ dxyn dxnn' triangle commute inM field_sum_of_halves)
  next
  case False
  with ⟨ε > 0⟩ nf dynn' show ?thesis by auto
  qed
qed
qed
qed
qed

```

lemma *MCauchy_interleaving*:

```

MCauchy (λn. if even n then σ(n div 2) else a) ↔
  range σ ⊆ M ∧ limitin mtopology σ a sequentially (is ?lhs=?rhs)
proof -
  have ?lhs ↔ (MCauchy σ ∧ a ∈ M ∧ (λn. d (σ n) a) → 0)
  by (simp add: MCauchy_interleaving_gen [where y = λn. a])
  also have ... = ?rhs
  by (metis MCauchy_def always_eventually_convergent_imp_MCauchy lim-
itin_metric_dist_null range_subsetD)
  finally show ?thesis .
qed

```

lemma *mcomplete_nest*:

```

mcomplete ↔
  (∀ C::nat ⇒ 'a set. (∀ n. closedin mtopology (C n)) ∧
  (∀ n. C n ≠ {}) ∧ decseq C ∧ (∀ ε>0. ∃ n a. C n ⊆ mball a ε)
  → ⋂ (range C) ≠ {}) (is ?lhs=?rhs)
proof

```

```

assume  $L$ : ?lhs
show ?rhs
  unfolding imp_conjL
proof (intro strip)
  fix  $C :: \text{nat} \Rightarrow 'a \text{ set}$ 
  assume  $\text{clo}: \forall n. \text{closedin } \text{m topology } (C\ n)$ 
  and  $\text{ne}: \forall n. C\ n \neq \{\}\ :: 'a \text{ set}$ 
  and  $\text{dec}: \text{decseq } C$ 
  and  $\text{cover } [\text{rule\_format}]: \forall \varepsilon > 0. \exists n\ a. C\ n \subseteq \text{mcball } a\ \varepsilon$ 
obtain  $\sigma$  where  $\sigma: \bigwedge n. \sigma\ n \in C\ n$ 
by (meson ne empty_iff_set_eq_iff)
have MCauchy  $\sigma$ 
  unfolding MCauchy_def
proof (intro conjI strip)
  show  $\text{range } \sigma \subseteq M$ 
  using  $\sigma\ \text{clo } \text{metric\_closedin\_iff\_sequentially\_closed}$  by auto
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  then obtain  $N\ a$  where  $N: C\ N \subseteq \text{mcball } a\ (\varepsilon/3)$ 
  using cover by fastforce
  have  $d\ (\sigma\ m)\ (\sigma\ n) < \varepsilon$  if  $N \leq m\ N \leq n$  for  $m\ n$ 
  proof –
    have  $d\ a\ (\sigma\ m) \leq \varepsilon/3\ d\ a\ (\sigma\ n) \leq \varepsilon/3$ 
    using  $\text{dec } N\ \sigma$  that by (fastforce simp: decseq_def)+
    then have  $d\ (\sigma\ m)\ (\sigma\ n) \leq \varepsilon/3 + \varepsilon/3$ 
    using triangle  $\sigma$  commute dec decseq_def subsetD that  $N$ 
    by (smt (verit, ccfv_threshold) in_mcball)
    also have  $\dots < \varepsilon$ 
    using  $\langle \varepsilon > 0 \rangle$  by auto
    finally show ?thesis .
  qed
  then show  $\exists N. \forall m\ n. N \leq m \longrightarrow N \leq n \longrightarrow d\ (\sigma\ m)\ (\sigma\ n) < \varepsilon$ 
  by blast
qed
then obtain  $x$  where  $x: \text{limitin } \text{m topology } \sigma\ x\ \text{sequentially}$ 
  using L mcomplete_def by blast
  have  $x \in C\ n$  for  $n$ 
  proof (rule limitin_closedin [OF x])
  show  $\text{closedin } \text{m topology } (C\ n)$ 
  by (simp add: clo)
  show  $\forall_F x\ \text{in } \text{sequentially}. \sigma\ x \in C\ n$ 
  by (metis  $\sigma\ \text{dec decseq_def eventually_sequentiallyI subsetD$ )
qed auto
  then show  $\bigcap (\text{range } C) \neq \{\}$ 
  by blast
qed
next
  assume  $R$ : ?rhs
  show ?lhs

```

```

  unfolding mcomplete_def
proof (intro strip)
  fix  $\sigma$ 
  assume MCauchy  $\sigma$ 
  then have range  $\sigma \subseteq M$ 
    using MCauchy_def by blast
  define C where C  $\equiv \lambda n. \text{mtopology\_closure\_of } (\sigma \text{ ` } \{n..\})$ 
  have  $\forall n. \text{closedin\_mtopology } (C \ n)$ 
    by (auto simp: C_def)
  moreover
  have ne:  $\bigwedge n. C \ n \neq \{\}$ 
    using  $\langle \text{MCauchy } \sigma \rangle$  by (auto simp: C_def MCauchy_def disjnt_iff closure_of_eq_empty_gen)
  moreover
  have dec: decseq C
    unfolding monotone_on_def
  proof (intro strip)
    fix  $m \ n :: \text{nat}$ 
    assume  $m \leq n$ 
    then have  $\{n..\} \subseteq \{m..\}$ 
      by auto
    then show  $C \ n \subseteq C \ m$ 
      unfolding C_def by (meson closure_of_mono image_mono)
  qed
  moreover
  have C:  $\exists N \ u. C \ N \subseteq \text{mcball } u \ \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    obtain N where  $\bigwedge m \ n. N \leq m \wedge N \leq n \implies d \ (\sigma \ m) \ (\sigma \ n) < \varepsilon$ 
      by (meson MCauchy_def  $\langle 0 < \varepsilon \rangle \langle \text{MCauchy } \sigma \rangle$ )
    then have  $\sigma \text{ ` } \{N..\} \subseteq \text{mcball } (\sigma \ N) \ \varepsilon$ 
      using MCauchy_def  $\langle \text{MCauchy } \sigma \rangle$  by (force simp: less_eq_real_def)
    then have  $C \ N \subseteq \text{mcball } (\sigma \ N) \ \varepsilon$ 
      by (simp add: C_def closure_of_minimal)
    then show ?thesis
      by blast
  qed
  ultimately obtain l where  $x: l \in \bigcap (\text{range } C)$ 
    by (metis R_ex_in_conv)
  then have *:  $\bigwedge \varepsilon \ N. 0 < \varepsilon \implies \exists n'. N \leq n' \wedge l \in M \wedge \sigma \ n' \in M \wedge d \ l \ (\sigma \ n') < \varepsilon$ 
    by (force simp: C_def metric_closure_of)
  then have  $l \in M$ 
    using gt_ex by blast
  show  $\exists l. \text{limitin\_mtopology } \sigma \ l \text{ sequentially}$ 
  unfolding limitin_metric
  proof (intro conjI strip exI)
    show  $l \in M$ 
      using  $\langle \forall n. \text{closedin\_mtopology } (C \ n) \rangle \text{closedin\_subset } x$  by fastforce
    fix  $\varepsilon :: \text{real}$ 

```

```

assume  $\varepsilon > 0$ 
obtain  $N$  where  $N: \bigwedge m n. N \leq m \wedge N \leq n \implies d(\sigma m)(\sigma n) < \varepsilon/2$ 
  by (meson MCauchy_def  $\langle 0 < \varepsilon \rangle$  MCauchy  $\sigma$  half_gt_zero)
with * [of  $\varepsilon/2$   $N$ ]
have  $\forall n \geq N. \sigma n \in M \wedge d(\sigma n) l < \varepsilon$ 
  by (smt (verit)  $\langle \text{range } \sigma \subseteq M \rangle$  commute field_sum_of_halves range_subsetD
triangle)
then show  $\forall_F n$  in sequentially.  $\sigma n \in M \wedge d(\sigma n) l < \varepsilon$ 
  using eventually_sequentially by blast
qed
qed
qed

```

lemma *mcomplete_nest_sing*:

```

mcomplete  $\longleftrightarrow$ 
 $(\forall C. (\forall n. \text{closedin\_mtopology } (C\ n)) \wedge$ 
 $(\forall n. C\ n \neq \{\}) \wedge \text{decseq } C \wedge (\forall \varepsilon > 0. \exists n\ a. C\ n \subseteq \text{mcball } a\ \varepsilon)$ 
 $\longrightarrow (\exists l. l \in M \wedge \bigcap (\text{range } C) = \{l\}))$ 

```

proof –

```

have *: False
if clo:  $\forall n. \text{closedin\_mtopology } (C\ n)$ 
  and cover:  $\forall \varepsilon > 0. \exists n\ a. C\ n \subseteq \text{mcball } a\ \varepsilon$ 
  and no_sing:  $\bigwedge y. y \in M \implies \bigcap (\text{range } C) \neq \{y\}$ 
  and l:  $\forall n. l \in C\ n$ 
for  $C :: \text{nat} \Rightarrow 'a\ \text{set}$  and  $l$ 

```

proof –

```

have inM:  $\bigwedge x. x \in \bigcap (\text{range } C) \implies x \in M$ 
  using closedin_metric clo by fastforce
then have  $l \in M$ 
  by (simp add: l)
have False if  $l': l' \in \bigcap (\text{range } C)$  and  $l' \neq l$  for  $l'$ 

```

proof –

```

have  $l' \in M$ 
  using inM  $l'$  by blast
obtain  $n\ a$  where  $na: C\ n \subseteq \text{mcball } a\ (d\ l\ l' / 3)$ 
  using inM  $\langle l \in M \rangle$   $l' \langle l' \neq l \rangle$  cover by force
then have  $d\ a\ l \leq (d\ l\ l' / 3)$   $d\ a\ l' \leq (d\ l\ l' / 3)$   $a \in M$ 
  using  $l\ l'\ na$  in_mcball by auto
then have  $d\ l\ l' \leq (d\ l\ l' / 3) + (d\ l\ l' / 3)$ 
  using  $\langle l \in M \rangle$   $\langle l' \in M \rangle$  mdist_reverse_triangle by fastforce
then show False
  using nonneg [of  $l\ l'$ ]  $\langle l' \neq l \rangle$   $\langle l \in M \rangle$   $\langle l' \in M \rangle$  zero by force

```

qed

then show *False*

```

  by (metis  $l \langle l \in M \rangle$  no_sing INT_I empty_iff_insertI1 is_singletonE
is_singletonI')

```

qed

show *?thesis*

```

unfolding mcomplete_nest_imp_conjL
apply (intro all_cong1_imp_cong_refl)
using *
by (smt (verit) Inter_iff_ex_in_conv_range_constant_range_eqI)
qed

lemma mcomplete_fip:
  mcomplete  $\longleftrightarrow$ 
  ( $\forall \mathcal{C}. (\forall C \in \mathcal{C}. \text{closedin\_mtopology } C) \wedge$ 
   ( $\forall e > 0. \exists C a. C \in \mathcal{C} \wedge C \subseteq \text{mcball } a \ e) \wedge (\forall \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{C} \longrightarrow \bigcap$ 
    $\mathcal{F} \neq \{\})$ 
    $\longrightarrow \bigcap \mathcal{C} \neq \{\})$ )
  (is ?lhs = ?rhs)
proof
assume L: ?lhs
show ?rhs
  unfolding mcomplete_nest_sing_imp_conjL
proof (intro strip)
fix C :: 'a set set
assume clo:  $\forall C \in \mathcal{C}. \text{closedin\_mtopology } C$ 
and cover:  $\forall e > 0. \exists C a. C \in \mathcal{C} \wedge C \subseteq \text{mcball } a \ e$ 
and fip:  $\forall \mathcal{F}. \text{finite } \mathcal{F} \longrightarrow \mathcal{F} \subseteq \mathcal{C} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
then have  $\forall n. \exists C. C \in \mathcal{C} \wedge (\exists a. C \subseteq \text{mcball } a \ (\text{inverse } (\text{Suc } n)))$ 
by simp
then obtain C where C:  $\bigwedge n. C \ n \in \mathcal{C}$ 
and coverC:  $\bigwedge n. \exists a. C \ n \subseteq \text{mcball } a \ (\text{inverse } (\text{Suc } n))$ 
by metis
define D where D  $\equiv \lambda n. \bigcap (C \ ' \ \{\dots n\})$ 
have cloD:  $\text{closedin\_mtopology } (D \ n)$  for n
unfolding D_def using clo C by blast
have neD:  $D \ n \neq \{\}$  for n
using fip C by (simp add: D_def image_subset_iff)
have decD:  $\text{decseq } D$ 
by (force simp: D_def decseq_def)
have coverD:  $\exists n \ a. D \ n \subseteq \text{mcball } a \ \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
proof -
obtain n where  $\text{inverse } (\text{Suc } n) < \varepsilon$ 
using  $\langle 0 < \varepsilon \rangle$  reals_Archimedean by blast
then obtain a where  $C \ n \subseteq \text{mcball } a \ \varepsilon$ 
by (meson coverC less_eq_real_def mcball_subset_concentric order_trans)
then show ?thesis
unfolding D_def by blast
qed
have *:  $a \in \bigcap \mathcal{C}$  if  $a: \bigcap (\text{range } D) = \{a\}$  and  $a \in M$  for a
proof -
have aC:  $a \in C \ n$  for n
using that by (auto simp: D_def)
have eqa:  $\bigwedge u. (\forall n. u \in C \ n) \implies a = u$ 
using that by (auto simp: D_def)

```

```

have a ∈ T if T ∈ C for T
proof -
  have cloT: closedin mtopology (T ∩ D n) for n
    using clo cloD that by blast
  have ⋂ (insert T (C ‘ {..n})) ≠ {} for n
    using that C by (intro fip [rule_format]) auto
  then have neT: T ∩ D n ≠ {} for n
    by (simp add: D_def)
  have decT: decseq (λn. T ∩ D n)
    by (force simp: D_def decseq_def)
  have coverT: ∃ n a. T ∩ D n ⊆ mball a ε if ε > 0 for ε
    by (meson coverD le_infI2 that)
  show ?thesis
    using L [unfolded mcomplete_nest_sing, rule_format, of λn. T ∩ D n] a
    by (force simp: cloT neT decT coverT)
qed
then show ?thesis by auto
qed
show ⋂ C ≠ {}
  by (metis L cloD neD decD coverD * empty_iff mcomplete_nest_sing)
qed
next
assume R [rule_format]: ?rhs
show ?lhs
  unfolding mcomplete_nest_imp_conjL
proof (intro strip)
  fix C :: nat ⇒ 'a set
  assume clo: ∀ n. closedin mtopology (C n)
  and ne: ∀ n. C n ≠ {}
  and dec: decseq C
  and cover: ∀ ε > 0. ∃ n a. C n ⊆ mball a ε
  have ⋂ (C ‘ N) ≠ {} if finite N for N
  proof -
    obtain k where N ⊆ {..k}
      using ⟨finite N⟩ finite_nat_iff_bounded_le by auto
    with dec have C k ⊆ ⋂ (C ‘ N) by (auto simp: decseq_def)
    then show ?thesis
      using ne by force
  qed
  with clo cover R [of range C] show ⋂ (range C) ≠ {}
  by (metis (no_types, opaque_lifting) finite_subset_image image_iff UNIV_I)
qed
qed

```

```

lemma mcomplete_fip_sing:
  mcomplete ⟷
  (∀ C. (∀ C ∈ C. closedin mtopology C) ∧
  (∀ ε > 0. ∃ c a. c ∈ C ∧ c ⊆ mball a ε) ∧

```



```

  (∀ $\mathcal{F}$ . finite  $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{C} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ )  $\longrightarrow$ 
  (∃ $l$ .  $l \in M \wedge \bigcap \mathcal{C} = \{l\}$ )
  (is ?lhs = ?rhs)
proof
  have *:  $l \in M \cap \mathcal{C} = \{l\}$ 
  if clo: Ball  $\mathcal{C}$  (closedin mtopology)
    and cover: ∀ $e>0$ . ∃ $C$  a.  $C \in \mathcal{C} \wedge C \subseteq \text{mcball } a \ e$ 
    and fin: ∀ $\mathcal{F}$ . finite  $\mathcal{F} \longrightarrow \mathcal{F} \subseteq \mathcal{C} \longrightarrow \bigcap \mathcal{F} \neq \{\}$ 
    and l:  $l \in \bigcap \mathcal{C}$ 
  for  $\mathcal{C} :: 'a \text{ set set}$  and  $l$ 
proof -
  show  $l \in M$ 
  by (meson Inf_lower2 clo cover gt_ex metric_closedin_iff_sequentially_closed
subsetD that(4))
  show  $\bigcap \mathcal{C} = \{l\}$ 
  proof (cases  $\mathcal{C} = \{\}$ )
    case True
    then show ?thesis
      using cover mbounded_pos by auto
  next
  case False
  have CM:  $\bigwedge a$ .  $a \in \bigcap \mathcal{C} \implies a \in M$ 
    using False clo closedin_subset by fastforce
  have  $l' \notin \bigcap \mathcal{C}$  if  $l' \neq l$  for  $l'$ 
  proof
    assume  $l'$ :  $l' \in \bigcap \mathcal{C}$ 
    with CM have  $l' \in M$  by blast
    with that  $\langle l \in M \rangle$  have  $gt0$ :  $0 < d \ l \ l'$ 
      by simp
    then obtain  $C$  a where  $C \in \mathcal{C}$  and  $C$ :  $C \subseteq \text{mcball } a \ (d \ l \ l' / 3)$ 
      using cover [rule_format, of  $d \ l \ l' / 3$ ] by auto
    then have  $d \ a \ l \leq (d \ l \ l' / 3)$   $d \ a \ l' \leq (d \ l \ l' / 3)$   $a \in M$ 
      using  $l \ l'$  in_mcball by auto
    then have  $d \ l \ l' \leq (d \ l \ l' / 3) + (d \ l \ l' / 3)$ 
      using  $\langle l \in M \rangle \langle l' \in M \rangle$  mdist_reverse_triangle by fastforce
    with  $gt0$  show False by auto
  qed
  then show ?thesis
    using  $l$  by fastforce
  qed
qed
assume  $L$ : ?lhs
with * show ?rhs
  unfolding mcomplete_fip imp_conjL ex_in_conv [symmetric]
  by (elim all_forward imp_forward2 asm_rl) (blast intro: elim:)
next
assume ?rhs then show ?lhs
  unfolding mcomplete_fip by (force elim!: all_forward)
qed

```

end

definition *mcomplete_of* :: 'a metric \Rightarrow bool
 where *mcomplete_of* $\equiv \lambda m. \text{Metric_space.mcomplete } (m\text{space } m) (m\text{dist } m)$

lemma (in *Metric_space*) *mcomplete_of* [*simp*]: *mcomplete_of* (*metric* (*M*,*d*)) =
mcomplete
 by (*simp add: mcomplete_of_def*)

lemma *mcomplete_trivial*: *Metric_space.mcomplete* {} ($\lambda x y. 0$)
 using *Metric_space.intro Metric_space.mcomplete_empty_mspace* by force

lemma *mcomplete_trivial_singleton*: *Metric_space.mcomplete* { $\lambda x. a$ } ($\lambda x y. 0$)
proof –
 interpret *Metric_space* { $\lambda x. a$ } $\lambda x y. 0$
 by *unfold_locales auto*
 show ?thesis
 unfolding *mcomplete_def MCauchy_def image_subset_iff* by (*metis UNIV_I*
limit_metric_sequentially)
qed

lemma *MCauchy_iff_Cauchy* [*iff*]: *Met_TC.MCauchy* = *Cauchy*
 by (*force simp: Cauchy_def Met_TC.MCauchy_def*)

lemma *mcomplete_iff_complete* [*iff*]:
Met_TC.mcomplete (*Pure.type* :: 'a::metric_space *itself*) \longleftrightarrow *complete* (*UNIV*::'a
set)
 by (*auto simp: Met_TC.mcomplete_def complete_def*)

context *Submetric*

begin

lemma *MCauchy_submetric*:
sub.MCauchy $\sigma \longleftrightarrow$ *range* $\sigma \subseteq A \wedge$ *MCauchy* σ
 using *MCauchy_def sub.MCauchy_def subset* by force

lemma *closedin_mcomplete_imp_mcomplete*:
assumes *clo*: *closedin* *mtopology* *A* **and** *mcomplete*
shows *sub.mcomplete*
 unfolding *sub.mcomplete_def*

proof (*intro strip*)
 fix σ
 assume *sub.MCauchy* σ
then have σ : *MCauchy* σ *range* $\sigma \subseteq A$
 using *MCauchy_submetric* by *blast+*
then obtain *x* **where** *x*: *limitin* *mtopology* σ *x* *sequentially*
 using \langle *mcomplete* \rangle **unfolding** *mcomplete_def* by *blast*
then have $x \in A$

```

using  $\sigma$  clo metric_closedin_iff_sequentially_closed by force
with  $\sigma$   $x$  show  $\exists x. \text{limitin sub.mtopology } \sigma \ x \text{ sequentially}$ 
using limitin_submetric_iff_range_subsetD by fastforce
qed

```

```

lemma sequentially_closedin_mcomplete_imp_mcomplete:
  assumes mcomplete and  $\bigwedge \sigma l. \text{range } \sigma \subseteq A \wedge \text{limitin mtopology } \sigma \ l \text{ sequentially}$ 
 $\implies l \in A$ 
  shows sub.mcomplete
  using assms closedin_mcomplete_imp_mcomplete metric_closedin_iff_sequentially_closed
  subset by blast

```

end

```

context Metric_space
begin

```

```

lemma mcomplete_Un:
  assumes  $A$ : Submetric  $M \ d \ A$  Metric_space.mcomplete  $A \ d$ 
  and  $B$ : Submetric  $M \ d \ B$  Metric_space.mcomplete  $B \ d$ 
  shows Submetric  $M \ d \ (A \cup B)$  Metric_space.mcomplete  $(A \cup B) \ d$ 
proof –
  show Submetric  $M \ d \ (A \cup B)$ 
  by (meson assms le_sup_iff Submetric_axioms_def Submetric_def)
  then interpret MAB: Metric_space  $A \cup B \ d$ 
  by (meson Submetric.subset subspace)
  interpret MA: Metric_space  $A \ d$ 
  by (meson A Submetric.subset subspace)
  interpret MB: Metric_space  $B \ d$ 
  by (meson B Submetric.subset subspace)
  show Metric_space.mcomplete  $(A \cup B) \ d$ 
  unfolding MAB.mcomplete_def
proof (intro strip)
  fix  $\sigma$ 
  assume MAB.MCauchy  $\sigma$ 
  then have range  $\sigma \subseteq A \cup B$ 
  using MAB.MCauchy_def by blast
  then have UNIV  $\subseteq \sigma - 'A \cup \sigma - 'B$ 
  by blast
  then consider infinite  $(\sigma - 'A) \mid$  infinite  $(\sigma - 'B)$ 
  using finite_subset by auto
  then show  $\exists x. \text{limitin MAB.mtopology } \sigma \ x \text{ sequentially}$ 
proof cases
  case 1
  then obtain  $r$  where strict_mono  $r$  and  $r: \bigwedge n::\text{nat}. r \ n \in \sigma - 'A$ 
  using infinite_enumerate by blast
  then have MA.MCauchy  $(\sigma \circ r)$ 
  using MA.MCauchy_def MAB.MCauchy_def MAB.MCauchy_subsequence

```

```

⟨MAB.MCauchy σ⟩ by auto
  with A obtain x where limitin MA.mtopology (σ ∘ r) x sequentially
  using MA.mcomplete_def by blast
  then have limitin MAB.mtopology (σ ∘ r) x sequentially
  by (metis MA.limit_metric_sequentially MAB.limit_metric_sequentially
UnCI)
  then show ?thesis
  using MAB.MCauchy_convergent_subsequence ⟨MAB.MCauchy σ⟩ ⟨strict_mono
r⟩ by blast
next
case 2
then obtain r where strict_mono r and r: ∧n::nat. r n ∈ σ -' B
  using infinite_enumerate by blast
  then have MB.MCauchy (σ ∘ r)
  using MB.MCauchy_def MAB.MCauchy_def MAB.MCauchy_subsequence
⟨MAB.MCauchy σ⟩ by auto
  with B obtain x where limitin MB.mtopology (σ ∘ r) x sequentially
  using MB.mcomplete_def by blast
  then have limitin MAB.mtopology (σ ∘ r) x sequentially
  by (metis MB.limit_metric_sequentially MAB.limit_metric_sequentially
UnCI)
  then show ?thesis
  using MAB.MCauchy_convergent_subsequence ⟨MAB.MCauchy σ⟩ ⟨strict_mono
r⟩ by blast
qed
qed
qed

```

```

lemma mcomplete_Union:
  assumes finite S
  and ∧A. A ∈ S ⇒ Submetric M d A ∧A. A ∈ S ⇒ Metric_space.mcomplete
A d
  shows Submetric M d (∪S) Metric_space.mcomplete (∪S) d
  using assms
  by (induction rule: finite_induct) (auto simp: mcomplete_Un)

```

```

lemma mcomplete_Inter:
  assumes finite S S ≠ {}
  and sub: ∧A. A ∈ S ⇒ Submetric M d A
  and comp: ∧A. A ∈ S ⇒ Metric_space.mcomplete A d
  shows Submetric M d (∩S) Metric_space.mcomplete (∩S) d
proof -
  show Submetric M d (∩S)
  using assms unfolding Submetric_def Submetric_axioms_def
  by (metis Inter_lower equalsOI inf.orderE le_inf_iff)
  then interpret MS: Submetric M d ∩S
  by (meson Submetric.subset subspace)
  show Metric_space.mcomplete (∩S) d
  unfolding MS.sub.mcomplete_def

```

```

proof (intro strip)
  fix  $\sigma$ 
  assume  $MS.sub.MCauchy\ \sigma$ 
  then have  $range\ \sigma \subseteq \bigcap \mathcal{S}$ 
    using  $MS.MCauchy\_submetric$  by blast
  obtain  $A$  where  $A \in \mathcal{S}$  and  $A: Metric\_space.mcomplete\ A\ d$ 
    using  $assms$  by blast
  then have  $range\ \sigma \subseteq A$ 
    using  $\langle range\ \sigma \subseteq \bigcap \mathcal{S} \rangle$  by blast
  interpret  $SA: Submetric\ M\ d\ A$ 
    by (meson  $\langle A \in \mathcal{S} \rangle$  sub Submetric.subset subspace)
  have  $MCauchy\ \sigma$ 
    using  $MS.MCauchy\_submetric\ \langle MS.sub.MCauchy\ \sigma \rangle$  by blast
  then obtain  $x$  where  $x: limitin\ SA.sub.mtopology\ \sigma\ x$  sequentially
  by (metis  $A\ SA.sub.MCauchy\_def\ SA.sub.mcomplete\_alt\ MCauchy\_def\ \langle range\ \sigma \subseteq A \rangle$ )
  show  $\exists x. limitin\ MS.sub.mtopology\ \sigma\ x$  sequentially
    apply (rule_tac  $x=x$  in  $exI$ )
    unfolding  $MS.limitin\_submetric\_iff$ 
  proof (intro conjI)
    show  $x \in \bigcap \mathcal{S}$ 
    proof clarsimp
      fix  $U$ 
      assume  $U \in \mathcal{S}$ 
      interpret  $SU: Submetric\ M\ d\ U$ 
        by (meson  $\langle U \in \mathcal{S} \rangle$  sub Submetric.subset subspace)
      have  $range\ \sigma \subseteq U$ 
        using  $\langle U \in \mathcal{S} \rangle\ \langle range\ \sigma \subseteq \bigcap \mathcal{S} \rangle$  by blast
      moreover have  $Metric\_space.mcomplete\ U\ d$ 
        by (simp add:  $\langle U \in \mathcal{S} \rangle$  comp)
      ultimately obtain  $x'$  where  $x': limitin\ SU.sub.mtopology\ \sigma\ x'$  sequentially
      using  $MCauchy\_def\ SU.sub.MCauchy\_def\ SU.sub.mcomplete\_alt\ \langle MCauchy\ \sigma \rangle$  by meson
      have  $x' = x$ 
      proof (intro limitin_metric_unique)
        show  $limitin\ mtopology\ \sigma\ x'$  sequentially
          by (meson  $SU.Submetric\_axioms\ Submetric.limitin\_submetric\_iff\ x'$ )
        show  $limitin\ mtopology\ \sigma\ x$  sequentially
          by (meson  $SA.Submetric\_axioms\ Submetric.limitin\_submetric\_iff\ x$ )
      qed auto
      then show  $x \in U$ 
        using  $SU.sub.limitin\_mspace\ x'$  by blast
    qed
  show  $\forall_F n$   $n$  in sequentially.  $\sigma\ n \in \bigcap \mathcal{S}$ 
    by (meson  $\langle range\ \sigma \subseteq \bigcap \mathcal{S} \rangle$  always_eventually_range_subsetD)
  show  $limitin\ mtopology\ \sigma\ x$  sequentially
    by (meson  $SA.Submetric\_axioms\ Submetric.limitin\_submetric\_iff\ x$ )
  qed
qed

```

1370

qed

lemma *mcomplete_Int*:

assumes *A*: *Submetric M d A Metric_space.mcomplete A d*
and *B*: *Submetric M d B Metric_space.mcomplete B d*
shows *Submetric M d (A ∩ B) Metric_space.mcomplete (A ∩ B) d*
using *mcomplete_Inter [of {A,B}] assms by force+*

6.7.11 Totally bounded subsets of metric spaces

definition *mtotally_bounded*

where *mtotally_bounded S* $\equiv \forall \varepsilon > 0. \exists K. \text{finite } K \wedge K \subseteq S \wedge S \subseteq (\bigcup_{x \in K} \text{mball } x \ \varepsilon)$

lemma *mtotally_bounded_empty [iff]*: *mtotally_bounded {}*

by (*simp add: mtotally_bounded_def*)

lemma *finite_imp_mtotally_bounded*:

$\llbracket \text{finite } S; S \subseteq M \rrbracket \implies \text{mtotally_bounded } S$
by (*auto simp: mtotally_bounded_def*)

lemma *mtotally_bounded_imp_subset*: *mtotally_bounded S* $\implies S \subseteq M$

by (*force simp: mtotally_bounded_def intro!: zero_less_one*)

lemma *mtotally_bounded_sing [simp]*:

mtotally_bounded {x} $\longleftrightarrow x \in M$

by (*meson empty_subsetI finite.simps finite_imp_mtotally_bounded insert_subset mtotally_bounded_imp_subset*)

lemma *mtotally_bounded_Un*:

assumes *mtotally_bounded S mtotally_bounded T*

shows *mtotally_bounded (S ∪ T)*

proof –

have $\exists K. \text{finite } K \wedge K \subseteq S \cup T \wedge S \cup T \subseteq (\bigcup_{x \in K} \text{mball } x \ e)$

if $e > 0$ **and** $K: \text{finite } K \wedge K \subseteq S \wedge S \subseteq (\bigcup_{x \in K} \text{mball } x \ e)$

and $L: \text{finite } L \wedge L \subseteq T \wedge T \subseteq (\bigcup_{x \in L} \text{mball } x \ e)$ **for** $K \ L \ e$

using that by (*rule_tac x=K ∪ L in exI*) *auto*

with assms show *?thesis*

unfolding *mtotally_bounded_def* **by** *presburger*

qed

lemma *mtotally_bounded_Union*:

assumes *finite f* $\wedge S. S \in f \implies \text{mtotally_bounded } S$

shows *mtotally_bounded (∪ f)*

using assms by (*induction f*) (*auto simp: mtotally_bounded_Un*)

lemma *mtotally_bounded_imp_mbounded*:

assumes *mtotally_bounded S*

```

shows mbounded S
proof -
  obtain K where finite K  $\wedge$   $K \subseteq S \wedge S \subseteq (\bigcup_{x \in K}. \text{mball } x \ 1)$ 
  using assms by (force simp: mtotally_bounded_def)
  then show ?thesis
  by (smt (verit) finite_imageI image_iff mbounded_Union mbounded_mball
    mbounded_subset)
qed

```

lemma *mtotally_bounded_sequentially*:

```

mtotally_bounded S  $\longleftrightarrow$ 
  S  $\subseteq$  M  $\wedge$  ( $\forall \sigma :: \text{nat} \Rightarrow 'a. \text{range } \sigma \subseteq S \longrightarrow (\exists r. \text{strict\_mono } r \wedge \text{MCauchy } (\sigma \circ r))$ )
(is _  $\longleftrightarrow$  _  $\wedge$  ?rhs)
proof (cases S  $\subseteq$  M)
  case True
  show ?thesis
  proof -
    { fix  $\sigma :: \text{nat} \Rightarrow 'a$ 
      assume L: mtotally_bounded S and  $\sigma: \text{range } \sigma \subseteq S$ 
      have  $\exists j > i. d (\sigma \ i) (\sigma \ j) < 3 * \varepsilon / 2 \wedge \text{infinite } (\sigma - ' \text{mball } (\sigma \ j) (\varepsilon / 2))$ 
      if inf: infinite ( $\sigma - ' \text{mball } (\sigma \ i) \ \varepsilon$ ) and  $\varepsilon > 0$  for  $i \ \varepsilon$ 
      proof -
        obtain K where finite K  $K \subseteq S$  and  $K: S \subseteq (\bigcup_{x \in K}. \text{mball } x \ (\varepsilon / 4))$ 
        by (metis L mtotally_bounded_def  $\langle \varepsilon > 0 \rangle$  zero_less_divide_iff zero_less_numeral)
        then have  $K\_imp\_ex: \bigwedge y. y \in S \implies \exists x \in K. d \ x \ y < \varepsilon / 4$ 
        by fastforce
        have False if  $\forall x \in K. d \ x \ (\sigma \ i) < \varepsilon + \varepsilon / 4 \longrightarrow \text{finite } (\sigma - ' \text{mball } x \ (\varepsilon / 4))$ 
        proof -
          have  $\exists w. w \in K \wedge d \ w \ (\sigma \ i) < 5 * \varepsilon / 4 \wedge d \ w \ (\sigma \ j) < \varepsilon / 4$ 
          if  $d (\sigma \ i) (\sigma \ j) < \varepsilon$  for  $j$ 
          proof -
            obtain w where  $w: d \ w \ (\sigma \ j) < \varepsilon / 4 \ w \in K$ 
            using  $K\_imp\_ex \ \sigma$  by blast
            then have  $d \ w \ (\sigma \ i) < \varepsilon + \varepsilon / 4$ 
            by (smt (verit, ccfv_SIG) True  $\langle K \subseteq S \rangle \ \sigma \ \text{rangeI} \ \text{subset\_eq} \ \text{that} \ \text{triangle}'$ )
          with w show ?thesis
          using in_mball by auto
        qed
        then have  $(\sigma - ' \text{mball } (\sigma \ i) \ \varepsilon) \subseteq (\bigcup_{x \in K}. \text{if } d \ x \ (\sigma \ i) < \varepsilon + \varepsilon / 4 \ \text{then } \sigma - ' \text{mball } x \ (\varepsilon / 4) \ \text{else } \{\})$ 
        using True  $\langle K \subseteq S \rangle$  by force
        then show False
        using finite_subset_inf  $\langle \text{finite } K \rangle$  that by fastforce
      qed
    }
  then obtain x where  $x \in K$  and  $dxi: d \ x \ (\sigma \ i) < \varepsilon + \varepsilon / 4$  and  $infx: \text{infinite } (\sigma - ' \text{mball } x \ (\varepsilon / 4))$ 

```

```

    by blast
  then obtain j where j ∈ (σ -' mball x (ε/4)) - {..i}
    using bounded_nat_set_is_finite by (meson Diff_infinite_finite fi-
nite_atMost)
  then have j > i and dxj: d x (σ j) < ε/4
    by auto
  have (σ -' mball x (ε/4)) ⊆ (σ -' mball y (ε/2)) if d x y < ε/4 y ∈ M
for y
  using that by (simp add: mball_subset vimage_mono)
  then have infj: infinite (σ -' mball (σ j) (ε/2))
    by (meson True ⟨d x (σ j) < ε/4⟩ σ in_mono infx rangeI finite_subset)
  have σ i ∈ M σ j ∈ M x ∈ M
    using True ⟨K ⊆ S⟩ ⟨x ∈ K⟩ σ by force+
  then have d (σ i) (σ j) ≤ d x (σ i) + d x (σ j)
    using triangle'' by blast
  also have ... < 3*ε/2
    using dxi dxj by auto
  finally have d (σ i) (σ j) < 3*ε/2 .
  with ⟨i < j⟩ infj show ?thesis by blast
qed
then obtain next where next: ∧i ε. [ε > 0; infinite (σ -' mball (σ i) ε)] ⇒
  next i ε > i ∧ d (σ i) (σ (next i ε)) < 3*ε/2 ∧ infinite (σ -' mball
(σ (next i ε)) (ε/2))
  by metis
  have mbounded S
    using L by (simp add: mtotally_bounded_imp_mbounded)
  then obtain B where B: ∀y ∈ S. d (σ 0) y ≤ B and B > 0
    by (meson σ mbounded_alt_pos range_subsetD)
  define eps where eps ≡ λn. (B+1) / 2^n
  have [simp]: eps (Suc n) = eps n / 2 eps n > 0 for n
    using ⟨B > 0⟩ by (auto simp: eps_def)
  have UNIV ⊆ σ -' mball (σ 0) (B+1)
    using B True σ unfolding image_iff subset_iff
    by (smt (verit, best) UNIV_I in_mball vimageI)
  then have inf0: infinite (σ -' mball (σ 0) (eps 0))
    using finite_subset by (auto simp: eps_def)
  define r where r ≡ rec_nat 0 (λn rec. next rec (eps n))
  have [simp]: r 0 = 0 r (Suc n) = next (r n) (eps n) for n
    by (auto simp: r_def)
  have σrM[simp]: σ (r n) ∈ M for n
    using True σ by blast
  have inf: infinite (σ -' mball (σ (r n)) (eps n)) for n
  proof (induction n)
    case 0 then show ?case
      by (simp add: inf0)
  next
    case (Suc n) then show ?case
      using next [of eps n r n] by simp

```



```

qed
then have  $r (Suc\ n) > r\ n$  for  $n$ 
  by (simp add: nat)
then have strict_mono  $r$ 
  by (simp add: strict_mono_Suc_iff)
have  $d\_less: d (\sigma (r\ n)) (\sigma (r (Suc\ n))) < 3 * eps\ n / 2$  for  $n$ 
  using nat [OF _ inf] by simp
have  $eps\_plus: eps (k + n) = eps\ n * (1/2)^k$  for  $k\ n$ 
  by (simp add: eps_def power_add field_simps)
have *:  $d (\sigma (r\ n)) (\sigma (r (k + n))) < 3 * eps\ n$  for  $n\ k$ 
proof -
  have  $d (\sigma (r\ n)) (\sigma (r (k+n))) \leq 3/2 * eps\ n * (\sum_{i<k}. (1/2)^i)$ 
  proof (induction  $k$ )
    case 0 then show ?case
      by simp
    next
      case (Suc  $k$ )
      have  $d (\sigma (r\ n)) (\sigma (r (Suc\ k + n))) \leq d (\sigma (r\ n)) (\sigma (r (k + n))) + d$ 
 $(\sigma (r (k + n))) (\sigma (r (Suc (k + n))))$ 
      by (metis  $\sigma rM$  add.commute add_Suc_right triangle)
      with  $d\_less[of\ k+n]$  Suc show ?case
      by (simp add: algebra_simps eps_plus)
  qed
  also have  $\dots < 3/2 * eps\ n * 2$ 
  using geometric_sum [of  $1/2::real\ k$ ] by simp
  finally show ?thesis by simp
qed
have  $\exists N. \forall n \geq N. \forall n' \geq N. d (\sigma (r\ n)) (\sigma (r\ n')) < \epsilon$  if  $\epsilon > 0$  for  $\epsilon$ 
proof -
  define  $N$  where  $N \equiv nat \lceil (\log 2 (6*(B+1) / \epsilon)) \rceil$ 
  have  $\S: b \leq 2 \wedge nat \lceil \log 2\ b \rceil$  for  $b$ 
    by (smt (verit) less_log_of_power real_nat_ceiling_ge)
  have  $N: 6 * eps\ N \leq \epsilon$ 
  using  $\S$  [of  $(6*(B+1) / \epsilon)$ ] that by (auto simp: N_def eps_def field_simps)
  have  $d (\sigma (r\ N)) (\sigma (r\ n)) < 3 * eps\ N$  if  $n \geq N$  for  $n$ 
    by (metis * add.commute nat_le_iff_add that)
  then have  $\forall n \geq N. \forall n' \geq N. d (\sigma (r\ n)) (\sigma (r\ n')) < 3 * eps\ N + 3 * eps\ N$ 
    by (smt (verit, best)  $\sigma rM$  triangle')
  with  $N$  show ?thesis
    by fastforce
qed
then have MCauchy  $(\sigma \circ r)$ 
  unfolding MCauchy_def using True  $\sigma$  by auto
then have  $\exists r. strict\_mono\ r \wedge MCauchy (\sigma \circ r)$ 
  using  $\langle strict\_mono\ r \rangle$  by blast
}
moreover
{ assume  $R: ?rhs$ 
  have mtotally_bounded  $S$ 

```

```

unfolding mtotally_bounded_def
proof (intro strip)
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  have False if  $\S: \bigwedge K. \llbracket \text{finite } K; K \subseteq S \rrbracket \implies \exists s \in S. s \notin (\bigcup x \in K. \text{mball } x \ \varepsilon)$ 
  proof –
    obtain f where  $f: \bigwedge K. \llbracket \text{finite } K; K \subseteq S \rrbracket \implies f \ K \in S \wedge f \ K \notin (\bigcup x \in K. \text{mball } x \ \varepsilon)$ 
    using  $\S$  by metis
    define  $\sigma$  where  $\sigma \equiv \text{wfrec } \text{less\_than} \ (\lambda \text{seq } n. f \ (\text{seq} \ ' \ \{..<n\}))$ 
    have  $\sigma\_eq: \sigma \ n = f \ (\sigma \ ' \ \{..<n\})$  for  $n$ 
      by (simp add: cut_apply def_wfrec [OF  $\sigma\_def$ ])
    have [simp]:  $\sigma \ n \in S$  for  $n$ 
      using wf_less_than
    proof (induction n rule: wf_induct_rule)
      case (less n) with f show ?case
        by (auto simp:  $\sigma\_eq$  [of  $n$ ])
    qed
    then have  $\text{range } \sigma \subseteq S$  by blast
    have  $\sigma: p < n \implies \varepsilon \leq d \ (\sigma \ p) \ (\sigma \ n)$  for  $n \ p$ 
      using  $f$  [of  $\sigma \ ' \ \{..<n\}$ ] True by (fastforce simp:  $\sigma\_eq$  [of  $n$ ] Ball_def)
    then obtain r where strict_mono r M Cauchy  $(\sigma \circ r)$ 
      by (meson R  $\langle \text{range } \sigma \subseteq S \rangle$ )
    with  $\langle 0 < \varepsilon \rangle$  obtain  $N$ 
      where  $N: \bigwedge n \ n'. \llbracket n \geq N; n' \geq N \rrbracket \implies d \ (\sigma \ (r \ n)) \ (\sigma \ (r \ n')) < \varepsilon$ 
      by (force simp: M Cauchy_def)
    show ?thesis
      using  $N$  [of  $N$  Suc  $(r \ N)$ ]  $\langle \text{strict\_mono } r \rangle$ 
    by (smt (verit) Suc_le_eq  $\sigma$  le_SucI order_refl strict_mono_imp_increasing)
    qed
  then show  $\exists K. \text{finite } K \wedge K \subseteq S \wedge S \subseteq (\bigcup x \in K. \text{mball } x \ \varepsilon)$ 
    by blast
  qed
}
ultimately show ?thesis
using True by blast
qed
qed (use mtotally_bounded_imp_subset in auto)

```

lemma *mtotally_bounded_subset*:

```

 $\llbracket \text{mtotally\_bounded } S; T \subseteq S \rrbracket \implies \text{mtotally\_bounded } T$ 
by (meson mtotally_bounded_sequentially_order_trans)

```

lemma *mtotally_bounded_submetric*:

```

assumes mtotally_bounded S S  $\subseteq T$   $T \subseteq M$ 
shows Metric_space.mtotally_bounded T d S

```

proof –

```

interpret Submetric M d T

```

```

    using ‹ $T \subseteq M$ › by unfold_locales
  show ?thesis
    using assms
    unfolding sub.mtotally_bounded_def mtotally_bounded_def
    by (force simp: subset_iff elim!: all_forward ex_forward)
qed

lemma mtotally_bounded_absolute:
  mtotally_bounded  $S \longleftrightarrow S \subseteq M \wedge \text{Metric\_space.mtotally\_bounded } S \ d \ S$ 
proof -
  have mtotally_bounded  $S$  if  $S \subseteq M \ \text{Metric\_space.mtotally\_bounded } S \ d \ S$ 
  proof -
    interpret Submetric  $M \ d \ S$ 
    using ‹ $S \subseteq M$ › by unfold_locales
    show ?thesis
      using that
      by (meson MCauchy_submetric mtotally_bounded_sequentially sub.mtotally_bounded_sequentially)
  qed
  moreover have mtotally_bounded  $S \implies \text{Metric\_space.mtotally\_bounded } S \ d \ S$ 
  by (simp add: mtotally_bounded_imp_subset mtotally_bounded_submetric)
  ultimately show ?thesis
    using mtotally_bounded_imp_subset by blast
qed

lemma mtotally_bounded_closure_of:
  assumes mtotally_bounded  $S$ 
  shows mtotally_bounded (mtopology_closure_of  $S$ )
proof -
  have  $S \subseteq M$ 
  by (simp add: assms mtotally_bounded_imp_subset)
  have mtotally_bounded(mtopology_closure_of  $S$ )
  unfolding mtotally_bounded_def
  proof (intro strip)
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    then obtain  $K$  where finite  $K \ K \subseteq S$  and  $K: S \subseteq (\bigcup_{x \in K} \text{mball } x \ (\varepsilon/2))$ 
    by (metis assms mtotally_bounded_def half_gt_zero)
    have mtopology_closure_of  $S \subseteq (\bigcup_{x \in K} \text{mball } x \ \varepsilon)$ 
    unfolding metric_closure_of
  proof clarsimp
    fix  $x$ 
    assume  $x \in M$  and  $x: \forall r > 0. \exists y \in S. y \in M \wedge d \ x \ y < r$ 
    then obtain  $y$  where  $y \in S$  and  $y: d \ x \ y < \varepsilon/2$ 
    using ‹ $0 < \varepsilon$ › half_gt_zero by blast
    then obtain  $x'$  where  $x' \in K \ y \in \text{mball } x' \ (\varepsilon/2)$ 
    using  $K$  by auto
    then have  $d \ x' \ x < \varepsilon/2 + \varepsilon/2$ 
    using triangle  $y \langle x \in M \rangle$  commute by fastforce
    then show  $\exists x' \in K. x' \in M \wedge d \ x' \ x < \varepsilon$ 
  qed

```

```

    using ⟨K ⊆ S⟩ ⟨S ⊆ M⟩ ⟨x' ∈ K⟩ by force
  qed
  then show ∃ K. finite K ∧ K ⊆ mtopology_closure_of S ∧ mtopology_closure_of
S ⊆ (⋃ x ∈ K. mball x ε)
    using closure_of_subset_Int ⟨K ⊆ S⟩ ⟨finite K⟩ K by fastforce
  qed
  then show ?thesis
    by (simp add: assms inf.absorb2 mtotally_bounded_imp_subset)
  qed

```

lemma *mtotally_bounded_closure_of_eq*:

$S \subseteq M \implies \text{mtotally_bounded} (\text{mtopology_closure_of } S) \iff \text{mtotally_bounded } S$

by (*metis closure_of_subset mtotally_bounded_closure_of mtotally_bounded_subset topspace_mtopology*)

lemma *mtotally_bounded_cauchy_sequence*:

assumes *MCauchy* σ

shows *mtotally_bounded* (*range* σ)

unfolding *MCauchy_def* *mtotally_bounded_def*

proof (*intro strip*)

fix $\varepsilon :: \text{real}$

assume $\varepsilon > 0$

then obtain N where $\bigwedge n. N \leq n \implies d (\sigma N) (\sigma n) < \varepsilon$

using *assms* by (*force simp: MCauchy_def*)

then have $\bigwedge m. \exists n \leq N. \sigma n \in M \wedge \sigma m \in M \wedge d (\sigma n) (\sigma m) < \varepsilon$

by (*metis MCauchy_def assms mdist_zero nle_le range_subsetD*)

then

show $\exists K. \text{finite } K \wedge K \subseteq \text{range } \sigma \wedge \text{range } \sigma \subseteq (\bigcup x \in K. \text{mball } x \varepsilon)$

by (*rule_tac x = σ ' {0..N} in exI*) *force*

qed

lemma *MCauchy_imp_mbounded*:

MCauchy $\sigma \implies \text{mbounded} (\text{range } \sigma)$

by (*simp add: mtotally_bounded_cauchy_sequence mtotally_bounded_imp_mbounded*)

6.7.12 Compactness in metric spaces

lemma *Bolzano_Weierstrass_property*:

assumes $S \subseteq U$ $S \subseteq M$

shows

$(\forall \sigma :: \text{nat} \Rightarrow 'a. \text{range } \sigma \subseteq S$

$\implies (\exists l r. l \in U \wedge \text{strict_mono } r \wedge \text{limitin } \text{mtopology} (\sigma \circ r) l \text{ sequentially}))$

\iff

$(\forall T. T \subseteq S \wedge \text{infinite } T \implies U \cap \text{mtopology_derived_set_of } T \neq \{\})$ (*is*

?lhs=?rhs)

proof

assume *L*: *?lhs*

show *?rhs*

```

proof clarify
  fix T
  assume T ⊆ S and infinite T
  and T: U ∩ mtopology derived_set_of T = {}
  then obtain σ :: nat ⇒ 'a where inj σ range σ ⊆ T
  by (meson infinite_countable_subset)
  with L obtain l r where l ∈ U strict_mono r
  and lr: limitin mtopology (σ ∘ r) l sequentially
  by (meson ‹T ⊆ S› subset_trans)
  then obtain ε where ε > 0 and ε: ⋀y. y ∈ T ⇒ y = l ∨ ¬ d l y < ε
  using T ‹T ⊆ S› ‹S ⊆ M›
  by (force simp: metric_derived_set_of limitin_metric disjoint_iff)
  with lr have ∀F n in sequentially. σ (r n) ∈ M ∧ d (σ (r n)) l < ε
  by (auto simp: limitin_metric)
  then obtain N where N: d (σ (r N)) l < ε d (σ (r (Suc N))) l < ε
  using less_le_not_le by (auto simp: eventually_sequentially)
  moreover have σ (r N) ≠ l ∨ σ (r (Suc N)) ≠ l
  by (meson ‹inj σ› ‹strict_mono r› injD n_not_Suc_n strict_mono_eq)
  ultimately
  show False
  using ε ‹range σ ⊆ T› commute by fastforce
qed
next
assume R: ?rhs
show ?lhs
proof (intro strip)
  fix σ :: nat ⇒ 'a
  assume range σ ⊆ S
  show ∃ l r. l ∈ U ∧ strict_mono r ∧ limitin mtopology (σ ∘ r) l sequentially
  proof (cases finite (range σ))
    case True
    then obtain m where infinite (σ - ' {σ m})
    by (metis image_iff inf_img_fin_dom nat_not_finite)
    then obtain r where [iff]: strict_mono r and r: ⋀n::nat. r n ∈ σ - ' {σ
m}
    using infinite_enumerate by blast
    have [iff]: σ m ∈ U σ m ∈ M
    using ‹range σ ⊆ S› assms by blast+
    show ?thesis
    proof (intro conjI exI)
      show limitin mtopology (σ ∘ r) (σ m) sequentially
      using r by (simp add: limitin_metric)
    qed auto
  next
  case False
  then obtain l where l ∈ U and l: l ∈ mtopology derived_set_of (range σ)
  by (meson R ‹range σ ⊆ S› disjoint_iff)
  then obtain g where g: ⋀ε. ε > 0 ⇒ σ (g ε) ≠ l ∧ d l (σ (g ε)) < ε
  by (simp add: metric_derived_set_of) metis

```

```

have range  $\sigma \subseteq M$ 
  using  $\langle \text{range } \sigma \subseteq S \rangle$  assms by auto
have  $l \in M$ 
  using  $l$  metric_derived_set_of by auto
define E where — a construction to ensure monotonicity
   $E \equiv \lambda \text{rec } n. \text{insert } (\text{inverse } (\text{Suc } n)) ((\lambda i. d \ l \ (\sigma \ i)) \ ' (\bigcup_{k < n. \{0..rec \ k\}}))$ 
-  $\{0\}$ 
define r where  $r \equiv \text{wfrec } \text{less\_than} \ (\lambda \text{rec } n. g \ (\text{Min } (E \ \text{rec } \ n)))$ 
have  $(\bigcup_{k < n. \{0..cut \ r \ \text{less\_than } \ n \ k\}}) = (\bigcup_{k < n. \{0..r \ k\})$  for n
  by (auto simp: cut_apply)
then have  $r\_eq: r \ n = g \ (\text{Min } (E \ r \ n))$  for n
  by (metis E_def def_wfrec [OF r_def] wf_less_than)
have  $dl\_pos[simp]: d \ l \ (\sigma \ (r \ n)) > 0$  for n
  using wf_less_than
proof (induction n rule: wf_induct_rule)
  case (less n)
  then have  $*$ :  $\text{Min } (E \ r \ n) > 0$ 
    using  $\langle l \in M \rangle \langle \text{range } \sigma \subseteq M \rangle$  by (auto simp: E_def image_subset_iff)
  show ?case
    using  $g \ [OF \ *] \ r\_eq \ [of \ n]$ 
    by (metis  $\langle l \in M \rangle \langle \text{range } \sigma \subseteq M \rangle \text{mdist\_pos\_less range\_subsetD})$ 
qed
then have non_l:  $\sigma \ (r \ n) \neq l$  for n
  using  $\langle \text{range } \sigma \subseteq M \rangle \text{mdist\_pos\_eq}$  by blast
have Min_pos:  $\text{Min } (E \ r \ n) > 0$  for n
using  $dl\_pos \ \langle l \in M \rangle \langle \text{range } \sigma \subseteq M \rangle$  by (auto simp: E_def image_subset_iff)
have d_small:  $d \ (\sigma \ (r \ n)) \ l < \text{inverse}(\text{Suc } n)$  for n
proof -
  have  $d \ (\sigma \ (r \ n)) \ l < \text{Min } (E \ r \ n)$ 
    by (simp add:  $\langle 0 < \text{Min } (E \ r \ n) \rangle \text{commute } g \ r\_eq)$ 
  also have  $\dots \leq \text{inverse}(\text{Suc } n)$ 
    by (simp add: E_def)
  finally show ?thesis .
qed
have d_lt_d:  $d \ l \ (\sigma \ (r \ n)) < d \ l \ (\sigma \ i)$  if  $\S: p < n \ i \leq r \ p \ \sigma \ i \neq l$  for i p n
proof -
  have  $1: d \ l \ (\sigma \ i) \in E \ r \ n$ 
    using  $\S \ \langle l \in M \rangle \langle \text{range } \sigma \subseteq M \rangle$ 
    by (force simp: E_def image_subset_iff image_iff)
  have  $d \ l \ (\sigma \ (g \ (\text{Min } (E \ r \ n)))) < \text{Min } (E \ r \ n)$ 
    by (rule conjunct2 [OF g [OF Min_pos]])
  also have  $\text{Min } (E \ r \ n) \leq d \ l \ (\sigma \ i)$ 
    using  $1$  unfolding E_def by (force intro!: Min.coboundedI)
  finally show ?thesis
    by (simp add: r_eq)
qed
have r:  $r \ p < r \ n$  if  $p < n$  for p n
using d_lt_d [OF that] non_l by (meson linorder_not_le order_less_irrefl)

```

```

show ?thesis
proof (intro exI conjI)
  show strict_mono r
    by (simp add: r strict_monoI)
  show limitin_mtopology ( $\sigma \circ r$ ) l sequentially
    unfolding limitin_metric
  proof (intro conjI strip ⟨l ∈ M⟩)
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    then have  $\forall_F n$  in sequentially.  $\text{inverse}(\text{Suc } n) < \varepsilon$ 
      using Archimedean_eventually_inverse by auto
    then show  $\forall_F n$  in sequentially.  $(\sigma \circ r) n \in M \wedge d((\sigma \circ r) n) l < \varepsilon$ 
      by (smt (verit) ⟨range  $\sigma \subseteq M$ ⟩ commute_comp_apply d_small eventually_mono range_subsetD)
    qed
  qed (use ⟨l ∈ U⟩ in auto)
qed
qed
qed

```

More on Bolzano Weierstrass

lemma Bolzano_Weierstrass_A:
assumes compactin_mtopology $S T \subseteq S$ infinite T
shows $S \cap \text{mtopology_derived_set_of } T \neq \{\}$
by (simp add: assms compactin_imp_Bolzano_Weierstrass)

lemma Bolzano_Weierstrass_B:
fixes $\sigma :: \text{nat} \Rightarrow 'a$
assumes $S \subseteq M$ range $\sigma \subseteq S$
and $\bigwedge T. [T \subseteq S \text{ infinite } T] \implies S \cap \text{mtopology_derived_set_of } T \neq \{\}$
shows $\exists l r. l \in S \wedge \text{strict_mono } r \wedge \text{limitin_mtopology } (\sigma \circ r) l \text{ sequentially}$
using Bolzano_Weierstrass_property assms **by** blast

lemma Bolzano_Weierstrass_C:
assumes $S \subseteq M$
assumes $\bigwedge \sigma :: \text{nat} \Rightarrow 'a. \text{range } \sigma \subseteq S \implies (\exists l r. l \in S \wedge \text{strict_mono } r \wedge \text{limitin_mtopology } (\sigma \circ r) l \text{ sequentially})$
shows mtotally_bounded S
unfolding mtotally_bounded_sequentially
by (metis convergent_imp_MCauchy assms image_comp image_mono subset_UNIV subset_trans)

lemma Bolzano_Weierstrass_D:
assumes $S \subseteq M$ $S \subseteq \bigcup \mathcal{C}$ **and** $\text{ope } U: \bigwedge U. U \in \mathcal{C} \implies \text{openin_mtopology } U$
assumes $\S: (\forall \sigma :: \text{nat} \Rightarrow 'a. \text{range } \sigma \subseteq S \longrightarrow (\exists l r. l \in S \wedge \text{strict_mono } r \wedge \text{limitin_mtopology } (\sigma \circ r) l \text{ sequentially}))$
shows $\exists \varepsilon > 0. \forall x \in S. \exists U \in \mathcal{C}. \text{mball } x \varepsilon \subseteq U$
proof (rule ccontr)

assume $\neg (\exists \varepsilon > 0. \forall x \in S. \exists U \in \mathcal{C}. \text{mball } x \ \varepsilon \subseteq U)$
then have $\forall n. \exists x \in S. \forall U \in \mathcal{C}. \neg \text{mball } x \ (\text{inverse } (Suc \ n)) \subseteq U$
by *simp*
then obtain σ **where** $\bigwedge n. \sigma \ n \in S$
and $\sigma: \bigwedge n \ U. U \in \mathcal{C} \implies \neg \text{mball } (\sigma \ n) \ (\text{inverse } (Suc \ n)) \subseteq U$
by *metis*
then obtain $l \ r$ **where** $l \in S$ *strict_mono* r
and $lr: \text{limitin } mtopology \ (\sigma \circ r) \ l$ *sequentially*
by (*meson* § *image_subsetI*)
with $\langle S \subseteq \bigcup \mathcal{C} \rangle$ **obtain** B **where** $l \in B \ B \in \mathcal{C}$
by *auto*
then obtain ε **where** $\varepsilon > 0$ **and** $\varepsilon: \bigwedge z. \llbracket z \in M; d \ z \ l < \varepsilon \rrbracket \implies z \in B$
by (*metis* *opeU* [*OF* $\langle B \in \mathcal{C} \rangle$] *commute* *in_mball* *openin_mtopology* *subset_iff*)
then have $\forall_F \ n$ *in* *sequentially*. $\sigma \ (r \ n) \in M \wedge d \ (\sigma \ (r \ n)) \ l < \varepsilon/2$
using lr *half_gt_zero* **unfolding** *limitin_metric_o_def* **by** *blast*
moreover have $\forall_F \ n$ *in* *sequentially*. $\text{inverse } (\text{real } (Suc \ n)) < \varepsilon/2$
using *Archimedean_eventually_inverse* $\langle 0 < \varepsilon \rangle$ *half_gt_zero* **by** *blast*
ultimately obtain n **where** $n: d \ (\sigma \ (r \ n)) \ l < \varepsilon/2$ $\text{inverse } (\text{real } (Suc \ n)) < \varepsilon/2$
by (*smt* (*verit*, *del_insts*) *eventually_sequentially* *le_add1* *le_add2*)
have $x \in B$ **if** $d \ (\sigma \ (r \ n)) \ x < \text{inverse } (Suc \ (r \ n))$ $x \in M$ **for** x
proof –
have $rle: \text{inverse } (\text{real } (Suc \ (r \ n))) \leq \text{inverse } (\text{real } (Suc \ n))$
using $\langle \text{strict_mono } r \rangle$ *strict_mono_imp_increasing* **by** *auto*
have $d \ x \ l \leq d \ (\sigma \ (r \ n)) \ x + d \ (\sigma \ (r \ n)) \ l$
using *that* **by** (*metis* *triangle* $\langle \bigwedge n. \sigma \ n \in S \rangle$ $\langle l \in S \rangle$ $\langle S \subseteq M \rangle$ *commute* *subsetD*)
also have $\dots < \varepsilon$
using *that* n rle **by** *linarith*
finally show *?thesis*
by (*simp* *add: \varepsilon* *that*)
qed
then show *False*
using σ [*of* $B \ r \ n$] **by** (*simp* *add: \langle B \in \mathcal{C} \rangle* *subset_iff*)
qed

lemma *Bolzano_Weierstrass_E*:

assumes *mtotally_bounded* $S \ S \subseteq M$
and $S: \bigwedge \mathcal{C}. \llbracket \bigwedge U. U \in \mathcal{C} \implies \text{openin } mtopology \ U; S \subseteq \bigcup \mathcal{C} \rrbracket \implies \exists \varepsilon > 0. \forall x \in S. \exists U \in \mathcal{C}. \text{mball } x \ \varepsilon \subseteq U$
shows *compactin* *mtopology* S
proof (*clarsimp* *simp: compactin_def* *assms*)
fix $\mathcal{U} :: 'a$ *set* *set*
assume $\mathcal{U}: \forall x \in \mathcal{U}. \text{openin } mtopology \ x$ **and** $S \subseteq \bigcup \mathcal{U}$
then obtain ε **where** $\varepsilon > 0$ **and** $\varepsilon: \bigwedge x. x \in S \implies \exists U \in \mathcal{U}. \text{mball } x \ \varepsilon \subseteq U$
by (*metis* S)
then obtain f **where** $f: \bigwedge x. x \in S \implies f \ x \in \mathcal{U} \wedge \text{mball } x \ \varepsilon \subseteq f \ x$
by *metis*
then obtain K **where** *finite* $K \ K \subseteq S$ **and** $K: S \subseteq (\bigcup x \in K. \text{mball } x \ \varepsilon)$


```

  by (metis ‹0 < ε› ‹mtotally_bounded S› mtotally_bounded_def)
show ∃F. finite F ∧ F ⊆ U ∧ S ⊆ ⋃F
proof (intro conjI exI)
  show finite (f ‘ K)
  by (simp add: ‹finite K›)
  show f ‘ K ⊆ U
  using ‹K ⊆ S› f by blast
  show S ⊆ ⋃(f ‘ K)
  using K ‹K ⊆ S› by (force dest: f)
qed
qed

```

lemma *compactin_eq_Bolzano_Weierstrass*:

```

compactin mtopology S ↔
  S ⊆ M ∧ (∀ T. T ⊆ S ∧ infinite T → S ∩ mtopology derived_set_of T ≠ {})
using Bolzano_Weierstrass_C Bolzano_Weierstrass_D Bolzano_Weierstrass_E
by (smt (verit, del_insts) Bolzano_Weierstrass_property compactin_imp_Bolzano_Weierstrass
compactin_subspace subset_refl topspace_mtopology)

```

lemma *compactin_sequentially*:

```

shows compactin mtopology S ↔
  S ⊆ M ∧
  ((∀ σ::nat⇒'a. range σ ⊆ S
  → (∃ l r. l ∈ S ∧ strict_mono r ∧ limitin mtopology (σ ∘ r) l sequentially)))
by (metis Bolzano_Weierstrass_property compactin_eq_Bolzano_Weierstrass
subset_refl)

```

lemma *compactin_imp_mtotally_bounded*:

```

compactin mtopology S ⇒ mtotally_bounded S
by (simp add: Bolzano_Weierstrass_C compactin_sequentially)

```

lemma *lebesgue_number*:

```

[[compactin mtopology S; S ⊆ ⋃C; ∧U. U ∈ C ⇒ openin mtopology U]]
⇒ ∃ε>0. ∀x ∈ S. ∃U ∈ C. mball x ε ⊆ U
by (simp add: Bolzano_Weierstrass_D compactin_sequentially)

```

lemma *compact_space_sequentially*:

```

compact_space mtopology ↔
  (∀ σ::nat⇒'a. range σ ⊆ M
  → (∃ l r. l ∈ M ∧ strict_mono r ∧ limitin mtopology (σ ∘ r) l sequentially))
by (simp add: compact_space_def compactin_sequentially)

```

lemma *compact_space_eq_Bolzano_Weierstrass*:

```

compact_space mtopology ↔
  (∀ S. S ⊆ M ∧ infinite S → mtopology derived_set_of S ≠ {})
using Int_absorb1 [OF derived_set_of_subset_topspace [of mtopology]]
by (force simp: compact_space_def compactin_eq_Bolzano_Weierstrass)

```

```

lemma compact_space_nest:
  compact_space_mtopology  $\longleftrightarrow$ 
    ( $\forall C. (\forall n::nat. \text{closedin\_mtopology } (C\ n)) \wedge (\forall n. C\ n \neq \{\}) \wedge \text{decseq } C \longrightarrow$ 
 $\bigcap (\text{range } C) \neq \{\}$ )
  (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof clarify
    fix C :: nat  $\Rightarrow$  'a set
    assume  $\forall n. \text{closedin\_mtopology } (C\ n)$ 
    and  $\forall n. C\ n \neq \{\}$ 
    and decseq C
    and  $\bigcap (\text{range } C) = \{\}$ 
    then obtain K where K: finite K  $\cap (C\ 'K) = \{\}$ 
    by (metis L compact_space_imp_nest)
    then obtain k where K  $\subseteq \{..k\}$ 
    using finite_nat_iff_bounded_le by auto
    then have C k  $\subseteq \bigcap (C\ 'K)$ 
    using <decseq C> by (auto simp:decseq_def)
    then show False
    by (simp add: K < $\forall n. C\ n \neq \{\}$ >)
  qed
next
  assume R [rule_format]: ?rhs
  show ?lhs
    unfolding compact_space_sequentially
  proof (intro strip)
    fix  $\sigma :: nat \Rightarrow$  'a
    assume  $\sigma: \text{range } \sigma \subseteq M$ 
    have mtopology_closure_of  $\sigma\ ' \{n..\}$   $\neq \{\}$  for n
    using <range  $\sigma \subseteq M$ > by (auto simp: closure_of_eq_empty_image_subset_iff)
    moreover have decseq ( $\lambda n. \text{mtopology\_closure\_of } \sigma\ ' \{n..\}$ )
    using closure_of_mono_image_mono by (smt (verit) atLeast_subset_iff
decseq_def)
    ultimately obtain l where l:  $\bigwedge n. l \in \text{mtopology\_closure\_of } \sigma\ ' \{n..\}$ 
    using R [of  $\lambda n. \text{mtopology\_closure\_of } (\sigma\ ' \{n..\})$ ] by auto
    then have l  $\in M$  and  $\bigwedge n. \forall r>0. \exists k \geq n. \sigma\ k \in M \wedge d\ l\ (\sigma\ k) < r$ 
    using metric_closure_of by fastforce+
    then obtain f where f:  $\bigwedge n\ r. r>0 \implies f\ n\ r \geq n \wedge \sigma\ (f\ n\ r) \in M \wedge d\ l\ (\sigma\ (f\ n\ r)) < r$ 
    by metis
    define r where r = rec_nat (f 0 1) ( $\lambda n\ rec. (f\ (Suc\ rec)\ (inverse\ (Suc\ (Suc\ n))))$ )
    have r: d l ( $\sigma\ (r\ n)$ ) < inverse(Suc n) for n
    by (induction n) (auto simp: rec_nat_0_imp [OF r_def] rec_nat_Suc_imp [OF r_def] f)
    have r n < r(Suc n) for n
    by (simp add: Suc_le_lessD f r_def)

```

```

then have strict_mono r
  by (simp add: strict_mono_Suc_iff)
moreover have limitin_mtopology ( $\sigma \circ r$ ) l sequentially
proof (clarsimp simp: limitin_metric ⟨ $l \in M$ ⟩)
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  then have ( $\forall_F n$  in sequentially.  $\text{inverse} (\text{real} (\text{Suc } n)) < \varepsilon$ )
    using Archimedean_eventually_inverse by blast
  then show  $\forall_F n$  in sequentially.  $\sigma (r n) \in M \wedge d (\sigma (r n)) l < \varepsilon$ 
    by eventually_elim (metis commute ⟨ $\text{range } \sigma \subseteq M$ ⟩ order_less_trans r
range_subsetD)
qed
ultimately show  $\exists l r. l \in M \wedge \text{strict\_mono } r \wedge \text{limitin\_mtopology } (\sigma \circ r) l$ 
  sequentially
  using ⟨ $l \in M$ ⟩ by blast
qed
qed

```

```

lemma (in discrete_metric) mcomplete_discrete_metric: disc.mcomplete
proof (clarsimp simp: disc.mcomplete_def)
  fix  $\sigma :: \text{nat} \Rightarrow 'a$ 
  assume disc.MCauchy  $\sigma$ 
  then obtain N where  $\bigwedge n. N \leq n \implies \sigma N = \sigma n$ 
  unfolding disc.MCauchy_def by (metis dd_def dual_order_refl order_less_irrefl
zero_less_one)
  moreover have  $\text{range } \sigma \subseteq M$ 
  using ⟨disc.MCauchy  $\sigma$ ⟩ disc.MCauchy_def by blast
  ultimately have limitin_disc_mtopology  $\sigma$  ( $\sigma N$ ) sequentially
  by (metis disc.limit_metric_sequentially disc.zero_range_subsetD)
  then show  $\exists x. \text{limitin\_disc\_mtopology } \sigma x$  sequentially ..
qed

```

```

lemma compact_space_imp_mcomplete: compact_space_mtopology  $\implies$  mcomplete
  by (simp add: compact_space_nest mcomplete_nest)

```

```

lemma (in Submetric) compactin_imp_mcomplete:
  compactin_mtopology A  $\implies$  sub.mcomplete
  by (simp add: compactin_subspace_mtopology_submetric sub.compact_space_imp_mcomplete)

```

```

lemma (in Submetric) mcomplete_imp_closedin:
  assumes sub.mcomplete
  shows closedin_mtopology A
proof -
  have  $l \in A$ 
  if  $\text{range } \sigma \subseteq A$  and  $l: \text{limitin\_mtopology } \sigma l$  sequentially
  for  $\sigma :: \text{nat} \Rightarrow 'a$  and  $l$ 
proof -
  have sub.MCauchy  $\sigma$ 

```

```

    using convergent_imp_MCauchy_subset that by (force simp: MCauchy_submetric)
    then have limitin_sub.mtopology  $\sigma$  l sequentially
      using assms unfolding sub.mcomplete_def
      using l limitin_metric_unique limitin_submetric_iff trivial_limit_sequentially
  by blast
  then show ?thesis
    using limitin_submetric_iff by blast
  qed
  then show ?thesis
    using metric_closedin_iff_sequentially_closed_subset by auto
  qed

```

lemma (in Submetric) closedin_eq_mcomplete:
 $mcomplete \implies (closedin\ mtopology\ A \longleftrightarrow sub.mcomplete)$
 using closedin_mcomplete_imp_mcomplete mcomplete_imp_closedin by blast

lemma compact_space_eq_mcomplete_mtotally_bounded:
 $compact_space\ mtopology \longleftrightarrow mcomplete \wedge mtotally_bounded\ M$
 by (meson Bolzano_Weierstrass_C compact_space_imp_mcomplete compact_space_sequentially
 limitin_mspace
 mcomplete_alt mtotally_bounded_sequentially_subset_refl)

lemma compact_closure_of_imp_mtotally_bounded:
 $\llbracket compactin\ mtopology\ (mtopology\ closure_of\ S); S \subseteq M \rrbracket$
 $\implies mtotally_bounded\ S$
 using compactin_imp_mtotally_bounded mtotally_bounded_closure_of_eq by
 blast

lemma mtotally_bounded_eq_compact_closure_of:
 assumes mcomplete
 shows mtotally_bounded $S \longleftrightarrow S \subseteq M \wedge compactin\ mtopology\ (mtopology\ clo-$
 $sure_of\ S)$
 (is ?lhs=?rhs)
proof
 assume L: ?lhs
 show ?rhs
 unfolding compactin_subspace
proof (intro conjI)
 show $S \subseteq M$
 using L by (simp add: mtotally_bounded_imp_subset)
 show $mtopology\ closure_of\ S \subseteq topspace\ mtopology$
 by (simp add: $\langle S \subseteq M \rangle$ closure_of_minimal)
 then have MSM: $mtopology\ closure_of\ S \subseteq M$
 by auto
 interpret S: Submetric M d mtopology closure_of S
proof qed (use MSM in auto)
 have S.sub.mtotally_bounded (mtopology closure_of S)
 using L mtotally_bounded_absolute mtotally_bounded_closure_of by blast

```

then
  show compact_space (subtopology mtopology (mtopology closure_of S))
  using S.closedin_mcomplete_imp_mcomplete S.mtopology_submetric S.sub.compact_space_eq_mcomplete_m
assms by force
qed
qed (auto simp: compact_closure_of_imp_mtotally_bounded)

```

```

lemma compact_closure_of_eq_Bolzano_Weierstrass:
  compactin mtopology (mtopology closure_of S)  $\longleftrightarrow$ 
  ( $\forall T. \text{infinite } T \wedge T \subseteq S \wedge T \subseteq M \longrightarrow \text{mtopology derived\_set\_of } T \neq \{\}$ )
(is ?lhs=?rhs)

```

```

proof
  assume L: ?lhs
  show ?rhs
  proof (intro strip)
    fix T
    assume T: infinite T  $\wedge$  T  $\subseteq$  S  $\wedge$  T  $\subseteq$  M
    show mtopology derived_set_of T  $\neq$  {}
    proof (intro compact_closure_of_imp_Bolzano_Weierstrass)
      show compactin mtopology (mtopology closure_of S)
      by (simp add: L)
    qed (use T in auto)
  qed
next
  have compactin mtopology (mtopology closure_of S)
  if  $\S: \bigwedge T. [\text{infinite } T; T \subseteq S] \implies \text{mtopology derived\_set\_of } T \neq \{\}$  and S  $\subseteq$ 
M for S
  unfolding compactin_sequentially
  proof (intro conjI strip)
    show MSM: mtopology closure_of S  $\subseteq$  M
    using closure_of_subset_topspace by fastforce
    fix  $\sigma :: \text{nat} \Rightarrow 'a$ 
    assume  $\sigma$ : range  $\sigma \subseteq$  mtopology closure_of S
    then have  $\exists y \in S. d (\sigma n) y < \text{inverse}(\text{Suc } n)$  for n
    by (simp add: metric_closure_of_image_subset_iff) (metis inverse_Suc
of_nat_Suc)
    then obtain  $\tau$  where  $\tau: \bigwedge n. \tau n \in S \wedge d (\sigma n) (\tau n) < \text{inverse}(\text{Suc } n)$ 
by metis
    then have range  $\tau \subseteq$  S
    by blast
  moreover
  have *:  $\forall T. T \subseteq S \wedge \text{infinite } T \longrightarrow \text{mtopology closure\_of } S \cap \text{mtopology}$ 
derived_set_of T  $\neq$  {}
  using  $\S(1)$  derived_set_of_mono derived_set_of_subset_closure_of by
fastforce
  moreover have S  $\subseteq$  mtopology closure_of S
  by (simp add:  $\langle S \subseteq M \rangle$  closure_of_subset)

```

ultimately obtain $l r$ where lr :
 $l \in \text{mtopology_closure_of } S \text{ strict_mono } r \text{ limitin_mtopology } (\tau \circ r) l \text{ sequentially}$
using *Bolzano_Weierstrass_property* $\langle S \subseteq M \rangle$ **by** *metis*
then have $l \in M$
using *limitin_mspace* **by** *blast*
have *dr_less*: $d((\sigma \circ r) n) ((\tau \circ r) n) < \text{inverse}(\text{Suc } n)$ **for** n
proof –
have $d((\sigma \circ r) n) ((\tau \circ r) n) < \text{inverse}(\text{Suc } (r n))$
using τ **by** *auto*
also have $\dots \leq \text{inverse}(\text{Suc } n)$
using *lr strict_mono_imp_increasing* **by** *auto*
finally show *?thesis* .
qed
have *limitin_mtopology* $(\sigma \circ r) l$ *sequentially*
unfolding *limitin_metric*
proof (*intro conjI strip*)
show $l \in M$
using *limitin_mspace lr* **by** *blast*
fix $\varepsilon :: \text{real}$
assume $\varepsilon > 0$
then have $\forall_F n$ *in sequentially*. $(\tau \circ r) n \in M \wedge d((\tau \circ r) n) l < \varepsilon/2$
using *lr half_gt_zero limitin_metric* **by** *blast*
moreover have $\forall_F n$ *in sequentially*. $\text{inverse}(\text{real } (\text{Suc } n)) < \varepsilon/2$
using *Archimedean_eventually_inverse* $\langle 0 < \varepsilon \rangle$ *half_gt_zero* **by** *blast*
then have $\forall_F n$ *in sequentially*. $d((\sigma \circ r) n) ((\tau \circ r) n) < \varepsilon/2$
by *eventually_elim* (*smt (verit, del_insts) dr_less*)
ultimately have $\forall_F n$ *in sequentially*. $d((\sigma \circ r) n) l < \varepsilon/2 + \varepsilon/2$
by *eventually_elim* (*smt (verit) triangle* $\langle l \in M \rangle$ *MSM* σ *comp_apply*
order_trans range_subsetD)
then show $\forall_F n$ *in sequentially*. $(\sigma \circ r) n \in M \wedge d((\sigma \circ r) n) l < \varepsilon$
apply *eventually_elim*
using $\langle \text{mtopology_closure_of } S \subseteq M \rangle$ σ **by** *auto*
qed
with lr **show** $\exists l r. l \in \text{mtopology_closure_of } S \wedge \text{strict_mono } r \wedge \text{limitin_mtopology } (\sigma \circ r) l \text{ sequentially}$
by *blast*
qed
then show *?rhs* \implies *?lhs*
by (*metis Int_subset_iff closure_of_restrict inf_le1 topspace_mtopology*)
qed
end

lemma (*in discrete_metric*) *mtotally_bounded_discrete_metric*:
 $\text{disc.mtotally_bounded } S \longleftrightarrow \text{finite } S \wedge S \subseteq M$ (**is** *?lhs=?rhs*)
proof
assume L : *?lhs*
show *?rhs*

```

proof
  show finite S
    by (metis (no_types) L closure_of_subset_Int compactin_discrete_topology
disc.mtotally_bounded_eq_compact_closure_of
disc.topspace_mtopology discrete_metric.mcomplete_discrete_metric
inf.absorb_iff2 mtopology_discrete_metric finite_subset)
    show  $S \subseteq M$ 
    by (simp add: L disc.mtotally_bounded_imp_subset)
  qed
qed (simp add: disc.finite_imp_mtotally_bounded)

```

```

context Metric_space
begin

```

```

lemma derived_set_of_infinite_openin_metric:
  mtopology derived_set_of S =
  { $x \in M. \forall U. x \in U \wedge \text{openin } mtopology U \longrightarrow \text{infinite}(S \cap U)$ }
  by (simp add: derived_set_of_infinite_openin Hausdorff_space_mtopology)

```

```

lemma derived_set_of_infinite_1:
  assumes infinite (S  $\cap$  mball x  $\epsilon$ )
  shows infinite (S  $\cap$  mcball x  $\epsilon$ )
  by (meson Int_mono assms finite_subset mball_subset_mcball subset_refl)

```

```

lemma derived_set_of_infinite_2:
  assumes openin mtopology U  $\wedge \epsilon. 0 < \epsilon \implies \text{infinite}(S \cap \text{mcball } x \ \epsilon)$  and  $x \in U$ 
  shows infinite (S  $\cap$  U)
  by (metis assms openin_mtopology_mcball finite_Int inf.absorb_iff2 inf_assoc)

```

```

lemma derived_set_of_infinite_mball:
  mtopology derived_set_of S = { $x \in M. \forall e > 0. \text{infinite}(S \cap \text{mball } x \ e)$ }
  unfolding derived_set_of_infinite_openin_metric
  by (metis (no_types, opaque_lifting) centre_in_mball_iff_openin_mball derived_set_of_infinite_1
derived_set_of_infinite_2)

```

```

lemma derived_set_of_infinite_mcball:
  mtopology derived_set_of S = { $x \in M. \forall e > 0. \text{infinite}(S \cap \text{mcball } x \ e)$ }
  unfolding derived_set_of_infinite_openin_metric
  by (metis (no_types, opaque_lifting) centre_in_mball_iff_openin_mball derived_set_of_infinite_1
derived_set_of_infinite_2)

```

```

end

```

6.7.13 Continuous functions on metric spaces

```

context Metric_space
begin

```

lemma *continuous_map_to_metric*:
 $continuous_map\ X\ mtopology\ f \longleftrightarrow$
 $(\forall x \in topspace\ X. \forall \varepsilon > 0. \exists U. openin\ X\ U \wedge x \in U \wedge (\forall y \in U. f\ y \in mball\ (f\ x)\ \varepsilon))$
(is ?lhs=?rhs)
proof
show $?lhs \implies ?rhs$
unfolding *continuous_map_eq_topcontinuous_at_topcontinuous_at_def*
by (*metis PiE centre_in_mball_iff openin_mball topspace_mtopology*)
next
assume $R: ?rhs$
then have $\forall x \in topspace\ X. f\ x \in M$
by (*meson gt_ex in_mball*)
moreover
have $\bigwedge x \in V. \llbracket x \in topspace\ X; openin\ mtopology\ V; f\ x \in V \rrbracket \implies \exists U. openin\ X\ U \wedge x \in U \wedge (\forall y \in U. f\ y \in V)$
unfolding *openin_mtopology* **by** (*metis Int_iff R inf.orderE*)
ultimately
show $?lhs$
by (*simp add: continuous_map_eq_topcontinuous_at_topcontinuous_at_def*)
qed

lemma *continuous_map_from_metric*:
 $continuous_map\ mtopology\ X\ f \longleftrightarrow$
 $f \in M \rightarrow topspace\ X \wedge$
 $(\forall a \in M. \forall U. openin\ X\ U \wedge f\ a \in U \longrightarrow (\exists r > 0. \forall x. x \in M \wedge d\ a\ x < r \longrightarrow f\ x \in U))$
proof (*cases f ' M \subseteq topspace X*)
case *True*
then show $?thesis$
by (*fastforce simp: continuous_map openin_mtopology subset_eq*)
next
case *False*
then show $?thesis$
by (*simp add: continuous_map_def image_subset_iff_funcset*)
qed

An abstract formulation, since the limits do not have to be sequential

lemma *continuous_map_uniform_limit*:
assumes *contf*: $\forall_F\ \xi\ in\ F. continuous_map\ X\ mtopology\ (f\ \xi)$
and *dfg*: $\bigwedge \varepsilon. 0 < \varepsilon \implies \forall_F\ \xi\ in\ F. \forall x \in topspace\ X. g\ x \in M \wedge d\ (f\ \xi\ x)\ (g\ x) < \varepsilon$
and *nontriv*: $\neg\ trivial_limit\ F$
shows $continuous_map\ X\ mtopology\ g$
unfolding *continuous_map_to_metric*
proof (*intro strip*)
fix x **and** $\varepsilon::real$
assume $x \in topspace\ X$ **and** $\varepsilon > 0$


```

then obtain  $\xi$  where  $k$ : continuous_map X mtopology (f  $\xi$ )
and  $gM$ :  $\forall x \in \text{topspace } X. g\ x \in M$ 
and  $third$ :  $\forall x \in \text{topspace } X. d\ (f\ \xi\ x)\ (g\ x) < \varepsilon/3$ 
using eventually_conj [OF contf] contf dfg [of  $\varepsilon/3$ ] eventually_happens' [OF
nontriv]
by (smt (verit, ccfv_SIG) zero_less_divide_iff)
then obtain  $U$  where  $U$ : openin X U  $x \in U$  and  $Uthird$ :  $\forall y \in U. d\ (f\ \xi\ y)\ (f$ 
 $\xi\ x) < \varepsilon/3$ 
unfolding continuous_map_to_metric
by (metis  $\langle 0 < \varepsilon \rangle \langle x \in \text{topspace } X \rangle$  commute_divide_pos_pos_in_mball zero_less_numeral)
have  $f\_inM$ :  $f\ \xi\ y \in M$  if  $y \in U$  for  $y$ 
using  $U\ k$  openin_subset that by (fastforce simp: continuous_map_def)
have  $d\ (g\ y)\ (g\ x) < \varepsilon$  if  $y \in U$  for  $y$ 
proof -
have  $g\ y \in M$ 
using  $U\ gM$  openin_subset that by blast
have  $d\ (g\ y)\ (g\ x) \leq d\ (g\ y)\ (f\ \xi\ x) + d\ (f\ \xi\ x)\ (g\ x)$ 
by (simp add: U  $\langle g\ y \in M \rangle \langle x \in \text{topspace } X \rangle f\_inM\ gM$  triangle)
also have  $\dots \leq d\ (g\ y)\ (f\ \xi\ y) + d\ (f\ \xi\ y)\ (f\ \xi\ x) + d\ (f\ \xi\ x)\ (g\ x)$ 
by (simp add: U  $\langle g\ y \in M \rangle$  commute_f_inM that triangle')
also have  $\dots < \varepsilon/3 + \varepsilon/3 + \varepsilon/3$ 
by (smt (verit) U(1) Uthird  $\langle x \in \text{topspace } X \rangle$  commute_openin_subset subsetD
that third)
finally show ?thesis by simp
qed
with  $U\ gM$  show  $\exists U. \text{openin } X\ U \wedge x \in U \wedge (\forall y \in U. g\ y \in \text{mball } (g\ x)\ \varepsilon)$ 
by (metis commute_in_mball in_mono openin_subset)
qed

```

lemma *continuous_map_uniform_limit_alt*:

```

assumes  $contf$ :  $\forall_F\ \xi$  in F. continuous_map X mtopology (f  $\xi$ )
and  $gim$ :  $g \in \text{topspace } X \rightarrow M$ 
and  $dfg$ :  $\bigwedge \varepsilon. 0 < \varepsilon \implies \forall_F\ \xi$  in F.  $\forall x \in \text{topspace } X. d\ (f\ \xi\ x)\ (g\ x) < \varepsilon$ 
and  $nontriv$ :  $\neg$  trivial_limit F
shows continuous_map X mtopology g
proof (rule continuous_map_uniform_limit [OF contf])
fix  $\varepsilon :: \text{real}$ 
assume  $\varepsilon > 0$ 
with  $gim\ dfg$  show  $\forall_F\ \xi$  in F.  $\forall x \in \text{topspace } X. g\ x \in M \wedge d\ (f\ \xi\ x)\ (g\ x) < \varepsilon$ 
by (simp add: Pi_iff)
qed (use nontriv in auto)

```

lemma *continuous_map_uniformly_Cauchy_limit*:

```

assumes mcomplete
assumes  $contf$ :  $\forall_F\ n$  in sequentially. continuous_map X mtopology (f n)
and  $Cauchy'$ :  $\bigwedge \varepsilon. \varepsilon > 0 \implies \exists N. \forall m\ n\ x. N \leq m \longrightarrow N \leq n \longrightarrow x \in \text{topspace } X \longrightarrow d\ (f\ m\ x)\ (f\ n\ x) < \varepsilon$ 

```

obtains g where
continuous_map X *mtopology* g
 $\bigwedge \varepsilon. 0 < \varepsilon \implies \forall_F n$ in *sequentially*. $\forall x \in \text{topspace } X. d(f\ n\ x)(g\ x) < \varepsilon$
proof –
have $\bigwedge x. x \in \text{topspace } X \implies \exists l.$ *limitin* *mtopology* $(\lambda n. f\ n\ x)\ l$ *sequentially*
using $\langle mcomplete \rangle$ [*unfolded mcomplete, rule_format*] *assms*
unfolding *continuous_map_def* $Pi_iff\ \text{topspace_mtopology}$
by (*smt* (*verit, del_insts*) *eventually_mono*)
then obtain g where $g: \bigwedge x. x \in \text{topspace } X \implies \text{limitin } \text{mtopology } (\lambda n. f\ n\ x)$
 $(g\ x)$ *sequentially*
by *metis*
show *thesis*
proof
show $\forall_F n$ in *sequentially*. $\forall x \in \text{topspace } X. d(f\ n\ x)(g\ x) < \varepsilon$
if $\varepsilon > 0$ **for** $\varepsilon :: \text{real}$
proof –
obtain N where $N: \bigwedge m\ n\ x. \llbracket N \leq m; N \leq n; x \in \text{topspace } X \rrbracket \implies d(f\ m$
 $x)(f\ n\ x) < \varepsilon/2$
by (*meson* *Cauchy'* $\langle 0 < \varepsilon \rangle$ *half_gt_zero*)
obtain P where $P: \bigwedge n\ x. \llbracket n \geq P; x \in \text{topspace } X \rrbracket \implies f\ n\ x \in M$
using *contf* **by** (*auto simp: eventually_sequentially_continuous_map_def*)
show *?thesis*
proof (*intro eventually_sequentiallyI strip*)
fix $n\ x$
assume $\max\ N\ P \leq n$ **and** $x: x \in \text{topspace } X$
obtain L where $g\ x \in M$ and $L: \forall n \geq L. f\ n\ x \in M \wedge d(f\ n\ x)(g\ x) < \varepsilon/2$
using g [*OF* x] $\langle \varepsilon > 0 \rangle$ **unfolding** *limitin_metric*
by (*metis* (*no_types, lifting*) *eventually_sequentially_half_gt_zero*)
define n' where $n' \equiv \text{Max}\{L, N, P\}$
have $L': \forall m \geq n'. f\ m\ x \in M \wedge d(f\ m\ x)(g\ x) < \varepsilon/2$
using L **by** (*simp add: n'_def*)
moreover
have $d(f\ n\ x)(f\ n'\ x) < \varepsilon/2$
using N [*of* $n\ n'\ x$] $\langle \max\ N\ P \leq n \rangle$ $n'_def\ x$ **by** *fastforce*
ultimately have $d(f\ n\ x)(g\ x) < \varepsilon/2 + \varepsilon/2$
by (*smt* (*verit, ccfv_SIG*) P $\langle g\ x \in M \rangle$ $\langle \max\ N\ P \leq n \rangle$ *le_refl*
max.bounded_iff mdist_zero triangle' x)
then show $d(f\ n\ x)(g\ x) < \varepsilon$ **by** *simp*
qed
qed
then show *continuous_map* X *mtopology* g
by (*smt* (*verit, del_insts*) *eventually_mono* g *limitin_mspace* *trivial_limit_sequentially*
continuous_map_uniform_limit [*OF contf*])
qed
qed

lemma *metric_continuous_map:*
assumes *Metric_space* M' d'
shows

```

    continuous_map mtopology (Metric_space.mtopology M' d') f  $\longleftrightarrow$ 
    f ' M  $\subseteq$  M'  $\wedge$  ( $\forall a \in M. \forall \varepsilon > 0. \exists \delta > 0. (\forall x. x \in M \wedge d a x < \delta \longrightarrow d' (f a)$ 
    (f x) <  $\varepsilon$ ))
    (is ?lhs = ?rhs)
  proof -
    interpret M': Metric_space M' d'
      by (simp add: assms)
    show ?thesis
    proof
      assume L: ?lhs
      show ?rhs
      proof (intro conjI strip)
        show f ' M  $\subseteq$  M'
          using L by (auto simp: continuous_map_def)
        fix a and  $\varepsilon :: real$ 
        assume a  $\in$  M and  $\varepsilon > 0$ 
        then have openin mtopology {x  $\in$  M. f x  $\in$  M'.mball (f a)  $\varepsilon$ } f a  $\in$  M'
          using L unfolding continuous_map_def by fastforce+
        then obtain  $\delta$  where  $\delta > 0$  mball a  $\delta \subseteq$  {x  $\in$  M. f x  $\in$  M'  $\wedge$  d' (f a) (f x)
        <  $\varepsilon$ }
          using <0 <  $\varepsilon$ > <a  $\in$  M> openin_mtopology by auto
        then show  $\exists \delta > 0. \forall x. x \in M \wedge d a x < \delta \longrightarrow d' (f a) (f x) < \varepsilon$ 
          using <a  $\in$  M> in_mball by blast
        qed
      next
        assume R: ?rhs
        show ?lhs
          unfolding continuous_map_def
        proof (intro conjI strip)
          fix U
          assume openin M'.mtopology U
          then show openin mtopology {x  $\in$  topspace mtopology. f x  $\in$  U}
            apply (simp add: continuous_map_def openin_mtopology M'.openin_mtopology
            subset_iff)
            by (metis R image_subset_iff)
          qed (use R in auto)
        qed
      qed
    end
  end

```

6.7.14 Completely metrizable spaces

These spaces are topologically complete

definition *completely_metrizable_space* **where**

```

completely_metrizable_space X  $\equiv$ 
 $\exists M d. Metric\_space M d \wedge Metric\_space.mcomplete M d \wedge X = Metric\_space.mtopology M d$ 

```

```

lemma empty_completely_metrizable_space:
  completely_metrizable_space trivial_topology
  unfolding completely_metrizable_space_def subtopology_eq_discrete_topology_empty
  [symmetric]
  by (metis Metric_space.mcomplete_empty_mspace discrete_metric.mtopology_discrete_metric
metric_M_dd)

lemma completely_metrizable_imp_metrizable_space:
  completely_metrizable_space X  $\implies$  metrizable_space X
  using completely_metrizable_space_def metrizable_space_def by auto

lemma (in Metric_space) completely_metrizable_space_mtopology:
  mcomplete  $\implies$  completely_metrizable_space mtopology
  using Metric_space_axioms completely_metrizable_space_def by blast

lemma completely_metrizable_space_discrete_topology:
  completely_metrizable_space (discrete_topology U)
  unfolding completely_metrizable_space_def
  by (metis discrete_metric.mcomplete_discrete_metric discrete_metric.mtopology_discrete_metric
metric_M_dd)

lemma completely_metrizable_space_euclidean:
  completely_metrizable_space (euclidean::'a::complete_space topology)
  using Met_TC.completely_metrizable_space_mtopology complete_UNIV by auto

lemma completely_metrizable_space_closedin:
  assumes X: completely_metrizable_space X and S: closedin X S
  shows completely_metrizable_space(subtopology X S)
proof –
  obtain M d where Metric_space M d and comp: Metric_space.mcomplete M d

      and Xeq: X = Metric_space.mtopology M d
      using assms completely_metrizable_space_def by blast
  then interpret Metric_space M d
      by blast
  show ?thesis
      unfolding completely_metrizable_space_def
  proof (intro conjI exI)
      show Metric_space S d
          using S Xeq closedin_subset subspace by force
      have sub: Submetric_axioms M S
          by (metis S Xeq closedin_metric Submetric_axioms_def)
      then show Metric_space.mcomplete S d
          using S Submetric.closedin_mcomplete_imp_mcomplete Submetric_def Xeq
comp by blast
      show subtopology X S = Metric_space.mtopology S d
          by (metis Metric_space_axioms Xeq sub Submetric.intro Submetric.mtopology_submetric)
  qed
qed

```

lemma *completely_metrizable_space_cbox*: *completely_metrizable_space* (top_of_set (cbox a b))
using *closed_closedin* *completely_metrizable_space_closedin* *completely_metrizable_space_euclidean*
by *blast*

lemma *homeomorphic_completely_metrizable_space_aux*:
assumes *homXY*: *X* *homeomorphic_space* *Y* **and** *X*: *completely_metrizable_space* *X*
shows *completely_metrizable_space* *Y*
proof –
obtain *f g* **where** *hmf*: *homeomorphic_map* *X* *Y* *f* **and** *hmg*: *homeomorphic_map* *Y* *X* *g*
and *fg*: $\bigwedge x. x \in \text{topspace } X \implies g(f\ x) = x \bigwedge y. y \in \text{topspace } Y \implies f(g\ y) = y$
and *fm*: $f \in \text{topspace } X \rightarrow \text{topspace } Y$ **and** *gm*: $g \in \text{topspace } Y \rightarrow \text{topspace } X$
using *homXY*
using *homeomorphic_space_unfold* **by** *blast*
obtain *M d* **where** *Md*: *Metric_space* *M* *d* *Metric_space.mcomplete* *M* *d* **and** *Xeq*: $X = \text{Metric_space.mtopology } M\ d$
using *X* **by** (*auto simp: completely_metrizable_space_def*)
then interpret *MX*: *Metric_space* *M* *d* **by** *metis*
define *D* **where** $D \equiv \lambda x\ y. d\ (g\ x)\ (g\ y)$
have *Metric_space* (topspace *Y*) *D*
proof
show $(D\ x\ y = 0) \longleftrightarrow (x = y)$ **if** $x \in \text{topspace } Y$ $y \in \text{topspace } Y$ **for** $x\ y$
unfolding *D_def*
by (*metis that MX.topspace_mtopology MX.zero Xeq fg gim Pi_iff*)
show $D\ x\ z \leq D\ x\ y + D\ y\ z$
if $x \in \text{topspace } Y$ $y \in \text{topspace } Y$ $z \in \text{topspace } Y$ **for** $x\ y\ z$
using *that MX.triangle Xeq gim* **by** (*auto simp: D_def*)
qed (*auto simp: D_def MX commute*)
then interpret *MY*: *Metric_space* (topspace *Y*) $\lambda x\ y. D\ x\ y$ **by** *metis*
show *?thesis*
unfolding *completely_metrizable_space_def*
proof (*intro exI conjI*)
show *Metric_space* (topspace *Y*) *D*
using *MY.Metric_space_axioms* **by** *blast*
have *gball*: $g\ ` \text{MY.mball } y\ r = \text{MX.mball } (g\ y)\ r$ **if** $y \in \text{topspace } Y$ **for** $y\ r$
using *that MX.topspace_mtopology Xeq gim hmg homeomorphic_imp_surjective_map*
unfolding *MX.mball_def MY.mball_def* **by** (*fastforce simp: D_def*)
have $\exists r > 0. \text{MY.mball } y\ r \subseteq S$ **if** *openin* *Y* *S* **and** $y \in S$ **for** *S*
proof –
have *openin* *X* (g`*S*)
using *hmg homeomorphic_map_openness_eq* **that** **by** *auto*
then obtain *r* **where** $r > 0$ $\text{MX.mball } (g\ y)\ r \subseteq g\ ` S$
using *MX.openin_mtopology Xeq* $\langle y \in S \rangle$ **by** *auto*
then show *?thesis*

```

      by (smt (verit, ccfv_SIG) MY.in_mball gball fg image_iff in_mono
openin_subset subsetI that(1))
    qed
  moreover have openin Y S
  if  $S \subseteq \text{topspace } Y$  and  $\bigwedge y. y \in S \implies \exists r > 0. MY.mball y r \subseteq S$  for S
  proof -
    have  $\bigwedge x. x \in g'S \implies \exists r > 0. MX.mball x r \subseteq g'S$ 
    by (smt (verit) gball imageE image_mono subset_iff that)
    then have openin X (g'S)
    using MX.openin_mtopology Xeq gim that(1) by auto
    then show ?thesis
    using hmg homeomorphic_map_openness_eq that(1) by blast
  qed
  ultimately show Yeq:  $Y = MY.mtopology$ 
  unfolding topology_eq MY.openin_mtopology by (metis openin_subset)

show MY.mcomplete
  unfolding MY.mcomplete_def
  proof (intro strip)
    fix  $\sigma$ 
    assume  $\sigma: MY.MCauchy \sigma$ 
    have  $MX.MCauchy (g \circ \sigma)$ 
    unfolding MX.MCauchy_def
  proof (intro conjI strip)
    show  $\text{range } (g \circ \sigma) \subseteq M$ 
    using MY.MCauchy_def Xeq  $\sigma$  gim by auto
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    then obtain N where  $\forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow D (\sigma n) (\sigma n') < \varepsilon$ 
    using MY.MCauchy_def  $\sigma$  by presburger
    then show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d ((g \circ \sigma) n) ((g \circ \sigma) n')$ 
    <  $\varepsilon$ 
    by (auto simp: o_def D_def)
  qed
  then obtain x where  $x: \text{limitin } MX.mtopology (g \circ \sigma) x$  sequentially  $x \in \text{topspace } X$ 
  using MX.limitin_mspace MX.topspace_mtopology Md Xeq unfolding
  MX.mcomplete_def
  by blast
  with x have  $\text{limitin } MY.mtopology (f \circ (g \circ \sigma)) (f x)$  sequentially
  by (metis Xeq Yeq continuous_map_limit hmf homeomorphic_imp_continuous_map)
  moreover have  $f \circ (g \circ \sigma) = \sigma$ 
  using  $\langle MY.MCauchy \sigma \rangle$  by (force simp add: fg MY.MCauchy_def subset_iff)
  ultimately have  $\text{limitin } MY.mtopology \sigma (f x)$  sequentially by simp
  then show  $\exists y. \text{limitin } MY.mtopology \sigma y$  sequentially
  by blast
  qed
  qed

```

qed

lemma *homeomorphic_completely_metrizable_space*:

X *homeomorphic_space* *Y*

\implies *completely_metrizable_space* *X* \longleftrightarrow *completely_metrizable_space* *Y*

by (*meson* *homeomorphic_completely_metrizable_space_aux* *homeomorphic_space_sym*)

lemma *completely_metrizable_space_retraction_map_image*:

assumes *r*: *retraction_map* *X* *Y* *r* **and** *X*: *completely_metrizable_space* *X*

shows *completely_metrizable_space* *Y*

proof –

obtain *s* **where** *s*: *retraction_maps* *X* *Y* *r* *s*

using *r* *retraction_map_def* **by** *blast*

then have *subtopology* *X* (*s* ‘ *topspace* *Y*) *homeomorphic_space* *Y*

using *retraction_maps_section_image2* **by** *blast*

then show *?thesis*

by (*metis* *X* *retract_of_space_imp_closedin* *retraction_maps_section_image1*

homeomorphic_completely_metrizable_space *completely_metrizable_space_closedin*

completely_metrizable_imp_metrizable_space *metrizable_imp_Hausdorff_space*

s)

qed

6.7.15 Product metric

For the nicest fit with the main Euclidean theories, we choose the Euclidean product, though other definitions of the product work.

definition *prod_dist* $\equiv \lambda d1\ d2\ (x,y)\ (x',y').\ \text{sqrt}(d1\ x\ x' \wedge 2 + d2\ y\ y' \wedge 2)$

locale *Metric_space12* = *M1*: *Metric_space* *M1* *d1* + *M2*: *Metric_space* *M2* *d2*
for *M1* *d1* *M2* *d2*

lemma (**in** *Metric_space12*) *prod_metric*: *Metric_space* (*M1* \times *M2*) (*prod_dist* *d1* *d2*)

proof

fix *x* *y* *z*

assume *xyz*: *x* \in *M1* \times *M2* *y* \in *M1* \times *M2* *z* \in *M1* \times *M2*

have $\text{sqrt}((d1\ x1\ z1)^2 + (d2\ x2\ z2)^2) \leq \text{sqrt}((d1\ x1\ y1)^2 + (d2\ x2\ y2)^2) + \text{sqrt}((d1\ y1\ z1)^2 + (d2\ y2\ z2)^2)$

(**is** $\text{sqrt}\ ?L \leq ?R$)

if *x* = (*x1*, *x2*) *y* = (*y1*, *y2*) *z* = (*z1*, *z2*)

for *x1* *x2* *y1* *y2* *z1* *z2*

proof –

have *tri*: *d1* *x1* *z1* \leq *d1* *x1* *y1* + *d1* *y1* *z1* *d2* *x2* *z2* \leq *d2* *x2* *y2* + *d2* *y2* *z2*

using *that* *xyz* *M1*.*triangle* [of *x1* *y1* *z1*] *M2*.*triangle* [of *x2* *y2* *z2*] **by** *auto*

show *?thesis*

proof (*rule* *real_le_lsqrt*)

have $?L \leq (d1\ x1\ y1 + d1\ y1\ z1)^2 + (d2\ x2\ y2 + d2\ y2\ z2)^2$

```

    using tri by (smt (verit) M1.nonneg M2.nonneg power_mono)
    also have ... ≤ ?R2
    by (metis real_sqrt_sum_squares_triangle_ineq sqrt_le_D)
    finally show ?L ≤ ?R2 .
  qed auto
  qed
  then show prod_dist d1 d2 x z ≤ prod_dist d1 d2 x y + prod_dist d1 d2 y z
    by (simp add: prod_dist_def case_prod_unfold)
  qed (auto simp: M1.commute M2.commute case_prod_unfold prod_dist_def)

sublocale Metric_space12 ⊆ Prod_metric: Metric_space M1 × M2 prod_dist d1
d2
  by (simp add: prod_metric)

```

For easy reference to theorems outside of the locale

```

lemma Metric_space12_mspace_mdistr:
  Metric_space12 (mspace m1) (mdistr m1) (mspace m2) (mdistr m2)
  by (simp add: Metric_space12_def Metric_space_mspace_mdistr)

```

```

definition prod_metric where
  prod_metric ≡ λm1 m2. metric (mspace m1 × mspace m2, prod_dist (mdistr m1)
(mdistr m2))

```

```

lemma submetric_prod_metric:
  submetric (prod_metric m1 m2) (S × T) = prod_metric (submetric m1 S)
(submetric m2 T)
  apply (simp add: prod_metric_def)
  by (simp add: submetric_def Metric_space_mspace_metric Metric_space_mdistr_metric
Metric_space12.prod_metric Metric_space12_def Metric_space_mspace_mdistr Times_Int_Times)

```

```

lemma mspace_prod_metric [simp]:
  mspace (prod_metric m1 m2) = mspace m1 × mspace m2
  by (simp add: prod_metric_def Metric_space_mspace_metric Metric_space12.prod_metric
Metric_space12_mspace_mdistr)

```

```

lemma mdistr_prod_metric [simp]:
  mdistr (prod_metric m1 m2) = prod_dist (mdistr m1) (mdistr m2)
  by (metis Metric_space_mdistr_metric Metric_space12.prod_metric Metric_space12_mspace_mdistr
prod_metric_def)

```

```

lemma prod_dist_dist [simp]: prod_dist dist dist = dist
  by (simp add: prod_dist_def dist_prod_def fun_eq_iff)

```

```

lemma prod_metric_euclidean [simp]:
  prod_metric euclidean_metric euclidean_metric = euclidean_metric
  by (simp add: prod_metric_def euclidean_metric_def)

```

```

context Metric_space12
begin

```


lemma *component_le_prod_metric:*

$d1\ x1\ x2 \leq prod_dist\ d1\ d2\ (x1,y1)\ (x2,y2)\ d2\ y1\ y2 \leq prod_dist\ d1\ d2\ (x1,y1)\ (x2,y2)$
by (*auto simp: prod_dist_def*)

lemma *prod_metric_le_components:*

$\llbracket x1 \in M1; y1 \in M1; x2 \in M2; y2 \in M2 \rrbracket$
 $\implies prod_dist\ d1\ d2\ (x1,x2)\ (y1,y2) \leq d1\ x1\ y1 + d2\ x2\ y2$
by (*auto simp: prod_dist_def sqrt_sum_squares_le_sum*)

lemma *mball_prod_metric_subset:*

$Prod_metric.mball\ (x,y)\ r \subseteq M1.mball\ x\ r \times M2.mball\ y\ r$
by *clarsimp (smt (verit, best) component_le_prod_metric)*

lemma *mcball_prod_metric_subset:*

$Prod_metric.mcball\ (x,y)\ r \subseteq M1.mcball\ x\ r \times M2.mcball\ y\ r$
by *clarsimp (smt (verit, best) component_le_prod_metric)*

lemma *mball_subset_prod_metric:*

$M1.mball\ x1\ r1 \times M2.mball\ x2\ r2 \subseteq Prod_metric.mball\ (x1,x2)\ (r1 + r2)$
using *prod_metric_le_components* **by** *force*

lemma *mcball_subset_prod_metric:*

$M1.mcball\ x1\ r1 \times M2.mcball\ x2\ r2 \subseteq Prod_metric.mcball\ (x1,x2)\ (r1 + r2)$
using *prod_metric_le_components* **by** *force*

lemma *mtopology_prod_metric:*

$Prod_metric.mtopology = prod_topology\ M1.mtopology\ M2.mtopology$
unfolding *prod_topology_def*

proof (*rule topology_base_unique [symmetric]*)

fix *U*

assume $U \in \{S \times T \mid S\ T.\ openin\ M1.mtopology\ S \wedge openin\ M2.mtopology\ T\}$

then obtain $S\ T$ **where** $Ueq: U = S \times T$

and $S: openin\ M1.mtopology\ S$ **and** $T: openin\ M2.mtopology\ T$

by *auto*

have $S \subseteq M1$

using $M1.openin_mtopology\ S$ **by** *auto*

have $T \subseteq M2$

using $M2.openin_mtopology\ T$ **by** *auto*

show $openin\ Prod_metric.mtopology\ U$

unfolding $Prod_metric.openin_mtopology$

proof (*intro conjI strip*)

show $U \subseteq M1 \times M2$

using Ueq **by** (*simp add: Sigma_mono* $\langle S \subseteq M1 \rangle \langle T \subseteq M2 \rangle$)

fix z

assume $z \in U$

then obtain $x1\ x2$ **where** $x1 \in S\ x2 \in T$ **and** $zeq: z = (x1,x2)$

using Ueq **by** *blast*

```

obtain  $r1$  where  $r1 > 0$  and  $r1: M1.mball\ x1\ r1 \subseteq S$ 
  by (meson  $M1.openin\_mtopology\ \langle openin\ M1.mtopology\ S \rangle\ \langle x1 \in S \rangle$ )
obtain  $r2$  where  $r2 > 0$  and  $r2: M2.mball\ x2\ r2 \subseteq T$ 
  by (meson  $M2.openin\_mtopology\ \langle openin\ M2.mtopology\ T \rangle\ \langle x2 \in T \rangle$ )
have  $Prod\_metric.mball\ (x1, x2)\ (min\ r1\ r2) \subseteq U$ 
proof (rule order_trans [OF mball_prod_metric_subset])
  show  $M1.mball\ x1\ (min\ r1\ r2) \times M2.mball\ x2\ (min\ r1\ r2) \subseteq U$ 
  using Ueq  $r1\ r2$  by force
qed
then show  $\exists r > 0. Prod\_metric.mball\ z\ r \subseteq U$ 
  by (smt (verit, del_insts) zeq  $\langle 0 < r1 \rangle\ \langle 0 < r2 \rangle$ )
qed
next
fix  $U\ z$ 
assume openin  $Prod\_metric.mtopology\ U$  and  $z \in U$ 
then have  $U \subseteq M1 \times M2$ 
  by (simp add: Prod_metric.openin_mtopology)
then obtain  $x\ y$  where  $x \in M1\ y \in M2$  and zeq:  $z = (x, y)$ 
  using  $\langle z \in U \rangle$  by blast
obtain  $r$  where  $r > 0$  and  $r: Prod\_metric.mball\ (x, y)\ r \subseteq U$ 
  by (metis Prod_metric.openin_mtopology  $\langle openin\ Prod\_metric.mtopology\ U \rangle$ 
 $\langle z \in U \rangle$  zeq)
define  $B1$  where  $B1 \equiv M1.mball\ x\ (r/2)$ 
define  $B2$  where  $B2 \equiv M2.mball\ y\ (r/2)$ 
have openin  $M1.mtopology\ B1$  openin  $M2.mtopology\ B2$ 
  by (simp_all add: B1_def B2_def)
moreover have  $(x, y) \in B1 \times B2$ 
  using  $\langle r > 0 \rangle$  by (simp add:  $\langle x \in M1 \rangle\ \langle y \in M2 \rangle$  B1_def B2_def)
moreover have  $B1 \times B2 \subseteq U$ 
  using  $r$  prod_metric_le_components by (force simp add: B1_def B2_def)
ultimately show  $\exists B. B \in \{S \times T \mid S\ T.\ openin\ M1.mtopology\ S \wedge openin\ M2.mtopology\ T\} \wedge z \in B \wedge B \subseteq U$ 
  by (auto simp: zeq)
qed

```

lemma *MCauchy_prod_metric*:

```

 $Prod\_metric.MCauchy\ \sigma \longleftrightarrow M1.MCauchy\ (fst \circ \sigma) \wedge M2.MCauchy\ (snd \circ \sigma)$ 
(is ?lhs  $\longleftrightarrow$  ?rhs)

```

proof *safe*

assume $L: ?lhs$

then **have** $range\ \sigma \subseteq M1 \times M2$

using *Prod_metric.MCauchy_def* **by** *blast*

then **have** $1: range\ (fst \circ \sigma) \subseteq M1$ **and** $2: range\ (snd \circ \sigma) \subseteq M2$

by *auto*

have $N1: \exists N. \forall n \geq N. \forall n' \geq N. d1\ (fst\ (\sigma\ n))\ (fst\ (\sigma\ n')) < \varepsilon$

and $N2: \exists N. \forall n \geq N. \forall n' \geq N. d2\ (snd\ (\sigma\ n))\ (snd\ (\sigma\ n')) < \varepsilon$ **if** $\varepsilon > 0$ **for** ε

:: *real*

using *that* L **unfolding** *Prod_metric.MCauchy_def*

by (*smt* (*verit*, *del_insts*) *add.commute* *add_less_imp_less_left* *add_right_mono*)

```

      component_le_prod_metric prod.collapse)+
show M1.MCauchy (fst ∘ σ)
  using 1 N1 M1.MCauchy_def by auto
have ∃ N. ∀ n ≥ N. ∀ n' ≥ N. d2 (snd (σ n)) (snd (σ n')) < ε if ε > 0 for ε :: real
  using that L unfolding Prod_metric.MCauchy_def
  by (smt (verit, del_insts) add.commute add_less_imp_less_left add_right_mono

      component_le_prod_metric prod.collapse)
show M2.MCauchy (snd ∘ σ)
  using 2 N2 M2.MCauchy_def by auto
next
assume M1: M1.MCauchy (fst ∘ σ) and M2: M2.MCauchy (snd ∘ σ)
then have subM12: range (fst ∘ σ) ⊆ M1 range (snd ∘ σ) ⊆ M2
  using M1.MCauchy_def M2.MCauchy_def by blast+
show ?lhs
  unfolding Prod_metric.MCauchy_def
  proof (intro conjI strip)
    show range σ ⊆ M1 × M2
      using subM12 by (smt (verit, best) SigmaI image_subset_iff o_apply prod.collapse)

    fix ε :: real
    assume ε > 0
    obtain N1 where N1: ∧ n n'. N1 ≤ n ⇒ N1 ≤ n' ⇒ d1 ((fst ∘ σ) n) ((fst
    ∘ σ) n') < ε/2
      by (meson M1.MCauchy_def ‹0 < ε› M1 zero_less_divide_iff zero_less_numeral)
    obtain N2 where N2: ∧ n n'. N2 ≤ n ⇒ N2 ≤ n' ⇒ d2 ((snd ∘ σ) n)
    ((snd ∘ σ) n') < ε/2
      by (meson M2.MCauchy_def ‹0 < ε› M2 zero_less_divide_iff zero_less_numeral)
    have prod_dist d1 d2 (σ n) (σ n') < ε
      if N1 ≤ n and N2 ≤ n and N1 ≤ n' and N2 ≤ n' for n n'
    proof -
      obtain a b a' b' where σ: σ n = (a,b) σ n' = (a',b')
        by fastforce+
      have prod_dist d1 d2 (a,b) (a',b') ≤ d1 a a' + d2 b b'
        by (metis ‹range σ ⊆ M1 × M2› σ mem_Sigma_iff prod_metric_le_components
        range_subsetD)
      also have ... < ε/2 + ε/2
        using N1 N2 σ that by fastforce
      finally show ?thesis
        by (simp add: σ)
    qed
    then show ∃ N. ∀ n n'. N ≤ n → N ≤ n' → prod_dist d1 d2 (σ n) (σ n')
    < ε
      by (metis order.trans linorder_le_cases)
    qed
  qed

```

```

lemma mcomplete_prod_metric:
  Prod_metric.mcomplete  $\longleftrightarrow$   $M1 = \{\}$   $\vee$   $M2 = \{\}$   $\vee$   $M1.mcomplete \wedge M2.mcomplete$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof (cases  $M1 = \{\}$   $\vee$   $M2 = \{\}$ )
  case False
  then obtain x y where  $x \in M1$   $y \in M2$ 
  by blast
  have  $M1.mcomplete \wedge M2.mcomplete \implies Prod\_metric.mcomplete$ 
  by (simp add: Prod_metric.mcomplete_def M1.mcomplete_def M2.mcomplete_def

      mtopology_prod_metric MCauchy_prod_metric limitin_pairwise)
  moreover
  { assume L: Prod_metric.mcomplete
    have  $M1.mcomplete$ 
    unfolding M1.mcomplete_def
    proof (intro strip)
      fix  $\sigma$ 
      assume  $M1.MCauchy \sigma$ 
      then have  $Prod\_metric.MCauchy (\lambda n. (\sigma n, y))$ 
      using  $\langle y \in M2 \rangle$  by (simp add: M1.MCauchy_def M2.MCauchy_def
MCauchy_prod_metric)
      then obtain z where limitin  $Prod\_metric.mtopology (\lambda n. (\sigma n, y))$  z se-
quentially
      using L Prod_metric.mcomplete_def by blast
      then show  $\exists x. limitin M1.mtopology \sigma x$  sequentially
      by (auto simp: Prod_metric.mcomplete_def M1.mcomplete_def
mtopology_prod_metric limitin_pairwise o_def)
    }
  qed
}
moreover
{ assume L: Prod_metric.mcomplete
  have  $M2.mcomplete$ 
  unfolding M2.mcomplete_def
  proof (intro strip)
    fix  $\sigma$ 
    assume  $M2.MCauchy \sigma$ 
    then have  $Prod\_metric.MCauchy (\lambda n. (x, \sigma n))$ 
    using  $\langle x \in M1 \rangle$  by (simp add: M2.MCauchy_def M1.MCauchy_def
MCauchy_prod_metric)
    then obtain z where limitin  $Prod\_metric.mtopology (\lambda n. (x, \sigma n))$  z se-
quentially
    using L Prod_metric.mcomplete_def by blast
    then show  $\exists x. limitin M2.mtopology \sigma x$  sequentially
    by (auto simp: Prod_metric.mcomplete_def M2.mcomplete_def
mtopology_prod_metric limitin_pairwise o_def)
  }
qed
}
ultimately show ?thesis
using False by blast

```

qed *auto*

lemma *mbounded_prod_metric*:

$Prod_metric.mbounded\ U \longleftrightarrow M1.mbounded\ (fst\ 'U) \wedge M2.mbounded\ (snd\ 'U)$

proof *–*

have $(\exists B. U \subseteq Prod_metric.mcball\ (x,y)\ B) \longleftrightarrow ((\exists B. (fst\ 'U) \subseteq M1.mcball\ x\ B) \wedge (\exists B. (snd\ 'U) \subseteq M2.mcball\ y\ B))$
(is *?lhs* \longleftrightarrow *?rhs*)

for *x y*

proof *safe*

fix *B*

assume $U \subseteq Prod_metric.mcball\ (x, y)\ B$

then have $(fst\ 'U) \subseteq M1.mcball\ x\ B \wedge (snd\ 'U) \subseteq M2.mcball\ y\ B$

using *mcball_prod_metric_subset* **by** *fastforce+*

then show $\exists B. (fst\ 'U) \subseteq M1.mcball\ x\ B \wedge \exists B. (snd\ 'U) \subseteq M2.mcball\ y\ B$

by *auto*

next

fix *B1 B2*

assume $(fst\ 'U) \subseteq M1.mcball\ x\ B1 \wedge (snd\ 'U) \subseteq M2.mcball\ y\ B2$

then have $fst\ 'U \times snd\ 'U \subseteq M1.mcball\ x\ B1 \times M2.mcball\ y\ B2$

by *blast*

also have $\dots \subseteq Prod_metric.mcball\ (x, y)\ (B1+B2)$

by *(intro mcball_subset_prod_metric)*

finally show $\exists B. U \subseteq Prod_metric.mcball\ (x, y)\ B$

by *(metis subsetD subsetI subset_fst_snd)*

qed

then show *?thesis*

by *(simp add: M1.mbounded_def M2.mbounded_def Prod_metric.mbounded_def)*

qed

lemma *mbounded_Times*:

$Prod_metric.mbounded\ (S \times T) \longleftrightarrow S = \{\} \vee T = \{\} \vee M1.mbounded\ S \wedge M2.mbounded\ T$

by *(auto simp: mbounded_prod_metric)*

lemma *mtotally_bounded_Times*:

$Prod_metric.mtotally_bounded\ (S \times T) \longleftrightarrow$

$S = \{\} \vee T = \{\} \vee M1.mtotally_bounded\ S \wedge M2.mtotally_bounded\ T$

(is *?lhs* \longleftrightarrow *_*)

proof *(cases S = {} \vee T = {})*

case *False*

then obtain *x y* **where** $x \in S\ y \in T$

by *auto*

have $M1.mtotally_bounded\ S$ **if** *L*: *?lhs*

unfolding *M1.mtotally_bounded_sequentially*

proof *(intro conjI strip)*

show $S \subseteq M1$

```

    using Prod_metric.mtotally_bounded_imp_subset ⟨y ∈ T⟩ that by blast
    fix σ :: nat ⇒ 'a
    assume range σ ⊆ S
    with L obtain r where strict_mono r Prod_metric.MCauchy ((λn. (σ n,y))
    ◦ r)
      unfolding Prod_metric.mtotally_bounded_sequentially
      by (smt (verit) SigmaI ⟨y ∈ T⟩ image_subset_iff)
    then have M1.MCauchy (fst ◦ (λn. (σ n,y)) ◦ r)
      by (simp add: MCauchy_prod_metric o_def)
    with ⟨strict_mono r⟩ show ∃ r. strict_mono r ∧ M1.MCauchy (σ ◦ r)
      by (auto simp add: o_def)
  qed
  moreover
  have M2.mtotally_bounded T if L: ?lhs
    unfolding M2.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show T ⊆ M2
      using Prod_metric.mtotally_bounded_imp_subset ⟨x ∈ S⟩ that by blast
      fix σ :: nat ⇒ 'b
      assume range σ ⊆ T
      with L obtain r where strict_mono r Prod_metric.MCauchy ((λn. (x,σ n))
      ◦ r)
        unfolding Prod_metric.mtotally_bounded_sequentially
        by (smt (verit) SigmaI ⟨x ∈ S⟩ image_subset_iff)
      then have M2.MCauchy (snd ◦ (λn. (x,σ n)) ◦ r)
        by (simp add: MCauchy_prod_metric o_def)
      with ⟨strict_mono r⟩ show ∃ r. strict_mono r ∧ M2.MCauchy (σ ◦ r)
        by (auto simp add: o_def)
    qed
  moreover have ?lhs if 1: M1.mtotally_bounded S and 2: M2.mtotally_bounded
  T
    unfolding Prod_metric.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show S × T ⊆ M1 × M2
      using that
      by (auto simp: M1.mtotally_bounded_sequentially M2.mtotally_bounded_sequentially)
    fix σ :: nat ⇒ 'a × 'b
    assume σ: range σ ⊆ S × T
    with 1 obtain r1 where r1: strict_mono r1 M1.MCauchy (fst ◦ σ ◦ r1)
      apply (clarsimp simp: M1.mtotally_bounded_sequentially image_subset_iff)
      by (metis SigmaE comp_eq_dest_lhs fst_conv)
    from σ 2 obtain r2 where r2: strict_mono r2 M2.MCauchy (snd ◦ σ ◦ r1 ◦
    r2)
      apply (clarsimp simp: M2.mtotally_bounded_sequentially image_subset_iff)
      by (smt (verit, best) comp_apply mem_Sigma_iff prod.collapse)
    then have M1.MCauchy (fst ◦ σ ◦ r1 ◦ r2)
      by (simp add: M1.MCauchy_subsequence r1)
    with r2 have Prod_metric.MCauchy (σ ◦ (r1 ◦ r2))
      by (simp add: MCauchy_prod_metric o_def)

```

```

    then show  $\exists r. \text{strict\_mono } r \wedge \text{Prod\_metric.MCauchy } (\sigma \circ r)$ 
      using r1 r2 strict_mono_o by blast
  qed
  ultimately show ?thesis
    using False by blast
qed auto

lemma mtotally_bounded_prod_metric:
  Prod_metric.mtotally_bounded  $U \longleftrightarrow$ 
   $M1.mtotally_bounded (\text{fst } 'U) \wedge M2.mtotally_bounded (\text{snd } 'U)$  (is ?lhs  $\longleftrightarrow$ 
?rhs)
proof
  assume L: ?lhs
  then have  $U \subseteq M1 \times M2$ 
  and *:  $\bigwedge \sigma. \text{range } \sigma \subseteq U \implies \exists r :: \text{nat} \Rightarrow \text{nat}. \text{strict\_mono } r \wedge \text{Prod\_metric.MCauchy}$ 
  ( $\sigma \circ r$ )
  by (simp_all add: Prod_metric.mtotally_bounded_sequentially)
  show ?rhs
  unfolding M1.mtotally_bounded_sequentially M2.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show  $\text{fst } 'U \subseteq M1$   $\text{snd } 'U \subseteq M2$ 
      using  $\langle U \subseteq M1 \times M2 \rangle$  by auto
  next
    fix  $\sigma :: \text{nat} \Rightarrow 'a$ 
    assume  $\text{range } \sigma \subseteq \text{fst } 'U$ 
    then obtain  $\zeta$  where  $\zeta: \bigwedge n. \sigma n = \text{fst } (\zeta n) \wedge \zeta n \in U$ 
      unfolding image_subset_iff image_iff by (meson UNIV_I)
    then obtain  $r$  where  $\text{strict\_mono } r \wedge \text{Prod\_metric.MCauchy } (\zeta \circ r)$ 
      by (metis * image_subset_iff)
    with  $\zeta$  show  $\exists r. \text{strict\_mono } r \wedge M1.MCauchy (\sigma \circ r)$ 
      by (auto simp: MCauchy_prod_metric o_def)
  next
    fix  $\sigma :: \text{nat} \Rightarrow 'b$ 
    assume  $\text{range } \sigma \subseteq \text{snd } 'U$ 
    then obtain  $\zeta$  where  $\zeta: \bigwedge n. \sigma n = \text{snd } (\zeta n) \wedge \zeta n \in U$ 
      unfolding image_subset_iff image_iff by (meson UNIV_I)
    then obtain  $r$  where  $\text{strict\_mono } r \wedge \text{Prod\_metric.MCauchy } (\zeta \circ r)$ 
      by (metis * image_subset_iff)
    with  $\zeta$  show  $\exists r. \text{strict\_mono } r \wedge M2.MCauchy (\sigma \circ r)$ 
      by (auto simp: MCauchy_prod_metric o_def)
  qed
next
  assume ?rhs
  then have  $\text{Prod\_metric.mtotally_bounded } ((\text{fst } 'U) \times (\text{snd } 'U))$ 
    by (simp add: mtotally_bounded_Times)
  then show ?lhs
    by (metis Prod_metric.mtotally_bounded_subset subset_fst_snd)
qed

```

end

lemma *metrizable_space_prod_topology*:
 $\text{metrizable_space } (\text{prod_topology } X Y) \longleftrightarrow$
 $(\text{prod_topology } X Y) = \text{trivial_topology} \vee \text{metrizable_space } X \wedge \text{metrizable_space } Y$
(is ?lhs \longleftrightarrow ?rhs)
proof (*cases* ($\text{prod_topology } X Y = \text{trivial_topology}$))
case *False*
then obtain $x y$ **where** $x \in \text{topspace } X$ $y \in \text{topspace } Y$
by *fastforce*
show *?thesis*
proof
show $?rhs \implies ?lhs$
unfolding *metrizable_space_def*
using *Metric_space12.mtopology_prod_metric*
by (*metis False Metric_space12.prod_metric Metric_space12_def*)
next
assume $L: ?lhs$
have $\text{metrizable_space } (\text{subtopology } (\text{prod_topology } X Y) (\text{topspace } X \times \{y\}))$
 $\text{metrizable_space } (\text{subtopology } (\text{prod_topology } X Y) (\{x\} \times \text{topspace } Y))$
using L *metrizable_space_subtopology* **by** *auto*
moreover
have $(\text{subtopology } (\text{prod_topology } X Y) (\text{topspace } X \times \{y\}))$ *homeomorphic_space* X
by (*metis* $\langle y \in \text{topspace } Y \rangle$ *homeomorphic_space_prod_topology_sing1* *homeomorphic_space_sym prod_topology_subtopology(2)*)
moreover
have $(\text{subtopology } (\text{prod_topology } X Y) (\{x\} \times \text{topspace } Y))$ *homeomorphic_space* Y
by (*metis* $\langle x \in \text{topspace } X \rangle$ *homeomorphic_space_prod_topology_sing2* *homeomorphic_space_sym prod_topology_subtopology(1)*)
ultimately show $?rhs$
by (*simp add: homeomorphic_metrizable_space*)
qed
qed *auto*

lemma *completely_metrizable_space_prod_topology*:
 $\text{completely_metrizable_space } (\text{prod_topology } X Y) \longleftrightarrow$
 $(\text{prod_topology } X Y) = \text{trivial_topology} \vee$
 $\text{completely_metrizable_space } X \wedge \text{completely_metrizable_space } Y$
(is ?lhs \longleftrightarrow ?rhs)
proof (*cases* ($\text{prod_topology } X Y = \text{trivial_topology}$))
case *False*
then obtain $x y$ **where** $x \in \text{topspace } X$ $y \in \text{topspace } Y$
by *fastforce*
show *?thesis*


```

proof
  show ?rhs  $\implies$  ?lhs
    unfolding completely_metrizable_space_def
    by (metis False Metric_space12.mtopology_prod_metric Metric_space12.mcomplete_prod_metric
      Metric_space12.prod_metric Metric_space12_def)
next
  assume L: ?lhs
  then have Hausdorff_space (prod_topology X Y)
    by (simp add: completely_metrizable_imp_metrizable_space metrizable_imp_Hausdorff_space)
  then have H: Hausdorff_space X  $\wedge$  Hausdorff_space Y
    using False Hausdorff_space_prod_topology by blast
  then have closedin (prod_topology X Y) (topspace X  $\times$  {y})  $\wedge$  closedin
(prod_topology X Y) ({x}  $\times$  topspace Y)
    using  $\langle x \in \text{topspace } X \rangle \langle y \in \text{topspace } Y \rangle$ 
    by (auto simp: closedin_Hausdorff_sing_eq closedin_prod_Times_iff)
  with L have completely_metrizable_space(subtopology (prod_topology X Y)
(topspace X  $\times$  {y}))
     $\wedge$  completely_metrizable_space(subtopology (prod_topology X Y) ({x}
 $\times$  topspace Y))
    by (simp add: completely_metrizable_space_closedin)
  moreover
  have (subtopology (prod_topology X Y) (topspace X  $\times$  {y})) homeomor-
phic_space X
    by (metis  $\langle y \in \text{topspace } Y \rangle$  homeomorphic_space_prod_topology_sing1 home-
omorphic_space_sym prod_topology_subtopology(2))
  moreover
  have (subtopology (prod_topology X Y) ({x}  $\times$  topspace Y)) homeomor-
phic_space Y
    by (metis  $\langle x \in \text{topspace } X \rangle$  homeomorphic_space_prod_topology_sing2 home-
omorphic_space_sym prod_topology_subtopology(1))
  ultimately show ?rhs
    by (simp add: homeomorphic_completely_metrizable_space)
qed
next
  case True then show ?thesis
    using empty_completely_metrizable_space by auto
qed

```

6.7.16 More sequential characterizations in a metric space

context Metric_space

begin

definition decreasing_dist :: (nat \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool

where decreasing_dist σ x \equiv (\forall m n. m < n \longrightarrow d (σ n) x < d (σ m) x)

lemma decreasing_dist_imp_inj: decreasing_dist σ a \implies inj σ

by (metis decreasing_dist_def dual_order.irrefl linorder_inj_onI')

lemma *eventually_atin_within_metric*:

eventually P (atin_within mtopology a S) \longleftrightarrow

($a \in M \longrightarrow (\exists \delta > 0. \forall x. x \in M \wedge x \in S \wedge 0 < d x a \wedge d x a < \delta \longrightarrow P x)$)

(is ?lhs=?rhs)

proof

assume *?lhs then show ?rhs*

unfolding *eventually_atin_within openin_mtopology subset_iff*

by *(metis commute_in_mball mdist_zero order_less_irrefl topspace_mtopology)*

next

assume *R: ?rhs*

show *?lhs*

proof *(cases a $\in M$)*

case *True*

then obtain δ **where** $\delta > 0$ **and** $\delta: \bigwedge x. \llbracket x \in M; x \in S; 0 < d x a; d x a < \delta \rrbracket \Longrightarrow P x$

using *R by blast*

then have *openin_mtopology (mball a δ) \wedge ($\forall x \in \text{mball } a \ \delta. x \in S \wedge x \neq a \longrightarrow P x$)*

by *(simp add: commute_openin_mball)*

then show *?thesis*

by *(metis True $\langle 0 < \delta \rangle$ centre_in_mball_iff eventually_atin_within)*

next

case *False*

with *R show ?thesis*

by *(simp add: eventually_atin_within)*

qed

qed

lemma *eventually_atin_within_A*:

assumes

($\bigwedge \sigma. \llbracket \text{range } \sigma \subseteq (S \cap M) - \{a\}; \text{decreasing_dist } \sigma \ a;$

inj σ ; limitin_mtopology $\sigma \ a$ sequentially \rrbracket

\Longrightarrow eventually ($\lambda n. P (\sigma n)$) sequentially)

shows *eventually P (atin_within mtopology a S)*

proof $-$

have *False if SP: $\bigwedge \delta. \delta > 0 \Longrightarrow \exists x \in M - \{a\}. d x a < \delta \wedge x \in S \wedge \neg P x$ and $a \in M$*

proof $-$

define Φ **where** $\Phi \equiv \lambda n x. x \in M - \{a\} \wedge d x a < \text{inverse } (\text{Suc } n) \wedge x \in S \wedge \neg P x$

obtain σ **where** $\sigma: \bigwedge n. \Phi n (\sigma n)$ **and** *dless: $\bigwedge n. d (\sigma (\text{Suc } n)) a < d (\sigma n) a$*

proof $-$

obtain $x0$ **where** $x0: \Phi 0 x0$

using *SP [OF zero_less_one] by (force simp: Φ _def)*

have $\exists y. \Phi (\text{Suc } n) y \wedge d y a < d x a$ **if** $\Phi n x$ **for** $n x$

using *SP [of min (inverse (Suc (Suc n))) (d x a)] $\langle a \in M \rangle$ that*

by *(auto simp: Φ _def)*

then obtain f **where** $f: \bigwedge n x. \Phi n x \Longrightarrow \Phi (\text{Suc } n) (f n x) \wedge d (f n x) a <$

```

d x a
  by metis
  show thesis
  proof
    show  $\Phi$  n (rec_nat x0 f n) for n
      by (induction n) (auto simp: x0 dest: f)
    with f show d (rec_nat x0 f (Suc n)) a < d (rec_nat x0 f n) a for n
      by auto
    qed
  qed
  have 1: range  $\sigma \subseteq (S \cap M) - \{a\}$ 
    using  $\sigma$  by (auto simp:  $\Phi$ _def)
  have d ( $\sigma$ (Suc (m+n))) a < d ( $\sigma$  n) a for m n
    by (induction m) (auto intro: order_less_trans dless)
  then have 2: decreasing_dist  $\sigma$  a
    unfolding decreasing_dist_def by (metis add.commute less_imp_Suc_add)
  have  $\forall_F$  xa in sequentially. d ( $\sigma$  xa) a <  $\varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    obtain N where inverse (Suc N) <  $\varepsilon$ 
      using  $\langle \varepsilon > 0 \rangle$  reals_Archimedean by blast
    with  $\sigma$  2 show ?thesis
      unfolding decreasing_dist_def by (smt (verit, best)  $\Phi$ _def eventually_at_top_dense)
    qed
  then have 4: limitin_mtopology  $\sigma$  a sequentially
    using  $\sigma$   $\langle a \in M \rangle$  by (simp add:  $\Phi$ _def limitin_metric)
  show False
    using 2 assms [OF 1 _ decreasing_dist_imp_inj 4]  $\sigma$  by (force simp:  $\Phi$ _def)
  qed
  then show ?thesis
    by (fastforce simp: eventually_atin_within_metric)
  qed

```

lemma eventually_atin_within_B:

```

assumes ev: eventually P (atin_within_mtopology a S)
  and ran: range  $\sigma \subseteq (S \cap M) - \{a\}$ 
  and lim: limitin_mtopology  $\sigma$  a sequentially
shows eventually ( $\lambda n. P$  ( $\sigma$  n)) sequentially
proof -
  have a  $\in M$ 
    using lim limitin_mspace by auto
  with ev obtain  $\delta$  where  $0 < \delta$ 
    and  $\delta: \bigwedge \sigma. \llbracket \sigma \in M; \sigma \in S; 0 < d \sigma a; d \sigma a < \delta \rrbracket \implies P \sigma$ 
    by (auto simp: eventually_atin_within_metric)
  then have *:  $\bigwedge n. \sigma$  n  $\in M \wedge d$  ( $\sigma$  n) a <  $\delta \implies P$  ( $\sigma$  n)
    using  $\langle a \in M \rangle$  ran by auto
  have  $\forall_F$  n in sequentially.  $\sigma$  n  $\in M \wedge d$  ( $\sigma$  n) a <  $\delta$ 
    using lim  $\langle 0 < \delta \rangle$  by (auto simp: limitin_metric)
  then show ?thesis

```

by (*simp add: * eventually_mono*)
qed

lemma *eventually_atin_within_sequentially*:
 $eventually\ P\ (atin_within\ mtopology\ a\ S) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq (S \cap M) - \{a\} \wedge$
 $limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_within_sequentially_inj*:
 $eventually\ P\ (atin_within\ mtopology\ a\ S) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq (S \cap M) - \{a\} \wedge inj\ \sigma \wedge$
 $limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_within_sequentially_decreasing*:
 $eventually\ P\ (atin_within\ mtopology\ a\ S) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq (S \cap M) - \{a\} \wedge decreasing_dist\ \sigma\ a \wedge$
 $limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_sequentially*:
 $eventually\ P\ (atin\ mtopology\ a) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq M - \{a\} \wedge limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 using *eventually_atin_within_sequentially* [**where** $S=UNIV$] by *simp*

lemma *eventually_atin_sequentially_inj*:
 $eventually\ P\ (atin\ mtopology\ a) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq M - \{a\} \wedge inj\ \sigma \wedge$
 $limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 using *eventually_atin_within_sequentially_inj* [**where** $S=UNIV$] by *simp*

lemma *eventually_atin_sequentially_decreasing*:
 $eventually\ P\ (atin\ mtopology\ a) \longleftrightarrow$
 $(\forall \sigma. range\ \sigma \subseteq M - \{a\} \wedge decreasing_dist\ \sigma\ a \wedge$
 $limitin\ mtopology\ \sigma\ a\ sequentially$
 $\longrightarrow eventually\ (\lambda n. P(\sigma\ n))\ sequentially)$
 using *eventually_atin_within_sequentially_decreasing* [**where** $S=UNIV$] by
simp

end

context *Metric_space12*

begin

lemma *limit_atin_sequentially_within*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin_within } M1.\text{mtopology } a \ S) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq S \cap M1 - \{a\} \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{by (auto simp: } M1.\text{eventually_atin_within_sequentially } \text{limitin_def)} \end{aligned}$$

lemma *limit_atin_sequentially_within_inj*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin_within } M1.\text{mtopology } a \ S) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq S \cap M1 - \{a\} \wedge \text{inj } \sigma \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{by (auto simp: } M1.\text{eventually_atin_within_sequentially_inj } \text{limitin_def)} \end{aligned}$$

lemma *limit_atin_sequentially_within_decreasing*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin_within } M1.\text{mtopology } a \ S) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq S \cap M1 - \{a\} \wedge M1.\text{decreasing_dist } \sigma \ a \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{by (auto simp: } M1.\text{eventually_atin_within_sequentially_decreasing } \text{limitin_def)} \end{aligned}$$

lemma *limit_atin_sequentially*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin } M1.\text{mtopology } a) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq M1 - \{a\} \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{using } \text{limit_atin_sequentially_within } [\text{where } S=UNIV] \text{ by simp} \end{aligned}$$

lemma *limit_atin_sequentially_inj*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin } M1.\text{mtopology } a) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq M1 - \{a\} \wedge \text{inj } \sigma \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{using } \text{limit_atin_sequentially_within_inj } [\text{where } S=UNIV] \text{ by simp} \end{aligned}$$

lemma *limit_atin_sequentially_decreasing*:

$$\begin{aligned} & \text{limitin } M2.\text{mtopology } f \ l \ (\text{atin } M1.\text{mtopology } a) \longleftrightarrow \\ & \quad l \in M2 \wedge \\ & \quad (\forall \sigma. \text{range } \sigma \subseteq M1 - \{a\} \wedge M1.\text{decreasing_dist } \sigma \ a \wedge \\ & \quad \quad \text{limitin } M1.\text{mtopology } \sigma \ a \ \text{sequentially} \\ & \quad \quad \longrightarrow \text{limitin } M2.\text{mtopology } (f \circ \sigma) \ l \ \text{sequentially}) \\ & \text{using } \text{limit_atin_sequentially_within_decreasing } [\text{where } S=UNIV] \text{ by simp} \end{aligned}$$

end

An experiment: same result as within the locale, but using metric space variables

lemma *limit_atin_sequentially_within*:

limitin (mtopology_of m2) f l (atin_within (mtopology_of m1) a S) \longleftrightarrow
l \in mspace m2 \wedge

*($\forall \sigma$. range $\sigma \subseteq S \cap$ mspace m1 $- \{a\} \wedge$
limitin (mtopology_of m1) σ a sequentially
 *\longrightarrow limitin (mtopology_of m2) (f \circ σ) l sequentially)**

using *Metric_space12.limit_atin_sequentially_within [OF Metric_space12_mspace_mdist]*
by (*metis mtopology_of_def*)

context *Metric_space*

begin

lemma *atin_within_imp_M*:

atin_within mtopology x S \neq bot \implies x \in M

by (*metis derived_set_of_trivial_limit_in_derived_set_of_topospace_mtopology*)

lemma *atin_within_sequentially_sequence*:

assumes *atin_within mtopology x S \neq bot*

obtains σ **where** *range $\sigma \subseteq S \cap M - \{x\}$*

decreasing_dist σ x inj σ limitin mtopology σ x sequentially

by (*metis eventually_atin_within_A eventually_False assms*)

lemma *derived_set_of_sequentially*:

mtopology derived_set_of S =

{x \in M. $\exists \sigma$. range $\sigma \subseteq S \cap M - \{x\} \wedge$ limitin mtopology σ x sequentially}

proof –

have *False*

if *range $\sigma \subseteq S \cap M - \{x\}$*

and *limitin mtopology σ x sequentially*

and *atin_within mtopology x S = bot*

for *x σ*

proof –

have $\forall_F n$ *in sequentially. P (σ n) for P*

using *that by (metis eventually_atin_within_B eventually_bot)*

then show *False*

by (*meson eventually_False_sequentially_eventually_mono*)

qed

then show *?thesis*

using *derived_set_of_trivial_limit*

by (*fastforce elim!: atin_within_sequentially_sequence intro: atin_within_imp_M*)

qed

lemma *derived_set_of_sequentially_alt*:

```

mtopology derived_set_of S =
  {x.  $\exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ }
proof -
  have *:  $\exists \sigma. \text{range } \sigma \subseteq S \cap M - \{x\} \wedge \text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ 
    if  $\sigma: \text{range } \sigma \subseteq S - \{x\}$  and  $\text{lim: limitin mtopology } \sigma \text{ } x \text{ sequentially}$  for  $x \sigma$ 
  proof -
    obtain N where  $\forall n \geq N. \sigma n \in M \wedge d(\sigma n) x < 1$ 
      using  $\text{lim limit\_metric\_sequentially}$  by fastforce
    with  $\sigma$  obtain a where  $a: a \in S \cap M - \{x\}$  by auto
    show ?thesis
  proof (intro conjI exI)
    show  $\text{range } (\lambda n. \text{if } \sigma n \in M \text{ then } \sigma n \text{ else } a) \subseteq S \cap M - \{x\}$ 
      using a  $\sigma$  by fastforce
    show  $\text{limitin mtopology } (\lambda n. \text{if } \sigma n \in M \text{ then } \sigma n \text{ else } a) \text{ } x \text{ sequentially}$ 
      using  $\text{lim limit\_metric\_sequentially}$  by fastforce
  qed
qed
show ?thesis
  by (auto simp:  $\text{limitin\_mspace derived\_set\_of\_sequentially intro!: *}$ )
qed

lemma derived_set_of_sequentially_inj:
  mtopology derived_set_of S =
    {x  $\in M. \exists \sigma. \text{range } \sigma \subseteq S \cap M - \{x\} \wedge \text{inj } \sigma \wedge \text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ }
proof -
  have False
    if  $x \in M$  and  $\text{range } \sigma \subseteq S \cap M - \{x\}$ 
      and  $\text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ 
      and  $\text{atin\_within mtopology } x \text{ } S = \text{bot}$ 
    for  $x \sigma$ 
  proof -
    have  $\forall_F n \text{ in sequentially. } P(\sigma n)$  for P
      using that  $\text{derived\_set\_of\_sequentially\_alt derived\_set\_of\_trivial\_limit}$  by
fastforce
    then show False
      by (meson  $\text{eventually\_False\_sequentially eventually\_mono}$ )
  qed
  then show ?thesis
    using  $\text{derived\_set\_of\_trivial\_limit}$ 
    by (fastforce  $\text{elim!: atin\_within\_sequentially\_sequence intro: atin\_within\_imp\_M}$ )
qed

lemma derived_set_of_sequentially_inj_alt:
  mtopology derived_set_of S =
    {x.  $\exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{inj } \sigma \wedge \text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ }
proof -
  have  $\exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{inj } \sigma \wedge \text{limitin mtopology } \sigma \text{ } x \text{ sequentially}$ 

```

if *atin_within mtopology x S ≠ bot* **for** *x*
by (*metis Diff_empty Int_subset_iff atin_within_sequentially_sequence subset_Diff_insert* that)
moreover have *False*
if *range (λx. σ (x::nat)) ⊆ S - {x}*
and *limitin mtopology σ x sequentially*
and *atin_within mtopology x S = bot*
for *x σ*
proof –
have $\forall_F n$ *in sequentially. P (σ n)* **for** *P*
using *that derived_set_of_sequentially_alt derived_set_of_trivial_limit* **by**
fastforce
then show *False*
by (*meson eventually_False_sequentially_eventually_mono*)
qed
ultimately show *?thesis*
using *derived_set_of_trivial_limit* **by** (*fastforce intro: atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_decreasing:*

mtopology derived_set_of S =
 $\{x \in M. \exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{decreasing_dist } \sigma x \wedge \text{limitin mtopology } \sigma x \text{ sequentially}\}$

proof –

have $\exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{decreasing_dist } \sigma x \wedge \text{limitin mtopology } \sigma x$
sequentially

if *atin_within mtopology x S ≠ bot* **for** *x*

by (*metis Diff_empty atin_within_sequentially_sequence le_infE subset_Diff_insert* that)

moreover have *False*

if $x \in M$ **and** *range σ ⊆ S - {x}*

and *limitin mtopology σ x sequentially*

and *atin_within mtopology x S = bot*

for *x σ*

proof –

have $\forall_F n$ *in sequentially. P (σ n)* **for** *P*

using *that derived_set_of_sequentially_alt derived_set_of_trivial_limit* **by**
fastforce

then show *False*

by (*meson eventually_False_sequentially_eventually_mono*)

qed

ultimately show *?thesis*

using *derived_set_of_trivial_limit* **by** (*fastforce intro: atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_decreasing_alt:*

mtopology derived_set_of S =
 $\{x. \exists \sigma. \text{range } \sigma \subseteq S - \{x\} \wedge \text{decreasing_dist } \sigma x \wedge \text{limitin mtopology } \sigma x \text{ sequentially}\}$

using *derived_set_of_sequentially_alt* *derived_set_of_sequentially_decreasing*
by *auto*

lemma *closure_of_sequentially*:
 $mtopology\ closure_of\ S =$
 $\{x \in M. \exists \sigma. range\ \sigma \subseteq S \cap M \wedge limit\ in\ mtopology\ \sigma\ x\ sequentially\}$
by (*auto simp: closure_of_derived_set_of_sequentially*)

end

6.7.17 Three strong notions of continuity for metric spaces

Lipschitz continuity

definition *Lipschitz_continuous_map*
where *Lipschitz_continuous_map* \equiv
 $\lambda m1\ m2\ f. f \in mspace\ m1 \rightarrow mspace\ m2 \wedge$
 $(\exists B. \forall x \in mspace\ m1. \forall y \in mspace\ m1. mdist\ m2\ (f\ x)\ (f\ y) \leq B * mdist\ m1\ x\ y)$

lemma *Lipschitz_continuous_map_image*:
 $Lipschitz_continuous_map\ m1\ m2\ f \implies f \in mspace\ m1 \rightarrow mspace\ m2$
by (*simp add: Lipschitz_continuous_map_def*)

lemma *Lipschitz_continuous_map_pos*:
 $Lipschitz_continuous_map\ m1\ m2\ f \longleftrightarrow$
 $f \in mspace\ m1 \rightarrow mspace\ m2 \wedge$
 $(\exists B > 0. \forall x \in mspace\ m1. \forall y \in mspace\ m1. mdist\ m2\ (f\ x)\ (f\ y) \leq B * mdist\ m1\ x\ y)$

proof –

have $B * mdist\ m1\ x\ y \leq (|B| + 1) * mdist\ m1\ x\ y$ $|B| + 1 > 0$ **for** $x\ y\ B$

by (*auto simp add: mult_right_mono*)

then show *?thesis*

unfolding *Lipschitz_continuous_map_def* **by** (*meson dual_order.trans*)

qed

lemma *Lipschitz_continuous_map_eq*:
assumes $Lipschitz_continuous_map\ m1\ m2\ f \wedge x. x \in mspace\ m1 \implies f\ x = g\ x$
shows $Lipschitz_continuous_map\ m1\ m2\ g$
using *Lipschitz_continuous_map_def* **assms** **by** (*simp add: Lipschitz_continuous_map_pos Pi_iff*)

lemma *Lipschitz_continuous_map_from_submetric*:
assumes $Lipschitz_continuous_map\ m1\ m2\ f$
shows $Lipschitz_continuous_map\ (submetric\ m1\ S)\ m2\ f$
unfolding *Lipschitz_continuous_map_def*
proof
show $f \in mspace\ (submetric\ m1\ S) \rightarrow mspace\ m2$

using *Lipschitz_continuous_map_pos* *assms* **by** *fastforce*
qed (*use assms in* \langle *fastforce simp: Lipschitz_continuous_map_def* \rangle)

lemma *Lipschitz_continuous_map_from_submetric_mono*:

\llbracket *Lipschitz_continuous_map* (*submetric* *m1* *T*) *m2* *f*; $S \subseteq T$ \rrbracket
 \implies *Lipschitz_continuous_map* (*submetric* *m1* *S*) *m2* *f*

by (*metis Lipschitz_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric*)

lemma *Lipschitz_continuous_map_into_submetric*:

Lipschitz_continuous_map *m1* (*submetric* *m2* *S*) *f* \longleftrightarrow
 $f \in$ *mspace* *m1* $\rightarrow S \wedge$ *Lipschitz_continuous_map* *m1* *m2* *f*

by (*auto simp: Lipschitz_continuous_map_def*)

lemma *Lipschitz_continuous_map_const*:

Lipschitz_continuous_map *m1* *m2* $(\lambda x. c) \longleftrightarrow$
 $mspace\ m1 = \{\}$ $\vee c \in$ *mspace* *m2*

unfolding *Lipschitz_continuous_map_def* *Pi_iff*

by (*metis all_not_in_conv mdist_nonneg mdist_zero mult_1*)

lemma *Lipschitz_continuous_map_id*:

Lipschitz_continuous_map *m1* *m1* $(\lambda x. x)$

unfolding *Lipschitz_continuous_map_def* **by** (*metis funcset_id mult_1 order_refl*)

lemma *Lipschitz_continuous_map_compose*:

assumes *f*: *Lipschitz_continuous_map* *m1* *m2* *f* **and** *g*: *Lipschitz_continuous_map* *m2* *m3* *g*

shows *Lipschitz_continuous_map* *m1* *m3* $(g \circ f)$

unfolding *Lipschitz_continuous_map_def*

proof

show $g \circ f \in$ *mspace* *m1* \rightarrow *mspace* *m3*

by (*smt (verit, best) Lipschitz_continuous_map_image Pi_iff comp_apply f g*)

obtain *B* **where** $B: \forall x \in$ *mspace* *m1*. $\forall y \in$ *mspace* *m1*. $mdist\ m2\ (f\ x)\ (f\ y) \leq B$
 $*\ mdist\ m1\ x\ y$

using *assms* **unfolding** *Lipschitz_continuous_map_def* **by** *presburger*

obtain *C* **where** $C > 0$ **and** $C: \forall x \in$ *mspace* *m2*. $\forall y \in$ *mspace* *m2*. $mdist\ m3\ (g\ x)\ (g\ y) \leq C * mdist\ m2\ x\ y$

using *assms* **unfolding** *Lipschitz_continuous_map_pos* **by** *metis*

show $\exists B. \forall x \in$ *mspace* *m1*. $\forall y \in$ *mspace* *m1*. $mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \leq B * mdist\ m1\ x\ y$

proof (*intro strip exI*)

fix *x* *y*

assume $\S: x \in$ *mspace* *m1* $y \in$ *mspace* *m1*

then have $mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \leq C * mdist\ m2\ (f\ x)\ (f\ y)$

using *C Lipschitz_continuous_map_image f* **by** *fastforce*

also have $\dots \leq C * B * mdist\ m1\ x\ y$

by (*simp add: $\S B \langle 0 < C \rangle$*)

finally show $mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \leq C * B * mdist\ m1\ x\ y$.
 qed
 qed

Uniform continuity

definition *uniformly_continuous_map*

where *uniformly_continuous_map* \equiv

$$\lambda m1\ m2\ f. f \in mspace\ m1 \rightarrow mspace\ m2 \wedge$$

$$(\forall \varepsilon > 0. \exists \delta > 0. \forall x \in mspace\ m1. \forall y \in mspace\ m1.$$

$$mdist\ m1\ y\ x < \delta \longrightarrow mdist\ m2\ (f\ y)\ (f\ x) < \varepsilon)$$

lemma *uniformly_continuous_map_funspace*:

uniformly_continuous_map\ m1\ m2\ f $\implies f \in mspace\ m1 \rightarrow mspace\ m2$
 by (*simp add: uniformly_continuous_map_def*)

lemma *ucmap_A*:

assumes *uniformly_continuous_map\ m1\ m2\ f*

and $(\lambda n. mdist\ m1\ (\rho\ n)\ (\sigma\ n)) \longrightarrow 0$

and $range\ \rho \subseteq mspace\ m1$ $range\ \sigma \subseteq mspace\ m1$

shows $(\lambda n. mdist\ m2\ (f\ (\rho\ n))\ (f\ (\sigma\ n))) \longrightarrow 0$

using *assms*

unfolding *uniformly_continuous_map_def image_subset_iff tendsto_iff*

apply *clarsimp*

by (*metis (mono_tags, lifting) eventually_sequentially*)

lemma *ucmap_B*:

assumes $\S: \bigwedge \rho\ \sigma. \llbracket range\ \rho \subseteq mspace\ m1; range\ \sigma \subseteq mspace\ m1;$

$(\lambda n. mdist\ m1\ (\rho\ n)\ (\sigma\ n)) \longrightarrow 0 \rrbracket$

$\implies (\lambda n. mdist\ m2\ (f\ (\rho\ n))\ (f\ (\sigma\ n))) \longrightarrow 0$

and $0 < \varepsilon$

and ρ : $range\ \rho \subseteq mspace\ m1$

and σ : $range\ \sigma \subseteq mspace\ m1$

and $(\lambda n. mdist\ m1\ (\rho\ n)\ (\sigma\ n)) \longrightarrow 0$

shows $\exists n. mdist\ m2\ (f\ (\rho\ (n::nat)))\ (f\ (\sigma\ n)) < \varepsilon$

using \S [*OF* $\rho\ \sigma$] *assms* by (*meson LIMSEQ_le_const linorder_not_less*)

lemma *ucmap_C*:

assumes $\S: \bigwedge \rho\ \sigma\ \varepsilon. \llbracket \varepsilon > 0; range\ \rho \subseteq mspace\ m1; range\ \sigma \subseteq mspace\ m1;$

$((\lambda n. mdist\ m1\ (\rho\ n)\ (\sigma\ n)) \longrightarrow 0) \rrbracket$

$\implies \exists n. mdist\ m2\ (f\ (\rho\ n))\ (f\ (\sigma\ n)) < \varepsilon$

and *fm*: $f \in mspace\ m1 \rightarrow mspace\ m2$

shows *uniformly_continuous_map\ m1\ m2\ f*

proof –

{assume $\neg (\forall \varepsilon > 0. \exists \delta > 0. \forall x \in mspace\ m1. \forall y \in mspace\ m1. mdist\ m1\ y\ x < \delta$
 $\longrightarrow mdist\ m2\ (f\ y)\ (f\ x) < \varepsilon)$

then obtain ε where $\varepsilon > 0$

and $\bigwedge n. \exists x \in mspace\ m1. \exists y \in mspace\ m1. mdist\ m1\ y\ x < inverse(Suc\ n) \wedge$
 $mdist\ m2\ (f\ y)\ (f\ x) \geq \varepsilon$

```

    by (meson inverse_Suc linorder_not_le)
  then obtain  $\rho$   $\sigma$  where space: range  $\rho \subseteq \text{mspace } m1$  range  $\sigma \subseteq \text{mspace } m1$ 
    and dist:  $\bigwedge n. \text{mdist } m1 (\sigma n) (\rho n) < \text{inverse}(\text{Suc } n) \wedge \text{mdist } m2 (f(\sigma n)) (f(\rho n)) \geq \varepsilon$ 
    by (metis image_subset_iff)
  have False
    using § [OF <math>\langle \varepsilon > 0 \rangle space] dist Lim_null_comparison
    by (smt (verit) LIMSEQ_norm_0 inverse_eq_divide mdist_commute mdist_nonneg real_norm_def)
  }
  moreover
  have  $t \in \text{mspace } m2$  if  $t \in f \text{ ` } \text{mspace } m1$  for  $t$ 
    using fim that by blast
  ultimately show ?thesis
    by (fastforce simp: uniformly_continuous_map_def)
qed

```

```

lemma uniformly_continuous_map_sequentially:
  uniformly_continuous_map  $m1$   $m2$   $f \longleftrightarrow$ 
   $f \in \text{mspace } m1 \rightarrow \text{mspace } m2 \wedge$ 
   $(\forall \rho \sigma. \text{range } \rho \subseteq \text{mspace } m1 \wedge \text{range } \sigma \subseteq \text{mspace } m1 \wedge (\lambda n. \text{mdist } m1 (\rho n) (\sigma n)) \longrightarrow 0$ 
   $(\sigma n)) \longrightarrow 0$ 
   $\longrightarrow (\lambda n. \text{mdist } m2 (f (\rho n)) (f (\sigma n))) \longrightarrow 0)$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (simp add: ucmapp_A uniformly_continuous_map_funspace)
  show ?rhs  $\implies$  ?lhs
    by (intro ucmapp_B ucmapp_C) auto
qed

```

```

lemma uniformly_continuous_map_sequentially_alt:
  uniformly_continuous_map  $m1$   $m2$   $f \longleftrightarrow$ 
   $f \in \text{mspace } m1 \rightarrow \text{mspace } m2 \wedge$ 
   $(\forall \varepsilon > 0. \forall \rho \sigma. \text{range } \rho \subseteq \text{mspace } m1 \wedge \text{range } \sigma \subseteq \text{mspace } m1 \wedge$ 
   $((\lambda n. \text{mdist } m1 (\rho n) (\sigma n)) \longrightarrow 0)$ 
   $\longrightarrow (\exists n. \text{mdist } m2 (f (\rho n)) (f (\sigma n)) < \varepsilon))$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    using uniformly_continuous_map_funspace by (intro conjI strip ucmapp_B |
  fastforce simp: ucmapp_A)+
  show ?rhs  $\implies$  ?lhs
    by (intro ucmapp_C) auto
qed

```

```

lemma uniformly_continuous_map_eq:

```

$$\llbracket \bigwedge x. x \in \text{mspace } m1 \implies f x = g x; \text{uniformly_continuous_map } m1 \ m2 \ f \rrbracket$$

$$\implies \text{uniformly_continuous_map } m1 \ m2 \ g$$
by (*simp add: uniformly_continuous_map_def Pi_iff*)

lemma *uniformly_continuous_map_from_submetric*:
assumes *uniformly_continuous_map* *m1 m2 f*
shows *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
unfolding *uniformly_continuous_map_def*
proof
show $f \in \text{mspace } (\text{submetric } m1 \ S) \rightarrow \text{mspace } m2$
using *assms* **by** (*auto simp: uniformly_continuous_map_def*)
qed (*use assms in <force simp: uniformly_continuous_map_def>*)

lemma *uniformly_continuous_map_from_submetric_mono*:

$$\llbracket \text{uniformly_continuous_map } (\text{submetric } m1 \ T) \ m2 \ f; S \subseteq T \rrbracket$$

$$\implies \text{uniformly_continuous_map } (\text{submetric } m1 \ S) \ m2 \ f$$
by (*metis uniformly_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric*)

lemma *uniformly_continuous_map_into_submetric*:

$$\text{uniformly_continuous_map } m1 \ (\text{submetric } m2 \ S) \ f \longleftrightarrow$$

$$f \in \text{mspace } m1 \rightarrow S \wedge \text{uniformly_continuous_map } m1 \ m2 \ f$$
by (*auto simp: uniformly_continuous_map_def*)

lemma *uniformly_continuous_map_const*:

$$\text{uniformly_continuous_map } m1 \ m2 \ (\lambda x. c) \longleftrightarrow$$

$$\text{mspace } m1 = \{\} \vee c \in \text{mspace } m2$$
unfolding *uniformly_continuous_map_def Pi_iff*
by (*metis empty_iff equals0I mdist_zero*)

lemma *uniformly_continuous_map_id* [*simp*]:

$$\text{uniformly_continuous_map } m1 \ m1 \ (\lambda x. x)$$
by (*metis funcset_id uniformly_continuous_map_def*)

lemma *uniformly_continuous_map_compose*:
assumes *f: uniformly_continuous_map* *m1 m2 f* **and** *g: uniformly_continuous_map* *m2 m3 g*
shows *uniformly_continuous_map* *m1 m3 (g o f)*
using *f g* **unfolding** *uniformly_continuous_map_def comp_apply Pi_iff*
by *metis*

lemma *uniformly_continuous_map_real_const* [*simp*]:

$$\text{uniformly_continuous_map } m \ \text{euclidean_metric} \ (\lambda x. c)$$
by (*simp add: euclidean_metric_def uniformly_continuous_map_const*)

Equivalence between "abstract" and "type class" notions

lemma *uniformly_continuous_map_euclidean* [*simp*]:

$$\text{uniformly_continuous_map } (\text{submetric } \text{euclidean_metric} \ S) \ \text{euclidean_metric} \ f$$

$$= \text{uniformly_continuous_on } S \ f$$

by (auto simp add: uniformly_continuous_map_def uniformly_continuous_on_def)

Cauchy continuity

definition *Cauchy_continuous_map* where

$$\begin{aligned} \text{Cauchy_continuous_map} &\equiv \\ &\lambda m1\ m2\ f. \forall \sigma. \text{Metric_space.MCauchy } (m\text{space } m1) (m\text{dist } m1) \sigma \\ &\quad \longrightarrow \text{Metric_space.MCauchy } (m\text{space } m2) (m\text{dist } m2) (f \circ \sigma) \end{aligned}$$

lemma *Cauchy_continuous_map_euclidean* [simp]:

$$\begin{aligned} \text{Cauchy_continuous_map } (\text{submetric euclidean_metric } S) \text{ euclidean_metric } f \\ = \text{Cauchy_continuous_on } S\ f \end{aligned}$$

by (auto simp add: Cauchy_continuous_map_def Cauchy_continuous_on_def
Cauchy_def Met_TC.subspace Metric_space.MCauchy_def)

lemma *Cauchy_continuous_map_funspace*:

assumes *Cauchy_continuous_map* $m1\ m2\ f$
shows $f \in m\text{space } m1 \rightarrow m\text{space } m2$

proof clarsimp

fix x

assume $x \in m\text{space } m1$

then have $\text{Metric_space.MCauchy } (m\text{space } m1) (m\text{dist } m1) (\lambda n. x)$

by (simp add: Metric_space.MCauchy_const Metric_space_mspace_mdists)

then have $\text{Metric_space.MCauchy } (m\text{space } m2) (m\text{dist } m2) (f \circ (\lambda n. x))$

by (meson Cauchy_continuous_map_def assms)

then show $f\ x \in m\text{space } m2$

by (simp add: Metric_space.MCauchy_def [OF Metric_space_mspace_mdists])

qed

lemma *Cauchy_continuous_map_eq*:

$$\begin{aligned} \llbracket \bigwedge x. x \in m\text{space } m1 \implies f\ x = g\ x; \text{Cauchy_continuous_map } m1\ m2\ f \rrbracket \\ \implies \text{Cauchy_continuous_map } m1\ m2\ g \end{aligned}$$

by (simp add: image_subset_iff Cauchy_continuous_map_def Metric_space.MCauchy_def
[OF Metric_space_mspace_mdists])

lemma *Cauchy_continuous_map_from_submetric*:

assumes *Cauchy_continuous_map* $m1\ m2\ f$

shows $\text{Cauchy_continuous_map } (\text{submetric } m1\ S) m2\ f$

using *assms*

by (simp add: image_subset_iff Cauchy_continuous_map_def Metric_space.MCauchy_def
[OF Metric_space_mspace_mdists])

lemma *Cauchy_continuous_map_from_submetric_mono*:

$$\begin{aligned} \llbracket \text{Cauchy_continuous_map } (\text{submetric } m1\ T) m2\ f; S \subseteq T \rrbracket \\ \implies \text{Cauchy_continuous_map } (\text{submetric } m1\ S) m2\ f \end{aligned}$$

by (metis Cauchy_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric)

lemma *Cauchy_continuous_map_into_submetric*:

Cauchy_continuous_map *m1* (*submetric* *m2* *S*) *f* \longleftrightarrow
 $f \in \text{mspace } m1 \rightarrow S \wedge \text{Cauchy_continuous_map } m1 \ m2 \ f$ (**is** *?lhs* \longleftrightarrow *?rhs*)

proof –

have *?lhs* \implies *Cauchy_continuous_map* *m1* *m2* *f*

by (*simp* *add*: *Cauchy_continuous_map_def* *Metric_space.MCauchy_def* [*OF* *Metric_space_mspace_mdists*])

moreover **have** *?rhs* \implies *?lhs*

by (*auto* *simp*: *Cauchy_continuous_map_def* *Metric_space.MCauchy_def* [*OF* *Metric_space_mspace_mdists*])

ultimately **show** *?thesis*

by (*metis* *Cauchy_continuous_map_funspace* *Int_iff* *funcsetI* *funcset_mem* *mspace_submetric*)

qed

lemma *Cauchy_continuous_map_const* [*simp*]:

Cauchy_continuous_map *m1* *m2* $(\lambda x. c) \longleftrightarrow \text{mspace } m1 = \{\}$ $\vee c \in \text{mspace } m2$

proof –

have $\text{mspace } m1 = \{\} \implies \text{Cauchy_continuous_map } m1 \ m2 \ (\lambda x. c)$

by (*simp* *add*: *Cauchy_continuous_map_def* *Metric_space.MCauchy_def* *Metric_space_mspace_mdists*)

moreover **have** $c \in \text{mspace } m2 \implies \text{Cauchy_continuous_map } m1 \ m2 \ (\lambda x. c)$

by (*simp* *add*: *Cauchy_continuous_map_def* *o_def* *Metric_space.MCauchy_const* [*OF* *Metric_space_mspace_mdists*])

ultimately **show** *?thesis*

using *Cauchy_continuous_map_funspace* **by** *blast*

qed

lemma *Cauchy_continuous_map_id* [*simp*]:

Cauchy_continuous_map *m1* *m1* $(\lambda x. x)$

by (*simp* *add*: *Cauchy_continuous_map_def* *o_def*)

lemma *Cauchy_continuous_map_compose*:

assumes *f*: *Cauchy_continuous_map* *m1* *m2* *f* **and** *g*: *Cauchy_continuous_map* *m2* *m3* *g*

shows *Cauchy_continuous_map* *m1* *m3* $(g \circ f)$

by (*metis* (*no_types*, *lifting*) *Cauchy_continuous_map_def* *f* *fun.map_comp* *g*)

lemma *Lipschitz_imp_uniformly_continuous_map*:

assumes *Lipschitz_continuous_map* *m1* *m2* *f*

shows *uniformly_continuous_map* *m1* *m2* *f*

proof –

have $f \in \text{mspace } m1 \rightarrow \text{mspace } m2$

by (*simp* *add*: *Lipschitz_continuous_map_image* *assms*)

moreover **have** $\exists \delta > 0. \forall x \in \text{mspace } m1. \forall y \in \text{mspace } m1. \text{mdist } m1 \ y \ x < \delta \longrightarrow \text{mdist } m2 \ (f \ y) \ (f \ x) < \varepsilon$

if $\varepsilon > 0$ **for** ε

proof –

obtain B **where** $\forall x \in \text{mspace } m1. \forall y \in \text{mspace } m1. \text{mdist } m2 (f x) (f y) \leq B * \text{mdist } m1 x y$
and $B > 0$
using *that assms by (force simp add: Lipschitz_continuous_map_pos)*
then have $\forall x \in \text{mspace } m1. \forall y \in \text{mspace } m1. \text{mdist } m1 y x < \varepsilon / B \longrightarrow \text{mdist } m2 (f y) (f x) < \varepsilon$
by (*smt (verit, ccfv_SIG) less_divide_eq mdist_nonneg mult.commute that zero_less_divide_iff*)
with $\langle B > 0 \rangle$ **show** *?thesis*
by (*metis divide_pos_pos that*)
qed
ultimately show *?thesis*
by (*auto simp: uniformly_continuous_map_def*)
qed

lemma *uniformly_imp_Cauchy_continuous_map:*
uniformly_continuous_map $m1$ $m2$ $f \implies \text{Cauchy_continuous_map } m1$ $m2$ f
unfolding *uniformly_continuous_map_def Cauchy_continuous_map_def*
apply (*simp add: image_subset_iff o_def Metric_space.MCauchy_def [OF Metric_space_mspace_mdist]*)
by (*metis funcset_mem*)

lemma *locally_Cauchy_continuous_map:*
assumes $\varepsilon > 0$
and $\S: \bigwedge x. x \in \text{mspace } m1 \implies \text{Cauchy_continuous_map (submetric } m1 (\text{mball_of } m1 x \varepsilon)) m2 f$
shows *Cauchy_continuous_map* $m1$ $m2$ f
unfolding *Cauchy_continuous_map_def*
proof (*intro strip*)
interpret $M1: \text{Metric_space } \text{mspace } m1 \text{mdist } m1$
by (*simp add: Metric_space_mspace_mdist*)
interpret $M2: \text{Metric_space } \text{mspace } m2 \text{mdist } m2$
by (*simp add: Metric_space_mspace_mdist*)
fix σ
assume $\sigma: M1.MCauchy \sigma$
with $\langle \varepsilon > 0 \rangle$ **obtain** N **where** $N: \bigwedge n n'. \llbracket n \geq N; n' \geq N \rrbracket \implies \text{mdist } m1 (\sigma n) (\sigma n') < \varepsilon$
using $M1.MCauchy_def$ **by** *fastforce*
then have $M1.\text{mball} (\sigma N) \varepsilon \subseteq \text{mspace } m1$
by (*auto simp: image_subset_iff M1.mball_def*)
then interpret $MS1: \text{Metric_space } \text{mball_of } m1 (\sigma N) \varepsilon \cap \text{mspace } m1 \text{mdist } m1$
by (*simp add: M1.subspace*)
show $M2.MCauchy (f \circ \sigma)$
proof (*rule M2.MCauchy_offset*)
have $M1.MCauchy (\sigma \circ (+) N)$
by (*simp add: M1.MCauchy_imp_MCauchy_suffix \sigma*)
moreover have $\text{range } (\sigma \circ (+) N) \subseteq M1.\text{mball} (\sigma N) \varepsilon$
using N [*OF order_refl*] $M1.MCauchy_def \sigma$ **by** *fastforce*


```

ultimately have  $MS1.MCauchy (\sigma \circ (+) N)$ 
unfolding  $M1.MCauchy\_def MS1.MCauchy\_def$  by (simp add: mball_of_def)
moreover have  $\sigma N \in mspace\ m1$ 
  using  $M1.MCauchy\_def \sigma$  by auto
ultimately show  $M2.MCauchy (f \circ \sigma \circ (+) N)$ 
  unfolding comp_assoc
  by (metis  $\S Cauchy\_continuous\_map\_def mdist\_submetric mspace\_submetric$ )
next
fix  $n$ 
have  $\sigma n \in mspace\ m1$ 
  by (meson  $Metric\_space.MCauchy\_def Metric\_space\_mspace\_mdist \sigma range\_subsetD$ )
then have  $\sigma n \in mball\_of\ m1 (\sigma n) \varepsilon$ 
  by (simp add:  $Metric\_space.centre\_in\_mball\_iff Metric\_space\_mspace\_mdist$ 
   $assms(1) mball\_of\_def$ )
then show  $(f \circ \sigma) n \in mspace\ m2$ 
  using  $Cauchy\_continuous\_map\_funspace [OF \S [of \sigma n]] \langle \sigma n \in mspace\ m1 \rangle$ 
  by auto
qed
qed

```

```

context  $Metric\_space12$ 
begin

```

```

lemma  $Cauchy\_continuous\_imp\_continuous\_map$ :
  assumes  $Cauchy\_continuous\_map (metric (M1,d1)) (metric (M2,d2)) f$ 
  shows  $continuous\_map\ M1.mtopology\ M2.mtopology\ f$ 
proof (clarsimp simp:  $continuous\_map\_atin$ )
  fix  $x$ 
  assume  $x \in M1$ 
  show  $limitin\ M2.mtopology\ f (f x) (atin\ M1.mtopology\ x)$ 
  unfolding  $limit\_atin\_sequentially$ 
proof (intro conjI strip)
  show  $f x \in M2$ 
  using  $Cauchy\_continuous\_map\_funspace \langle x \in M1 \rangle assms$  by fastforce
  fix  $\sigma$ 
  assume  $range\ \sigma \subseteq M1 - \{x\} \wedge limitin\ M1.mtopology\ \sigma\ x\ sequentially$ 
  then have  $M1.MCauchy (\lambda n. if\ even\ n\ then\ \sigma (n\ div\ 2)\ else\ x)$ 
  by (force simp add:  $M1.MCauchy\_interleaving$ )
  then have  $M2.MCauchy (f \circ (\lambda n. if\ even\ n\ then\ \sigma (n\ div\ 2)\ else\ x))$ 
  using  $assms$  by (simp add:  $Cauchy\_continuous\_map\_def$ )
  then show  $limitin\ M2.mtopology (f \circ \sigma) (f x) sequentially$ 
  using  $M2.MCauchy\_interleaving [of\ f \circ \sigma\ f\ x]$ 
  by (simp add:  $o\_def if\_distrib cong: if\_cong$ )
qed
qed

```

```

lemma  $continuous\_imp\_Cauchy\_continuous\_map$ :
  assumes  $M1.mcomplete$ 
  and  $f: continuous\_map\ M1.mtopology\ M2.mtopology\ f$ 

```

```

shows Cauchy_continuous_map (metric (M1,d1)) (metric (M2,d2)) f
unfolding Cauchy_continuous_map_def
proof clarsimp
  fix  $\sigma$ 
  assume  $\sigma: M1.MCauchy\ \sigma$ 
  then obtain y where y: limitin M1.mtopology  $\sigma$  y sequentially
    using M1.mcomplete_def assms by blast
  have  $range\ (f \circ \sigma) \subseteq M2$ 
  using  $\sigma$  f by (simp add: M2.subspace M1.MCauchy_def M1.metric_continuous_map
image_subset_iff)
  then show M2.MCauchy ( $f \circ \sigma$ )
    using continuous_map_limit [OF f y] M2.convergent_imp_MCauchy
    by blast
qed

end

```

The same outside the locale

```

lemma Cauchy_continuous_imp_continuous_map:
  assumes Cauchy_continuous_map m1 m2 f
  shows continuous_map (mtopology_of m1) (mtopology_of m2) f
  using assms Metric_space12.Cauchy_continuous_imp_continuous_map [OF Metric_space12_mspace_mdists]
  by (auto simp add: mtopology_of_def)

```

```

lemma continuous_imp_Cauchy_continuous_map:
  assumes Metric_space.mcomplete (mspace m1) (mdist m1)
  and continuous_map (mtopology_of m1) (mtopology_of m2) f
  shows Cauchy_continuous_map m1 m2 f
  using assms Metric_space12.continuous_imp_Cauchy_continuous_map [OF Metric_space12_mspace_mdists]
  by (auto simp add: mtopology_of_def)

```

```

lemma uniformly_continuous_imp_continuous_map:
  uniformly_continuous_map m1 m2 f
   $\implies$  continuous_map (mtopology_of m1) (mtopology_of m2) f
  by (simp add: Cauchy_continuous_imp_continuous_map uniformly_imp_Cauchy_continuous_map)

```

```

lemma Lipschitz_continuous_imp_continuous_map:
  Lipschitz_continuous_map m1 m2 f
   $\implies$  continuous_map (mtopology_of m1) (mtopology_of m2) f
  by (simp add: Lipschitz_imp_uniformly_continuous_map uniformly_continuous_imp_continuous_map)

```

```

lemma Lipschitz_imp_Cauchy_continuous_map:
  Lipschitz_continuous_map m1 m2 f
   $\implies$  Cauchy_continuous_map m1 m2 f
  by (simp add: Lipschitz_imp_uniformly_continuous_map uniformly_imp_Cauchy_continuous_map)

```

```

lemma Cauchy_imp_uniformly_continuous_map:

```

```

assumes f: Cauchy_continuous_map m1 m2 f
and tbo: Metric_space.mtotally_bounded (mspace m1) (mdist m1) (mspace
m1)
shows uniformly_continuous_map m1 m2 f
unfolding uniformly_continuous_map_sequentially_alt_imp_conjL
proof (intro conjI strip)
show f ∈ mspace m1 → mspace m2
  by (simp add: Cauchy_continuous_map_funspace f)
interpret M1: Metric_space mspace m1 mdist m1
  by (simp add: Metric_space_mspace_mdist)
interpret M2: Metric_space mspace m2 mdist m2
  by (simp add: Metric_space_mspace_mdist)
fix ε :: real and ϱ σ
assume ε > 0
  and ϱ: range ϱ ⊆ mspace m1
  and σ: range σ ⊆ mspace m1
  and 0: (λn. mdist m1 (ϱ n) (σ n)) → 0
then obtain r1 where strict_mono r1 and r1: M1.MCauchy (ϱ ∘ r1)
  using M1.mtotally_bounded_sequentially tbo by meson
then obtain r2 where strict_mono r2 and r2: M1.MCauchy (σ ∘ r1 ∘ r2)
  by (metis M1.mtotally_bounded_sequentially tbo σ image_comp image_subset_iff
range_subsetD)
define r where r ≡ r1 ∘ r2
have r: strict_mono r
  by (simp add: r_def ⟨strict_mono r1⟩ ⟨strict_mono r2⟩ strict_mono_o)
define η where η ≡ λn. if even n then (ϱ ∘ r) (n div 2) else (σ ∘ r) (n div 2)
have M1.MCauchy η
  unfolding η_def M1.MCauchy_interleaving_gen
proof (intro conjI)
  show M1.MCauchy (ϱ ∘ r)
    by (simp add: M1.MCauchy_subsequence ⟨strict_mono r2⟩ fun.map_comp
r1 r_def)
  show M1.MCauchy (σ ∘ r)
    by (simp add: fun.map_comp r2 r_def)
  show (λn. mdist m1 ((ϱ ∘ r) n) ((σ ∘ r) n)) → 0
    using LIMSEQ_subseq_LIMSEQ [OF 0 r] by (simp add: o_def)
qed
then have Metric_space.MCauchy (mspace m2) (mdist m2) (f ∘ η)
  by (meson Cauchy_continuous_map_def f)
then have (λn. mdist m2 (f (ϱ (r n))) (f (σ (r n)))) → 0
  using M2.MCauchy_interleaving_gen [of f ∘ ϱ ∘ r f ∘ σ ∘ r]
  by (simp add: η_def o_def if_distrib cong: if_cong)
then show ∃n. mdist m2 (f (ϱ n)) (f (σ n)) < ε
  by (meson LIMSEQ_le_const ⟨0 < ε⟩ linorder_not_le)
qed

lemma continuous_imp_uniformly_continuous_map:
  compact_space (mtopology_of m1) ∧
  continuous_map (mtopology_of m1) (mtopology_of m2) f

```

\implies *uniformly_continuous_map* *m1* *m2* *f*
by (*simp* *add*: *Cauchy_imp_uniformly_continuous_map* *continuous_imp_Cauchy_continuous_map*
Metric_space.compact_space_eq_mcomplete_mtotally_bounded
Metric_space_mspace_mdists_topo_of_def)

lemma *continuous_eq_Cauchy_continuous_map*:
Metric_space.mcomplete (*mspace* *m1*) (*mdist* *m1*) \implies
continuous_map (*mtopology_of* *m1*) (*mtopology_of* *m2*) *f* \longleftrightarrow *Cauchy_continuous_map*
m1 *m2* *f*
using *Cauchy_continuous_imp_continuous_map* *continuous_imp_Cauchy_continuous_map*
by *blast*

lemma *continuous_eq_uniformly_continuous_map*:
compact_space (*mtopology_of* *m1*)
 \implies *continuous_map* (*mtopology_of* *m1*) (*mtopology_of* *m2*) *f* \longleftrightarrow
uniformly_continuous_map *m1* *m2* *f*
using *continuous_imp_uniformly_continuous_map* *uniformly_continuous_imp_continuous_map*
by *blast*

lemma *Cauchy_eq_uniformly_continuous_map*:
Metric_space.mtotally_bounded (*mspace* *m1*) (*mdist* *m1*) (*mspace* *m1*)
 \implies *Cauchy_continuous_map* *m1* *m2* *f* \longleftrightarrow *uniformly_continuous_map* *m1*
m2 *f*
using *Cauchy_imp_uniformly_continuous_map* *uniformly_imp_Cauchy_continuous_map*
by *blast*

lemma *Lipschitz_continuous_map_projections*:
Lipschitz_continuous_map (*prod_metric* *m1* *m2*) *m1* *fst*
Lipschitz_continuous_map (*prod_metric* *m1* *m2*) *m2* *snd*
by (*simp* *add*: *Lipschitz_continuous_map_def* *prod_dist_def* *fst_Pi* *snd_Pi*;
metis *mult_numeral_1* *real_sqrt_sum_squares_ge1* *real_sqrt_sum_squares_ge2*)⁺

lemma *Lipschitz_continuous_map_pairwise*:
Lipschitz_continuous_map *m* (*prod_metric* *m1* *m2*) *f* \longleftrightarrow
Lipschitz_continuous_map *m* *m1* (*fst* \circ *f*) \wedge *Lipschitz_continuous_map* *m* *m2*
(*snd* \circ *f*)
(is *?lhs* \longleftrightarrow *?rhs*)

proof
show *?lhs* \implies *?rhs*
by (*simp* *add*: *Lipschitz_continuous_map_compose* *Lipschitz_continuous_map_projections*)
have *Lipschitz_continuous_map* *m* (*prod_metric* *m1* *m2*) ($\lambda x. (f1\ x, f2\ x)$)
if *f1*: *Lipschitz_continuous_map* *m* *m1* *f1* **and** *f2*: *Lipschitz_continuous_map*
m *m2* *f2* **for** *f1* *f2*
proof –
obtain *B1* **where** *B1* > 0
and *B1*: $\bigwedge x\ y. [x \in \text{mspace } m; y \in \text{mspace } m] \implies \text{mdist } m1\ (f1\ x)\ (f1\ y) \leq$
B1 * *mdist* *m* *x* *y*
by (*meson* *Lipschitz_continuous_map_pos* *f1*)
obtain *B2* **where** *B2* > 0

```

    and B2:  $\bigwedge x y. [x \in \text{mspace } m; y \in \text{mspace } m] \implies \text{mdist } m2 (f2 x) (f2 y) \leq$ 
    B2 *  $\text{mdist } m x y$ 
    by (meson Lipschitz_continuous_map_pos f2)
  show ?thesis
    unfolding Lipschitz_continuous_map_pos
  proof (intro exI conjI strip)
    have f1im:  $f1 \in \text{mspace } m \rightarrow \text{mspace } m1$ 
      by (simp add: Lipschitz_continuous_map_image f1)
    moreover have f2im:  $f2 \in \text{mspace } m \rightarrow \text{mspace } m2$ 
      by (simp add: Lipschitz_continuous_map_image f2)
    ultimately show  $(\lambda x. (f1 x, f2 x)) \in \text{mspace } m \rightarrow \text{mspace } (\text{prod\_metric } m1$ 
    m2)
      by auto
    show  $B1+B2 > 0$ 
      using  $\langle 0 < B1 \rangle \langle 0 < B2 \rangle$  by linarith
    fix x y
    assume xy:  $x \in \text{mspace } m y \in \text{mspace } m$ 
    with f1im f2im have  $\text{mdist } (\text{prod\_metric } m1 m2) (f1 x, f2 x) (f1 y, f2 y) \leq$ 
     $\text{mdist } m1 (f1 x) (f1 y) + \text{mdist } m2 (f2 x) (f2 y)$ 
      unfolding mdist_prod_metric
    by (intro Metric_space12.prod_metric_le_components [OF Metric_space12_mspace_mdistr])
  auto
    also have  $\dots \leq (B1+B2) * \text{mdist } m x y$ 
      using B1 [OF xy] B2 [OF xy] by (simp add: vector_space_over_itself.scale_left_distrib)

    finally show  $\text{mdist } (\text{prod\_metric } m1 m2) (f1 x, f2 x) (f1 y, f2 y) \leq (B1+B2)$ 
    *  $\text{mdist } m x y$  .
  qed
  qed
  then show ?rhs  $\implies$  ?lhs
    by force
  qed

lemma uniformly_continuous_map_pairwise:
  uniformly_continuous_map m (prod_metric m1 m2) f  $\longleftrightarrow$ 
  uniformly_continuous_map m m1 (fst  $\circ$  f)  $\wedge$  uniformly_continuous_map m
  m2 (snd  $\circ$  f)
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  show ?lhs  $\implies$  ?rhs
    by (simp add: Lipschitz_continuous_map_projections Lipschitz_imp_uniformly_continuous_map
    uniformly_continuous_map_compose)
  have uniformly_continuous_map m (prod_metric m1 m2)  $(\lambda x. (f1 x, f2 x))$ 
    if f1: uniformly_continuous_map m m1 f1 and f2: uniformly_continuous_map
    m m2 f2 for f1 f2
  proof -
    show ?thesis
      unfolding uniformly_continuous_map_def
    proof (intro conjI strip)

```

```

have f1im: f1 ∈ mspace m → mspace m1
  by (simp add: uniformly_continuous_map_funspace f1)
moreover have f2im: f2 ∈ mspace m → mspace m2
  by (simp add: uniformly_continuous_map_funspace f2)
ultimately show (λx. (f1 x, f2 x)) ∈ mspace m → mspace (prod_metric m1
m2)
  by auto
fix ε:: real
assume ε > 0
obtain δ1 where δ1 > 0
  and δ1: ∧x y. [x ∈ mspace m; y ∈ mspace m; mdist m y x < δ1] ⇒ mdist
m1 (f1 y) (f1 x) < ε/2
  by (metis ‹0 < ε› f1 half_gt_zero uniformly_continuous_map_def)
obtain δ2 where δ2 > 0
  and δ2: ∧x y. [x ∈ mspace m; y ∈ mspace m; mdist m y x < δ2] ⇒ mdist
m2 (f2 y) (f2 x) < ε/2
  by (metis ‹0 < ε› f2 half_gt_zero uniformly_continuous_map_def)
show ∃δ > 0. ∀x ∈ mspace m. ∀y ∈ mspace m. mdist m y x < δ → mdist
(prod_metric m1 m2) (f1 y, f2 y) (f1 x, f2 x) < ε
proof (intro exI conjI strip)
  show min δ1 δ2 > 0
    using ‹0 < δ1› ‹0 < δ2› by auto
  fix x y
  assume xy: x ∈ mspace m y ∈ mspace m and d: mdist m y x < min δ1 δ2
  have *: mdist m1 (f1 y) (f1 x) < ε/2 mdist m2 (f2 y) (f2 x) < ε/2
    using δ1 δ2 d xy by auto
  have mdist (prod_metric m1 m2) (f1 y, f2 y) (f1 x, f2 x) ≤ mdist m1 (f1
y) (f1 x) + mdist m2 (f2 y) (f2 x)
    unfolding mdist_prod_metric using f1im f2im xy
  by (intro Metric_space12.prod_metric_le_components [OF Metric_space12_mspace_mdists])
auto
  also have ... < ε/2 + ε/2
    using * by simp
  finally show mdist (prod_metric m1 m2) (f1 y, f2 y) (f1 x, f2 x) < ε
    by simp
qed
qed
qed
then show ?rhs ⇒ ?lhs
  by force
qed

```

lemma *Cauchy_continuous_map_pairwise:*

Cauchy_continuous_map m (prod_metric m1 m2) f ↔ Cauchy_continuous_map m m1 (fst ∘ f) ∧ Cauchy_continuous_map m m2 (snd ∘ f)

by (auto simp: Cauchy_continuous_map_def Metric_space12.MCauchy_prod_metric[OF Metric_space12_mspace_mdists] comp_assoc)

lemma *Lipschitz_continuous_map_paired:*

Lipschitz_continuous_map m (*prod_metric* $m1$ $m2$) $(\lambda x. (f\ x, g\ x)) \longleftrightarrow$
Lipschitz_continuous_map m $m1$ $f \wedge$ *Lipschitz_continuous_map* m $m2$ g
by (*simp add: Lipschitz_continuous_map_pairwise_o_def*)

lemma *uniformly_continuous_map_paired*:

uniformly_continuous_map m (*prod_metric* $m1$ $m2$) $(\lambda x. (f\ x, g\ x)) \longleftrightarrow$
uniformly_continuous_map m $m1$ $f \wedge$ *uniformly_continuous_map* m $m2$ g
by (*simp add: uniformly_continuous_map_pairwise_o_def*)

lemma *Cauchy_continuous_map_paired*:

Cauchy_continuous_map m (*prod_metric* $m1$ $m2$) $(\lambda x. (f\ x, g\ x)) \longleftrightarrow$
Cauchy_continuous_map m $m1$ $f \wedge$ *Cauchy_continuous_map* m $m2$ g
by (*simp add: Cauchy_continuous_map_pairwise_o_def*)

lemma *mbounded_Lipschitz_continuous_image*:

assumes f : *Lipschitz_continuous_map* $m1$ $m2$ f **and** S : *Metric_space.mbounded*
 $(m\ space\ m1)$ $(m\ dist\ m1)$ S

shows *Metric_space.mbounded* $(m\ space\ m2)$ $(m\ dist\ m2)$ $(f\ 'S)$

proof –

obtain B **where** fim : $f \in m\ space\ m1 \rightarrow m\ space\ m2$

and $B > 0$ **and** B : $\bigwedge x\ y. \llbracket x \in m\ space\ m1; y \in m\ space\ m1 \rrbracket \implies m\ dist\ m2\ (f\ x)$
 $(f\ y) \leq B * m\ dist\ m1\ x\ y$

by (*metis Lipschitz_continuous_map_pos f*)

show *?thesis*

unfolding *Metric_space.mbounded_alt_pos* [*OF Metric_space_mspace_mdists*]

proof

obtain C **where** $S \subseteq m\ space\ m1$ **and** $C > 0$ **and** C : $\bigwedge x\ y. \llbracket x \in S; y \in S \rrbracket \implies$
 $m\ dist\ m1\ x\ y \leq C$

using S **by** (*auto simp: Metric_space.mbounded_alt_pos* [*OF Metric_space_mspace_mdists*])

show $f\ 'S \subseteq m\ space\ m2$

using fim $\langle S \subseteq m\ space\ m1 \rangle$ **by** *blast*

have $\bigwedge x\ y. \llbracket x \in S; y \in S \rrbracket \implies m\ dist\ m2\ (f\ x)\ (f\ y) \leq B * C$

by (*smt* (*verit*) $B\ C\ \langle 0 < B \rangle\ \langle S \subseteq m\ space\ m1 \rangle\ m\ dist_nonneg\ mult_mono$
 $subsetD$)

moreover **have** $B * C > 0$

by (*simp add:* $\langle 0 < B \rangle\ \langle 0 < C \rangle$)

ultimately show $\exists B > 0. \forall x \in f\ 'S. \forall y \in f\ 'S. m\ dist\ m2\ x\ y \leq B$

by *auto*

qed

qed

lemma *mtotally_bounded_Cauchy_continuous_image*:

assumes f : *Cauchy_continuous_map* $m1$ $m2$ f **and** S : *Metric_space.mtotally_bounded*
 $(m\ space\ m1)$ $(m\ dist\ m1)$ S

shows *Metric_space.mtotally_bounded* $(m\ space\ m2)$ $(m\ dist\ m2)$ $(f\ 'S)$

unfolding *Metric_space.mtotally_bounded_sequentially* [*OF Metric_space_mspace_mdists*]

proof (*intro conjI strip*)

have $S \subseteq m\ space\ m1$

using S **by** (*simp add: Metric_space.mtotally_bounded_sequentially* [*OF Met-*

```

ric_space_mspace_mdistr])
  then show  $f' S \subseteq \text{mspace } m2$ 
    using Cauchy_continuous_map_funspace f by blast
  fix  $\sigma :: \text{nat} \Rightarrow 'b$ 
  assume range  $\sigma \subseteq f' S$ 
  then have  $\forall n. \exists x. \sigma n = f x \wedge x \in S$ 
    by (meson imageE range_subsetD)
  then obtain  $\rho$  where  $\rho: \bigwedge n. \sigma n = f (\rho n)$  range  $\rho \subseteq S$ 
    by (metis image_subset_iff)
  then have  $\sigma = f \circ \rho$ 
    by fastforce
  obtain r where strict_mono r Metric_space.MCauchy (mspace m1) (mdist m1)
    ( $\rho \circ r$ )
    by (meson  $\rho S$  Metric_space.mtotally_bounded_sequentially[OF Metric_space_mspace_mdistr])
  then have Metric_space.MCauchy (mspace m2) (mdist m2) ( $f \circ \rho \circ r$ )
    using f unfolding Cauchy_continuous_map_def by (metis fun.map_comp)
  then show  $\exists r. \text{strict\_mono } r \wedge \text{Metric\_space.MCauchy } (mspace m2) (mdist$ 
     $m2) (\sigma \circ r)$ 
    using  $\langle \sigma = f \circ \rho \rangle \langle \text{strict\_mono } r \rangle$  by blast
qed

```

lemma Lipschitz_coefficient_pos:

```

  assumes  $x \in \text{mspace } m$   $y \in \text{mspace } m$   $f x \neq f y$ 
    and  $f \in \text{mspace } m \rightarrow \text{mspace } m2$ 
    and  $\bigwedge x y. \llbracket x \in \text{mspace } m; y \in \text{mspace } m \rrbracket$ 
       $\implies \text{mdist } m2 (f x) (f y) \leq k * \text{mdist } m x y$ 
  shows  $0 < k$ 
  using assms by (smt (verit, best) Pi_iff mdist_nonneg mdist_zero mult_nonpos_nonneg)

```

lemma Lipschitz_continuous_map_metric:

```

  Lipschitz_continuous_map (prod_metric m m) euclidean_metric ( $\lambda(x,y). \text{mdist}$ 
     $m x y$ )
  proof -
  have  $\bigwedge x y x' y'. \llbracket x \in \text{mspace } m; y \in \text{mspace } m; x' \in \text{mspace } m; y' \in \text{mspace } m \rrbracket$ 
     $\implies |\text{mdist } m x y - \text{mdist } m x' y'| \leq 2 * \text{sqrt} ((\text{mdist } m x x')^2 + (\text{mdist } m$ 
     $y y')^2)$ 
    by (smt (verit, del_insts) mdist_commute mdist_triangle real_sqrt_sum_squares_ge2)
  then show ?thesis
    by (fastforce simp add: Lipschitz_continuous_map_def prod_dist_def dist_real_def)
  qed

```

lemma Lipschitz_continuous_map_mdistr:

```

  assumes f: Lipschitz_continuous_map m m' f
    and g: Lipschitz_continuous_map m m' g
  shows Lipschitz_continuous_map m euclidean_metric ( $\lambda x. \text{mdist } m' (f x) (g x)$ )
    (is Lipschitz_continuous_map m _ ?h)
  proof -
  have eq: ?h = (( $\lambda(x,y). \text{mdist } m' x y$ )  $\circ (\lambda x. (f x, g x))$ )
    by force

```



```

show ?thesis
  unfolding eq
proof (rule Lipschitz_continuous_map_compose)
  show Lipschitz_continuous_map m (prod_metric m' m') ( $\lambda x. (f x, g x)$ )
    by (simp add: Lipschitz_continuous_map_paired f g)
  show Lipschitz_continuous_map (prod_metric m' m') euclidean_metric ( $\lambda(x,y). mdist m' x y$ )
    by (simp add: Lipschitz_continuous_map_metric)
qed
qed

lemma uniformly_continuous_map_mdists:
  assumes f: uniformly_continuous_map m m' f
    and g: uniformly_continuous_map m m' g
  shows uniformly_continuous_map m euclidean_metric ( $\lambda x. mdist m' (f x) (g x)$ )
    (is uniformly_continuous_map m _ ?h)
proof -
  have eq: ?h = (( $\lambda(x,y). mdist m' x y$ )  $\circ$  ( $\lambda x. (f x, g x)$ ))
    by force
  show ?thesis
    unfolding eq
  proof (rule uniformly_continuous_map_compose)
  show uniformly_continuous_map m (prod_metric m' m') ( $\lambda x. (f x, g x)$ )
    by (simp add: uniformly_continuous_map_paired f g)
  show uniformly_continuous_map (prod_metric m' m') euclidean_metric ( $\lambda(x,y). mdist m' x y$ )
    by (simp add: Lipschitz_continuous_map_metric Lipschitz_imp_uniformly_continuous_map)
qed
qed

lemma Cauchy_continuous_map_mdists:
  assumes f: Cauchy_continuous_map m m' f
    and g: Cauchy_continuous_map m m' g
  shows Cauchy_continuous_map m euclidean_metric ( $\lambda x. mdist m' (f x) (g x)$ )
    (is Cauchy_continuous_map m _ ?h)
proof -
  have eq: ?h = (( $\lambda(x,y). mdist m' x y$ )  $\circ$  ( $\lambda x. (f x, g x)$ ))
    by force
  show ?thesis
    unfolding eq
  proof (rule Cauchy_continuous_map_compose)
  show Cauchy_continuous_map m (prod_metric m' m') ( $\lambda x. (f x, g x)$ )
    by (simp add: Cauchy_continuous_map_paired f g)
  show Cauchy_continuous_map (prod_metric m' m') euclidean_metric ( $\lambda(x,y). mdist m' x y$ )
    by (simp add: Lipschitz_continuous_map_metric Lipschitz_imp_Cauchy_continuous_map)
qed
qed

```

lemma *mtopology_of_prod_metric* [*simp*]:

$$\text{mtopology_of } (\text{prod_metric } m1 \ m2) = \text{prod_topology } (\text{mtopology_of } m1) \\ (\text{mtopology_of } m2)$$
by (*simp add: mtopology_of_def Metric_space12.mtopology_prod_metric*[*OF Metric_space12_mspace_md*])

lemma *continuous_map_metric*:

$$\text{continuous_map } (\text{prod_topology } (\text{mtopology_of } m) (\text{mtopology_of } m)) \ \text{euclidean} \\ (\lambda(x,y). \text{mdist } m \ x \ y)$$
using *Lipschitz_continuous_imp_continuous_map* [*OF Lipschitz_continuous_map_metric*]
by *auto*

lemma *continuous_map_mdalt*:
assumes *continuous_map X* (*prod_topology (mtopology_of m) (mtopology_of m)*) *f*
shows *continuous_map X euclidean* ($\lambda x. \text{case_prod } (\text{mdist } m) (f \ x)$)
(*is continuous_map _ _ ?h*)
proof –
have *eq: ?h = case_prod (mdist m) o f*
by *force*
show *?thesis*
unfolding *eq*
using *assms continuous_map_compose continuous_map_metric* **by** *blast*
qed

lemma *continuous_map_md* [*continuous_intros*]:
assumes *f: continuous_map X (mtopology_of m) f*
and *g: continuous_map X (mtopology_of m) g*
shows *continuous_map X euclidean* ($\lambda x. \text{mdist } m (f \ x) (g \ x)$)
(*is continuous_map X _ ?h*)
proof –
have *eq: ?h = ((λ(x,y). mdist m x y) o (λx. (f x, g x)))*
by *force*
show *?thesis*
unfolding *eq*
proof (*rule continuous_map_compose*)
show *continuous_map X (prod_topology (mtopology_of m) (mtopology_of m))*
($\lambda x. (f \ x, g \ x)$)
by (*simp add: continuous_map_paired f g*)
qed (*simp add: continuous_map_metric*)
qed

lemma *continuous_on_md*:
 $a \in \text{mspace } m \implies \text{continuous_map } (\text{mtopology_of } m) \ \text{euclidean } (\text{mdist } m \ a)$
by (*simp add: continuous_map_md*)

6.7.18 Isometries

lemma (in *Metric_space12*) *isometry_imp_embedding_map*:
assumes *fim*: $f \in M1 \rightarrow M2$ **and** *d*: $\bigwedge x y. \llbracket x \in M1; y \in M1 \rrbracket \implies d2 (f x) (f y) = d1 x y$
shows *embedding_map* *M1.mtopology* *M2.mtopology* *f*
proof –
have *inj_on* *f* *M1*
by (*metis* *M1.zero* *d* *inj_onI*)
then obtain *g* **where** $g: \bigwedge x. x \in M1 \implies g (f x) = x$
by (*metis* *inv_into_f_f*)
have *homeomorphic_maps* *M1.mtopology* (*subtopology* *M2.mtopology* (*f* ‘ *topspace* *M1.mtopology*)) *f* *g*
unfolding *homeomorphic_maps_def*
proof (*intro* *conjI*; *clarsimp*)
show *continuous_map* *M1.mtopology* (*subtopology* *M2.mtopology* (*f* ‘ *M1*)) *f*
proof (*rule* *continuous_map_into_subtopology*)
show *continuous_map* *M1.mtopology* *M2.mtopology* *f*
by (*metis* *M1.metric_continuous_map* *M2.Metric_space_axioms* *d* *fim* *image_subset_iff_funcset*)
qed *simp*
have *Lipschitz_continuous_map* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*)) (*metric*(*M1*,*d1*))
g
unfolding *Lipschitz_continuous_map_def*
proof (*intro* *conjI* *exI* *strip*; *simp*)
show $d1 (g x) (g y) \leq 1 * d2 x y$ **if** $x \in f$ ‘ *M1* $\wedge x \in M2$ **and** $y \in f$ ‘ *M1* $\wedge y \in M2$ **for** $x y$
using *that* *d* *g* **by** *force*
qed (*use* *g* **in** *auto*)
then have *continuous_map* (*mtopology_of* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*))) *M1.mtopology* *g*
using *Lipschitz_continuous_imp_continuous_map* **by** *force*
moreover have *mtopology_of* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*)) = *subtopology* *M2.mtopology* (*f* ‘ *M1*)
by (*simp* *add*: *mtopology_of_submetric*)
ultimately show *continuous_map* (*subtopology* *M2.mtopology* (*f* ‘ *M1*)) *M1.mtopology*
g
by *simp*
qed (*use* *g* **in** *auto*)
then show *?thesis*
by (*auto* *simp*: *embedding_map_def* *homeomorphic_map_maps*)
qed

lemma (in *Metric_space12*) *isometry_imp_homeomorphic_map*:
assumes *fim*: f ‘ *M1* = *M2* **and** *d*: $\bigwedge x y. \llbracket x \in M1; y \in M1 \rrbracket \implies d2 (f x) (f y) = d1 x y$
shows *homeomorphic_map* *M1.mtopology* *M2.mtopology* *f*
by (*metis* *image_eqI* *M1.topspace_mtopology* *M2.subtopology_mspace* *d* *embedding_map_def* *fim* *isometry_imp_embedding_map* *Pi_iff*)

6.7.19 "Capped" equivalent bounded metrics and general product metrics

definition (in *Metric_space*) *capped_dist* **where**

capped_dist $\equiv \lambda \delta x y. \text{if } \delta > 0 \text{ then } \min \delta (d x y) \text{ else } d x y$

lemma (in *Metric_space*) *capped_dist*: *Metric_space* *M* (*capped_dist* δ)

proof

fix *x y*

assume $x \in M y \in M$

then show (*capped_dist* $\delta x y = 0$) = ($x = y$)

by (*smt* (*verit*, *best*) *capped_dist_def* *zero*)

fix *z*

assume $z \in M$

show *capped_dist* $\delta x z \leq \text{capped_dist } \delta x y + \text{capped_dist } \delta y z$

unfolding *capped_dist_def* **using** $\langle x \in M \rangle \langle y \in M \rangle \langle z \in M \rangle$

by (*smt* (*verit*, *del_insts*) *Metric_space.mdist_pos_eq* *Metric_space_axioms* *mdist_reverse_triangle*)

qed (use *capped_dist_def* *commute* **in** *auto*)

definition *capped_metric* **where**

capped_metric $\delta m \equiv \text{metric}(\text{mspace } m, \text{Metric_space.capped_dist } (\text{mdist } m) \delta)$

lemma *capped_metric*:

capped_metric $\delta m = (\text{if } \delta \leq 0 \text{ then } m \text{ else } \text{metric}(\text{mspace } m, \lambda x y. \min \delta (\text{mdist } m x y)))$

proof –

interpret *Metric_space* *mspace* *m* *mdist* *m*

by (*simp* *add*: *Metric_space_mspace_mdist*)

show *?thesis*

by (*auto* *simp*: *capped_metric_def* *capped_dist_def*)

qed

lemma *capped_metric_mspace* [*simp*]:

mspace (*capped_metric* δm) = *mspace* *m*

apply (*simp* *add*: *capped_metric_not_le*)

by (*smt* (*verit*, *ccfv_threshold*) *Metric_space.mspace_metric* *Metric_space_def* *Metric_space_mspace_mdist*)

lemma *capped_metric_mdist*:

mdist (*capped_metric* δm) = $(\lambda x y. \text{if } \delta \leq 0 \text{ then } \text{mdist } m x y \text{ else } \min \delta (\text{mdist } m x y))$

apply (*simp* *add*: *capped_metric_not_le*)

by (*smt* (*verit*, *del_insts*) *Metric_space.mdist_metric* *Metric_space_def* *Metric_space_mspace_mdist*)

lemma *mdist_capped_le*: *mdist* (*capped_metric* δm) $x y \leq \text{mdist } m x y$

by (*simp* *add*: *capped_metric_mdist*)

lemma *mdist_capped*: $\delta > 0 \implies \text{mdist} (\text{capped_metric } \delta \ m) \ x \ y \leq \delta$
by (*simp add: capped_metric_mdists*)

lemma *mball_of_capped_metric* [*simp*]:

assumes $x \in \text{mspace } m \ r > \delta \ \delta > 0$

shows $\text{mball_of} (\text{capped_metric } \delta \ m) \ x \ r = \text{mspace } m$

proof –

interpret *Metric_space* $\text{mspace } m \ \text{mdist } m$

by (*simp add: Metric_space_mspace_mdists*)

have $\text{Metric_space.mball} (\text{mspace } m) (\text{mdist} (\text{capped_metric } \delta \ m)) \ x \ r \subseteq \text{mspace } m$

m

by (*metis Metric_space.mball_subset_mspace Metric_space_mspace_mdists capped_metric_mspace*)

moreover have $\text{mspace } m \subseteq \text{Metric_space.mball} (\text{mspace } m) (\text{mdist} (\text{capped_metric } \delta \ m)) \ x \ r$

by (*smt (verit) Metric_space.in_mball Metric_space_mspace_mdists assms capped_metric_mspace mdists_capped subset_eq*)

ultimately show *?thesis*

by (*simp add: mball_of_def*)

qed

lemma *Metric_space_capped_dist*[*simp*]:

$\text{Metric_space} (\text{mspace } m) (\text{Metric_space.capped_dist} (\text{mdist } m) \ \delta)$

using *Metric_space.capped_dist Metric_space_mspace_mdists* **by** *blast*

lemma *mtopology_capped_metric*:

$\text{mtopology_of} (\text{capped_metric } \delta \ m) = \text{mtopology_of } m$

proof (*cases* $\delta > 0$)

case *True*

interpret *Metric_space* $\text{mspace } m \ \text{mdist } m$

by (*simp add: Metric_space_mspace_mdists*)

interpret *Cap*: *Metric_space* $\text{mspace } m \ \text{mdist} (\text{capped_metric } \delta \ m)$

by (*metis Metric_space_mspace_mdists capped_metric_mspace*)

show *?thesis*

unfolding *topology_eq*

proof

fix *S*

show $\text{openin} (\text{mtopology_of} (\text{capped_metric } \delta \ m)) \ S = \text{openin} (\text{mtopology_of } m) \ S$

proof (*cases* $S \subseteq \text{mspace } m$)

case *True*

have $\text{mball } x \ r \subseteq \text{Cap.mball } x \ r$ **for** $x \ r$

by (*smt (verit, ccfv_SIG) Cap.in_mball in_mball mdists_capped_le subsetI*)

moreover have $\exists r > 0. \text{Cap.mball } x \ r \subseteq S$ **if** $r > 0 \ x \in S$ **and** $r: \text{mball } x \ r \subseteq$

S **for** $r \ x$

proof (*intro exI conjI*)

show $\min (\delta/2) \ r > 0$

using $\langle r > 0 \rangle \ \langle \delta > 0 \rangle$ **by** *force*

show $\text{Cap.mball } x (\min (\delta/2) \ r) \subseteq S$

```

    using that
    by clarsimp (smt (verit) capped_metric_md_dist field_sum_of_halves
in_mball subsetD)
  qed
  ultimately have  $(\exists r > 0. \text{Cap.mball } x \ r \subseteq S) = (\exists r > 0. \text{mball } x \ r \subseteq S)$  if  $x \in S$  for  $x$ 
  by (meson subset_trans that)
  then show ?thesis
  by (simp add: mtopology_of_def openin_mtopology Cap.openin_mtopology)
  qed (simp add: openin_closedin_eq)
  qed
qed (simp add: capped_metric)

```

Might have been easier to prove this within the locale to start with (using Self)

```

lemma (in Metric_space) mtopology_capped_metric:
  Metric_space.mtopology M (capped_dist  $\delta$ ) = mtopology
  using mtopology_capped_metric [of  $\delta$  metric(M,d)]
  by (simp add: Metric_space.mtopology_of capped_dist capped_metric_def)

```

```

lemma (in Metric_space) MCauchy_capped_metric:
  Metric_space.MCauchy M (capped_dist  $\delta$ )  $\sigma \longleftrightarrow$  MCauchy  $\sigma$ 
proof (cases  $\delta > 0$ )
  case True
  interpret Cap: Metric_space M capped_dist  $\delta$ 
  by (simp add: capped_dist)
  show ?thesis
  proof
    assume  $\sigma: \text{Cap.MCauchy } \sigma$ 
    show MCauchy  $\sigma$ 
    unfolding MCauchy_def
  proof (intro conjI strip)
    show range  $\sigma \subseteq M$ 
    using Cap.MCauchy_def  $\sigma$  by presburger
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    with True  $\sigma$ 
    obtain  $N$  where  $\forall n \ n'. N \leq n \longrightarrow N \leq n' \longrightarrow \text{capped\_dist } \delta (\sigma \ n) (\sigma \ n') < \min \ \delta \ \varepsilon$ 
    unfolding Cap.MCauchy_def by (metis min_less_iff_conj)
    with True show  $\exists N. \forall n \ n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma \ n) (\sigma \ n') < \varepsilon$ 
    by (force simp: capped_dist_def)
  qed
  qed
next
  assume MCauchy  $\sigma$ 
  then show Cap.MCauchy  $\sigma$ 
  unfolding MCauchy_def Cap.MCauchy_def by (force simp: capped_dist_def)
  qed
qed (simp add: capped_dist_def)

```

lemma (in *Metric_space*) *mcomplete_capped_metric*:

Metric_space.mcomplete M (*capped_dist* δ) \longleftrightarrow *mcomplete*
by (*simp add: MCauchy_capped_metric Metric_space.mcomplete_def capped_dist*
mtopology_capped_metric mcomplete_def)

lemma *bounded_equivalent_metric*:

assumes $\delta > 0$
obtains m' **where** *mspace* $m' =$ *mspace* m *mtopology_of* $m' =$ *mtopology_of* m
 $\bigwedge x y. \text{mdist } m' x y < \delta$
proof
let $?m =$ *capped_metric* $(\delta/2)$ m
fix $x y$
show $\text{mdist } ?m x y < \delta$
by (*smt (verit, best) assms field_sum_of_halves mdist_capped*)
qed (*auto simp: mtopology_capped_metric*)

A technical lemma needed below

lemma *Sup_metric_cartesian_product*:

fixes $I m$
defines $S \equiv \text{PiE } I (\text{mspace } \circ m)$
defines $D \equiv \lambda x y. \text{if } x \in S \wedge y \in S \text{ then } \text{SUP } i \in I. \text{mdist } (m i) (x i) (y i) \text{ else } 0$
defines $m' \equiv \text{metric}(S, D)$
assumes $I \neq \{\}$
and $c: \bigwedge i x y. \llbracket i \in I; x \in \text{mspace}(m i); y \in \text{mspace}(m i) \rrbracket \implies \text{mdist } (m i) x y \leq c$
shows *Metric_space* $S D$
and $\forall x \in S. \forall y \in S. \forall b. D x y \leq b \longleftrightarrow (\forall i \in I. \text{mdist } (m i) (x i) (y i) \leq b)$ (is *?the2*)
proof –
have *bdd*: *bdd_above* $((\lambda i. \text{mdist } (m i) (x i) (y i)) \text{ ` } I)$
if $x \in S y \in S$ **for** $x y$
using c **that** **by** (*force simp: S_def bdd_above_def*)
have *D_iff*: $D x y \leq b \longleftrightarrow (\forall i \in I. \text{mdist } (m i) (x i) (y i) \leq b)$
if $x \in S y \in S$ **for** $x y b$
using *that* $\langle I \neq \{\} \rangle$ **by** (*simp add: D_def PiE_iff cSup_le_iff bdd*)
show *Metric_space* $S D$
proof
fix $x y$
show *D0*: $0 \leq D x y$
using *bdd*
apply (*simp add: D_def*)
by (*meson* $\langle I \neq \{\} \rangle$ *cSUP_upper dual_order.trans ex_in_conv mdist_nonneg*)
show $D x y = D y x$
by (*simp add: D_def mdist_commute*)
assume $x \in S$ **and** $y \in S$
then
have $D x y = 0 \longleftrightarrow (\forall i \in I. \text{mdist } (m i) (x i) (y i) = 0)$

```

    using D0 D_iff [of x y 0] nle_le by fastforce
  also have ...  $\longleftrightarrow$   $x = y$ 
    using  $\langle x \in S \rangle \langle y \in S \rangle$  by (fastforce simp: S_def PiE_iff extensional_def)
  finally show (D x y = 0)  $\longleftrightarrow$  (x = y) .
  fix z
  assume z  $\in$  S
  have mdist (m i) (x i) (z i)  $\leq$  D x y + D y z if i  $\in$  I for i
  proof -
    have mdist (m i) (x i) (z i)  $\leq$  mdist (m i) (x i) (y i) + mdist (m i) (y i) (z
i)
    by (metis PiE_E S_def  $\langle x \in S \rangle \langle y \in S \rangle \langle z \in S \rangle$  comp_apply mdist_triangle
that)
  also have ...  $\leq$  D x y + D y z
    using  $\langle x \in S \rangle \langle y \in S \rangle \langle z \in S \rangle$  by (meson D_iff add_mono order_refl that)
  finally show ?thesis .
  qed
  then show D x z  $\leq$  D x y + D y z
    by (simp add: D_iff  $\langle x \in S \rangle \langle z \in S \rangle$ )
  qed
  then interpret Metric_space S D .
  show ?the2
  proof (intro strip)
    show (D x y  $\leq$  b) = ( $\forall i \in I$ . mdist (m i) (x i) (y i)  $\leq$  b)
      if x  $\in$  S and y  $\in$  S for x y b
      using that by (simp add: D_iff m'_def)
  qed
  qed

```

lemma metrizable_topology_A:

```

  assumes metrizable_space (product_topology X I)
  shows (product_topology X I) = trivial_topology  $\vee$  ( $\forall i \in I$ . metrizable_space
(X i))
  by (meson assms metrizable_space_retraction_map_image retraction_map_product_projection)

```

lemma metrizable_topology_C:

```

  assumes completely_metrizable_space (product_topology X I)
  shows (product_topology X I) = trivial_topology  $\vee$  ( $\forall i \in I$ . completely_metrizable_space
(X i))
  by (meson assms completely_metrizable_space_retraction_map_image retraction_map_product_projection)

```

lemma metrizable_topology_B:

```

  fixes a X I
  defines L  $\equiv$  {i  $\in$  I.  $\nexists$  a. topspace (X i)  $\subseteq$  {a}}
  assumes topspace (product_topology X I)  $\neq$  {}
    and met: metrizable_space (product_topology X I)
    and  $\bigwedge i$ . i  $\in$  I  $\implies$  metrizable_space (X i)
  shows countable L
  proof -

```



```

have  $\bigwedge i. \exists p q. i \in L \longrightarrow p \in \text{topspace}(X i) \wedge q \in \text{topspace}(X i) \wedge p \neq q$ 
  unfolding  $L\_def$  by blast
then obtain  $\varphi \psi$  where  $\varphi: \bigwedge i. i \in L \implies \varphi i \in \text{topspace}(X i) \wedge \psi i \in \text{topspace}(X i) \wedge \varphi i \neq \psi i$ 
  by metis
obtain  $z$  where  $z: z \in (\prod_{E} i \in I. \text{topspace}(X i))$ 
  using  $assms(2)$  by fastforce
define  $p$  where  $p \equiv \lambda i. \text{if } i \in L \text{ then } \varphi i \text{ else } z i$ 
define  $q$  where  $q \equiv \lambda i j. \text{if } j = i \text{ then } \psi i \text{ else } p j$ 
have  $p: p \in \text{topspace}(\text{product\_topology } X I)$ 
  using  $z \varphi$  by (auto simp:  $p\_def L\_def$ )
then have  $q: \bigwedge i. i \in L \implies q i \in \text{topspace}(\text{product\_topology } X I)$ 
  by (auto simp:  $L\_def q\_def \varphi$ )
have  $fin: \text{finite } \{i \in L. q i \notin U\}$  if  $U: \text{openin}(\text{product\_topology } X I) U \wedge p \in U$ 
for  $U$ 
proof -
  obtain  $V$  where  $V: \text{finite } \{i \in I. V i \neq \text{topspace}(X i)\} (\forall i \in I. \text{openin}(X i) (V i)) p \in \text{Pi}_{E} I V \text{ Pi}_{E} I V \subseteq U$ 
    using  $U$  by (force simp:  $\text{openin\_product\_topology\_alt}$ )
  moreover
  have  $V x \neq \text{topspace}(X x)$  if  $x \in L$  and  $q x \notin U$  for  $x$ 
    using that  $V q$ 
  by (smt (verit, del_insts)  $\text{Pi}_{E\_iff} q\_def \text{subset\_eq} \text{topspace\_product\_topology}$ )
  then have  $\{i \in L. q i \notin U\} \subseteq \{i \in I. V i \neq \text{topspace}(X i)\}$ 
    by (force simp:  $L\_def$ )
  ultimately show ?thesis
    by (meson  $\text{finite\_subset}$ )
qed
obtain  $M d$  where  $\text{Metric\_space } M d$  and  $XI: \text{product\_topology } X I = \text{Metric\_space.mtopology } M d$ 
  using  $\text{met metrizable\_space\_def}$  by blast
then interpret  $\text{Metric\_space } M d$ 
  by blast
define  $C$  where  $C \equiv \bigcup n::\text{nat}. \{i \in L. q i \notin \text{mball } p (\text{inverse}(\text{Suc } n))\}$ 
have  $\text{finite } \{i \in L. q i \notin \text{mball } p (\text{inverse}(\text{real}(\text{Suc } n)))\}$  for  $n$ 
  using  $XI p$  by (intro  $fin$ ; force)
then have  $\text{countable } C$ 
  unfolding  $C\_def$ 
  by (meson  $\text{countableI\_type} \text{countable\_UN} \text{countable\_finite}$ )
moreover have  $L \subseteq C$ 
proof (clarsimp simp:  $C\_def$ )
  fix  $i$ 
  assume  $i \in L$  and  $q i \in M$  and  $p \in M$ 
  then show  $\exists n. \neg d p (q i) < \text{inverse}(1 + \text{real } n)$ 
    using  $\text{reals\_Archimedean}$  [of  $d p (q i)$ ]
    by (metis  $\varphi \text{mdist\_pos\_eq not\_less\_iff\_gr\_or\_eq of\_nat\_Suc } p\_def q\_def$ )
qed
ultimately show ?thesis
  using  $\text{countable\_subset}$  by blast

```

qed

lemma metrizable_topology_DD:

```

  assumes topspace (product_topology X I) ≠ {}
    and co: countable {i ∈ I. ∃ a. topspace (X i) ⊆ {a}}
    and m: ∧ i. i ∈ I ⇒ X i = mtopology_of (m i)
  obtains M d where Metric_space M d product_topology X I = Metric_space.mtopology
  M d
    (∧ i. i ∈ I ⇒ mcomplete_of (m i)) ⇒ Metric_space.mcomplete
  M d
  proof (cases I = {})
    case True
      then show ?thesis
        by (metis discrete_metric.mcomplete_discrete_metric discrete_metric.mtopology_discrete_metric
            metric_M_dd product_topology_empty_discrete that)
    next
      case False
        obtain nk and C:: nat set where nk: {i ∈ I. ∃ a. topspace (X i) ⊆ {a}} = nk
        ' C and inj_on nk C
          using co by (force simp: countable_as_injective_image_subset)
        then obtain kn where kn: ∧ w. w ∈ C ⇒ kn (nk w) = w
          by (metis inv_into_f_f)
        define cm where cm ≡ λ i. capped_metric (inverse(Suc(kn i))) (m i)
        have mspace_cm: mspace (cm i) = mspace (m i) for i
          by (simp add: cm_def)
        have c1: ∧ i x y. mdist (cm i) x y ≤ 1
          by (simp add: cm_def capped_metric_md_dist min_le_iff_disj divide_simps)
        then have bdd: bdd_above ((λ i. mdist (cm i) (x i) (y i)) ' I) for x y
          by (meson bdd_above.I2)
        define M where M ≡ Pi_E I (mspace ∘ cm)
        define d where d ≡ λ x y. if x ∈ M ∧ y ∈ M then SUP i ∈ I. mdist (cm i) (x i)
          (y i) else 0

        have d_le1: d x y ≤ 1 for x y
          using ⟨I ≠ {}⟩ c1 by (simp add: d_def bdd cSUP_le_iff)
        with ⟨I ≠ {}⟩ Sup_metric_cartesian_product [of I cm]
        have Metric_space M d
          and *: ∀ x ∈ M. ∀ y ∈ M. ∀ b. (d x y ≤ b) ↔ (∀ i ∈ I. mdist (cm i) (x i) (y i) ≤
          b)
          by (auto simp: False bdd M_def d_def cSUP_le_iff intro: c1)
        then interpret Metric_space M d
          by metis
        have le_d: mdist (cm i) (x i) (y i) ≤ d x y if i ∈ I x ∈ M y ∈ M for i x y
          using * that by blast
        have product_m: Pi_E I (λ i. mspace (m i)) = topspace(product_topology X I)
          using m by force

        define m' where m' = metric (M,d)
        define J where J ≡ λ U. {i ∈ I. U i ≠ topspace (X i)}

```

```

have 1:  $\exists U. \text{finite } (J U) \wedge (\forall i \in I. \text{openin } (X i) (U i)) \wedge x \in \text{Pi}_E I U \wedge \text{Pi}_E I U \subseteq \text{mball } z r$ 
if  $x \in M z \in M$  and  $r: 0 < r d z x < r$  for  $x z r$ 
proof –
  have  $x: \bigwedge i. i \in I \implies x i \in \text{topspace}(X i)$ 
    using  $M\_def\ m\ \text{mspace\_cm}\ \text{that}(1)$  by auto
  have  $z: \bigwedge i. i \in I \implies z i \in \text{topspace}(X i)$ 
    using  $M\_def\ m\ \text{mspace\_cm}\ \text{that}(2)$  by auto
  obtain  $R$  where  $0 < R d z x < R R < r$ 
    using  $r\ \text{dense}$  by (smt (verit, ccfv_threshold))
  define  $U$  where  $U \equiv \lambda i. \text{if } R \leq \text{inverse}(\text{Suc}(kn\ i)) \text{ then } \text{mball\_of } (m\ i) (z\ i)$ 
  R else topspace(X i)
  show ?thesis
  proof (intro exI conjI)
    obtain  $n$  where  $n: \text{real } n * R > 1$ 
      using  $\langle 0 < R \rangle\ \text{ex\_less\_of\_nat\_mult}$  by blast
    have  $\text{finite } (nk\ ' (C \cap \{..n\}))$ 
      by force
    moreover
      have  $\exists m. m \in C \wedge m \leq n \wedge i = nk\ m$ 
      if  $R: R \leq \text{inverse } (1 + \text{real } (kn\ i))$  and  $i \in I$ 
      and  $\text{neq}: \text{mball\_of } (m\ i) (z\ i) R \neq \text{topspace } (X\ i)$  for  $i$ 
    proof –
      interpret  $MI: \text{Metric\_space } \text{mspace } (m\ i)\ \text{mdist } (m\ i)$ 
      by auto
      have  $MI.\text{mball } (z\ i) R \neq \text{topspace } (X\ i)$ 
      by (metis mball_of_def neq)
      then have  $\nexists a. \text{topspace } (X\ i) \subseteq \{a\}$ 
      using  $\langle 0 < R \rangle\ m\ \text{subset\_antisym } \langle i \in I \rangle\ z$  by fastforce
      then have  $i \in nk\ ' C$ 
      using  $nk\ \langle i \in I \rangle$  by auto
      then show ?thesis
      by (smt (verit, ccfv_SIG) R \langle 0 < R \rangle image_iff kn lift_Suc_mono_less_iff mult_mono n not_le_imp_less_of_nat_0_le_iff_of_nat_Suc_right_inverse)
    qed
    then have  $J U \subseteq nk\ ' (C \cap \{..n\})$ 
      by (auto simp: image_iff Bex_def J_def U_def split: if_split_asm)
    ultimately show  $\text{finite } (J U)$ 
      using  $\text{finite\_subset}$  by blast
    show  $\forall i \in I. \text{openin } (X\ i) (U\ i)$ 
      by (simp add: Metric_space.openin_mball U_def mball_of_def mtopology_of_def m)
    have  $xin: x \in \text{Pi}_E I (\text{topspace} \circ X)$ 
      using  $M\_def\ \langle x \in M \rangle\ x$  by auto
    moreover
      have  $\bigwedge i. \llbracket i \in I; R \leq \text{inverse } (1 + \text{real } (kn\ i)) \rrbracket \implies \text{mdist } (m\ i) (z\ i) (x\ i) < R$ 
      by (smt (verit, ccfv_SIG) \langle d z x < R \rangle capped_metric_mdists cm_def le_d_of_nat_Suc that)

```

```

ultimately show  $x \in Pi_E I U$ 
  using  $m z$  by (auto simp:  $U\_def Pi_E\_iff$ )
show  $Pi_E I U \subseteq mball z r$ 
proof
  fix  $y$ 
  assume  $y: y \in Pi_E I U$ 
  then have  $y \in M$ 
    by (force simp:  $Pi_E\_iff M\_def U\_def m mspace\_cm split: if\_split\_asm$ )
  moreover
  have  $\forall i \in I. mdist (cm i) (z i) (y i) \leq R$ 
    by (smt (verit)  $Pi_E\_mem U\_def cm\_def in\_mball\_of inverse\_Suc$ 
 $mdist\_capped mdist\_capped\_le y$ )
  then have  $d z y \leq R$ 
    by (simp add:  $\langle y \in M \rangle \langle z \in M \rangle *$ )
  ultimately show  $y \in mball z r$ 
    using  $\langle R < r \rangle \langle z \in M \rangle$  by force
qed
qed
qed
have  $\exists r > 0. mball x r \subseteq S$ 
  if finite  $(J U)$  and  $x: x \in Pi_E I U$  and  $S: Pi_E I U \subseteq S$ 
  and  $U: \bigwedge i. i \in I \implies openin (X i) (U i)$ 
  and  $x \in S$  for  $U S x$ 
proof -
  { fix  $i$ 
    assume  $i \in J U$ 
    then have  $i \in I$ 
      by (auto simp:  $J\_def$ )
    then have  $openin (mtopology\_of (m i)) (U i)$ 
      using  $U m$  by force
    then have  $openin (mtopology\_of (cm i)) (U i)$ 
      by (simp add:  $Abstract\_Metric\_Spaces.mtopology\_capped\_metric cm\_def$ )
    then have  $\exists r > 0. mball\_of (cm i) (x i) r \subseteq U i$ 
      using  $x$ 
    by (simp add:  $Metric\_space.openin\_mtopology Pi_E\_mem \langle i \in I \rangle mball\_of\_def$ 
 $mtopology\_of\_def$ )
  }
  then obtain  $rf$  where  $rf: \bigwedge j. j \in J U \implies rf j > 0 \wedge mball\_of (cm j) (x j)$ 
 $(rf j) \subseteq U j$ 
    by metis
  define  $r$  where  $r \equiv Min (insert 1 (rf ' J U))$ 
  show ?thesis
proof (intro exI conjI)
  show  $r > 0$ 
    by (simp add:  $\langle finite (J U) \rangle r\_def rf$ )
  have  $r [simp]: \bigwedge j. j \in J U \implies r \leq rf j r \leq 1$ 
    by (auto simp:  $r\_def that(1)$ )
  have *:  $mball\_of (cm i) (x i) r \subseteq U i$  if  $i \in I$  for  $i$ 
  proof (cases  $i \in J U$ )

```

```

    case True
    with r show ?thesis
      by (smt (verit) Metric_space.in_mball Metric_space_mspace_mdistr
mball_of_def rf subset_eq)
    next
    case False
    then show ?thesis
      by (simp add: J_def cm_def m_subset_eq that)
  qed
  show mball x r  $\subseteq$  S
    by (smt (verit) x * in_mball_of M_def Metric_space.in_mball Metric_
space_axioms PiE_iff le_d o_apply subset_eq S)
  qed
  qed
  have  $\exists x \in M$ 
    if  $\S: \bigwedge x. x \in S \implies \exists U. \text{finite } (J U) \wedge (\forall i \in I. \text{openin } (X i) (U i)) \wedge x \in \text{Pi}_E I U \wedge \text{Pi}_E I U \subseteq S$ 
    and  $x \in S$  for  $S x$ 
    using  $\S [OF \langle x \in S \rangle] m\_openin\_subset$  by (fastforce simp: M_def PiE_iff
cm_def)
  show thesis
  proof
    show Metric_space M d
      using Metric_space_axioms by blast
    show eq: product_topology X I = Metric_space.mtopology M d
      unfolding topology_eq openin_mtopology openin_product_topology_alt
      using J_def 1 2 3 subset_iff zero by (smt (verit, ccfv_threshold))
    show mcomplete if  $\bigwedge i. i \in I \implies mcomplete\_of (m i)$ 
      unfolding mcomplete_def
    proof (intro strip)
      fix  $\sigma$ 
      assume  $\sigma: M\text{Cauchy } \sigma$ 
      have  $\exists y. i \in I \longrightarrow \text{limitin } (X i) (\lambda n. \sigma n i) y$  sequentially for  $i$ 
      proof (cases  $i \in I$ )
      case True
      interpret MI: Metric_space mspace (m i) mdist (m i)
        by auto
      have  $\bigwedge \sigma. MI.M\text{Cauchy } \sigma \longrightarrow (\exists x. \text{limitin } MI.mtopology \sigma x \text{ sequentially})$ 
        by (meson MI.mcomplete_def True mcomplete_of_def that)
      moreover have MI.MCauchy  $(\lambda n. \sigma n i)$ 
      unfolding MI.MCauchy_def
    proof (intro conjI strip)
      show range  $(\lambda n. \sigma n i) \subseteq mspace (m i)$ 
        by (smt (verit, ccfv_threshold) MCauchy_def PiE_iff True  $\sigma$  eq im-
age_subset_iff m_topspace_mtopology topspace_mtopology_of topspace_product_topology)
      fix  $\varepsilon::\text{real}$ 
      define r where  $r \equiv \min \varepsilon (\text{inverse}(\text{Suc } (kn i)))$ 
      assume  $\varepsilon > 0$ 
      then have  $r > 0$ 

```

```

    by (simp add: r_def)
  then obtain N where N:  $\bigwedge n n'. N \leq n \wedge N \leq n' \implies d (\sigma n) (\sigma n') < r$ 
r
    using  $\sigma$  unfolding MCauchy_def by meson
  show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow \text{mdist } (m i) (\sigma n i) (\sigma n' i) < \varepsilon$ 
  proof (intro strip exI)
    fix n n'
    assume  $N \leq n$  and  $N \leq n'$ 
    then have  $\text{mdist } (cm i) (\sigma n i) (\sigma n' i) < r$ 
    using *
    by (smt (verit) Metric_space.MCauchy_def Metric_space_axioms N
      True  $\sigma$  rangeI subsetD)
    then
    show  $\text{mdist } (m i) (\sigma n i) (\sigma n' i) < \varepsilon$ 
    unfolding cm_def r_def
    by (smt (verit, ccfv_SIG) capped_metric_mdists)
  qed
qed
ultimately show ?thesis
  by (simp add: m_mtopology_of_def)
qed auto
then obtain y where  $\bigwedge i. i \in I \implies \text{limitin } (X i) (\lambda n. \sigma n i) (y i)$  sequentially
  by metis
with  $\sigma$  show  $\exists x. \text{limitin mtopology } \sigma x$  sequentially
  apply (rule_tac x= $\lambda i \in I. y i$  in exI)
  apply (simp add: MCauchy_def limitin_componentwise_flip: eq)
  by (metis eq eventually_at_top_linorder range_subsetD topspace_mtopology
    topspace_product_topology)
  qed
  qed
qed

```

lemma metrizable_topology_D:

```

  assumes topspace (product_topology X I)  $\neq \{\}$ 
    and co: countable  $\{i \in I. \nexists a. \text{topspace } (X i) \subseteq \{a\}\}$ 
    and met:  $\bigwedge i. i \in I \implies \text{metrizable\_space } (X i)$ 
  shows metrizable_space (product_topology X I)
proof -
  have  $\bigwedge i. i \in I \implies \exists m. X i = \text{mtopology\_of } m$ 
  by (metis Metric_space.mtopology_of met metrizable_space_def)
  then obtain m where m:  $\bigwedge i. i \in I \implies X i = \text{mtopology\_of } (m i)$ 
  by metis
  then show ?thesis
    using metrizable_topology_DD [of X I m] assms by (force simp: metrizable_space_def)
  qed

```

lemma metrizable_topology_E:

```

assumes topspace (product_topology X I) ≠ {}
and countable {i ∈ I. ∄ a. topspace (X i) ⊆ {a}}
and met: ∧i. i ∈ I ⇒ completely_metrizable_space (X i)
shows completely_metrizable_space (product_topology X I)
proof –
have ∧i. i ∈ I ⇒ ∃ m. mcomplete_of m ∧ X i = mtopology_of m
using met Metric_space.mtopology_of Metric_space.mcomplete_of unfolding
completely_metrizable_space_def
by metis
then obtain m where ∧i. i ∈ I ⇒ mcomplete_of (m i) ∧ X i = mtopology_of
(m i)
by metis
then show ?thesis
using metrizable_topology_DD [of X I m] assms unfolding metrizable_space_def
by (metis (full_types) completely_metrizable_space_def)
qed

```

proposition *metrizable_space_product_topology*:

```

metrizable_space (product_topology X I) ↔
  (product_topology X I) = trivial_topology ∨
  countable {i ∈ I. ¬ (∃ a. topspace(X i) ⊆ {a})} ∧
  (∀ i ∈ I. metrizable_space (X i))
by (metis (mono_tags, lifting) empty_metrizable_space metrizable_topology_A
metrizable_topology_B metrizable_topology_D subtopology_eq_discrete_topology_empty)

```

proposition *completely_metrizable_space_product_topology*:

```

completely_metrizable_space (product_topology X I) ↔
  (product_topology X I) = trivial_topology ∨
  countable {i ∈ I. ¬ (∃ a. topspace(X i) ⊆ {a})} ∧
  (∀ i ∈ I. completely_metrizable_space (X i))
by (smt (verit, del_insts) Collect_cong completely_metrizable_imp_metrizable_space
empty_completely_metrizable_space metrizable_topology_B metrizable_topology_C
metrizable_topology_E subtopology_eq_discrete_topology_empty)

```

lemma *completely_metrizable_Euclidean_space*:

```

completely_metrizable_space(Euclidean_space n)
unfolding Euclidean_space_def
proof (rule completely_metrizable_space_closedin)
show completely_metrizable_space (powertop_real (UNIV::nat set))
by (simp add: completely_metrizable_space_product_topology completely_metrizable_space_euclidean)
show closedin (powertop_real UNIV) {x. ∀ i ≥ n. x i = 0}
using closedin_Euclidean_space topspace_Euclidean_space by auto
qed

```

lemma *metrizable_Euclidean_space*:

```

metrizable_space(Euclidean_space n)
by (simp add: completely_metrizable_Euclidean_space completely_metrizable_imp_metrizable_space)

```

lemma *locally_connected_Euclidean_space*:

locally_connected_space(*Euclidean_space* n)

by (*simp* add: *locally_path_connected_Euclidean_space* *locally_path_connected_imp_locally_connected*)

end

6.8 Infinite sums

In this theory, we introduce the definition of infinite sums, i.e., sums ranging over an infinite, potentially uncountable index set with no particular ordering. (This is different from series. Those are sums indexed by natural numbers, and the order of the index set matters.)

Our definition is quite standard: $s := \sum_{x \in A} f(x)$ is the limit of finite sums $s_F := \sum_{x \in F} f(x)$ for increasing F . That is, s is the limit of the net s_F where F are finite subsets of A ordered by inclusion. We believe that this is the standard definition for such sums. See, e.g., Definition 4.11 in [2]. This definition is quite general: it is well-defined whenever f takes values in some commutative monoid endowed with a Hausdorff topology. (Examples are reals, complex numbers, normed vector spaces, and more.)

theory *Infinite_Sum*

imports

Elementary_Topology

HOL-Library.Extended_Nonnegative_Real

HOL-Library.Complex_Order

begin

6.8.1 Definition and syntax

definition *HAS_SUM* :: $\langle 'a \Rightarrow 'b :: \{comm_monoid_add, topological_space\} \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow bool \rangle$

where *has_sum_def*: $\langle HAS_SUM\ f\ A\ x \equiv (sum\ f \longrightarrow x)\ (finite_subsets_at_top\ A) \rangle$

abbreviation *has_sum* (**infixr** *has'_sum* 46) **where**

$(f\ has_sum\ S)\ A \equiv HAS_SUM\ f\ A\ S$

definition *summable_on* :: $\langle 'a \Rightarrow 'b :: \{comm_monoid_add, topological_space\} \Rightarrow 'a\ set \Rightarrow bool$ (**infixr** *summable'_on* 46) **where**

$f\ summable_on\ A \equiv (\exists x. (f\ has_sum\ x)\ A)$

definition *infsum* :: $\langle 'a \Rightarrow 'b :: \{comm_monoid_add, t2_space\} \Rightarrow 'a\ set \Rightarrow 'b$ **where**

$infsum\ f\ A = (if\ f\ summable_on\ A\ then\ Lim\ (finite_subsets_at_top\ A)\ (sum\ f)\ else\ 0)$

abbreviation *abs_summable_on* :: $\langle 'a \Rightarrow 'b :: real_normed_vector \rangle \Rightarrow 'a\ set \Rightarrow bool$ (**infixr** *abs'_summable'_on* 46) **where**

$f \text{ abs_summable_on } A \equiv (\lambda x. \text{ norm } (f x)) \text{ summable_on } A$

syntax (ASCII)

$_ \text{ infsum} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b::\text{topological_comm_monoid_add} \ ((\exists \text{INFSUM } _ / _)) [0, 51, 10] 10)$

syntax

$_ \text{ infsum} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b::\text{topological_comm_monoid_add} \ ((\exists \sum_{\infty} (_ / _)) / _)) [0, 51, 10] 10)$

translations — Beware of argument permutation!

$\sum_{\infty} i \in A. b \equiv \text{CONST infsum } (\lambda i. b) A$

syntax (ASCII)

$_ \text{ univinfsum} :: \text{pttrn} \Rightarrow 'a \Rightarrow 'a \ ((\exists \text{INFSUM } _ / _)) [0, 10] 10)$

syntax

$_ \text{ univinfsum} :: \text{pttrn} \Rightarrow 'a \Rightarrow 'a \ ((\exists \sum_{\infty} _ / _)) [0, 10] 10)$

translations

$\sum_{\infty} x. t \equiv \text{CONST infsum } (\lambda x. t) (\text{CONST UNIV})$

syntax (ASCII)

$_ \text{ qinfsum} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((\exists \text{INFSUM } _ \mid _ / _)) [0, 0, 10] 10)$

syntax

$_ \text{ qinfsum} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a \ ((\exists \sum_{\infty} _ \mid (_) / _)) [0, 0, 10] 10)$

translations

$\sum_{\infty} x \mid P. t \Rightarrow \text{CONST infsum } (\lambda x. t) \{x. P\}$

print_translation <

let

$\text{fun sum_tr}' [Abs (x, Tx, t), Const (@\{\text{const_syntax Collect}\}, _)] \$ Abs (y, Ty, P)] =$

if $x <> y$ then raise Match

else

let

val $x' = \text{Syntax_Trans.mark_bound_body } (x, Tx);$

val $t' = \text{subst_bound } (x', t);$

val $P' = \text{subst_bound } (x', P);$

in

$\text{Syntax.const } @\{\text{syntax_const_qinfsum}\} \$ \text{Syntax_Trans.mark_bound_abs}$

$(x, Tx) \$ P' \$ t'$

end

| $\text{sum_tr}' _ = \text{raise Match};$

in $[\text{@}\{\text{const_syntax infsum}\}, K \text{ sum_tr}'] \text{ end}$

>

6.8.2 General properties

lemma *infsumI*:

fixes $f g :: 'a \Rightarrow 'b::\{\text{comm_monoid_add}, t2_space\}$

assumes $\langle f \text{ has_sum } x \rangle A$

shows $\langle \text{infsum } f A = x \rangle$

by (metis assms finite_subsets_at_top_neq_bot infsum_def summable_on_def has_sum_def tendsto_Lim)

lemma *infsum_eqI*:
 fixes $f\ g :: \langle 'a \Rightarrow 'b :: \{comm_monoid_add, t2_space\} \rangle$
 assumes $\langle x = y \rangle$
 assumes $\langle (f\ has_sum\ x)\ A \rangle$
 assumes $\langle (g\ has_sum\ y)\ B \rangle$
 shows $\langle infsum\ f\ A = infsum\ g\ B \rangle$
 using *assms infsumI* by blast

lemma *infsum_eqI'*:
 fixes $f\ g :: \langle 'a \Rightarrow 'b :: \{comm_monoid_add, t2_space\} \rangle$
 assumes $\langle \bigwedge x. (f\ has_sum\ x)\ A \longleftrightarrow (g\ has_sum\ x)\ B \rangle$
 shows $\langle infsum\ f\ A = infsum\ g\ B \rangle$
 by (metis assms infsum_def infsum_eqI summable_on_def)

lemma *infsum_not_exists*:
 fixes $f :: \langle 'a \Rightarrow 'b :: \{comm_monoid_add, t2_space\} \rangle$
 assumes $\langle \neg f\ summable_on\ A \rangle$
 shows $\langle infsum\ f\ A = 0 \rangle$
 by (simp add: assms infsum_def)

lemma *summable_iff_has_sum_infsum*: $f\ summable_on\ A \longleftrightarrow (f\ has_sum\ (infsum\ f\ A))\ A$
 using *infsumI summable_on_def* by blast

lemma *has_sum_infsum[simp]*:
 assumes $\langle f\ summable_on\ S \rangle$
 shows $\langle (f\ has_sum\ (infsum\ f\ S))\ S \rangle$
 using *assms summable_iff_has_sum_infsum* by blast

lemma *has_sum_cong_neutral*:
 fixes $f\ g :: \langle 'a \Rightarrow 'b :: \{comm_monoid_add, topological_space\} \rangle$
 assumes $\langle \bigwedge x. x \in T - S \implies g\ x = 0 \rangle$
 assumes $\langle \bigwedge x. x \in S - T \implies f\ x = 0 \rangle$
 assumes $\langle \bigwedge x. x \in S \cap T \implies f\ x = g\ x \rangle$
 shows $\langle (f\ has_sum\ x)\ S \longleftrightarrow (g\ has_sum\ x)\ T \rangle$

proof –

have $\langle eventually\ P\ (filtermap\ (sum\ f)\ (finite_subsets_at_top\ S)) \rangle$
 $= eventually\ P\ (filtermap\ (sum\ g)\ (finite_subsets_at_top\ T)) \rangle$ for P

proof

assume $\langle eventually\ P\ (filtermap\ (sum\ f)\ (finite_subsets_at_top\ S)) \rangle$

then obtain $F0$ where $\langle finite\ F0 \rangle$ and $\langle F0 \subseteq S \rangle$ and $F0_P: \langle \bigwedge F. finite\ F$

$\implies F \subseteq S \implies F \supseteq F0 \implies P\ (sum\ f\ F) \rangle$

by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)

define $F0'$ where $\langle F0' = F0 \cap T \rangle$

have [simp]: $\langle finite\ F0' \rangle \langle F0' \subseteq T \rangle$

by (simp_all add: F0'_def $\langle finite\ F0 \rangle$)

```

have ⟨P (sum g F)⟩ if ⟨finite F⟩ ⟨F ⊆ T⟩ ⟨F ⊇ F0'⟩ for F
proof -
  have ⟨P (sum f ((F∩S) ∪ (F0∩S)))⟩
    by (intro F0_P) (use ⟨F0 ⊆ S⟩ ⟨finite F0⟩ that in auto)
  also have ⟨sum f ((F∩S) ∪ (F0∩S)) = sum g F⟩
    by (intro sum.mono_neutral_cong) (use that ⟨finite F0⟩ F0'_def assms in
auto)
  finally show ?thesis .
qed
with ⟨F0' ⊆ T⟩ ⟨finite F0'⟩ show ⟨eventually P (filtermap (sum g) (finite_subsets_at_top
T))⟩
  by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)
next
  assume ⟨eventually P (filtermap (sum g) (finite_subsets_at_top T))⟩
  then obtain F0 where ⟨finite F0⟩ and ⟨F0 ⊆ T⟩ and F0_P: ⟨∧F. finite F
⇒ F ⊆ T ⇒ F ⊇ F0 ⇒ P (sum g F)⟩
    by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)
  define F0' where ⟨F0' = F0 ∩ S⟩
  have [simp]: ⟨finite F0'⟩ ⟨F0' ⊆ S⟩
    by (simp_all add: F0'_def ⟨finite F0⟩)
  have ⟨P (sum f F)⟩ if ⟨finite F⟩ ⟨F ⊆ S⟩ ⟨F ⊇ F0'⟩ for F
  proof -
    have ⟨P (sum g ((F∩T) ∪ (F0∩T)))⟩
      by (intro F0_P) (use ⟨F0 ⊆ T⟩ ⟨finite F0⟩ that in auto)
    also have ⟨sum g ((F∩T) ∪ (F0∩T)) = sum f F⟩
      by (intro sum.mono_neutral_cong) (use that ⟨finite F0⟩ F0'_def assms in
auto)
    finally show ?thesis .
  qed
  with ⟨F0' ⊆ S⟩ ⟨finite F0'⟩ show ⟨eventually P (filtermap (sum f) (finite_subsets_at_top
S))⟩
    by (metis (no_types, lifting) eventually_filtermap eventually_finite_subsets_at_top)
  qed

  then have tendsto_x: (sum f ⟶ x) (finite_subsets_at_top S) ⟷ (sum g
⟶ x) (finite_subsets_at_top T) for x
    by (simp add: le_filter_def filterlim_def)

  then show ?thesis
    by (simp add: has_sum_def)
qed

lemma summable_on_cong_neutral:
  fixes f g :: 'a ⇒ 'b::{comm_monoid_add, topological_space}
  assumes ⟨∧x. x∈T-S ⇒ g x = 0⟩
  assumes ⟨∧x. x∈S-T ⇒ f x = 0⟩
  assumes ⟨∧x. x∈S∩T ⇒ f x = g x⟩
  shows f summable_on S ⟷ g summable_on T
  using has_sum_cong_neutral[of T S g f, OF assms]

```

by (*simp add: summable_on_def*)

lemma *infsum_cong_neutral*:

fixes $f g :: 'a \Rightarrow 'b :: \{comm_monoid_add, t2_space\}$

assumes $\langle \bigwedge x. x \in T - S \implies g x = 0 \rangle$

assumes $\langle \bigwedge x. x \in S - T \implies f x = 0 \rangle$

assumes $\langle \bigwedge x. x \in S \cap T \implies f x = g x \rangle$

shows $\langle infsum f S = infsum g T \rangle$

by (*smt (verit, best) assms has_sum_cong_neutral infsum_eqI'*)

lemma *has_sum_cong*:

assumes $\bigwedge x. x \in A \implies f x = g x$

shows $(f \text{ has_sum } x) A \longleftrightarrow (g \text{ has_sum } x) A$

using *assms* **by** (*intro has_sum_cong_neutral*) *auto*

lemma *summable_on_cong*:

assumes $\bigwedge x. x \in A \implies f x = g x$

shows $f \text{ summable_on } A \longleftrightarrow g \text{ summable_on } A$

by (*metis assms summable_on_def has_sum_cong*)

lemma *infsum_cong*:

assumes $\bigwedge x. x \in A \implies f x = g x$

shows $infsum f A = infsum g A$

using *assms infsum_eqI' has_sum_cong* **by** *blast*

lemma *summable_on_cofin_subset*:

fixes $f :: 'a \Rightarrow 'b :: \text{topological_ab_group_add}$

assumes $f \text{ summable_on } A$ **and** [*simp*]: *finite F*

shows $f \text{ summable_on } (A - F)$

proof –

from *assms(1)* **obtain** x **where** $\lim f: (sum f \longrightarrow x)$ (*finite_subsets_at_top A*)

unfolding *summable_on_def has_sum_def* **by** *auto*

define F' **where** $F' = F \cap A$

with *assms* **have** *finite F'* **and** $A - F = A - F'$

by *auto*

have *filtermap* $((\cup) F')$ (*finite_subsets_at_top (A - F)*)

\leq *finite_subsets_at_top A*

proof (*rule filter_leI*)

fix P **assume** *eventually P (finite_subsets_at_top A)*

then obtain X **where** [*simp*]: *finite X* **and** $XA: X \subseteq A$

and $P: \forall Y. \text{finite } Y \wedge X \subseteq Y \wedge Y \subseteq A \longrightarrow P Y$

unfolding *eventually_finite_subsets_at_top* **by** *auto*

define X' **where** $X' = X - F$

hence [*simp*]: *finite X'* **and** [*simp*]: $X' \subseteq A - F$

using XA **by** *auto*

hence *finite Y* $\wedge X' \subseteq Y \wedge Y \subseteq A - F \longrightarrow P (F' \cup Y)$ **for** Y

using $P XA$ **unfolding** X'_def **using** F'_def $\langle \text{finite } F' \rangle$ **by** *blast*

thus *eventually P (filtermap ((\cup) F') (finite_subsets_at_top (A - F)))*

```

unfolding eventually_filtermap eventually_finite_subsets_at_top
by (rule_tac x=X' in exI, simp)
qed
with lim_f have (sum f  $\longrightarrow$  x) (filtermap (( $\cup$ )F') (finite_subsets_at_top
(A-F)))
using tendsto_mono by blast
have (( $\lambda$ G. sum f (F'  $\cup$  G))  $\longrightarrow$  x) (finite_subsets_at_top (A - F))
if ((sum f  $\circ$  ( $\cup$ ) F')  $\longrightarrow$  x) (finite_subsets_at_top (A - F))
using that unfolding o_def by auto
hence (( $\lambda$ G. sum f (F'  $\cup$  G))  $\longrightarrow$  x) (finite_subsets_at_top (A-F))
using tendsto_compose_filtermap [symmetric]
by (simp add:  $\langle$ (sum f  $\longrightarrow$  x) (filtermap (( $\cup$ ) F') (finite_subsets_at_top (A
- F))) $\rangle$ 
tendsto_compose_filtermap)
have  $\forall Y. \text{finite } Y \wedge Y \subseteq A - F \longrightarrow \text{sum } f (F' \cup Y) = \text{sum } f F' + \text{sum } f Y$ 
by (metis Diff_disjoint Int_Diff  $\langle$ A - F = A - F' $\rangle$   $\langle$ finite F' $\rangle$  inf.orderE
sum.union_disjoint)
hence  $\forall_F x \text{ in } \text{finite\_subsets\_at\_top } (A - F). \text{sum } f (F' \cup x) = \text{sum } f F' +$ 
sum f x
unfolding eventually_finite_subsets_at_top
using exI [where x = {}]
by (simp add:  $\langle$  $\bigwedge P. P \{ \} \implies \exists x. P x$  $\rangle$ )
hence (( $\lambda$ G. sum f F' + sum f G)  $\longrightarrow$  x) (finite_subsets_at_top (A-F))
using tendsto_cong [THEN iffD1, rotated]
 $\langle$ ( $\lambda$ G. sum f (F'  $\cup$  G))  $\longrightarrow$  x) (finite_subsets_at_top (A - F)) $\rangle$  by
fastforce
hence (( $\lambda$ G. sum f F' + sum f G)  $\longrightarrow$  sum f F' + (x - sum f F')) (finite_subsets_at_top
(A-F))
by simp
hence (sum f  $\longrightarrow$  x - sum f F') (finite_subsets_at_top (A-F))
using tendsto_add_const_iff by blast
thus f summable_on (A - F)
unfolding summable_on_def has_sum_def by auto
qed

```

lemma

```

fixes f :: 'a  $\Rightarrow$  'b::{topological_ab_group_add}
assumes  $\langle$ (f has_sum b) B $\rangle$  and  $\langle$ (f has_sum a) A $\rangle$  and AB: A  $\subseteq$  B
shows has_sum_Diff: (f has_sum (b - a)) (B - A)

```

proof -

```

have finite_subsets1:

```

```

finite_subsets_at_top (B - A)  $\leq$  filtermap ( $\lambda$ F. F - A) (finite_subsets_at_top
B)

```

```

proof (rule filter_leI)

```

```

fix P assume eventually P (filtermap ( $\lambda$ F. F - A) (finite_subsets_at_top B))

```

```

then obtain X where finite X and X  $\subseteq$  B

```

```

and P: finite Y  $\wedge$  X  $\subseteq$  Y  $\wedge$  Y  $\subseteq$  B  $\longrightarrow$  P (Y - A) for Y

```

```

unfolding eventually_filtermap eventually_finite_subsets_at_top by auto

```

hence $\text{finite } (X-A)$ **and** $X-A \subseteq B - A$
by *auto*
moreover **have** $\text{finite } Y \wedge X-A \subseteq Y \wedge Y \subseteq B - A \longrightarrow P Y$ **for** Y
using $P[\text{where } Y=Y \cup X] \langle \text{finite } X \rangle \langle X \subseteq B \rangle$
by (*metis Diff_subset Int_Diff Un_Diff finite_Un inf.orderE le_sup_iff sup.orderE sup_ge2*)
ultimately **show** *eventually* $P (\text{finite_subsets_at_top } (B - A))$
unfolding *eventually_finite_subsets_at_top* **by** *meson*
qed
have *finite_subsets2*:
filtermap $(\lambda F. F \cap A) (\text{finite_subsets_at_top } B) \leq \text{finite_subsets_at_top } A$
apply (*rule filter_leI*)
using *assms* **unfolding** *eventually_filtermap eventually_finite_subsets_at_top*
by (*metis Int_subset_iff finite_Int inf_le2 subset_trans*)

from *assms*(1) **have** $\text{lim}B: (\text{sum } f \longrightarrow b) (\text{finite_subsets_at_top } B)$
using *has_sum_def* **by** *auto*
from *assms*(2) **have** $\text{lim}A: (\text{sum } f \longrightarrow a) (\text{finite_subsets_at_top } A)$
using *has_sum_def* **by** *blast*
have $(\lambda F. \text{sum } f (F \cap A)) \longrightarrow a (\text{finite_subsets_at_top } B)$
proof (*subst asm_rl [of (\lambda F. sum f (F \cap A)) = sum f \circ (\lambda F. F \cap A)]*)
show $(\lambda F. \text{sum } f (F \cap A)) = \text{sum } f \circ (\lambda F. F \cap A)$
unfolding *o_def* **by** *auto*
show $(\text{sum } f \circ (\lambda F. F \cap A)) \longrightarrow a (\text{finite_subsets_at_top } B)$
unfolding *o_def*
using *tendsto_compose_filtermap finite_subsets2 limA tendsto_mono*
 $\langle (\lambda F. \text{sum } f (F \cap A)) = \text{sum } f \circ (\lambda F. F \cap A) \rangle$ **by** *fastforce*
qed

with $\text{lim}B$ **have** $(\lambda F. \text{sum } f F - \text{sum } f (F \cap A)) \longrightarrow b - a (\text{finite_subsets_at_top } B)$
using *tendsto_diff* **by** *blast*
have $\text{sum } f X - \text{sum } f (X \cap A) = \text{sum } f (X - A)$ **if** *finite* X **and** $X \subseteq B$ **for**
 $X :: 'a \text{ set}$
using *that* **by** (*metis add_diff_cancel_left' sum.Int_Diff*)
hence $\forall_F x \text{ in } \text{finite_subsets_at_top } B. \text{sum } f x - \text{sum } f (x \cap A) = \text{sum } f (x - A)$
by (*rule eventually_finite_subsets_at_top_weakI*)
hence $(\lambda F. \text{sum } f (F - A)) \longrightarrow b - a (\text{finite_subsets_at_top } B)$
using *tendsto_cong [THEN iffD1, rotated]*
 $\langle (\lambda F. \text{sum } f F - \text{sum } f (F \cap A)) \longrightarrow b - a (\text{finite_subsets_at_top } B) \rangle$
by *fastforce*
hence $(\text{sum } f \longrightarrow b - a) (\text{filtermap } (\lambda F. F - A) (\text{finite_subsets_at_top } B))$
by (*subst tendsto_compose_filtermap[symmetric], simp add: o_def*)
thus *?thesis*
using *finite_subsets1 has_sum_def tendsto_mono* **by** *blast*
qed

lemma

fixes $f :: 'a \Rightarrow 'b::\{\text{topological_ab_group_add}\}$
 assumes f summable_on B and f summable_on A and $A \subseteq B$
 shows summable_on_Diff: f summable_on $(B-A)$
 by (meson assms summable_on_def has_sum_Diff)

lemma

fixes $f :: 'a \Rightarrow 'b::\{\text{topological_ab_group_add}, t2_space\}$
 assumes f summable_on B and f summable_on A and $AB: A \subseteq B$
 shows infsum_Diff: $\text{infsum } f (B - A) = \text{infsum } f B - \text{infsum } f A$
 by (metis AB assms has_sum_Diff infsumI summable_on_def)

lemma has_sum_mono_neutral:

fixes $f :: 'a \Rightarrow 'b::\{\text{ordered_comm_monoid_add}, \text{linorder_topology}\}$

assumes $\langle f \text{ has_sum } a \rangle A$ and $\langle g \text{ has_sum } b \rangle B$
 assumes $\langle \bigwedge x. x \in A \cap B \implies f x \leq g x \rangle$
 assumes $\langle \bigwedge x. x \in A - B \implies f x \leq 0 \rangle$
 assumes $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$
 shows $a \leq b$

proof -

define $f' g'$ where $\langle f' x = (\text{if } x \in A \text{ then } f x \text{ else } 0) \rangle$ and $\langle g' x = (\text{if } x \in B \text{ then } g x \text{ else } 0) \rangle$ for x

have [simp]: $\langle f \text{ summable_on } A \rangle \langle g \text{ summable_on } B \rangle$

using assms(1,2) summable_on_def by auto

have $\langle f' \text{ has_sum } a \rangle (A \cup B)$

by (smt (verit, best) DiffE IntE Un_iff f'_def assms(1) has_sum_cong_neutral)

then have $f' \text{ lim: } \langle (\text{sum } f' \longrightarrow a) (\text{finite_subsets_at_top } (A \cup B)) \rangle$

by (meson has_sum_def)

have $\langle g' \text{ has_sum } b \rangle (A \cup B)$

by (smt (verit, best) DiffD1 DiffD2 IntE UnCI g'_def assms(2) has_sum_cong_neutral)

then have $g' \text{ lim: } \langle (\text{sum } g' \longrightarrow b) (\text{finite_subsets_at_top } (A \cup B)) \rangle$

using has_sum_def by blast

have $\bigwedge X i. \llbracket X \subseteq A \cup B; i \in X \rrbracket \implies f' i \leq g' i$

using assms by (auto simp: f'_def g'_def)

then have $\langle \forall_F x \text{ in } \text{finite_subsets_at_top } (A \cup B). \text{sum } f' x \leq \text{sum } g' x \rangle$

by (intro eventually_finite_subsets_at_top_weakI sum_mono)

then show ?thesis

using f'_lim finite_subsets_at_top_neq_bot g'_lim tendsto_le by blast

qed

lemma infsum_mono_neutral:

fixes $f :: 'a \Rightarrow 'b::\{\text{ordered_comm_monoid_add}, \text{linorder_topology}\}$

assumes f summable_on A and g summable_on B

assumes $\langle \bigwedge x. x \in A \cap B \implies f x \leq g x \rangle$

assumes $\langle \bigwedge x. x \in A - B \implies f x \leq 0 \rangle$

assumes $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$

shows $\text{infsum } f A \leq \text{infsum } g B$

by (smt (verit, best) assms has_sum_infsun has_sum_mono_neutral)

lemma has_sum_mono:

fixes f :: 'a ⇒ 'b :: {ordered_comm_monoid_add, linorder_topology}
 assumes (f has_sum x) A and (g has_sum y) A
 assumes ⟨ $\bigwedge x. x \in A \implies f x \leq g x$ ⟩
 shows $x \leq y$
 using assms has_sum_mono_neutral by force

lemma infsum_mono:

fixes f :: 'a ⇒ 'b :: {ordered_comm_monoid_add, linorder_topology}
 assumes f summable_on A and g summable_on A
 assumes ⟨ $\bigwedge x. x \in A \implies f x \leq g x$ ⟩
 shows $\text{infsum } f A \leq \text{infsum } g A$
 by (meson assms has_sum_infsun has_sum_mono)

lemma has_sum_finite[simp]:

assumes finite F
 shows (f has_sum (sum f F)) F
 using assms

by (auto intro: tendsto_Lim simp: finite_subsets_at_top_finite infsum_def has_sum_def principal_eq_bot_iff)

lemma summable_on_finite[simp]:

fixes f :: 'a ⇒ 'b :: {comm_monoid_add, topological_space}
 assumes finite F
 shows f summable_on F
 using assms summable_on_def has_sum_finite by blast

lemma infsum_finite[simp]:

assumes finite F
 shows $\text{infsum } f F = \text{sum } f F$
 by (simp add: assms infsumI)

lemma has_sum_finite_approximation:

fixes f :: 'a ⇒ 'b :: {comm_monoid_add, metric_space}
 assumes (f has_sum x) A and $\varepsilon > 0$
 shows $\exists F. \text{finite } F \wedge F \subseteq A \wedge \text{dist } (\text{sum } f F) x \leq \varepsilon$

proof –

have (sum f \longrightarrow x) (finite_subsets_at_top A)

by (meson assms(1) has_sum_def)

hence *: $\forall F. F \text{ in } (\text{finite_subsets_at_top } A). \text{dist } (\text{sum } f F) x < \varepsilon$

using assms(2) by (rule tendstoD)

thus ?thesis

unfolding eventually_finite_subsets_at_top by fastforce

qed

lemma infsum_finite_approximation:

fixes f :: 'a ⇒ 'b :: {comm_monoid_add, metric_space}


```

  assumes  $f$  summable_on  $A$  and  $\varepsilon > 0$ 
  shows  $\exists F. \text{finite } F \wedge F \subseteq A \wedge \text{dist } (\text{sum } f F) (\text{infsum } f A) \leq \varepsilon$ 
proof -
  from assms have  $(f \text{ has\_sum } (\text{infsum } f A)) A$ 
  by (simp add: summable_iff_has_sum_infsum)
  from this and  $\langle \varepsilon > 0 \rangle$  show ?thesis
  by (rule has_sum_finite_approximation)
qed

lemma abs_summable_summable:
  fixes  $f :: \langle 'a \Rightarrow 'b :: \text{banach} \rangle$ 
  assumes  $\langle f \text{ abs\_summable\_on } A \rangle$ 
  shows  $\langle f \text{ summable\_on } A \rangle$ 
proof -
  from assms obtain  $L$  where  $\text{lim}: \langle (\text{sum } (\lambda x. \text{norm } (f x)) \longrightarrow L) (\text{finite\_subsets\_at\_top } A) \rangle$ 
  unfolding has_sum_def summable_on_def by blast
  then have *:  $\langle \text{cauchy\_filter } (\text{filtermap } (\text{sum } (\lambda x. \text{norm } (f x)))) (\text{finite\_subsets\_at\_top } A) \rangle$ 
  by (auto intro!: nhds_imp_cauchy_filter simp: filterlim_def)
  have  $\langle \exists P. \text{eventually } P (\text{finite\_subsets\_at\_top } A) \wedge$ 
     $(\forall F F'. P F \wedge P F' \longrightarrow \text{dist } (\text{sum } f F) (\text{sum } f F') < e) \rangle$  if  $\langle e > 0 \rangle$  for  $e$ 
  proof -
    define  $d$   $P$  where  $\langle d = e/4 \rangle$  and  $\langle P F \longleftrightarrow \text{finite } F \wedge F \subseteq A \wedge \text{dist } (\text{sum } (\lambda x. \text{norm } (f x)) F) L < d \rangle$  for  $F$ 
    then have  $\langle d > 0 \rangle$ 
    by (simp add: d_def that)
    have  $\text{ev\_}P: \langle \text{eventually } P (\text{finite\_subsets\_at\_top } A) \rangle$ 
    using lim
    by (auto simp add: P_def[abs_def]  $\langle 0 < d \rangle$  eventually_conj_iff eventually_finite_subsets_at_top_weakI tendsto_iff)

    moreover have  $\langle \text{dist } (\text{sum } f F1) (\text{sum } f F2) < e \rangle$  if  $\langle P F1 \rangle$  and  $\langle P F2 \rangle$  for  $F1 F2$ 
    proof -
      from ev_P
      obtain  $F'$  where  $\langle \text{finite } F' \rangle$  and  $\langle F' \subseteq A \rangle$  and  $P\_sup\_F': \langle \text{finite } F \wedge F \supseteq F' \wedge F \subseteq A \implies P F \rangle$  for  $F$ 
      by atomize_elim (simp add: eventually_finite_subsets_at_top)
      define  $F$  where  $\langle F = F' \cup F1 \cup F2 \rangle$ 
      have  $\langle \text{finite } F \rangle$  and  $\langle F \subseteq A \rangle$ 
      using F_def P_def[abs_def] that  $\langle \text{finite } F' \rangle \langle F' \subseteq A \rangle$  by auto
      have  $\text{dist\_}F: \langle \text{dist } (\text{sum } (\lambda x. \text{norm } (f x)) F) L < d \rangle$ 
      by (metis F_def  $\langle F \subseteq A \rangle P\_def P\_sup\_F' \langle \text{finite } F \rangle \text{le\_supE order\_refl}$ )

      have  $\text{dist\_}F\_subset: \langle \text{dist } (\text{sum } f F) (\text{sum } f F') < 2*d \rangle$  if  $F': \langle F' \subseteq F \rangle \langle P F' \rangle$  for  $F'$ 
      proof -
        have  $\langle \text{dist } (\text{sum } f F) (\text{sum } f F') = \text{norm } (\text{sum } f (F - F')) \rangle$ 

```

```

    unfolding dist_norm using ⟨finite F⟩ F' by (subst sum_diff) auto
  also have ⟨... ≤ norm (∑ x∈F-F'. norm (f x))⟩
    by (rule order.trans[OF sum_norm_le[OF order.refl]]) auto
  also have ⟨... = dist (∑ x∈F. norm (f x)) (∑ x∈F'. norm (f x))⟩
    unfolding dist_norm using ⟨finite F⟩ F' by (subst sum_diff) auto
  also have ⟨... < 2 * d⟩
    using dist_F F' unfolding P_def dist_norm real_norm_def by linarith
  finally show ⟨dist (sum f F) (sum f F') < 2*d⟩ .
qed

  have ⟨dist (sum f F1) (sum f F2) ≤ dist (sum f F) (sum f F1) + dist (sum
f F) (sum f F2)⟩
    by (rule dist_triangle3)
  also have ⟨... < 2 * d + 2 * d⟩
    by (intro add_strict_mono dist_F_subset that) (auto simp: F_def)
  also have ⟨... ≤ e⟩
    by (auto simp: d_def)
  finally show ⟨dist (sum f F1) (sum f F2) < e⟩ .
qed
then show ?thesis
  using ev_P by blast
qed
then have ⟨cauchy_filter (filtermap (sum f) (finite_subsets_at_top A))⟩
  by (simp add: cauchy_filter_metric_filtermap)
moreover have complete (UNIV::'b set)
  by (meson Cauchy_convergent UNIV_I complete_def convergent_def)
ultimately obtain L' where ⟨(sum f → L') (finite_subsets_at_top A)⟩
  using complete_uniform[where S=UNIV] by (force simp add: filterlim_def)
then show ?thesis
  using summable_on_def has_sum_def by blast
qed

```

The converse of *abs_summable_summable* does not hold: Consider the Hilbert space of square-summable sequences. Let e_i denote the sequence with 1 in the i th position and 0 elsewhere. Let $f(i) := e_i/i$ for $i \geq 1$. We have $\neg f \text{ abs_summable_on } UNIV$ because $\|f(i)\| = 1/i$ and thus the sum over $\|f(i)\|$ diverges. On the other hand, we have $f \text{ summable_on } UNIV$; the limit is the sequence with $1/i$ in the i th position.

(We have not formalized this separating example here because to the best of our knowledge, this Hilbert space has not been formalized in Isabelle/HOL yet.)

```

lemma norm_has_sum_bound:
  fixes f :: 'b ⇒ 'a::real_normed_vector
    and A :: 'b set
  assumes ((λx. norm (f x)) has_sum n) A
  assumes (f has_sum a) A
  shows norm a ≤ n
proof -

```

```

have norm a ≤ n + ε if ε > 0 for ε
proof-
  have ∃ F. norm (a - sum f F) ≤ ε ∧ finite F ∧ F ⊆ A
    using has_sum_finite_approximation[where A=A and f=f and ε=ε] assms
  <0 < ε>
    by (metis dist_commute dist_norm)
  then obtain F where norm (a - sum f F) ≤ ε
    and finite F and F ⊆ A
    by (simp add: atomize_elim)
  hence norm a ≤ norm (sum f F) + ε
  by (metis add.commute diff_add_cancel dual_order.refl norm_triangle_mono)
  also have ... ≤ sum (λx. norm (f x)) F + ε
    using norm_sum by auto
  also have ... ≤ n + ε
  proof (intro add_right_mono [OF has_sum_mono_neutral])
    show ((λx. norm (f x)) has_sum (∑ x∈F. norm (f x))) F
      by (simp add: <finite F>)
  qed (use <F ⊆ A> assms in auto)
  finally show ?thesis
    by assumption
qed
thus ?thesis
  using linordered_field_class.field_le_epsilon by blast
qed

```

```

lemma norm_infsum_bound:
  fixes f :: 'b ⇒ 'a::real_normed_vector
    and A :: 'b set
  assumes f_abs_summable_on A
  shows norm (infsum f A) ≤ infsum (λx. norm (f x)) A
proof (cases f_summable_on A)
  case True
    have ((λx. norm (f x)) has_sum (∑∞ x∈A. norm (f x))) A
      by (simp add: assms)
    then show ?thesis
      by (metis True has_sum_infsum norm_has_sum_bound)
  next
    case False
    obtain t where t_def: (sum (λx. norm (f x)) ⟶ t) (finite_subsets_at_top
A)
    using assms unfolding summable_on_def has_sum_def by blast
    have sumpos: sum (λx. norm (f x)) X ≥ 0
      for X
      by (simp add: sum_nonneg)
    have tgeq0: t ≥ 0
    proof(rule ccontr)
      define S::real set where S = {s. s < 0}
      assume ¬ 0 ≤ t
      hence t < 0 by simp

```

hence $t \in S$
 unfolding S_def by *blast*
 moreover have *open S*
 by (*metis S_def lessThan_def open_real_lessThan*)
 ultimately have $\forall_F X$ in *finite_subsets_at_top A*. $(\sum x \in X. \text{norm } (f x)) \in S$
 using t_def unfolding *tendsto_def* by *blast*
 hence $\exists X. (\sum x \in X. \text{norm } (f x)) \in S$
 by (*metis (no_types, lifting) eventually_mono filterlim_iff finite_subsets_at_top_neq_bot tendsto_Lim*)
 then obtain X where $(\sum x \in X. \text{norm } (f x)) \in S$
 by *blast*
 hence $(\sum x \in X. \text{norm } (f x)) < 0$
 unfolding S_def by *auto*
 thus *False* by (*simp add: leD sumpos*)
 qed
 have $\exists ! h. (\text{sum } (\lambda x. \text{norm } (f x)) \longrightarrow h)$ (*finite_subsets_at_top A*)
 using t_def *finite_subsets_at_top_neq_bot tendsto_unique* by *blast*
 hence $t = (\text{Topological_Spaces.Lim } (\text{finite_subsets_at_top } A) (\text{sum } (\lambda x. \text{norm } (f x))))$
 using t_def unfolding *Topological_Spaces.Lim_def*
 by (*metis the_equality*)
 hence $\text{Lim } (\text{finite_subsets_at_top } A) (\text{sum } (\lambda x. \text{norm } (f x))) \geq 0$
 using *tgeq0* by *blast*
 thus *?thesis* unfolding *infsum_def*
 using *False* by *auto*
 qed

lemma *infsum_tendsto*:

assumes $\langle f \text{ summable_on } S \rangle$
 shows $\langle (\lambda F. \text{sum } f F) \longrightarrow \text{infsum } f S \rangle$ (*finite_subsets_at_top S*)
 using *assms has_sum_def has_sum_infsum* by *blast*

lemma *has_sum_0*:

assumes $\langle \bigwedge x. x \in M \implies f x = 0 \rangle$
 shows $\langle f \text{ has_sum } 0 \rangle M$
 by (*metis assms finite.intros(1) has_sum_cong has_sum_cong_neutral has_sum_finite sum.neutral_const*)

lemma *summable_on_0*:

assumes $\langle \bigwedge x. x \in M \implies f x = 0 \rangle$
 shows $\langle f \text{ summable_on } M \rangle$
 using *assms summable_on_def has_sum_0* by *blast*

lemma *infsum_0*:

assumes $\langle \bigwedge x. x \in M \implies f x = 0 \rangle$
 shows $\langle \text{infsum } f M = 0 \rangle$
 by (*metis assms finite_subsets_at_top_neq_bot infsum_def has_sum_0 has_sum_def tendsto_Lim*)

Variants of *infsum_0* etc. suitable as *simp*-rules

lemma *infsum_0_simp*[simp]: $\langle \text{infsum } (\lambda_. 0) M = 0 \rangle$
 by (*simp_all add: infsum_0*)

lemma *summable_on_0_simp*[simp]: $\langle (\lambda_. 0) \text{ summable_on } M \rangle$
 by (*simp_all add: summable_on_0*)

lemma *has_sum_0_simp*[simp]: $\langle ((\lambda_. 0) \text{ has_sum } 0) M \rangle$
 by (*simp_all add: has_sum_0*)

lemma *has_sum_add*:

fixes $f g :: 'a \Rightarrow 'b :: \{\text{topological_comm_monoid_add}\}$

assumes $\langle f \text{ has_sum } a \rangle A$

assumes $\langle g \text{ has_sum } b \rangle A$

shows $\langle ((\lambda x. f x + g x) \text{ has_sum } (a + b)) A \rangle$

proof –

from *assms* **have** $\text{lim_}f: \langle \text{sum } f \longrightarrow a \rangle (\text{finite_subsets_at_top } A)$

and $\text{lim_}g: \langle \text{sum } g \longrightarrow b \rangle (\text{finite_subsets_at_top } A)$

by (*simp_all add: has_sum_def*)

then **have** $\text{lim}: \langle \text{sum } (\lambda x. f x + g x) \longrightarrow a + b \rangle (\text{finite_subsets_at_top } A)$

unfolding *sum.distrib* **by** (*rule tendsto_add*)

then **show** *?thesis*

by (*simp_all add: has_sum_def*)

qed

lemma *summable_on_add*:

fixes $f g :: 'a \Rightarrow 'b :: \{\text{topological_comm_monoid_add}\}$

assumes $\langle f \text{ summable_on } A \rangle$

assumes $\langle g \text{ summable_on } A \rangle$

shows $\langle (\lambda x. f x + g x) \text{ summable_on } A \rangle$

by (*metis (full_types) assms summable_on_def has_sum_add*)

lemma *infsum_add*:

fixes $f g :: 'a \Rightarrow 'b :: \{\text{topological_comm_monoid_add, t2_space}\}$

assumes $\langle f \text{ summable_on } A \rangle$

assumes $\langle g \text{ summable_on } A \rangle$

shows $\langle \text{infsum } (\lambda x. f x + g x) A = \text{infsum } f A + \text{infsum } g A \rangle$

proof –

have $\langle ((\lambda x. f x + g x) \text{ has_sum } (\text{infsum } f A + \text{infsum } g A)) A \rangle$

by (*simp add: assms has_sum_add*)

then **show** *?thesis*

using *infsumI* **by** *blast*

qed

lemma *has_sum_Un_disjoint*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_comm_monoid_add}\}$

assumes $\langle f \text{ has_sum } a \rangle A$

assumes $\langle f \text{ has_sum } b \rangle B$

```

assumes disj:  $A \cap B = \{\}$ 
shows  $\langle f \text{ has\_sum } (a + b) \rangle (A \cup B)$ 
proof -
  define fA fB where  $\langle fA \ x = (if \ x \in A \ \text{then } f \ x \ \text{else } 0) \rangle$ 
    and  $\langle fB \ x = (if \ x \notin A \ \text{then } f \ x \ \text{else } 0) \rangle$  for x
  have fA:  $\langle fA \ \text{has\_sum } a \rangle (A \cup B)$ 
    by (smt (verit, ccfv_SIG) DiffD1 DiffD2 UnCI fA_def assms(1) has_sum_cong_neutral
inf_sup_absorb)
  have fB:  $\langle fB \ \text{has\_sum } b \rangle (A \cup B)$ 
    by (smt (verit, best) DiffD1 DiffD2 IntE Un_iff fB_def assms(2) disj disjoint_iff_has_sum_cong_neutral)
  have fAB:  $\langle f \ x = fA \ x + fB \ x \rangle$  for x
    unfolding fA_def fB_def by simp
  show ?thesis
    unfolding fAB
    using fA fB by (rule has_sum_add)
qed

```

```

lemma summable_on_Un_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::topological_comm_monoid_add
  assumes f summable_on A
  assumes f summable_on B
  assumes disj:  $A \cap B = \{\}$ 
  shows  $\langle f \ \text{summable\_on } (A \cup B) \rangle$ 
  by (meson assms disj summable_on_def has_sum_Un_disjoint)

```

```

lemma infsum_Un_disjoint:
  fixes f :: 'a  $\Rightarrow$  'b::{topological_comm_monoid_add, t2_space}
  assumes f summable_on A
  assumes f summable_on B
  assumes disj:  $A \cap B = \{\}$ 
  shows  $\langle \text{infsum } f \ (A \cup B) = \text{infsum } f \ A + \text{infsum } f \ B \rangle$ 
  by (intro infsumI has_sum_Un_disjoint has_sum_infsum assms)

```

```

lemma norm_summable_imp_has_sum:
  fixes f :: nat  $\Rightarrow$  'a :: banach
  assumes summable ( $\lambda n. \text{norm } (f \ n)$ ) and f sums S
  shows (f has_sum S) (UNIV :: nat set)
  unfolding has_sum_def tendsto_iff eventually_finite_subsets_at_top
proof clarsimp
  fix  $\varepsilon$ ::real
  assume  $\varepsilon > 0$ 
  from assms obtain S' where S': ( $\lambda n. \text{norm } (f \ n)$ ) sums S'
    by (auto simp: summable_def)
  with  $\langle \varepsilon > 0 \rangle$  obtain N where N:  $\bigwedge n. n \geq N \implies |S' - (\sum_{i < n. \text{norm } (f \ i)}| < \varepsilon$ 
  by (auto simp: tendsto_iff eventually_at_top_linorder sums_def dist_norm abs_minus_commute)
  have dist (sum f Y) S  $< \varepsilon$  if finite Y  $\{.. < N\} \subseteq Y$  for Y

```

```

proof -
  from that have  $(\lambda n. \text{if } n \in Y \text{ then } 0 \text{ else } f \ n) \text{ sums } (S - \text{sum } f \ Y)$ 
    by (intro sums_If_finite_set'[OF ⟨f sums S⟩]) (auto simp: sum_negf)
  hence  $S - \text{sum } f \ Y = (\sum n. \text{if } n \in Y \text{ then } 0 \text{ else } f \ n)$ 
    by (simp add: sums_iff)
  also have  $\text{norm } \dots \leq (\sum n. \text{norm } (\text{if } n \in Y \text{ then } 0 \text{ else } f \ n))$ 
    by (rule summable_norm[OF summable_comparison_test'[OF assms(1)]])
  auto
  also have  $\dots \leq (\sum n. \text{if } n < N \text{ then } 0 \text{ else } \text{norm } (f \ n))$ 
    using that by (intro suminf_le summable_comparison_test'[OF assms(1)])
  auto
  also have  $(\lambda n. \text{if } n \in \{..<N\} \text{ then } 0 \text{ else } \text{norm } (f \ n)) \text{ sums } (S' - (\sum i<N. \text{norm } (f \ i)))$ 
    by (intro sums_If_finite_set'[OF S']) (auto simp: sum_negf)
  hence  $(\sum n. \text{if } n < N \text{ then } 0 \text{ else } \text{norm } (f \ n)) = S' - (\sum i<N. \text{norm } (f \ i))$ 
    by (simp add: sums_iff)
  also have  $S' - (\sum i<N. \text{norm } (f \ i)) \leq |S' - (\sum i<N. \text{norm } (f \ i))|$  by simp
  also have  $\dots < \varepsilon$  by (rule N) auto
  finally show ?thesis by (simp add: dist_norm norm_minus_commute)
qed
then show  $\exists X. \text{finite } X \wedge (\forall Y. \text{finite } Y \wedge X \subseteq Y \longrightarrow \text{dist } (\text{sum } f \ Y) \ S < \varepsilon)$ 
    by (meson finite_lessThan subset_UNIV)
qed

```

lemma norm_summable_imp_summable_on:

```

fixes f :: nat => 'a :: banach
assumes summable  $(\lambda n. \text{norm } (f \ n))$ 
shows f summable_on UNIV
using norm_summable_imp_has_sum[OF assms, of suminf f] assms
by (auto simp: sums_iff summable_on_def dest: summable_norm_cancel)

```

The following lemma indeed needs a complete space (as formalized by the premise *complete UNIV*). The following two counterexamples show this:

- Consider the real vector space V of sequences with finite support, and with the ℓ_2 -norm (sum of squares). Let e_i denote the sequence with a 1 at position i . Let $f : \mathbb{Z} \rightarrow V$ be defined as $f(n) := e_{|n|}/n$ (with $f(0) := 0$). We have that $\sum_{n \in \mathbb{Z}} f(n) = 0$ (it even converges absolutely). But $\sum_{n \in \mathbb{N}} f(n)$ does not exist (it would converge against a sequence with infinite support).
- Let f be a positive rational valued function such that $\sum_{x \in B} f(x)$ is $\sqrt{2}$ and $\sum_{x \in A} f(x)$ is 1 (over the reals, with $A \subseteq B$). Then $\sum_{x \in B} f(x)$ does not exist over the rationals. But $\sum_{x \in A} f(x)$ exists.

The lemma also requires uniform continuity of the addition. And example of a topological group with continuous but not uniformly continuous addition would be the positive reals with the usual multiplication as the addition.

We do not know whether the lemma would also hold for such topological groups.

```

lemma summable_on_subset_aux:
  fixes  $A B$  and  $f :: \langle 'a \Rightarrow 'b :: \{ab\_group\_add, uniform\_space\} \rangle$ 
  assumes  $\langle complete (UNIV :: 'b set) \rangle$ 
  assumes plus_cont:  $\langle uniformly\_continuous\_on UNIV (\lambda(x::'b,y). x+y) \rangle$ 
  assumes  $\langle f \text{ summable\_on } A \rangle$ 
  assumes  $\langle B \subseteq A \rangle$ 
  shows  $\langle f \text{ summable\_on } B \rangle$ 
proof -
  let  $?filter\_fB = \langle filtermap (sum f) (finite\_subsets\_at\_top B) \rangle$ 
  from  $\langle f \text{ summable\_on } A \rangle$ 
  obtain  $S$  where  $\langle (sum f \longrightarrow S) (finite\_subsets\_at\_top A) \rangle$  (is  $\langle (sum f \longrightarrow S) ?filter\_A \rangle$ )
    using summable_on_def has_sum_def by blast
  then have cauchy_fA:  $\langle cauchy\_filter (filtermap (sum f) (finite\_subsets\_at\_top A)) \rangle$  (is  $\langle cauchy\_filter ?filter\_fA \rangle$ )
    by (auto intro!: nhds_imp_cauchy_filter simp: filterlim_def)

  have  $\langle cauchy\_filter (filtermap (sum f) (finite\_subsets\_at\_top B)) \rangle$ 
  proof (unfold cauchy_filter_def, rule filter_leI)
    fix  $E :: \langle ('b \times 'b) \Rightarrow bool \rangle$  assume  $\langle eventually E \text{ uniformity} \rangle$ 
    then obtain  $E'$  where  $\langle eventually E' \text{ uniformity} \rangle$  and  $E'E'E: \langle E' (x, y) \longrightarrow E' (y, z) \longrightarrow E (x, z) \rangle$  for  $x y z$ 
      using uniformity_trans by blast
    obtain  $D$  where  $\langle eventually D \text{ uniformity} \rangle$  and  $DE: \langle D (x, y) \Longrightarrow E' (x+c, y+c) \rangle$  for  $x y c$ 
      using plus_cont  $\langle eventually E' \text{ uniformity} \rangle$ 
      unfolding uniformly_continuous_on_uniformity filterlim_def le_filter_def uniformity_prod_def
      by (auto simp: case_prod_beta eventually_filtermap eventually_prod_same uniformity_refl)
    have  $DE': E' (x, y)$  if  $D (x + c, y + c)$  for  $x y c$ 
      using  $DE$  [of  $x + c y + c - c$ ] that by simp

    from  $\langle eventually D \text{ uniformity} \rangle$  and cauchy_fA have  $\langle eventually D (?filter\_fA \times_F ?filter\_fA) \rangle$ 
      unfolding cauchy_filter_def le_filter_def by simp
    then obtain  $P1 P2$ 
      where  $ev\_P1: \langle eventually (\lambda F. P1 (sum f F)) ?filter\_A \rangle$ 
      and  $ev\_P2: \langle eventually (\lambda F. P2 (sum f F)) ?filter\_A \rangle$ 
      and  $P1P2E: \langle P1 x \Longrightarrow P2 y \Longrightarrow D (x, y) \rangle$  for  $x y$ 
      unfolding eventually_prod_filter eventually_filtermap
      by auto
    from  $ev\_P1$  obtain  $F1$  where  $F1: \langle finite F1 \rangle \langle F1 \subseteq A \rangle \langle \bigwedge F. F \supseteq F1 \Longrightarrow finite F \Longrightarrow F \subseteq A \Longrightarrow P1 (sum f F) \rangle$ 
      by (metis eventually_finite_subsets_at_top)
    from  $ev\_P2$  obtain  $F2$  where  $F2: \langle finite F2 \rangle \langle F2 \subseteq A \rangle \langle \bigwedge F. F \supseteq F2 \Longrightarrow finite F \Longrightarrow F \subseteq A \Longrightarrow P2 (sum f F) \rangle$ 

```



```

    by (metis eventually_finite_subsets_at_top)
  define F0 F0A F0B where ‹F0 ≡ F1 ∪ F2› and ‹F0A ≡ F0 - B› and ‹F0B
≡ F0 ∩ B›
  have [simp]: ‹finite F0› ‹F0 ⊆ A›
    using ‹F1 ⊆ A› ‹F2 ⊆ A› ‹finite F1› ‹finite F2› unfolding F0_def by
blast+

  have *: E' (sum f F1', sum f F2')
    if F1' ⊇ F0B F2' ⊇ F0B finite F1' finite F2' F1' ⊆ B F2' ⊆ B for F1' F2'
  proof (intro DE'[where c = sum f F0A] P1P2E)
    have P1 (sum f (F1' ∪ F0A))
      using that assms F1(1,2) F2(1,2) by (intro F1) (auto simp: F0A_def
F0B_def F0_def)
    thus P1 (sum f F1' + sum f F0A)
      by (subst (asm) sum.union_disjoint) (use that in ‹auto simp: F0A_def›)
    next
    have P2 (sum f (F2' ∪ F0A))
      using that assms F1(1,2) F2(1,2) by (intro F2) (auto simp: F0A_def
F0B_def F0_def)
    thus P2 (sum f F2' + sum f F0A)
      by (subst (asm) sum.union_disjoint) (use that in ‹auto simp: F0A_def›)

  qed

  have eventually (λx. E' (x, sum f F0B)) (filtermap (sum f) (finite_subsets_at_top
B))
  and eventually (λx. E' (sum f F0B, x)) (filtermap (sum f) (finite_subsets_at_top
B))
    unfolding eventually_filtermap eventually_finite_subsets_at_top
    by (rule exI[of _ F0B]; use * in ‹force simp: F0B_def›)+
  then
  show ‹eventually E (?filter_fB ×F ?filter_fB)›
    unfolding eventually_prod_filter
    using E'E'E by blast
  qed

  then obtain x where ‹?filter_fB ≤ nhds x›
    using cauchy_filter_complete_converges[of ?filter_fB UNIV] ‹complete (UNIV
:: _)›
    by (auto simp: filtermap_bot_iff)
  then have ‹(sum f ⟶ x) (finite_subsets_at_top B)›
    by (auto simp: filterlim_def)
  then show ?thesis
    by (auto simp: summable_on_def has_sum_def)
  qed

```

A special case of *summable_on_subset_aux* for Banach spaces with fewer premises.

lemma *summable_on_subset_banach*:

```

fixes  $A B$  and  $f :: \langle 'a \Rightarrow 'b::\text{banach} \rangle$ 
assumes  $\langle f \text{ summable\_on } A \rangle$ 
assumes  $\langle B \subseteq A \rangle$ 
shows  $\langle f \text{ summable\_on } B \rangle$ 
by (meson Cauchy_convergent UNIV_I assms complete_def convergent_def isU-Cont_plus summable_on_subset_aux)

```

```

lemma has_sum_empty[simp]:  $\langle (f \text{ has\_sum } 0) \{\} \rangle$ 
by (meson ex_in_conv has_sum_0)

```

```

lemma summable_on_empty[simp]:  $\langle f \text{ summable\_on } \{\} \rangle$ 
by auto

```

```

lemma infsun_empty[simp]:  $\langle \text{infsun } f \{\} = 0 \rangle$ 
by simp

```

```

lemma sum_has_sum:
fixes  $f :: 'a \Rightarrow 'b::\text{topological\_comm\_monoid\_add}$ 
assumes  $\langle \text{finite } A \rangle$ 
assumes  $\langle \bigwedge a. a \in A \Longrightarrow (f \text{ has\_sum } (s a)) (B a) \rangle$ 
assumes  $\langle \bigwedge a a'. a \in A \Longrightarrow a' \in A \Longrightarrow a \neq a' \Longrightarrow B a \cap B a' = \{\} \rangle$ 
shows  $\langle (f \text{ has\_sum } (\text{sum } s A)) (\bigcup a \in A. B a) \rangle$ 
using assms
proof (induction)
case empty
then show ?case
by simp
next
case (insert x A)
have  $\langle (f \text{ has\_sum } (s x)) (B x) \rangle$ 
by (simp add: insert.premis)
moreover have IH:  $\langle (f \text{ has\_sum } (\text{sum } s A)) (\bigcup a \in A. B a) \rangle$ 
using insert by simp
ultimately have  $\langle (f \text{ has\_sum } (s x + \text{sum } s A)) (B x \cup (\bigcup a \in A. B a)) \rangle$ 
using insert by (intro has_sum_Un_disjoint) auto
then show ?case
using insert.hyps by auto
qed

```

```

lemma summable_on_finite_union_disjoint:
fixes  $f :: 'a \Rightarrow 'b::\text{topological\_comm\_monoid\_add}$ 
assumes finite:  $\langle \text{finite } A \rangle$ 
assumes conv:  $\langle \bigwedge a. a \in A \Longrightarrow f \text{ summable\_on } (B a) \rangle$ 
assumes disj:  $\langle \bigwedge a a'. a \in A \Longrightarrow a' \in A \Longrightarrow a \neq a' \Longrightarrow B a \cap B a' = \{\} \rangle$ 
shows  $\langle f \text{ summable\_on } (\bigcup a \in A. B a) \rangle$ 
using sum_has_sum [of A f B] assms unfolding summable_on_def by metis

```

```

lemma sum_infsun:

```

```

fixes  $f :: 'a \Rightarrow 'b :: \{ \text{topological\_comm\_monoid\_add}, \text{t2\_space} \}$ 
assumes  $\text{finite}: \langle \text{finite } A \rangle$ 
assumes  $\text{conv}: \langle \bigwedge a. a \in A \implies f \text{ summable\_on } (B a) \rangle$ 
assumes  $\text{disj}: \langle \bigwedge a a'. a \in A \implies a' \in A \implies a \neq a' \implies B a \cap B a' = \{ \} \rangle$ 
shows  $\langle \text{sum } (\lambda a. \text{infsum } f (B a)) A = \text{infsum } f (\bigcup_{a \in A} B a) \rangle$ 
by ( $\text{metis } (\text{no\_types}, \text{lifting}) \text{ assms has\_sum\_infsum infsumI sum\_has\_sum}$ )

```

The lemmas *infsum_comm_additive_general* and *infsum_comm_additive* (and variants) below both state that the infinite sum commutes with a continuous additive function. *infsum_comm_additive_general* is stated more for more general type classes at the expense of a somewhat less compact formulation of the premises. E.g., by avoiding the constant *additive* which introduces an additional sort constraint (group instead of monoid). For example, extended reals (*ereal*, *ennreal*) are not covered by *infsum_comm_additive*.

lemma *has_sum_comm_additive_general*:

```

fixes  $f :: \langle 'b :: \{ \text{comm\_monoid\_add}, \text{topological\_space} \} \Rightarrow 'c :: \{ \text{comm\_monoid\_add}, \text{topological\_space} \} \rangle$ 
assumes  $\text{f\_sum}: \langle \bigwedge F. \text{finite } F \implies F \subseteq S \implies \text{sum } (f \circ g) F = f (\text{sum } g F) \rangle$ 
  — Not using additive because it would add sort constraint ab_group_add
assumes  $\text{cont}: \langle f \text{ --}x \rightarrow f x \rangle$ 
  — For t2_space, this is equivalent to isCont f x by isCont_def.
assumes  $\text{infsum}: \langle (g \text{ has\_sum } x) S \rangle$ 
shows  $\langle ((f \circ g) \text{ has\_sum } (f x)) S \rangle$ 

```

proof —

```

have  $\langle (\text{sum } g \longrightarrow x) (\text{finite\_subsets\_at\_top } S) \rangle$ 
  using infsum_has_sum_def by blast
then have  $\langle ((f \circ \text{sum } g) \longrightarrow f x) (\text{finite\_subsets\_at\_top } S) \rangle$ 
  by ( $\text{meson cont filterlim\_def tendsto\_at\_iff\_tendsto\_nhds tendsto\_compose\_filtermap tendsto\_mono}$ )
then have  $\langle (\text{sum } (f \circ g) \longrightarrow f x) (\text{finite\_subsets\_at\_top } S) \rangle$ 
  using tendsto\_cong f_sum
  by ( $\text{simp add: Lim\_transform\_eventually eventually\_finite\_subsets\_at\_top\_weakI}$ )
then show  $\langle ((f \circ g) \text{ has\_sum } (f x)) S \rangle$ 
  using has_sum_def by blast

```

qed

lemma *summable_on_comm_additive_general*:

```

fixes  $f :: \langle 'b :: \{ \text{comm\_monoid\_add}, \text{topological\_space} \} \Rightarrow 'c :: \{ \text{comm\_monoid\_add}, \text{topological\_space} \} \rangle$ 
assumes  $\langle \bigwedge F. \text{finite } F \implies F \subseteq S \implies \text{sum } (f \circ g) F = f (\text{sum } g F) \rangle$ 
  — Not using additive because it would add sort constraint ab_group_add
assumes  $\langle \bigwedge x. (g \text{ has\_sum } x) S \implies f \text{ --}x \rightarrow f x \rangle$ 
  — For t2_space, this is equivalent to isCont f x by isCont_def.
assumes  $\langle g \text{ summable\_on } S \rangle$ 
shows  $\langle (f \circ g) \text{ summable\_on } S \rangle$ 
by ( $\text{meson assms summable\_on\_def has\_sum\_comm\_additive\_general has\_sum\_def infsum\_tendsto}$ )

```

lemma *infsum_comm_additive_general*:

```

fixes  $f :: \langle 'b :: \{ \text{comm\_monoid\_add}, \text{t2\_space} \} \Rightarrow 'c :: \{ \text{comm\_monoid\_add}, \text{t2\_space} \} \rangle$ 

```

assumes $f_sum: \langle \bigwedge F. finite\ F \implies F \subseteq S \implies sum\ (f \circ g)\ F = f\ (sum\ g\ F) \rangle$
 — Not using *additive* because it would add sort constraint *ab_group_add*
assumes $\langle isCont\ f\ (infsun\ g\ S) \rangle$
assumes $\langle g\ summable_on\ S \rangle$
shows $\langle infsun\ (f \circ g)\ S = f\ (infsun\ g\ S) \rangle$
using *assms*
by (*intro infsunI has_sum_comm_additive_general has_sum_infsun*) (*auto simp: isCont_def*)

lemma *has_sum_comm_additive*:
fixes $f :: \langle 'b :: \{ab_group_add, topological_space\} \Rightarrow 'c :: \{ab_group_add, topological_space\} \rangle$
assumes $\langle additive\ f \rangle$
assumes $\langle f\ -x \rightarrow f\ x \rangle$
 — For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes $infsun: \langle (g\ has_sum\ x)\ S \rangle$
shows $\langle ((f \circ g)\ has_sum\ (f\ x))\ S \rangle$
using *assms*
by (*intro has_sum_comm_additive_general has_sum_infsun*) (*auto simp: isCont_def additive.sum*)

lemma *summable_on_comm_additive*:
fixes $f :: \langle 'b :: \{ab_group_add, t2_space\} \Rightarrow 'c :: \{ab_group_add, topological_space\} \rangle$
assumes $\langle additive\ f \rangle$
assumes $\langle isCont\ f\ (infsun\ g\ S) \rangle$
assumes $\langle g\ summable_on\ S \rangle$
shows $\langle (f \circ g)\ summable_on\ S \rangle$
by (*meson assms summable_on_def has_sum_comm_additive has_sum_infsun isContD*)

lemma *infsun_comm_additive*:
fixes $f :: \langle 'b :: \{ab_group_add, t2_space\} \Rightarrow 'c :: \{ab_group_add, t2_space\} \rangle$
assumes $\langle additive\ f \rangle$
assumes $\langle isCont\ f\ (infsun\ g\ S) \rangle$
assumes $\langle g\ summable_on\ S \rangle$
shows $\langle infsun\ (f \circ g)\ S = f\ (infsun\ g\ S) \rangle$
by (*rule infsun_comm_additive_general; auto simp: assms additive.sum*)

lemma *nonneg_bdd_above_has_sum*:
fixes $f :: \langle 'a \Rightarrow 'b :: \{conditionally_complete_linorder, ordered_comm_monoid_add, linorder_topology\} \rangle$
assumes $\langle \bigwedge x. x \in A \implies f\ x \geq 0 \rangle$
assumes $\langle bdd_above\ (sum\ f\ ' \{F. F \subseteq A \wedge finite\ F\}) \rangle$
shows $\langle (f\ has_sum\ (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F))\ A \rangle$
proof —
have $\langle (sum\ f \longrightarrow (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F))\ (finite_subsets_at_top\ A) \rangle$
proof (*rule order_tendstoI*)
fix a **assume** $\langle a < (SUP\ F \in \{F. finite\ F \wedge F \subseteq A\}. sum\ f\ F) \rangle$
then obtain F **where** $\langle a < sum\ f\ F \rangle$ **and** $\langle finite\ F \rangle$ **and** $\langle F \subseteq A \rangle$

```

    by (metis (mono_tags, lifting) Collect_cong Collect_empty_eq assms(2)
        empty_subsetI finite.emptyI less_cSUP_iff mem_Collect_eq)
    have  $\bigwedge Y. \llbracket \text{finite } Y; F \subseteq Y; Y \subseteq A \rrbracket \implies a < \text{sum } f Y$ 
    by (meson DiffE  $\langle a < \text{sum } f F \rangle$  assms(1) less_le_trans subset_iff sum_mono2)
    then show  $\langle \forall_F x \text{ in finite_subsets\_at\_top } A. a < \text{sum } f x \rangle$ 
    by (metis  $\langle F \subseteq A \rangle$   $\langle \text{finite } F \rangle$  eventually_finite_subsets_at_top)
  next
    fix a assume *:  $\langle (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) < a \rangle$ 
    have  $\text{sum } f F \leq (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F)$  if  $\langle F \subseteq A \rangle$  and  $\langle \text{finite } F \rangle$  for F
    by (rule cSUP_upper) (use that assms(2) in  $\langle \text{auto simp: conj_commute} \rangle$ )
    then show  $\langle \forall_F x \text{ in finite_subsets\_at\_top } A. \text{sum } f x < a \rangle$ 
    by (metis (no_types, lifting) * eventually_finite_subsets_at_top_weakI order_le_less_trans)
  qed
  then show ?thesis
    using has_sum_def by blast
qed

```

lemma nonneg_bdd_above_summable_on:

```

fixes f ::  $\langle 'a \Rightarrow 'b :: \{ \text{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology} \} \rangle$ 
assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
assumes  $\langle \text{bdd\_above } (\text{sum } f ' \{F. F \subseteq A \wedge \text{finite } F\}) \rangle$ 
shows  $\langle f \text{ summable\_on } A \rangle$ 
using assms summable_on_def nonneg_bdd_above_has_sum by blast

```

lemma nonneg_bdd_above_infsum:

```

fixes f ::  $\langle 'a \Rightarrow 'b :: \{ \text{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology} \} \rangle$ 
assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
assumes  $\langle \text{bdd\_above } (\text{sum } f ' \{F. F \subseteq A \wedge \text{finite } F\}) \rangle$ 
shows  $\langle \text{infsum } f A = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) \rangle$ 
using assms by (auto intro!: infsumI nonneg_bdd_above_has_sum)

```

lemma nonneg_has_sum_complete:

```

fixes f ::  $\langle 'a \Rightarrow 'b :: \{ \text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology} \} \rangle$ 
assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
shows  $\langle (f \text{ has\_sum } (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F)) A \rangle$ 
using assms nonneg_bdd_above_has_sum by blast

```

lemma nonneg_summable_on_complete:

```

fixes f ::  $\langle 'a \Rightarrow 'b :: \{ \text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology} \} \rangle$ 
assumes  $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$ 
shows  $\langle f \text{ summable\_on } A \rangle$ 
using assms nonneg_bdd_above_summable_on by blast

```

lemma nonneg_infsum_complete:

```

fixes f ::  $\langle 'a \Rightarrow 'b :: \{ \text{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology} \} \rangle$ 

```

assumes $\langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$
shows $\langle \text{infsum } f A = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{sum } f F) \rangle$
using *assms nonneg_bdd_above_infsum* **by** *blast*

lemma *has_sum_nonneg*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{ordered_comm_monoid_add, linorder_topology}\}$
assumes $(f \text{ has_sum } a) M$
and $\bigwedge x. x \in M \implies 0 \leq f x$
shows $a \geq 0$

by (*metis (no_types, lifting) DiffD1 assms empty_iff has_sum_0 has_sum_mono_neutral order_refl*)

lemma *infsum_nonneg*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{ordered_comm_monoid_add, linorder_topology}\}$
assumes $\bigwedge x. x \in M \implies 0 \leq f x$
shows $\text{infsum } f M \geq 0$ (**is** *?lhs \geq _*)

by (*metis assms has_sum_infsum has_sum_nonneg infsum_not_exists linorder_linear*)

lemma *has_sum_mono2*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_comm_monoid_add, linorder_topology}\}$
assumes $(f \text{ has_sum } S) A (f \text{ has_sum } S') B A \subseteq B$
assumes $\bigwedge x. x \in B - A \implies f x \geq 0$
shows $S \leq S'$

by (*metis add_0 add_right_mono assms diff_add_cancel has_sum_Diff has_sum_nonneg*)

lemma *infsum_mono2*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_comm_monoid_add, linorder_topology}\}$
assumes $f \text{ summable_on } A f \text{ summable_on } B A \subseteq B$
assumes $\bigwedge x. x \in B - A \implies f x \geq 0$
shows $\text{infsum } f A \leq \text{infsum } f B$

by (*rule has_sum_mono2[OF has_sum_infsum has_sum_infsum]*) (*use assms in auto*)

lemma *finite_sum_le_has_sum*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_comm_monoid_add, linorder_topology}\}$
assumes $(f \text{ has_sum } S) A \text{ finite } B B \subseteq A$
assumes $\bigwedge x. x \in A - B \implies f x \geq 0$
shows $\text{sum } f B \leq S$

by (*meson assms has_sum_finite has_sum_mono2*)

lemma *finite_sum_le_infsum*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_comm_monoid_add, linorder_topology}\}$
assumes $f \text{ summable_on } A \text{ finite } B B \subseteq A$
assumes $\bigwedge x. x \in A - B \implies f x \geq 0$
shows $\text{sum } f B \leq \text{infsum } f A$

by (*rule finite_sum_le_has_sum[OF has_sum_infsum]*) (*use assms in auto*)

lemma *has_sum_reindex*:

assumes $\langle \text{inj_on } h A \rangle$

```

  shows ⟨(g has_sum x) (h ` A) ⟷ ((g ∘ h) has_sum x) A⟩
proof -
  have ⟨(g has_sum x) (h ` A) ⟷ (sum g ⟶ x) (finite_subsets_at_top (h `
A))⟩
  by (simp add: has_sum_def)
  also have ⟨... ⟷ ((λF. sum g (h ` F)) ⟶ x) (finite_subsets_at_top A)⟩
  by (metis assms filterlim_filtermap_filtermap_image_finite_subsets_at_top)
  also have ⟨... ⟷ (sum (g ∘ h) ⟶ x) (finite_subsets_at_top A)⟩
proof (intro tendsto_cong eventually_finite_subsets_at_top_weakI sum_reindex)
  show ∧X. [finite X; X ⊆ A] ⟹ inj_on h X
  using assms subset_inj_on by blast
qed
  also have ⟨... ⟷ ((g ∘ h) has_sum x) A⟩
  by (simp add: has_sum_def)
  finally show ?thesis .
qed

```

lemma *summable_on_reindex*:

```

  assumes ⟨inj_on h A⟩
  shows ⟨g summable_on (h ` A) ⟷ (g ∘ h) summable_on A⟩
  by (simp add: assms summable_on_def has_sum_reindex)

```

lemma *infsum_reindex*:

```

  assumes ⟨inj_on h A⟩
  shows ⟨infsum g (h ` A) = infsum (g ∘ h) A⟩
  by (metis assms has_sum_infsum has_sum_reindex infsumI infsum_def)

```

lemma *summable_on_reindex_bij_betw*:

```

  assumes bij_betw g A B
  shows (λx. f (g x)) summable_on A ⟷ f summable_on B
  by (smt (verit) assms bij_betw_def o_apply summable_on_cong summable_on_reindex)

```

lemma *infsum_reindex_bij_betw*:

```

  assumes bij_betw g A B
  shows infsum (λx. f (g x)) A = infsum f B
  by (metis (mono_tags, lifting) assms bij_betw_def infsum_cong infsum_reindex
o_def)

```

lemma *sum_uniformity*:

```

  assumes plus_cont: ⟨uniformly_continuous_on UNIV (λ(x::'b):{uniform_space, comm_monoid_add}, y).
x+y)⟩
  assumes EE: ⟨eventually E uniformity⟩
  obtains D where ⟨eventually D uniformity⟩
  and ⟨∧M::'a set. ∧f f' :: 'a ⇒ 'b. card M ≤ n ∧ (∀ m∈M. D (f m, f' m)) ⟹
E (sum f M, sum f' M)⟩
proof (atomize_elim, insert EE, induction n arbitrary: E rule:nat_induct)
  case 0
  then show ?case

```

```

    by (metis card_eq_0_iff_equals0D le_zero_eq sum.infinite sum.not_neutral_contains_not_neutral
uniformity_refl)
next
  case (Suc n)
  from plus_cont[unfolded uniformly_continuous_on_uniformity filterlim_def le_filter_def,
rule_format, OF Suc.premis]
  obtain D1 D2 where ⟨eventually D1 uniformity⟩ and ⟨eventually D2 uniformity⟩

    and D1D2E: ⟨D1 (x, y) ⟹ D2 (x', y') ⟹ E (x + x', y + y')⟩ for x y x' y'
  apply atomize_elim
  by (auto simp: eventually_prod_filter case_prod_beta uniformity_prod_def
eventually_filtermap)

  from Suc.IH[OF ⟨eventually D2 uniformity⟩]
  obtain D3 where ⟨eventually D3 uniformity⟩ and D3: ⟨card M ≤ n ⟹
(∀ m∈M. D3 (f m, f' m)) ⟹ D2 (sum f M, sum f' M)⟩
  for M :: 'a set and f f'
  by metis

  define D where ⟨D x ≡ D1 x ∧ D3 x⟩ for x
  have ⟨eventually D uniformity⟩
  using D_def ⟨eventually D1 uniformity⟩ ⟨eventually D3 uniformity⟩ eventu-
ally_elim2 by blast

  have ⟨E (sum f M, sum f' M)⟩
  if ⟨card M ≤ Suc n⟩ and DM: ⟨∀ m∈M. D (f m, f' m)⟩
  for M :: 'a set and f f'
  proof (cases ⟨card M = 0⟩)
  case True
  then show ?thesis
  by (metis Suc.premis card_eq_0_iff sum.empty sum.infinite uniformity_refl)
  next
  case False
  with ⟨card M ≤ Suc n⟩ obtain N x where ⟨card N ≤ n⟩ and ⟨x ∉ N⟩ and
⟨M = insert x N⟩
  by (metis card_Suc_eq less_Suc_eq_0_disj less_Suc_eq_le)

  from DM have ⟨∧m. m∈N ⟹ D (f m, f' m)⟩
  using ⟨M = insert x N⟩ by blast
  with D3[OF ⟨card N ≤ n⟩]
  have D2_N: ⟨D2 (sum f N, sum f' N)⟩
  using D_def by blast

  from DM
  have ⟨D (f x, f' x)⟩
  using ⟨M = insert x N⟩ by blast
  then have ⟨D1 (f x, f' x)⟩
  by (simp add: D_def)

```



```

with  $D2\_N$ 
have  $\langle E (f x + \text{sum } f N, f' x + \text{sum } f' N) \rangle$ 
  using  $D1D2E$  by presburger

then show  $\langle E (\text{sum } f M, \text{sum } f' M) \rangle$ 
  by (metis  $\text{False} \langle M = \text{insert } x N \rangle \langle x \notin N \rangle \text{card.infinite finite\_insert}$ 
 $\text{sum.insert}$ )
qed
with  $\langle \text{eventually } D \text{ uniformity} \rangle$  show ?case
  by auto
qed

lemma has_sum_Sigma:
  fixes  $A :: 'a \text{ set}$  and  $B :: 'a \Rightarrow 'b \text{ set}$ 
    and  $f :: \langle 'a \times 'b \Rightarrow 'c :: \{ \text{comm\_monoid\_add, uniform\_space} \} \rangle$ 
  assumes plus_cont:  $\langle \text{uniformly\_continuous\_on UNIV } (\lambda(x::'c,y). x+y) \rangle$ 
  assumes summableAB:  $\langle f \text{ has\_sum } a \rangle (\text{Sigma } A B)$ 
  assumes summableB:  $\langle \bigwedge x. x \in A \implies ((\lambda y. f (x, y)) \text{ has\_sum } b x) (B x) \rangle$ 
  shows  $\langle b \text{ has\_sum } a \rangle A$ 
proof -
  define  $F \text{ FB } FA$  where  $\langle F = \text{finite\_subsets\_at\_top } (\text{Sigma } A B) \rangle$  and  $\langle FB x$ 
 $= \text{finite\_subsets\_at\_top } (B x) \rangle$ 
    and  $\langle FA = \text{finite\_subsets\_at\_top } A \rangle$  for  $x$ 

  from summableB
  have  $\text{sum\_b}: \langle (\text{sum } (\lambda y. f (x, y)) \longrightarrow b x) (FB x) \rangle$  if  $\langle x \in A \rangle$  for  $x$ 
    using  $FB\_def[\text{abs\_def}] \text{has\_sum\_def}$  that by auto
  from summableAB
  have  $\text{sum\_S}: \langle (\text{sum } f \longrightarrow a) F \rangle$ 
    using  $F\_def \text{has\_sum\_def}$  by blast

  have finite_proj:  $\langle \text{finite } \{b \mid b. (a,b) \in H\} \rangle$  if  $\langle \text{finite } H \rangle$  for  $H :: \langle ('a \times 'b) \text{ set} \rangle$ 
and  $a$ 
    by (metis (no_types, lifting) finite_imageI finite_subset image_eqI mem_Collect_eq
 $\text{snd\_conv}$  subsetI that)

  have  $\langle (\text{sum } b \longrightarrow a) FA \rangle$ 
proof (rule tendsto_iff_uniformity[THEN iffD2, rule_format])
    fix  $E :: \langle ('c \times 'c) \Rightarrow \text{bool} \rangle$ 
    assume  $\langle \text{eventually } E \text{ uniformity} \rangle$ 
    then obtain  $D$  where  $D\_uni: \langle \text{eventually } D \text{ uniformity} \rangle$  and  $DDE': \langle \bigwedge x y z.$ 
 $D (x, y) \implies D (y, z) \implies E (x, z) \rangle$ 
    by (metis (no_types, lifting)  $\langle \text{eventually } E \text{ uniformity} \rangle$  uniformity_transE)
    from  $\text{sum\_S}$  obtain  $G$  where  $\langle \text{finite } G \rangle$  and  $\langle G \subseteq \text{Sigma } A B \rangle$ 
    and  $G\_sum: \langle G \subseteq H \implies H \subseteq \text{Sigma } A B \implies \text{finite } H \implies D (\text{sum } f H,$ 
 $a) \rangle$  for  $H$ 
    unfolding tendsto_iff_uniformity
    by (metis (mono_tags, lifting)  $D\_uni$   $F\_def$  eventually_finite_subsets_at_top)
    have  $\langle \text{finite } (\text{fst } ` G) \rangle$  and  $\langle \text{fst } ` G \subseteq A \rangle$ 

```

```

using ⟨finite G⟩ ⟨G ⊆ Sigma A B⟩ by auto
thm uniformity_prod_def
define Ga where ⟨Ga a = {b. (a,b) ∈ G}⟩ for a
have Ga_fin: ⟨finite (Ga a)⟩ and Ga_B: ⟨Ga a ⊆ B a⟩ for a
using ⟨finite G⟩ ⟨G ⊆ Sigma A B⟩ finite_proj by (auto simp: Ga_def
finite_proj)

have ⟨E (sum b M, a)⟩ if ⟨M ⊇ fst ‘ G⟩ and ⟨finite M⟩ and ⟨M ⊆ A⟩ for M
proof –
define FMB where ⟨FMB = finite_subsets_at_top (Sigma M B)⟩
have ⟨eventually (λH. D (∑ a∈M. b a, ∑ (a,b)∈H. f (a,b))) FMB⟩
proof –
obtain D' where D'_uni: ⟨eventually D' uniformity⟩
and ⟨card M' ≤ card M ∧ (∀ m∈M'. D' (g m, g' m)) ⇒ D (sum g M',
sum g' M')⟩
for M' :: ⟨'a set⟩ and g g'
using sum_uniformity[OF plus_cont ⟨eventually D uniformity⟩] by blast
then have D'_sum_D: ⟨(∀ m∈M. D' (g m, g' m)) ⇒ D (sum g M, sum
g' M)⟩ for g g'
by auto

obtain Ha where ⟨Ha a ⊇ Ga a⟩ and Ha_fin: ⟨finite (Ha a)⟩ and Ha_B:
⟨Ha a ⊆ B a⟩
and D'_sum_Ha: ⟨Ha a ⊆ L ⇒ L ⊆ B a ⇒ finite L ⇒ D' (b a, sum
(λb. f (a,b)) L)⟩ if ⟨a ∈ A⟩ for a L
proof –
from sum_b[unfolded tendsto_iff_uniformity, rule_format, OF _ D'_uni[THEN
uniformity_sym]]
obtain Ha0 where ⟨finite (Ha0 a)⟩ and ⟨Ha0 a ⊆ B a⟩
and ⟨Ha0 a ⊆ L ⇒ L ⊆ B a ⇒ finite L ⇒ D' (b a, sum (λb. f (a,b))
L)⟩ if ⟨a ∈ A⟩ for a L
unfolding FB_def eventually_finite_subsets_at_top unfolding prod.case
by metis
moreover define Ha where ⟨Ha a = Ha0 a ∪ Ga a⟩ for a
ultimately show ?thesis
using that[where Ha=Ha]
using Ga_fin Ga_B by auto
qed

have ⟨D (∑ a∈M. b a, ∑ (a,b)∈H. f (a,b))⟩ if ⟨finite H⟩ and ⟨H ⊆ Sigma
M B⟩ and ⟨H ⊇ Sigma M Ha⟩ for H
proof –
define Ha' where ⟨Ha' a = {b | b. (a,b) ∈ H}⟩ for a
have [simp]: ⟨finite (Ha' a)⟩ and [simp]: ⟨Ha' a ⊇ Ha a⟩ and [simp]: ⟨Ha'
a ⊆ B a⟩ if ⟨a ∈ M⟩ for a
unfolding Ha'_def using ⟨finite H⟩ ⟨H ⊆ Sigma M B⟩ ⟨Sigma M Ha
⊆ H⟩ that finite_proj by auto
have ⟨Sigma M Ha' = H⟩
using that by (auto simp: Ha'_def)

```

```

then have *:  $\langle (\sum (a,b) \in H. f (a,b)) = (\sum a \in M. \sum b \in Ha' a. f (a,b)) \rangle$ 
  by (simp add:  $\langle \text{finite } M \rangle$  sum.Sigma)
have  $\langle D' (b a, \text{sum } (\lambda b. f (a,b)) (Ha' a)) \rangle$  if  $\langle a \in M \rangle$  for  $a$ 
  using  $D'_\text{sum\_Ha}$   $\langle M \subseteq A \rangle$  that by auto
then have  $\langle D (\sum a \in M. b a, \sum a \in M. \text{sum } (\lambda b. f (a,b)) (Ha' a)) \rangle$ 
  by (rule_tac  $D'_\text{sum\_D}$ , auto)
with * show ?thesis
  by auto
qed
moreover have  $\langle \text{Sigma } M \text{ Ha } \subseteq \text{Sigma } M B \rangle$ 
  using  $Ha\_B$   $\langle M \subseteq A \rangle$  by auto
ultimately show ?thesis
  unfolding  $FMB\_def$   $\text{eventually\_finite\_subsets\_at\_top}$ 
by (metis (no_types, lifting)  $Ha\_fin$   $\text{finite\_SigmaI}$  subsetD that(2) that(3))
qed
moreover have  $\langle \text{eventually } (\lambda H. D (\sum (a,b) \in H. f (a,b), a)) \text{ FMB} \rangle$ 
  unfolding  $FMB\_def$   $\text{eventually\_finite\_subsets\_at\_top}$ 
proof (rule exI[of _ G], safe)
  fix  $Y$  assume  $Y: \text{finite } Y \text{ } G \subseteq Y \text{ } Y \subseteq \text{Sigma } M B$ 
  thus  $D (\sum (a,b) \in Y. f (a, b), a)$ 
    using  $G\_sum$ [of  $Y$ ]  $Y$  using that(3) by fastforce
qed (use  $\langle \text{finite } G \rangle$   $\langle G \subseteq \text{Sigma } A B \rangle$  that in auto)
ultimately have  $\langle \forall_F x \text{ in } FMB. E (\text{sum } b M, a) \rangle$ 
  by  $\text{eventually\_elim}$  (use  $DDE'$  in auto)
then show  $\langle E (\text{sum } b M, a) \rangle$ 
  using  $FMB\_def$  by force
qed
then show  $\langle \forall_F x \text{ in } FA. E (\text{sum } b x, a) \rangle$ 
  using  $\langle \text{finite } (fst 'G) \rangle$  and  $\langle fst 'G \subseteq A \rangle$ 
  by (metis (mono_tags, lifting)  $FA\_def$   $\text{eventually\_finite\_subsets\_at\_top}$ )
qed
then show ?thesis
  by (simp add:  $FA\_def$   $\text{has\_sum\_def}$ )
qed

```

lemma summable_on_Sigma:

```

fixes  $A :: 'a \text{ set}$  and  $B :: 'a \Rightarrow 'b \text{ set}$ 
  and  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{ \text{comm\_monoid\_add}, \text{t2\_space}, \text{uniform\_space} \}$ 
assumes  $\text{plus\_cont}: \langle \text{uniformly\_continuous\_on } UNIV (\lambda(x::'c,y). x+y) \rangle$ 
assumes  $\text{summableAB}: \langle \lambda(x,y). f x y \text{ summable\_on } (\text{Sigma } A B) \rangle$ 
assumes  $\text{summableB}: \langle \bigwedge x. x \in A \Longrightarrow (f x) \text{ summable\_on } (B x) \rangle$ 
shows  $\langle (\lambda x. \text{infsun } (f x) (B x)) \text{ summable\_on } A \rangle$ 
proof –
  from  $\text{summableAB}$  obtain  $a$  where  $a: \langle ((\lambda(x,y). f x y) \text{ has\_sum } a) (\text{Sigma } A B) \rangle$ 
  using  $\text{has\_sum\_infsun}$  by blast
  from  $\text{summableB}$  have  $b: \langle \bigwedge x. x \in A \Longrightarrow (f x \text{ has\_sum } \text{infsun } (f x) (B x)) (B x) \rangle$ 
  by (auto intro!:  $\text{has\_sum\_infsun}$ )

```

```

show ?thesis
  using plus_cont a b
  by (smt (verit) has_sum_Sigma[where f= $\langle \lambda(x,y). f x y \rangle$ ] has_sum_cong
old.prod.case summable_on_def)
qed

```

```

lemma infsum_Sigma:
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f :: 'a  $\times$  'b  $\Rightarrow$  'c::{comm_monoid_add, t2_space, uniform_space}
  assumes plus_cont:  $\langle$ uniformly_continuous_on UNIV  $(\lambda(x::'c,y). x+y)$  $\rangle$ 
  assumes summableAB: f summable_on (Sigma A B)
  assumes summableB:  $\langle \bigwedge x. x \in A \implies (\lambda y. f(x,y))$  summable_on (B x) $\rangle$ 
  shows infsum f (Sigma A B) = infsum  $(\lambda x. \text{infsum } (\lambda y. f(x,y)) (B x))$  A
proof -
  from summableAB have a:  $\langle$ f has_sum infsum f (Sigma A B) $\rangle$  (Sigma A B)
  using has_sum_infsum by blast
  from summableB have b:  $\langle \bigwedge x. x \in A \implies ((\lambda y. f(x,y))$  has_sum infsum  $(\lambda y. f$ 
(x,y)) (B x)) (B x) $\rangle$ 
  by (auto intro!: has_sum_infsum)
  show ?thesis
  using plus_cont a b by (auto intro: infsumI[symmetric] has_sum_Sigma simp:
summable_on_def)
qed

```

```

lemma infsum_Sigma':
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::{comm_monoid_add, t2_space, uniform_space}
  assumes plus_cont:  $\langle$ uniformly_continuous_on UNIV  $(\lambda(x::'c,y). x+y)$  $\rangle$ 
  assumes summableAB:  $(\lambda(x,y). f x y)$  summable_on (Sigma A B)
  assumes summableB:  $\langle \bigwedge x. x \in A \implies (f x)$  summable_on (B x) $\rangle$ 
  shows  $\langle$ infsum  $(\lambda x. \text{infsum } (f x) (B x))$  A = infsum  $(\lambda(x,y). f x y)$  (Sigma A
B) $\rangle$ 
  using infsum_Sigma[of  $\langle \lambda(x,y). f x y \rangle$  A B]
  using assms by auto

```

A special case of `infsum_Sigma` etc. for Banach spaces. It has less premises.

```

lemma
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::banach
  assumes [simp]:  $(\lambda(x,y). f x y)$  summable_on (Sigma A B)
  shows infsum_Sigma'_banach:  $\langle$ infsum  $(\lambda x. \text{infsum } (f x) (B x))$  A = infsum
 $(\lambda(x,y). f x y)$  (Sigma A B) $\rangle$  (is ?thesis1)
  and summable_on_Sigma_banach:  $\langle$  $(\lambda x. \text{infsum } (f x) (B x))$  summable_on A $\rangle$ 
(is ?thesis2)
proof -
  have fsum:  $\langle$ f x summable_on (B x) $\rangle$  if  $\langle x \in A \rangle$  for x
  proof -
  from assms
  have  $\langle$  $(\lambda(x,y). f x y)$  summable_on (Pair x ' B x) $\rangle$ 

```

```

    by (meson image_subset_iff summable_on_subset_banach mem_Sigma_iff
that)
    then have ⟨((λ(x,y). f x y) ∘ Pair x) summable_on (B x)⟩
    by (metis summable_on_reindex inj_on_def prod.inject)
    then show ?thesis
    by (auto simp: o_def)
qed
show ?thesis1
using fsum assms infsum_Sigma' isUCont_plus by blast
show ?thesis2
using fsum assms isUCont_plus summable_on_Sigma by blast
qed

```

```

lemma infsum_Sigma_banach:
  fixes A :: 'a set and B :: 'a ⇒ 'b set
  and f :: 'a × 'b ⇒ 'c::banach
  assumes [simp]: f summable_on (Sigma A B)
  shows ⟨infsum (λx. infsum (λy. f (x,y)) (B x)) A = infsum f (Sigma A B)⟩
  using assms by (simp add: infsum_Sigma'_banach)

```

```

lemma infsum_swap:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a ⇒ 'b ⇒ 'c::{comm_monoid_add,t2_space,uniform_space}
  assumes plus_cont: ⟨uniformly_continuous_on UNIV (λ(x::'c,y). x+y)⟩
  assumes ⟨λ(x, y). f x y summable_on (A × B)⟩
  assumes ⟨∧a. a∈A ⇒ (f a) summable_on B⟩
  assumes ⟨∧b. b∈B ⇒ (λa. f a b) summable_on A⟩
  shows ⟨infsum (λx. infsum (λy. f x y) B) A = infsum (λy. infsum (λx. f x y)
A) B⟩
proof -
  have (λ(x, y). f y x) ∘ prod.swap summable_on A × B
  by (simp add: assms(2) summable_on_cong)
  then have fyx: ⟨λ(x, y). f y x summable_on (B × A)⟩
  by (metis has_sum_reindex infsum_reindex inj_swap product_swap summable_iff_has_sum_infsum)
  have ⟨infsum (λx. infsum (λy. f x y) B) A = infsum (λ(x,y). f x y) (A × B)⟩
  using assms infsum_Sigma' by blast
  also have ⟨... = infsum (λ(x,y). f y x) (B × A)⟩
  apply (subst product_swap[symmetric])
  apply (subst infsum_reindex)
  using assms by (auto simp: o_def)
  also have ⟨... = infsum (λy. infsum (λx. f x y) A) B⟩
  by (smt (verit) fyx assms(1) assms(4) infsum_Sigma' infsum_cong)
  finally show ?thesis .
qed

```

```

lemma infsum_swap_banach:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a ⇒ 'b ⇒ 'c::banach
  assumes ⟨λ(x, y). f x y summable_on (A × B)⟩

```

shows $\text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda y. \text{infsum } (\lambda x. f x y) A) B$

proof –

have \S : $\langle (\lambda(x, y). f y x) \text{ summable_on } (B \times A) \rangle$
by (*metis* (*mono_tags*, *lifting*) *assms* *case_swap inj_swap o_apply product_swap summable_on_cong summable_on_reindex*)
have $\langle \text{infsum } (\lambda x. \text{infsum } (\lambda y. f x y) B) A = \text{infsum } (\lambda(x, y). f x y) (A \times B) \rangle$
using *assms infsum_Sigma'_banach* **by** *blast*
also have $\langle \dots = \text{infsum } (\lambda(x, y). f y x) (B \times A) \rangle$
apply (*subst product_swap[symmetric]*)
apply (*subst infsum_reindex*)
using *assms* **by** (*auto simp: o_def*)
also have $\langle \dots = \text{infsum } (\lambda y. \text{infsum } (\lambda x. f x y) A) B \rangle$
by (*metis* (*mono_tags*, *lifting*) \S *infsum_Sigma'_banach infsum_cong*)
finally show *?thesis* .

qed

lemma *nonneg_infsum_le_0D*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_ab_group_add, linorder_topology}\}$

assumes $\text{infsum } f A \leq 0$

and *abs_sum*: $f \text{ summable_on } A$

and *nneg*: $\bigwedge x. x \in A \implies f x \geq 0$

and $x \in A$

shows $f x = 0$

proof (*rule ccontr*)

assume $\langle f x \neq 0 \rangle$

have *ex*: $\langle f \text{ summable_on } (A - \{x\}) \rangle$

by (*rule summable_on_cofin_subset*) (*use assms in auto*)

have *pos*: $\langle \text{infsum } f (A - \{x\}) \geq 0 \rangle$

by (*rule infsum_nonneg*) (*use nneg in auto*)

have [*trans*]: $\langle x \geq y \implies y > z \implies x > z \rangle$ **for** $x y z :: 'b$ **by** *auto*

have $\langle \text{infsum } f A = \text{infsum } f (A - \{x\}) + \text{infsum } f \{x\} \rangle$

by (*subst infsum_Un_disjoint[symmetric]*) (*use assms ex in <auto simp: insert_absorb>*)

also have $\langle \dots \geq \text{infsum } f \{x\} \rangle$ (**is** $\langle _ \geq \dots \rangle$)

using *pos* **by** (*rule add_increasing*) *simp*

also have $\langle \dots = f x \rangle$ (**is** $\langle _ = \dots \rangle$)

by (*subst infsum_finite*) *auto*

also have $\langle \dots > 0 \rangle$

using $\langle f x \neq 0 \rangle$ *assms(4)* *nneg* **by** *fastforce*

finally show *False*

using *assms* **by** *auto*

qed

lemma *nonneg_has_sum_le_0D*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, ordered_ab_group_add, linorder_topology}\}$

assumes ($f \text{ has_sum } a$) $A \langle a \leq 0 \rangle$

and $\bigwedge x. x \in A \implies f x \geq 0$
and $x \in A$
shows $f x = 0$
by (*metis* *assms* *infsunI* *nonneg_infsun_le_0D* *summable_on_def*)

lemma *has_sum_cmult_left*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle f \text{ has_sum } a \rangle A$
shows $((\lambda x. f x * c) \text{ has_sum } (a * c)) A$
using *assms* *tendsto_mult_right*
by (*force* *simp* *add*: *has_sum_def* *sum_distrib_right*)

lemma *infsun_cmult_left*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{t2_space}, \text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle c \neq 0 \implies f \text{ summable_on } A \rangle$
shows $\text{infsun } (\lambda x. f x * c) A = \text{infsun } f A * c$
using *assms* *has_sum_cmult_left* *infsunI* *summable_iff_has_sum_infsun* **by**
fastforce

lemma *summable_on_cmult_left*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{t2_space}, \text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle f \text{ summable_on } A \rangle$
shows $(\lambda x. f x * c) \text{ summable_on } A$
using *assms* *summable_on_def* *has_sum_cmult_left* **by** *blast*

lemma *has_sum_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle f \text{ has_sum } a \rangle A$
shows $((\lambda x. c * f x) \text{ has_sum } (c * a)) A$
using *assms* *tendsto_mult_left*
by (*force* *simp* *add*: *has_sum_def* *sum_distrib_left*)

lemma *infsun_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{t2_space}, \text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle c \neq 0 \implies f \text{ summable_on } A \rangle$
shows $\langle \text{infsun } (\lambda x. c * f x) A = c * \text{infsun } f A \rangle$
using *assms* *has_sum_cmult_right* *infsunI* *summable_iff_has_sum_infsun* **by**
fastforce

lemma *summable_on_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{t2_space}, \text{topological_semigroup_mult}, \text{semiring_0}\}$
assumes $\langle f \text{ summable_on } A \rangle$
shows $(\lambda x. c * f x) \text{ summable_on } A$
using *assms* *summable_on_def* *has_sum_cmult_right* **by** *blast*

lemma *summable_on_cmult_left'*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{t2_space}, \text{topological_semigroup_mult}, \text{division_ring}\}$
assumes $\langle c \neq 0 \rangle$
shows $(\lambda x. f x * c) \text{ summable_on } A \longleftrightarrow f \text{ summable_on } A$

proof

```

assume ⟨f summable_on A⟩
then show ⟨(λx. f x * c) summable_on A⟩
  by (rule summable_on_cmult_left)
next
assume ⟨(λx. f x * c) summable_on A⟩
then have ⟨(λx. f x * c * inverse c) summable_on A⟩
  by (rule summable_on_cmult_left)
then show ⟨f summable_on A⟩
  by (smt (verit, del_insts) assms divide_inverse nonzero_divide_eq_eq summable_on_cong)
qed

```

lemma *summable_on_cmult_right*':

```

fixes f :: 'a ⇒ 'b :: {t2_space, topological_semigroup_mult, division_ring}
assumes ⟨c ≠ 0⟩
shows (λx. c * f x) summable_on A ↔ f summable_on A
  by (metis (no_types, lifting) assms left_inverse mult_assoc mult_1 summable_on_cmult_right summable_on_cong)

```

lemma *infsum_cmult_left*':

```

fixes f :: 'a ⇒ 'b :: {t2_space, topological_semigroup_mult, division_ring}
shows infsum (λx. f x * c) A = infsum f A * c
by (metis (full_types) infsum_cmult_left infsum_not_exists mult_eq_0_iff summable_on_cmult_left)

```

lemma *infsum_cmult_right*':

```

fixes f :: 'a ⇒ 'b :: {t2_space, topological_semigroup_mult, division_ring}
shows infsum (λx. c * f x) A = c * infsum f A
  by (metis (full_types) infsum_cmult_right infsum_not_exists mult_eq_0_iff summable_on_cmult_right)

```

lemma *has_sum_constant*[*simp*]:

```

assumes ⟨finite F⟩
shows ⟨(λ_. c) has_sum of_nat (card F) * c⟩ F
by (metis assms has_sum_finite sum_constant)

```

lemma *infsum_constant*[*simp*]:

```

assumes ⟨finite F⟩
shows ⟨infsum (λ_. c) F = of_nat (card F) * c⟩
by (simp add: assms)

```

lemma *infsum_diverge_constant*:

— This probably does not really need all of *archimedean_field* but Isabelle/HOL has no type class such as, e.g., "archimedean ring".

```

fixes c :: 'a::{archimedean_field, comm_monoid_add, linorder_topology, topological_semigroup_mult}
assumes ⟨infinite A⟩ and ⟨c ≠ 0⟩
shows ⟨¬ (λ_. c) summable_on A⟩
proof (rule notI)
  assume ⟨(λ_. c) summable_on A⟩

```



```

then have  $\langle \lambda_. \text{inverse } c * c \rangle \text{ summable\_on } A$ 
  by (rule summable_on_cmult_right)
then have [simp]:  $\langle \lambda_. 1 :: 'a \rangle \text{ summable\_on } A$ 
  using assms by auto
have  $\langle \text{infsum } (\lambda_. 1) A \geq d \rangle$  for  $d :: 'a$ 
proof –
  obtain  $n :: \text{nat}$  where  $\langle \text{of\_nat } n \geq d \rangle$ 
    by (meson real_arch_simple)
  from assms
  obtain  $F$  where  $\langle F \subseteq A \rangle$  and  $\langle \text{finite } F \rangle$  and  $\langle \text{card } F = n \rangle$ 
    by (meson infinite_arbitrarily_large)
  note  $\langle d \leq \text{of\_nat } n \rangle$ 
  also have  $\langle \text{of\_nat } n = \text{infsum } (\lambda_. 1 :: 'a) F \rangle$ 
    by (simp add:  $\langle \text{card } F = n \rangle \langle \text{finite } F \rangle$ )
  also have  $\langle \dots \leq \text{infsum } (\lambda_. 1 :: 'a) A \rangle$ 
    apply (rule infsum_mono_neutral)
    using  $\langle \text{finite } F \rangle \langle F \subseteq A \rangle$  by auto
  finally show ?thesis .
qed
then show False
  by (meson linordered_field_no_ub_not_less)
qed

```

lemma *has_sum_constant_archimedean*[simp]:

— This probably does not really need all of *archimedean_field* but Isabelle/HOL has no type class such as, e.g., "archimedean ring".

```

fixes  $c :: \langle 'a :: \{ \text{archimedean\_field, comm\_monoid\_add, linorder\_topology, topological\_semigroup\_mult} \} \rangle$ 
shows  $\langle \text{infsum } (\lambda_. c) A = \text{of\_nat } (\text{card } A) * c \rangle$ 
by (metis infsum_0 infsum_constant infsum_diverge_constant infsum_not_exists sum.infinite sum_constant)

```

lemma *has_sum_uminus*:

```

fixes  $f :: \langle 'a \Rightarrow 'b :: \text{topological\_ab\_group\_add} \rangle$ 
shows  $\langle ((\lambda x. - f x) \text{ has\_sum } a) A \longleftrightarrow (f \text{ has\_sum } (- a)) A \rangle$ 
by (auto simp add: sum_negf[abs_def] tendsto_minus_cancel_left has_sum_def)

```

lemma *summable_on_uminus*:

```

fixes  $f :: \langle 'a \Rightarrow 'b :: \text{topological\_ab\_group\_add} \rangle$ 
shows  $\langle (\lambda x. - f x) \text{ summable\_on } A \longleftrightarrow f \text{ summable\_on } A \rangle$ 
by (metis summable_on_def has_sum_uminus verit_minus_simplify(4))

```

lemma *infsum_uminus*:

```

fixes  $f :: \langle 'a \Rightarrow 'b :: \{ \text{topological\_ab\_group\_add, t2\_space} \} \rangle$ 
shows  $\langle \text{infsum } (\lambda x. - f x) A = - \text{infsum } f A \rangle$ 
by (metis (full_types) add.inverse_inverse add.inverse_neutral infsumI infsum_def has_sum_infsum has_sum_uminus)

```

lemma *has_sum_le_finite_sums*:

```

fixes a :: ⟨'a::{comm_monoid_add,topological_space,linorder_topology}⟩
assumes ⟨f has_sum a⟩ A
assumes ⟨ $\bigwedge F. \text{finite } F \implies F \subseteq A \implies \text{sum } f F \leq b$ ⟩
shows ⟨a ≤ b⟩
by (metis assms eventually_finite_subsets_at_top_weakI finite_subsets_at_top_neq_bot
has_sum_def tendsto_upperbound)

```

```

lemma infsum_le_finite_sums:
fixes b :: ⟨'a::{comm_monoid_add,topological_space,linorder_topology}⟩
assumes ⟨f summable_on A⟩
assumes ⟨ $\bigwedge F. \text{finite } F \implies F \subseteq A \implies \text{sum } f F \leq b$ ⟩
shows ⟨infsum f A ≤ b⟩
by (meson assms has_sum_infsum has_sum_le_finite_sums)

```

```

lemma summable_on_scaleR_left [intro]:
fixes c :: ⟨'a :: real_normed_vector⟩
assumes c ≠ 0  $\implies$  f summable_on A
shows (λx. f x *R c) summable_on A
proof (cases ⟨c = 0⟩)
case False
then have (λy. y *R c) ∘ f summable_on A
using assms by (auto simp add: scaleR_left.additive_axioms summable_on_comm_additive)
then show ?thesis
by (metis (mono_tags, lifting) comp_apply summable_on_cong)
qed auto

```

```

lemma summable_on_scaleR_right [intro]:
fixes f :: ⟨'a  $\Rightarrow$  'b :: real_normed_vector⟩
assumes c ≠ 0  $\implies$  f summable_on A
shows (λx. c *R f x) summable_on A
proof (cases ⟨c = 0⟩)
case False
then have (*R) c ∘ f summable_on A
using assms by (auto simp add: scaleR_right.additive_axioms summable_on_comm_additive)
then show ?thesis
by (metis (mono_tags, lifting) comp_apply summable_on_cong)
qed auto

```

```

lemma infsum_scaleR_left:
fixes c :: ⟨'a :: real_normed_vector⟩
assumes c ≠ 0  $\implies$  f summable_on A
shows infsum (λx. f x *R c) A = infsum f A *R c
proof (cases ⟨c = 0⟩)
case False
then have infsum ((λy. y *R c) ∘ f) A = infsum f A *R c
using assms by (auto simp add: scaleR_left.additive_axioms infsum_comm_additive)
then show ?thesis

```

by (metis (mono_tags, lifting) comp_apply infsum_cong)
qed auto

lemma infsum_scaleR_right:

fixes $f :: \langle 'a \Rightarrow 'b :: \text{real_normed_vector} \rangle$

shows $\text{infsum } (\lambda x. c *_R f x) A = c *_R \text{infsum } f A$

proof -

consider (summable) $\langle f \text{ summable_on } A \rangle \mid (c0) \langle c = 0 \rangle \mid (\text{not_summable}) \langle \neg f \text{ summable_on } A \rangle \langle c \neq 0 \rangle$

by auto

then show ?thesis

proof cases

case summable

then have $\text{infsum } ((*_R) c \circ f) A = c *_R \text{infsum } f A$

by (auto simp add: scaleR_right.additive_axioms infsum_comm_additive)

then show ?thesis

by (metis (mono_tags, lifting) comp_apply infsum_cong)

next

case c0

then show ?thesis by auto

next

case not_summable

have $\langle \neg (\lambda x. c *_R f x) \text{ summable_on } A \rangle$

proof (rule notI)

assume $\langle (\lambda x. c *_R f x) \text{ summable_on } A \rangle$

then have $\langle (\lambda x. \text{inverse } c *_R c *_R f x) \text{ summable_on } A \rangle$

using summable_on_scaleR_right by blast

with not_summable show False

by simp

qed

then show ?thesis

by (simp add: infsum_not_exists not_summable(1))

qed

qed

lemma infsum_Un_Int:

fixes $f :: 'a \Rightarrow 'b :: \{\text{topological_ab_group_add, } t2_space\}$

assumes $f \text{ summable_on } A - B$ $f \text{ summable_on } B - A$ $\langle f \text{ summable_on } A \cap B \rangle$

shows $\text{infsum } f (A \cup B) = \text{infsum } f A + \text{infsum } f B - \text{infsum } f (A \cap B)$

proof -

obtain $\langle f \text{ summable_on } A \rangle \langle f \text{ summable_on } B \rangle$

using assms by (metis Int_Diff_Un Int_Diff_disjoint inf_commute summable_on_Un_disjoint)

then have $\langle \text{infsum } f (A \cup B) = \text{infsum } f A + \text{infsum } f (B - A) \rangle$

using assms(2) infsum_Un_disjoint by fastforce

moreover have $\langle \text{infsum } f (B - A) = \text{infsum } f B - \text{infsum } f (A \cap B) \rangle$

using assms by (metis Diff_Int2 Un_Int_eq(2) $\langle f \text{ summable_on } B \rangle$ inf_le2 infsum_Diff)

ultimately show *?thesis*
 by *auto*
 qed

lemma *inj_combinator'*:
 assumes $x \notin F$
 shows $\langle \text{inj_on } (\lambda(g, y). g(x := y)) \text{ } (Pi_E F B \times B x) \rangle$
 proof –
 have $\text{inj_on } ((\lambda(y, g). g(x := y)) \circ \text{prod.swap}) \text{ } (Pi_E F B \times B x)$
 using *inj_combinator[of x F B]* *assms* by (*intro comp_inj_on*) (*auto simp: product_swap*)
 thus *?thesis*
 by (*simp add: o_def*)
 qed

lemma *infsum_prod_PiE*:
 — See also *infsum_prod_PiE_abs* below with incomparable premises.
 fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{comm_monoid_mult, topological_semigroup_mult, division_ring, banach}\}$
 assumes *finite: finite A*
 assumes $\bigwedge x. x \in A \implies f x \text{ summable_on } B x$
 assumes $(\lambda g. \prod x \in A. f x (g x)) \text{ summable_on } (Pi_E A B)$
 shows $\text{infsum } (\lambda g. \prod x \in A. f x (g x)) \text{ } (Pi_E A B) = (\prod x \in A. \text{infsum } (f x) \text{ } (B x))$
 proof (*use finite assms(2–) in induction*)
 case *empty*
 then show *?case*
 by *auto*
 next
 case (*insert x F*)
 have *pi*: $\langle Pi_E (\text{insert } x F) B = (\lambda(g, y). g(x := y)) \text{ } (Pi_E F B \times B x) \rangle$
 unfolding *PiE_insert_eq*
 by (*subst swap_product [symmetric]*) (*simp add: image_image case_prod_unfold*)
 have *prod*: $\langle (\prod x' \in F. f x' ((p(x := y)) x')) = (\prod x' \in F. f x' (p x')) \rangle$ for $p y$
 by (*rule prod.cong*) (*use insert.hyps in auto*)
 have *inj*: $\langle \text{inj_on } (\lambda(g, y). g(x := y)) \text{ } (Pi_E F B \times B x) \rangle$
 using $\langle x \notin F \rangle$ by (*rule inj_combinator'*)

 have *summable1*: $\langle (\lambda g. \prod x \in \text{insert } x F. f x (g x)) \text{ summable_on } Pi_E (\text{insert } x F) B \rangle$
 using *insert.premis(2)* .
 also have $\langle Pi_E (\text{insert } x F) B = (\lambda(g, y). g(x := y)) \text{ } (Pi_E F B \times B x) \rangle$
 by (*simp only: pi*)
 also have $(\lambda g. \prod x \in \text{insert } x F. f x (g x)) \text{ summable_on } \dots \iff$
 $((\lambda g. \prod x \in \text{insert } x F. f x (g x)) \circ (\lambda(g, y). g(x := y))) \text{ summable_on}$
 $(Pi_E F B \times B x)$
 using *inj* by (*rule summable_on_reindex*)
 also have $(\prod z \in F. f z ((g(x := y)) z)) = (\prod z \in F. f z (g z))$ for $g y$
 using *insert.hyps* by (*intro prod.cong*) *auto*

hence $((\lambda g. \prod x \in \text{insert } x \ F. f \ x \ (g \ x)) \circ (\lambda (g,y). g(x:=y))) =$
 $(\lambda (p, y). f \ x \ y * (\prod x' \in F. f \ x' \ (p \ x')))$
using *insert.hyps* **by** (*auto simp: fun_eq_iff cong: prod.cong_simp*)
finally have *summable2*: $\langle (\lambda (p, y). f \ x \ y * (\prod x' \in F. f \ x' \ (p \ x'))) \text{ summable_on } Pi_E \ F \ B \times B \ x \rangle$.

then have $\langle (\lambda p. \sum_{\infty} y \in B \ x. f \ x \ y * (\prod x' \in F. f \ x' \ (p \ x'))) \text{ summable_on } Pi_E \ F \ B \rangle$
by (*rule summable_on_Sigma_banach*)
then have $\langle (\lambda p. (\sum_{\infty} y \in B \ x. f \ x \ y) * (\prod x' \in F. f \ x' \ (p \ x'))) \text{ summable_on } Pi_E \ F \ B \rangle$
by (*metis (mono_tags, lifting) infsum_cmult_left' infsum_cong summable_on_cong*)
then have *summable3*: $\langle (\lambda p. (\prod x' \in F. f \ x' \ (p \ x'))) \text{ summable_on } Pi_E \ F \ B \rangle$ **if**
 $\langle (\sum_{\infty} y \in B \ x. f \ x \ y) \neq 0 \rangle$
using *summable_on_cmult_right' that by blast*

have $\langle (\sum_{\infty} g \in Pi_E \ (\text{insert } x \ F) \ B. \prod x \in \text{insert } x \ F. f \ x \ (g \ x))$
 $= (\sum_{\infty} (p,y) \in Pi_E \ F \ B \times B \ x. \prod x' \in \text{insert } x \ F. f \ x' \ ((p(x:=y)) \ x')) \rangle$
by (*smt (verit, ccfv_SIG) comp_apply infsum_cong infsum_reindex inj pi prod.cong_split_def*)
also have $\langle \dots = (\sum_{\infty} (p, y) \in Pi_E \ F \ B \times B \ x. f \ x \ y * (\prod x' \in F. f \ x' \ ((p(x:=y)) \ x'))) \rangle$
using *insert.hyps* **by** *auto*
also have $\langle \dots = (\sum_{\infty} (p, y) \in Pi_E \ F \ B \times B \ x. f \ x \ y * (\prod x' \in F. f \ x' \ (p \ x'))) \rangle$
using *prod* **by** *presburger*
also have $\langle \dots = (\sum_{\infty} p \in Pi_E \ F \ B. \sum_{\infty} y \in B \ x. f \ x \ y * (\prod x' \in F. f \ x' \ (p \ x'))) \rangle$
using *infsum_Sigma'_banach summable2* **by** *force*
also have $\langle \dots = (\sum_{\infty} y \in B \ x. f \ x \ y) * (\sum_{\infty} p \in Pi_E \ F \ B. \prod x' \in F. f \ x' \ (p \ x')) \rangle$
by (*smt (verit) infsum_cmult_left' infsum_cmult_right' infsum_cong*)
also have $\langle \dots = (\prod x \in \text{insert } x \ F. \text{infsum } (f \ x) \ (B \ x)) \rangle$
using *insert summable3* **by** *auto*
finally show *?case*
by *simp*

qed

lemma *infsum_prod_PiE_abs*:
— See also *infsum_prod_PiE* above with incomparable premises.
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{banach, real_normed_div_algebra, comm_semiring_1}\}$
assumes *finite*: *finite A*
assumes $\bigwedge x. x \in A \implies f \ x \ \text{abs_summable_on } B \ x$
shows $\text{infsum } (\lambda g. \prod x \in A. f \ x \ (g \ x)) \ (Pi_E \ A \ B) = (\prod x \in A. \text{infsum } (f \ x) \ (B \ x))$
proof (*use finite assms(2) in induction*)
case *empty*
then show *?case*
by *auto*
next
case (*insert x A*)

```

have pi: ⟨PiE (insert x F) B = (λ(g,y). g(x:=y)) ‘ (PiE F B × B x)⟩ for x F
and B :: 'a ⇒ 'b set
  unfolding PiE_insert_eq
  by (subst swap_product [symmetric]) (simp add: image_image case_prod_unfold)
have prod: ⟨(∏ x'∈A. f x' ((p(x:=y)) x')) = (∏ x'∈A. f x' (p x'))⟩ for p y
  by (rule prod.cong) (use insert.hyps in auto)
have inj: ⟨inj_on (λ(g, y). g(x := y)) (PiE A B × B x)⟩
  using ⟨x ∉ A⟩ by (rule inj_combinator')

define s where ⟨s x = infsum (λy. norm (f x y)) (B x)⟩ for x

have ⟨(∑ p∈P. norm (∏ x∈F. f x (p x))) ≤ prod s F⟩
  if P: ⟨P ⊆ PiE F B⟩ and [simp]: ⟨finite P⟩ ⟨finite F⟩
  and sum: ⟨∧x. x ∈ F ⇒ f x abs_summable_on B x⟩ for P F
proof -
  define B' where ⟨B' x = {p x | p. p∈P}⟩ for x
  have fin_B'[simp]: ⟨finite (B' x)⟩ for x
    using that by (auto simp: B'_def)
  have [simp]: ⟨finite (PiE F B')⟩
    by (simp add: finite_PiE)
  have [simp]: ⟨P ⊆ PiE F B'⟩
    using that by (auto simp: B'_def)
  have B'B: ⟨B' x ⊆ B x⟩ if ⟨x ∈ F⟩ for x
    unfolding B'_def using P that
    by auto
  have s_bound: ⟨(∑ y∈B' x. norm (f x y)) ≤ s x⟩ if ⟨x ∈ F⟩ for x
    by (metis B'B fin_B' finite_sum_le_has_sum has_sum_infsum norm_ge_zero
s_def sum that)
  have ⟨(∑ p∈P. norm (∏ x∈F. f x (p x))) ≤ (∑ p∈PiE F B'. norm (∏ x∈F. f
x (p x)))⟩
    by (simp add: sum_mono2)
  also have ⟨... = (∑ p∈PiE F B'. ∏ x∈F. norm (f x (p x)))⟩
    by (simp add: prod_norm)
  also have ⟨... = (∏ x∈F. ∑ y∈B' x. norm (f x y))⟩
  proof (use ⟨finite F⟩ in induction)
    case empty
    then show ?case by simp
  next
  case (insert x F)
  have inj: ⟨inj_on (λ(g, y). g(x := y)) (PiE F B' × B' x)⟩
    by (simp add: inj_combinator' insert.hyps)
  then have ⟨(∑ p∈PiE (insert x F) B'. ∏ x∈insert x F. norm (f x (p x)))
= (∑ (p,y)∈PiE F B' × B' x. ∏ x'∈insert x F. norm (f x' ((p(x := y))
x'))))⟩
    by (simp add: pi_sum.reindex case_prod_unfold)
  also have ⟨... = (∑ (p, y)∈PiE F B' × B' x. norm (f x y) * (∏ x'∈F. norm
(f x' (p x'))))⟩
    by (smt (verit, del_insts) fun_upd_apply insert.hyps prod.cong prod.insert
split_def sum.cong)

```

```

    also have ‹... = (∑ y∈B' x. norm (f x y)) * (∑ p∈Pi_E F B'. ∏ x'∈F. norm
(f x' (p x'))›
    by (simp add: sum_product sum.swap [of Pi_E F B'] sum.cartesian_product)
    also have ‹... = (∏ x∈insert x F. ∑ y∈B' x. norm (f x y))›
    using insert by force
    finally show ?case .
qed
also have ‹... ≤ (∏ x∈F. s x)›
    using s_bound by (simp add: prod_mono sum_nonneg)
    finally show ?thesis .
qed
then have bdd_above
  (sum (λg. norm (∏ x∈insert x A. f x (g x))) ‹{F. F ⊆ Pi_E (insert x A) B ∧
finite F}›)
  using insert.hyps insert.prem by (intro bdd_aboveI) blast
  then have ‹(λg. ∏ x∈insert x A. f x (g x)) abs_summable_on Pi_E (insert x A)
B›
    using nonneg_bdd_above_summable_on
    by (metis (mono_tags, lifting) Collect_cong norm_ge_zero)
  also have ‹Pi_E (insert x A) B = (λ(g,y). g(x:=y)) ‹(Pi_E A B × B x)›
    by (simp only: pi)
  also have (λg. ∏ x∈insert x A. f x (g x)) abs_summable_on ... ‹↔›
    ((λg. ∏ x∈insert x A. f x (g x)) ∘ (λ(g,y). g(x:=y))) abs_summable_on
(Pi_E A B × B x)
    using inj by (subst summable_on_reindex) (auto simp: o_def)
  also have (∏ z∈A. f z ((g(x:=y)) z)) = (∏ z∈A. f z (g z)) for g y
    using insert.hyps by (intro prod.cong) auto
  hence ((λg. ∏ x∈insert x A. f x (g x)) ∘ (λ(g,y). g(x:=y))) =
    (λ(p, y). f x y * (∏ x'∈A. f x' (p x')))
    using insert.hyps by (auto simp: fun_eq_iff cong: prod.cong_simp)
  finally have summable2: ‹(λ(p, y). f x y * (∏ x'∈A. f x' (p x'))) abs_summable_on
Pi_E A B × B x› .

have ‹(∑ ∞ g∈Pi_E (insert x A) B. ∏ x∈insert x A. f x (g x))
= (∑ ∞ (p,y)∈Pi_E A B × B x. ∏ x'∈insert x A. f x' ((p(x:=y)) x'))›
  using inj by (simp add: pi infsum_reindex o_def case_prod_unfold)
  also have ‹... = (∑ ∞ (p,y) ∈ Pi_E A B × B x. f x y * (∏ x'∈A. f x' (p x'))›
    using prod insert.hyps by auto
  also have ‹... = (∑ ∞ p∈Pi_E A B. ∑ ∞ y∈B x. f x y * (∏ x'∈A. f x' (p x'))›
    using abs_summable_summable infsum_Sigma'_banach summable2 by fast-
force
  also have ‹... = (∑ ∞ y∈B x. f x y) * (∑ ∞ p∈Pi_E A B. ∏ x'∈A. f x' (p x'))›
    by (smt (verit, best) infsum_cmult_left' infsum_cmult_right' infsum_cong)
  finally show ?case
    by (simp add: insert)
qed

```

6.8.3 Absolute convergence

```

lemma abs_summable_countable:
  assumes ‹f abs_summable_on A›
  shows ‹countable {x∈A. f x ≠ 0}›
proof -
  have fin: ‹finite {x∈A. norm (f x) ≥ t}› if ‹t > 0› for t
  proof (rule ccontr)
    assume *: ‹infinite {x ∈ A. t ≤ norm (f x)}›
    have ‹infsum (λx. norm (f x)) A ≥ b› for b
    proof -
      obtain b' where b': ‹of_nat b' ≥ b / t›
      by (meson real_arch_simple)
      from *
      obtain F where cardF: ‹card F ≥ b'› and ‹finite F› and F: ‹F ⊆ {x ∈ A.
t ≤ norm (f x)}›
      by (meson finite_if_finite_subsets_card_bdd_nle_le)
      have ‹b ≤ of_nat b' * t›
      using b' ‹t > 0› by (simp add: field_simps split: if_splits)
      also have ‹... ≤ of_nat (card F) * t›
      by (simp add: cardF that)
      also have ‹... = sum (λx. t) F›
      by simp
      also have ‹... ≤ sum (λx. norm (f x)) F›
      by (metis (mono_tags, lifting) F in_mono mem_Collect_eq sum_mono)
      also have ‹... = infsum (λx. norm (f x)) F›
      using ‹finite F› by (rule infsum_finite[symmetric])
      also have ‹... ≤ infsum (λx. norm (f x)) A›
      by (rule infsum_mono_neutral) (use ‹finite F› assms F in auto)
      finally show ?thesis .
    qed
  then show False
  by (meson gt_ex linorder_not_less)
qed
have ‹countable (⋃ i∈{1..}. {x∈A. norm (f x) ≥ 1/of_nat i})›
  by (rule countable_UN) (use fin in ‹auto intro!: countable_finite›)
also have ‹... = {x∈A. f x ≠ 0}›
proof safe
  fix x assume x: x ∈ A f x ≠ 0
  define i where i = max 1 (nat (ceiling (1 / norm (f x))))
  have i ≥ 1
  by (simp add: i_def)
  moreover have real i ≥ 1 / norm (f x)
  unfolding i_def by linarith
  hence 1 / real i ≤ norm (f x) using ‹f x ≠ 0›
  by (auto simp: divide_simps mult_ac)
  ultimately show x ∈ (⋃ i∈{1..}. {x ∈ A. 1 / real i ≤ norm (f x)})
  using ‹x ∈ A› by auto
qed auto
finally show ?thesis .

```


qed

lemma *summable_on_iff_abs_summable_on_real*:

fixes $f :: \langle 'a \Rightarrow \text{real} \rangle$
 shows $\langle f \text{ summable_on } A \longleftrightarrow f \text{ abs_summable_on } A \rangle$
 proof (rule iffI)
 assume $\langle f \text{ summable_on } A \rangle$
 define $n \ A_p \ A_n$
 where $\langle n \equiv \lambda x. \text{norm } (f \ x) \rangle$ and $\langle A_p = \{x \in A. f \ x \geq 0\} \rangle$ and $\langle A_n = \{x \in A. f \ x < 0\} \rangle$ for x
 have $A: \langle A_p \cup A_n = A \rangle \langle A_p \cap A_n = \{\} \rangle$
 by (auto simp: $A_p_def \ A_n_def$)
 from $\langle f \text{ summable_on } A \rangle$ have $\langle f \text{ summable_on } A_p \rangle \langle f \text{ summable_on } A_n \rangle$
 using $A_p_def \ A_n_def \text{ summable_on_subset_banach}$ by fastforce+
 then have $\langle n \text{ summable_on } A_p \rangle$
 by (smt (verit) $A_p_def \ n_def \text{ mem_Collect_eq \ real_norm_def \ summable_on_cong}$)
 moreover have $\langle n \text{ summable_on } A_n \rangle$
 by (smt (verit, best) $\langle f \text{ summable_on } A_n \rangle \text{ summable_on_uminus } A_n_def \ n_def \text{ summable_on_cong \ mem_Collect_eq \ real_norm_def}$)
 ultimately show $\langle n \text{ summable_on } A \rangle$
 using $A \text{ summable_on_Un_disjoint}$ by blast
 next
 show $\langle f \text{ abs_summable_on } A \implies f \text{ summable_on } A \rangle$
 using $\text{abs_summable_summable}$ by blast
 qed

lemma *abs_summable_on_Sigma_iff*:

shows $f \text{ abs_summable_on } \text{Sigma } A \ B \longleftrightarrow$
 $(\forall x \in A. (\lambda y. f \ (x, y)) \text{ abs_summable_on } B \ x) \wedge$
 $((\lambda x. \text{infsum } (\lambda y. \text{norm } (f \ (x, y))) \ (B \ x)) \text{ abs_summable_on } A)$
 proof (intro iffI conjI ballI)
 assume *asm*: $\langle f \text{ abs_summable_on } \text{Sigma } A \ B \rangle$
 then have $\langle (\lambda x. \text{infsum } (\lambda y. \text{norm } (f \ (x, y))) \ (B \ x)) \text{ summable_on } A \rangle$
 by (simp add: $\text{cond_case_prod_eta \ summable_on_Sigma_banach}$)
 then show $\langle (\lambda x. \sum_{\infty y \in B} x. \text{norm } (f \ (x, y))) \text{ abs_summable_on } A \rangle$
 using $\text{summable_on_iff_abs_summable_on_real}$ by force

 show $\langle (\lambda y. f \ (x, y)) \text{ abs_summable_on } B \ x \rangle$ if $\langle x \in A \rangle$ for x
 proof -
 from *asm* have $\langle f \text{ abs_summable_on } \text{Pair } x \ ' B \ x \rangle$
 by (simp add: $\text{image_subset_iff \ summable_on_subset_banach \ that}$)
 then show ?thesis
 by (metis (mono_tags, lifting) $\text{o_def \ inj_on_def \ summable_on_reindex \ prod.inject \ summable_on_cong}$)
 qed
 next
 assume *asm*: $\langle (\forall x \in A. (\lambda xa. f \ (x, xa)) \text{ abs_summable_on } B \ x) \wedge$
 $(\lambda x. \sum_{\infty y \in B} x. \text{norm } (f \ (x, y))) \text{ abs_summable_on } A \rangle$

```

have ⟨(∑ xy∈F. norm (f xy)) ≤ (∑ ∞x∈A. ∑ ∞y∈B x. norm (f (x, y)))⟩
  if ⟨F ⊆ Sigma A B⟩ and [simp]: ⟨finite F⟩ for F
proof -
  have [simp]: ⟨(SIGMA x:fst ' F. {y. (x, y) ∈ F}) = F⟩
    by (auto intro!: set_eqI simp add: Domain.DomainI fst_eq_Domain)
  have [simp]: ⟨finite {y. (x, y) ∈ F}⟩ for x
    by (metis ⟨finite F⟩ Range.intros finite_Range finite_subset mem_Collect_eq
subsetI)
  have ⟨(∑ xy∈F. norm (f xy)) = (∑ x∈fst ' F. ∑ y∈{y. (x,y)∈F}. norm (f
(x,y)))⟩
    by (simp add: sum.Sigma)
  also have ⟨... = (∑ ∞x∈fst ' F. ∑ ∞y∈{y. (x,y)∈F}. norm (f (x,y)))⟩
    by auto
  also have ⟨... ≤ (∑ ∞x∈fst ' F. ∑ ∞y∈B x. norm (f (x,y)))⟩
    using asm that(1) by (intro infsum_mono infsum_mono_neutral) auto
  also have ⟨... ≤ (∑ ∞x∈A. ∑ ∞y∈B x. norm (f (x,y)))⟩
    by (rule infsum_mono_neutral) (use asm that(1) in ⟨auto simp add: inf-
sum_nonneg⟩)
  finally show ?thesis .
qed
then show ⟨f abs_summable_on Sigma A B⟩
  by (intro nonneg_bdd_above_summable_on) (auto simp: bdd_above_def)
qed

```

lemma *abs_summable_on_comparison_test*:

```

assumes g abs_summable_on A
assumes ∧x. x ∈ A ⇒ norm (f x) ≤ norm (g x)
shows f abs_summable_on A
proof (rule nonneg_bdd_above_summable_on)
  show bdd_above (sum (λx. norm (f x)) ' {F. F ⊆ A ∧ finite F})
  proof (rule bdd_aboveI2)
    fix F assume F: F ∈ {F. F ⊆ A ∧ finite F}
    have ⟨sum (λx. norm (f x)) F ≤ sum (λx. norm (g x)) F⟩
      using assms F by (intro sum_mono) auto
    also have ⟨... = infsum (λx. norm (g x)) F⟩
      using F by simp
    also have ⟨... ≤ infsum (λx. norm (g x)) A⟩
      by (smt (verit) F assms(1) infsum_mono2 mem_Collect_eq norm_ge_zero
summable_on_subset_banach)
    finally show (∑ x∈F. norm (f x)) ≤ (∑ ∞x∈A. norm (g x)) .
  qed
qed auto

```

lemma *abs_summable_iff_bdd_above*:

```

fixes f :: ⟨'a ⇒ 'b::real_normed_vector⟩
shows ⟨f abs_summable_on A ⟷ bdd_above (sum (λx. norm (f x)) ' {F. F ⊆ A
∧ finite F})⟩
proof (rule iffI)
  assume ⟨f abs_summable_on A⟩

```

```

show ‹bdd_above (sum (λx. norm (f x)) ‹{F. F ⊆ A ∧ finite F}›)›
proof (rule bdd_aboveI2)
  fix F assume F: F ∈ {F. F ⊆ A ∧ finite F}
  show (∑ x∈F. norm (f x)) ≤ (∑ ∞ x∈A. norm (f x))
    by (rule finite_sum_le_infsum) (use ‹f abs_summable_on A› F in auto)
qed
next
assume ‹bdd_above (sum (λx. norm (f x)) ‹{F. F ⊆ A ∧ finite F}›)›
then show ‹f abs_summable_on A›
  by (simp add: nonneg_bdd_above_summable_on)
qed

lemma abs_summable_product:
  fixes x :: 'a ⇒ 'b :: {real_normed_div_algebra, banach, second_countable_topology}
  assumes x2_sum: (λi. (x i) * (x i)) abs_summable_on A
    and y2_sum: (λi. (y i) * (y i)) abs_summable_on A
  shows (λi. x i * y i) abs_summable_on A
proof (rule nonneg_bdd_above_summable_on)
  show bdd_above (sum (λxa. norm (x xa * y xa)) ‹{F. F ⊆ A ∧ finite F}›)
  proof (rule bdd_aboveI2)
    fix F assume F: ‹F ∈ {F. F ⊆ A ∧ finite F}›
    then have r1: finite F and b4: F ⊆ A
      by auto
    have a1: (∑ ∞ i∈F. norm (x i * x i)) ≤ (∑ ∞ i∈A. norm (x i * x i))
      by (metis (no_types, lifting) b4 infsum_mono2 norm_ge_zero summable_on_subset_banach
x2_sum)
    have norm (x i * y i) ≤ norm (x i * x i) + norm (y i * y i) for i
      unfolding norm_mult by (smt (verit, best) abs_norm_cancel mult_mono
not_sum_squares_lt_zero)
    hence (∑ i∈F. norm (x i * y i)) ≤ (∑ i∈F. norm (x i * x i) + norm (y i *
y i))
      by (simp add: sum_mono)
    also have ... = (∑ i∈F. norm (x i * x i)) + (∑ i∈F. norm (y i * y i))
      by (simp add: sum.distrib)
    also have ... = (∑ ∞ i∈F. norm (x i * x i)) + (∑ ∞ i∈F. norm (y i * y i))
      by (simp add: ‹finite F›)
    also have ... ≤ (∑ ∞ i∈A. norm (x i * x i)) + (∑ ∞ i∈A. norm (y i * y i))
      using F assms
      by (intro add_mono infsum_mono2) auto
    finally show ‹(∑ xa∈F. norm (x xa * y xa)) ≤ (∑ ∞ i∈A. norm (x i * x i))
+ (∑ ∞ i∈A. norm (y i * y i))›
      by simp
  qed
qed
qed auto

```

6.8.4 Extended reals and nats

lemma *summable_on_ennreal[simp]*: $\langle f :: _ \Rightarrow \text{ennreal} \rangle$ *summable_on S* and
summable_on_enat[simp]: $\langle f :: _ \Rightarrow \text{enat} \rangle$ *summable_on S*
 by (*simp_all add: nonneg_summable_on_complete*)

lemma *has_sum_superconst_infinite_ennreal*:

fixes $f :: \langle 'a \Rightarrow \text{ennreal} \rangle$
assumes *geqb*: $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$
assumes $b: \langle b > 0 \rangle$
assumes $\langle \text{infinite } S \rangle$
shows $\langle f \text{ has_sum } \infty \rangle S$

proof –

have $\langle \text{sum } f \longrightarrow \infty \rangle (\text{finite_subsets_at_top } S)$

proof (*rule order_tendstoI*)

fix $y :: \text{ennreal}$ **assume** $\langle y < \infty \rangle$

then have $\langle y / b < \infty \rangle \langle y < \text{top} \rangle$

using *b ennreal_divide_eq_top_iff top.not_eq_extremum* **by force+**

then obtain F **where** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq S \rangle$ **and** $\text{card} F: \langle \text{card } F > y / b \rangle$

using $\langle \text{infinite } S \rangle$

by (*metis ennreal_Ex_less_of_nat infinite_arbitrarily_large infinity_ennreal_def*)

moreover have $\langle \text{sum } f Y > y \rangle$ **if** $\langle \text{finite } Y \rangle$ **and** $\langle F \subseteq Y \rangle$ **and** $\langle Y \subseteq S \rangle$ **for**

Y

proof –

have $\langle y < b * \text{card } F \rangle$

by (*metis b <y < top> cardF divide_less_ennreal ennreal_mult_eq_top_iff gr_implies_not_zero mult.commute top.not_eq_extremum*)

also have $\langle \dots \leq b * \text{card } Y \rangle$

by (*meson b card_mono less_imp_le mult_left_mono of_nat_le_iff that*)

also have $\langle \dots = \text{sum } (\lambda _. b) Y \rangle$

by (*simp add: mult.commute*)

also have $\langle \dots \leq \text{sum } f Y \rangle$

using *geqb* **by** (*meson subset_eq sum_mono that(3)*)

finally show *?thesis* .

qed

ultimately show $\langle \forall F. x \text{ in } \text{finite_subsets_at_top } S. y < \text{sum } f x \rangle$

unfolding *eventually_finite_subsets_at_top* **by auto**

qed auto

then show *?thesis*

by (*simp add: has_sum_def*)

qed

lemma *infsum_superconst_infinite_ennreal*:

fixes $f :: \langle 'a \Rightarrow \text{ennreal} \rangle$

assumes $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$

assumes $\langle b > 0 \rangle$

assumes $\langle \text{infinite } S \rangle$

shows $\text{infsum } f S = \infty$

using *assms infsumI has_sum_superconst_infinite_ennreal* **by blast**

lemma *infsum_superconst_infinite_ereal*:

fixes $f :: \langle 'a \Rightarrow \text{ereal} \rangle$

assumes $\text{geqb}: \langle \bigwedge x. x \in S \implies f x \geq b \rangle$

assumes $b: \langle b > 0 \rangle$

assumes $\langle \text{infinite } S \rangle$

shows $\text{infsum } f S = \infty$

proof –

obtain b' **where** $b': \langle \text{e2ennreal } b' = b \rangle$ **and** $\langle b' > 0 \rangle$

using b **by** *blast*

have $0 < \text{e2ennreal } b$

using b' b

by (*metis dual_order.refl enn2ereal_e2ennreal gr_zeroI order_less_le zero_ennreal.abs_eq*)

hence $*$: $\langle \text{infsum } (\text{e2ennreal} \circ f) S = \infty \rangle$

using *assms b'*

by (*intro infsum_superconst_infinite_ennreal[where b=b'] (auto intro!: e2ennreal_mono)*)

have $\langle \text{infsum } f S = \text{infsum } (\text{enn2ereal} \circ (\text{e2ennreal} \circ f)) S \rangle$

using $\text{geqb } b$ **by** (*intro infsum_cong*) (*fastforce simp: enn2ereal_e2ennreal*)

also have $\langle \dots = \text{enn2ereal } \infty \rangle$

using $*$ **by** (*simp add: infsum_comm_additive_general continuous_at_enn2ereal nonneg_summable_on_complete*)

also have $\langle \dots = \infty \rangle$

by *simp*

finally show *?thesis* .

qed

lemma *has_sum_superconst_infinite_ereal*:

fixes $f :: \langle 'a \Rightarrow \text{ereal} \rangle$

assumes $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$

assumes $\langle b > 0 \rangle$

assumes $\langle \text{infinite } S \rangle$

shows $(f \text{ has_sum } \infty) S$

by (*metis Infty_neq_0(1) assms infsum_def has_sum_infsum infsum_superconst_infinite_ereal*)

lemma *infsum_superconst_infinite_enat*:

fixes $f :: \langle 'a \Rightarrow \text{enat} \rangle$

assumes $\text{geqb}: \langle \bigwedge x. x \in S \implies f x \geq b \rangle$

assumes $b: \langle b > 0 \rangle$

assumes $\langle \text{infinite } S \rangle$

shows $\text{infsum } f S = \infty$

proof –

have $\langle \text{ennreal_of_enat } (\text{infsum } f S) = \text{infsum } (\text{ennreal_of_enat} \circ f) S \rangle$

by (*simp flip: infsum_comm_additive_general*)

also have $\langle \dots = \infty \rangle$

by (*metis assms(3) b comp_def ennreal_of_enat_0 ennreal_of_enat_le_iff geqb infsum_superconst_infinite_ennreal leD leI*)

also have $\langle \dots = \text{ennreal_of_enat } \infty \rangle$

by *simp*

finally show *?thesis*

by (rule ennreal_of_enat_inj[THEN iffD1])
qed

lemma *has_sum_superconst_infinite_enat*:

fixes $f :: \langle 'a \Rightarrow \text{enat} \rangle$

assumes $\langle \bigwedge x. x \in S \implies f x \geq b \rangle$

assumes $\langle b > 0 \rangle$

assumes $\langle \text{infinite } S \rangle$

shows $\langle f \text{ has_sum } \infty \rangle S$

by (metis assms i0_lb has_sum_infsup infsup_superconst_infinite_enat nonneg_summable_on_complete)

This lemma helps to relate a real-valued infsum to a supremum over extended nonnegative reals.

lemma *infsup_nonneg_is_SUPREMUM_ennreal*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes *summable*: $f \text{ summable_on } A$

and *fnn*: $\bigwedge x. x \in A \implies f x \geq 0$

shows $\text{ennreal } (\text{infsup } f A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ennreal } (\text{sum } f F)))$

proof –

have $\S: \bigwedge F. [\text{finite } F; F \subseteq A] \implies \text{sum } (\text{ennreal } \circ f) F = \text{ennreal } (\text{sum } f F)$

by (metis (mono_tags, lifting) comp_def fnn subsetD sum.cong sum_ennreal)

then have $\langle \text{ennreal } (\text{infsup } f A) = \text{infsup } (\text{ennreal } \circ f) A \rangle$

by (simp add: infsup_comm_additive_general local.summable)

also have $\langle \dots = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ennreal } (\text{sum } f F))) \rangle$

by (metis (mono_tags, lifting) \S image_cong mem_Collect_eq nonneg_infsup_complete zero_le)

finally show *?thesis* .

qed

This lemma helps to related a real-valued infsum to a supremum over extended reals.

lemma *infsup_nonneg_is_SUPREMUM_ereal*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes *summable*: $f \text{ summable_on } A$

and *fnn*: $\bigwedge x. x \in A \implies f x \geq 0$

shows $\text{ereal } (\text{infsup } f A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal } (\text{sum } f F)))$

proof –

have $\bigwedge F. [\text{finite } F; F \subseteq A] \implies \text{sum } (\text{ereal } \circ f) F = \text{ereal } (\text{sum } f F)$

by *auto*

then have $\langle \text{ereal } (\text{infsup } f A) = \text{infsup } (\text{ereal } \circ f) A \rangle$

by (simp add: infsup_comm_additive_general local.summable)

also have $\langle \dots = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal } (\text{sum } f F))) \rangle$

by (subst nonneg_infsup_complete) (simp_all add: assms)

finally show *?thesis* .

qed

6.8.5 Real numbers

Most lemmas in the general property section already apply to real numbers. A few ones that are specific to reals are given here.

lemma *infsum_nonneg_is_SUPREMUM_real*:

fixes $f :: 'a \Rightarrow \text{real}$
assumes *summable*: $f \text{ summable_on } A$
and *fnn*: $\bigwedge x. x \in A \implies f x \geq 0$
shows $\text{infsum } f A = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{sum } f F))$
proof –
have $*$: $\text{ereal } (\text{infsum } f A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal } (\text{sum } f F)))$
using *assms* **by** (*rule infsum_nonneg_is_SUPREMUM_ereal*)
also have $\dots = \text{ereal } (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{sum } f F))$
by (*metis (no_types, lifting) * MInfty_neq_ereal(2) PInfty_neq_ereal(2) SUP_cong abs_eq_infinity_cases ereal_SUP*)
finally show *?thesis* **by** *simp*
qed

lemma *has_sum_nonneg_SUPREMUM_real*:

fixes $f :: 'a \Rightarrow \text{real}$
assumes $f \text{ summable_on } A$ **and** $\bigwedge x. x \in A \implies f x \geq 0$
shows $(f \text{ has_sum } (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{sum } f F))) A$
by (*metis (mono_tags, lifting) assms has_sum_infsum infsum_nonneg_is_SUPREMUM_real*)

lemma *summable_countable_real*:

fixes $f :: \langle 'a \Rightarrow \text{real} \rangle$
assumes $\langle f \text{ summable_on } A \rangle$
shows $\langle \text{countable } \{x \in A. f x \neq 0\} \rangle$
using *abs_summable_countable assms summable_on_iff_abs_summable_on_real*
by *blast*

6.8.6 Complex numbers

lemma *has_sum_cnj_iff[simp]*:

fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$
shows $\langle ((\lambda x. \text{cnj } (f x)) \text{ has_sum } \text{cnj } a) M \longleftrightarrow (f \text{ has_sum } a) M \rangle$
by (*simp add: has_sum_def lim_cnj del: cnj_sum add: cnj_sum[symmetric, abs_def, of f]*)

lemma *summable_on_cnj_iff[simp]*:

$(\lambda i. \text{cnj } (f i)) \text{ summable_on } A \longleftrightarrow f \text{ summable_on } A$
by (*metis complex_cnj_cnj summable_on_def has_sum_cnj_iff*)

lemma *infsum_cnj[simp]*: $\langle \text{infsum } (\lambda x. \text{cnj } (f x)) M = \text{cnj } (\text{infsum } f M) \rangle$

by (*metis complex_cnj_zero infsumI has_sum_cnj_iff infsum_def summable_on_cnj_iff has_sum_infsum*)

lemma *has_sum_Re*:

```

assumes (f has_sum a) M
shows (( $\lambda x.$  Re (f x)) has_sum Re a) M
using has_sum_comm_additive[where f=Re]
using assms tendsto_Re by (fastforce simp add: o_def additive_def)

```

```

lemma infsum_Re:
assumes f summable_on M
shows infsum ( $\lambda x.$  Re (f x)) M = Re (infsum f M)
by (simp add: assms has_sum_Re infsumI)

```

```

lemma summable_on_Re:
assumes f summable_on M
shows ( $\lambda x.$  Re (f x)) summable_on M
by (metis assms has_sum_Re summable_on_def)

```

```

lemma has_sum_Im:
assumes (f has_sum a) M
shows (( $\lambda x.$  Im (f x)) has_sum Im a) M
using has_sum_comm_additive[where f=Im]
using assms tendsto_Im by (fastforce simp add: o_def additive_def)

```

```

lemma infsum_Im:
assumes f summable_on M
shows infsum ( $\lambda x.$  Im (f x)) M = Im (infsum f M)
by (simp add: assms has_sum_Im infsumI)

```

```

lemma summable_on_Im:
assumes f summable_on M
shows ( $\lambda x.$  Im (f x)) summable_on M
by (metis assms has_sum_Im summable_on_def)

```

```

lemma nonneg_infsum_le_0D_complex:
fixes f :: 'a  $\Rightarrow$  complex
assumes infsum f A  $\leq$  0
  and abs_sum: f summable_on A
  and nneg:  $\bigwedge x. x \in A \implies f x \geq 0$ 
  and x  $\in$  A
shows f x = 0
proof -
  have  $\langle \text{Im } (f x) = 0 \rangle$ 
    using assms(4) less_eq_complex_def nneg by auto
  moreover have  $\langle \text{Re } (f x) = 0 \rangle$ 
    using assms by (auto simp add: summable_on_Re infsum_Re less_eq_complex_def
intro: nonneg_infsum_le_0D[where A=A])
  ultimately show ?thesis
    by (simp add: complex_eqI)
qed

```

```

lemma nonneg_has_sum_le_0D_complex:

```



```

fixes  $f :: 'a \Rightarrow \text{complex}$ 
assumes  $(f \text{ has\_sum } a) \ A \ \text{and} \ \langle a \leq 0 \rangle$ 
and  $\bigwedge x. x \in A \implies f \ x \geq 0 \ \text{and} \ x \in A$ 
shows  $f \ x = 0$ 
by  $(\text{metis } \text{assms } \text{infsumI } \text{nonneg\_infsum\_le\_0D\_complex } \text{summable\_on\_def})$ 

```

The lemma *infsum_mono_neutral* above applies to various linear ordered monoids such as the reals but not to the complex numbers. Thus we have a separate corollary for those:

lemma *infsum_mono_neutral_complex*:

```

fixes  $f :: 'a \Rightarrow \text{complex}$ 
assumes  $[\text{simp}]: f \text{ summable\_on } A$ 
and  $[\text{simp}]: g \text{ summable\_on } B$ 
assumes  $\langle \bigwedge x. x \in A \cap B \implies f \ x \leq g \ x \rangle$ 
assumes  $\langle \bigwedge x. x \in A - B \implies f \ x \leq 0 \rangle$ 
assumes  $\langle \bigwedge x. x \in B - A \implies g \ x \geq 0 \rangle$ 
shows  $\langle \text{infsum } f \ A \leq \text{infsum } g \ B \rangle$ 
proof  $-$ 
have  $\langle \text{infsum } (\lambda x. \text{Re } (f \ x)) \ A \leq \text{infsum } (\lambda x. \text{Re } (g \ x)) \ B \rangle$ 
by  $(\text{smt } (\text{verit } \text{assms } \text{infsum\_cong } \text{infsum\_mono\_neutral\_less\_eq\_complex\_def } \text{summable\_on\_Re } \text{zero\_complex.simps}(1)))$ 
then have  $\text{Re}: \langle \text{Re } (\text{infsum } f \ A) \leq \text{Re } (\text{infsum } g \ B) \rangle$ 
by  $(\text{metis } \text{assms}(1-2) \ \text{infsum\_Re})$ 
have  $\langle \text{infsum } (\lambda x. \text{Im } (f \ x)) \ A = \text{infsum } (\lambda x. \text{Im } (g \ x)) \ B \rangle$ 
by  $(\text{smt } (\text{verit, best } \text{assms}(3-5) \ \text{infsum\_cong\_neutral\_less\_eq\_complex\_def } \text{zero\_complex.simps}(2)))$ 
then have  $\text{Im}: \langle \text{Im } (\text{infsum } f \ A) = \text{Im } (\text{infsum } g \ B) \rangle$ 
by  $(\text{metis } \text{assms}(1-2) \ \text{infsum\_Im})$ 
from  $\text{Re } \text{Im}$  show  $?thesis$ 
by  $(\text{auto } \text{simp: } \text{less\_eq\_complex\_def})$ 
qed

```

lemma *infsum_mono_complex*:

— For real, *infsum_mono* can be used. But *complex* does not have the right typeclass.

```

fixes  $f \ g :: 'a \Rightarrow \text{complex}$ 
assumes  $f\_sum: f \text{ summable\_on } A \ \text{and} \ g\_sum: g \text{ summable\_on } A$ 
assumes  $\text{leq}: \bigwedge x. x \in A \implies f \ x \leq g \ x$ 
shows  $\text{infsum } f \ A \leq \text{infsum } g \ A$ 
by  $(\text{metis } \text{DiffE } \text{IntD1 } f\_sum \ g\_sum \ \text{infsum\_mono\_neutral\_complex } \text{leq})$ 

```

lemma *infsum_nonneg_complex*:

```

fixes  $f :: 'a \Rightarrow \text{complex}$ 
assumes  $f \text{ summable\_on } M$ 
and  $\bigwedge x. x \in M \implies 0 \leq f \ x$ 
shows  $\text{infsum } f \ M \geq 0$  (is ?lhs  $\geq$   $\_$ )
by  $(\text{metis } \text{assms } \text{infsum\_0\_simp } \text{summable\_on\_0\_simp } \text{infsum\_mono\_complex})$ 

```

```

lemma infsum_cmod:
  assumes f summable_on M
    and fnn:  $\bigwedge x. x \in M \implies 0 \leq f x$ 
  shows infsum  $(\lambda x. cmod (f x)) M = cmod (infsum f M)$ 
proof -
  have  $\langle complex\_of\_real (infsum (\lambda x. cmod (f x)) M) = infsum (\lambda x. complex\_of\_real (cmod (f x))) M \rangle$ 
  proof (rule infsum_comm_additive[symmetric, unfolded o_def])
    have  $(\lambda z. Re (f z))$  summable_on M
    using assms summable_on_Re by blast
    also have ?this  $\longleftrightarrow f$  abs_summable_on M
    using fnn by (intro summable_on_cong) (auto simp: less_eq_complex_def cmod_def)
    finally show ... .
  qed (auto simp: additive_def)
  also have  $\langle \dots = infsum f M \rangle$ 
  using fnn cmod_eq_Re complex_is_Real_iff less_eq_complex_def by (force cong: infsum_cong)
  finally show ?thesis
  by (metis abs_of_nonneg infsum_def le_less_trans norm_ge_zero norm_infsum_bound norm_of_real not_le order_refl)
qed

```

```

lemma summable_on_iff_abs_summable_on_complex:
  fixes f ::  $\langle 'a \Rightarrow complex \rangle$ 
  shows  $\langle f$  summable_on A  $\longleftrightarrow f$  abs_summable_on A  $\rangle$ 
proof (rule iffI)
  assume  $\langle f$  summable_on A  $\rangle$ 
  define i r ni nr n where  $\langle i x = Im (f x) \rangle$  and  $\langle r x = Re (f x) \rangle$ 
  and  $\langle ni x = norm (i x) \rangle$  and  $\langle nr x = norm (r x) \rangle$  and  $\langle n x = norm (f x) \rangle$  for
  x
  from  $\langle f$  summable_on A  $\rangle$  have  $\langle i$  summable_on A  $\rangle$ 
  by (simp add: i_def[abs_def] summable_on_Im)
  then have [simp]:  $\langle ni$  summable_on A  $\rangle$ 
  using ni_def[abs_def] summable_on_iff_abs_summable_on_real by force

  from  $\langle f$  summable_on A  $\rangle$  have  $\langle r$  summable_on A  $\rangle$ 
  by (simp add: r_def[abs_def] summable_on_Re)
  then have [simp]:  $\langle nr$  summable_on A  $\rangle$ 
  by (metis nr_def summable_on_cong summable_on_iff_abs_summable_on_real)

  have n_sum:  $\langle n x \leq nr x + ni x \rangle$  for x
  by (simp add: n_def nr_def ni_def r_def i_def cmod_le)

  have *:  $\langle (\lambda x. nr x + ni x)$  summable_on A  $\rangle$ 
  by (simp add: summable_on_add)
  have bdd_above (sum n  $\{ F. F \subseteq A \wedge finite F \}$ )
  apply (rule bdd_aboveI[where M= $\langle infsum (\lambda x. nr x + ni x) A \rangle$ ])

```

```

using * n_sum by (auto simp flip: infsum_finite simp: ni_def nr_def intro!:
infsum_mono_neutral)
then show  $\langle n \text{ summable\_on } A \rangle$ 
by (simp add: n_def nonneg_bdd_above_summable_on)
next
show  $\langle f \text{ abs\_summable\_on } A \implies f \text{ summable\_on } A \rangle$ 
using abs_summable_summable by blast
qed

```

```

lemma summable_countable_complex:
fixes f ::  $\langle 'a \Rightarrow \text{complex} \rangle$ 
assumes  $\langle f \text{ summable\_on } A \rangle$ 
shows  $\langle \text{countable } \{x \in A. f\ x \neq 0\} \rangle$ 
using abs_summable_countable assms summable_on_iff_abs_summable_on_complex
by blast

```

```

inductive (in topological_space) convergent_filter ::  $'a \text{ filter} \Rightarrow \text{bool}$  where
  F  $\leq \text{nhds } x \implies \text{convergent\_filter } F$ 

```

```

lemma (in topological_space) convergent_filter_iff: convergent_filter F  $\longleftrightarrow (\exists x. F \leq \text{nhds } x)$ 
by (auto simp: convergent_filter.simps)

```

```

lemma (in uniform_space) cauchy_filter_mono:
  cauchy_filter F  $\implies F' \leq F \implies \text{cauchy\_filter } F'$ 
unfolding cauchy_filter_def by (meson dual_order.trans prod_filter_mono)

```

```

lemma (in uniform_space) convergent_filter_cauchy:
assumes convergent_filter F
shows cauchy_filter F
using assms cauchy_filter_mono nhds_imp_cauchy_filter[OF order_refl]
by (auto simp: convergent_filter_iff)

```

```

lemma (in topological_space) convergent_filter_bot [simp, intro]: convergent_filter
  bot
by (simp add: convergent_filter_iff)

```

```

class complete_uniform_space = uniform_space +
assumes cauchy_filter_convergent': cauchy_filter (F ::  $'a \text{ filter}$ )  $\implies F \neq \text{bot} \implies \text{convergent\_filter } F$ 

```

```

lemma (in complete_uniform_space)
  cauchy_filter_convergent: cauchy_filter (F ::  $'a \text{ filter}$ )  $\implies \text{convergent\_filter } F$ 
using cauchy_filter_convergent'[of F] by (cases F = bot) auto

```

lemma (in *complete_uniform_space*) *convergent_filter_iff_cauchy*:
convergent_filter $F \longleftrightarrow$ *cauchy_filter* F
using *convergent_filter_cauchy* *cauchy_filter_convergent* **by** *blast*

definition *countably_generated_filter* :: 'a filter \Rightarrow bool **where**
countably_generated_filter $F \longleftrightarrow (\exists U :: \text{nat} \Rightarrow \text{'a set}. F = (\text{INF } (n::\text{nat}).$
principal ($U\ n$)))

lemma *countably_generated_filter_has_antimono_basis*:

assumes *countably_generated_filter* F

obtains $B :: \text{nat} \Rightarrow \text{'a set}$

where *antimono* B **and** $F = (\text{INF } n. \text{principal } (B\ n))$ **and**

$\bigwedge P. \text{eventually } P\ F \longleftrightarrow (\exists i. \forall x \in B\ i. P\ x)$

proof –

from *assms* **obtain** B **where** $B: F = (\text{INF } (n::\text{nat}). \text{principal } (B\ n))$

unfolding *countably_generated_filter_def* **by** *blast*

define B' **where** $B' = (\lambda n. \bigcap_{k \leq n}. B\ k)$

have *antimono* B'

unfolding *decseq_def* B'_def **by** *force*

have $(\text{INF } n. \text{principal } (B'\ n)) = (\text{INF } n. \text{INF } k \in \{..n\}. \text{principal } (B\ k))$

unfolding B'_def **by** (*intro* *INF_cong_refl* *INF_principal_finite* [*symmetric*])

auto

also have $\dots = (\text{INF } (n::\text{nat}). \text{principal } (B\ n))$

apply (*intro antisym*)

apply (*meson* *INF_lower* *INF_mono* *atMost_iff* *order_refl*)

apply (*meson* *INF_greatest* *INF_lower* *UNIV_I*)

done

also have $\dots = F$

by (*simp add: B*)

finally have $F: F = (\text{INF } n. \text{principal } (B'\ n))$..

moreover have *eventually* $P\ F \longleftrightarrow (\exists i. \text{eventually } P\ (\text{principal } (B'\ i)))$ **for** P

unfolding F **using** $\langle \text{antimono } B' \rangle$

apply (*subst* *eventually_INF_base*)

apply (*auto simp: decseq_def*)

by (*meson* *nat_le_linear*)

ultimately show *?thesis*

using $\langle \text{antimono } B' \rangle$ *that*[*of B'*] **unfolding** *eventually_principal* **by** *blast*

qed

lemma (in *uniform_space*) *cauchy_filter_iff*:

cauchy_filter $F \longleftrightarrow (\forall P. \text{eventually } P\ \text{uniformity} \longrightarrow (\exists X. \text{eventually } (\lambda x. x \in X)\ F \wedge (\forall z \in X \times X. P\ z)))$

unfolding *cauchy_filter_def* *le_filter_def*

apply (*auto simp: eventually_prod_same*)

apply (*metis* (*full_types*) *eventually_mono* *mem_Collect_eq*)

apply *blast*

done

lemma (in *uniform_space*) *controlled_sequences_convergent_imp_complete_aux_sequence*:

fixes $U :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$

fixes $F :: 'a \text{ filter}$

assumes *cauchy_filter* $F \neq \text{bot}$

assumes $\bigwedge n. \text{eventually } (\lambda z. z \in U \ n) \text{ uniformity}$

obtains $g \ G \text{ where}$

$\text{antimono } G \ \bigwedge n. g \ n \in G \ n$

$\bigwedge n. \text{eventually } (\lambda x. x \in G \ n) \ F \ \bigwedge n. G \ n \times G \ n \subseteq U \ n$

proof –

have $\exists C. \text{eventually } (\lambda x. x \in C) \ F \ \wedge \ C \times C \subseteq U \ n \ \text{for } n$

using *assms(1) assms(3)[of n]* **unfolding** *cauchy_filter_iff* **by** *blast*

then obtain $G \text{ where } G: \bigwedge n. \text{eventually } (\lambda x. x \in G \ n) \ F \ \bigwedge n. G \ n \times G \ n \subseteq U \ n$

by *metis*

define $G' \text{ where } G' = (\lambda n. \bigcap_{k \leq n}. G \ k)$

have $1: \text{eventually } (\lambda x. x \in G' \ n) \ F \ \text{for } n$

using G **by** (*auto simp: G'_def intro: eventually_ball_finite*)

have $2: G' \ n \times G' \ n \subseteq U \ n \ \text{for } n$

using G **unfolding** G'_def **by** *fast*

have $3: \text{antimono } G'$

unfolding G'_def *decseq_def* **by** *force*

have $\exists g. g \in G' \ n \ \text{for } n$

using 1 *assms(2)* *eventually_happens'* **by** *auto*

then obtain $g \ \text{where } g: \bigwedge n. g \ n \in G' \ n$

by *metis*

from $g \ 1 \ 2 \ 3$ *that[of G' g]* **show** *?thesis*

by *metis*

qed

definition *lift_filter* $:: ('a \text{ set} \Rightarrow 'b \text{ filter}) \Rightarrow 'a \text{ filter} \Rightarrow 'b \text{ filter} \ \text{where}$

lift_filter $f \ F = (\text{INF } X \in \{X. \text{eventually } (\lambda x. x \in X) \ F\}. f \ X)$

lemma *lift_filter_top* [*simp*]: *lift_filter* $g \ \text{top} = g \ \text{UNIV}$

proof –

have $\{X. \forall x::'b. x \in X\} = \{\text{UNIV}\}$

by *auto*

thus *?thesis*

by (*simp add: lift_filter_def*)

qed

lemma *eventually_lift_filter_iff*:

assumes *mono* g

shows $\text{eventually } P \ (\text{lift_filter } g \ F) \longleftrightarrow (\exists X. \text{eventually } (\lambda x. x \in X) \ F \ \wedge \ \text{eventually } P \ (g \ X))$

unfolding *lift_filter_def*

proof (*subst eventually_INF_base, goal_cases*)

```

    case 1
  thus ?case by (auto intro: exI[of _ UNIV])
next
  case (2 X Y)
  thus ?case
    by (auto intro!: exI[of _ X ∩ Y] eventually_conj monoD[OF assms])
qed auto

```

```

lemma lift_filter_le:
  assumes eventually (λx. x ∈ X) F g X ≤ F'
  shows lift_filter g F ≤ F'
  unfolding lift_filter_def
  by (metis INF_lower2 assms mem_Collect_eq)

```

```

definition lift_filter' :: ('a set ⇒ 'b set) ⇒ 'a filter ⇒ 'b filter where
  lift_filter' f F = lift_filter (principal ∘ f) F

```

```

lemma lift_filter'_top [simp]: lift_filter' g top = principal (g UNIV)
  by (simp add: lift_filter'_def)

```

```

lemma eventually_lift_filter'_iff:
  assumes mono g
  shows eventually P (lift_filter' g F) ↔ (∃ X. eventually (λx. x ∈ X) F ∧
(∀ x ∈ g X. P x))
  unfolding lift_filter'_def using assms
  by (subst eventually_lift_filter'_iff) (auto simp: mono_def eventually_principal)

```

```

lemma lift_filter'_le:
  assumes eventually (λx. x ∈ X) F principal (g X) ≤ F'
  shows lift_filter' g F ≤ F'
  unfolding lift_filter'_def using assms
  by (intro lift_filter_le[where X = X]) auto

```

```

lemma (in uniform_space) comp_uniformity_le_uniformity:
  lift_filter' (λX. X O X) uniformity ≤ uniformity
  unfolding le_filter_def
proof safe
  fix P assume P: eventually P uniformity
  have [simp]: mono (λX::('a × 'a) set. X O X)
    by (intro monoI) auto
  from P obtain P' where P': eventually P' uniformity (∧ x y z. P' (x, y) ⇒
P' (y, z) ⇒ P (x, z))
  using uniformity_transE by blast
  show eventually P (lift_filter' (λX. X O X) uniformity)
    by (auto simp: eventually_lift_filter'_iff intro!: exI[of _ {x. P' x}] P')
qed

```

```

lemma (in uniform_space) comp_mem_uniformity_sets:

```

```

assumes eventually ( $\lambda z. z \in X$ ) uniformity
obtains  $Y$  where eventually ( $\lambda z. z \in Y$ ) uniformity  $Y \circ Y \subseteq X$ 
proof –
  have [simp]: mono ( $\lambda X. ('a \times 'a)$  set.  $X \circ X$ )
    by (intro monoI) auto
  have eventually ( $\lambda z. z \in X$ ) (lift_filter' ( $\lambda X. X \circ X$ ) uniformity)
    using assms comp_uniformity_le_uniformity using filter_leD by blast
  thus ?thesis using that
    by (auto simp: eventually_lift_filter'_iff)
qed

```

```

lemma (in uniform_space) le_nhds_of_cauchy_adhp_aux:
  assumes  $\bigwedge P. \text{eventually } P \text{ uniformity} \implies (\exists X. \text{eventually } (\lambda y. y \in X) F \wedge$ 
  ( $\forall z \in X \times X. P z$ )  $\wedge (\exists y. P (x, y) \wedge y \in X))$ 
  shows  $F \leq \text{nhds } x$ 
  unfolding le_filter_def
proof safe
  fix  $P$  assume eventually  $P$  (nhds  $x$ )
  hence  $\forall_F z$  in uniformity.  $z \in \{z. \text{fst } z = x \longrightarrow P (\text{snd } z)\}$ 
    by (simp add: eventually_nhds_uniformity case_prod_unfold)
  then obtain  $Y$  where  $Y: \forall_F z$  in uniformity.  $z \in Y \circ Y \subseteq \{z. \text{fst } z = x$ 
   $\longrightarrow P (\text{snd } z)\}$ 
    using comp_mem_uniformity_sets by blast
  obtain  $X \ y$  where  $Xy: \text{eventually } (\lambda y. y \in X) F \ X \times X \subseteq Y \ (x, y) \in Y \ y \in X$ 
    using assms[OF Y(1)] by blast
  have *:  $P \ x$  if  $x \in X$  for  $x$ 
    using  $Y(2) \ Xy(2-4)$  that unfolding relcomp_unfold by force
  show eventually  $P \ F$ 
    by (rule eventually_mono[OF Xy(1)]) (use * in auto)
qed

```

```

lemma (in uniform_space) eventually_uniformity_imp_nhds:
  assumes eventually  $P$  uniformity
  shows eventually ( $\lambda y. P (x, y)$ ) (nhds  $x$ )
  using assms unfolding eventually_nhds_uniformity by (elim eventually_mono)
auto

```

```

lemma (in uniform_space) controlled_sequences_convergent_imp_complete_aux:
  fixes  $U :: \text{nat} \Rightarrow ('a \times 'a)$  set
  assumes gen: countably_generated_filter (uniformity ::  $('a \times 'a)$  filter)
  assumes  $U: \bigwedge n. \text{eventually } (\lambda z. z \in U \ n)$  uniformity
  assumes conv:  $\bigwedge (u :: \text{nat} \Rightarrow 'a). (\bigwedge N \ m \ n. N \leq m \implies N \leq n \implies (u \ m, u \ n)$ 
   $\in U \ N) \implies \text{convergent } u$ 
  assumes cauchy_filter  $F$ 
  shows convergent_filter  $F$ 
proof (cases  $F = \text{bot}$ )
  case False
  note  $F = \langle \text{cauchy\_filter } F \rangle \langle F \neq \text{bot} \rangle$ 
  from gen obtain  $B :: \text{nat} \Rightarrow ('a \times 'a)$  set where  $B:$ 

```

```

antimono B uniformity = (INF n. principal (B n))
 $\wedge P. \text{eventually } P \text{ uniformity} \longleftrightarrow (\exists i. \forall x \in B \ i. P \ x)$ 
using countably_generated_filter_has_antimono_basis by blast

have ev_B: eventually ( $\lambda z. z \in B \ n$ ) uniformity for n
by (subst B(3)) auto
hence ev_B': eventually ( $\lambda z. z \in B \ n \cap U \ n$ ) uniformity for n
using U by (auto intro: eventually_conj)

obtain g G where gG: antimono G  $\wedge n. g \ n \in G \ n$ 
 $\wedge n. \text{eventually} (\lambda x. x \in G \ n) F \wedge n. G \ n \times G \ n \subseteq B \ n \cap U \ n$ 
using controlled_sequences_convergent_imp_complete_aux_sequence[of F  $\lambda n. B \ n \cap U \ n, OF \ F \ ev\_B'$ ]
by metis

have convergent g
proof (rule conv)
  fix N m n :: nat
  assume mn:  $N \leq m \ N \leq n$ 
  have (g m, g n)  $\in G \ m \times G \ n$ 
  using gG by auto
  also from mn have  $\dots \subseteq G \ N \times G \ N$ 
  by (intro Sigma_mono gG antimonoD[OF gG(1)])
  also have  $\dots \subseteq U \ N$ 
  using gG by blast
  finally show (g m, g n)  $\in U \ N$  .
qed
then obtain L where G:  $g \longrightarrow L$ 
unfolding convergent_def by blast

have  $F \leq nhds \ L$ 
proof (rule le_nhds_of_cauchy_adhp_aux)
  fix P :: 'a  $\times$  'a  $\Rightarrow$  bool
  assume P: eventually P uniformity
  hence eventually ( $\lambda n. \forall x \in B \ n. P \ x$ ) sequentially
  using  $\langle \text{antimono } B \rangle$  unfolding B(3) eventually_sequentially_decseq_def by
blast
  moreover have eventually ( $\lambda n. P \ (L, g \ n)$ ) sequentially
  using P eventually_compose_filterlim eventually_uniformity_imp_nhds G
by blast
  ultimately have eventually ( $\lambda n. (\forall x \in B \ n. P \ x) \wedge P \ (L, g \ n)$ ) sequentially
  by eventually_elim auto
  then obtain n where  $\forall x \in B \ n. P \ x \ P \ (L, g \ n)$ 
  unfolding eventually_at_top_linorder by blast
  then show  $\exists X. (\forall_F \ y \ \text{in } F. y \in X) \wedge (\forall z \in X \times X. P \ z) \wedge (\exists y. P \ (L, y) \wedge y \in X)$ 
  using gG by blast+
qed
thus convergent_filter F

```


by (auto simp: convergent_filter_iff)
qed auto

theorem (in uniform_space) controlled_sequences_convergent_imp_complete:
fixes $U :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$
assumes $\text{gen: countably_generated_filter (uniformity :: ('a \times 'a) \text{ filter})}$
assumes $U: \bigwedge n. \text{eventually } (\lambda z. z \in U \ n) \ \text{uniformity}$
assumes $\text{conv: } \bigwedge (u :: \text{nat} \Rightarrow 'a). (\bigwedge N \ m \ n. N \leq m \implies N \leq n \implies (u \ m, u \ n) \in U \ N) \implies \text{convergent } u$
shows $\text{class.complete_uniform_space open uniformity}$
by unfold_locales (use assms controlled_sequences_convergent_imp_complete_aux in blast)

lemma filtermap_prod_filter: $\text{filtermap (map_prod } f \ g) (F \times_F G) = \text{filtermap } f \ F \times_F \text{filtermap } g \ G$

proof (intro antisym)

show $\text{filtermap (map_prod } f \ g) (F \times_F G) \leq \text{filtermap } f \ F \times_F \text{filtermap } g \ G$
by (auto simp: le_filter_def eventually_filtermap eventually_prod_filter)

next

show $\text{filtermap } f \ F \times_F \text{filtermap } g \ G \leq \text{filtermap (map_prod } f \ g) (F \times_F G)$

unfolding le_filter_def

proof safe

fix P assume $P: \text{eventually } P (\text{filtermap (map_prod } f \ g) (F \times_F G))$

then obtain $Pf \ Pg$ where $*: \text{eventually } Pf \ F \ \text{eventually } Pg \ G \ \forall x. Pf \ x \longrightarrow (\forall y. Pg \ y \longrightarrow P (f \ x, g \ y))$

by (auto simp: eventually_filtermap eventually_prod_filter)

define Pf' where $Pf' = (\lambda x. \exists y. x = f \ y \wedge Pf \ y)$

define Pg' where $Pg' = (\lambda x. \exists y. x = g \ y \wedge Pg \ y)$

from $*(1)$ **have** $\forall_F x \ \text{in } F. Pf' (f \ x)$

by eventually_elim (auto simp: Pf'_def)

moreover from $*(2)$ **have** $\forall_F x \ \text{in } G. Pg' (g \ x)$

by eventually_elim (auto simp: Pg'_def)

moreover have $(\forall x \ y. Pf' \ x \longrightarrow Pg' \ y \longrightarrow P (x, y))$

using $*(3)$ **by** (auto simp: Pf'_def Pg'_def)

ultimately show $\text{eventually } P (\text{filtermap } f \ F \times_F \text{filtermap } g \ G)$

unfolding eventually_prod_filter eventually_filtermap

by blast

qed

qed

lemma (in uniform_space) Cauchy_seq_iff_tendsto:

$\text{Cauchy } f \longleftrightarrow \text{filterlim (map_prod } f \ f) \ \text{uniformity (at_top } \times_F \ \text{at_top)}$

unfolding Cauchy_uniform cauchy_filter_def filterlim_def filtermap_prod_filter

..

theorem (in uniform_space) controlled_seq_imp_Cauchy_seq:

1502

```

fixes  $U :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$ 
assumes  $U: \bigwedge P. \text{eventually } P \text{ uniformity} \implies (\exists n. \forall x \in U n. P x)$ 
assumes  $\text{controlled}: \bigwedge N m n. N \leq m \implies N \leq n \implies (f m, f n) \in U N$ 
shows  $\text{Cauchy } f$ 
unfolding  $\text{Cauchy\_seq\_iff\_tendsto}$ 
proof –
  show  $\text{filterlim } (\text{map\_prod } f f) \text{ uniformity } (\text{sequentially} \times_F \text{sequentially})$ 
    unfolding  $\text{filterlim\_def } \text{le\_filter\_def}$ 
  proof safe
    fix  $P :: 'a \times 'a \Rightarrow \text{bool}$ 
    assume  $P: \text{eventually } P \text{ uniformity}$ 
    from  $U[\text{OF this}]$  obtain  $N$  where  $\forall x \in U N. P x$ 
    by blast
    then show  $\text{eventually } P (\text{filtermap } (\text{map\_prod } f f) (\text{sequentially} \times_F \text{sequentially}))$ 
      unfolding  $\text{eventually\_filtermap } \text{eventually\_prod\_sequentially}$ 
      by  $(\text{metis } \text{controlled } \text{map\_prod\_simp})$ 
    qed
  qed

lemma (in  $\text{uniform\_space}$ )  $\text{Cauchy\_seq\_convergent\_imp\_complete\_aux}$ :
  fixes  $U :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$ 
  assumes  $\text{gen}: \text{countably\_generated\_filter } (\text{uniformity} :: ('a \times 'a) \text{ filter})$ 
  assumes  $\text{conv}: \bigwedge (u :: \text{nat} \Rightarrow 'a). \text{Cauchy } u \implies \text{convergent } u$ 
  assumes  $\text{cauchy\_filter } F$ 
  shows  $\text{convergent\_filter } F$ 
proof –
  from  $\text{gen}$  obtain  $B :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$  where  $B$ :
     $\text{antimono } B \text{ uniformity} = (\text{INF } n. \text{principal } (B n))$ 
     $\bigwedge P. \text{eventually } P \text{ uniformity} \longleftrightarrow (\exists i. \forall x \in B i. P x)$ 
    using  $\text{countably\_generated\_filter\_has\_antimono\_basis}$  by blast

  show ?thesis
  proof  $(\text{rule } \text{controlled\_sequences\_convergent\_imp\_complete\_aux}[\text{where } U = B])$ 
    show  $\forall_F z \text{ in } \text{uniformity}. z \in B n \text{ for } n$ 
      unfolding  $B(\exists)$  by blast
    next
      fix  $f :: \text{nat} \Rightarrow 'a$ 
      assume  $f: \bigwedge N m n. N \leq m \implies N \leq n \implies (f m, f n) \in B N$ 
      have  $\text{Cauchy } f$  using  $f B$ 
        by  $(\text{intro } \text{controlled\_seq\_imp\_Cauchy\_seq}[\text{where } U = B]) \text{ auto}$ 
      with  $\text{conv}$  show  $\text{convergent } f$ 
        by simp
    qed fact+
  qed

theorem (in  $\text{uniform\_space}$ )  $\text{Cauchy\_seq\_convergent\_imp\_complete}$ :
  fixes  $U :: \text{nat} \Rightarrow ('a \times 'a) \text{ set}$ 

```

```

assumes gen: countably_generated_filter (uniformity :: ('a × 'a) filter)
assumes conv:  $\bigwedge (u :: \text{nat} \Rightarrow 'a). \text{Cauchy } u \implies \text{convergent } u$ 
shows class.complete_uniform_space open_uniformity
by unfold_locales (use assms Cauchy_seq_convergent_imp_complete_aux in blast)

```

lemma (**in** *metric_space*) *countably_generated_uniformity*:

```

countably_generated_filter uniformity
proof -
have (INF  $e \in \{0 < ..\}$ . principal  $\{(x, y). \text{dist } (x::'a) \ y < e\}$ ) =
  (INF  $n \in \text{UNIV}$ . principal  $\{(x, y). \text{dist } x \ y < 1 / \text{real } (\text{Suc } n)\}$ ) (is  $?F = ?G$ )
unfolding uniformity_dist
proof (intro antisym)
have  $?G = (\text{INF } e \in (\lambda n. 1 / \text{real } (\text{Suc } n)) \text{ 'UNIV}. \text{principal } \{(x, y). \text{dist } x \ y < e\})$ 
  (by (simp add: image_image)
also have  $\dots \geq ?F$ 
by (intro INF_superset_mono) auto
finally show  $?F \leq ?G$  .
next
show  $?G \leq ?F$ 
unfolding le_filter_def
proof safe
fix  $P$  assume eventually  $P ?F$ 
then obtain  $\varepsilon$  where  $\varepsilon > 0$  eventually  $P$  (principal  $\{(x, y). \text{dist } x \ y < \varepsilon\}$ )
proof (subst (asm) eventually_INF_base, goal_cases)
case ( $2 \ \varepsilon 1 \ \varepsilon 2$ )
thus  $?case$ 
by (intro bezI[of _ min  $\varepsilon 1 \ \varepsilon 2$ ]) auto
qed auto
from  $\langle \varepsilon > 0 \rangle$  obtain  $n$  where  $1 / \text{real } (\text{Suc } n) < \varepsilon$ 
using nat_approx_posE by blast
then have eventually  $P$  (principal  $\{(x, y). \text{dist } x \ y < 1 / \text{real } (\text{Suc } n)\}$ )
using  $\varepsilon(2)$  by (auto simp: eventually_principal)
thus eventually  $P ?G$ 
by (intro eventually_INF1) auto
qed
qed
thus countably_generated_filter uniformity
unfolding countably_generated_filter_def uniformity_dist by fast
qed

```

subclass (**in** *complete_space*) *complete_uniform_space*

proof (*rule Cauchy_seq_convergent_imp_complete*)

show *convergent* f **if** *Cauchy* f **for** f

using *Cauchy_convergent* **that** **by** *blast*

qed (*fact countably_generated_uniformity*)

lemma (in *complete_uniform_space*) *complete_UNIV_cuspace* [intro]: *complete_UNIV*

unfolding *complete_uniform* **using** *cauchy_filter_convergent*
by (*auto simp: convergent_filter.simps*)

lemma *norm_infsum_le*:

assumes (*f has_sum S*) *X*
assumes (*g has_sum T*) *X*
assumes $\bigwedge x. x \in X \implies \text{norm } (f\ x) \leq g\ x$
shows $\text{norm } S \leq T$

proof (*rule tendsto_le*)

show $((\lambda Y. \text{norm } (\sum_{x \in Y}. f\ x)) \longrightarrow \text{norm } S) \text{ (finite_subsets_at_top } X)$

using *assms(1)* **unfolding** *has_sum_def* **by** (*intro tendsto_norm*)

show $((\lambda Y. \sum_{x \in Y}. g\ x) \longrightarrow T) \text{ (finite_subsets_at_top } X)$

using *assms(2)* **unfolding** *has_sum_def* .

show $\forall_F x \text{ in finite_subsets_at_top } X. \text{norm } (\text{sum } f\ x) \leq (\sum_{x \in x}. g\ x)$

by (*simp add: assms(3) eventually_finite_subsets_at_top_weakI subsetD sum_norm_le*)

qed *auto*

lemma *has_sum_imp_summable*: (*f has_sum S*) *A* \implies *f summable_on A*

by (*auto simp: summable_on_def*)

lemma *has_sum_reindex_bij_betw*:

assumes *bij_betw g A B*
shows $((\lambda x. f\ (g\ x)) \text{ has_sum } S) \text{ } A = (f \text{ has_sum } S) \text{ } B$

proof –

have $((\lambda x. f\ (g\ x)) \text{ has_sum } S) \text{ } A \longleftrightarrow (f \text{ has_sum } S) \text{ } (g\ ‘ A)$

by (*subst has_sum_reindex*) (*use assms in <auto dest: bij_betw_imp_inj_on simp: o_def>*)

then show *?thesis*

using *assms bij_betw_imp_surj_on* **by** *blast*

qed

lemma *has_sum_reindex_bij_witness*:

assumes $\bigwedge a. a \in S \implies i\ (j\ a) = a$

assumes $\bigwedge a. a \in S \implies j\ a \in T$

assumes $\bigwedge b. b \in T \implies j\ (i\ b) = b$

assumes $\bigwedge b. b \in T \implies i\ b \in S$

assumes $\bigwedge a. a \in S \implies h\ (j\ a) = g\ a$

assumes $s = s'$

shows $(g \text{ has_sum } s) \text{ } S = (h \text{ has_sum } s') \text{ } T$

by (*smt (verit, del_insts) assms bij_betwI' has_sum_cong has_sum_reindex_bij_betw*)

lemma *has_sum_homomorphism*:

```

  assumes (f has_sum S) A h 0 = 0  $\wedge$  a b. h (a + b) = h a + h b continuous_on
  UNIV h
  shows (( $\lambda$ x. h (f x)) has_sum (h S)) A
  proof -
  have sum (h  $\circ$  f) X = h (sum f X) for X
    by (induction X rule: infinite_finite_induct) (simp_all add: assms)
  hence sum_h: sum (h  $\circ$  f) = h  $\circ$  sum f
    by (intro ext) auto
  then have ((h  $\circ$  f) has_sum h S) A
    using assms
  by (metis UNIV_I continuous_on_def has_sum_comm_additive_general o_apply)
  thus ?thesis
    by (simp add: o_def)
  qed

```

```

lemma summable_on_homomorphism:
  assumes f summable_on A h 0 = 0  $\wedge$  a b. h (a + b) = h a + h b continuous_on
  UNIV h
  shows ( $\lambda$ x. h (f x)) summable_on A
  proof -
  from assms(1) obtain S where (f has_sum S) A
    by (auto simp: summable_on_def)
  hence (( $\lambda$ x. h (f x)) has_sum h S) A
    by (rule has_sum_homomorphism) (use assms in auto)
  thus ?thesis
    by (auto simp: summable_on_def)
  qed

```

```

lemma infsum_homomorphism_strong:
  fixes h :: 'a :: {t2_space, topological_comm_monoid_add}  $\Rightarrow$ 
    'b :: {t2_space, topological_comm_monoid_add}
  assumes ( $\lambda$ x. h (f x)) summable_on A  $\longleftrightarrow$  f summable_on A
  assumes h 0 = 0
  assumes  $\wedge$ S. (f has_sum S) A  $\implies$  (( $\lambda$ x. h (f x)) has_sum (h S)) A
  shows infsum ( $\lambda$ x. h (f x)) A = h (infsum f A)
  by (metis assms has_sum_infsum infsumI infsum_not_exists)

```

```

lemma has_sum_bounded_linear:
  assumes bounded_linear h and (f has_sum S) A
  shows (( $\lambda$ x. h (f x)) has_sum h S) A
  proof -
  interpret bounded_linear h by fact
  from assms(2) show ?thesis
    by (rule has_sum_homomorphism) (auto simp: add intro!: continuous_on)
  qed

```

```

lemma summable_on_bounded_linear:
  assumes bounded_linear h and f summable_on A
  shows ( $\lambda$ x. h (f x)) summable_on A

```

by (metis assms has_sum_bounded_linear summable_on_def)

lemma *summable_on_bounded_linear_iff*:

assumes *bounded_linear h* and *bounded_linear h'* and $\bigwedge x. h' (h x) = x$

shows $(\lambda x. h (f x))$ *summable_on* *A* \longleftrightarrow *f summable_on A*

by (metis (full_types) assms summable_on_bounded_linear summable_on_cong)

lemma *infsum_bounded_linear_strong*:

fixes *h* :: 'a :: *real_normed_vector* \Rightarrow 'b :: *real_normed_vector*

assumes $(\lambda x. h (f x))$ *summable_on* *A* \longleftrightarrow *f summable_on A*

assumes *bounded_linear h*

shows *infsum* $(\lambda x. h (f x))$ *A* = *h (infsum f A)*

proof –

interpret *bounded_linear h* by fact

show ?thesis

by (rule *infsum_homomorphism_strong*)

(insert assms, auto intro: *add_continuous_on_has_sum_bounded_linear*)

qed

lemma *infsum_bounded_linear_strong'*:

fixes *mult* :: 'c :: *zero* \Rightarrow 'a :: *real_normed_vector* \Rightarrow 'b :: *real_normed_vector*

assumes $c \neq 0 \implies (\lambda x. mult\ c\ (f\ x))$ *summable_on* *A* \longleftrightarrow *f summable_on A*

assumes *bounded_linear (mult c)*

assumes [simp]: $\bigwedge x. mult\ 0\ x = 0$

shows *infsum* $(\lambda x. mult\ c\ (f\ x))$ *A* = *mult c (infsum f A)*

by (metis assms *infsum_0 infsum_bounded_linear_strong'*)

lemma *has_sum_of_nat*: $(f\ has_sum\ S)$ *A* $\implies ((\lambda x. of_nat\ (f\ x))\ has_sum\ of_nat\ S)$ *A*

by (erule *has_sum_homomorphism*) (auto intro!: *continuous_intros*)

lemma *has_sum_of_int*: $(f\ has_sum\ S)$ *A* $\implies ((\lambda x. of_int\ (f\ x))\ has_sum\ of_int\ S)$ *A*

by (erule *has_sum_homomorphism*) (auto intro!: *continuous_intros*)

lemma *summable_on_of_nat*: *f summable_on A* $\implies (\lambda x. of_nat\ (f\ x))$ *summable_on A*

by (erule *summable_on_homomorphism*) (auto intro!: *continuous_intros*)

lemma *summable_on_of_int*: *f summable_on A* $\implies (\lambda x. of_int\ (f\ x))$ *summable_on A*

by (erule *summable_on_homomorphism*) (auto intro!: *continuous_intros*)

lemma *summable_on_discrete_iff*:

fixes *f* :: 'a \Rightarrow 'b :: {*discrete_topology, topological_comm_monoid_add, cancel_comm_monoid_add*}

shows *f summable_on A* \longleftrightarrow *finite* $\{x \in A. f\ x \neq 0\}$

proof

assume *: *finite* $\{x \in A. f\ x \neq 0\}$

```

  hence  $f$  summable_on  $\{x \in A. f\ x \neq 0\}$ 
    by (rule summable_on_finite)
  then show  $f$  summable_on  $A$ 
    by (smt (verit) DiffE mem_Collect_eq summable_on_cong_neutral)
next
  assume  $f$  summable_on  $A$ 
  then obtain  $S$  where ( $f$  has_sum  $S$ )  $A$ 
    by (auto simp: summable_on_def)
  hence  $\forall_F x$  in finite_subsets_at_top  $A. \text{sum } f\ x = S$ 
    unfolding has_sum_def tendsto_discrete .
  then obtain  $X$  where  $X: \text{finite } X \ X \subseteq A \wedge Y. \text{finite } Y \implies X \subseteq Y \implies Y \subseteq A \implies \text{sum } f\ Y = S$ 
    unfolding eventually_finite_subsets_at_top by metis
  have  $\{x \in A. f\ x \neq 0\} \subseteq X$ 
  proof
    fix  $x$  assume  $x: x \in \{x \in A. f\ x \neq 0\}$ 
    show  $x \in X$ 
    proof (rule ccontr)
      assume [simp]:  $x \notin X$ 
      have  $\text{sum } f\ (\text{insert } x\ X) = S$ 
        using  $X\ x$  by (intro  $X$ ) auto
      then have  $f\ x = 0$ 
        using  $X$  by auto
      with  $x$  show False
        by auto
    qed
  qed
  thus finite  $\{x \in A. f\ x \neq 0\}$ 
    using  $X(1)$  finite_subset by blast
qed

lemma has_sum_imp_sums: ( $f$  has_sum  $S$ ) ( $UNIV :: \text{nat set}$ )  $\implies f$  sums  $S$ 
  unfolding sums_def has_sum_def by (rule filterlim_compose[OF filterlim_lessThan_at_top])

lemma summable_on_imp_summable:  $f$  summable_on ( $UNIV :: \text{nat set}$ )  $\implies$ 
  summable  $f$ 
  unfolding summable_on_def summable_def by (auto dest: has_sum_imp_sums)

lemma summable_on_UNIV_nonneg_real_iff:
  assumes  $\bigwedge n. f\ n \geq (0 :: \text{real})$ 
  shows  $f$  summable_on  $UNIV \iff$  summable  $f$ 
  using assms by (auto intro: norm_summable_imp_summable_on summable_on_imp_summable)

lemma summable_on_imp_bounded_partial_sums:
  fixes  $f :: \_ \Rightarrow 'a :: \{\text{topological\_comm\_monoid\_add, linorder\_topology}\}$ 
  assumes  $f: f$  summable_on  $A$ 
  shows  $\exists C. \text{eventually } (\lambda X. \text{sum } f\ X \leq C)$  (finite_subsets_at_top  $A$ )
proof -
  from assms obtain  $S$  where  $S: (\text{sum } f \longrightarrow S)$  (finite_subsets_at_top  $A$ )

```

```

    unfolding summable_on_def has_sum_def by blast
  show ?thesis
  proof (cases  $\exists C. C > S$ )
    case True
    then obtain C where C:  $C > S$ 
    by blast
    have  $\forall_F X \text{ in } \text{finite\_subsets\_at\_top } A. \text{sum } f X < C$ 
    using S C by (rule order_tendstoD(2))
    thus ?thesis
    by (meson eventually_mono nless_le)
  next
    case False thus ?thesis
    by (meson not_eventuallyD not_le_imp_less)
  qed
qed

```

```

lemma has_sum_mono':
  fixes S S' :: 'a :: {linorder_topology, ordered_comm_monoid_add, topological_comm_monoid_add}
  assumes f: (f has_sum S) A (f has_sum S') B
  and AB:  $A \subseteq B \wedge x. x \in B - A \implies f x \geq 0$ 
  shows  $S \leq S'$ 
  using AB has_sum_mono_neutral[OF f] by fastforce

```

```

context
  assumes SORT_CONSTRAINT('a :: {topological_comm_monoid_add, order_topology,
    ordered_comm_monoid_add, conditionally_complete_linorder})
begin

```

Any family of non-negative numbers with bounded partial sums is summable, and the sum is simply the supremum of the partial sums.

```

lemma nonneg_bounded_partial_sums_imp_has_sum_SUP:
  assumes nonneg:  $\wedge x. x \in A \implies f x \geq (0::'a)$ 
  and bound: eventually ( $\lambda X. \text{sum } f X \leq C$ ) (finite_subsets_at_top A)
  shows (f has_sum (SUP  $X \in \{X. X \subseteq A \wedge \text{finite } X\}. \text{sum } f X$ )) A
proof -
  from bound obtain X0
  where X0:  $X0 \subseteq A \text{ finite } X0 \wedge X. X0 \subseteq X \implies X \subseteq A \implies \text{finite } X \implies \text{sum } f X \leq C$ 
  by (force simp: eventually_finite_subsets_at_top)
  have bound':  $\text{sum } f X \leq C \text{ if } X \subseteq A \text{ finite } X \text{ for } X$ 
  proof -
    have  $\text{sum } f X \leq \text{sum } f (X \cup X0)$ 
    using that X0 assms(1) by (intro sum_mono2) auto
    also have  $\dots \leq C$ 
    by (simp add: X0 that)
  finally show ?thesis .
qed

```



```

hence bdd: bdd_above (sum f ' {X. X ⊆ A ∧ finite X})
  by (auto simp: bdd_above_def)

show ?thesis unfolding has_sum_def
proof (rule increasing_tendsto)
  show  $\forall_F X$  in finite_subsets_at_top A. sum f X ≤ Sup (sum f ' {X. X ⊆ A
  ∧ finite X})
  by (intro eventually_finite_subsets_at_top_weakI cSUP_upper[OF _ bdd])
auto
next
  fix y assume y < Sup (sum f ' {X. X ⊆ A ∧ finite X})
  then obtain X where X: X ⊆ A finite X y < sum f X
  by (subst (asm) less_cSUP_iff[OF _ bdd]) auto
  from X have eventually (λX'. X ⊆ X' ∧ X' ⊆ A ∧ finite X') (finite_subsets_at_top
  A)
  by (auto simp: eventually_finite_subsets_at_top)
  thus eventually (λX'. y < sum f X') (finite_subsets_at_top A)
  proof eventually_elim
  case (elim X')
  note ⟨y < sum f X⟩
  also have sum f X ≤ sum f X'
  using nonneg_elim by (intro sum_mono2) auto
  finally show ?case .
  qed
qed
qed

lemma nonneg_bounded_partial_sums_imp_summable_on:
  assumes nonneg:  $\bigwedge x. x \in A \implies f x \geq (0::'a)$ 
  and bound: eventually (λX. sum f X ≤ C) (finite_subsets_at_top A)
  shows f summable_on A
  using nonneg_bounded_partial_sums_imp_has_sum_SUP[OF assms] by (auto
  simp: summable_on_def)

end

context
  assumes SORT_CONSTRAINT('a :: {topological_comm_monoid_add, linorder_topology,
  ordered_comm_monoid_add, conditionally_complete_linorder})
begin

lemma summable_on_comparison_test:
  assumes f summable_on A and  $\bigwedge x. x \in A \implies g x \leq f x$  and  $\bigwedge x. x \in A \implies$ 
   $(0::'a) \leq g x$ 
  shows g summable_on A
proof -
  obtain C where C:  $\forall_F X$  in finite_subsets_at_top A. sum f X ≤ C
  using assms(1) summable_on_imp_bounded_partial_sums by blast
  show ?thesis

```


qed

lemma *infsum_insert*:

fixes $f :: _ \Rightarrow 'a :: \{\text{topological_comm_monoid_add, t2_space}\}$
 assumes f summable_on A $a \notin A$
 shows $\text{infsum } f (\text{insert } a A) = f a + \text{infsum } f A$
 by (meson assms has_sum_insert infsumI summable_iff_has_sum_infsum)

lemma *has_sum_SigmaD*:

fixes $f :: 'b \times 'c \Rightarrow 'a :: \{\text{topological_comm_monoid_add, t3_space}\}$
 assumes $\text{sum1}: (f \text{ has_sum } S) (\text{Sigma } A B)$
 assumes $\text{sum2}: \bigwedge x. x \in A \implies ((\lambda y. f (x, y)) \text{ has_sum } g x) (B x)$
 shows $(g \text{ has_sum } S) A$
 unfolding *has_sum_def tendsto_def eventually_finite_subsets_at_top*
 proof (safe, goal_cases)
 case (1 X)
 with *nhds_closed*[of $S X$] obtain X'
 where $X': S \in X'$ closed $X' X' \subseteq X$ eventually $(\lambda y. y \in X') (\text{nhds } S)$ by blast
 from $X' \setminus \{a\}$ obtain X'' where $X'': S \in X''$ open $X'' X'' \subseteq X'$
 by (auto simp: eventually_nhds)
 with sum1 obtain $Y :: ('b \times 'c)$ set
 where $Y: Y \subseteq \text{Sigma } A B$ finite Y
 $\bigwedge Z. Y \subseteq Z \implies Z \subseteq \text{Sigma } A B \implies \text{finite } Z \implies \text{sum } f Z \in X''$
 unfolding *has_sum_def tendsto_def eventually_finite_subsets_at_top* by
 force
 define $Y1 :: 'b$ set where $Y1 = \text{fst } ' Y$
 from Y have $Y1: Y1 \subseteq A$ by (auto simp: $Y1_def$)
 define $Y2 :: 'b \Rightarrow 'c$ set where $Y2 = (\lambda x. \{y. (x, y) \in Y\})$
 have $Y2: \text{finite } (Y2 x) Y2 x \subseteq B x$ if $x \in A$ for x
 using that $Y(1,2)$ unfolding $Y2_def$
 by (force simp: *image_iff intro: finite_subset*[of $_ \text{snd } ' Y$])+

 show ?case
 proof (rule *exI*[of $_ Y1$], safe, goal_cases)
 case (3 Z)
 define H where $H = (\text{INF } x \in Z. \text{filtercomap } (\lambda p. p x) (\text{finite_subsets_at_top } (B x)))$

 have $\text{sum } g Z \in X'$
 proof (rule *Lim_in_closed_set*)
 show closed X' by fact
 next
 show $((\lambda B'. \text{sum } (\lambda x. \text{sum } (\lambda y. f (x, y)) (B' x)) Z) \longrightarrow \text{sum } g Z) H$
 unfolding H_def
 proof (intro *tendsto_sum filterlim_INF'*)
 fix x assume $x: x \in Z$
 with 3 have $x \in A$ by auto
 from sum2 [OF this] have $(\text{sum } (\lambda y. f (x, y)) \longrightarrow g x) (\text{finite_subsets_at_top } (B x))$

```

    by (simp add: has_sum_def)
  thus (( $\lambda B'. \text{sum } (\lambda y. f(x, y)) (B' x) \longrightarrow g x$ )
    (filtercomap ( $\lambda p. p x$ ) (finite_subsets_at_top (B x))))
    by (rule filterlim_compose[OF _ filterlim_filtercomap])
  qed auto
next
show  $\forall_F h \text{ in } H. \text{sum } (\lambda x. \text{sum } (\lambda y. f(x, y)) (h x)) Z \in X'$ 
  unfolding H_def
proof (subst eventually_INF_finite[OF  $\langle \text{finite } Z \rangle$ ], rule exI, safe)
  fix x assume x:  $x \in Z$ 
  hence x':  $x \in A$  using  $\exists$  by auto
  show eventually ( $\lambda h. \text{finite } (h x) \wedge Y2 x \subseteq h x \wedge h x \subseteq B x$ )
    (filtercomap ( $\lambda p. p x$ ) (finite_subsets_at_top (B x))) using  $\exists Y2$ [OF
x]
    by (intro eventually_filtercomapI)
      (auto simp: eventually_finite_subsets_at_top intro: exI[of _ Y2 x])
next
fix h
assume *:  $\forall x \in Z. \text{finite } (h x) \wedge Y2 x \subseteq h x \wedge h x \subseteq B x$ 
hence  $\text{sum } (\lambda x. \text{sum } (\lambda y. f(x, y)) (h x)) Z = \text{sum } f (\text{Sigma } Z h)$ 
  using  $\langle \text{finite } Z \rangle$  by (subst sum.Sigma) auto
also have  $\dots \in X''$ 
  using *  $\exists Y(1,2)$  by (intro Y; force simp: Y1_def Y2_def)
also have  $X'' \subseteq X'$  by fact
finally show  $\text{sum } (\lambda x. \text{sum } (\lambda y. f(x, y)) (h x)) Z \in X'$ .
qed
next
have  $H = (\text{INF } x \in \text{SIGMA } x:Z. \{X. \text{finite } X \wedge X \subseteq B x\}. \text{principal } \{y. \text{finite } (y (fst x)) \wedge \text{snd } x \subseteq y (fst x) \wedge y (fst x) \subseteq B (fst x)\})$ 
  unfolding H_def finite_subsets_at_top_def filtercomap_INF filtercomap_principal
  by (simp add: INF_Sigma)
also have  $\dots \neq \text{bot}$ 
proof (rule INF_filter_not_bot, subst INF_principal_finite, goal_cases)
  case (2 X)
  define H' where
     $H' = (\bigcap x \in X. \{y. \text{finite } (y (fst x)) \wedge \text{snd } x \subseteq y (fst x) \wedge y (fst x) \subseteq B (fst x)\})$ 
  from 2 have  $(\lambda x. \bigcup (y, Y) \in X. \text{if } x = y \text{ then } Y \text{ else } \{\}) \in H'$ 
    by (force split: if_splits simp: H'_def)
  hence  $H' \neq \{\}$  by blast
  thus  $\text{principal } H' \neq \text{bot}$  by (simp add: principal_eq_bot_iff)
qed
finally show  $H \neq \text{bot}$ .
qed
also have  $X' \subseteq X$  by fact
finally show  $\text{sum } g Z \in X$ .
qed (insert Y(1,2), auto simp: Y1_def)
qed

```

lemma *has_sum_unique*:

fixes $f :: _ \Rightarrow 'a :: \{topological_comm_monoid_add, t2_space\}$
assumes $(f \text{ has_sum } x) \ A \ (f \text{ has_sum } y) \ A$
shows $x = y$
using *assms unfolding has_sum_def using tendsto_unique finite_subsets_at_top_neq_bot*
by *blast*

lemma *has_sum_SigmaI*:

fixes $f :: _ \Rightarrow 'a :: \{topological_comm_monoid_add, t3_space\}$
assumes $f: \bigwedge x. x \in A \implies ((\lambda y. f(x, y)) \text{ has_sum } g \ x) \ (B \ x)$
assumes $g: (g \text{ has_sum } S) \ A$
assumes *summable*: $f \text{ summable_on } \Sigma A \ B$
shows $(f \text{ has_sum } S) \ (\Sigma A \ B)$
by *(metis f g has_sum_SigmaD has_sum_infsun has_sum_unique local.summable)*

lemma *summable_on_SigmaD1*:

fixes $f :: _ \Rightarrow _ \Rightarrow 'a :: \{complete_uniform_space, uniform_topological_group_add, ab_group_add, topological_comm_monoid_add\}$
assumes $f: (\lambda(x,y). f \ x \ y) \text{ summable_on } \Sigma A \ B$
assumes $x: x \in A$
shows $f \ x \text{ summable_on } B \ x$

proof –

have $(\lambda(x,y). f \ x \ y) \text{ summable_on } \Sigma \{x\} \ B$
using f **by** *(rule summable_on_subset) (use x in auto)*
also have $?this \longleftrightarrow ((\lambda y. f \ x \ y) \circ \text{snd}) \text{ summable_on } \Sigma \{x\} \ B$
by *(intro summable_on_cong) auto*
also have $\dots \longleftrightarrow (\lambda y. f \ x \ y) \text{ summable_on } \text{snd} \ ' \Sigma \{x\} \ B$
by *(intro summable_on_reindex [symmetric] inj_onI) auto*
also have $\text{snd} \ ' \Sigma \{x\} \ B = B \ x$
by *(force simp: Sigma_def)*
finally show $?thesis$.

qed

lemma *has_sum_swap*:

$(f \text{ has_sum } S) \ (A \times B) \longleftrightarrow ((\lambda(x,y). f \ (y,x)) \text{ has_sum } S) \ (B \times A)$

proof –

have $\text{bij_betw} \ (\lambda(x,y). (y,x)) \ (B \times A) \ (A \times B)$
by *(rule bij_betwI[of _ _ _ \lambda(x,y). (y,x)]) auto*
from *has_sum_reindex_bij_betw[OF this, where f = f]* **show** $?thesis$
by *(simp add: case_prod_unfold)*

qed

lemma *summable_on_swap*:

$f \text{ summable_on } (A \times B) \longleftrightarrow (\lambda(x,y). f \ (y,x)) \text{ summable_on } (B \times A)$
by *(metis has_sum_swap summable_on_def)*

lemma *has_sum_cmult_right_iff*:

```

fixes  $c :: 'a :: \{topological\_semigroup\_mult, field\}$ 
assumes  $c \neq 0$ 
shows  $((\lambda x. c * f x) \text{ has\_sum } S) A \longleftrightarrow (f \text{ has\_sum } (S / c)) A$ 
using  $\text{has\_sum\_cmult\_right}[of f A S/c c]$ 
 $\text{has\_sum\_cmult\_right}[of \lambda x. c * f x A S \text{ inverse } c]$  assms
by (auto simp: field_simps)

```

```

lemma has_sum_cmult_left_iff:
fixes  $c :: 'a :: \{topological\_semigroup\_mult, field\}$ 
assumes  $c \neq 0$ 
shows  $((\lambda x. f x * c) \text{ has\_sum } S) A \longleftrightarrow (f \text{ has\_sum } (S / c)) A$ 
by (smt (verit, best) assms has_sum_cmult_right_iff has_sum_cong mult.commute)

```

```

lemma finite_nonzero_values_imp_summable_on:
assumes  $\text{finite } \{x \in X. f x \neq 0\}$ 
shows  $f \text{ summable\_on } X$ 
by (smt (verit, del_insts) Diff_iff assms mem_Collect_eq summable_on_cong_neutral summable_on_finite)

```

```

lemma summable_on_of_int_iff:
 $(\lambda x :: 'a. \text{of\_int } (f x) :: 'b :: \text{real\_normed\_algebra\_1}) \text{ summable\_on } A \longleftrightarrow f$ 
 $\text{summable\_on } A$ 

```

proof

```

assume  $f \text{ summable\_on } A$ 
thus  $(\lambda x. \text{of\_int } (f x)) \text{ summable\_on } A$ 
by (rule summable_on_homomorphism) auto

```

next

```

assume  $(\lambda x. \text{of\_int } (f x) :: 'b) \text{ summable\_on } A$ 
then obtain  $S$  where  $((\lambda x. \text{of\_int } (f x) :: 'b) \text{ has\_sum } S) A$ 
by (auto simp: summable_on_def)
hence  $(\text{sum } (\lambda x. \text{of\_int } (f x) :: 'b) \longrightarrow S) (\text{finite\_subsets\_at\_top } A)$ 
unfolding has_sum_def .
moreover have  $1/2 > (0 :: \text{real})$ 
by auto

```

```

ultimately have eventually  $(\lambda X. \text{dist } (\text{sum } (\lambda x. \text{of\_int } (f x) :: 'b) X) S < 1/2)$ 
 $(\text{finite\_subsets\_at\_top } A)$ 

```

```

unfolding tendsto_iff by blast

```

```

then obtain  $X$  where  $\text{finite } X \ X \subseteq A$ 

```

```

 $\bigwedge Y. \text{finite } Y \implies X \subseteq Y \implies Y \subseteq A \implies \text{dist } (\text{sum } (\lambda x. \text{of\_int } (f x)) Y) S$ 
 $< 1/2$ 

```

```

unfolding eventually_finite_subsets_at_top by metis

```

```

have  $\text{sum } f Y = \text{sum } f X$  if  $\text{finite } Y \ X \subseteq Y \ Y \subseteq A$  for  $Y$ 

```

proof –

```

have  $\text{dist } (\text{sum } (\lambda x. \text{of\_int } (f x)) X) S < 1/2$ 

```

```

by (intro X) auto

```

```

moreover have  $\text{dist } (\text{sum } (\lambda x. \text{of\_int } (f x)) Y) S < 1/2$ 

```

```

by (intro X that)

```

```

ultimately have  $\text{dist } (\text{sum } (\lambda x. \text{of\_int } (f x)) X) (\text{sum } (\lambda x. \text{of\_int } (f x) :: 'b))$ 

```

```

Y) <
      1/2 + 1/2
  using dist_triangle_less_add by blast
  thus ?thesis
    by (simp add: dist_norm_flip: of_int_sum of_int_diff)
qed
then have {x∈A. f x ≠ 0} ⊆ X
  by (smt (verit) X finite_insert insert_iff mem_Collect_eq subset_eq sum.insert)
with ‹finite X› have finite {x∈A. f x ≠ 0}
  using finite_subset by blast
thus f summable_on A
  by (rule finite_nonzero_values_imp_summable_on)
qed

```

```

lemma summable_on_of_nat_iff:
  (λx::'a. of_nat (f x) :: 'b :: real_normed_algebra_1) summable_on A ⟷ f
  summable_on A
proof
  assume f summable_on A
  thus (λx. of_nat (f x) :: 'b) summable_on A
    by (rule summable_on_homomorphism) auto
next
  assume (λx. of_nat (f x) :: 'b) summable_on A
  hence (λx. of_int (int (f x)) :: 'b) summable_on A
    by simp
  also have ?this ⟷ (λx. int (f x)) summable_on A
    by (rule summable_on_of_int_iff)
  also have ... ⟷ f summable_on A
    by (simp add: summable_on_discrete_iff)
  finally show f summable_on A .
qed

```

```

lemma infsum_of_nat:
  infsum (λx::'a. of_nat (f x) :: 'b :: {real_normed_algebra_1}) A = of_nat
  (infsum f A)
  by (metis has_sum_infsum has_sum_of_nat infsumI infsum_def of_nat_0
  summable_on_of_nat_iff)

```

```

lemma infsum_of_int:
  infsum (λx::'a. of_int (f x) :: 'b :: {real_normed_algebra_1}) A = of_int (infsum
  f A)
  by (metis has_sum_infsum has_sum_of_int infsumI infsum_not_exists of_int_0
  summable_on_of_int_iff)

```

```

lemma summable_on_SigmaI:
  fixes f :: _ ⇒ 'a :: {linorder_topology, ordered_comm_monoid_add, topologi-
  cal_comm_monoid_add,
  conditionally_complete_linorder}

```

1516

```

assumes f:  $\bigwedge x. x \in A \implies ((\lambda y. f (x, y)) \text{ has\_sum } g x) (B x)$ 
assumes g: g summable_on A
assumes f_nonneg:  $\bigwedge x y. x \in A \implies y \in B x \implies f (x, y) \geq (0 :: 'a)$ 
shows f summable_on Sigma A B
proof -
  have g_nonneg: g x  $\geq 0$  if x  $\in A$  for x
    using f by (rule has_sum_nonneg) (use f_nonneg that in auto)
  obtain C where C: eventually ( $\lambda X. \text{sum } g X \leq C$ ) (finite_subsets_at_top A)
    using summable_on_imp_bounded_partial_sums[OF g] by blast

have sum_g_le: sum g X  $\leq C$  if X: finite X X  $\subseteq A$  for X
proof -
  from C obtain X' where X':
    finite X' X'  $\subseteq A \wedge Y. \text{finite } Y \implies X' \subseteq Y \implies Y \subseteq A \implies \text{sum } g Y \leq C$ 
  unfolding eventually_finite_subsets_at_top by metis
  have sum_g_X  $\leq \text{sum } g (X \cup X')$ 
    using X X' by (intro sum_mono2 g_nonneg) auto
  also have ...  $\leq C$ 
    using X X'(1,2) by (intro X'(3)) auto
  finally show ?thesis .
qed

have sum_f_Y  $\leq C$  if Y: finite Y Y  $\subseteq \text{Sigma } A B$  for Y
proof -
  define Y1 and Y2 where Y1 = fst ' Y and Y2 = ( $\lambda x. \text{snd ' } \{z \in Y. \text{fst } z = x\}$ )
  have Y12: Y = Sigma Y1 Y2
    unfolding Y1_def Y2_def by force
  have [intro]: finite Y1  $\wedge x. x \in Y1 \implies \text{finite } (Y2 x)$ 
    using Y unfolding Y1_def Y2_def by auto
  have Y12_subset: Y1  $\subseteq A \wedge x. Y2 x \subseteq B x$ 
    using Y by (auto simp: Y1_def Y2_def)

have sum_f_Y = sum f (Sigma Y1 Y2)
  by (simp add: Y12)
also have ... = ( $\sum x \in Y1. \sum y \in Y2 x. f (x, y)$ )
  by (subst sum.Sigma) auto
also have ...  $\leq (\sum x \in Y1. g x)$ 
proof (rule sum_mono)
  fix x assume x: x  $\in Y1$ 
  show ( $\sum y \in Y2 x. f (x, y)$ )  $\leq g x$ 
  proof (rule has_sum_mono')
    show (( $\lambda y. f (x, y)$ ) has_sum ( $\sum y \in Y2 x. f (x, y)$ )) (Y2 x)
      using x by (intro has_sum_finite) auto
    show (( $\lambda y. f (x, y)$ ) has_sum g x) (B x)
      by (rule f) (use x Y12_subset in auto)
    show f (x, y)  $\geq 0$  if y  $\in B x - Y2 x$  for y
      using x that Y12_subset by (intro f_nonneg) auto
  qed (use Y12_subset in auto)

```



```

qed
also have ... ≤ C
  using Y12_subset by (intro sum_g_le) auto
finally show ?thesis .
qed

```

```

hence  $\forall_F X$  in finite_subsets_at_top (Sigma A B). sum f X ≤ C
  unfolding eventually_finite_subsets_at_top by auto
thus ?thesis
  by (metis SigmaE f_nonneg nonneg_bounded_partial_sums_imp_summable_on)
qed

```

lemma *summable_on_UnionI*:

```

fixes f :: _ ⇒ 'a :: {linorder_topology, ordered_comm_monoid_add, topological_comm_monoid_add,
  conditionally_complete_linorder}
assumes f:  $\bigwedge x. x \in A \implies (f \text{ has\_sum } g \ x) \ (B \ x)$ 
assumes g: g summable_on A
assumes f_nonneg:  $\bigwedge x \ y. x \in A \implies y \in B \ x \implies f \ y \geq (0 :: 'a)$ 
assumes disj: disjoint_family_on B A
shows f summable_on ( $\bigcup_{x \in A}. B \ x$ )
proof -
  have f ∘ snd summable_on Sigma A B
    using assms by (intro summable_on_SigmaI[where g = g]) auto
  also have ?this  $\longleftrightarrow$  f summable_on (snd ' Sigma A B) using assms
    by (subst summable_on_reindex; force simp: disjoint_family_on_def inj_on_def)
  also have snd ' (Sigma A B) = ( $\bigcup_{x \in A}. B \ x$ )
    by force
  finally show ?thesis .
qed

```

lemma *summable_on_SigmaD*:

```

fixes f :: 'a × 'b ⇒ 'c :: {topological_comm_monoid_add, t3_space}
assumes sum1: f summable_on (Sigma A B)
assumes sum2:  $\bigwedge x. x \in A \implies (\lambda y. f \ (x, \ y)) \text{ summable\_on } (B \ x)$ 
shows  $(\lambda x. \text{infsum } (\lambda y. f \ (x, \ y)) \ (B \ x)) \text{ summable\_on } A$ 
using assms unfolding summable_on_def
by (smt (verit, del_insts) assms has_sum_SigmaD has_sum_cong has_sum_infsum)

```

lemma *summable_on_UnionD*:

```

fixes f :: 'a ⇒ 'c :: {topological_comm_monoid_add, t3_space}
assumes sum1: f summable_on ( $\bigcup_{x \in A}. B \ x$ )
assumes sum2:  $\bigwedge x. x \in A \implies f \text{ summable\_on } (B \ x)$ 
assumes disj: disjoint_family_on B A
shows  $(\lambda x. \text{infsum } f \ (B \ x)) \text{ summable\_on } A$ 
proof -
  have ( $\bigcup_{x \in A}. B \ x$ ) = snd ' Sigma A B
    by (force simp: Sigma_def)
  with sum1 have f summable_on (snd ' Sigma A B)

```

1518

```

    by simp
  also have ?this  $\longleftrightarrow$  (f  $\circ$  snd) summable_on (Sigma A B)
    using disj by (intro summable_on_reindex inj_onI) (force simp: disjoint_family_on_def)
  finally show  $(\lambda x. \text{infsum } f (B x))$  summable_on A
    using summable_on_SigmaD[of f  $\circ$  snd A B] sum2 by simp
qed

```

lemma *summable_on_Union_iff*:

```

  fixes f ::  $\_ \Rightarrow 'a :: \{\text{linorder\_topology, ordered\_comm\_monoid\_add, topological\_comm\_monoid\_add, conditionally\_complete\_linorder, t3\_space}\}$ 
  assumes f:  $\bigwedge x. x \in A \implies (f \text{ has\_sum } g x) (B x)$ 
  assumes f_nonneg:  $\bigwedge x y. x \in A \implies y \in B x \implies f y \geq 0$ 
  assumes disj: disjoint_family_on B A
  shows f summable_on  $(\bigcup_{x \in A}. B x) \longleftrightarrow g$  summable_on A

```

proof

```

  assume g summable_on A
  thus f summable_on  $(\bigcup_{x \in A}. B x)$ 
    using summable_on_UnionI[of A f B g] assms by auto
next
  assume f summable_on  $(\bigcup_{x \in A}. B x)$ 
  hence  $(\lambda x. \text{infsum } f (B x))$  summable_on A
    using assms by (intro summable_on_UnionD) (auto dest: has_sum_imp_summable)
  also have ?this  $\longleftrightarrow g$  summable_on A
    using assms by (intro summable_on_cong) (auto simp: infsumI)
  finally show g summable_on A .
qed

```

lemma *has_sum_Sigma'*:

```

  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  and f ::  $\langle 'a \times 'b \Rightarrow 'c :: \{\text{comm\_monoid\_add, uniform\_space, uniform\_topological\_group\_add}\} \rangle$ 
  assumes summableAB: (f has_sum a) (Sigma A B)
  assumes summableB:  $\langle \bigwedge x. x \in A \implies ((\lambda y. f (x, y)) \text{ has\_sum } (b x)) (B x) \rangle$ 
  shows (b has_sum a) A
  by (intro has_sum_Sigma[OF _ assms] uniformly_continuous_add)

```

lemma *abs_summable_on_comparison_test'*:

```

  assumes g summable_on A
  assumes  $\bigwedge x. x \in A \implies \text{norm } (f x) \leq g x$ 
  shows  $(\lambda x. \text{norm } (f x))$  summable_on A
proof (rule Infinite_Sum.abs_summable_on_comparison_test)
  have g summable_on A  $\longleftrightarrow (\lambda x. \text{norm } (g x))$  summable_on A
    by (metis summable_on_iff_abs_summable_on_real)
  with assms show  $(\lambda x. \text{norm } (g x))$  summable_on A by blast
qed (use assms in fastforce)

```

lemma *has_sum_geometric_from_1*:

```

  fixes z :: 'a :: {real_normed_field, banach}
  assumes norm z < 1

```

```

  shows ((λn. z ^ n) has_sum (z / (1 - z))) {1..}
proof -
  have [simp]: z ≠ 1
    using assms by auto
  have (λn. z ^ Suc n) sums (1 / (1 - z) - 1)
    using geometric_sums[of z] assms by (subst sums_Suc_iff) auto
  also have 1 / (1 - z) - 1 = z / (1 - z)
    by (auto simp: field_simps)
  finally have (λn. z ^ Suc n) sums (z / (1 - z)) .
  moreover have summable (λn. norm (z ^ Suc n))
    using assms
  by (subst summable_Suc_iff) (auto simp: norm_power intro!: summable_geometric)
  ultimately have ((λn. z ^ Suc n) has_sum (z / (1 - z))) UNIV
    by (intro norm_summable_imp_has_sum)
  also have ?this ⟷ ?thesis
    by (intro has_sum_reindex_bij_witness[of _ λn. n-1 λn. n+1]) auto
  finally show ?thesis .
qed

```

```

lemma has_sum_divide_const:
  fixes f :: 'a ⇒ 'b :: {topological_semigroup_mult, field, semiring_0}
  shows (f has_sum S) A ⟹ ((λx. f x / c) has_sum (S / c)) A
  using has_sum_cmult_right[of f A S inverse c] by (simp add: field_simps)

```

```

lemma has_sum_uminusI:
  fixes f :: 'a ⇒ 'b :: {topological_semigroup_mult, ring_1}
  shows (f has_sum S) A ⟹ ((λx. -f x) has_sum (-S)) A
  using has_sum_cmult_right[of f A S -1] by simp

```

end

6.9 Ordered Euclidean Space

```

theory Ordered_Euclidean_Space
imports
  Convex_Euclidean_Space Abstract_Limits
  HOL-Library.Product_Order
begin

```

An ordering on euclidean spaces that will allow us to talk about intervals

```

class ordered_euclidean_space = ord + inf + sup + abs + Inf + Sup + eu-
clidean_space +
  assumes eucl_le: x ≤ y ⟷ (∀ i ∈ Basis. x · i ≤ y · i)
  assumes eucl_less_le_not_le: x < y ⟷ x ≤ y ∧ ¬ y ≤ x
  assumes eucl_inf: inf x y = (∑ i ∈ Basis. inf (x · i) (y · i) *R i)
  assumes eucl_sup: sup x y = (∑ i ∈ Basis. sup (x · i) (y · i) *R i)
  assumes eucl_Inf: Inf X = (∑ i ∈ Basis. (INF x ∈ X. x · i) *R i)
  assumes eucl_Sup: Sup X = (∑ i ∈ Basis. (SUP x ∈ X. x · i) *R i)
  assumes eucl_abs: |x| = (∑ i ∈ Basis. |x · i| *R i)

```

1520

begin

subclass *order*

by *standard*

(*auto simp: eucl_le eucl_less_le_not_le intro!: euclidean_eqI antisym intro: order.trans*)

subclass *ordered_ab_group_add_abs*

by *standard (auto simp: eucl_le inner_add_left eucl_abs abs_leI)*

subclass *ordered_real_vector*

by *standard (auto simp: eucl_le intro!: mult_left_mono mult_right_mono)*

subclass *lattice*

by *standard (auto simp: eucl_inf eucl_sup eucl_le)*

subclass *distrib_lattice*

by *standard (auto simp: eucl_inf eucl_sup sup_inf_distrib1 intro!: euclidean_eqI)*

subclass *conditionally_complete_lattice*

proof

fix *z::'a and X::'a set*

assume $X \neq \{\}$

hence $\bigwedge i. (\lambda x. x \cdot i) \text{ ' } X \neq \{\}$ **by** *simp*

thus $(\bigwedge x. x \in X \implies z \leq x) \implies z \leq \text{Inf } X (\bigwedge x. x \in X \implies x \leq z) \implies \text{Sup } X \leq z$

by (*auto simp: eucl_Inf eucl_Sup eucl_le intro!: cInf_greatest cSup_least*)

qed (*force intro!: cInf_lower cSup_upper*

simp: bdd_below_def bdd_above_def preorder_class.bdd_below_def preorder_class.bdd_above_def eucl_Inf eucl_Sup eucl_le)⁺

lemma *inner_Basis_inf_left*: $i \in \text{Basis} \implies \text{inf } x y \cdot i = \text{inf } (x \cdot i) (y \cdot i)$

and *inner_Basis_sup_left*: $i \in \text{Basis} \implies \text{sup } x y \cdot i = \text{sup } (x \cdot i) (y \cdot i)$

by (*simp_all add: eucl_inf eucl_sup inner_sum_left inner_Basis if_distrib cong: if_cong*)

lemma *inner_Basis_INF_left*: $i \in \text{Basis} \implies (\text{INF } x \in X. f x) \cdot i = (\text{INF } x \in X. f x \cdot i)$

and *inner_Basis_SUP_left*: $i \in \text{Basis} \implies (\text{SUP } x \in X. f x) \cdot i = (\text{SUP } x \in X. f x \cdot i)$

using *eucl_Sup [of f ' X] eucl_Inf [of f ' X]* **by** (*simp_all add: image_comp*)

lemma *abs_inner*: $i \in \text{Basis} \implies |x| \cdot i = |x \cdot i|$

by (*auto simp: eucl_abs*)

lemma

abs_scaleR: $|a *_R b| = |a| *_R |b|$

by (*auto simp: eucl_abs abs_mult intro!: euclidean_eqI*)

lemma *interval_inner_leI*:
assumes $x \in \{a .. b\}$ $0 \leq i$
shows $a \cdot i \leq x \cdot i$ $x \cdot i \leq b \cdot i$
using *assms*
unfolding *euclidean_inner*[of a i] *euclidean_inner*[of x i] *euclidean_inner*[of b i]
by (*auto intro!*: *ordered_comm_monoid_add_class.sum_mono* *mult_right_mono* *simp: eucl_le*)

lemma *inner_nonneg_nonneg*:
shows $0 \leq a \implies 0 \leq b \implies 0 \leq a \cdot b$
using *interval_inner_leI*[of 0 a b]
by *auto*

lemma *inner_Basis_mono*:
shows $a \leq b \implies c \in \text{Basis} \implies a \cdot c \leq b \cdot c$
by (*simp add: eucl_le*)

lemma *Basis_nonneg*[*intro, simp*]: $i \in \text{Basis} \implies 0 \leq i$
by (*auto simp: eucl_le inner_Basis*)

lemma *Sup_eq_maximum_componentwise*:
fixes $s :: 'a$ *set*
assumes i : $\bigwedge b. b \in \text{Basis} \implies X \cdot b = i \cdot b \cdot b$
assumes sup : $\bigwedge b x. b \in \text{Basis} \implies x \in s \implies x \cdot b \leq X \cdot b$
assumes i_s : $\bigwedge b. b \in \text{Basis} \implies (i \cdot b \cdot b) \in (\lambda x. x \cdot b) ' s$
shows $\text{Sup } s = X$
using *assms*
unfolding *eucl_Sup* *euclidean_representation_sum*
by (*auto intro!*: *conditionally_complete_lattice_class.cSup_eq_maximum*)

lemma *Inf_eq_minimum_componentwise*:
assumes i : $\bigwedge b. b \in \text{Basis} \implies X \cdot b = i \cdot b \cdot b$
assumes sup : $\bigwedge b x. b \in \text{Basis} \implies x \in s \implies X \cdot b \leq x \cdot b$
assumes i_s : $\bigwedge b. b \in \text{Basis} \implies (i \cdot b \cdot b) \in (\lambda x. x \cdot b) ' s$
shows $\text{Inf } s = X$
using *assms*
unfolding *eucl_Inf* *euclidean_representation_sum*
by (*auto intro!*: *conditionally_complete_lattice_class.cInf_eq_minimum*)

end

proposition *compact_attains_Inf_componentwise*:
fixes $b :: 'a :: \text{ordered_euclidean_space}$
assumes $b \in \text{Basis}$ **assumes** $X \neq \{\}$ *compact* X
obtains x **where** $x \in X$ $x \cdot b = \text{Inf } X \cdot b \wedge \forall y. y \in X \implies x \cdot b \leq y \cdot b$
proof *atomize_elim*
let $?proj = (\lambda x. x \cdot b) ' X$

```

from assms have compact ?proj ?proj ≠ {}
  by (auto intro!: compact_continuous_image_continuous_intros)
from compact_attains_inf[OF this]
obtain s x
  where s:  $s \in (\lambda x. x \cdot b) \text{ ' } X \wedge t. t \in (\lambda x. x \cdot b) \text{ ' } X \implies s \leq t$ 
    and x:  $x \in X \ s = x \cdot b \wedge y. y \in X \implies x \cdot b \leq y \cdot b$ 
  by auto
hence Inf ?proj = x · b
  by (auto intro!: conditionally_complete_lattice_class.cInf_eq_minimum)
hence  $x \cdot b = \text{Inf } X \cdot b$ 
  by (auto simp: eucl_Inf_inner_sum_left inner_Basis if_distrib  $\langle b \in \text{Basis} \rangle$ 
    cong: if_cong)
with x show  $\exists x. x \in X \wedge x \cdot b = \text{Inf } X \cdot b \wedge (\forall y. y \in X \longrightarrow x \cdot b \leq y \cdot b)$ 
by blast
qed

```

proposition

```

compact_attains_Sup_componentwise:
fixes b::'a::ordered_euclidean_space
assumes  $b \in \text{Basis}$  assumes  $X \neq \{\}$  compact X
obtains x where  $x \in X \ x \cdot b = \text{Sup } X \cdot b \wedge y. y \in X \implies y \cdot b \leq x \cdot b$ 
proof atomize_elim
let ?proj =  $(\lambda x. x \cdot b) \text{ ' } X$ 
from assms have compact ?proj ?proj ≠ {}
  by (auto intro!: compact_continuous_image_continuous_intros)
from compact_attains_sup[OF this]
obtain s x
  where s:  $s \in (\lambda x. x \cdot b) \text{ ' } X \wedge t. t \in (\lambda x. x \cdot b) \text{ ' } X \implies t \leq s$ 
    and x:  $x \in X \ s = x \cdot b \wedge y. y \in X \implies y \cdot b \leq x \cdot b$ 
  by auto
hence Sup ?proj = x · b
  by (auto intro!: cSup_eq_maximum)
hence  $x \cdot b = \text{Sup } X \cdot b$ 
  by (auto simp: eucl_Sup[where 'a='a'] inner_sum_left inner_Basis if_distrib
     $\langle b \in \text{Basis} \rangle$ 
    cong: if_cong)
with x show  $\exists x. x \in X \wedge x \cdot b = \text{Sup } X \cdot b \wedge (\forall y. y \in X \longrightarrow y \cdot b \leq x \cdot b)$ 
by blast
qed

```

lemma *tendsto_sup*[*tendsto_intros*]:

```

fixes X :: 'a  $\Rightarrow$  'b::ordered_euclidean_space
assumes  $(X \longrightarrow x)$  net  $(Y \longrightarrow y)$  net
shows  $((\lambda i. \text{sup } (X \ i) (Y \ i)) \longrightarrow \text{sup } x \ y)$  net
  unfolding sup_max eucl_sup by (intro assms tendsto_intros)

```

lemma *tendsto_inf*[*tendsto_intros*]:

```

fixes X :: 'a  $\Rightarrow$  'b::ordered_euclidean_space
assumes  $(X \longrightarrow x)$  net  $(Y \longrightarrow y)$  net

```

shows $((\lambda i. \text{inf } (X \ i) \ (Y \ i)) \longrightarrow \text{inf } x \ y) \text{ net}$
unfolding inf_min eucl_inf **by** $(\text{intro assms tendsto_intros})$

lemma $\text{tendsto_Inf}[\text{tendsto_intros}]$:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{ordered_euclidean_space}$
assumes $\text{finite } K \wedge i. i \in K \implies ((\lambda x. f \ x \ i) \longrightarrow l \ i) \ F$
shows $((\lambda x. \text{Inf } (f \ x \ 'K)) \longrightarrow \text{Inf } (l \ 'K)) \ F$
using assms
by $(\text{induction } K \ \text{rule: finite_induct}) \ (\text{auto simp: cInf_insert_If tendsto_inf})$

lemma $\text{tendsto_Sup}[\text{tendsto_intros}]$:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{ordered_euclidean_space}$
assumes $\text{finite } K \wedge i. i \in K \implies ((\lambda x. f \ x \ i) \longrightarrow l \ i) \ F$
shows $((\lambda x. \text{Sup } (f \ x \ 'K)) \longrightarrow \text{Sup } (l \ 'K)) \ F$
using assms
by $(\text{induction } K \ \text{rule: finite_induct}) \ (\text{auto simp: cSup_insert_If tendsto_sup})$

lemma $\text{continuous_map_Inf} \ [\text{continuous_intros}]$:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{ordered_euclidean_space}$
assumes $\text{finite } K \wedge i. i \in K \implies \text{continuous_map } X \ \text{euclidean } (\lambda x. f \ x \ i)$
shows $\text{continuous_map } X \ \text{euclidean } (\lambda x. \text{INF } i \in K. f \ x \ i)$
using assms **by** $(\text{simp add: continuous_map_atin tendsto_Inf})$

lemma $\text{continuous_map_Sup} \ [\text{continuous_intros}]$:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{ordered_euclidean_space}$
assumes $\text{finite } K \wedge i. i \in K \implies \text{continuous_map } X \ \text{euclidean } (\lambda x. f \ x \ i)$
shows $\text{continuous_map } X \ \text{euclidean } (\lambda x. \text{SUP } i \in K. f \ x \ i)$
using assms **by** $(\text{simp add: continuous_map_atin tendsto_Sup})$

lemma $\text{tendsto_componentwise_max}$:

assumes $f: (f \longrightarrow l) \ F$ **and** $g: (g \longrightarrow m) \ F$
shows $((\lambda x. (\sum i \in \text{Basis}. \text{max } (f \ x \cdot i) \ (g \ x \cdot i) \ *_{\mathbb{R}} \ i)) \longrightarrow (\sum i \in \text{Basis}. \text{max } (l \cdot i) \ (m \cdot i) \ *_{\mathbb{R}} \ i)) \ F$
by $(\text{intro tendsto_intros assms})$

lemma $\text{tendsto_componentwise_min}$:

assumes $f: (f \longrightarrow l) \ F$ **and** $g: (g \longrightarrow m) \ F$
shows $((\lambda x. (\sum i \in \text{Basis}. \text{min } (f \ x \cdot i) \ (g \ x \cdot i) \ *_{\mathbb{R}} \ i)) \longrightarrow (\sum i \in \text{Basis}. \text{min } (l \cdot i) \ (m \cdot i) \ *_{\mathbb{R}} \ i)) \ F$
by $(\text{intro tendsto_intros assms})$

instance $\text{real} :: \text{ordered_euclidean_space}$

by standard auto

lemma in_Basis_prod_iff :

fixes $i :: 'a :: \text{euclidean_space} * 'b :: \text{euclidean_space}$
shows $i \in \text{Basis} \iff \text{fst } i = 0 \wedge \text{snd } i \in \text{Basis} \vee \text{snd } i = 0 \wedge \text{fst } i \in \text{Basis}$
by $(\text{cases } i) \ (\text{auto simp: Basis_prod_def})$

1524

instantiation *prod* :: (*abs*, *abs*) *abs*
begin

definition $|x| = (|fst\ x|, |snd\ x|)$

instance ..

end

instance *prod* :: (*ordered_euclidean_space*, *ordered_euclidean_space*) *ordered_euclidean_space*
by *standard*
 (*auto intro!*: *add_mono simp add: euclidean_representation_sum' Ball_def*
inner_prod_def
 in_Basis_prod_iff inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left
Inf_prod_def
 inner_Basis_SUP_left Sup_prod_def less_prod_def less_eq_prod_def eucl_le[**where**
'a='a]
 eucl_le[**where 'a='b]** *abs_prod_def abs_inner*)

Instantiation for intervals on *ordered_euclidean_space*

proposition

fixes *a* :: '*a*::*ordered_euclidean_space*
shows *cbx_interval*: *cbx a b* = {*a..b*}
 and *interval_cbx*: {*a..b*} = *cbx a b*
 and *eucl_le_atMost*: {*x. $\forall i \in Basis. x \cdot i \leq a \cdot i$* } = {*..a*}
 and *eucl_le_atLeast*: {*x. $\forall i \in Basis. a \cdot i \leq x \cdot i$* } = {*a..*}
by (*auto simp: eucl_le*[**where 'a='a]** *eucl_less_def box_def cbx_def*)

lemma *sums_vec_nth* :

assumes *f sums a*
shows ($\lambda x. f\ x\ \$\ i$) *sums a* \$ *i*
using *assms unfolding sums_def*
by (*auto dest: tendsto_vec_nth* [**where $i=i$**])

lemma *summable_vec_nth* :

assumes *summable f*
shows *summable* ($\lambda x. f\ x\ \$\ i$)
using *assms unfolding summable_def*
by (*blast intro: sums_vec_nth*)

lemma *closed_eucl_atLeastAtMost*[*simp, intro*]:

fixes *a* :: '*a*::*ordered_euclidean_space*
shows *closed* {*a..b*}
by (*simp add: cbx_interval[symmetric] closed_cbx*)

lemma *closed_eucl_atMost*[*simp, intro*]:

fixes *a* :: '*a*::*ordered_euclidean_space*
shows *closed* {*..a*}
by (*simp add: closed_interval_left eucl_le_atMost[symmetric]*)


```

lemma closed_eucl_atLeast[simp, intro]:
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
  shows closed { $a..$ }
  by (simp add: closed_interval_right eucl_le_atLeast[symmetric])

lemma bounded_closed_interval [simp]:
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
  shows bounded { $a .. b$ }
  using bounded_cbox[of a b]
  by (metis interval_cbox)

lemma convex_closed_interval [simp]:
  fixes  $a :: 'a::ordered\_euclidean\_space$ 
  shows convex { $a .. b$ }
  using convex_box[of a b]
  by (metis interval_cbox)

lemma image_smult_interval: $(\lambda x. m *_R (x::ordered\_euclidean\_space))$  ‘ { $a .. b$ } =
  (if { $a .. b$ } = {} then {} else if  $0 \leq m$  then { $m *_R a .. m *_R b$ } else { $m *_R b .. m *_R a$ })
  using image_smult_cbox[of m a b]
  by (simp add: cbox_interval)

lemma [simp]:
  fixes  $a b::'a::ordered\_euclidean\_space$ 
  shows is_interval ic: is_interval { $..a$ }
    and is_interval ci: is_interval { $a..$ }
    and is_interval cc: is_interval { $b..a$ }
  by (force simp: is_interval_def eucl_le[where ‘ $a='a$ ’])+
```

lemma *connected_interval* [*simp*]:
fixes $a b::'a::ordered_euclidean_space$
shows *connected* { $a..b$ }
using *is_interval_cc is_interval_connected* **by blast**

lemma *compact_interval* [*simp*]:
fixes $a b::'a::ordered_euclidean_space$
shows *compact* { $a .. b$ }
by (*metis compact_cbox interval_cbox*)

no_notation
eucl_less (**infix** $<e$ 50)

lemma *One_nonneg*: $0 \leq (\sum \text{Basis}::'a::ordered_euclidean_space)$
by (*auto intro: sum_nonneg*)

lemma

```

fixes a b::'a::ordered_euclidean_space
shows bdd_above_cbox[intro, simp]: bdd_above (cbox a b)
  and bdd_below_cbox[intro, simp]: bdd_below (cbox a b)
  and bdd_above_box[intro, simp]: bdd_above (box a b)
  and bdd_below_box[intro, simp]: bdd_below (box a b)
unfolding atomize_conj
by (metis bdd_above_Icc bdd_above_mono bdd_below_Icc bdd_below_mono
bounded_box
  bounded_subset_cbox_symmetric interval_cbox)

```

```

instantiation vec :: (ordered_euclidean_space, finite) ordered_euclidean_space
begin

```

```

definition inf x y = ( $\chi$  i. inf (x $ i) (y $ i))
definition sup x y = ( $\chi$  i. sup (x $ i) (y $ i))
definition Inf X = ( $\chi$  i. (INF x∈X. x $ i))
definition Sup X = ( $\chi$  i. (SUP x∈X. x $ i))
definition |x| = ( $\chi$  i. |x $ i|)

```

```

instance

```

```

  apply standard
  unfolding euclidean_representation_sum'
  apply (auto simp: less_eq_vec_def inf_vec_def sup_vec_def Inf_vec_def Sup_vec_def
inner_axis
  Basis_vec_def inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left
  inner_Basis_SUP_left eucl_le[where 'a='a] less_le_not_le abs_vec_def abs_inner)
  done

```

```

end

```

```

end

```

6.10 Arcwise-Connected Sets

```

theory Arcwise_Connected

```

```

imports Path_Connected Ordered_Euclidean_Space HOL-Computational_Algebra.Primes
begin

```

```

lemma path_connected_interval [simp]:
  fixes a b::'a::ordered_euclidean_space
  shows path_connected {a..b}
  using is_interval_cc is_interval_path_connected by blast

```

```

lemma segment_to_closest_point:
  fixes S :: 'a :: euclidean_space set
  shows  $\llbracket$ closed S; S  $\neq$   $\{\}$  $\rrbracket \implies$  open_segment a (closest_point S a)  $\cap$  S =  $\{\}$ 
  unfolding disjoint_iff
  by (metis closest_point_le dist_commute dist_in_open_segment not_le)

```

lemma *segment_to_point_exists*:
fixes $S :: 'a :: \text{euclidean_space set}$
assumes $\text{closed } S \ S \neq \{\}$
obtains b **where** $b \in S \ \text{open_segment } a \ b \cap S = \{\}$
by (*metis* *assms* *segment_to_closest_point* *closest_point_exists* *that*)

6.10.1 The Brouwer reduction theorem

theorem *Brouwer_reduction_theorem_gen*:
fixes $S :: 'a :: \text{euclidean_space set}$
assumes $\text{closed } S \ \varphi \ S$
and $\varphi: \bigwedge F. \llbracket \bigwedge n. \text{closed}(F \ n); \bigwedge n. \varphi(F \ n); \bigwedge n. F(\text{Suc } n) \subseteq F \ n \rrbracket \implies \varphi(\bigcap(\text{range } F))$
obtains T **where** $T \subseteq S \ \text{closed } T \ \varphi \ T \ \bigwedge U. \llbracket U \subseteq S; \text{closed } U; \varphi \ U \rrbracket \implies \neg (U \subset T)$
proof –
obtain $B :: \text{nat} \Rightarrow 'a \ \text{set}$
where $\text{inj } B \ \bigwedge n. \text{open}(B \ n)$ **and** $\text{open_cov}: \bigwedge S. \text{open } S \implies \exists K. S = \bigcup(B \ ` \ K)$
by (*metis* *Setcompr_eq_image* *that* *univ_second_countable_sequence*)
define A **where** $A \equiv \text{rec_nat } S \ (\lambda n \ a. \ \text{if } \exists U. U \subseteq a \wedge \text{closed } U \wedge \varphi \ U \wedge U \cap (B \ n) = \{\} \ \text{then } \text{SOME } U. U \subseteq a \wedge \text{closed } U \wedge \varphi \ U \wedge U \cap (B \ n) = \{\} \ \text{else } a)$
have $[\text{simp}]: A \ 0 = S$
by (*simp* *add: A_def*)
have $ASuc: A(\text{Suc } n) = (\text{if } \exists U. U \subseteq A \ n \wedge \text{closed } U \wedge \varphi \ U \wedge U \cap (B \ n) = \{\} \ \text{then } \text{SOME } U. U \subseteq A \ n \wedge \text{closed } U \wedge \varphi \ U \wedge U \cap (B \ n) = \{\} \ \text{else } A \ n)$ **for** n
by (*auto* *simp: A_def*)
have $\text{sub}: \bigwedge n. A(\text{Suc } n) \subseteq A \ n$
by (*auto* *simp: ASuc* *dest!: someI_ex*)
have $\text{subS}: A \ n \subseteq S$ **for** n
by (*induction* n) (*use* *sub* **in** *auto*)
have $\text{clo}: \text{closed } (A \ n) \wedge \varphi \ (A \ n)$ **for** n
by (*induction* n) (*auto* *simp: assms* *ASuc* *dest!: someI_ex*)
show *?thesis*
proof
show $\bigcap(\text{range } A) \subseteq S$
using $\langle \bigwedge n. A \ n \subseteq S \rangle$ **by** *blast*
show $\text{closed } (\bigcap(A \ ` \ \text{UNIV}))$
using *clo* **by** *blast*
show $\varphi (\bigcap(A \ ` \ \text{UNIV}))$
by (*simp* *add: clo* φ *sub*)
show $\neg U \subset \bigcap(A \ ` \ \text{UNIV})$ **if** $U \subseteq S \ \text{closed } U \ \varphi \ U$ **for** U
proof –
have $\exists y. x \notin A \ y$ **if** $x \notin U$ **and** $U \text{sub}: U \subseteq (\bigcap x. A \ x)$ **for** x
proof –

```

obtain  $e$  where  $e > 0$  and  $e: \text{ball } x \ e \subseteq -U$ 
  using  $\langle \text{closed } U \rangle \langle x \notin U \rangle \text{openE [of } -U]$  by blast
moreover obtain  $K$  where  $K: \text{ball } x \ e = \bigcup (B \text{ ' } K)$ 
  using open_cov [of ball x e] by auto
ultimately have  $\bigcup (B \text{ ' } K) \subseteq -U$ 
  by blast
have  $K \neq \{\}$ 
  using  $\langle 0 < e \rangle \langle \text{ball } x \ e = \bigcup (B \text{ ' } K) \rangle$  by auto
then obtain  $n$  where  $n \in K \ x \in B \ n$ 
  by (metis K UN_E  $\langle 0 < e \rangle$  centre_in_ball)
then have  $U \cap B \ n = \{\}$ 
  using  $K \ e$  by auto
show ?thesis
proof (cases  $\exists U \subseteq A \ n. \text{closed } U \wedge \varphi \ U \wedge U \cap B \ n = \{\}$ )
  case True
    then show ?thesis
      apply (rule_tac  $x = \text{Suc } n$  in exI)
      apply (simp add: ASuc)
      apply (erule someI2_ex)
      using  $\langle x \in B \ n \rangle$  by blast
    next
      case False
        then show ?thesis
          by (meson Inf_lower Usub  $\langle U \cap B \ n = \{\} \rangle \langle \varphi \ U \rangle \langle \text{closed } U \rangle$  range_eqI
subset_trans)
          qed
        qed
        with that show ?thesis
          by (meson Inter_iff psubsetE rangeI subsetI)
        qed
      qed
    qed
  qed

```

corollary *Brouwer_reduction_theorem*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes compact S  $\varphi \ S \ S \neq \{\}$ 
and  $\varphi: \bigwedge F. [\bigwedge n. \text{compact } (F \ n); \bigwedge n. F \ n \neq \{\}; \bigwedge n. \varphi (F \ n); \bigwedge n. F (\text{Suc } n) \subseteq F \ n] \implies \varphi (\bigcap (\text{range } F))$ 
obtains  $T$  where  $T \subseteq S$  compact T  $T \neq \{\}$   $\varphi \ T$ 
   $\bigwedge U. [U \subseteq S; \text{closed } U; U \neq \{\}; \varphi \ U] \implies \neg (U \subset T)$ 
proof (rule Brouwer_reduction_theorem_gen [of  $S \ \lambda T. T \neq \{\} \wedge T \subseteq S \wedge \varphi \ T$ ])
fix  $F$ 
assume cloF:  $\bigwedge n. \text{closed } (F \ n)$ 
and  $F: \bigwedge n. F \ n \neq \{\} \wedge F \ n \subseteq S \wedge \varphi (F \ n)$  and  $F\text{sub}: \bigwedge n. F (\text{Suc } n) \subseteq F \ n$ 
show  $\bigcap (F \text{ ' } UNIV) \neq \{\} \wedge \bigcap (F \text{ ' } UNIV) \subseteq S \wedge \varphi (\bigcap (F \text{ ' } UNIV))$ 
proof (intro conjI)
show  $\bigcap (F \text{ ' } UNIV) \neq \{\}$ 
  by (metis F Fsub  $\langle \text{compact } S \rangle$  cloF closed_Int_compact compact_nest)

```

```

inf.orderE lift_Suc_antimono_le)
  show  $\bigcap (F \text{ ' } UNIV) \subseteq S$ 
    using F by blast
  show  $\varphi (\bigcap (F \text{ ' } UNIV))$ 
    by (metis F Fsub  $\varphi$  <compact S> cloF closed_Int_compact inf.orderE)
qed
next
show  $S \neq \{\} \wedge S \subseteq S \wedge \varphi S$ 
  by (simp add: assms)
qed (meson assms compact_imp_closed seq_compact_closed_subset seq_compact_eq_compact)+

```

6.10.2 Arcwise Connections

6.10.3 Density of points with dyadic rational coordinates

proposition *closure_dyadic_rationals:*

$$\text{closure } (\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i :: 'a :: \text{euclidean_space} \in \text{Basis}. (f \ i / 2^k) *_{\mathbb{R}} i \}) = UNIV$$

proof –

have $x \in \text{closure } (\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbb{Z}. \{ \sum i \in \text{Basis}. (f \ i / 2^k) *_{\mathbb{R}} i \})$ **for**
 $x :: 'a$

proof (clarsimp simp: closure_approachable)

fix $e :: \text{real}$

assume $e > 0$

then obtain k **where** $k: (1/2)^k < e / \text{DIM}('a)$

by (meson DIM_positive divide_less_eq_1_pos_of_nat_0_less_iff_one_less_numeral_iff
real_arch_pow_inv semiring_norm(76) zero_less_divide_iff zero_less_numeral)

have $\text{dist } (\sum i \in \text{Basis}. (\text{real_of_int } [2^k * (x \cdot i)] / 2^k) *_{\mathbb{R}} i) \ x =$

$\text{dist } (\sum i \in \text{Basis}. (\text{real_of_int } [2^k * (x \cdot i)] / 2^k) *_{\mathbb{R}} i) \ (\sum i \in \text{Basis}. (x \cdot$
 $i) *_{\mathbb{R}} i)$

by (simp add: euclidean_representation)

also have $\dots = \text{norm } ((\sum i \in \text{Basis}. (\text{real_of_int } [2^k * (x \cdot i)] / 2^k) *_{\mathbb{R}} i -$
 $(x \cdot i) *_{\mathbb{R}} i))$

by (simp add: dist_norm sum_subtractf)

also have $\dots \leq \text{DIM}('a) * ((1/2)^k)$

proof (rule sum_norm_bound, simp add: algebra_simps)

fix $i :: 'a$

assume $i \in \text{Basis}$

then have $\text{norm } ((\text{real_of_int } [x \cdot i * 2^k] / 2^k) *_{\mathbb{R}} i - (x \cdot i) *_{\mathbb{R}} i) =$
 $|\text{real_of_int } [x \cdot i * 2^k] / 2^k - x \cdot i|$

by (simp add: scaleR_left_diff_distrib [symmetric])

also have $\dots \leq (1/2)^k$

by (simp add: divide_simps) linarith

finally show $\text{norm } ((\text{real_of_int } [x \cdot i * 2^k] / 2^k) *_{\mathbb{R}} i - (x \cdot i) *_{\mathbb{R}} i) \leq$
 $(1/2)^k$.

qed

also have $\dots < \text{DIM}('a) * (e / \text{DIM}('a))$

using DIM_positive k linordered_comm_semiring_strict_class.comm_mult_strict_left_mono
of_nat_0_less_iff **by** blast

also have $\dots = e$

by *simp*
 finally have $\text{dist} (\sum i \in \text{Basis}. (\lfloor 2^k * (x \cdot i) \rfloor / 2^k) *_R i) x < e$.
 with *Ints_of_int*
 show $\exists k. \exists f \in \text{Basis} \rightarrow \mathbf{Z}. \text{dist} (\sum b \in \text{Basis}. (f b / 2^k) *_R b) x < e$
 by *fastforce*
 qed
 then show *?thesis* by *auto*
 qed

corollary *closure_rational_coordinates*:

$\text{closure} (\bigcup f \in \text{Basis} \rightarrow \mathbf{Q}. \{ \sum i :: 'a :: \text{euclidean_space} \in \text{Basis}. f i *_R i \}) =$
UNIV

proof –

have $*$: $(\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbf{Z}. \{ \sum i :: 'a \in \text{Basis}. (f i / 2^k) *_R i \})$
 $\subseteq (\bigcup f \in \text{Basis} \rightarrow \mathbf{Q}. \{ \sum i \in \text{Basis}. f i *_R i \})$

proof *clarsimp*

fix k and $f :: 'a \Rightarrow \text{real}$

assume $f: f \in \text{Basis} \rightarrow \mathbf{Z}$

show $\exists x \in \text{Basis} \rightarrow \mathbf{Q}. (\sum i \in \text{Basis}. (f i / 2^k) *_R i) = (\sum i \in \text{Basis}. x i *_R$
 $i)$

apply (*rule_tac* $x = \lambda i. f i / 2^k$ in *bestI*)

using *Ints_subset_Rats* f by *auto*

qed

show *?thesis*

using *closure_dyadic_rationals* *closure_mono* [*OF* $*$] by *blast*

qed

lemma *closure_dyadic_rationals_in_convex_set*:

$\llbracket \text{convex } S; \text{interior } S \neq \{\} \rrbracket$

$\implies \text{closure}(S \cap$

$(\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbf{Z}.$

$\{ \sum i :: 'a :: \text{euclidean_space} \in \text{Basis}. (f i / 2^k) *_R i \})) =$

$\text{closure } S$

by (*simp* add: *closure_dyadic_rationals* *closure_convex_Int_superset*)

lemma *closure_rationals_in_convex_set*:

$\llbracket \text{convex } S; \text{interior } S \neq \{\} \rrbracket$

$\implies \text{closure}(S \cap (\bigcup f \in \text{Basis} \rightarrow \mathbf{Q}. \{ \sum i :: 'a :: \text{euclidean_space} \in \text{Basis}. f i$
 $*_R i \})) =$

$\text{closure } S$

by (*simp* add: *closure_rational_coordinates* *closure_convex_Int_superset*)

Every path between distinct points contains an arc, and hence path connection is equivalent to arcwise connection for distinct points. The proof is based on Whyburn's "Topological Analysis".

lemma *closure_dyadic_rationals_in_convex_set_pos_1*:

fixes $S :: \text{real set}$

assumes *convex* S and *intnz*: $\text{interior } S \neq \{\}$ and *pos*: $\bigwedge x. x \in S \implies 0 \leq x$

shows $\text{closure}(S \cap (\bigcup k m. \{ \text{of_nat } m / 2^k \})) = \text{closure } S$

proof –

have $\exists m. f \ 1 / 2^k = \text{real } m / 2^k$ **if** $(f \ 1) / 2^k \in S$ **for** k **and** $f :: \text{real} \Rightarrow \text{real}$

using *that* **by** (*force simp: Ints_def zero_le_divide_iff power_le_zero_eq dest: pos_zero_le_imp_eq_int*)

then have $S \cap (\bigcup k m. \{\text{real } m / 2^k\}) = S \cap (\bigcup k. \bigcup f \in \text{Basis} \rightarrow \mathbf{Z}. \{\sum i \in \text{Basis}. (f \ i / 2^k) *_{\mathbf{R}} i\})$

by *force*

then show *?thesis*

using *closure_dyadic_rationals_in_convex_set [OF <convex S> intrnz]* **by** *simp* **qed**

definition *dyadics* :: $'a::\text{field_char_0}$ **set** **where** $\text{dyadics} \equiv \bigcup k m. \{\text{of_nat } m / 2^k\}$

lemma *real_in_dyadics* [*simp*]: $\text{real } m \in \text{dyadics}$

by (*simp add: dyadics_def*) (*metis divide_numeral_1 numeral_One power_0*)

lemma *nat_neq_4k1*: $\text{of_nat } m \neq (4 * \text{of_nat } k + 1) / (2 * 2^n :: 'a::\text{field_char_0})$

proof

assume $\text{of_nat } m = (4 * \text{of_nat } k + 1) / (2 * 2^n :: 'a)$

then have $\text{of_nat } (m * (2 * 2^n)) = \text{of_nat } (\text{Suc } (4 * k)) :: 'a$

by (*simp add: field_split_simps*)

then have $m * (2 * 2^n) = \text{Suc } (4 * k)$

using *of_nat_eq_iff* **by** *blast*

then have $\text{odd } (m * (2 * 2^n))$

by *simp*

then show *False*

by *simp*

qed

lemma *nat_neq_4k3*: $\text{of_nat } m \neq (4 * \text{of_nat } k + 3) / (2 * 2^n :: 'a::\text{field_char_0})$

proof

assume $\text{of_nat } m = (4 * \text{of_nat } k + 3) / (2 * 2^n :: 'a)$

then have $\text{of_nat } (m * (2 * 2^n)) = \text{of_nat } (4 * k + 3) :: 'a$

by (*simp add: field_split_simps*)

then have $m * (2 * 2^n) = (4 * k) + 3$

using *of_nat_eq_iff* **by** *blast*

then have $\text{odd } (m * (2 * 2^n))$

by *simp*

then show *False*

by *simp*

qed

lemma *iff_4k*:

assumes $r = \text{real } k$ *odd* k

shows $(4 * \text{real } m + r) / (2 * 2^n) = (4 * \text{real } m' + r) / (2 * 2^{n'}) \iff m = m' \wedge n = n'$

proof –

```

{ assume (4 * real m + r) / (2 * 2^n) = (4 * real m' + r) / (2 * 2^n')
  then have real ((4 * m + k) * (2 * 2^n')) = real ((4 * m' + k) * (2 * 2^n))
    using assms by (auto simp: field_simps)
  then have (4 * m + k) * (2 * 2^n') = (4 * m' + k) * (2 * 2^n)
    using of_nat_eq_iff by blast
  then have (4 * m + k) * (2^n') = (4 * m' + k) * (2^n)
    by linarith
  then obtain 4*m + k = 4*m' + k n=n'
    using prime_power_cancel2 [OF two_is_prime_nat] assms
  by (metis even_mult_iff even_numeral odd_add)
  then have m=m' n=n'
    by auto
}
then show ?thesis by blast

```

qed

lemma *neg_4k1_k43*: $(4 * \text{real } m + 1) / (2 * 2^n) \neq (4 * \text{real } m' + 3) / (2 * 2^{n'})$

proof

```

assume (4 * real m + 1) / (2 * 2^n) = (4 * real m' + 3) / (2 * 2^n')
then have real (Suc (4 * m) * (2 * 2^n')) = real ((4 * m' + 3) * (2 * 2^n))
  by (auto simp: field_simps)
then have Suc (4 * m) * (2 * 2^n') = (4 * m' + 3) * (2 * 2^n)
  using of_nat_eq_iff by blast
then have Suc (4 * m) * (2^n') = (4 * m' + 3) * (2^n)
  by linarith
then have Suc (4 * m) = (4 * m' + 3)
  by (rule prime_power_cancel2 [OF two_is_prime_nat]) auto
then have 1 + 2 * m' = 2 * m
  using ⟨Suc (4 * m) = 4 * m' + 3⟩ by linarith
then show False
  using even_Suc by presburger

```

qed

lemma *dyadic_413_cases*:

```

obtains (of_nat m::'a::field_char_0) / 2^k ∈ Nats
| m' k' where k' < k (of_nat m::'a) / 2^k = of_nat (4*m' + 1) / 2^Suc k'
| m' k' where k' < k (of_nat m::'a) / 2^k = of_nat (4*m' + 3) / 2^Suc k'

```

proof (*cases m > 0*)

```

case False
then have m=0 by simp
with that show ?thesis by auto

```

next

```

case True
obtain k' m' where m': odd m' and k': m = m' * 2^k'
  using prime_power_canonical [OF two_is_prime_nat True] by blast
then obtain q r where q: m' = 4*q + r and r: r < 4
  by (metis not_add_less2 split_div zero_neq_numeral)

```



```

show ?thesis
proof (cases k ≤ k')
  case True
  have (of_nat m:: 'a) / 2^k = of_nat m' * (2 ^ k' / 2^k)
    using k' by (simp add: field_simps)
  also have ... = (of_nat m'::'a) * 2 ^ (k'-k)
    using k' True by (simp add: power_diff)
  also have ... ∈ ℕ
    by (metis Nats_mult of_nat_in_Nats of_nat_numeral of_nat_power)
  finally show ?thesis by (auto simp: that)
next
case False
then obtain kd where kd: Suc kd = k - k'
  using Suc_diff_Suc not_less by blast
have (of_nat m:: 'a) / 2^k = of_nat m' * (2 ^ k' / 2^k)
  using k' by (simp add: field_simps)
also have ... = (of_nat m'::'a) / 2 ^ (k-k')
  using k' False by (simp add: power_diff)
also have ... = ((of_nat r + 4 * of_nat q)::'a) / 2 ^ (k-k')
  using q by force
finally have meq: (of_nat m:: 'a) / 2^k = (of_nat r + 4 * of_nat q) / 2 ^ (k
- k') .
have r ≠ 0 r ≠ 2
  using q m' by presburger+
with r consider r = 1 | r = 3
  by linarith
then show ?thesis
proof cases
  assume r = 1
  with meq kd that(2) [of kd q] show ?thesis
    by simp
next
  assume r = 3
  with meq kd that(3) [of kd q] show ?thesis
    by simp
qed
qed
qed

```

lemma *dyadics_iff*:

```

(dyadics :: 'a::field_char_0 set) =
  Nats ∪ (∪ k m. {of_nat (4*m + 1) / 2^Suc k}) ∪ (∪ k m. {of_nat (4*m +
3) / 2^Suc k})
  (is _ = ?rhs)

```

proof

```

show dyadics ⊆ ?rhs
  unfolding dyadics_def
  apply clarify

```

```

    apply (rule dyadic_413_cases, force+)
  done
next
  have range of_nat  $\subseteq (\bigcup k m. \{(of\_nat m::'a) / 2^k\})$ 
    by clarsimp (metis divide_numeral_1 numeral_One power_0)
  moreover have  $\bigwedge k m. \exists k' m'. ((1::'a) + 4 * of\_nat m) / 2^{Suc k} = of\_nat m' / 2^{k'}$ 
    by (metis (no_types) of_nat_Suc of_nat_mult of_nat_numeral)
  moreover have  $\bigwedge k m. \exists k' m'. (4 * of\_nat m + (3::'a)) / 2^{Suc k} = of\_nat m' / 2^{k'}$ 
    by (metis of_nat_add of_nat_mult of_nat_numeral)
  ultimately show ?rhs  $\subseteq$  dyadics
    by (auto simp: dyadics_def Nats_def)
qed

```

```

function (domintros) dyad_rec :: [nat  $\Rightarrow$  'a, 'a $\Rightarrow$ 'a, 'a $\Rightarrow$ 'a, real]  $\Rightarrow$  'a where
  dyad_rec b l r (real m) = b m
  | dyad_rec b l r ((4 * real m + 1) / 2^{Suc n}) = l (dyad_rec b l r ((2*m + 1) / 2^n))
  | dyad_rec b l r ((4 * real m + 3) / 2^{Suc n}) = r (dyad_rec b l r ((2*m + 1) / 2^n))
  | x  $\notin$  dyadics  $\implies$  dyad_rec b l r x = undefined
  using iff_4k [of _ 1] iff_4k [of _ 3]
    apply (simp_all add: nat_neq_4k1 nat_neq_4k3 neq_4k1_k43 dyadics_iff Nats_def)
  by (fastforce simp: field_simps)+

```

```

lemma dyadics_levels: dyadics =  $(\bigcup K. \bigcup k < K. \bigcup m. \{of\_nat m / 2^k\})$ 
  unfolding dyadics_def by auto

```

```

lemma dyad_rec_level_termination:

```

```

  assumes k < K
  shows dyad_rec_dom(b, l, r, real m / 2^k)
  using assms
  proof (induction K arbitrary: k m)
  case 0
  then show ?case by auto
  next
  case (Suc K)
  then consider k = K | k < K
    using less_antisym by blast
  then show ?case
  proof cases
  assume k = K
  show ?case
  proof (rule dyadic_413_cases [of m k, where 'a=real])
  show real m / 2^k  $\in$   $\mathbf{N} \implies$  dyad_rec_dom (b, l, r, real m / 2^k)
    by (force simp: Nats_def nat_neq_4k1 nat_neq_4k3 intro: dyad_rec.domintros)

```

```

    show ?case if  $k' < k$  and eq:  $\text{real } m / 2^k = \text{real } (4 * m' + 1) / 2^{\text{Suc } k'}$ 
  for  $m' k'$ 
  proof -
    have  $\text{dyad\_rec\_dom } (b, l, r, (4 * \text{real } m' + 1) / 2^{\text{Suc } k'})$ 
    proof (rule  $\text{dyad\_rec.domintros}$ )
      fix  $m n$ 
      assume  $(4 * \text{real } m' + 1) / (2 * 2^k) = (4 * \text{real } m + 1) / (2 * 2^n)$ 
      then have  $m' = m \ k' = n$  using  $\text{iff\_4k } [\text{of\_ } 1]$ 
      by auto
      have  $\text{dyad\_rec\_dom } (b, l, r, \text{real } (2 * m + 1) / 2^k)$ 
      using  $\text{Suc.IH } \langle k = K \rangle \langle k' < k \rangle$  by blast
      then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^n)$ 
      using  $\langle k' = n \rangle$  by (auto simp:  $\text{algebra\_simps}$ )
    next
      fix  $m n$ 
      assume  $(4 * \text{real } m' + 1) / (2 * 2^k) = (4 * \text{real } m + 3) / (2 * 2^n)$ 
      then have False
      by ( $\text{metis } \text{neq\_4k1\_k43}$ )
      then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^n) ..$ 
    qed
    then show ?case by ( $\text{simp add: eq add\_ac}$ )
  qed
  show ?case if  $k' < k$  and eq:  $\text{real } m / 2^k = \text{real } (4 * m' + 3) / 2^{\text{Suc } k'}$ 
  for  $m' k'$ 
  proof -
    have  $\text{dyad\_rec\_dom } (b, l, r, (4 * \text{real } m' + 3) / 2^{\text{Suc } k'})$ 
    proof (rule  $\text{dyad\_rec.domintros}$ )
      fix  $m n$ 
      assume  $(4 * \text{real } m' + 3) / (2 * 2^k) = (4 * \text{real } m + 1) / (2 * 2^n)$ 
      then have False
      by ( $\text{metis } \text{neq\_4k1\_k43}$ )
      then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^n) ..$ 
    next
      fix  $m n$ 
      assume  $(4 * \text{real } m' + 3) / (2 * 2^k) = (4 * \text{real } m + 3) / (2 * 2^n)$ 
      then have  $m' = m \ k' = n$  using  $\text{iff\_4k } [\text{of\_ } 3]$ 
      by auto
      have  $\text{dyad\_rec\_dom } (b, l, r, \text{real } (2 * m + 1) / 2^k)$ 
      using  $\text{Suc.IH } \langle k = K \rangle \langle k' < k \rangle$  by blast
      then show  $\text{dyad\_rec\_dom } (b, l, r, (2 * \text{real } m + 1) / 2^n)$ 
      using  $\langle k' = n \rangle$  by (auto simp:  $\text{algebra\_simps}$ )
    qed
    then show ?case by ( $\text{simp add: eq add\_ac}$ )
  qed
  qed
  next
  assume  $k < K$ 
  then show ?case
  using  $\text{Suc.IH}$  by blast

```

1536

qed
qed

lemma *dyad_rec_termination*: $x \in \text{dyadics} \implies \text{dyad_rec_dom}(b, l, r, x)$
by (*auto simp: dyadics_levels intro: dyad_rec_level_termination*)

lemma *dyad_rec_of_nat* [*simp*]: $\text{dyad_rec } b \ l \ r \ (\text{real } m) = b \ m$
by (*simp add: dyad_rec.psimps dyad_rec_termination*)

lemma *dyad_rec_41* [*simp*]: $\text{dyad_rec } b \ l \ r \ ((4 * \text{real } m + 1) / 2^{\wedge} (\text{Suc } n)) = l$
($\text{dyad_rec } b \ l \ r \ ((2 * m + 1) / 2^{\wedge} n)$)

proof (*rule dyad_rec.psimps*)

show $\text{dyad_rec_dom } (b, l, r, (4 * \text{real } m + 1) / 2^{\wedge} \text{Suc } n)$

by (*metis add.commute dyad_rec_level_termination lessI of_nat_Suc of_nat_mult of_nat_numeral*)

qed

lemma *dyad_rec_43* [*simp*]: $\text{dyad_rec } b \ l \ r \ ((4 * \text{real } m + 3) / 2^{\wedge} (\text{Suc } n)) =$
 $r \ (\text{dyad_rec } b \ l \ r \ ((2 * m + 1) / 2^{\wedge} n))$

proof (*rule dyad_rec.psimps*)

show $\text{dyad_rec_dom } (b, l, r, (4 * \text{real } m + 3) / 2^{\wedge} \text{Suc } n)$

by (*metis dyad_rec_level_termination lessI of_nat_add of_nat_mult of_nat_numeral*)

qed

lemma *dyad_rec_41_times2*:

assumes $n > 0$

shows $\text{dyad_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 1) / 2^{\wedge} \text{Suc } n)) = l \ (\text{dyad_rec } b \ l \ r$
 $(2 * (2 * \text{real } m + 1) / 2^{\wedge} n))$

proof -

obtain n' where $n' : n = \text{Suc } n'$

using *assms not0_implies_Suc* by *blast*

have $\text{dyad_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 1) / 2^{\wedge} \text{Suc } n)) = \text{dyad_rec } b \ l \ r \ ((2 *$
 $(4 * \text{real } m + 1)) / (2 * 2^{\wedge} n))$

by *auto*

also have $\dots = \text{dyad_rec } b \ l \ r \ ((4 * \text{real } m + 1) / 2^{\wedge} n)$

by (*subst mult_divide_mult_cancel_left*) *auto*

also have $\dots = l \ (\text{dyad_rec } b \ l \ r \ ((2 * \text{real } m + 1) / 2^{\wedge} n'))$

by (*simp add: add.commute [of 1] n' del: power_Suc*)

also have $\dots = l \ (\text{dyad_rec } b \ l \ r \ ((2 * (2 * \text{real } m + 1)) / (2 * 2^{\wedge} n'))$

by (*subst mult_divide_mult_cancel_left*) *auto*

also have $\dots = l \ (\text{dyad_rec } b \ l \ r \ (2 * (2 * \text{real } m + 1) / 2^{\wedge} n))$

by (*simp add: add.commute n'*)

finally show *?thesis* .

qed

lemma *dyad_rec_43_times2*:

assumes $n > 0$

shows $\text{dyad_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 3) / 2^{\wedge} \text{Suc } n)) = r \ (\text{dyad_rec } b \ l \ r$

$(2 * (2 * \text{real } m + 1) / 2^{\wedge}n))$

proof –

obtain n' **where** $n': n = \text{Suc } n'$

using *assms not0_implies_Suc* **by** *blast*

have $\text{dyad_rec } b \ l \ r \ (2 * ((4 * \text{real } m + 3) / 2^{\wedge}\text{Suc } n)) = \text{dyad_rec } b \ l \ r \ ((2 * (4 * \text{real } m + 3)) / (2 * 2^{\wedge}n))$

by *auto*

also have $\dots = \text{dyad_rec } b \ l \ r \ ((4 * \text{real } m + 3) / 2^{\wedge}n)$

by (*subst mult_divide_mult_cancel_left*) *auto*

also have $\dots = r \ (\text{dyad_rec } b \ l \ r \ ((2 * \text{real } m + 1) / 2^{\wedge}n'))$

by (*simp add: n' del: power_Suc*)

also have $\dots = r \ (\text{dyad_rec } b \ l \ r \ ((2 * (2 * \text{real } m + 1)) / (2 * 2^{\wedge}n')))$

by (*subst mult_divide_mult_cancel_left*) *auto*

also have $\dots = r \ (\text{dyad_rec } b \ l \ r \ (2 * (2 * \text{real } m + 1) / 2^{\wedge}n))$

by (*simp add: n'*)

finally show *?thesis* .

qed

definition *dyad_rec2*

where $\text{dyad_rec2 } u \ v \ lc \ rc \ x =$

$\text{dyad_rec } (\lambda z. (u, v)) \ (\lambda(a, b). (a, lc \ a \ b \ (\text{midpoint } a \ b))) \ (\lambda(a, b). (rc \ a \ b \ (\text{midpoint } a \ b), b)) \ (2 * x)$

abbreviation *leftrec* **where** $\text{leftrec } u \ v \ lc \ rc \ x \equiv \text{fst } (\text{dyad_rec2 } u \ v \ lc \ rc \ x)$

abbreviation *rightrec* **where** $\text{rightrec } u \ v \ lc \ rc \ x \equiv \text{snd } (\text{dyad_rec2 } u \ v \ lc \ rc \ x)$

lemma *leftrec_base*: $\text{leftrec } u \ v \ lc \ rc \ (\text{real } m / 2) = u$

by (*simp add: dyad_rec2_def*)

lemma *leftrec_41*: $n > 0 \implies \text{leftrec } u \ v \ lc \ rc \ ((4 * \text{real } m + 1) / 2^{\wedge}(\text{Suc } n)) = \text{leftrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n)$

unfolding *dyad_rec2_def dyad_rec_41_times2*

by (*simp add: case_prod_beta*)

lemma *leftrec_43*: $n > 0 \implies$

$\text{leftrec } u \ v \ lc \ rc \ ((4 * \text{real } m + 3) / 2^{\wedge}(\text{Suc } n)) =$

$rc \ (\text{leftrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n)) \ (\text{rightrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n))$

$(\text{midpoint } (\text{leftrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n)) \ (\text{rightrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n)))$

unfolding *dyad_rec2_def dyad_rec_43_times2*

by (*simp add: case_prod_beta*)

lemma *rightrec_base*: $\text{rightrec } u \ v \ lc \ rc \ (\text{real } m / 2) = v$

by (*simp add: dyad_rec2_def*)

lemma *rightrec_41*: $n > 0 \implies$

$\text{rightrec } u \ v \ lc \ rc \ ((4 * \text{real } m + 1) / 2^{\wedge}(\text{Suc } n)) =$

$lc \ (\text{leftrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n)) \ (\text{rightrec } u \ v \ lc \ rc \ ((2 * \text{real } m + 1) / 2^{\wedge}n))$

1538

* $real\ m + 1) / 2^{\wedge}n$)
 (midpoint (leftrec u v lc rc ((2 * real m + 1) / 2[^]n)) (rightrec u v lc
 rc ((2 * real m + 1) / 2[^]n)))
unfolding dyad_rec2_def dyad_rec_41_times2
by (simp add: case_prod_beta)

lemma rightrec_43: $n > 0 \implies rightrec\ u\ v\ lc\ rc\ ((4 * real\ m + 3) / 2^{\wedge}(Suc\ n)) = rightrec\ u\ v\ lc\ rc\ ((2 * real\ m + 1) / 2^{\wedge}n)$
unfolding dyad_rec2_def dyad_rec_43_times2
by (simp add: case_prod_beta)

lemma dyadics_in_open_unit_interval:
 $\{0 <..<1\} \cap (\bigcup k\ m.\ \{real\ m / 2^{\wedge}k\}) = (\bigcup k.\ \bigcup m \in \{0 <..<2^{\wedge}k\}.\ \{real\ m / 2^{\wedge}k\})$
by (auto simp: field_split_simps)

lemma padic_rational_approximation_straddle:
assumes $\varepsilon > 0\ p > 1$
obtains $n\ q\ r$
where $of_int\ q / p^{\wedge}n < x\ x < of_int\ r / p^{\wedge}n\ |q / p^{\wedge}n - r / p^{\wedge}n| < \varepsilon$
proof -
obtain n **where** $n: 2 / \varepsilon < p^{\wedge}n$
using $\langle p > 1 \rangle\ real_arch_pow$ **by** blast
define q **where** $q \equiv \lfloor p^{\wedge}n * x \rfloor - 1$
show thesis
proof
show $q / p^{\wedge}n < x\ x < real_of_int\ (q+2) / p^{\wedge}n$
using *assms* **by** (simp_all add: q_def divide_simps floor_less_cancel
 mult.commute)
show $|q / p^{\wedge}n - real_of_int\ (q+2) / p^{\wedge}n| < \varepsilon$
using *assms* n **by** (simp add: q_def divide_simps mult.commute)
qed
qed

lemma padic_rational_approximation_straddle_pos:
assumes $\varepsilon > 0\ p > 1\ x > 0$
obtains $n\ q\ r$
where $of_nat\ q / p^{\wedge}n < x\ x < of_nat\ r / p^{\wedge}n\ |q / p^{\wedge}n - r / p^{\wedge}n| < \varepsilon$
proof -
obtain $n\ q\ r$
where $*$: $of_int\ q / p^{\wedge}n < x\ x < of_int\ r / p^{\wedge}n\ |q / p^{\wedge}n - r / p^{\wedge}n| < \varepsilon$
using padic_rational_approximation_straddle *assms* **by** metis
then **have** $r \geq 0$
using *assms* **by** (smt (verit, best) divide_nonpos_pos of_int_0_le_iff_zero_less_power)
show thesis
proof
show $real\ (max\ 0\ (nat\ q)) / p^{\wedge}n < x$
using $*$ **by** (metis *assms*(3) div_0 max_nat.left_neutral nat_eq_iff2 of_nat_0

```

of_nat_nat)
  show  $x < \text{real } (\text{nat } r) / p^{\wedge} n$ 
    using  $\langle r \geq 0 \rangle$  * by force
  show  $|\text{real } (\text{max } 0 (\text{nat } q)) / p^{\wedge} n - \text{real } (\text{nat } r) / p^{\wedge} n| < \varepsilon$ 
    using * assms by (simp add: divide_simps)
qed
qed

lemma padic_rational_approximation_straddle_pos_le:
  assumes  $\varepsilon > 0$   $p > 1$   $x \geq 0$ 
  obtains  $n$   $q$   $r$ 
    where  $\text{of\_nat } q / p^{\wedge} n \leq x < \text{of\_nat } r / p^{\wedge} n$   $|q / p^{\wedge} n - r / p^{\wedge} n| < \varepsilon$ 
proof -
  obtain  $n$   $q$   $r$ 
    where *:  $\text{of\_int } q / p^{\wedge} n < x < \text{of\_int } r / p^{\wedge} n$   $|q / p^{\wedge} n - r / p^{\wedge} n| < \varepsilon$ 
    using padic_rational_approximation_straddle assms by metis
  then have  $r \geq 0$ 
    using assms by (smt (verit, best) divide_nonpos_pos of_int_0_le_iff_zero_less_power)
  show thesis
proof
  show  $\text{real } (\text{max } 0 (\text{nat } q)) / p^{\wedge} n \leq x$ 
    using * assms(3) nle_le by fastforce
  show  $x < \text{real } (\text{nat } r) / p^{\wedge} n$ 
    using  $\langle r \geq 0 \rangle$  * by force
  show  $|\text{real } (\text{max } 0 (\text{nat } q)) / p^{\wedge} n - \text{real } (\text{nat } r) / p^{\wedge} n| < \varepsilon$ 
    using * assms by (simp add: divide_simps)
qed
qed

```

Definition by recursion on dyadic rationals in [0,1]

```

lemma recursion_on_dyadic_fractions:
  assumes base:  $R$   $a$   $b$ 
  and step:  $\bigwedge x y. R$   $x$   $y \implies \exists z. R$   $x$   $z \wedge R$   $z$   $y$  and trans:  $\bigwedge x y z. \llbracket R$   $x$   $y; R$   $y$ 
 $z \rrbracket \implies R$   $x$   $z$ 
  shows  $\exists f :: \text{real} \implies 'a. f$   $0 = a \wedge f$   $1 = b \wedge$ 
    ( $\forall x \in \text{dyadics} \cap \{0..1\}. \forall y \in \text{dyadics} \cap \{0..1\}. x < y \implies R$   $(f$   $x)$   $(f$ 
 $y)$ )
proof -
  obtain mid where mid:  $R$   $x$   $y \implies R$   $x$   $(\text{mid } x$   $y)$   $R$   $x$   $y \implies R$   $(\text{mid } x$   $y)$   $y$  for
 $x$   $y$ 
    using step by metis
  define  $g$  where  $g \equiv \text{rec\_nat } (\lambda k. \text{if } k = 0 \text{ then } a \text{ else } b)$   $(\lambda n$   $r$   $k. \text{if even } k \text{ then}$ 
 $r$   $(k \text{ div } 2)$   $\text{else mid } (r$   $((k - 1) \text{ div } 2))$   $(r$   $((\text{Suc } k) \text{ div } 2))$ )
  have  $g$   $0$  [simp]:  $g$   $0 = (\lambda k. \text{if } k = 0 \text{ then } a \text{ else } b)$ 
    by (simp add: g_def)
  have  $g$   $\text{Suc}$  [simp]:  $\bigwedge n. g$   $(\text{Suc } n) = (\lambda k. \text{if even } k \text{ then } g$   $n$   $(k \text{ div } 2)$   $\text{else mid } (g$ 
 $n$   $((k - 1) \text{ div } 2))$   $(g$   $n$   $((\text{Suc } k) \text{ div } 2))$ )
    by (auto simp: g_def)

```

```

have g_eq_g: 2 ^ d * k = k'  $\implies$  g n k = g (n + d) k' for n d k k'
  by (induction d arbitrary: k k') auto
have g n k = g n' k' if real k / 2 ^ n = real k' / 2 ^ n' n'  $\leq$  n for k n k' n'
proof -
  have real k = real k' * 2 ^ (n-n')
    using that by (simp add: power_diff divide_simps)
  then have k = k' * 2 ^ (n-n')
    using of_nat_eq_iff by fastforce
  with g_eq_g show ?thesis
    by (metis le_add_diff_inverse mult.commute that(2))
qed
then have g_eq_g: g n k = g n' k' if real k / 2 ^ n = real k' / 2 ^ n' for k n
k' n'
  by (metis nat_le_linear that)
then obtain f where ( $\lambda(k,n). g n k$ ) = f  $\circ$  ( $\lambda(k,n). k / 2 ^ n$ )
  using function_factors_left by (smt (verit, del_insts) case_prod_beta')
then have f_eq_g:  $\bigwedge k n. f(\text{real } k / 2 ^ n) = g n k$ 
  by (simp add: fun_eq_iff)
show ?thesis
proof (intro exI conjI strip)
  show f 0 = a
    by (metis f_eq_g g0 div_0 of_nat_0)
  show f 1 = b
    by (metis f_eq_g g0 div_by_1 of_nat_1_eq_iff power_0 zero_neq_one)
  show R (f x) (f y)
    if x:  $x \in \text{dyadics} \cap \{0..1\}$  and y:  $y \in \text{dyadics} \cap \{0..1\}$  and  $x < y$  for x y
  proof -
    obtain n1 k1 where xeq:  $x = \text{real } k1 / 2 ^{n1}$  k1  $\leq 2 ^{n1}$ 
      using x by (auto simp: dyadics_def)
    obtain n2 k2 where yeq:  $y = \text{real } k2 / 2 ^{n2}$  k2  $\leq 2 ^{n2}$ 
      using y by (auto simp: dyadics_def)
    have xcommon:  $x = \text{real}(2 ^{n2} * k1) / 2 ^{(n1+n2)}$ 
      using xeq by (simp add: power_add)
    have ycommon:  $y = \text{real}(2 ^{n1} * k2) / 2 ^{(n1+n2)}$ 
      using yeq by (simp add: power_add)
    have *: R (g n j) (g n k) if j < k k  $\leq 2 ^ n$  for n j k
      using that
    proof (induction n arbitrary: j k)
      case 0
      then show ?case
        by (simp add: base)
      next
      case (Suc n)
      show ?case
      proof (cases even j)
        case True
        then obtain a where [simp]:  $j = 2 * a$ 
          by blast
        show ?thesis

```



```

proof (cases even k)
  case True
    with Suc show ?thesis
      by (auto elim!: evenE)
  next
    case False
      then obtain b where [simp]: k = Suc (2*b)
        using oddE by fastforce
      show ?thesis
        using Suc
        apply simp
        by (smt (verit, ccfv_SIG) less_Suc_eq linorder_not_le local.trans
mid(1) nat_mult_less_cancel1 pos2)
      qed
  next
    case False
      then obtain a where [simp]: j = Suc (2*a)
        using oddE by fastforce
      show ?thesis
        proof (cases even k)
          case True
            then obtain b where [simp]: k = 2*b
              by blast
            show ?thesis
              using Suc
              apply simp
            by (smt (verit, ccfv_SIG) Suc_leI Suc_lessD le_trans lessI linorder_neqE_nat
linorder_not_le local.trans mid(2) nat_mult_less_cancel1 pos2)
          next
            case False
              then obtain b where [simp]: k = Suc (2*b)
                using oddE by fastforce
              show ?thesis
                using Suc
                apply simp
              by (smt (verit) Suc_leI le_trans lessI less_or_eq_imp_le linorder_neqE_nat
linorder_not_le local.trans mid(1) mid(2) nat_mult_less_cancel1 pos2)
            qed
          qed
        show ?thesis
          unfolding xcommon ycommon f_eq_g
        proof (rule *)
          show  $2^{n2} * k1 < 2^{n1} * k2$ 
            using of_nat_less_iff ⟨x < y⟩ by (fastforce simp: xeq yeq field_simps)
          show  $2^{n1} * k2 \leq 2^{n1 + n2}$ 
            by (simp add: power_add yeq)
        qed
      qed

```

1542

qed
qed

lemma *dyadics_add*:
 assumes $x \in \text{dyadics}$ $y \in \text{dyadics}$
 shows $x+y \in \text{dyadics}$
proof –
 obtain $i\ j\ m\ n$ **where** $x = \text{of_nat } i / 2^m$ **and** $y = \text{of_nat } j / 2^n$
 using *assms* **by** (*auto simp: dyadics_def*)
 have $x_{\text{common}}: x = \text{of_nat}(2^n * i) / 2^{(m+n)}$
 using x **by** (*simp add: power_add*)
 moreover
 have $y_{\text{common}}: y = \text{of_nat}(2^m * j) / 2^{(m+n)}$
 using y **by** (*simp add: power_add*)
 ultimately have $x+y = (\text{of_nat}(2^n * i + 2^m * j)) / 2^{(m+n)}$
 by (*simp add: field_simps*)
 then show *?thesis*
 unfolding *dyadics_def* **by** *blast*
qed

lemma *dyadics_diff*:
 fixes $x :: 'a::\text{linordered_field}$
 assumes $x \in \text{dyadics}$ $y \in \text{dyadics}$ $y \leq x$
 shows $x-y \in \text{dyadics}$
proof –
 obtain $i\ j\ m\ n$ **where** $x = \text{of_nat } i / 2^m$ **and** $y = \text{of_nat } j / 2^n$
 using *assms* **by** (*auto simp: dyadics_def*)
 have $j_le_i: j * 2^m \leq i * 2^n$
 using *of_nat_le_iff* $\langle y \leq x \rangle$ **unfolding** $x\ y$ **by** (*fastforce simp add: divide_simps*)
 have $x_{\text{common}}: x = \text{of_nat}(2^n * i) / 2^{(m+n)}$
 using x **by** (*simp add: power_add*)
 moreover
 have $y_{\text{common}}: y = \text{of_nat}(2^m * j) / 2^{(m+n)}$
 using y **by** (*simp add: power_add*)
 ultimately have $x-y = (\text{of_nat}(2^n * i - 2^m * j)) / 2^{(m+n)}$
 by (*simp add: xcommon ycommon field_simps j_le_i of_nat_diff*)
 then show *?thesis*
 unfolding *dyadics_def* **by** *blast*
qed

theorem *homeomorphic_monotone_image_interval*:
 fixes $f :: \text{real} \Rightarrow 'a::\{\text{real_normed_vector, complete_space}\}$
 assumes *cont_f*: *continuous_on* $\{0..1\}$ f
 and *conn*: $\bigwedge y. \text{connected } (\{0..1\} \cap f^{-1}\{y\})$
 and *f_1not0*: $f\ 1 \neq f\ 0$
 shows $(f^{-1}\{0..1\})$ *homeomorphic* $\{0..1::\text{real}\}$

```

proof -
  have  $\exists c d. a \leq c \wedge c \leq m \wedge m \leq d \wedge d \leq b \wedge$ 
     $(\forall x \in \{c..d\}. f x = f m) \wedge$ 
     $(\forall x \in \{a..<c\}. (f x \neq f m)) \wedge$ 
     $(\forall x \in \{d<..b\}. (f x \neq f m)) \wedge$ 
     $(\forall x \in \{a..<c\}. \forall y \in \{d<..b\}. f x \neq f y)$ 
  if  $m \in \{a..b\}$  and  $ab01: \{a..b\} \subseteq \{0..1\}$  for  $a b m$ 
  proof -
    have  $comp: compact (f -' \{f m\} \cap \{0..1\})$ 
    by (simp add: compact_eq_bounded_closed bounded_Int closed_vimage_Int cont_f)
    obtain  $c0 d0$  where  $cd0: \{0..1\} \cap f -' \{f m\} = \{c0..d0\}$ 
    using connected_compact_interval_1 [of  $\{0..1\} \cap f -' \{f m\}$ ] conn comp
    by (metis Int_commute)
    with that have  $m \in cbox c0 d0$ 
    by auto
    obtain  $c d$  where  $cd: \{a..b\} \cap f -' \{f m\} = \{c..d\}$ 
    using ab01 cd0
    by (rule_tac c=max a c0 and d=min b d0 in that) auto
    then have  $cdab: \{c..d\} \subseteq \{a..b\}$ 
    by blast
    show ?thesis
    proof (intro exI conjI ballI)
      show  $a \leq c d \leq b$ 
      using cdab cd m by auto
      show  $c \leq m m \leq d$ 
      using cd m by auto
      show  $\bigwedge x. x \in \{c..d\} \implies f x = f m$ 
      using cd by blast
      show  $f x \neq f m$  if  $x \in \{a..<c\}$  for  $x$ 
      using that m cd [THEN equalityD1, THEN subsetD] <c ≤ m> by force
      show  $f x \neq f m$  if  $x \in \{d<..b\}$  for  $x$ 
      using that m cd [THEN equalityD1, THEN subsetD, of x] <m ≤ d> by force
      show  $f x \neq f y$  if  $x \in \{a..<c\}$   $y \in \{d<..b\}$  for  $x y$ 
      proof (cases f x = f m  $\vee$  f y = f m)
        case True
        then show ?thesis
        using  $\langle \bigwedge x. x \in \{a..<c\} \implies f x \neq f m \rangle$  that by auto
      next
      case False
      have False if  $f x = f y$ 
      proof -
        have  $x \leq m m \leq y$ 
        using  $\langle c \leq m \rangle \langle x \in \{a..<c\} \rangle \langle m \leq d \rangle \langle y \in \{d<..b\} \rangle$  by auto
        then have  $x \in (\{0..1\} \cap f -' \{f y\})$   $y \in (\{0..1\} \cap f -' \{f y\})$ 
        using  $\langle x \in \{a..<c\} \rangle \langle y \in \{d<..b\} \rangle$  ab01 by (auto simp: that)
        then have  $m \in (\{0..1\} \cap f -' \{f y\})$ 
        by (meson <m ≤ y> <x ≤ m> is_interval_connected_1 conn [of f y]
is_interval_1)

```

```

    with False show False by auto
  qed
  then show ?thesis by auto
  qed
  qed
  qed
  then obtain leftcut rightcut where LR:
     $\bigwedge a b m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies$ 
     $(a \leq \text{leftcut } a b m \wedge \text{leftcut } a b m \leq m \wedge m \leq \text{rightcut } a b m \wedge \text{rightcut}$ 
 $a b m \leq b \wedge$ 
     $(\forall x \in \{\text{leftcut } a b m.. \text{rightcut } a b m\}. f x = f m) \wedge$ 
     $(\forall x \in \{a.. < \text{leftcut } a b m\}. f x \neq f m) \wedge$ 
     $(\forall x \in \{\text{rightcut } a b m <.. b\}. f x \neq f m) \wedge$ 
     $(\forall x \in \{a.. < \text{leftcut } a b m\}. \forall y \in \{\text{rightcut } a b m <.. b\}. f x \neq f y))$ 
  apply atomize
  apply (clarsimp simp only: imp_conjL [symmetric] choice_iff choice_iff')
  apply (rule that, blast)
  done
  then have left_right:  $\bigwedge a b m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies a \leq \text{leftcut } a$ 
 $b m \wedge \text{rightcut } a b m \leq b$ 
  and left_right_m:  $\bigwedge a b m. \llbracket m \in \{a..b\}; \{a..b\} \subseteq \{0..1\} \rrbracket \implies \text{leftcut } a b m$ 
 $\leq m \wedge m \leq \text{rightcut } a b m$ 
  by auto
  have left_neg:  $\llbracket a \leq x; x < \text{leftcut } a b m; a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies$ 
 $f x \neq f m$ 
  and right_neg:  $\llbracket \text{rightcut } a b m < x; x \leq b; a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket$ 
 $\implies f x \neq f m$ 
  and left_right_neg:  $\llbracket a \leq x; x < \text{leftcut } a b m; \text{rightcut } a b m < y; y \leq b; a \leq$ 
 $m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies f x \neq f m$ 
  and feqm:  $\llbracket \text{leftcut } a b m \leq x; x \leq \text{rightcut } a b m; a \leq m; m \leq b; \{a..b\} \subseteq$ 
 $\{0..1\} \rrbracket$ 
 $\implies f x = f m$  for  $a b m x y$ 
  by (meson atLeastAtMost_iff greaterThanAtMost_iff atLeastLessThan_iff LR)+
  have f_eqI:  $\bigwedge a b m x y. \llbracket \text{leftcut } a b m \leq x; x \leq \text{rightcut } a b m; \text{leftcut } a b m \leq$ 
 $y; y \leq \text{rightcut } a b m;$ 
 $a \leq m; m \leq b; \{a..b\} \subseteq \{0..1\} \rrbracket \implies f x = f y$ 
  by (metis feqm)
  define u where u  $\equiv \text{rightcut } 0 1 0$ 
  have lc[simp]:  $\text{leftcut } 0 1 0 = 0$  and u01:  $0 \leq u u \leq 1$ 
  using LR [of 0 0 1] by (auto simp: u_def)
  have f0u:  $\bigwedge x. x \in \{0..u\} \implies f x = f 0$ 
  using LR [of 0 0 1] unfolding u_def [symmetric]
  by (metis <leftcut 0 1 0 = 0> atLeastAtMost_iff order_refl zero_le_one)
  have fu1:  $\bigwedge x. x \in \{u<..1\} \implies f x \neq f 0$ 
  using LR [of 0 0 1] unfolding u_def [symmetric] by fastforce
  define v where v  $\equiv \text{leftcut } u 1 1$ 
  have rc[simp]:  $\text{rightcut } u 1 1 = 1$  and v01:  $u \leq v v \leq 1$ 
  using LR [of 1 u 1] u01 by (auto simp: v_def)
  have fuv:  $\bigwedge x. x \in \{u..<v\} \implies f x \neq f 1$ 

```

```

    using LR [of 1 u 1] u01 v_def by fastforce
  have f0v:  $\bigwedge x. x \in \{0..<v\} \implies f x \neq f 1$ 
    by (metis f_1not0 atLeastAtMost_iff atLeastLessThan_iff f0u fuv linear)
  have fv1:  $\bigwedge x. x \in \{v..1\} \implies f x = f 1$ 
    using LR [of 1 u 1] u01 v_def by (metis atLeastAtMost_iff atLeastatMost_subset_iff
order_refl rc)
  define a where a  $\equiv$  leftrec u v leftcut rightcut
  define b where b  $\equiv$  rightrec u v leftcut rightcut
  define c where c  $\equiv$   $\lambda x. \text{midpoint } (a x) (b x)$ 
  have a_real [simp]: a (real j) = u for j
    using a_def leftrec_base
  by (metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral)
  have b_real [simp]: b (real j) = v for j
    using b_def rightrec_base
  by (metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral)
  have a41: a ((4 * real m + 1) / 2^Suc n) = a ((2 * real m + 1) / 2^n) if n >
0 for m n
    using that a_def leftrec_41 by blast
  have b41: b ((4 * real m + 1) / 2^Suc n) =
    leftcut (a ((2 * real m + 1) / 2^n))
      (b ((2 * real m + 1) / 2^n))
      (c ((2 * real m + 1) / 2^n)) if n > 0 for m n
    using that a_def b_def c_def rightrec_41 by blast
  have a43: a ((4 * real m + 3) / 2^Suc n) =
    rightcut (a ((2 * real m + 1) / 2^n))
      (b ((2 * real m + 1) / 2^n))
      (c ((2 * real m + 1) / 2^n)) if n > 0 for m n
    using that a_def b_def c_def leftrec_43 by blast
  have b43: b ((4 * real m + 3) / 2^Suc n) = b ((2 * real m + 1) / 2^n) if n >
0 for m n
    using that b_def rightrec_43 by blast
  have uabv:  $u \leq a (\text{real } m / 2^n) \wedge a (\text{real } m / 2^n) \leq b (\text{real } m / 2^n) \wedge$ 
 $b (\text{real } m / 2^n) \leq v$  for m n
  proof (induction n arbitrary: m)
    case 0
      then show ?case by (simp add: v01)
    next
      case (Suc n p)
        show ?case
        proof (cases even p)
          case True
            then obtain m where p = 2*m by (metis evenE)
            then show ?thesis
              by (simp add: Suc.IH)
          next
            case False
              then obtain m where m: p = 2*m + 1 by (metis oddE)
              show ?thesis
              proof (cases n)

```

```

case 0
then show ?thesis
  by (simp add: a_def b_def leftrec_base rightrec_base v01)
next
case (Suc n')
then have n > 0 by simp
have a_le_c: a (real m / 2^n) ≤ c (real m / 2^n) for m
  unfolding c_def by (metis Suc.IH ge_midpoint_1)
have c_le_b: c (real m / 2^n) ≤ b (real m / 2^n) for m
  unfolding c_def by (metis Suc.IH le_midpoint_1)
have c_ge_u: c (real m / 2^n) ≥ u for m
  using Suc.IH a_le_c order_trans by blast
have c_le_v: c (real m / 2^n) ≤ v for m
  using Suc.IH c_le_b order_trans by blast
have a_ge_0: 0 ≤ a (real m / 2^n) for m
  using Suc.IH order_trans u01(1) by blast
have b_le_1: b (real m / 2^n) ≤ 1 for m
  using Suc.IH order_trans v01(2) by blast
have left_le: leftcut (a ((real m) / 2^n)) (b ((real m) / 2^n)) (c ((real m)
/ 2^n)) ≤ c ((real m) / 2^n) for m
  by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
have right_ge: rightcut (a ((real m) / 2^n)) (b ((real m) / 2^n)) (c ((real
m) / 2^n)) ≥ c ((real m) / 2^n) for m
  by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
show ?thesis
proof (cases even m)
case True
then obtain r where r: m = 2*r by (metis evenE)
show ?thesis
  using order_trans [OF left_le c_le_v, of 1+2*r] Suc.IH [of m+1]
  using a_le_c [of m+1] c_le_b [of m+1] a_ge_0 [of m+1] b_le_1 [of
m+1] left_right ⟨n > 0⟩
  by (simp_all add: r m add.commute [of 1] a41 b41 del: power_Suc)
next
case False
then obtain r where r: m = 2*r + 1 by (metis oddE)
show ?thesis
  using order_trans [OF c_ge_u right_ge, of 1+2*r] Suc.IH [of m]
  using a_le_c [of m] c_le_b [of m] a_ge_0 [of m] b_le_1 [of m]
left_right ⟨n > 0⟩
  apply (simp_all add: r m add.commute [of 3] a43 b43 del: power_Suc)
  by (simp add: add.commute)
qed
qed
qed
have a_ge_0 [simp]: 0 ≤ a(m / 2^n) and b_le_1 [simp]: b(m / 2^n) ≤ 1 for
m::nat and n
  using uabv order_trans u01 v01 by blast+

```

```

then have  $b\_ge\_0$  [simp]:  $0 \leq b(m / 2^n)$  and  $a\_le\_1$  [simp]:  $a(m / 2^n) \leq 1$ 
for  $m::nat$  and  $n$ 
  using  $uabv$   $order\_trans$  by  $blast+$ 
  have  $alec$  [simp]:  $a(m / 2^n) \leq c(m / 2^n)$  and  $cleb$  [simp]:  $c(m / 2^n) \leq b(m / 2^n)$ 
for  $m::nat$  and  $n$ 
  by ( $auto$   $simp$ :  $c\_def$   $ge\_midpoint\_1$   $le\_midpoint\_1$   $uabv$ )
  have  $c\_ge\_0$  [simp]:  $0 \leq c(m / 2^n)$  and  $c\_le\_1$  [simp]:  $c(m / 2^n) \leq 1$ 
for  $m::nat$  and  $n$ 
  using  $a\_ge\_0$   $alec$   $b\_le\_1$   $cleb$   $order\_trans$  by  $blast+$ 
  have  $\llbracket d = m - n; odd\ j; |real\ i / 2^m - real\ j / 2^n| < 1 / 2^n \rrbracket$ 
     $\implies (a(j / 2^n)) \leq (c(i / 2^m)) \wedge (c(i / 2^m)) \leq (b(j / 2^n))$  for  $d\ i\ j\ m$ 
 $n$ 
  proof ( $induction\ d$   $arbitrary: j\ n$   $rule: less\_induct$ )
    case ( $less\ d\ j\ n$ )
      show ?case
      proof ( $cases\ m \leq n$ )
        case  $True$ 
          have  $|2^n| * |real\ i / 2^m - real\ j / 2^n| = 0$ 
          proof ( $rule\ Ints\_nonzero\_abs\_less1$ )
            have  $(real\ i * 2^n - real\ j * 2^m) / 2^m = (real\ i * 2^n) / 2^m - (real\ j * 2^m) / 2^m$ 
            using  $diff\_divide\_distrib$  by  $blast$ 
            also have  $\dots = (real\ i * 2^{(n-m)}) - (real\ j)$ 
            using  $True$  by ( $auto\ simp: power\_diff\_field\_simps$ )
            also have  $\dots \in \mathbb{Z}$ 
            by  $simp$ 
            finally have  $(real\ i * 2^n - real\ j * 2^m) / 2^m \in \mathbb{Z}$  .
          with  $True\ Ints\_abs$  show  $|2^n| * |real\ i / 2^m - real\ j / 2^n| \in \mathbb{Z}$ 
          by ( $fastforce\ simp: field\_split\_simps$ )
          show  $\llbracket |2^n| * |real\ i / 2^m - real\ j / 2^n| < 1 \rrbracket$ 
          using  $less.premis$  by ( $auto\ simp: field\_split\_simps$ )
        qed
        then have  $real\ i / 2^m = real\ j / 2^n$ 
        by  $auto$ 
        then show ?thesis
        by  $auto$ 
      next
        case  $False$ 
        then have  $n < m$  by  $auto$ 
        obtain  $k$  where  $k: j = Suc\ (2*k)$ 
        using  $\langle odd\ j \rangle\ oddE$  by  $fastforce$ 
        show ?thesis
        proof ( $cases\ n > 0$ )
          case  $False$ 
            then have  $a\ (real\ j / 2^n) = u$ 
            by  $simp$ 
            also have  $\dots \leq c\ (real\ i / 2^m)$ 
            using  $alec\ uabv$  by ( $blast\ intro: order\_trans$ )
            finally have  $ac: a\ (real\ j / 2^n) \leq c\ (real\ i / 2^m)$  .
          qed
        qed
      qed

```

```

have c (real i / 2m) ≤ v
  using cleb uabv by (blast intro: order_trans)
also have ... = b (real j / 2n)
  using False by simp
finally show ?thesis
  by (auto simp: ac)
next
case True show ?thesis
proof (cases i / 2m j / 2n rule: linorder_cases)
case less
  moreover have real (4 * k + 1) / 2Suc n + 1 / (2Suc n) = real j
/ 2n
  using k by (force simp: field_split_simps)
  moreover have |real i / 2m - j / 2n| < 2 / (2Suc n)
  using less.prem1 by simp
  ultimately have closer: |real i / 2m - real (4 * k + 1) / 2Suc n|
< 1 / (2Suc n)
  using less.prem1 by linarith
  have a (real (4 * k + 1) / 2Suc n) ≤ c (i / 2m) ∧
c (real i / 2m) ≤ b (real (4 * k + 1) / 2Suc n)
  proof (rule less.IH [OF _ refl])
  show m - Suc n < d
    using ⟨n < m⟩ diff_less_mono2 less.prem1(1) lessI by presburger
  show |real i / 2m - real (4 * k + 1) / 2Suc n| < 1 / 2Suc n
    using closer ⟨n < m⟩ ⟨d = m - n⟩ by (auto simp: field_split_simps
⟨n < m⟩ diff_less_mono2)
  qed auto
  then show ?thesis
  using LR [of c((2*k + 1) / 2n) a((2*k + 1) / 2n) b((2*k + 1) /
2n)]
  using alec [of 2*k+1] cleb [of 2*k+1] a_ge_0 [of 2*k+1] b_le_1 [of
2*k+1]
  using k a41 b41 ⟨0 < n⟩
  by (simp add: add commute)
next
case equal then show ?thesis by simp
next
case greater
  moreover have real (4 * k + 3) / 2Suc n - 1 / (2Suc n) = real j
/ 2n
  using k by (force simp: field_split_simps)
  moreover have |real i / 2m - real j / 2n| < 2 * 1 / (2Suc n)
  using less.prem1 by simp
  ultimately have closer: |real i / 2m - real (4 * k + 3) / 2Suc n|
< 1 / (2Suc n)
  using less.prem1 by linarith
  have a (real (4 * k + 3) / 2Suc n) ≤ c (real i / 2m) ∧
c (real i / 2m) ≤ b (real (4 * k + 3) / 2Suc n)
  proof (rule less.IH [OF _ refl])

```



```

    show  $m - \text{Suc } n < d$ 
      using  $\langle n < m \rangle$  diff_less_mono2 less.prem1 by blast
    show  $|\text{real } i / 2^m - \text{real } (4 * k + 3) / 2^{\text{Suc } n}| < 1 / 2^{\text{Suc } n}$ 
      using closer  $\langle n < m \rangle$   $\langle d = m - n \rangle$  by (auto simp: field_split_simps
 $\langle n < m \rangle$  diff_less_mono2)
    qed auto
    then show ?thesis
      using LR [of  $c((2*k + 1) / 2^n)$   $a((2*k + 1) / 2^n)$   $b((2*k + 1) /$ 
 $2^n)$ ]
      using alec [of  $2*k+1$ ] cleb [of  $2*k+1$ ] a_ge_0 [of  $2*k+1$ ] b_le_1 [of
 $2*k+1$ ]
      using k a43 b43  $\langle 0 < n \rangle$ 
      by (simp add: add.commute)
    qed
  qed
  qed
  then have aj_le_ci:  $a(\text{real } j / 2^n) \leq c(\text{real } i / 2^m)$ 
    and ci_le_bj:  $c(\text{real } i / 2^m) \leq b(\text{real } j / 2^n)$  if odd j  $|\text{real } i / 2^m -$ 
 $\text{real } j / 2^n| < 1/2^n$  for  $i j m n$ 
    using that by blast+
  have close_ab:  $\text{odd } m \implies |a(\text{real } m / 2^n) - b(\text{real } m / 2^n)| \leq 2 / 2^n$ 
for  $m n$ 
  proof (induction n arbitrary: m)
    case 0
      with u01 v01 show ?case by auto
    next
      case (Suc n m)
        with oddE obtain  $k$  where  $m = \text{Suc } (2*k)$  by fastforce
        show ?case
          proof (cases n > 0)
            case False
              with u01 v01 show ?thesis
                by (simp add: a_def b_def leftrec_base rightrec_base)
            next
              case True
                show ?thesis
                  proof (cases even k)
                    case True
                      then obtain  $j$  where  $k = 2*j$  by (metis evenE)
                      have  $|a((2 * \text{real } j + 1) / 2^n) - (b((2 * \text{real } j + 1) / 2^n))| \leq 2/2^n$ 
 $^n$ 
                      proof -
                        have odd (Suc k)
                          using True by auto
                        then show ?thesis
                          by (metis (no_types) Groups.add_ac(2) Suc.IH j of_nat_Suc of_nat_mult
of_nat_numeral)
                      qed
                    case False

```

```

moreover have  $a ((2 * \text{real } j + 1) / 2^{\wedge} n) \leq$ 
   $\text{leftcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of 2*j+1]
  by (auto simp: add commute left_right)
moreover have  $\text{leftcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n)) \leq$ 
   $c ((2 * \text{real } j + 1) / 2^{\wedge} n)$ 
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of 2*j+1]
  by (auto simp: add commute left_right_m)
ultimately have  $|a ((2 * \text{real } j + 1) / 2^{\wedge} n) -$ 
   $\text{leftcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n))|$ 
   $\leq 2 / 2^{\wedge} \text{Suc } n$ 
  by (simp add: c_def midpoint_def)
with j k <n > 0 show ?thesis
  by (simp add: add commute [of 1] a41 b41 del: power_Suc)
next
case False
then obtain j where j: k = 2*j + 1 by (metis oddE)
have  $|a ((2 * \text{real } j + 1) / 2^{\wedge} n) - (b ((2 * \text{real } j + 1) / 2^{\wedge} n))| \leq 2 / 2^{\wedge} n$ 
  using Suc.IH [OF False] j by (auto simp: algebra_simps)
moreover have  $c ((2 * \text{real } j + 1) / 2^{\wedge} n) \leq$ 
   $\text{rightcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of 2*j+1]
  by (auto simp: add commute left_right_m)
moreover have  $\text{rightcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n)) \leq$ 
   $b ((2 * \text{real } j + 1) / 2^{\wedge} n)$ 
  using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1] b_le_1 [of 2*j+1]
  by (auto simp: add commute left_right)
ultimately have  $|\text{rightcut } (a ((2 * \text{real } j + 1) / 2^{\wedge} n)) (b ((2 * \text{real } j + 1) / 2^{\wedge} n))$ 
   $(c ((2 * \text{real } j + 1) / 2^{\wedge} n)) -$ 
   $b ((2 * \text{real } j + 1) / 2^{\wedge} n)| \leq 2 / 2^{\wedge} \text{Suc } n$ 
  by (simp add: c_def midpoint_def)
with j k <n > 0 show ?thesis
  by (simp add: add commute [of 3] a43 b43 del: power_Suc)
qed
qed
qed
have m1_to_3: 4 * real k - 1 = real (4 * (k-1)) + 3 if  $0 < k$  for k
  using that by auto
have fb_eq_fa: [0 < j; 2*j < 2^{\wedge} n]  $\implies f(b((2 * \text{real } j - 1) / 2^{\wedge} n)) = f(a((2$ 

```

```

* real  $j + 1 / 2^n$ ) for  $n j$ 
proof (induction n arbitrary: j)
  case 0
  then show ?case by auto
next
  case (Suc n j) show ?case
proof (cases n > 0)
  case False
  with Suc.premis show ?thesis by auto
next
  case True
  show ?thesis proof (cases even j)
  case True
  then obtain k where k:  $j = 2 * k$  by (metis evenE)
  with <0 < j> have  $k > 0 \ 2 * k < 2^n$ 
  using Suc.premis(2) k by auto
  with k <0 < n> Suc.IH [of k] show ?thesis
  by (simp add: m1_to_3 a41 b43 del: power_Suc) (auto simp: of_nat_diff)
next
  case False
  then obtain k where k:  $j = 2 * k + 1$  by (metis oddE)
  have f (leftcut (a ((2 * k + 1) / 2^n)) (b ((2 * k + 1) / 2^n)) (c ((2 * k
+ 1) / 2^n)))
    = f (c ((2 * k + 1) / 2^n))
    f (c ((2 * k + 1) / 2^n))
    = f (rightcut (a ((2 * k + 1) / 2^n)) (b ((2 * k + 1) / 2^n)) (c ((2
* k + 1) / 2^n)))
  using alec [of 2*k+1 n] cleb [of 2*k+1 n] a_ge_0 [of 2*k+1 n] b_le_1
[of 2*k+1 n] k
  using left_right_m [of c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k +
1) / 2^n)]
  by (auto simp: add commute feqm [OF order_refl] feqm [OF _ order_refl,
symmetric])
  then
  show ?thesis
  by (simp add: k add commute [of 1] add commute [of 3] a43 b41 <0 < n>
del: power_Suc)
  qed
qed
qed
have f_eq_fc:  $\llbracket 0 < j; j < 2^n \rrbracket$ 
   $\implies f(b((2*j - 1) / 2^{Suc n})) = f(c(j / 2^n)) \wedge$ 
   $f(a((2*j + 1) / 2^{Suc n})) = f(c(j / 2^n))$  for n and j::nat
proof (induction n arbitrary: j)
  case 0
  then show ?case by auto
next
  case (Suc n)
  show ?case

```

```

proof (cases even j)
  case True
  then obtain k where k: j = 2*k by (metis evenE)
  then have less2n: k < 2 ^ n
    using Suc.prem2 by auto
  have 0 < k using ‹0 < j› k by linarith
  then have m1_to_3: real (4 * k - Suc 0) = real (4 * (k-1)) + 3
    by auto
  then show ?thesis
    using Suc.IH [of k] k ‹0 < k›
    by (simp add: less2n add.commute [of 1] m1_to_3 a41 b43 del: power_Suc)
(auto simp: of_nat_diff)
  next
  case False
  then obtain k where k: j = 2*k + 1 by (metis oddE)
  with Suc.prem2 have k < 2^n by auto
  show ?thesis
    using alec [of 2*k+1 Suc n] cleb [of 2*k+1 Suc n] a_ge_0 [of 2*k+1 Suc
n] b_le_1 [of 2*k+1 Suc n] k
    using left_right_m [of c((2*k + 1) / 2 ^ Suc n) a((2*k + 1) / 2 ^ Suc n)
b((2*k + 1) / 2 ^ Suc n)]
    apply (simp add: add.commute [of 1] add.commute [of 3] m1_to_3 b41 a43
del: power_Suc)
    apply (force intro: feqm)
  done
  qed
qed
define D01 where D01 ≡ {0 <..have cloD01 [simp]: closure D01 = {0..1}
  unfolding D01_def
  by (subst closure_dyadic_rationals_in_convex_set_pos_1) auto
have uniformly_continuous_on D01 (f ∘ c)
proof (clarsimp simp: uniformly_continuous_on_def)
  fix e::real
  assume 0 < e
  have ucontf: uniformly_continuous_on {0..1} f
    by (simp add: compact_uniformly_continuous [OF cont_f])
  then obtain d where 0 < d and d: ∀x x'. ‹x ∈ {0..1}; x' ∈ {0..1}; norm
(x' - x) < d› ⇒ norm (f x' - f x) < e/2
    unfolding uniformly_continuous_on_def dist_norm
    by (metis ‹0 < e› less_divide_eq_numeral1(1) mult_zero_left)
  obtain n where n: 1/2^n < min d 1
  by (metis ‹0 < d› divide_less_eq_1 less_numeral_extra(1) min_def one_less_numeral_iff
power_one_over_real_arch_pow_inv semiring_norm(76) zero_less_numeral)
  with gr0I have n > 0
    by (force simp: field_split_simps)
  show ∃ d>0. ∀x∈D01. ∀x'∈D01. dist x' x < d ⇒ dist (f (c x')) (f (c x)) <
e
  proof (intro exI ballI impI conjI)

```

```

  show (0::real) < 1/2^n by auto
next
  have dist_fc_close: dist (f(c(real i / 2^m))) (f(c(real j / 2^n))) < e/2
    if i: 0 < i < 2^m and j: 0 < j < 2^n and clo: abs(i / 2^m - j /
2^n) < 1/2^n for i j m
  proof -
    have abs3: |x - a| < e  $\implies$  x = a  $\vee$  |x - (a - e/2)| < e/2  $\vee$  |x - (a +
e/2)| < e/2 for x a e::real
    by linarith
    consider i / 2^m = j / 2^n
    | |i / 2^m - (2 * j - 1) / 2^(Suc n)| < 1/2^(Suc n)
    | |i / 2^m - (2 * j + 1) / 2^(Suc n)| < 1/2^(Suc n)
    using abs3 [OF clo] j by (auto simp: field_simps of_nat_diff)
  then show ?thesis
  proof cases
    case 1 with ⟨0 < e⟩ show ?thesis by auto
  next
    case 2
    have *: abs(a - b) ≤ 1/2^n ∧ 1/2^n < d ∧ a ≤ c ∧ c ≤ b  $\implies$  b -
c < d for a b c
    by auto
    have norm (c (real i / 2^m) - b (real (2 * j - 1) / 2^(Suc n))) < d
      using 2 j n close_ab [of 2*j-1 Suc n]
      using b_ge_0 [of 2*j-1 Suc n] b_le_1 [of 2*j-1 Suc n]
      using aj_le_ci [of 2*j-1 i m Suc n]
      using ci_le_bj [of 2*j-1 i m Suc n]
      apply (simp add: divide_simps of_nat_diff del: power_Suc)
      apply (auto simp: divide_simps intro!: *)
    done
    moreover have f(c(j / 2^n)) = f(b ((2*j - 1) / 2^(Suc n)))
      using f_eq_fc [OF j] by metis
    ultimately show ?thesis
      by (metis dist_norm atLeastAtMost_iff b_ge_0 b_le_1 c_ge_0 c_le_1
d)
  next
    case 3
    have *: abs(a - b) ≤ 1/2^n ∧ 1/2^n < d ∧ a ≤ c ∧ c ≤ b  $\implies$  c -
a < d for a b c
    by auto
    have norm (c (real i / 2^m) - a (real (2 * j + 1) / 2^(Suc n))) < d
      using 3 j n close_ab [of 2*j+1 Suc n]
      using b_ge_0 [of 2*j+1 Suc n] b_le_1 [of 2*j+1 Suc n]
      using aj_le_ci [of 2*j+1 i m Suc n]
      using ci_le_bj [of 2*j+1 i m Suc n]
      apply (simp add: divide_simps of_nat_diff del: power_Suc)
      apply (auto simp: divide_simps intro!: *)
    done
    moreover have f(c(j / 2^n)) = f(a ((2*j + 1) / 2^(Suc n)))
      using f_eq_fc [OF j] by metis

```

```

ultimately show ?thesis
  by (metis dist_norm a_ge_0 atLeastAtMost_iff a_ge_0 a_le_1 c_ge_0
c_le_1 d)
qed
qed
show dist (f (c x')) (f (c x)) < e
  if x ∈ D01 x' ∈ D01 dist x' x < 1/2^n for x x'
  using that unfolding D01_def dyadics_in_open_unit_interval
proof clarsimp
  fix i k::nat and m p
  assume i: 0 < i i < 2^m and k: 0 < k k < 2^p
  assume clo: dist (real k / 2^p) (real i / 2^m) < 1/2^n
  obtain j::nat where 0 < j j < 2^n
    and clo_ij: abs(i / 2^m - j / 2^n) < 1/2^n
    and clo_kj: abs(k / 2^p - j / 2^n) < 1/2^n
  proof -
    have max (2^n * i / 2^m) (2^n * k / 2^p) ≥ 0
      by (auto simp: le_max_iff_disj)
    then obtain j where floor (max (2^n * i / 2^m) (2^n * k / 2^p)) = int j
      using zero_le_floor zero_le_imp_eq_int by blast
    then have j_le: real j ≤ max (2^n * i / 2^m) (2^n * k / 2^p)
      and less_j1: max (2^n * i / 2^m) (2^n * k / 2^p) < real j + 1
      using floor_correct [of max (2^n * i / 2^m) (2^n * k / 2^p)] by
linarith+
    show thesis
    proof (cases j = 0)
      case True
      show thesis
      proof
        show (1::nat) < 2^n
          by (metis Suc_1 ‹0 < n› lessI one_less_power)
        show |real i / 2^m - real 1/2^n| < 1/2^n
          using i less_j1 by (simp add: dist_norm field_simps True)
        show |real k / 2^p - real 1/2^n| < 1/2^n
          using k less_j1 by (simp add: dist_norm field_simps True)
        qed simp
      next
      case False
      have 1: real j * 2^m < real i * 2^n
        if j: real j * 2^p ≤ real k * 2^n and k: real k * 2^m < real i * 2^p
        for i k m p
      proof -
        have real j * 2^p * 2^m ≤ real k * 2^n * 2^m
          using j by simp
        moreover have real k * 2^m * 2^n < real i * 2^p * 2^n
          using k by simp
        ultimately have real j * 2^p * 2^m < real i * 2^p * 2^n
          by (simp only: mult_ac)
    end
  end

```

```

    then show ?thesis
      by simp
  qed
  have 2:  $\text{real } j * 2^m < 2^m + \text{real } i * 2^n$ 
    if j:  $\text{real } j * 2^p \leq \text{real } k * 2^n$  and k:  $\text{real } k * (2^m * 2^n) < 2^m * 2^p + \text{real } i * (2^n * 2^p)$ 
    for i k m p
  proof -
    have  $\text{real } j * 2^p * 2^m \leq \text{real } k * (2^m * 2^n)$ 
      using j by simp
    also have  $\dots < 2^m * 2^p + \text{real } i * (2^n * 2^p)$ 
      by (rule k)
    finally have  $(\text{real } j * 2^m) * 2^p < (2^m + \text{real } i * 2^n) * 2^p$ 
      by (simp add: algebra_simps)
    then show ?thesis
      by simp
  qed
  have 3:  $\text{real } j * 2^p < 2^p + \text{real } k * 2^n$ 
    if j:  $\text{real } j * 2^m \leq \text{real } i * 2^n$  and i:  $\text{real } i * 2^p \leq \text{real } k * 2^n$ 
  proof -
    have  $\text{real } j * 2^m * 2^p \leq \text{real } i * 2^n * 2^p$ 
      using j by simp
    moreover have  $\text{real } i * 2^p * 2^n \leq \text{real } k * 2^m * 2^n$ 
      using i by simp
    ultimately have  $\text{real } j * 2^m * 2^p \leq \text{real } k * 2^m * 2^n$ 
      by (simp only: mult_ac)
    then have  $\text{real } j * 2^p \leq \text{real } k * 2^n$ 
      by simp
    also have  $\dots < 2^p + \text{real } k * 2^n$ 
      by auto
    finally show ?thesis by simp
  qed
  show ?thesis
  proof
    have  $2^n * \text{real } i / 2^m < 2^n * \text{real } k / 2^p < 2^n$ 
      using i k by (auto simp: field_simps)
    then have  $\max(2^n * i / 2^m) (2^n * k / 2^p) < 2^n$ 
      by simp
    with j_le have  $\text{real } j < 2^n$  by linarith
    then show  $j < 2^n$ 
      by auto
    have  $|\text{real } i * 2^n - \text{real } j * 2^m| < 2^m$ 
      using clo_less_j1 j_le
      by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
        split: if_split_asm dest: 1 2)
    then show  $|\text{real } i / 2^m - \text{real } j / 2^n| < 1/2^n$ 
      by (auto simp: field_split_simps)
    have  $|\text{real } k * 2^n - \text{real } j * 2^p| < 2^p$ 

```

```

    using clo_less_j1_j_le
    by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
split: if_split_asm dest: 3 2)
    then show  $|real\ k / 2^p - real\ j / 2^n| < 1/2^n$ 
    by (auto simp: le_max_iff_disj field_split_simps dist_norm)
    qed (use False in simp)
  qed
  qed
  show  $dist\ (f\ (c\ (real\ k / 2^p)))\ (f\ (c\ (real\ i / 2^m))) < e$ 
  proof (rule dist_triangle_half_l)
    show  $dist\ (f\ (c\ (real\ k / 2^p)))\ (f\ (c\ (j / 2^n))) < e/2$ 
    using  $\langle 0 < j \rangle \langle j < 2^n \rangle k\ clo\_kj$ 
    by (intro dist_fc_close) auto
    show  $dist\ (f\ (c\ (real\ i / 2^m)))\ (f\ (c\ (real\ j / 2^n))) < e/2$ 
    using  $\langle 0 < j \rangle \langle j < 2^n \rangle i\ clo\_ij$ 
    by (intro dist_fc_close) auto
  qed
  qed
  qed
  then obtain h where ucont_h: uniformly_continuous_on {0..1} h
  and fc_eq:  $\bigwedge x. x \in D01 \implies (f \circ c)\ x = h\ x$ 
  proof (rule uniformly_continuous_on_extension_on_closure [of D01 f \circ c])
    qed (use closure_subset [of D01] in <auto intro!: that>)
  then have cont_h: continuous_on {0..1} h
  using uniformly_continuous_imp_continuous by blast
  have h_eq:  $h\ (real\ k / 2^m) = f\ (c\ (real\ k / 2^m))$  if  $0 < k < 2^m$  for k
  m
  using fc_eq that by (force simp: D01_def)
  have h' {0..1} = f' {0..1}
  proof
    have  $h' (closure\ D01) \subseteq f' \{0..1\}$ 
    proof (rule image_closure_subset)
      show continuous_on (closure D01) h
      using cont_h by simp
      show closed (f' {0..1})
      using compact_continuous_image [OF cont_f] compact_imp_closed by
blast
      show  $h' D01 \subseteq f' \{0..1\}$ 
      by (force simp: dyadics_in_open_unit_interval D01_def h_eq)
    qed
  qed
  with cloD01 show  $h' \{0..1\} \subseteq f' \{0..1\}$  by simp
  have a12 [simp]:  $a\ (1/2) = u$ 
  by (metis a_def leftrec_base numeral_One of_nat_numeral)
  have b12 [simp]:  $b\ (1/2) = v$ 
  by (metis b_def rightrec_base numeral_One of_nat_numeral)
  have f' {0..1}  $\subseteq closure(h' D01)$ 
  proof (clarsimp simp: closure_approachable dyadics_in_open_unit_interval
D01_def)

```



```

fix x e::real
assume 0 ≤ x x ≤ 1 0 < e
have ucont_f: uniformly_continuous_on {0..1} f
  using compact_uniformly_continuous cont_f by blast
then obtain δ where δ > 0
  and δ: ∧x x'. [|x ∈ {0..1}; x' ∈ {0..1}; dist x' x < δ|] ⇒ norm (f x' - f
x) < e
  using ⟨0 < e⟩ by (auto simp: uniformly_continuous_on_def dist_norm)
have *: ∃ m::nat. ∃ y. odd m ∧ 0 < m ∧ m < 2 ^ n ∧ y ∈ {a(m / 2 ^ n) ..
b(m / 2 ^ n)} ∧ f y = f x
  if n ≠ 0 for n
  using that
proof (induction n)
case 0 then show ?case by auto
next
case (Suc n)
show ?case
proof (cases n=0)
case True
consider x ∈ {0..u} | x ∈ {u..v} | x ∈ {v..1}
  using ⟨0 ≤ x⟩ ⟨x ≤ 1⟩ by force
then have ∃ y ≥ a (real 1/2). y ≤ b (real 1/2) ∧ f y = f x
proof cases
case 1
then show ?thesis
  using uabv [of 1 1] f0u [of u] f0u [of x] by force
next
case 2
then show ?thesis
  by (rule_tac x=x in exI) auto
next
case 3
then show ?thesis
  using uabv [of 1 1] fv1 [of v] fv1 [of x] by force
qed
with ⟨n=0⟩ show ?thesis
  by (rule_tac x=1 in exI) auto
next
case False
with Suc obtain m y
  where odd m 0 < m and mless: m < 2 ^ n
  and y: y ∈ {a (real m / 2 ^ n)..b (real m / 2 ^ n)} and feq: f y = f x
  by metis
then obtain j where j: m = 2*j + 1 by (metis oddE)
have j4: 4 * j + 1 < 2 ^ Suc n
  using mless j by (simp add: algebra_simps)

consider y ∈ {a((2*j + 1) / 2 ^ n) .. b((4*j + 1) / 2 ^ (Suc n))}
  | y ∈ {b((4*j + 1) / 2 ^ (Suc n)) .. a((4*j + 3) / 2 ^ (Suc n))}

```

```

| y ∈ {a((4*j + 3) / 2 ^ (Suc n)) .. b((2*j + 1) / 2 ^ n)}
using y j by force
then show ?thesis
proof cases
case 1
show ?thesis
proof (intro exI conjI)
show y ∈ {a (real (4 * j + 1) / 2 ^ Suc n)..b (real (4 * j + 1) / 2 ^
Suc n)}
using mless j ⟨n ≠ 0⟩ 1 by (simp add: a41 b41 add.commute [of 1]
del: power_Suc)
qed (use feq j4 in auto)
next
case 2
show ?thesis
proof (intro exI conjI)
show b (real (4 * j + 1) / 2 ^ Suc n) ∈ {a (real (4 * j + 1) / 2 ^
Suc n)..b (real (4 * j + 1) / 2 ^ Suc n)}
using ⟨n ≠ 0⟩ alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1
n] b_le_1 [of 2*j+1 n]
using left_right [of c((2*j + 1) / 2 ^ n) a((2*j + 1) / 2 ^ n) b((2*j
+ 1) / 2 ^ n)]
by (simp add: a41 b41 add.commute [of 1] del: power_Suc)
show f (b (real (4 * j + 1) / 2 ^ Suc n)) = f x
using ⟨n ≠ 0⟩ 2
using alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1 n]
b_le_1 [of 2*j+1 n]
by (force simp add: b41 a43 add.commute [of 1] feq [symmetric] simp
del: power_Suc intro: f_eqI)
qed (use j4 in auto)
next
case 3
show ?thesis
proof (intro exI conjI)
show 4 * j + 3 < 2 ^ Suc n
using mless j by simp
show f y = f x
by fact
show y ∈ {a (real (4 * j + 3) / 2 ^ Suc n) .. b (real (4 * j + 3) / 2
^ Suc n)}
using 3 False b43 [of n j] by (simp add: add.commute)
qed (use 3 in auto)
qed
qed
obtain n where n: 1/2^n < min (δ / 2) 1
by (metis ⟨0 < δ⟩ divide_less_eq_1 less_numeral_extra(1) min_less_iff_conj
one_less_numeral_iff_power_one_over_real_arch_pow_inv semiring_norm(76)
zero_less_divide_iff zero_less_numeral)

```

```

with gr0I have n ≠ 0
  by fastforce
with * obtain m::nat and y
  where odd m 0 < m and mless: m < 2 ^ n
    and y: a(m / 2 ^ n) ≤ y ∧ y ≤ b(m / 2 ^ n) and feq: f x = f y
  by (metis atLeastAtMost_iff)
then have 0 ≤ y y ≤ 1
  by (meson a_ge_0 b_le_1 order.trans)+
moreover have y < δ + c (real m / 2 ^ n) c (real m / 2 ^ n) < δ + y
  using y alec [of m n] cleb [of m n] n field_sum_of_halves close_ab [OF
<odd m>, of n]
  by linarith+
moreover note <0 < m> mless <0 ≤ x> <x ≤ 1>
ultimately have dist (h (real m / 2 ^ n)) (f x) < e
  by (auto simp: dist_norm h_eq feq δ)
then show ∃ k. ∃ m ∈ {0 <..< 2 ^ k}. dist (h (real m / 2 ^ k)) (f x) < e
  using <0 < m> greaterThanLessThan_iff mless by blast
qed
also have ... ⊆ h ' {0..1}
proof (rule closure_minimal)
  show h ' D01 ⊆ h ' {0..1}
    using cloD01 closure_subset by blast
  show closed (h ' {0..1})
    using compact_continuous_image [OF cont_h] compact_imp_closed by
auto
qed
finally show f ' {0..1} ⊆ h ' {0..1} .
qed
moreover have inj_on h {0..1}
proof -
  have u < v
  by (metis atLeastAtMost_iff f0u f_1not0 fv1 order.not_eq_order_implies_strict
u01(1) u01(2) v01(1))
  have f_not_fu: ∧x. [[u < x; x ≤ v]] ⇒ f x ≠ f u
  by (metis atLeastAtMost_iff f0u fv1 greaterThanAtMost_iff order_refl or-
der_trans u01(1) v01(2))
  have f_not_fv: ∧x. [[u ≤ x; x < v]] ⇒ f x ≠ f v
  by (metis atLeastAtMost_iff order_refl order_trans v01(2) atLeastLessThan_iff
fv fv1)
  have a_less_b:
    a(j / 2 ^ n) < b(j / 2 ^ n) ∧
    (∀ x. a(j / 2 ^ n) < x → x ≤ b(j / 2 ^ n) → f x ≠ f(a(j / 2 ^ n))) ∧
    (∀ x. a(j / 2 ^ n) ≤ x → x < b(j / 2 ^ n) → f x ≠ f(b(j / 2 ^ n))) for n
and j::nat
proof (induction n arbitrary: j)
  case 0 then show ?case
    by (simp add: <u < v> f_not_fu f_not_fv)
next
  case (Suc n j) show ?case

```

```

proof (cases n > 0)
  case False then show ?thesis
    by (auto simp: a_def b_def leftrec_base rightrec_base ⟨u < v⟩ f_not_fu
f_not_fv)
  next
    case True show ?thesis
    proof (cases even j)
      case True
        with ⟨0 < n⟩ Suc.IH show ?thesis
          by (auto elim!: evenE)
      next
        case False
          then obtain k where k: j = 2*k + 1 by (metis oddE)
          then show ?thesis
          proof (cases even k)
            case True
              then obtain m where m: k = 2*m by (metis evenE)
              have fleft: f (leftcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n))) (c
((2*m + 1) / 2^n)) =
                f (c((2*m + 1) / 2^n))
                using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
                using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
                by (auto intro: f_eqI)
              show ?thesis
              proof (intro conjI impI notI allI)
                have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
                proof -
                  have f (c ((1 + real m * 2) / 2 ^ n)) = f (a ((1 + real m * 2) / 2
^ n))
                  using k m ⟨0 < n⟩ fleft that a41 [of n m] b41 [of n m]
                  using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
                  using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
                  by (auto simp: algebra_simps)
                  moreover have a (real (1 + m * 2) / 2 ^ n) < c (real (1 + m *
2) / 2 ^ n)
                  using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
                  moreover have c (real (1 + m * 2) / 2 ^ n) ≤ b (real (1 + m * 2)
/ 2 ^ n)
                  using cleb by blast
                ultimately show ?thesis
                using Suc.IH [of 1 + m * 2] by force
              qed
            then show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
          next
            fix x

```

```

      assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
(a (real j / 2 ^ Suc n))
    then show False
      using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct1]
      using k m <0 < n> a41 [of n m] b41 [of n m]
      using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
      using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
      by (auto simp: algebra_simps)
    next
      fix x
      assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
    then show False
      using k m <0 < n> a41 [of n m] b41 [of n m] fleft left_neq
      using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
      by (auto simp: algebra_simps)
    qed
  next
  case False
  with oddE obtain m where m: k = Suc (2*m) by fastforce
  have fright: f (rightcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n))
(c ((2*m + 1) / 2^n))) = f (c((2*m + 1) / 2^n))
    using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
    using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
    by (auto intro: f_eqI [OF _ order_refl])
  show ?thesis
  proof (intro conjI impI notI allI)
    have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
    proof -
      have f (c ((1 + real m * 2) / 2 ^ n)) = f (b ((1 + real m * 2) / 2
^ n))
        using k m <0 < n> fright that a43 [of n m] b43 [of n m]
        using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
        using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
        by (auto simp: algebra_simps)
      moreover have a (real (1 + m * 2) / 2 ^ n) ≤ c (real (1 + m *
2) / 2 ^ n)
        using alec by blast
      moreover have c (real (1 + m * 2) / 2 ^ n) < b (real (1 + m * 2)
/ 2 ^ n)
        using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
      ultimately show ?thesis

```

```

    using Suc.IH [of 1 + m * 2] by force
  qed
  then show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
next
  fix x
  assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
(a (real j / 2 ^ Suc n))
  then show False
    using k m < 0 < n> a43 [of n m] b43 [of n m] fright right_neq
    using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
    by (auto simp: algebra_simps)
  next
  fix x
  assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
  then show False
    using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct2]
    using k m < 0 < n> a43 [of n m] b43 [of n m]
    using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
    using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
    by (auto simp: algebra_simps fright simp del: power_Suc)
  qed
  qed
  qed
  qed
  have c_gt_0 [simp]: 0 < c(m / 2^n) and c_less_1 [simp]: c(m / 2^n) < 1
for m::nat and n
  using a_less_b [of m n] apply (simp_all add: c_def midpoint_def)
  using a_ge_0 [of m n] b_le_1 [of m n] by linarith+
  have approx: ∃ j n. odd j ∧ n ≠ 0 ∧
    real i / 2^m ≤ real j / 2^n ∧
    real j / 2^n ≤ real k / 2^p ∧
    |real i / 2^m - real j / 2^n| < 1/2^n ∧
    |real k / 2^p - real j / 2^n| < 1/2^n
  if 0 < i i < 2^m 0 < k k < 2^p i / 2^m < k / 2^p m + p = N for N m
p i k
  using that
  proof (induction N arbitrary: m p i k rule: less_induct)
  case (less N)
  then consider i / 2^m ≤ 1/2 1/2 ≤ k / 2^p | k / 2^p < 1/2 | k / 2^p ≥
1/2 1/2 < i / 2^m
    by linarith
  then show ?case
  proof cases
  case 1

```

```

with less.prem $s$  show ?thesis
  by (rule_tac  $x=1$  in  $exI$ ) + (fastforce simp: field_split_simps)
next
case 2 show ?thesis
proof (cases  $m$ )
  case 0 with less.prem $s$  show ?thesis
    by auto
next
case (Suc  $m'$ ) show ?thesis
proof (cases  $p$ )
  case 0 with less.prem $s$  show ?thesis by auto
next
case (Suc  $p'$ )
  have  $\S$ : False if  $real\ i * 2^{\wedge} p' < real\ k * 2^{\wedge} m'$   $k < 2^{\wedge} p'$   $2^{\wedge} m' \leq i$ 
  proof -
    have  $real\ k * 2^{\wedge} m' < 2^{\wedge} p' * 2^{\wedge} m'$ 
      using that by simp
    then have  $real\ i * 2^{\wedge} p' < 2^{\wedge} p' * 2^{\wedge} m'$ 
      using that by linarith
    with that show ?thesis by simp
  qed
  moreover have  $*$ :  $real\ i / 2^{\wedge} m' < real\ k / 2^{\wedge} p'$   $k < 2^{\wedge} p'$ 
  using less.prem $s$   $\langle m = Suc\ m' \rangle$  2 Suc by (force simp: field_split_simps) +
  moreover have  $i < 2^{\wedge} m'$ 
    using  $\S *$  by (clarsimp simp: divide_simps linorder_not_le) (meson
linorder_not_le)
  ultimately show ?thesis
    using less.IH [of  $m'+p'$   $i\ m'\ k\ p'$ ] less.prem $s$   $\langle m = Suc\ m' \rangle$  2 Suc
    by (force simp: field_split_simps)
  qed
qed
next
case 3 show ?thesis
proof (cases  $m$ )
  case 0 with less.prem $s$  show ?thesis
    by auto
next
case (Suc  $m'$ ) show ?thesis
proof (cases  $p$ )
  case 0 with less.prem $s$  show ?thesis by auto
next
case (Suc  $p'$ )
  have  $real\ (i - 2^{\wedge} m') / 2^{\wedge} m' < real\ (k - 2^{\wedge} p') / 2^{\wedge} p'$ 
    using less.prem $s$   $\langle m = Suc\ m' \rangle$  Suc 3 by (auto simp: field_simps
of_nat_diff)
  moreover have  $k - 2^{\wedge} p' < 2^{\wedge} p'$   $i - 2^{\wedge} m' < 2^{\wedge} m'$ 
    using less.prem $s$  Suc  $\langle m = Suc\ m' \rangle$  by auto
  moreover
  have  $2^{\wedge} p' \leq k$   $2^{\wedge} p' \neq k$ 

```

```

    using less.premis ⟨m = Suc m'⟩ Suc 3 by auto
  then have 2 ^ p' < k
    by linarith
  ultimately show ?thesis
    using less.IH [of m'+p' i - 2 ^ m' m' k - 2 ^ p' p'] less.premis ⟨m =
Suc m'⟩ Suc 3
    apply (clarsimp simp: field_simps of_nat_diff)
    apply (rule_tac x=2 ^ n + j in exI, simp)
    apply (rule_tac x=Suc n in exI)
    apply (auto simp: field_simps)
  done
qed
qed
qed
qed
have clec: c(real i / 2 ^ m) ≤ c(real j / 2 ^ n)
  if i: 0 < i i < 2 ^ m and j: 0 < j j < 2 ^ n and ij: i / 2 ^ m < j / 2 ^ n for
m i n j
proof -
  obtain j' n' where odd j' n' ≠ 0
  and i_le_j: real i / 2 ^ m ≤ real j' / 2 ^ n'
  and j_le_j: real j' / 2 ^ n' ≤ real j / 2 ^ n
  and clo_ij: |real i / 2 ^ m - real j' / 2 ^ n'| < 1/2 ^ n'
  and clo_jj: |real j / 2 ^ n - real j' / 2 ^ n'| < 1/2 ^ n'
  using approx [of i m j n m+n] that i j ij by auto
  with oddE obtain q where q: j' = Suc (2*q) by fastforce
  have c (real i / 2 ^ m) ≤ c((2*q + 1) / 2 ^ n')
  proof (cases i / 2 ^ m = (2*q + 1) / 2 ^ n')
    case True then show ?thesis by simp
  next
    case False
    with i_le_j clo_ij q have |real i / 2 ^ m - real (4 * q + 1) / 2 ^ Suc n'|
< 1 / 2 ^ Suc n'
    by (auto simp: field_split_simps)
    then have c(i / 2 ^ m) ≤ b(real(4 * q + 1) / 2 ^ (Suc n'))
    by (meson ci_le_bj even_mult_iff even_numeral even_plus_one_iff)
    then show ?thesis
    using alec [of 2*q+1 n'] cleb [of 2*q+1 n'] a_ge_0 [of 2*q+1 n'] b_le_1
[of 2*q+1 n'] b41 [of n' q] ⟨n' ≠ 0⟩
    using left_right_m [of c((2*q + 1) / 2 ^ n') a((2*q + 1) / 2 ^ n') b((2*q
+ 1) / 2 ^ n')]
    by (auto simp: algebra_simps)
  qed
  also have ... ≤ c(real j / 2 ^ n)
  proof (cases j / 2 ^ n = (2*q + 1) / 2 ^ n')
    case True
    then show ?thesis by simp
  next
    case False
    case False

```



```

with j_le_j q have less:  $(2*q + 1) / 2^{n'} < j / 2^n$ 
  by auto
have *:  $\llbracket q < i; \text{abs}(i - q) < s*2; r = q + s \rrbracket \implies \text{abs}(i - r) < s$  for  $i q s$ 
r::real
  by auto
have  $|\text{real } j / 2^n - \text{real } (4 * q + 3) / 2^{\text{Suc } n'}| < 1 / 2^{\text{Suc } n'}$ 
  by (rule * [OF less]) (use j_le_j clo_jj q in <auto simp: field_split_simps>)
then have  $a(\text{real}(4*q + 3) / 2^{\text{Suc } n'}) \leq c(j / 2^n)$ 
  by (metis Suc3_eq_add_3 add commute aj_le_ci even_Suc even_mult_iff
even_numeral)
then show ?thesis
  using alec [of  $2*q+1$   $n'$ ] cleb [of  $2*q+1$   $n'$ ] a_ge_0 [of  $2*q+1$   $n'$ ] b_le_1
[of  $2*q+1$   $n'$ ] a43 [of  $n'$   $q$ ] < $n' \neq 0$ >
  using left_right_m [of  $c((2*q + 1) / 2^{n'})$   $a((2*q + 1) / 2^{n'})$   $b((2*q$ 
+ 1) /  $2^{n'})$ ]
  by (auto simp: algebra_simps)
qed
finally show ?thesis .
qed
have  $x = y$  if  $0 \leq x$   $x \leq 1$   $0 \leq y$   $y \leq 1$   $h x = h y$  for  $x y$ 
  using that
proof (induction x y rule: linorder_class.linorder_less_wlog)
case (less x1 x2)
  obtain m n where  $m: 0 < m$   $m < 2^n$ 
    and x12:  $x1 < m / 2^n$   $m / 2^n < x2$ 
    and neq:  $h x1 \neq h (\text{real } m / 2^n)$ 
  proof -
  have  $(x1 + x2) / 2 \in \text{closure } D01$ 
    using cloD01 less.hyps less.premis by auto
  with less obtain y where  $y \in D01$  and dist_y:  $\text{dist } y ((x1 + x2) / 2) <$ 
 $(x2 - x1) / 64$ 
    unfolding closure_approachable
  by (metis diff_gt_0_iff_gt less_divide_eq_numeral1(1) mult_zero_left)
  obtain m n where  $m: 0 < m$   $m < 2^n$ 
    and clo:  $|\text{real } m / 2^n - (x1 + x2) / 2| < (x2 - x1) / 64$ 
    and n:  $1/2^n < (x2 - x1) / 128$ 
  proof -
  have  $\min 1 ((x2 - x1) / 128) > 0$   $1/2 < (1::\text{real})$ 
    using less by auto
  then obtain N where  $N: 1/2^N < \min 1 ((x2 - x1) / 128)$ 
    by (metis power_one_over real_arch_pow_inv)
  then have  $N > 0$ 
    using less_divide_eq_1 by force
  obtain p q where  $p: p < 2^q$   $p \neq 0$  and yeq:  $y = \text{real } p / 2^q$ 
    using < $y \in D01$ > by (auto simp: zero_less_divide_iff D01_def)
  show ?thesis
  proof
  show  $0 < 2^N * p$ 
    using p by auto

```

```

show  $2^N * p < 2^{(N+q)}$ 
  by (simp add: p power_add)
  have  $|real (2^N * p) / 2^{(N+q)} - (x1 + x2) / 2| = |real p / 2^{q - (x1 + x2) / 2}|$ 
    by (simp add: power_add)
  also have  $... = |y - (x1 + x2) / 2|$ 
    by (simp add: yeq)
  also have  $... < (x2 - x1) / 64$ 
    using dist_y by (simp add: dist_norm)
  finally show  $|real (2^N * p) / 2^{(N+q)} - (x1 + x2) / 2| < (x2 - x1) / 64$  .
  have  $(1::real) / 2^{(N+q)} \leq 1/2^N$ 
    by (simp add: field_simps)
  also have  $... < (x2 - x1) / 128$ 
    using N by force
  finally show  $1/2^{(N+q)} < (x2 - x1) / 128$  .
qed
qed
obtain  $m' n' m'' n''$  where  $0 < m' m' < 2^{n'} x1 < m' / 2^{n'} m' / 2^{n'}$ 
 $< x2$ 
  and  $0 < m'' m'' < 2^{n''} x1 < m'' / 2^{n''} m'' / 2^{n''} < x2$ 
  and neg: h (real m'' / 2^{n''}) ≠ h (real m' / 2^{n'})
proof
  show  $0 < Suc (2*m)$ 
    by simp
  show m21: Suc (2*m) < 2^{Suc n}
    using m by auto
  show  $x1 < real (Suc (2 * m)) / 2^{Suc n}$ 
    using clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $real (Suc (2 * m)) / 2^{Suc n} < x2$ 
    using n clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $0 < 4*m + 3$ 
    by simp
  have  $m+1 \leq 2^n$ 
    using m by simp
  then have  $4 * (m+1) \leq 4 * (2^n)$ 
    by simp
  then show m43: 4*m + 3 < 2^{(n+2)}
    by (simp add: algebra_simps)
  show  $x1 < real (4 * m + 3) / 2^{(n+2)}$ 
    using clo by (simp add: field_simps abs_if_split: if_split_asm)
  show  $real (4 * m + 3) / 2^{(n+2)} < x2$ 
    using n clo by (simp add: field_simps abs_if_split: if_split_asm)
  have c_fold: midpoint (a ((2 * real m + 1) / 2^{Suc n})) (b ((2 * real m + 1) / 2^{Suc n})) = c ((2 * real m + 1) / 2^{Suc n})
    by (simp add: c_def)
  define R where  $R \equiv rightcut (a ((2 * real m + 1) / 2^{Suc n})) (b ((2 * real m + 1) / 2^{Suc n})) (c ((2 * real m + 1) / 2^{Suc n}))$ 
  have  $R < b ((2 * real m + 1) / 2^{Suc n})$ 

```

```

      unfolding R_def using a_less_b [of 4*m + 3 n+2] a43 [of Suc n m]
      b43 [of Suc n m]
      by simp
      then have Rless: R < midpoint R (b ((2 * real m + 1) / 2 ^ Suc n))
      by (simp add: midpoint_def)
      have midR_le: midpoint R (b ((2 * real m + 1) / 2 ^ Suc n)) ≤ b ((2 *
      real m + 1) / (2 * 2 ^ n))
      using ⟨R < b ((2 * real m + 1) / 2 ^ Suc n)⟩
      by (simp add: midpoint_def)
      have (real (Suc (2 * m)) / 2 ^ Suc n) ∈ D01 real (4 * m + 3) / 2 ^ (n
      + 2) ∈ D01
      by (simp_all add: D01_def m21 m43 del: power_Suc of_nat_Suc
      of_nat_add add_2_eq_Suc') blast+
      then show h (real (4 * m + 3) / 2 ^ (n + 2)) ≠ h (real (Suc (2 * m))
      / 2 ^ Suc n)
      using a_less_b [of 4*m + 3 n+2, THEN conjunct1]
      using a43 [of Suc n m] b43 [of Suc n m]
      using alec [of 2*m+1 Suc n] cleb [of 2*m+1 Suc n] a_ge_0 [of 2*m+1
      Suc n] b_le_1 [of 2*m+1 Suc n]
      apply (simp add: fc_eq [symmetric] c_def del: power_Suc)
      apply (simp only: add commute [of 1] c_fold R_def [symmetric])
      apply (rule right_neq)
      using Rless apply (simp add: R_def)
      apply (rule midR_le, auto)
      done
    qed
    then show ?thesis by (metis that)
  qed
  have m_div: 0 < m / 2 ^ n m / 2 ^ n < 1
  using m by (auto simp: field_split_simps)
  have closure0m: {0..m / 2 ^ n} = closure ({0<..

```

```

have less_2I:  $\bigwedge k i. \text{real } i / 2^k < 1 \implies i < 2^k$ 
  by simp
have h' ( $\{0 < \dots < m / 2^n\} \cap (\bigcup q p. \{\text{real } p / 2^q\}) \subseteq f' \{0..c(m / 2^n)\}$ )
proof clarsimp
  fix p q
  assume p:  $0 < \text{real } p / 2^q$   $\text{real } p / 2^q < \text{real } m / 2^n$ 
  then have [simp]:  $0 < p$ 
    by (simp add: field_split_simps)
  have [simp]:  $p < 2^q$ 
    by (blast intro: p less_2I m_div less_trans)
  have f ( $c(\text{real } p / 2^q) \in f' \{0..c(\text{real } m / 2^n)\}$ )
    by (auto simp: clec p m)
  then show  $h(\text{real } p / 2^q) \in f' \{0..c(\text{real } m / 2^n)\}$ 
    by (simp add: h_eq)
qed
with m_div have h' ( $\{0 .. m / 2^n\} \subseteq f' \{0 .. c(m / 2^n)\}$ )
  apply (subst closure0m)
  by (rule image_closure_subset [OF cont_h' closed_f]) auto
then have hx1:  $h x1 \in f' \{0 .. c(m / 2^n)\}$ 
  using x12 less.prem1 by auto
then obtain t1 where t1:  $h x1 = f t1$   $0 \leq t1$   $t1 \leq c(m / 2^n)$ 
  by auto
have h' ( $\{m / 2^n < \dots < 1\} \cap (\bigcup q p. \{\text{real } p / 2^q\}) \subseteq f' \{c(m / 2^n) .. 1\}$ )
proof clarsimp
  fix p q
  assume p:  $\text{real } m / 2^n < \text{real } p / 2^q$  and [simp]:  $p < 2^q$ 
  then have [simp]:  $0 < p$ 
    using gr_zeroI m_div by fastforce
  have f ( $c(\text{real } p / 2^q) \in f' \{c(m / 2^n) .. 1\}$ )
    by (auto simp: clec p m)
  then show  $h(\text{real } p / 2^q) \in f' \{c(m / 2^n) .. 1\}$ 
    by (simp add: h_eq)
qed
with m have h' ( $\{m / 2^n .. 1\} \subseteq f' \{c(m / 2^n) .. 1\}$ )
  apply (subst closurem1)
  by (rule image_closure_subset [OF cont_h' closed_f]) auto
then have hx2:  $h x2 \in f' \{c(m / 2^n) .. 1\}$ 
  using x12 less.prem1 by auto
then obtain t2 where t2:  $h x2 = f t2$   $c(m / 2^n) \leq t2$   $t2 \leq 1$ 
  by auto
with t1 less_neq have False
  using conn [of h x2, unfolded is_interval_connected_1 [symmetric] is_interval_1,
rule_format, of t1 t2 c(m / 2^n)]
  by (simp add: h_eq m)
then show ?case by blast
qed auto
then show ?thesis

```

```

    by (auto simp: inj_on_def)
  qed
  ultimately have {0..1::real} homeomorphic f ' {0..1}
    using homeomorphic_compact [OF _ cont_h] by blast
  then show ?thesis
    using homeomorphic_sym by blast
qed

theorem path_contains_arc:
  fixes p :: real  $\Rightarrow$  'a::{complete_space,real_normed_vector}
  assumes path p and a: pathstart p = a and b: pathfinish p = b and a  $\neq$  b
  obtains q where arc q path_image q  $\subseteq$  path_image p pathstart q = a pathfinish
q = b
proof -
  have ucont_p: uniformly_continuous_on {0..1} p
    using <path p> unfolding path_def
  by (metis compact_Icc compact_uniformly_continuous)
  define  $\varphi$  where  $\varphi \equiv \lambda S. S \subseteq \{0..1\} \wedge 0 \in S \wedge 1 \in S \wedge$ 
    ( $\forall x \in S. \forall y \in S. \text{open\_segment } x y \cap S = \{\}$ )  $\longrightarrow p x = p y$ 
  obtain T where closed T  $\varphi$  T and T:  $\bigwedge U. \llbracket \text{closed } U; \varphi U \rrbracket \Longrightarrow \neg (U \subset T)$ 
  proof (rule Brouwer_reduction_theorem_gen [of {0..1}  $\varphi$ ])
    have *:  $\{x <..<y\} \cap \{0..1\} = \{x <..<y\}$  if  $0 \leq x y \leq 1$   $x \leq y$  for  $x y :: \text{real}$ 
      using that by auto
    show  $\varphi$  {0..1}
      by (auto simp:  $\varphi$ _def open_segment_eq_real_ivl *)
    show  $\varphi$  ( $\bigcap (F ' UNIV)$ )
      if  $\bigwedge n. \text{closed } (F n)$  and  $\varphi: \bigwedge n. \varphi (F n)$  and Fsub:  $\bigwedge n. F (Suc n) \subseteq F n$  for
F
    proof -
      have F01:  $\bigwedge n. F n \subseteq \{0..1\} \wedge 0 \in F n \wedge 1 \in F n$ 
        and pq:  $\bigwedge n x y. \llbracket x \in F n; y \in F n; \text{open\_segment } x y \cap F n = \{\} \rrbracket \Longrightarrow p$ 
 $x = p y$ 
        by (metis  $\varphi$   $\varphi$ _def)+
      have pqF: False if  $\forall u. x \in F u \forall x. y \in F x$   $\text{open\_segment } x y \cap (\bigcap x. F x)$ 
 $= \{\}$  and neg:  $p x \neq p y$ 
        for  $x y$ 
        using that
      proof (induction  $x y$  rule: linorder_class.linorder_less_wlog)
        case (less  $x y$ )
        have xy:  $x \in \{0..1\} y \in \{0..1\}$ 
          by (metis less.prems subsetCE F01)+
        have norm( $p x - p y$ ) / 2 > 0
          using less by auto
        then obtain  $e$  where  $e > 0$ 
          and  $e: \bigwedge u v. \llbracket u \in \{0..1\}; v \in \{0..1\}; \text{dist } v u < e \rrbracket \Longrightarrow \text{dist } (p v) (p u)$ 
 $< \text{norm}(p x - p y) / 2$ 
          by (metis uniformly_continuous_onE [OF ucont_p])
        have minxy:  $\min e (y - x) < (y - x) * (3 / 2)$ 

```

```

    by (subst min_less_iff_disj) (simp add: less)
  define w where w ≡ x + (min e (y - x) / 3)
  define z where z ≡ y - (min e (y - x) / 3)
  have w < z and w: w ∈ {x<..\bigwedge T. T \in \text{range } F \implies \text{closed } T
    by (metis ⟨ $\bigwedge n. \text{closed } (F n)$ ⟩ image_iff)
  have eq:  $\{w..z\} \cap \bigcap (F ' UNIV) = \{\}$ 
    using less w z by (simp add: open_segment_eq_real_ivl disjoint_iff)
  then obtain K where finite K and K:  $\{w..z\} \cap (\bigcap (F ' K)) = \{\}$ 
    by (metis finite_subset_image compact_imp_fip [OF compact_interval
Fclo])
  then have K ≠  $\{\}$ 
    using ⟨w < z⟩ ⟨ $\{w..z\} \cap \bigcap (F ' K) = \{\}$ ⟩ by auto
  define n where n ≡ Max K
  have n ∈ K unfolding n_def by (metis ⟨K ≠  $\{\}$ ⟩ ⟨finite K⟩ Max_in)
  have F n ⊆  $\bigcap (F ' K)$ 
  unfolding n_def by (metis Fsub Max_ge ⟨K ≠  $\{\}$ ⟩ ⟨finite K⟩ cINF_greatest
lift_Suc_antimono_le)
  with K have wzF_null:  $\{w..z\} \cap F n = \{\}$ 
    by (metis disjoint_iff_not_equal subset_eq)
  obtain u where u: u ∈ F n u ∈ {x..w} ({u..w} - {u}) ∩ F n =  $\{\}$ 
  proof (cases w ∈ F n)
    case True
      then show ?thesis
        by (metis wzF_null ⟨w < z⟩ atLeastAtMost_iff disjoint_iff_not_equal
less_eq_real_def)
    next
      case False
        obtain u where u ∈ F n u ∈ {x..w} {u<..\{\}
        proof (rule segment_to_point_exists [of F n ∩ {x..w} w])
          show closed (F n ∩ {x..w})
            by (metis ⟨ $\bigwedge n. \text{closed } (F n)$ ⟩ closed_Int closed_real_atLeastAtMost)
          show F n ∩ {x..w} ≠  $\{\}$ 
            by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.prem1) less_eq_real_def w)
        qed (auto simp: open_segment_eq_real_ivl intro!: that)
        with False show thesis
          by (auto simp add: disjoint_iff less_eq_real_def intro!: that)
      qed
  obtain v where v: v ∈ F n v ∈ {z..y} ({z..v} - {v}) ∩ F n =  $\{\}$ 
  proof (cases z ∈ F n)
    case True
      have z ∈ {w..z}
        using ⟨w < z⟩ by auto
      then show ?thesis
        by (metis wzF_null Int_iff True empty_iff)
    next

```

```

case False
show ?thesis
proof (rule segment_to_point_exists [of F n  $\cap$  {z..y} z])
  show closed (F n  $\cap$  {z..y})
    by (metis  $\langle \wedge n. \text{closed } (F n) \rangle \text{ closed\_Int closed\_atLeastAtMost}$ )
  show F n  $\cap$  {z..y}  $\neq$  {}
  by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.premis(2) less_eq_real_def z)
  show  $\wedge b. \llbracket b \in F n \cap \{z..y\}; \text{open\_segment } z b \cap (F n \cap \{z..y\}) = \{\} \rrbracket$ 
 $\implies$  thesis
  proof
    show  $\wedge b. \llbracket b \in F n \cap \{z..y\}; \text{open\_segment } z b \cap (F n \cap \{z..y\}) = \{\} \rrbracket$ 
 $\implies (\{z..b\} - \{b\}) \cap F n = \{\}$ 
    using False by (auto simp: open_segment_eq_real_ivl less_eq_real_def)
  qed auto
qed
qed
obtain u v where  $u \in \{0..1\}$   $v \in \{0..1\}$   $\text{norm}(u - x) < e$   $\text{norm}(v - y)$ 
 $< e$   $p u = p v$ 
proof
  show  $u \in \{0..1\}$   $v \in \{0..1\}$ 
    by (metis F01  $\langle u \in F n \rangle \langle v \in F n \rangle \text{subsetD}$ ) $+$ 
  show  $\text{norm}(u - x) < e$   $\text{norm}(v - y) < e$ 
    using  $\langle u \in \{x..w\} \rangle \langle v \in \{z..y\} \rangle$  atLeastAtMost_iff real_norm_def wx
z y by auto
  show  $p u = p v$ 
proof (rule peq)
    show  $u \in F n$   $v \in F n$ 
      by (auto simp: u v)
    have False if  $\xi \in F n$   $u < \xi$   $\xi < v$  for  $\xi$ 
proof  $-$ 
      have  $\xi \notin \{z..v\}$ 
      by (metis DiffI disjoint_iff_not_equal less_irrefl singletonD that(1,3))
v(3)
      moreover have  $\xi \notin \{w..z\} \cap F n$ 
        by (metis equalsOD wzF_null)
      ultimately have  $\xi \in \{u..w\}$ 
        using that by auto
      then show ?thesis
        by (metis DiffI disjoint_iff_not_equal less_eq_real_def not_le
singletonD that(1,2) u(3))
    qed
    moreover
      have  $\llbracket \xi \in F n; v < \xi; \xi < u \rrbracket \implies \text{False}$  for  $\xi$ 
        using  $\langle u \in \{x..w\} \rangle \langle v \in \{z..y\} \rangle \langle w < z \rangle$  by simp
      ultimately
        show  $\text{open\_segment } u v \cap F n = \{\}$ 
          by (force simp: open_segment_eq_real_ivl)
    qed

```

```

qed
then show ?case
  using e [of x u] e [of y v] xy
  by (metis dist_norm dist_triangle_half_r order_less_irrefl)
qed (auto simp: open_segment_commute)
show ?thesis
  unfolding  $\varphi\_def$  by (metis (no_types, opaque_lifting) INT_I Inf_lower2
rangeI that(3) F01 subsetCE pqF)
qed
show closed {0..1::real} by auto
qed (meson  $\varphi\_def$ )
then have  $T \subseteq \{0..1\}$   $0 \in T$   $1 \in T$ 
  and  $peq: \bigwedge x y. \llbracket x \in T; y \in T; open\_segment\ x\ y \cap T = \{\} \rrbracket \implies p\ x = p\ y$ 
  unfolding  $\varphi\_def$  by metis+
then have  $T \neq \{\}$  by auto
define h where  $h \equiv \lambda x. p(SOME\ y. y \in T \wedge open\_segment\ x\ y \cap T = \{\})$ 
have  $p\ y = p\ z$  if  $y \in T$   $z \in T$  and  $xyT: open\_segment\ x\ y \cap T = \{\}$  and  $xzT: open\_segment\ x\ z \cap T = \{\}$ 
  for  $x\ y\ z$ 
proof (cases  $x \in T$ )
  case True
  with that show ?thesis by (metis  $\langle \varphi\ T \rangle \varphi\_def$ )
next
  case False
  have  $insert\ x\ (open\_segment\ x\ y \cup open\_segment\ x\ z) \cap T = \{\}$ 
    by (metis False Int_Un_distrib2 Int_insert_left Un_empty_right xyT xzT)
  moreover have  $open\_segment\ y\ z \cap T \subseteq insert\ x\ (open\_segment\ x\ y \cup open\_segment\ x\ z) \cap T$ 
    by (auto simp: open_segment_eq_real_ivl)
  ultimately have  $open\_segment\ y\ z \cap T = \{\}$ 
    by blast
  with that  $peq$  show ?thesis by metis
qed
then have  $h\_eq\_p\_gen: h\ x = p\ y$  if  $y \in T$   $open\_segment\ x\ y \cap T = \{\}$  for  $x$ 
  y
  using that unfolding  $h\_def$ 
  by (metis (mono_tags, lifting) some_eq_ex)
then have  $h\_eq\_p: \bigwedge x. x \in T \implies h\ x = p\ x$ 
  by simp
have disjoint:  $\bigwedge x. \exists y. y \in T \wedge open\_segment\ x\ y \cap T = \{\}$ 
  by (meson  $\langle T \neq \{\} \rangle \langle closed\ T \rangle segment\_to\_point\_exists$ )
have  $heq: h\ x = h\ x'$  if  $open\_segment\ x\ x' \cap T = \{\}$  for  $x\ x'$ 
proof (cases  $x \in T \vee x' \in T$ )
  case True
  then show ?thesis
    by (metis  $h\_eq\_p\ h\_eq\_p\_gen\ open\_segment\_commute\ that$ )
next
  case False
  obtain  $y\ y'$  where  $y \in T$   $open\_segment\ x\ y \cap T = \{\}$   $h\ x = p\ y$ 

```



```

     $y' \in T$   $\text{open\_segment } x' y' \cap T = \{\}$   $h x' = p y'$ 
    by (meson disjoint h_eq_p_gen)
    moreover have  $\text{open\_segment } y y' \subseteq (\text{insert } x (\text{insert } x' (\text{open\_segment } x y \cup \text{open\_segment } x' y' \cup \text{open\_segment } x x^{\wedge})))$ 
    by (auto simp: open_segment_eq_real_ivl)
    ultimately show ?thesis
    using False that by (fastforce simp add: h_eq_p intro!: peq)
  qed
  have  $h \text{ ' } \{0..1\}$  homeomorphic  $\{0..1::\text{real}\}$ 
  proof (rule homeomorphic_monotone_image_interval)
    show continuous_on  $\{0..1\}$  h
    proof (clarsimp simp add: continuous_on_iff)
      fix  $u \varepsilon::\text{real}$ 
      assume  $0 < \varepsilon$   $0 \leq u$   $u \leq 1$ 
      then obtain  $\delta$  where  $\delta > 0$  and  $\delta: \bigwedge v. v \in \{0..1\} \implies \text{dist } v u < \delta \longrightarrow$ 
 $\text{dist } (p v) (p u) < \varepsilon / 2$ 
      using ucont_p [unfolded uniformly_continuous_on_def]
      by (metis atLeastAtMost_iff half_gt_zero_iff)
      then have  $\text{dist } (h v) (h u) < \varepsilon$  if  $v \in \{0..1\}$   $\text{dist } v u < \delta$  for  $v$ 
      proof (cases open_segment u v  $\cap T = \{\}$ )
        case True
        then show ?thesis
        using  $\langle 0 < \varepsilon \rangle$  heq by auto
      next
        case False
        have  $w \in T: \text{closed } (\text{closed\_segment } u v \cap T) \text{ closed\_segment } u v \cap T \neq \{\}$ 
        using False open_closed_segment by (auto simp:  $\langle \text{closed } T \rangle$  closed_Int)
        obtain  $w$  where  $w \in T$  and  $w: w \in \text{closed\_segment } u v \text{ open\_segment } u v \cap T = \{\}$ 
        proof (rule segment_to_point_exists [OF  $w \in T$ ])
          fix  $b$ 
          assume  $b \in \text{closed\_segment } u v \cap T$   $\text{open\_segment } u b \cap (\text{closed\_segment } u v \cap T) = \{\}$ 
          then show thesis
          by (metis IntD1 IntD2 ends_in_segment(1) inf.orderE inf_assoc subset_oc_segment that)
        qed
        then have  $p u w: \text{dist } (p u) (p w) < \varepsilon / 2$ 
        by (metis (no_types)  $\langle T \subseteq \{0..1\} \rangle \langle \text{dist } v u < \delta \rangle \delta$  dist_commute dist_in_closed_segment le_less_trans subsetCE)
        obtain  $z$  where  $z \in T$  and  $z: z \in \text{closed\_segment } u v \text{ open\_segment } v z \cap T = \{\}$ 
        proof (rule segment_to_point_exists [OF  $w \in T$ ])
          fix  $b$ 
          assume  $b \in \text{closed\_segment } u v \cap T$   $\text{open\_segment } v b \cap (\text{closed\_segment } u v \cap T) = \{\}$ 
          then show thesis
          by (metis IntD1 IntD2 ends_in_segment(2) inf.orderE inf_assoc subset_oc_segment that)
        qed
      end
    end
  end

```

```

qed
then have  $\text{dist } (p \ u) \ (p \ z) < \varepsilon / 2$ 
by (metis  $\langle T \subseteq \{0..1\} \rangle \langle \text{dist } v \ u < \delta \rangle \delta \ \text{dist\_commute} \ \text{dist\_in\_closed\_segment}$ 
le_less_trans subsetCE)
then show ?thesis
using puw by (metis (no_types)  $\langle w \in T \rangle \langle z \in T \rangle \ \text{dist\_commute}$ 
 $\text{dist\_triangle\_half\_1} \ h\_eq\_p\_gen \ w(2) \ z(2)$ )
qed
with  $\langle 0 < \delta \rangle$  show  $\exists \delta > 0. \forall v \in \{0..1\}. \text{dist } v \ u < \delta \longrightarrow \text{dist } (h \ v) \ (h \ u) < \varepsilon$ 
by blast
qed
show connected ( $\{0..1\} \cap h \ -' \ \{z\}$ ) for z
proof (clarsimp simp add: connected_iff_connected_component)
fix u v
assume huv_eq:  $h \ v = h \ u$  and uv:  $0 \leq u \ u \leq 1 \ 0 \leq v \ v \leq 1$ 
have  $\exists T. \text{connected } T \wedge T \subseteq \{0..1\} \wedge T \subseteq h \ -' \ \{h \ u\} \wedge u \in T \wedge v \in T$ 
proof (intro exI conjI)
show connected (closed_segment u v)
by simp
show closed_segment u v  $\subseteq \{0..1\}$ 
by (simp add: uv closed_segment_eq_real_ivl)
have pxy:  $p \ x = p \ y$ 
if  $T \subseteq \{0..1\} \ 0 \in T \ 1 \in T \ x \in T \ y \in T$ 
and disjT:  $\text{open\_segment } x \ y \cap (T - \text{open\_segment } u \ v) = \{\}$ 
and xynot:  $x \notin \text{open\_segment } u \ v \ y \notin \text{open\_segment } u \ v$ 
for x y
proof (cases open_segment x y \cap open_segment u v = \{\})
case True
then show ?thesis
by (metis Diff_Int_distrib Diff_empty peq disjT  $\langle x \in T \rangle \langle y \in T \rangle$ )
next
case False
then have  $\text{open\_segment } x \ u \cup \text{open\_segment } y \ v \subseteq \text{open\_segment } x \ y$ 
 $- \text{open\_segment } u \ v \vee$ 
 $\text{open\_segment } y \ u \cup \text{open\_segment } x \ v \subseteq \text{open\_segment } x \ y -$ 
 $\text{open\_segment } u \ v$  (is ?xuyv \vee ?yuxv)
using xynot by (fastforce simp add: open_segment_eq_real_ivl not_le
not_less split: if_split_asm)
then show  $p \ x = p \ y$ 
proof
assume ?xuyv
then have  $\text{open\_segment } x \ u \cap T = \{\} \ \text{open\_segment } y \ v \cap T = \{\}$ 
using disjT by auto
then have  $h \ x = h \ y$ 
using heq huv_eq by auto
then show ?thesis
using h_eq_p  $\langle x \in T \rangle \langle y \in T \rangle$  by auto
next
assume ?yuxv

```

```

    then have open_segment  $y u \cap T = \{\}$  open_segment  $x v \cap T = \{\}$ 
      using disjT by auto
    then have  $h x = h y$ 
      using heq [of  $y u$ ] heq [of  $x v$ ] huv_eq by auto
    then show ?thesis
      using h_eq_p  $\langle x \in T \rangle \langle y \in T \rangle$  by auto
  qed
qed
have  $\neg T - \text{open\_segment } u v \subset T$ 
proof (rule T)
  show closed  $(T - \text{open\_segment } u v)$ 
    by (simp add: closed_Diff [OF  $\langle \text{closed } T \rangle$ ] open_segment_eq_real_ivl)
  have  $0 \notin \text{open\_segment } u v$   $1 \notin \text{open\_segment } u v$ 
    using open_segment_eq_real_ivl uv by auto
  then show  $\varphi (T - \text{open\_segment } u v)$ 
    using  $\langle T \subseteq \{0..1\} \rangle \langle 0 \in T \rangle \langle 1 \in T \rangle$ 
    by (auto simp:  $\varphi\_def$ ) (meson peq pxy)
  qed
  then have open_segment  $u v \cap T = \{\}$ 
    by blast
  then show closed_segment  $u v \subseteq h - \{h u\}$ 
    by (force intro: heq simp: open_segment_eq_real_ivl closed_segment_eq_real_ivl
split: if_split_asm)+
  qed auto
  then show connected_component  $(\{0..1\} \cap h - \{h u\}) u v$ 
    by (simp add: connected_component_def)
  qed
  show  $h 1 \neq h 0$ 
    by (metis  $\langle \varphi T \rangle \varphi\_def a \langle a \neq b \rangle b h\_eq\_p \text{pathfinish\_def } \text{pathstart\_def}$ )
  qed
  then obtain  $f$  and  $g :: \text{real} \Rightarrow 'a$ 
    where gfeq:  $(\forall x \in h - \{0..1\}. (g(f x) = x))$  and fhim:  $f - h - \{0..1\} = \{0..1\}$ 
  and contf: continuous_on  $(h - \{0..1\}) f$ 
    and fgeq:  $(\forall y \in \{0..1\}. (f(g y) = y))$  and pag: path_image  $g = h - \{0..1\}$ 
  and contg: continuous_on  $\{0..1\} g$ 
    by (auto simp: homeomorphic_def homeomorphism_def path_image_def)
  then have arc  $g$ 
    by (metis arc_def path_def inj_on_def)
  obtain  $u v$  where  $u \in \{0..1\}$   $a = g u$   $v \in \{0..1\}$   $b = g v$ 
    by (metis (mono_tags, opaque_lifting)  $\langle \varphi T \rangle \varphi\_def a b fhim gfeq h\_eq\_p \text{imageI } \text{path\_image\_def } \text{pathfinish\_def } \text{pathfinish\_in\_path\_image } \text{pathstart\_def } \text{pathstart\_in\_path\_image}$ )
  then have  $a \in \text{path\_image } g$   $b \in \text{path\_image } g$ 
    using path_image_def by blast+
  have  $ph: \text{path\_image } h \subseteq \text{path\_image } p$ 
    by (metis image_mono image_subset_iff path_image_def disjoint h_eq_p_gen
 $\langle T \subseteq \{0..1\} \rangle$ )
  show ?thesis
  proof

```

```

show pathstart (subpath u v g) = a pathfinish (subpath u v g) = b
  by (simp_all add: ⟨a = g u⟩ ⟨b = g v⟩)
show path_image (subpath u v g) ⊆ path_image p
  by (metis ⟨u ∈ {0..1}⟩ ⟨v ∈ {0..1}⟩ order_trans pag path_image_def
path_image_subpath_subset ph)
show arc (subpath u v g)
  using ⟨arc g⟩ ⟨a = g u⟩ ⟨b = g v⟩ ⟨u ∈ {0..1}⟩ ⟨v ∈ {0..1}⟩ arc_subpath_arc
⟨a ≠ b⟩ by blast
qed
qed

```

corollary *path_connected_arcwise*:

```

fixes S :: 'a::{complete_space,real_normed_vector} set
shows path_connected S ↔
  (∀ x ∈ S. ∀ y ∈ S. x ≠ y → (∃ g. arc g ∧ path_image g ⊆ S ∧ pathstart g
= x ∧ pathfinish g = y))
  (is ?lhs = ?rhs)
proof (intro iffI impI ballI)
  fix x y
  assume path_connected S x ∈ S y ∈ S x ≠ y
  then obtain p where p: path p path_image p ⊆ S pathstart p = x pathfinish p
= y
  by (force simp: path_connected_def)
  then show ∃ g. arc g ∧ path_image g ⊆ S ∧ pathstart g = x ∧ pathfinish g = y
  by (metis ⟨x ≠ y⟩ order_trans path_contains_arc)
next
assume R [rule_format]: ?rhs
show ?lhs
  unfolding path_connected_def
proof (intro ballI)
  fix x y
  assume x ∈ S y ∈ S
  show ∃ g. path g ∧ path_image g ⊆ S ∧ pathstart g = x ∧ pathfinish g = y
  proof (cases x = y)
    case True with ⟨x ∈ S⟩ path_component_def path_component_refl show
?thesis
    by blast
  next
  case False with R [OF ⟨x ∈ S⟩ ⟨y ∈ S⟩] show ?thesis
  by (auto intro: arc_imp_path)
qed
qed
qed

```

corollary *arc_connected_trans*:

```

fixes g :: real ⇒ 'a::{complete_space,real_normed_vector}
assumes arc g arc h pathfinish g = pathstart h pathstart g ≠ pathfinish h

```

obtains i **where** $\text{arc } i \text{ path_image } i \subseteq \text{path_image } g \cup \text{path_image } h$
 $\text{pathstart } i = \text{pathstart } g \text{ pathfinish } i = \text{pathfinish } h$
by (*metis* (*no_types*, *opaque_lifting*) *arc_imp_path* *assms* *path_contains_arc*
path_image_join *path_join* *pathfinish_join* *pathstart_join*)

6.10.4 Accessibility of frontier points

lemma *dense_accessible_frontier_points*:

fixes $S :: 'a::\{\text{complete_space, real_normed_vector}\} \text{ set}$
assumes *open* S **and** *opeSV*: *openin* (*top_of_set* (*frontier* S)) V **and** $V \neq \{\}$
obtains g **where** $\text{arc } g \text{ } \{0..<1\} \subseteq S \text{ pathstart } g \in S \text{ pathfinish } g \in V$
proof –
obtain z **where** $z \in V$
using $\langle V \neq \{\} \rangle$ **by** *auto*
then obtain r **where** $r > 0$ **and** r : $\text{ball } z \ r \cap \text{frontier } S \subseteq V$
by (*metis* *openin_contains_ball* *opeSV*)
then have $z \in \text{frontier } S$
using $\langle z \in V \rangle$ *opeSV* *openin_contains_ball* **by** *blast*
then have $z \in \text{closure } S \ z \notin S$
by (*simp* *add*: *frontier_def* *assms* *interior_open*)
with $\langle r > 0 \rangle$ **have** *infinite* ($S \cap \text{ball } z \ r$)
by (*auto* *simp*: *closure_def* *islimpt_eq_infinite_ball*)
then obtain y **where** $y \in S$ **and** y : $y \in \text{ball } z \ r$
using *infinite_imp_nonempty* **by** *force*
then have $y \notin \text{frontier } S$
by (*meson* $\langle \text{open } S \rangle$ *disjoint_iff_not_equal* *frontier_disjoint_eq*)
have $y \neq z$
using $\langle y \in S \rangle \langle z \notin S \rangle$ **by** *blast*
have *path_connected*($\text{ball } z \ r$)
by (*simp* *add*: *convex_imp_path_connected*)
with $y \langle r > 0 \rangle$ **obtain** g **where** *arc* g **and** pig : $\text{path_image } g \subseteq \text{ball } z \ r$
and g : $\text{pathstart } g = y \text{ pathfinish } g = z$
using $\langle y \neq z \rangle$ **by** (*force* *simp*: *path_connected_arcwise*)
have *continuous_on* $\{0..1\}$ g
using $\langle \text{arc } g \rangle$ *arc_imp_path* *path_def* **by** *blast*
then have *compact* ($g - \text{frontier } S \cap \{0..1\}$)
by (*simp* *add*: *bounded_Int* *closed_Diff* *closed_vimage_Int* *compact_eq_bounded_closed*)
moreover have $g - \text{frontier } S \cap \{0..1\} \neq \{\}$
proof –
have $\exists r. r \in g - \text{frontier } S \wedge r \in \{0..1\}$
by (*metis* $\langle z \in \text{frontier } S \rangle$ *g(2)* *imageE* *path_image_def* *pathfinish_in_path_image*
vimageI2)
then show *?thesis*
by *blast*
qed
ultimately obtain t **where** gt : $g \ t \in \text{frontier } S$ **and** $0 \leq t \leq 1$
and t : $\bigwedge u. \llbracket g \ u \in \text{frontier } S; 0 \leq u; u \leq 1 \rrbracket \implies t \leq u$
by (*force* *simp*: *dest!*: *compact_attains_inf*)
moreover have $t \neq 0$

```

    by (metis ⟨y ∉ frontier S⟩ g(1) gt pathstart_def)
  ultimately have t01: 0 < t ≤ 1
    by auto
  have V ⊆ frontier S
    using opeSV openin_contains_ball by blast
  show ?thesis
  proof
    show arc (subpath 0 t g)
      by (simp add: ⟨0 ≤ t⟩ ⟨t ≤ 1⟩ ⟨arc g⟩ ⟨t ≠ 0⟩ arc_subpath_arc)
    have g 0 ∈ S
      by (metis ⟨y ∈ S⟩ g(1) pathstart_def)
    then show pathstart (subpath 0 t g) ∈ S
      by auto
    have g t ∈ V
      by (metis IntI atLeastAtMost_iff gt image_eqI path_image_def pig r subsetCE
    ⟨0 ≤ t⟩ ⟨t ≤ 1⟩)
    then show pathfinish (subpath 0 t g) ∈ V
      by auto
    then have inj_on (subpath 0 t g) {0..1}
      using t01 ⟨arc (subpath 0 t g)⟩ arc_imp_inj_on by blast
    then have subpath 0 t g ‘ {0..<1} ⊆ subpath 0 t g ‘ {0..1} - {subpath 0 t g
    1}
      by (force simp: dest: inj_onD)
    moreover have False if subpath 0 t g ‘ ({0..<1}) - S ≠ {}
    proof -
      have contg: continuous_on {0..1} g
        using ⟨arc g⟩ by (auto simp: arc_def path_def)
      have subpath 0 t g ‘ {0..<1} ∩ frontier S ≠ {}
        proof (rule connected_Int_frontier [OF _ _ that])
          show connected (subpath 0 t g ‘ {0..<1})
            proof (rule connected_continuous_image)
              show continuous_on {0..<1} (subpath 0 t g)
                by (meson ⟨arc (subpath 0 t g)⟩ arc_def atLeastLessThan_subseteq_atLeastAtMost_iff
              continuous_on_subset order_refl path_def)
            qed auto
          show subpath 0 t g ‘ {0..<1} ∩ S ≠ {}
            using ⟨y ∈ S⟩ g(1) by (force simp: subpath_def image_def pathstart_def)
          qed
        then obtain x where x ∈ subpath 0 t g ‘ {0..<1} x ∈ frontier S
          by blast
        with t01 ⟨0 ≤ t⟩ mult_le_one t show False
          by (fastforce simp: subpath_def)
        qed
    then have subpath 0 t g ‘ {0..1} - {subpath 0 t g 1} ⊆ S
      using subsetD by fastforce
    ultimately show subpath 0 t g ‘ {0..<1} ⊆ S
      by auto
    qed
  qed

```

```

lemma dense_accessible_frontier_points_connected:
  fixes S :: 'a::{complete_space,real_normed_vector} set
  assumes open S connected S x ∈ S V ≠ {}
    and ope: openin (top_of_set (frontier S)) V
  obtains g where arc g g ' {0..<1} ⊆ S pathstart g = x pathfinish g ∈ V
proof -
  have V ⊆ frontier S
    using ope openin_imp_subset by blast
  with ⟨open S⟩ ⟨x ∈ S⟩ have x ∉ V
    using interior_open by (auto simp: frontier_def)
  obtain g where arc g and g: g ' {0..<1} ⊆ S pathstart g ∈ S pathfinish g ∈ V
    by (metis dense_accessible_frontier_points [OF ⟨open S⟩ ope ⟨V ≠ {}⟩])
  then have path_connected S
    by (simp add: assms connected_open_path_connected)
  with ⟨pathstart g ∈ S⟩ ⟨x ∈ S⟩ have path_component S x (pathstart g)
    by (simp add: path_connected_component)
  then obtain f where path f and f: path_image f ⊆ S pathstart f = x pathfinish
f = pathstart g
    by (auto simp: path_component_def)
  then have path (f +++ g)
    by (simp add: ⟨arc g⟩ arc_imp_path)
  then obtain h where arc h
    and h: path_image h ⊆ path_image (f +++ g) pathstart h = x
pathfinish h = pathfinish g
    using path_contains_arc [of f +++ g x pathfinish g] ⟨x ∉ V⟩ ⟨pathfinish g ∈
V⟩ f
    by (metis pathfinish_join pathstart_join)
  have path_image h ⊆ path_image f ∪ path_image g
    using h(1) path_image_join_subset by auto
  then have h ' {0..1} - {h 1} ⊆ S
    using f g h
    apply (simp add: path_image_def pathfinish_def subset_iff image_def Bex_def)
    by (metis le_less)
  then have h ' {0..<1} ⊆ S
    using ⟨arc h⟩ by (force simp: arc_def dest: inj_onD)
  then show thesis
    using ⟨arc h⟩ g(3) h that by presburger
qed

```

```

lemma dense_access_fp_aux:
  fixes S :: 'a::{complete_space,real_normed_vector} set
  assumes S: open S connected S
    and opeSU: openin (top_of_set (frontier S)) U
    and opeSV: openin (top_of_set (frontier S)) V
    and V ≠ {} ¬ U ⊆ V
  obtains g where arc g pathstart g ∈ U pathfinish g ∈ V g ' {0<..

```

```

have S ≠ {}
  using opeSV ⟨V ≠ {}⟩ by (metis frontier_empty openin_subtopology_empty)
then obtain x where x ∈ S by auto
obtain g where arc g and g: g ‘ {0..<1} ⊆ S pathstart g = x pathfinish g ∈ V
  using dense_accessible_frontier_points_connected [OF S ⟨x ∈ S⟩ ⟨V ≠ {}⟩
opeSV] by blast
obtain h where arc h and h: h ‘ {0..<1} ⊆ S pathstart h = x pathfinish h ∈ U
- {pathfinish g}
proof (rule dense_accessible_frontier_points_connected [OF S ⟨x ∈ S⟩])
  show U - {pathfinish g} ≠ {}
    using ⟨pathfinish g ∈ V⟩ ⟨¬ U ⊆ V⟩ by blast
  show openin (top_of_set (frontier S)) (U - {pathfinish g})
    by (simp add: opeSU openin_delete)
qed auto
obtain γ where arc γ
  and γ: path_image γ ⊆ path_image (reversepath h +++ g)
  pathstart γ = pathfinish h pathfinish γ = pathfinish g
proof (rule path_contains_arc [of (reversepath h +++ g) pathfinish h pathfinish
g])
  show path (reversepath h +++ g)
    by (simp add: ⟨arc g⟩ ⟨arc h⟩ ⟨pathstart g = x⟩ ⟨pathstart h = x⟩ arc_imp_path)
  show pathstart (reversepath h +++ g) = pathfinish h
    pathfinish (reversepath h +++ g) = pathfinish g
    by auto
  show pathfinish h ≠ pathfinish g
    using ⟨pathfinish h ∈ U - {pathfinish g}⟩ by auto
qed auto
show ?thesis
proof
  show arc γ pathstart γ ∈ U pathfinish γ ∈ V
    using γ ⟨arc γ⟩ ⟨pathfinish h ∈ U - {pathfinish g}⟩ ⟨pathfinish g ∈ V⟩ by
auto
  have path_image γ ⊆ path_image h ∪ path_image g
    by (metis γ(1) g(2) h(2) path_image_join path_image_reversepath pathfin-
ish_reversepath)
  then have γ ‘ {0..1} - {γ 0, γ 1} ⊆ S
    using γ g h
    apply (simp add: path_image_def pathstart_def pathfinish_def subset_iff
image_def Bex_def)
    by (metis linorder_neqE_linordered_idom not_less)
  then show γ ‘ {0<..<1} ⊆ S
    using ⟨arc h⟩ ⟨arc γ⟩
    by (metis arc_imp_simple_path path_image_def pathfinish_def pathstart_def
simple_path_endless)
qed
qed

```

lemma dense_accessible_frontier_point_pairs:

fixes S :: 'a::{complete_space,real_normed_vector} set


```

assumes S: open S connected S
  and opeSU: openin (top_of_set (frontier S)) U
  and opeSV: openin (top_of_set (frontier S)) V
  and  $U \neq \{\}$   $V \neq \{\}$   $U \neq V$ 
  obtains g where arc g pathstart  $g \in U$  pathfinish  $g \in V$   $g \text{ ' } \{0 < .. < 1\} \subseteq S$ 
proof -
  consider  $\neg U \subseteq V \mid \neg V \subseteq U$ 
  using  $\langle U \neq V \rangle$  by blast
  then show ?thesis
  proof cases
    case 1 then show ?thesis
      using assms dense_access_fp_aux [OF S opeSU opeSV] that by blast
    next
    case 2
      obtain g where arc g and g: pathstart  $g \in V$  pathfinish  $g \in U$   $g \text{ ' } \{0 < .. < 1\} \subseteq S$ 
      using assms dense_access_fp_aux [OF S opeSV opeSU] 2 by blast
      show ?thesis
      proof
        show arc (reversepath g)
          by (simp add:  $\langle \text{arc } g \rangle$  arc_reversepath)
        show pathstart (reversepath g)  $\in U$  pathfinish (reversepath g)  $\in V$ 
          using g by auto
        show reversepath  $g \text{ ' } \{0 < .. < 1\} \subseteq S$ 
          using g by (auto simp: reversepath_def)
      qed
    qed
  qed
end

```

6.11 The Urysohn lemma, its consequences and other advanced material about metric spaces

```

theory Urysohn
imports Abstract_Topological_Spaces Abstract_Metric_Spaces Infinite_Sum Ar-
cwise_Connected
begin

```

6.11.1 Urysohn lemma and Tietze's theorem

```

proposition Urysohn_lemma:
  fixes a b :: real
  assumes normal_space X closedin X S closedin X T disjnt S T  $a \leq b$ 
  obtains f where continuous_map X (top_of_set {a..b}) f  $f \text{ ' } S \subseteq \{a\}$   $f \text{ ' } T \subseteq \{b\}$ 
proof -
  obtain U where openin X U  $S \subseteq U$   $X \text{ closure\_of } U \subseteq \text{topspace } X - T$ 

```

```

using assms unfolding normal_space_alt disjnt_def
by (metis Diff_mono Un_Diff_Int closedin_def subset_eq sup_bot_right)
have  $\exists G :: \text{real} \Rightarrow 'a \text{ set. } G\ 0 = U \wedge G\ 1 = \text{topspace } X - T \wedge$ 
      ( $\forall x \in \text{dyadics} \cap \{0..1\}. \forall y \in \text{dyadics} \cap \{0..1\}. x < y \longrightarrow \text{openin } X$ 
      ( $G\ x) \wedge \text{openin } X\ (G\ y) \wedge X\ \text{closure\_of}\ (G\ x) \subseteq G\ y$ )
proof (rule recursion_on_dyadic_fractions)
  show  $\text{openin } X\ U \wedge \text{openin } X\ (\text{topspace } X - T) \wedge X\ \text{closure\_of}\ U \subseteq \text{topspace}$ 
   $X - T$ 
  using  $\langle X\ \text{closure\_of}\ U \subseteq \text{topspace } X - T \rangle \langle \text{openin } X\ U \rangle \langle \text{closedin } X\ T \rangle$  by
  blast
  show  $\exists z. (\text{openin } X\ x \wedge \text{openin } X\ z \wedge X\ \text{closure\_of}\ x \subseteq z) \wedge \text{openin } X\ z \wedge$ 
   $\text{openin } X\ y \wedge X\ \text{closure\_of}\ z \subseteq y$ 
  if  $\text{openin } X\ x \wedge \text{openin } X\ y \wedge X\ \text{closure\_of}\ x \subseteq y$  for  $x\ y$ 
  by (meson that closedin_closure_of_normal_space_alt  $\langle \text{normal\_space } X \rangle$ )
  show  $\text{openin } X\ x \wedge \text{openin } X\ z \wedge X\ \text{closure\_of}\ x \subseteq z$ 
  if  $\text{openin } X\ x \wedge \text{openin } X\ y \wedge X\ \text{closure\_of}\ x \subseteq y$  and  $\text{openin } X\ y \wedge \text{openin}$ 
   $X\ z \wedge X\ \text{closure\_of}\ y \subseteq z$  for  $x\ y\ z$ 
  by (meson that closure_of_subset openin_subset subset_trans)
qed
then obtain  $G :: \text{real} \Rightarrow 'a \text{ set}$ 
  where  $G0: G\ 0 = U$  and  $G1: G\ 1 = \text{topspace } X - T$ 
  and  $G: \bigwedge x\ y. \llbracket x \in \text{dyadics}; y \in \text{dyadics}; 0 \leq x; x < y; y \leq 1 \rrbracket$ 
   $\implies \text{openin } X\ (G\ x) \wedge \text{openin } X\ (G\ y) \wedge X\ \text{closure\_of}\ (G\ x) \subseteq$ 
   $G\ y$ 
  by (smt (verit, del_insts) Int_iff atLeastAtMost_iff)
define  $f$  where  $f \equiv \lambda x. \text{Inf}(\text{insert } 1\ \{r. r \in \text{dyadics} \cap \{0..1\} \wedge x \in G\ r\})$ 
have  $f\_ge: f\ x \geq 0$  if  $x \in \text{topspace } X$  for  $x$ 
  unfolding  $f\_def$  by (force intro: cInf_greatest)
moreover have  $f\_le1: f\ x \leq 1$  if  $x \in \text{topspace } X$  for  $x$ 
proof -
  have  $\text{bdd\_below}\ \{r \in \text{dyadics} \cap \{0..1\}. x \in G\ r\}$ 
  by (auto simp: bdd_below_def)
  then show ?thesis
  by (auto simp: f_def cInf_lower)
qed
ultimately have  $\text{fim}: f\ ' \text{topspace } X \subseteq \{0..1\}$ 
  by (auto simp: f_def)
have  $0: 0 \in \text{dyadics} \cap \{0..1::\text{real}\}$  and  $1: 1 \in \text{dyadics} \cap \{0..1::\text{real}\}$ 
  by (force simp: dyadics_def)
then have  $\text{opeG}: \text{openin } X\ (G\ r)$  if  $r \in \text{dyadics} \cap \{0..1\}$  for  $r$ 
  using  $G\ G0 \langle \text{openin } X\ U \rangle \text{less\_eq\_real\_def}$  that by auto
have  $x \in G\ 0$  if  $x \in S$  for  $x$ 
  using  $G0 \langle S \subseteq U \rangle$  that by blast
with  $0$  have  $\text{fimS}: f\ ' S \subseteq \{0\}$ 
  unfolding  $f\_def$  by (force intro!: cInf_eq_minimum)
have False if  $r \in \text{dyadics } 0 \leq r\ r < 1\ x \in G\ r\ x \in T$  for  $r\ x$ 
  using  $G\ [\text{of } r\ 1]\ 1$ 
  by (smt (verit, best) DiffD2 G1 Int_iff closure_of_subset inf.orderE openin_subset
  that)

```

```

then have  $r \geq 1$  if  $r \in \text{dyadics}$   $0 \leq r$   $r \leq 1$   $x \in G$   $r$   $x \in T$  for  $r$   $x$ 
  using linorder_not_le that by blast
then have fmT:  $f \text{ ' } T \subseteq \{1\}$ 
  unfolding f_def by (force intro!: cInf_eq_minimum)
have fle1:  $f z \leq 1$  for  $z$ 
  by (force simp: f_def intro: cInf_lower)
have fle:  $f z \leq x$  if  $x \in \text{dyadics} \cap \{0..1\}$   $z \in G$   $x$  for  $z$   $x$ 
  using that by (force simp: f_def intro: cInf_lower)
have *:  $b \leq f z$  if  $b \leq 1 \wedge x. \llbracket x \in \text{dyadics} \cap \{0..1\}; z \in G x \rrbracket \implies b \leq x$  for  $z$   $b$ 
  using that by (force simp: f_def intro: cInf_greatest)
have **:  $r \leq f x$  if  $r: r \in \text{dyadics} \cap \{0..1\}$   $x \notin G$   $r$  for  $r$   $x$ 
proof (rule *)
  show  $r \leq s$  if  $s \in \text{dyadics} \cap \{0..1\}$  and  $x \in G$   $s$  for  $s :: \text{real}$ 
  using that  $r \in G$  [of s r] by (force simp: dest: closure_of_subset openin_subset)
qed (use that in force)

have  $\exists U. \text{openin } X$   $U \wedge x \in U \wedge (\forall y \in U. |f y - f x| < \varepsilon)$ 
if  $x \in \text{topspace } X$  and  $0 < \varepsilon$  for  $x$   $\varepsilon$ 
proof -
  have A:  $\exists r. r \in \text{dyadics} \cap \{0..1\} \wedge r < y \wedge |r - y| < d$  if  $0 < y$   $y \leq 1$   $0 < d$ 
for  $y$   $d :: \text{real}$ 
  proof -
    obtain  $n$   $q$   $r$ 
      where  $\text{of\_nat } q / 2^{\wedge n} < y$   $y < \text{of\_nat } r / 2^{\wedge n}$   $|q / 2^{\wedge n} - r / 2^{\wedge n}| < d$ 
      by (smt (verit, del_insts) padic_rational_approximation_straddle_pos  $\langle 0 < d \rangle$   $\langle 0 < y \rangle$ )
    then show ?thesis
      unfolding dyadics_def
      using divide_eq_0_iff that(2) by fastforce
  qed
  have B:  $\exists r. r \in \text{dyadics} \cap \{0..1\} \wedge y < r \wedge |r - y| < d$  if  $0 \leq y$   $y < 1$   $0 < d$ 
for  $y$   $d :: \text{real}$ 
  proof -
    obtain  $n$   $q$   $r$ 
      where  $\text{of\_nat } q / 2^{\wedge n} \leq y$   $y < \text{of\_nat } r / 2^{\wedge n}$   $|q / 2^{\wedge n} - r / 2^{\wedge n}| < d$ 
      using padic_rational_approximation_straddle_pos_le
      by (smt (verit, del_insts)  $\langle 0 < d \rangle$   $\langle 0 \leq y \rangle$ )
    then show ?thesis
      apply (clarsimp simp: dyadics_def)
      using divide_eq_0_iff  $\langle y < 1 \rangle$ 
      by (smt (verit) divide_nonneg_nonneg divide_self of_nat_0_le_iff of_nat_1 power_0_zero_le_power)
  qed
show ?thesis
proof (cases  $f x = 0$ )
  case True
    with  $B[0]$  obtain  $r$  where  $r: r \in \text{dyadics} \cap \{0..1\}$   $0 < r$   $|r| < \varepsilon/2$ 
      by (smt (verit)  $\langle 0 < \varepsilon \rangle$  half_gt_zero)
    show ?thesis

```

```

proof (intro exI conjI)
  show openin X (G r)
    using opeG r(1) by blast
  show x ∈ G r
    using True ** r by force
  show ∀ y ∈ G r. |f y - f x| < ε
    using f_ge ‹openin X (G r)› fle openin_subset r by (fastforce simp: True)
qed
next
case False
show ?thesis
proof (cases f x = 1)
  case True
    with A[of 1] obtain r where r: r ∈ dyadics ∩ {0..1} r < 1 |r-1| < ε/2
      by (smt (verit) ‹0 < ε› half_gt_zero)
    define G' where G' ≡ topspace X - X closure_of G r
    show ?thesis
    proof (intro exI conjI)
      show openin X G'
        unfolding G'_def by fastforce
      obtain r' where r' ∈ dyadics ∧ 0 ≤ r' ∧ r' ≤ 1 ∧ r < r' ∧ |r' - r| <
1 - r
        using B r by force
      moreover
        have 1 - r ∈ dyadics 0 ≤ r
          using 1 r dyadics_diff by force+
        ultimately have x ∉ X closure_of G r
          using G True r fle by force
        then show x ∈ G'
          by (simp add: G'_def that)
        show ∀ y ∈ G'. |f y - f x| < ε
          using ** f_le1 in_closure_of r by (fastforce simp: True G'_def)
      qed
    next
      case False
        have 0 < f x f x < 1
          using fle1 f_ge that(1) ‹f x ≠ 0› ‹f x ≠ 1› by (metis order_le_less) +
        obtain r where r: r ∈ dyadics ∩ {0..1} r < f x |r - f x| < ε / 2
          using A ‹0 < ε› ‹0 < f x› ‹f x < 1› by (smt (verit, best) half_gt_zero)
        obtain r' where r': r' ∈ dyadics ∩ {0..1} f x < r' |r' - f x| < ε / 2
          using B ‹0 < ε› ‹0 < f x› ‹f x < 1› by (smt (verit, best) half_gt_zero)
        have r < 1
          using ‹f x < 1› r(2) by force
        show ?thesis
        proof (intro conjI exI)
          show openin X (G r' - X closure_of G r)
            using closedin_closure_of opeG r' by blast
          have x ∈ X closure_of G r ⇒ False
            using B [of r f x - r] r ‹r < 1› G [of r] fle by force

```

```

    then show  $x \in G \text{ r}' - X \text{ closure\_of } G \text{ r}$ 
      using ** r' by fastforce
    show  $\forall y \in G \text{ r}' - X \text{ closure\_of } G \text{ r}. |f y - f x| < \varepsilon$ 
      using r r' ** G closure_of_subset field_sum_of_halves fle openin_subset
subset_eq
      by (smt (verit) DiffE opeG)
    qed
  qed
  qed
  then have contf: continuous_map X (top_of_set {0..1}) f
    by (force simp: Met_TC.continuous_map_to_metric dist_real_def continuous_map_in_subtopology fim simp flip: mtopology_is_euclidean)
  define g where  $g \equiv \lambda x. a + (b - a) * f x$ 
  show thesis
  proof
    have continuous_map X euclideanreal g
      using contf <a ≤ b> unfolding g_def by (auto simp: continuous_intros continuous_map_in_subtopology)
    moreover have  $g \text{ ' } (\text{topspace } X) \subseteq \{a..b\}$ 
      using mult_left_le [of f _ b-a] contf <a ≤ b>
      by (simp add: g_def add.commute continuous_map_in_subtopology image_subset_iff le_diff_eq)
    ultimately show continuous_map X (top_of_set {a..b}) g
      by (meson continuous_map_in_subtopology)
    show  $g \text{ ' } S \subseteq \{a\}$   $g \text{ ' } T \subseteq \{b\}$ 
      using fimS fimT by (auto simp: g_def)
  qed
qed

lemma Urysohn_lemma_alt:
  fixes a b :: real
  assumes normal_space X closedin X S closedin X T disjnt S T
  obtains f where continuous_map X euclideanreal f f ' S ⊆ {a} f ' T ⊆ {b}
  by (metis Urysohn_lemma assms continuous_map_in_subtopology disjnt_sym linear)

lemma normal_space_iff_Urysohn_gen_alt:
  assumes a ≠ b
  shows normal_space X ↔
    (∀ S T. closedin X S ∧ closedin X T ∧ disjnt S T
      → (∃ f. continuous_map X euclideanreal f ∧ f ' S ⊆ {a} ∧ f ' T ⊆ {b}))
  (is ?lhs=?rhs)
proof
  show ?lhs ⇒ ?rhs
    by (metis Urysohn_lemma_alt)
next
  assume R: ?rhs

```

```

show ?lhs
  unfolding normal_space_def
proof clarify
  fix S T
  assume closedin X S and closedin X T and disjnt S T
  with R obtain f where contf: continuous_map X euclideanreal f and f ' S
  ⊆ {a} f ' T ⊆ {b}
  by meson
  show ∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U V
proof (intro conjI exI)
  show openin X {x ∈ topspace X. f x ∈ ball a (|a - b| / 2)}
  by (force intro!: openin_continuous_map_preimage [OF contf])
  show openin X {x ∈ topspace X. f x ∈ ball b (|a - b| / 2)}
  by (force intro!: openin_continuous_map_preimage [OF contf])
  show S ⊆ {x ∈ topspace X. f x ∈ ball a (|a - b| / 2)}
  using ⟨closedin X S⟩ closedin_subset ⟨f ' S ⊆ {a}⟩ assms by force
  show T ⊆ {x ∈ topspace X. f x ∈ ball b (|a - b| / 2)}
  using ⟨closedin X T⟩ closedin_subset ⟨f ' T ⊆ {b}⟩ assms by force
  have ∧x. [x ∈ topspace X; dist a (f x) < |a-b|/2; dist b (f x) < |a-b|/2]
  ⇒ False
  by (smt (verit, best) dist_real_def dist_triangle_half_l)
  then show disjnt {x ∈ topspace X. f x ∈ ball a (|a-b| / 2)} {x ∈ topspace
  X. f x ∈ ball b (|a-b| / 2)}
  using disjnt_iff by fastforce
qed
qed
qed

```

lemma *normal_space_iff_Urysohn_gen*:

fixes $a b :: \text{real}$

shows

$a < b \implies$

$\text{normal_space } X \iff$

$(\forall S T. \text{closedin } X S \wedge \text{closedin } X T \wedge \text{disjnt } S T$

$\longrightarrow (\exists f. \text{continuous_map } X (\text{top_of_set } \{a..b\}) f \wedge$
 $f ' S \subseteq \{a\} \wedge f ' T \subseteq \{b\}))$

by (*metis linear not_le Urysohn_lemma normal_space_iff_Urysohn_gen_alt continuous_map_in_subtopology*)

lemma *normal_space_iff_Urysohn_alt*:

$\text{normal_space } X \iff$

$(\forall S T. \text{closedin } X S \wedge \text{closedin } X T \wedge \text{disjnt } S T$

$\longrightarrow (\exists f. \text{continuous_map } X \text{ euclideanreal } f \wedge$
 $f ' S \subseteq \{0\} \wedge f ' T \subseteq \{1\}))$

by (*rule normal_space_iff_Urysohn_gen_alt*) *auto*

lemma *normal_space_iff_Urysohn*:

$\text{normal_space } X \iff$

$(\forall S T. \text{closedin } X S \wedge \text{closedin } X T \wedge \text{disjnt } S T$

$$\longrightarrow (\exists f::'a \Rightarrow \text{real}. \text{continuous_map } X (\text{top_of_set } \{0..1\}) f \wedge f ' S \subseteq \{0\} \wedge f ' T \subseteq \{1\}))$$

by (rule normal_space_iff_Urysohn_gen) auto

lemma normal_space_perfect_map_image:

$\llbracket \text{normal_space } X; \text{perfect_map } X Y f \rrbracket \Longrightarrow \text{normal_space } Y$

unfolding perfect_map_def proper_map_def

using normal_space_continuous_closed_map_image **by** fastforce

lemma Hausdorff_normal_space_closed_continuous_map_image:

$\llbracket \text{normal_space } X; \text{closed_map } X Y f; \text{continuous_map } X Y f;$

$f ' \text{topspace } X = \text{topspace } Y; \text{t1_space } Y \rrbracket$

$\Longrightarrow \text{Hausdorff_space } Y$

by (metis normal_space_continuous_closed_map_image normal_t1_imp_Hausdorff_space)

lemma normal_Hausdorff_space_closed_continuous_map_image:

$\llbracket \text{normal_space } X; \text{Hausdorff_space } X; \text{closed_map } X Y f;$

$\text{continuous_map } X Y f; f ' \text{topspace } X = \text{topspace } Y \rrbracket$

$\Longrightarrow \text{normal_space } Y \wedge \text{Hausdorff_space } Y$

by (meson normal_space_continuous_closed_map_image normal_t1_eq_Hausdorff_space t1_space_closed_map_image)

lemma Lindelof_cover:

assumes regular_space X **and** Lindelof_space X **and** S ≠ {}

and clo: closedin X S closedin X T disjnt S T

obtains h :: nat ⇒ 'a set **where**

$\bigwedge n. \text{openin } X (h n) \wedge \bigwedge n. \text{disjnt } T (X \text{closure_of } (h n))$ **and** $S \subseteq \bigcup (\text{range } h)$

proof –

have $\exists U. \text{openin } X U \wedge x \in U \wedge \text{disjnt } T (X \text{closure_of } U)$

if $x \in S$ **for** x

using <regular_space X> **unfolding** regular_space

by (metis (full_types) Diff_iff <disjnt S T> clo_closure_of_eq disjnt_iff_in_closure_of that)

then obtain h **where** oh: $\bigwedge x. x \in S \Longrightarrow \text{openin } X (h x)$

and xh: $\bigwedge x. x \in S \Longrightarrow x \in h x$

and dh: $\bigwedge x. x \in S \Longrightarrow \text{disjnt } T (X \text{closure_of } h x)$

by metis

have Lindelof_space(subtopology X S)

by (simp add: Lindelof_space_closedin_subtopology <Lindelof_space X> <closedin X S>)

then obtain U **where** U: countable U $\wedge U \subseteq h ' S \wedge S \subseteq \bigcup U$

unfolding Lindelof_space_subtopology_subset [OF closedin_subset [OF <closedin X S>]]

by (smt (verit, del_insts) oh xh UN_I image_iff subsetI)

with <S ≠ {}> **have** U ≠ {}

by blast

show ?thesis

proof

show openin X (from_nat_into U n) **for** n

```

    by (metis  $\mathcal{U}$  from_nat_into image_iff  $\langle \mathcal{U} \neq \{\} \rangle$  oh subsetD)
  show disjoint T (X closure_of (from_nat_into U) n) for n
    using dh from_nat_into [OF  $\langle \mathcal{U} \neq \{\} \rangle$ ]
    by (metis  $\mathcal{U}$  f_inv_into_f inv_into_into subset_eq)
  show  $S \subseteq \bigcup (\text{range } (\text{from\_nat\_into } \mathcal{U}))$ 
    by (simp add:  $\mathcal{U} \langle \mathcal{U} \neq \{\} \rangle$ )
qed
qed

lemma regular_Lindelof_imp_normal_space:
  assumes regular_space X and Lindelof_space X
  shows normal_space X
  unfolding normal_space_def
proof clarify
  fix S T
  assume clo: closedin X S closedin X T and disjoint S T
  show  $\exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjoint } U V$ 
  proof (cases  $S = \{\} \vee T = \{\}$ )
    case True
    with clo show ?thesis
    by (meson closedin_def disjoint_empty1 disjoint_empty2 openin_empty openin_topspace
subset_empty)
  next
    case False
    obtain h :: nat  $\Rightarrow$  'a set where
      opeh:  $\bigwedge n. \text{openin } X (h n)$  and dish:  $\bigwedge n. \text{disjoint } T (X \text{ closure\_of } (h n))$ 
      and Sh:  $S \subseteq \bigcup (\text{range } h)$ 
    by (metis Lindelof_cover False  $\langle \text{disjoint } S T \rangle$  assms clo)
    obtain k :: nat  $\Rightarrow$  'a set where
      opek:  $\bigwedge n. \text{openin } X (k n)$  and disk:  $\bigwedge n. \text{disjoint } S (X \text{ closure\_of } (k n))$ 
      and Tk:  $T \subseteq \bigcup (\text{range } k)$ 
    by (metis Lindelof_cover False  $\langle \text{disjoint } S T \rangle$  assms clo disjoint_sym)
  define U where  $U \equiv \bigcup i. h i - (\bigcup j < i. X \text{ closure\_of } k j)$ 
  define V where  $V \equiv \bigcup i. k i - (\bigcup j \leq i. X \text{ closure\_of } h j)$ 
  show ?thesis
  proof (intro exI conjI)
    show openin X U openin X V
      unfolding U_def V_def
      by (force intro!: opek opeh closedin_Union closedin_closure_of)+
    show  $S \subseteq U \wedge T \subseteq V$ 
      using Sh Tk dish disk by (fastforce simp: U_def V_def disjoint_iff)+
    have  $\bigwedge x i j. \llbracket x \in k i; x \in h j; \forall j \leq i. x \notin X \text{ closure\_of } h j \rrbracket$ 
       $\implies \exists i < j. x \in X \text{ closure\_of } k i$ 
      by (metis in_closure_of linorder_not_less opek openin_subset subsetD)
    then show disjoint U V
      by (force simp: U_def V_def disjoint_iff)
  qed
qed
qed
qed

```



```

theorem Tietze_extension_closed_real_interval:
  assumes normal_space  $X$  and closedin  $X$   $S$ 
    and contf: continuous_map (subtopology  $X$   $S$ ) euclideanreal  $f$ 
    and fm:  $f \text{ ' } S \subseteq \{a..b\}$  and  $a \leq b$ 
  obtains  $g$ 
  where continuous_map  $X$  euclideanreal  $g$ 
     $\bigwedge x. x \in S \implies g \ x = f \ x \ g \ \text{ ' } \textit{topspace } X \subseteq \{a..b\}$ 
proof -
  define  $c$  where  $c \equiv \max |a| |b| + 1$ 
  have  $0 < c$  and  $c: \bigwedge x. x \in S \implies |f \ x| \leq c$ 
    using fm by (auto simp: c_def image_subset_iff)
  define good where
    good  $\equiv \lambda g \ n. \textit{continuous_map } X \textit{ euclideanreal } g \wedge (\forall x \in S. |f \ x - g \ x| \leq c * (2/3)^{\wedge} n)$ 
  have step:  $\exists g. \textit{good } g \ (\textit{Suc } n) \wedge$ 
     $(\forall x \in \textit{topspace } X. |g \ x - h \ x| \leq c * (2/3)^{\wedge} n / 3)$ 
  if  $h: \textit{good } h \ n$  for  $n \ h$ 
proof -
  have pos:  $0 < c * (2/3)^{\wedge} n$ 
    by (simp add: <0 < c>)
  have S_eq:  $S = \textit{topspace}(\textit{subtopology } X \ S)$  and  $S \subseteq \textit{topspace } X$ 
    using  $\langle \textit{closedin } X \ S \rangle$  closedin_subset by auto
  define  $d$  where  $d \equiv c/3 * (2/3)^{\wedge} n$ 
  define SA where  $SA \equiv \{x \in S. f \ x - h \ x \in \{..-d\}\}$ 
  define SB where  $SB \equiv \{x \in S. f \ x - h \ x \in \{d..\}\}$ 
  have contfh: continuous_map (subtopology  $X$   $S$ ) euclideanreal  $(\lambda x. f \ x - h \ x)$ 
    using that
  by (simp add: contf good_def continuous_map_diff continuous_map_from_subtopology)
  then have closedin (subtopology  $X$   $S$ ) SA
    unfolding SA_def
    by (smt (verit, del_insts) closed_closedin continuous_map_closedin Collect_cong S_eq closed_real_atMost)
  then have closedin  $X$  SA
    using  $\langle \textit{closedin } X \ S \rangle$  closedin_trans_full by blast
  moreover have closedin (subtopology  $X$   $S$ ) SB
    unfolding SB_def
    using closedin_continuous_map_preimage_gen [OF contfh]
    by (metis (full_types) S_eq closed_atLeast closed_closedin closedin_topospace)
  then have closedin  $X$  SB
    using  $\langle \textit{closedin } X \ S \rangle$  closedin_trans_full by blast
  moreover have disjnt SA SB
    using pos by (auto simp: d_def disjnt_def SA_def SB_def)
  moreover have  $-d \leq d$ 
    using pos by (auto simp: d_def)
  ultimately
  obtain  $g$  where contg: continuous_map  $X$  (top_of_set  $\{-d..d\}$ )  $g$ 
    and ga:  $g \ \text{ ' } SA \subseteq \{-d\}$  and gb:  $g \ \text{ ' } SB \subseteq \{d\}$ 
    using Urysohn_lemma  $\langle \textit{normal\_space } X \rangle$  by metis

```

```

then have g_le_d:  $\bigwedge x. x \in \text{topspace } X \implies |g x| \leq d$ 
  by (fastforce simp: abs_le_iff continuous_map_def minus_le_iff)
have g_eq_d:  $\bigwedge x. \llbracket x \in S; f x - h x \leq -d \rrbracket \implies g x = -d$ 
  using ga by (auto simp: SA_def)
have g_eq_negd:  $\bigwedge x. \llbracket x \in S; f x - h x \geq d \rrbracket \implies g x = d$ 
  using gb by (auto simp: SB_def)
show ?thesis
  unfolding good_def
proof (intro conjI strip exI)
  show continuous_map X euclideanreal ( $\lambda x. h x + g x$ )
    using contg continuous_map_add continuous_map_in_subtopology that
    unfolding good_def by blast
  show  $|f x - (h x + g x)| \leq c * (2 / 3) ^ \text{Suc } n$  if  $x \in S$  for  $x$ 
    proof -
      have x:  $x \in \text{topspace } X$ 
        using  $\langle S \subseteq \text{topspace } X \rangle$  that by auto
      have  $|f x - h x| \leq c * (2/3) ^ n$ 
        using good_def h that by blast
      with g_eq_d [OF that] g_eq_negd [OF that] g_le_d [OF x]
      have  $|f x - (h x + g x)| \leq d + d$ 
        unfolding d_def by linarith
      then show ?thesis
        by (simp add: d_def)
    qed
  show  $|h x + g x - h x| \leq c * (2 / 3) ^ n / 3$  if  $x \in \text{topspace } X$  for  $x$ 
    using that d_def g_le_d by auto
  qed
qed
then obtain nextg where nextg:  $\bigwedge h n. \text{good } h n \implies$ 
   $\text{good } (\text{nextg } h n) (\text{Suc } n) \wedge (\forall x \in \text{topspace } X. |\text{nextg } h n x - h x| \leq c * (2/3) ^ n / 3)$ 
  by metis
define g where  $g \equiv \text{rec\_nat } (\lambda x. 0) (\lambda n r. \text{nextg } r n)$ 
have [simp]:  $g 0 x = 0$  for  $x$ 
  by (auto simp: g_def)
have g_Suc:  $g(\text{Suc } n) = \text{nextg } (g n) n$  for  $n$ 
  by (auto simp: g_def)
have good:  $\text{good } (g n) n$  for  $n$ 
proof (induction n)
  case 0
  with c show ?case
    by (auto simp: good_def)
  qed (simp add: g_Suc nextg)
have *:  $\bigwedge n x. x \in \text{topspace } X \implies |g(\text{Suc } n) x - g n x| \leq c * (2/3) ^ n / 3$ 
  using nextg g_Suc good by force
obtain h where conth: continuous_map X euclideanreal h
  and h:  $\bigwedge \varepsilon. 0 < \varepsilon \implies \forall_F n$  in sequentially.  $\forall x \in \text{topspace } X. \text{dist } (g n x) (h x) < \varepsilon$ 
proof (rule Met_TC.continuous_map_uniformly_Cauchy_limit)

```

```

show  $\forall_F n$  in sequentially. continuous_map X (Met_TC.mtopology) (g n)
  using good good_def by fastforce
show  $\exists N. \forall m n x. N \leq m \longrightarrow N \leq n \longrightarrow x \in \text{topspace } X \longrightarrow \text{dist } (g m x)$ 
   $(g n x) < \varepsilon$ 
  if  $\varepsilon > 0$  for  $\varepsilon$ 
proof -
  have  $\forall_F n$  in sequentially.  $|(2/3) ^ n| < \varepsilon/c$ 
  proof (rule Archimedean_eventually_pow_inverse)
    show  $0 < \varepsilon / c$ 
    by (simp add:  $\langle 0 < c \rangle$  that)
  qed auto
then obtain N where  $N: \bigwedge n. n \geq N \implies |(2/3) ^ n| < \varepsilon/c$ 
  by (meson eventually_sequentially_order_le_less_trans)
have  $|g m x - g n x| < \varepsilon$ 
  if  $N \leq m \wedge N \leq n$  and  $x: x \in \text{topspace } X$   $m \leq n$  for  $m n x$ 
proof (cases  $m < n$ )
  case True
  have 23:  $(\sum k = m..<n. (2/3) ^ k) = 3 * ((2/3) ^ m - (2/3::real) ^ n)$ 
    using  $\langle m \leq n \rangle$ 
    by (induction n) (auto simp: le_Suc_eq)
  have  $|g m x - g n x| \leq |\sum k = m..<n. g (Suc k) x - g k x|$ 
    by (subst sum_Suc_diff' [OF  $\langle m \leq n \rangle$ ]) linarith
  also have  $\dots \leq (\sum k = m..<n. |g (Suc k) x - g k x|)$ 
    by (rule sum_abs)
  also have  $\dots \leq (\sum k = m..<n. c * (2/3) ^ k / 3)$ 
    by (meson * sum_mono x(1))
  also have  $\dots = (c/3) * (\sum k = m..<n. (2/3) ^ k)$ 
    by (simp add: sum_distrib_left)
  also have  $\dots = (c/3) * 3 * ((2/3) ^ m - (2/3) ^ n)$ 
    by (simp add: sum_distrib_left 23)
  also have  $\dots < (c/3) * 3 * ((2/3) ^ m)$ 
    using  $\langle 0 < c \rangle$  by auto
  also have  $\dots < \varepsilon$ 
    using N [OF  $\langle N \leq m \rangle$ ]  $\langle 0 < c \rangle$  by (simp add: field_simps)
  finally show ?thesis .
qed (use  $\langle 0 < \varepsilon \rangle \langle m \leq n \rangle$  in auto)
then show ?thesis
  by (metis dist_commute_lessI dist_real_def nle_le)
qed
qed auto
define  $\varphi$  where  $\varphi \equiv \lambda x. \max a (\min (h x) b)$ 
show ?thesis
proof
show continuous_map X euclidean  $\varphi$ 
  unfolding  $\varphi$ _def using conth by (intro continuous_intros) auto
show  $\varphi x = f x$  if  $x \in S$  for  $x$ 
proof -
  have  $x: x \in \text{topspace } X$ 
    using  $\langle \text{closedin } X S \rangle$  closedin_subset that by blast

```

```

have h x = f x
proof (rule Met_TC.limitin_metric_unique)
  show limitin Met_TC.mtopology (λn. g n x) (h x) sequentially
    using h x by (force simp: tendsto_iff eventually_sequentially)
  show limitin Met_TC.mtopology (λn. g n x) (f x) sequentially
  proof (clarsimp simp: tendsto_iff)
    fix ε::real
    assume ε > 0
    then have ∀_F n in sequentially. |(2/3) ^ n| < ε/c
      by (intro Archimedean_eventually_pow_inverse) (auto simp: <c > 0)
    then show ∀_F n in sequentially. dist (g n x) (f x) < ε
      apply eventually_elim
      by (smt (verit) good x good_def <c > 0) dist_real_def mult.commute
pos_less_divide_eq that)
  qed
  qed auto
  then show ?thesis
    using that fim by (auto simp: φ_def)
  qed
  then show φ 'topspace X ⊆ {a..b}
    using fim <a ≤ b> by (auto simp: φ_def)
  qed
qed

```

theorem *Tietze_extension_realinterval:*

```

assumes XS: normal_space X closedin X S and T: is_interval T T ≠ {}
  and contf: continuous_map (subtopology X S) euclideanreal f
  and f 'S ⊆ T
obtains g where continuous_map X euclideanreal g g 'topspace X ⊆ T ∧ x.
x ∈ S ⇒ g x = f x
proof -
  define Φ where
    Φ ≡ λT::real set. ∀f. continuous_map (subtopology X S) euclidean f →
f'S ⊆ T
    → (∃ g. continuous_map X euclidean g ∧ g 'topspace X ⊆ T ∧ (∀ x
∈ S. g x = f x))
  have Φ T
    if *: ∧ T. [[bounded T; is_interval T; T ≠ {}]] ⇒ Φ T
    and is_interval T T ≠ {} for T
  unfolding Φ_def
proof (intro strip)
  fix f
  assume contf: continuous_map (subtopology X S) euclideanreal f
    and f 'S ⊆ T
  have Φ T: Φ ((λx. x / (1 + |x|)) ' T)
proof (rule *)
  show bounded ((λx. x / (1 + |x|)) ' T)
    using shrink_range [of T] by (force intro: boundedI [where B=1])

```

```

show is_interval (( $\lambda x. x / (1 + |x|)$ ) '  $T$ )
  using connected_shrink that(2) is_interval_connected_1 by blast
show ( $\lambda x. x / (1 + |x|)$ ) '  $T \neq \{\}$ 
  using ' $T \neq \{\}$ ' by auto
qed
moreover have continuous_map (subtopology  $X$   $S$ ) euclidean (( $\lambda x. x / (1 + |x|)$ )  $\circ$   $f$ )
  by (metis contf_continuous_map_compose continuous_map_into_fulltopology
continuous_map_real_shrink)
moreover have (( $\lambda x. x / (1 + |x|)$ )  $\circ$   $f$ ) '  $S \subseteq (\lambda x. x / (1 + |x|))$  '  $T$ 
  using ' $f$  '  $S \subseteq T$ ' by auto
ultimately obtain  $g$ 
  where contg: continuous_map  $X$  euclidean  $g$ 
    and gim:  $g$  ' topspace  $X \subseteq (\lambda x. x / (1 + |x|))$  '  $T$ 
    and geq:  $\bigwedge x. x \in S \implies g\ x = ((\lambda x. x / (1 + |x|)) \circ f)\ x$ 
  using  $\Phi T$  unfolding  $\Phi\_def$  by force
show  $\exists g. \text{continuous\_map } X \text{ euclideanreal } g \wedge g \text{ ' } \text{topspace } X \subseteq T \wedge (\forall x \in S. g\ x = f\ x)$ 
proof (intro conjI exI)
  have continuous_map  $X$  (top_of_set  $\{-1 <..< 1\}$ )  $g$ 
    using contg continuous_map_in_subtopology gim shrink_range by blast
  then show continuous_map  $X$  euclideanreal (( $\lambda x. x / (1 - |x|)$ )  $\circ$   $g$ )
    by (rule continuous_map_compose) (auto simp: continuous_on_real_grow)
  show (( $\lambda x. x / (1 - |x|)$ )  $\circ$   $g$ ) ' topspace  $X \subseteq T$ 
    using gim real_grow_shrink by fastforce
  show  $\forall x \in S. ((\lambda x. x / (1 - |x|)) \circ g)\ x = f\ x$ 
    using geq real_grow_shrink by force
qed
qed
moreover have  $\Phi\ T$ 
  if bounded  $T$  is_interval  $T\ T \neq \{\}$  for  $T$ 
  unfolding  $\Phi\_def$ 
proof (intro strip)
  fix  $f$ 
  assume contf: continuous_map (subtopology  $X$   $S$ ) euclideanreal  $f$ 
    and  $f$  '  $S \subseteq T$ 
  obtain  $a\ b$  where ab: closure  $T = \{a..b\}$ 
  by (meson 'bounded  $T$ ' 'is_interval  $T$ ' compact_closure_connected_compact_interval_1
    connected_imp_connected_closure_is_interval_connected)
  with ' $T \neq \{\}$ ' have  $a \leq b$  by auto
  have  $f$  '  $S \subseteq \{a..b\}$ 
    using ' $f$  '  $S \subseteq T$ ' ab closure_subset by auto
  then obtain  $g$  where contg: continuous_map  $X$  euclideanreal  $g$ 
    and gf:  $\bigwedge x. x \in S \implies g\ x = f\ x$  and gim:  $g$  ' topspace  $X \subseteq \{a..b\}$ 
    using Tietze_extension_closed_real_interval [OF  $X S$  contf _ ' $a \leq b$ '] by
metis
  define  $W$  where  $W \equiv \{x \in \text{topspace } X. g\ x \in \text{closure } T - T\}$ 
  have  $\{a..b\} - \{a, b\} \subseteq T$ 

```

```

using that
by (metis ab atLeastAtMost_diff_ends convex_interior_closure interior_atLeastAtMost_real

interior_subset is_interval_convex)
with finite_imp_compact have compact (closure T - T)
by (metis Diff_eq_empty_iff Diff_insert2 ab finite.emptyI finite_Diff_insert)
then have closedin X W
unfolding W_def using closedin_continuous_map_preimage [OF contg]
compact_imp_closed by force
moreover have disjnt W S
unfolding W_def disjnt_iff using ⟨f ' S ⊆ T⟩ gf by blast
ultimately obtain h :: 'a ⇒ real
where conth: continuous_map X (top_of_set {0..1}) h
and him: h ' W ⊆ {0} h ' S ⊆ {1}
by (metis XS normal_space_iff Urysohn)
then have him01: h ' topspace X ⊆ {0..1}
by (meson continuous_map_in_subtopology)
obtain z where z ∈ T
using ⟨T ≠ {}⟩ by blast
define g' where g' ≡ λx. z + h x * (g x - z)
show ∃g. continuous_map X euclidean g ∧ g ' topspace X ⊆ T ∧ (∀x∈S. g x
= f x)
proof (intro exI conjI)
show continuous_map X euclideanreal g'
unfolding g'_def using contg conth continuous_map_in_subtopology
by (intro continuous_intros) auto
show g' ' topspace X ⊆ T
unfolding g'_def
proof clarify
fix x
assume x ∈ topspace X
show z + h x * (g x - z) ∈ T
proof (cases g x ∈ T)
case True
define w where w ≡ z + h x * (g x - z)
have |h x| * |g x - z| ≤ |g x - z| |1 - h x| * |g x - z| ≤ |g x - z|
using him01 ⟨x ∈ topspace X⟩ by (force simp: intro: mult_left_le_one_le)+
then consider z ≤ w ∧ w ≤ g x | g x ≤ w ∧ w ≤ z
unfolding w_def by (smt (verit) left_diff_distrib mult_cancel_right2
mult_minus_right zero_less_mult_iff)
then show ?thesis
using ⟨is_interval T⟩ unfolding w_def is_interval_1 by (metis True
⟨z ∈ T⟩)
next
case False
then have g x ∈ closure T
using ⟨x ∈ topspace X⟩ ab gim by blast
then have h x = 0
using him False ⟨x ∈ topspace X⟩ by (auto simp: W_def image_subset_iff)

```

```

      then show ?thesis
        by (simp add: ‹z ∈ T›)
    qed
  qed
  show  $\forall x \in S. g' x = f x$ 
    using gf_him by (auto simp: W_def g'_def)
  qed
  qed
  ultimately show thesis
    using assms that unfolding  $\Phi\_def$  by best
  qed

lemma normal_space_iff_Tietze:
  normal_space X  $\longleftrightarrow$ 
  ( $\forall f S. \text{closedin } X S \wedge$ 
    continuous_map (subtopology X S) euclidean f
     $\longrightarrow (\exists g:: 'a \Rightarrow \text{real. continuous\_map } X \text{ euclidean } g \wedge (\forall x \in S. g x = f$ 
  x)))
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (metis Tietze_extension_realinterval empty_not_UNIV is_interval_univ
  subset_UNIV)
next
  assume R: ?rhs
  show ?lhs
    unfolding normal_space_iff_Urysohn_alt
  proof clarify
    fix S T
    assume closedin X S
      and closedin X T
      and disjnt S T
    then have cloST: closedin X (S  $\cup$  T)
      by (simp add: closedin_Un)
    moreover
    have continuous_map (subtopology X (S  $\cup$  T)) euclideanreal ( $\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } 1$ )
      unfolding continuous_map_closedin
    proof (intro conjI strip)
      fix C :: real set
      define D where D  $\equiv \{x \in \text{topspace } X. \text{if } x \in S \text{ then } 0 \in C \text{ else } x \in T \wedge 1 \in C\}$ 
      have D  $\in \{\{\}, S, T, S \cup T\}$ 
        unfolding D_def
        using closedin_subset [OF ‹closedin X S›] closedin_subset [OF ‹closedin X T›] ‹disjnt S T›
        by (auto simp: disjnt_iff)
      then have closedin X D

```

```

    using ‹closedin X S› ‹closedin X T› closedin_empty by blast
  with closedin_subset_topspace
  show closedin (subtopology X (S ∪ T)) {x ∈ topspace (subtopology X (S ∪
T))}. (if x ∈ S then 0::real else 1) ∈ C}
  apply (simp add: D_def)
  by (smt (verit, best) Collect_cong Collect_mono_iff Un_def closedin_subset_topspace)
qed auto
ultimately obtain g :: 'a ⇒ real where
  contg: continuous_map X euclidean g and gf: ∀ x ∈ S ∪ T. g x = (if x ∈ S
then 0 else 1)
  using R by blast
  then show ∃ f. continuous_map X euclideanreal f ∧ f ' S ⊆ {0} ∧ f ' T ⊆
{1}
  by (smt (verit) Un_iff ‹disjnt S T› disjnt_iff image_subset_iff insert_iff)
  qed
  qed

```

6.11.2 Random metric space stuff

```

lemma metrizable_imp_k_space:
  assumes metrizable_space X
  shows k_space X
proof -
  obtain M d where Metric_space M d and Xeq: X = Metric_space.mtopology
M d
  using assms unfolding metrizable_space_def by metis
  then interpret Metric_space M d
  by blast
  show ?thesis
  unfolding k_space Xeq
proof clarsimp
  fix S
  assume S ⊆ M and S: ∀ K. compactin mtopology K ⟶ closedin (subtopology
mtopology K) (K ∩ S)
  have l ∈ S
  if σ: range σ ⊆ S and l: limitin mtopology σ l sequentially for σ l
proof -
  define K where K ≡ insert l (range σ)
  interpret Submetric M d K
  proof
  show K ⊆ M
  unfolding K_def using l limitin_mspace ‹S ⊆ M› σ by blast
  qed
  have compactin mtopology K
  unfolding K_def using ‹S ⊆ M› σ
  by (force intro: compactin_sequence_with_limit [OF l])
  then have *: closedin sub.mtopology (K ∩ S)
  by (simp add: S mtopology_submetric)
  have σ n ∈ K ∩ S for n

```



```

    by (simp add: K_def range_subsetD  $\sigma$ )
  moreover have limitin sub.mtopology  $\sigma$  l sequentially
    using l
    unfolding sub.limit_metric_sequentially limit_metric_sequentially
    by (force simp: K_def)
  ultimately have  $l \in K \cap S$ 
    by (meson *  $\sigma$  image_subsetI sub.metric_closedin_iff_sequentially_closed)

  then show ?thesis
    by simp
qed
then show closedin mtopology S
  unfolding metric_closedin_iff_sequentially_closed
  using  $\langle S \subseteq M \rangle$  by blast
qed
qed

lemma (in Metric_space) k_space_mtopology: k_space mtopology
  by (simp add: metrizable_imp_k_space metrizable_space_mtopology)

lemma k_space_euclideanreal: k_space (euclidean :: 'a::metric_space topology)
  using metrizable_imp_k_space metrizable_space_euclidean by auto

```

6.11.3 Hereditarily normal spaces

```

lemma hereditarily_B:
  assumes  $\bigwedge S T. \text{separatedin } X S T$ 
     $\implies \exists U V. \text{openin } X U \wedge \text{openin } X V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U V$ 
  shows hereditarily normal_space X
  unfolding hereditarily_def
proof (intro strip)
  fix W
  assume  $W \subseteq \text{topspace } X$ 
  show normal_space (subtopology X W)
    unfolding normal_space_def
  proof clarify
    fix S T
    assume clo: closedin (subtopology X W) S closedin (subtopology X W) T
      and disjnt S T
    then have separatedin (subtopology X W) S T
      by (simp add: separatedin_closed_sets)
    then obtain U V where openin X U  $\wedge$  openin X V  $\wedge$   $S \subseteq U \wedge T \subseteq V \wedge$ 
      disjnt U V
      using assms [of S T] by (meson separatedin_subtopology)
    then show  $\exists U V. \text{openin } (\text{subtopology } X W) U \wedge \text{openin } (\text{subtopology } X W) V \wedge$ 
       $S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U V$ 
      apply (simp add: openin_subtopology_alt)
      by (meson clo closedin_imp_subset disjnt_subset1 disjnt_subset2 inf_le2)
  qed
qed

```

qed

lemma *hereditarily_C*:

assumes *separatedin* X S T **and** *norm*: $\bigwedge U. \text{openin } X \ U \implies \text{normal_space}$
(*subtopology* X U)

shows $\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U \ V$

proof –

define ST **where** $ST \equiv X \ \text{closure_of } S \cap X \ \text{closure_of } T$

have $\text{sub}X$: $S \subseteq \text{topspace } X \ T \subseteq \text{topspace } X$

by (*meson* $\langle \text{separatedin } X \ S \ T \rangle \ \text{separation_closedin_Un_gen}$) $+$

have sub : $S \subseteq \text{topspace } X - ST \ T \subseteq \text{topspace } X - ST$

unfolding ST_def

by (*metis* $\text{Diff_mono } \text{Diff_triv } \langle \text{separatedin } X \ S \ T \rangle \ \text{Int_lower1 } \text{Int_lower2}$
separatedin_def) $+$

have *normal_space* (*subtopology* X ($\text{topspace } X - ST$))

by (*simp* *add*: $ST_def \ \text{norm } \text{closedin_Int } \text{openin_diff}$)

moreover **have** disjnt (*subtopology* X ($\text{topspace } X - ST$) *closure_of* S)
(*subtopology* X ($\text{topspace } X - ST$) *closure_of* T)

using $\text{Int_absorb1 } ST_def \ \text{sub}$ **by** (*fastforce* *simp*: $\text{disjnt_iff } \text{closure_of_subtopology}$)

ultimately show *?thesis*

using $\text{sub } \text{sub}X$

apply (*simp* *add*: *normal_space_closures*)

by (*metis* $ST_def \ \text{closedin_Int } \text{closedin_closure_of } \text{closedin_def } \text{openin_trans_full}$)

qed

lemma *hereditarily_normal_space*:

hereditarily_normal_space $X \iff (\forall U. \text{openin } X \ U \implies \text{normal_space}(\text{subtopology}$
 $X \ U))$

by (*metis* *hereditarily_B* *hereditarily_C* *hereditarily*)

lemma *hereditarily_normal_separation*:

hereditarily_normal_space $X \iff$

$(\forall S \ T. \text{separatedin } X \ S \ T$

$\implies (\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U$

$V))$

by (*metis* *hereditarily_B* *hereditarily_C* *hereditarily*)

lemma *metrizable_imp_hereditarily_normal_space*:

metrizable_space $X \implies \text{hereditarily_normal_space } X$

by (*simp* *add*: *hereditarily_metrizable_imp_normal_space* *metrizable_space_subtopology*)

lemma *metrizable_space_separation*:

$\llbracket \text{metrizable_space } X; \text{separatedin } X \ S \ T \rrbracket$

$\implies \exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge S \subseteq U \wedge T \subseteq V \wedge \text{disjnt } U \ V$

by (*metis* *hereditarily* *hereditarily_C* *metrizable_imp_hereditarily_normal_space*)

lemma *hereditarily_normal_separation_pairwise*:

hereditarily_normal_space $X \iff$

```

  (∀U. finite U ∧ (∀S ∈ U. S ⊆ topspace X) ∧ pairwise (separatedin X) U
    → (∃F. (∀S ∈ U. openin X (F S) ∧ S ⊆ F S) ∧
        pairwise (λS T. disjnt (F S) (F T)) U))
  (is ?lhs ←→ ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof clarify
    fix U
    assume finite U and U: ∀S ∈ U. S ⊆ topspace X
    and puU: pairwise (separatedin X) U
    have ∃ V W. openin X V ∧ openin X W ∧ S ⊆ V ∧
      (∀ T. T ∈ U ∧ T ≠ S → T ⊆ W) ∧ disjnt V W
    if S ∈ U for S
    proof -
      have separatedin X S (⋃(U - {S}))
      by (metis U ⟨finite U⟩ puU finite_Diff pairwise_alt separatedin_Union(1)
that)
      with L show ?thesis
      unfolding hereditarily_normal_separation
      by (smt (verit) Diff_iff UnionI empty_iff insert_iff subset_iff)
    qed
    then obtain F G
    where *: ⋀S. S ∈ U ⇒ S ⊆ F S ∧ (∀ T. T ∈ U ∧ T ≠ S → T ⊆ G S)
    and ope: ⋀S. S ∈ U ⇒ openin X (F S) ∧ openin X (G S)
    and dis: ⋀S. S ∈ U ⇒ disjnt (F S) (G S)
    by metis
    define H where H ≡ λS. F S ∩ (⋂ T ∈ U - {S}. G T)
    show ∃F. (∀S ∈ U. openin X (F S) ∧ S ⊆ F S) ∧ pairwise (λS T. disjnt (F
S) (F T)) U
    proof (intro exI conjI strip)
      show openin X (H S) if S ∈ U for S
      unfolding H_def
      by (smt (verit) ope that DiffD1 ⟨finite U⟩ finite_Diff finite_imageI imageE
openin_Int_Inter)
      show S ⊆ H S if S ∈ U for S
      unfolding H_def using * that by auto
      show pairwise (λS T. disjnt (H S) (H T)) U
      using dis by (fastforce simp: disjnt_iff pairwise_alt H_def)
    qed
  qed
next
  assume R: ?rhs
  show ?lhs
  unfolding hereditarily_normal_separation
  proof (intro strip)
    fix S T
    assume separatedin X S T
    show ∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U V

```

```

proof (cases T=S)
  case True
  then show ?thesis
    using ‹separatedin X S T› by force
next
  case False
  have pairwise (separatedin X) {S, T}
    by (simp add: ‹separatedin X S T› pairwise_insert separatedin_sym)
  moreover have  $\forall S \in \{S, T\}. S \subseteq \text{topspace } X$ 
    by (metis ‹separatedin X S T› insertE separatedin_def singletonD)
  ultimately show ?thesis
    using R by (smt (verit) False finite.emptyI finite.insertI insertCI pairwiseD)
qed
qed
qed

```

```

lemma hereditarily_normal_space_perfect_map_image:
  [[hereditarily_normal_space X; perfect_map X Y f]]  $\implies$  hereditarily_normal_space
  Y
  unfolding perfect_map_def proper_map_def
  by (meson hereditarily_normal_space_continuous_closed_map_image)

```

```

lemma regular_second_countable_imp_hereditarily_normal_space:
  assumes regular_space X  $\wedge$  second_countable X
  shows hereditarily_normal_space X
  unfolding hereditarily
  proof (intro regular_Lindelof_imp_normal_space strip)
  show regular_space (subtopology X S) for S
    by (simp add: assms regular_space_subtopology)
  show Lindelof_space (subtopology X S) for S
    using assms by (simp add: second_countable_imp_Lindelof_space second_countable_subtopology)
qed

```

6.11.4 Completely regular spaces

```

definition completely_regular_space where
  completely_regular_space X  $\equiv$ 
   $\forall S x. \text{closedin } X S \wedge x \in \text{topspace } X - S$ 
   $\longrightarrow (\exists f::'a \Rightarrow \text{real}. \text{continuous\_map } X (\text{top\_of\_set } \{0..1\}) f \wedge$ 
   $f x = 0 \wedge (f \text{ ` } S \subseteq \{1\}))$ 

```

```

lemma homeomorphic_completely_regular_space_aux:
  assumes X: completely_regular_space X and hom: X homeomorphic_space Y
  shows completely_regular_space Y
proof –
  obtain f g where hmf: homeomorphic_map X Y f and hmg: homeomorphic_map
  Y X g
  and fg: ( $\forall x \in \text{topspace } X. g(f x) = x$ )  $\wedge$  ( $\forall y \in \text{topspace } Y. f(g y) = y$ )
  using assms X homeomorphic_maps_map homeomorphic_space_def by fast-

```

```

force
show ?thesis
  unfolding completely_regular_space_def
proof clarify
  fix S x
  assume A: closedin Y S and x: x ∈ topspace Y and x ∉ S
  then have closedin X (g`S)
    using hmg homeomorphic_map_closedness_eq by blast
  moreover have g x ∉ g`S
  by (meson A x ⟨x ∉ S⟩ closedin_subset hmg homeomorphic_imp_injective_map
inj_on_image_mem_iff)
  ultimately obtain φ where φ: continuous_map X (top_of_set {0..1::real})
φ ∧ φ (g x) = 0 ∧ φ ` g`S ⊆ {1}
  by (metis DiffI X completely_regular_space_def hmg homeomorphic_imp_surjective_map
image_eqI x)
  then have continuous_map Y (top_of_set {0..1::real}) (φ ∘ g)
  by (meson continuous_map_compose hmg homeomorphic_imp_continuous_map)
  then show ∃ ψ. continuous_map Y (top_of_set {0..1::real}) ψ ∧ ψ x = 0 ∧
ψ ` S ⊆ {1}
  by (metis φ comp_apply image_comp)
qed
qed

```

```

lemma homeomorphic_completely_regular_space:
  assumes X homeomorphic_space Y
  shows completely_regular_space X ↔ completely_regular_space Y
  by (meson assms homeomorphic_completely_regular_space_aux homeomorphic_space_sym)

```

```

lemma completely_regular_space_alt:
  completely_regular_space X ↔
  (∀ S x. closedin X S → x ∈ topspace X - S
→ (∃ f. continuous_map X euclideanreal f ∧ f x = 0 ∧ f ` S ⊆ {1}))

```

```

proof -
  have ∃ f. continuous_map X (top_of_set {0..1::real}) f ∧ f x = 0 ∧ f ` S ⊆
{1}
  if closedin X S x ∈ topspace X - S and f: continuous_map X euclideanreal f
∧ f x = 0 ∧ f ` S ⊆ {1}
  for S x f
  proof (intro exI conjI)
  show continuous_map X (top_of_set {0..1}) (λx. max 0 (min (f x) 1))
  using that
  by (auto simp: continuous_map_in_subtopology intro!: continuous_map_real_max
continuous_map_real_min)
  qed (use that in auto)
  with continuous_map_in_subtopology show ?thesis
  unfolding completely_regular_space_def by metis
qed

```

As above, but with *openin*

lemma *completely_regular_space_alt'*:
completely_regular_space $X \longleftrightarrow$
 $(\forall S x. \text{openin } X S \longrightarrow x \in S$
 $\longrightarrow (\exists f. \text{continuous_map } X \text{ euclideanreal } f \wedge f x = 0 \wedge f' (\text{topspace } X$
 $- S) \subseteq \{1\}))$
apply (*simp add: completely_regular_space_alt all_closedin*)
by (*meson openin_subset subsetD*)

lemma *completely_regular_space_gen_alt*:
fixes $a b :: \text{real}$
assumes $a \neq b$
shows *completely_regular_space* $X \longleftrightarrow$
 $(\forall S x. \text{closedin } X S \longrightarrow x \in \text{topspace } X - S$
 $\longrightarrow (\exists f. \text{continuous_map } X \text{ euclidean } f \wedge f x = a \wedge (f' S \subseteq \{b\})))$

proof –
have $\exists f. \text{continuous_map } X \text{ euclideanreal } f \wedge f x = 0 \wedge f' S \subseteq \{1\}$
if $\text{closedin } X S x \in \text{topspace } X - S$
and $f: \text{continuous_map } X \text{ euclidean } f \wedge f x = a \wedge f' S \subseteq \{b\}$
for $S x f$
proof (*intro exI conjI*)
show *continuous_map* $X \text{ euclideanreal } ((\lambda x. \text{inverse}(b - a) * (x - a)) \circ f)$
using that **by** (*intro continuous_intros*) *auto*
qed (*use that assms in auto*)
moreover
have $\exists f. \text{continuous_map } X \text{ euclidean } f \wedge f x = a \wedge f' S \subseteq \{b\}$
if $\text{closedin } X S x \in \text{topspace } X - S$
and $f: \text{continuous_map } X \text{ euclideanreal } f \wedge f x = 0 \wedge f' S \subseteq \{1\}$
for $S x f$
proof (*intro exI conjI*)
show *continuous_map* $X \text{ euclideanreal } ((\lambda x. a + (b - a) * x) \circ f)$
using that **by** (*intro continuous_intros*) *auto*
qed (*use that in auto*)
ultimately show *?thesis*
unfolding *completely_regular_space_alt* **by** *meson*
qed

As above, but with *openin*

lemma *completely_regular_space_gen_alt'*:
fixes $a b :: \text{real}$
assumes $a \neq b$
shows *completely_regular_space* $X \longleftrightarrow$
 $(\forall S x. \text{openin } X S \longrightarrow x \in S$
 $\longrightarrow (\exists f. \text{continuous_map } X \text{ euclidean } f \wedge f x = a \wedge (f' (\text{topspace } X$
 $- S) \subseteq \{b\})))$
apply (*simp add: completely_regular_space_gen_alt[OF assms] all_closedin*)
by (*meson openin_subset subsetD*)

lemma *completely_regular_space_gen*:
fixes $a b :: \text{real}$

```

assumes  $a < b$ 
shows completely_regular_space  $X \longleftrightarrow$ 
  ( $\forall S x. \text{closedin } X S \wedge x \in \text{topspace } X - S$ 
     $\longrightarrow (\exists f. \text{continuous\_map } X (\text{top\_of\_set } \{a..b\}) f \wedge f x = a \wedge f ' S$ 
 $\subseteq \{b\}))$ 
proof -
  have  $\exists f. \text{continuous\_map } X (\text{top\_of\_set } \{a..b\}) f \wedge f x = a \wedge f ' S \subseteq \{b\}$ 
    if  $\text{closedin } X S \wedge x \in \text{topspace } X - S$ 
    and  $f: \text{continuous\_map } X \text{ euclidean } f \wedge f x = a \wedge f ' S \subseteq \{b\}$ 
    for  $S x f$ 
  proof (intro exI conjI)
    show  $\text{continuous\_map } X (\text{top\_of\_set } \{a..b\}) (\lambda x. \max a (\min (f x) b))$ 
      using that assms
      by (auto simp: continuous_map_in_subtopology intro!: continuous_map_real_max
continuous_map_real_min)
    qed (use that assms in auto)
  with continuous_map_in_subtopology assms show ?thesis
    using completely_regular_space_gen_alt [of a b]
    by (smt (verit) atLeastAtMost_singleton atLeastAtMost_empty_singletonI)
qed

```

lemma *normal_imp_completely_regular_space_A:*

```

assumes normal_space  $X$  t1_space  $X$ 
shows completely_regular_space  $X$ 
unfolding completely_regular_space_alt
proof clarify
  fix  $x S$ 
  assume  $A: \text{closedin } X S \wedge x \notin S$ 
  assume  $x \in \text{topspace } X$ 
  then have  $\text{closedin } X \{x\}$ 
    by (simp add: <t1_space X> closedin_t1_singleton)
  with  $A$  <normal_space X> have  $\exists f. \text{continuous\_map } X \text{ euclideanreal } f \wedge f ' \{x\}$ 
 $\subseteq \{0\} \wedge f ' S \subseteq \{1\}$ 
    using assms unfolding normal_space_iff_Urysohn_alt disjnt_iff by blast
  then show  $\exists f. \text{continuous\_map } X \text{ euclideanreal } f \wedge f x = 0 \wedge f ' S \subseteq \{1\}$ 
    by auto
qed

```

lemma *normal_imp_completely_regular_space_B:*

```

assumes normal_space  $X$  regular_space  $X$ 
shows completely_regular_space  $X$ 
unfolding completely_regular_space_alt
proof clarify
  fix  $x S$ 
  assume  $\text{closedin } X S \wedge x \notin S \wedge x \in \text{topspace } X$ 
  then obtain  $U C$  where  $\text{openin } X U \wedge \text{closedin } X C \wedge x \in U \wedge U \subseteq C \wedge C \subseteq \text{topspace } X - S$ 
    using assms
  unfolding neighbourhood_base_of_closedin [symmetric] neighbourhood_base_of

```

closedin_def **by** (*metis Diff_iff*)
then obtain *f* **where** *continuous_map X euclideanreal f* $\wedge f ' C \subseteq \{0\} \wedge f ' S \subseteq \{1\}$
using *assms unfolding normal_space_iff Urysohn_alt*
by (*metis Diff_iff* \langle *closedin X S* \rangle *disjnt_iff subsetD*)
then show $\exists f. \text{continuous_map } X \text{ euclideanreal } f \wedge f x = 0 \wedge f ' S \subseteq \{1\}$
by (*meson* $\langle U \subseteq C \rangle \langle x \in U \rangle$ *image_subset_iff singletonD subsetD*)
qed

lemma *normal_imp_completely_regular_space_gen*:
 $\llbracket \text{normal_space } X; t1_space X \vee \text{Hausdorff_space } X \vee \text{regular_space } X \rrbracket \implies$
completely_regular_space X
using *normal_imp_completely_regular_space_A normal_imp_completely_regular_space_B*
t1_or_Hausdorff_space **by** *blast*

lemma *normal_imp_completely_regular_space*:
 $\llbracket \text{normal_space } X; \text{Hausdorff_space } X \vee \text{regular_space } X \rrbracket \implies \text{completely_regular_space } X$
by (*simp add: normal_imp_completely_regular_space_gen*)

lemma (**in** *Metric_space*) *completely_regular_space_mtopology*:
completely_regular_space mtopology
by (*simp add: normal_imp_completely_regular_space normal_space_mtopology*
regular_space_mtopology)

lemma *metrizable_imp_completely_regular_space*:
metrizable_space X \implies *completely_regular_space X*
by (*simp add: metrizable_imp_normal_space metrizable_imp_regular_space normal_imp_completely_regular_space*)

lemma *completely_regular_space_discrete_topology*:
completely_regular_space(discrete_topology U)
by (*simp add: normal_imp_completely_regular_space normal_space_discrete_topology*)

lemma *completely_regular_space_subtopology*:
assumes *completely_regular_space X*
shows *completely_regular_space (subtopology X S)*
unfolding *completely_regular_space_def*
proof *clarify*
fix *A x*
assume *closedin (subtopology X S) A* **and** *x: x \in topspace (subtopology X S)*
and *x \notin A*
then obtain *T* **where** *closedin X T A = S \cap T x \in topspace X x \in S*
by (*force simp: closedin_subtopology_alt image_iff*)
then show $\exists f. \text{continuous_map (subtopology X S) (top_of_set } \{0::\text{real..1}\}) f$
 $\wedge f x = 0 \wedge f ' A \subseteq \{1\}$
using *assms* $\langle x \notin A \rangle$
apply (*simp add: completely_regular_space_def continuous_map_from_subtopology*)
using *continuous_map_from_subtopology* **by** *fastforce*

qed

lemma *completely_regular_space_retraction_map_image*:

$\llbracket \text{retraction_map } X \ Y \ r; \text{ completely_regular_space } X \rrbracket \implies \text{completely_regular_space } Y$

using *completely_regular_space_subtopology hereditary_imp_retractive_property homeomorphic_completely_regular_space* **by** *blast*

lemma *completely_regular_imp_regular_space*:

assumes *completely_regular_space* X

shows *regular_space* X

proof –

have $*$: $\exists U \ V. \text{openin } X \ U \wedge \text{openin } X \ V \wedge a \in U \wedge C \subseteq V \wedge \text{disjnt } U \ V$

if *contf*: *continuous_map* $X \ \text{euclideanreal} \ f$ **and** $a: a \in \text{topspace } X - C$ **and** *closedin* $X \ C$

and *fm*: $f \text{ ' } \text{topspace } X \subseteq \{0..1\}$ **and** $f0: f \ a = 0$ **and** $f1: f \text{ ' } C \subseteq \{1\}$

for $C \ a \ f$

proof (*intro exI conjI*)

show *openin* $X \ \{x \in \text{topspace } X. f \ x \in \{..<1 / 2\}\}$ *openin* $X \ \{x \in \text{topspace } X. f \ x \in \{1 / 2<..\}\}$

using *openin_continuous_map_preimage* [*OF contf*]

by (*meson open_lessThan open_greaterThan open_openin*)+

show $a \in \{x \in \text{topspace } X. f \ x \in \{..<1 / 2\}\}$

using $a \ f0$ **by** *auto*

show $C \subseteq \{x \in \text{topspace } X. f \ x \in \{1 / 2<..\}\}$

using $\langle \text{closedin } X \ C \rangle f1$ *closedin_subset* **by** *auto*

qed (*auto simp: disjnt_iff*)

show *?thesis*

using *assms*

unfolding *completely_regular_space_def regular_space_def continuous_map_in_subtopology*

by (*meson **)

qed

proposition *locally_compact_regular_imp_completely_regular_space*:

assumes *locally_compact_space* X *Hausdorff_space* $X \ \vee$ *regular_space* X

shows *completely_regular_space* X

unfolding *completely_regular_space_def*

proof *clarify*

fix $S \ x$

assume *closedin* $X \ S$ **and** $x \in \text{topspace } X$ **and** $x \notin S$

have *regular_space* X

using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*

then have *nbase: neighbourhood_base_of* $(\lambda C. \text{compactin } X \ C \wedge \text{closedin } X \ C)$

X

using *assms(1) locally_compact_regular_space_neighbourhood_base* **by** *blast*

then obtain $U \ M$ **where** *openin* $X \ U$ *compactin* $X \ M$ *closedin* $X \ M$ $x \in U$ $U \subseteq M$ $M \subseteq \text{topspace } X - S$

unfolding *neighbourhood_base_of* **by** (*metis (no_types, lifting) Diff_iff <closedin*

```

X S› ⟨x ∈ topspace X› ⟨x ∉ S› closedin_def)
  then have M ⊆ topspace X
    by blast
  obtain V K where openin X V closedin X K x ∈ V V ⊆ K K ⊆ U
    by (metis (no_types, lifting) ⟨openin X U› ⟨x ∈ U› neighbourhood_base_of
nbase)
  have compact_space (subtopology X M)
    by (simp add: ⟨compactin X M› compact_space_subtopology)
  then have normal_space (subtopology X M)
    by (simp add: ⟨regular_space X› compact_Hausdorff_or_regular_imp_normal_space
regular_space_subtopology)
  moreover have closedin (subtopology X M) K
    using ⟨K ⊆ U› ⟨U ⊆ M› ⟨closedin X K› closedin_subset_topspace by fastforce
  moreover have closedin (subtopology X M) (M - U)
    by (simp add: ⟨closedin X M› ⟨openin X U› closedin_diff closedin_subset_topspace)
  moreover have disjnt K (M - U)
    by (meson DiffD2 ⟨K ⊆ U› disjnt_iff subsetD)
  ultimately obtain f::'a⇒real where contf: continuous_map (subtopology X M)
(top_of_set {0..1}) f
    and f0: f ' K ⊆ {0} and f1: f ' (M - U) ⊆ {1}
    using Urysohn_lemma [of subtopology X M K M-U 0 1] by auto
  then obtain g::'a⇒real where contg: continuous_map (subtopology X M) eu-
clidean g and gim: g ' M ⊆ {0..1}
    and g0: ∧x. x ∈ K ⇒ g x = 0 and g1: ∧x. [x ∈ M; x ∉ U] ⇒ g x = 1
    using ⟨M ⊆ topspace X› by (force simp: continuous_map_in_subtopology
image_subset_iff)
  show ∃f::'a⇒real. continuous_map X (top_of_set {0..1}) f ∧ f x = 0 ∧ f ' S
⊆ {1}
  proof (intro exI conjI)
    show continuous_map X (top_of_set {0..1}) (λx. if x ∈ M then g x else 1)
      unfolding continuous_map_closedin
    proof (intro strip conjI)
      fix C
      assume C: closedin (top_of_set {0::real..1}) C
      have eq: {x ∈ topspace X. (if x ∈ M then g x else 1) ∈ C} = {x ∈ M. g x ∈
C} ∪ (if 1 ∈ C then topspace X - U else {})
        using ⟨U ⊆ M› ⟨M ⊆ topspace X› g1 by auto
      show closedin X {x ∈ topspace X. (if x ∈ M then g x else 1) ∈ C}
        unfolding eq
      proof (intro closedin_Un)
        have closedin euclidean C
          using C closed_closedin_closed_trans by blast
        then have closedin (subtopology X M) {x ∈ M. g x ∈ C}
          using closedin_continuous_map_preimage_gen [OF contg] ⟨M ⊆ topspace
X› by auto
        then show closedin X {x ∈ M. g x ∈ C}
          using ⟨closedin X M› closedin_trans_full by blast
      qed (use ⟨openin X U› in force)
    qed (use gim in force)

```

```

show (if  $x \in M$  then  $g\ x$  else 1) = 0
  using  $\langle U \subseteq M \rangle \langle V \subseteq K \rangle g0 \langle x \in U \rangle \langle x \in V \rangle$  by auto
show ( $\lambda x.$  if  $x \in M$  then  $g\ x$  else 1) '  $S \subseteq \{1\}$ 
  using  $\langle M \subseteq \text{topspace } X - S \rangle$  by auto
qed
qed

lemma completely_regular_eq_regular_space:
  locally_compact_space  $X$ 
     $\implies$  (completely_regular_space  $X \longleftrightarrow$  regular_space  $X$ )
using completely_regular_imp_regular_space locally_compact_regular_imp_completely_regular_space

by blast

lemma completely_regular_space_prod_topology:
  completely_regular_space (prod_topology  $X\ Y$ )  $\longleftrightarrow$ 
    (prod_topology  $X\ Y$ ) = trivial_topology  $\vee$ 
    completely_regular_space  $X \wedge$  completely_regular_space  $Y$  (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
    by (rule topological_property_of_prod_component)
    (auto simp: completely_regular_space_subtopology_homeomorphic_completely_regular_space)
next
  assume  $R$ : ?rhs
  show ?lhs
  proof (cases (prod_topology  $X\ Y$ ) = trivial_topology)
    case False
    then have  $X$ : completely_regular_space  $X$  and  $Y$ : completely_regular_space
 $Y$ 
      using  $R$  by blast+
    show ?thesis
    unfolding completely_regular_space_alt'
  proof clarify
    fix  $W\ x\ y$ 
    assume openin (prod_topology  $X\ Y$ )  $W$  and  $(x, y) \in W$ 
    then obtain  $U\ V$  where openin  $X\ U$  openin  $Y\ V$   $x \in U\ y \in V$   $U \times V \subseteq W$ 
      by (force simp: openin_prod_topology_alt)
    then have  $x \in \text{topspace } X\ y \in \text{topspace } Y$ 
      using openin_subset by fastforce+
    obtain  $f$  where contf: continuous_map  $X$  euclideanreal  $f$  and  $f\ x = 0$ 
      and  $f1$ :  $\bigwedge x. x \in \text{topspace } X \implies x \notin U \implies f\ x = 1$ 
      using  $X \langle \text{openin } X\ U \rangle \langle x \in U \rangle$  unfolding completely_regular_space_alt'
      by (smt (verit, best) Diff_iff_image_subset_iff_singletonD)
    obtain  $g$  where contg: continuous_map  $Y$  euclideanreal  $g$  and  $g\ y = 0$ 
      and  $g1$ :  $\bigwedge y. y \in \text{topspace } Y \implies y \notin V \implies g\ y = 1$ 
      using  $Y \langle \text{openin } Y\ V \rangle \langle y \in V \rangle$  unfolding completely_regular_space_alt'
      by (smt (verit, best) Diff_iff_image_subset_iff_singletonD)
    define  $h$  where  $h \equiv \lambda(x,y). 1 - (1 - f\ x) * (1 - g\ y)$ 
    show  $\exists h.$  continuous_map (prod_topology  $X\ Y$ ) euclideanreal  $h \wedge h\ (x,y) =$ 

```

```

0 ∧ h ‘ (topspace (prod_topology X Y) - W) ⊆ {1}
proof (intro exI conjI)
  have continuous_map (prod_topology X Y) euclideanreal (f ∘ fst)
    using contf continuous_map_of_fst by blast
  moreover
  have continuous_map (prod_topology X Y) euclideanreal (g ∘ snd)
    using contg continuous_map_of_snd by blast
  ultimately
  show continuous_map (prod_topology X Y) euclideanreal h
    unfolding o_def h_def case_prod_unfold
    by (intro continuous_intros) auto
  show h (x, y) = 0
    by (simp add: h_def ⟨f x = 0⟩ ⟨g y = 0⟩)
  show h ‘ (topspace (prod_topology X Y) - W) ⊆ {1}
    using ⟨U × V ⊆ W⟩ f1 g1 by (force simp: h_def)
  qed
qed
qed (force simp: completely_regular_space_def)
qed

```

proposition *completely_regular_space_product_topology:*

```

completely_regular_space (product_topology X I) ↔
(∃ i ∈ I. X i = trivial_topology) ∨ (∀ i ∈ I. completely_regular_space (X i))
(is ?lhs ↔ ?rhs)

```

proof

assume ?lhs **then show** ?rhs

by (rule topological_property_of_product_component)

(auto simp: completely_regular_space_subtopology_homeomorphic_completely_regular_space)

next

assume R: ?rhs

show ?lhs

proof (cases ∃ i ∈ I. X i = trivial_topology)

case False

show ?thesis

unfolding completely_regular_space_alt'

proof clarify

fix W x

assume W: openin (product_topology X I) W **and** x ∈ W

then obtain U **where** finU: finite {i ∈ I. U i ≠ topspace (X i)}

and ope: $\bigwedge i. i \in I \implies \text{openin } (X i) (U i)$

and x: x ∈ $\text{Pi}_E I U$ **and** $\text{Pi}_E I U \subseteq W$

by (auto simp: openin_product_topology_alt)

have $\forall i \in I. \text{openin } (X i) (U i) \wedge x i \in U i$

$\longrightarrow (\exists f. \text{continuous_map } (X i) \text{ euclideanreal } f \wedge$

$f (x i) = 0 \wedge (\forall x \in \text{topspace}(X i). x \notin U i \longrightarrow f x = 1))$

using R **unfolding** completely_regular_space_alt'

by (smt (verit) DiffI False image_subset_iff singletonD)

with ope x **have** $\bigwedge i. \exists f. i \in I \longrightarrow \text{continuous_map } (X i) \text{ euclideanreal } f \wedge$

```

      f (x i) = 0  $\wedge$  ( $\forall x \in \text{topspace } (X i). x \notin U i \longrightarrow f x = 1$ )
    by auto
  then obtain f where f:  $\bigwedge i. i \in I \implies \text{continuous\_map } (X i) \text{ euclideanreal}$ 
  (f i)  $\wedge$ 
      f i (x i) = 0  $\wedge$  ( $\forall x \in \text{topspace } (X i). x \notin U i$ 
 $\longrightarrow f i x = 1$ )
    by metis
  define h where h  $\equiv \lambda z. 1 - \text{prod } (\lambda i. 1 - f i (z i)) \{i \in I. U i \neq \text{topspace}(X$ 
  i)\}
  show  $\exists h. \text{continuous\_map } (\text{product\_topology } X I) \text{ euclideanreal } h \wedge h x = 0$ 
 $\wedge$ 
      h ' $(\text{topspace } (\text{product\_topology } X I) - W) \subseteq \{1\}$ 
  proof (intro conjI exI)
    have continuous_map (product_topology X I) euclidean (f i  $\circ$  ( $\lambda x. x i$ )) if
  i  $\in I$  for i
      using f that
    by (blast intro: continuous_intros continuous_map_product_projection)
  then show continuous_map (product_topology X I) euclideanreal h
      unfolding h_def o_def by (intro continuous_intros) (auto simp: finU)
  show h x = 0
      by (simp add: h_def f)
  show h ' $(\text{topspace } (\text{product\_topology } X I) - W) \subseteq \{1\}$ 
      proof -
        have  $\exists i. i \in I \wedge U i \neq \text{topspace } (X i) \wedge f i (x' i) = 1$ 
          if  $x' \in (\prod_E i \in I. \text{topspace } (X i))$   $x' \notin W$  for  $x'$ 
          using that  $\langle \text{Pi}_E I U \subseteq W \rangle$  by (smt (verit, best)  $\text{Pi}_E\_iff f \text{ in\_mono}$ )
        then show ?thesis
          by (auto simp: f h_def finU)
      qed
    qed
  qed
  qed (force simp: completely_regular_space_def)
  qed

```

lemma zero_dimensional_imp_completely_regular_space:

```

  assumes X dim_le 0
  shows completely_regular_space X
  proof (clarsimp simp: completely_regular_space_def)
    fix C a
    assume closedin X C and a  $\in$  topspace X and a  $\notin$  C
    then obtain U where closedin X U openin X U a  $\in$  U and U:  $U \subseteq \text{topspace}$ 
  X - C
      using assms by (force simp add: closedin_def dimension_le_0_neighbourhood_base_of_clopen
  open_neighbourhood_base_of)
  show  $\exists f. \text{continuous\_map } X (\text{top\_of\_set } \{0::\text{real}..1\}) f \wedge f a = 0 \wedge f ' $C \subseteq$ 
  \{1\}$ 
    proof (intro exI conjI)
      have continuous_map X euclideanreal ( $\lambda x. \text{if } x \in U \text{ then } 0 \text{ else } 1$ )

```

```

using ⟨closedin X U⟩ ⟨openin X U⟩ apply (clarsimp simp: continuous_map_def
closedin_def)
by (smt (verit) Diff_iff mem_Collect_eq openin_subopen subset_eq)
then show continuous_map X (top_of_set {0::real..1}) (λx. if x ∈ U then 0
else 1)
by (auto simp: continuous_map_in_subtopology)
qed (use U ⟨a ∈ U⟩ in auto)
qed

```

```

lemma zero_dimensional_imp_regular_space: X dim_le 0 ⇒ regular_space X
by (simp add: completely_regular_imp_regular_space zero_dimensional_imp_completely_regular_spa

```

```

lemma (in Metric_space) t1_space_mtopology:
  t1_space mtopology
using Hausdorff_space_mtopology t1_or_Hausdorff_space by blast

```

6.11.5 More generally, the k-ification functor

```

definition kifification_open
where kifification_open ≡
  λX S. S ⊆ topspace X ∧ (∀ K. compactin X K ⟶ openin (subtopology X
K) (K ∩ S))

```

```

definition kifification
where kifification X ≡ topology (kifification_open X)

```

```

lemma istopology_kifification_open: istopology (kifification_open X)
unfolding istopology_def
proof (intro conjI strip)
show kifification_open X (S ∩ T)
if kifification_open X S and kifification_open X T for S T
using that unfolding kifification_open_def
by (smt (verit, best) inf.idem inf_commute inf_left_commute le_infI2 openin_Int)
show kifification_open X (⋃ K) if ∀ K ∈ K. kifification_open X K for K
using that unfolding kifification_open_def Int_Union by blast
qed

```

```

lemma openin_kifification:
  openin (kifification X) U ⟷
  U ⊆ topspace X ∧
  (∀ K. compactin X K ⟶ openin (subtopology X K) (K ∩ U))
by (metis topology_inverse' kifification_def istopology_kifification_open kifification_open_def)

```

```

lemma openin_kifification_finer:
  openin X S ⟹ openin (kifification X) S
by (simp add: openin_kifification openin_subset openin_subtopology_Int2)

```

```

lemma topspace_kifification [simp]:
  topspace(kifification X) = topspace X

```

by (*meson openin_kification openin_kification_finer openin_topspace subset_antisym*)

lemma *closedin_kification*:

$closedin (kification X) U \longleftrightarrow$
 $U \subseteq topspace X \wedge$
 $(\forall K. compactin X K \longrightarrow closedin (subtopology X K) (K \cap U))$

proof (*cases U ⊆ topspace X*)

case *True*

then show *?thesis*

by (*simp add: closedin_def Diff_Int_distrib inf_commute le_infI2 openin_kification*)

qed (*simp add: closedin_def*)

lemma *closedin_kification_finer*: $closedin X S \implies closedin (kification X) S$

by (*simp add: closedin_def openin_kification_finer*)

lemma *kification_eq_self*: $kification X = X \longleftrightarrow k_space X$

unfolding *fun_eq_iff openin_kification k_space_alt openin_inject [symmetric]*

by (*metis openin_closedin_eq*)

lemma *compactin_kification* [*simp*]:

$compactin (kification X) K \longleftrightarrow compactin X K$ (**is** *?lhs* \longleftrightarrow *?rhs*)

proof

assume *?lhs* **then show** *?rhs*

by (*simp add: compactin_contractive openin_kification_finer*)

next

assume *R*: *?rhs*

show *?lhs*

unfolding *compactin_def*

proof (*intro conjI strip*)

show $K \subseteq topspace (kification X)$

by (*simp add: R compactin_subset_topspace*)

fix *U*

assume *U*: $Ball U (openin (kification X)) \wedge K \subseteq \bigcup U$

then have $*$: $\bigwedge U. U \in \mathcal{U} \implies U \subseteq topspace X \wedge openin (subtopology X K) (K \cap U)$

by (*simp add: R openin_kification*)

have $K \subseteq topspace X compact_space (subtopology X K)$

using *R compactin_subspace* **by** *force+*

then have $\exists V. finite V \wedge V \subseteq (\lambda U. K \cap U) \text{ ‘ } \mathcal{U} \wedge \bigcup V = topspace (subtopology X K)$

unfolding *compact_space*

by (*smt (verit, del_insts) Int_Union U * image_iff inf.order_iff inf_commute topspace_subtopology*)

then show $\exists \mathcal{F}. finite \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$

by (*metis Int_Union ‹K ⊆ topspace X› finite_subset_image inf.orderI topspace_subtopology_subset*)

qed

qed

1612

lemma *compact_space_kification* [*simp*]:
 $compact_space(kification\ X) \longleftrightarrow compact_space\ X$
 by (*simp add: compact_space_def*)

lemma *kification_kification* [*simp*]:
 $kification(kification\ X) = kification\ X$
 unfolding *openin_inject* [*symmetric*]
proof
 fix *U*
 show $openin\ (kification\ (kification\ X))\ U = openin\ (kification\ X)\ U$
 proof
 show $openin\ (kification\ (kification\ X))\ U \implies openin\ (kification\ X)\ U$
 by (*metis compactin_kification_inf_commute openin_kification openin_subtopology*
 topspace_kification)
 qed (*simp add: openin_kification_finer*)
qed

lemma *k_space_kification* [*iff*]: $k_space(kification\ X)$
 using *kification_eq_self* **by** *fastforce*

lemma *continuous_map_into_kification*:
 assumes *k_space X*
 shows $continuous_map\ X\ (kification\ Y)\ f \longleftrightarrow continuous_map\ X\ Y\ f$ (**is** *?lhs*
 \longleftrightarrow *?rhs*)
proof
 assume *?lhs* **then show** *?rhs*
 by (*simp add: continuous_map_def openin_kification_finer*)
next
 assume *R: ?rhs*
 have $openin\ X\ \{x \in topspace\ X.\ f\ x \in V\}$ **if** *V: openin (kification Y) V* **for** *V*
 proof –
 have $openin\ (subtopology\ X\ K)\ (K \cap \{x \in topspace\ X.\ f\ x \in V\})$
 if *compactin X K* **for** *K*
 proof –
 have $compactin\ Y\ (f\ 'K)$
 using *R image_compactin* **that** **by** *blast*
 then have $openin\ (subtopology\ Y\ (f\ 'K))\ (f\ 'K \cap V)$
 by (*meson V openin_kification*)
 then obtain *U* **where** $U: openin\ Y\ U\ f\ 'K \cap V = U \cap f\ 'K$
 by (*meson openin_subtopology*)
 show *?thesis*
 unfolding *openin_subtopology*
 proof (*intro conjI exI*)
 show $openin\ X\ \{x \in topspace\ X.\ f\ x \in U\}$
 using *R U openin_continuous_map_preimage_gen* **by** (*simp add: con-*
 tinuous_map_def)
 qed (*use U in auto*)
 qed
 then show *?thesis*


```

    by (metis (full_types) Collect_subset assms k_space_open)
qed
with R show ?lhs
  by (simp add: continuous_map_def)
qed

```

```

lemma continuous_map_from_kification:
  continuous_map X Y f  $\implies$  continuous_map (kification X) Y f
  by (simp add: continuous_map_openin_preimage_eq openin_kification_finer)

```

```

lemma continuous_map_kification:
  continuous_map X Y f  $\implies$  continuous_map (kification X) (kification Y) f
  by (simp add: continuous_map_from_kification continuous_map_into_kification)

```

```

lemma subtopology_kification_compact:
  assumes compactin X K
  shows subtopology (kification X) K = subtopology X K
  unfolding openin_inject [symmetric]
proof
  fix U
  show openin (subtopology (kification X) K) U = openin (subtopology X K) U
  by (metis assms inf_commute openin_kification openin_subset openin_subtopology)
qed

```

```

lemma subtopology_kification_finer:
  assumes openin (subtopology (kification X) S) U
  shows openin (kification (subtopology X S)) U
  using assms
  by (fastforce simp: openin_subtopology_alt image_iff openin_kification subtopology_subtopology compactin_subtopology)

```

```

lemma proper_map_from_kification:
  assumes k_space Y
  shows proper_map (kification X) Y f  $\iff$  proper_map X Y f (is ?lhs  $\iff$  ?rhs)
proof
  assume ?lhs then show ?rhs
  by (simp add: closed_map_def closedin_kification_finer proper_map_alt)
next
  assume R: ?rhs
  have compactin Y K  $\implies$  compactin X {x  $\in$  topspace X. f x  $\in$  K} for K
  using R proper_map_alt by auto
  with R show ?lhs
  by (simp add: assms proper_map_into_k_space_eq subtopology_kification_compact)
qed

```

```

lemma perfect_map_from_kification:
  [[k_space Y; perfect_map X Y f]]  $\implies$  perfect_map(kification X) Y f

```

1614

by (simp add: continuous_map_from_kification perfect_map_def proper_map_from_kification)

lemma *k_space_perfect_map_image_eq*:

assumes *Hausdorff_space X perfect_map X Y f*

shows $k_space\ X \longleftrightarrow k_space\ Y$

proof

show $k_space\ X \implies k_space\ Y$

using *k_space_perfect_map_image* **assms** **by** *blast*

assume $k_space\ Y$

have *homeomorphic_map (kification X) X id*

unfolding *homeomorphic_eq_injective_perfect_map*

proof (intro *conjI perfect_map_from_composition_right* [where $f = id$])

show *perfect_map (kification X) Y (f ∘ id)*

by (simp add: $\langle k_space\ Y \rangle$ *assms*(2) *perfect_map_from_kification*)

show *continuous_map (kification X) X id*

by (simp add: *continuous_map_from_kification*)

qed (use *assms perfect_map_def* **in** *auto*)

then show $k_space\ X$

using *homeomorphic_k_space homeomorphic_space* **by** *blast*

qed

6.11.6 One-point compactifications and the Alexandroff extension construction

lemma *one_point_compactification_dense*:

$\llbracket compact_space\ X; \neg\ compactin\ X\ (topspace\ X - \{a\}) \rrbracket \implies X\ closure_of\ (topspace\ X - \{a\}) = topspace\ X$

unfolding *closure_of_complement*

by (*metis Diff_empty closedin_compact_space interior_of_eq_empty openin_closedin_eq subset_singletonD*)

lemma *one_point_compactification_interior*:

$\llbracket compact_space\ X; \neg\ compactin\ X\ (topspace\ X - \{a\}) \rrbracket \implies X\ interior_of\ \{a\} = \{\}$

by (simp add: *interior_of_eq_empty_complement one_point_compactification_dense*)

proposition *kc_space_one_point_compactification_gen*:

assumes *compact_space X*

shows $kc_space\ X \longleftrightarrow$

$openin\ X\ (topspace\ X - \{a\}) \wedge (\forall K. compactin\ X\ K \wedge a \notin K \longrightarrow closedin\ X\ K) \wedge$

$k_space\ (subtopology\ X\ (topspace\ X - \{a\})) \wedge kc_space\ (subtopology\ X\ (topspace\ X - \{a\}))$

(is *?lhs* \longleftrightarrow *?rhs*)

proof

assume *L*: *?lhs* **show** *?rhs*

proof (intro *conjI strip*)

show *openin X (topspace X - {a})*

using *L kc_imp_t1_space t1_space_openin_delete_alt* **by** *auto*

```

then show  $k\_space$  (subtopology  $X$  (topspace  $X - \{a\}$ ))
  by (simp add:  $L$  assms  $k\_space\_open\_subtopology\_aux$ )
show  $closedin$   $X$   $k$  if  $compactin$   $X$   $k \wedge a \notin k$  for  $k :: 'a$  set
  using  $L$   $kc\_space\_def$  that by blast
show  $kc\_space$  (subtopology  $X$  (topspace  $X - \{a\}$ ))
  by (simp add:  $L$   $kc\_space\_subtopology$ )
qed
next
assume  $R$ : ?rhs
show ?lhs
  unfolding  $kc\_space\_def$ 
proof (intro strip)
  fix  $S$ 
  assume  $compactin$   $X$   $S$ 
  then have  $S \subseteq topspace$   $X$ 
    by (simp add:  $compactin\_subset\_topspace$ )
  show  $closedin$   $X$   $S$ 
  proof (cases  $a \in S$ )
  case  $True$ 
  then have  $topspace$   $X - S = topspace$   $X - \{a\} - (S - \{a\})$ 
    by auto
  moreover have  $openin$   $X$  ( $topspace$   $X - \{a\} - (S - \{a\})$ )
  proof (rule  $openin\_trans\_full$ )
  show  $openin$  (subtopology  $X$  (topspace  $X - \{a\}$ )) ( $topspace$   $X - \{a\} - (S - \{a\})$ )
  proof
  show  $openin$  (subtopology  $X$  (topspace  $X - \{a\}$ )) ( $topspace$   $X - \{a\}$ )
    using  $R$   $openin\_open\_subtopology$  by blast
  have  $closedin$  (subtopology  $X$  ( $(topspace$   $X - \{a\}) \cap K$ )) ( $K \cap (S - \{a\})$ )
    if  $compactin$   $X$   $K$   $K \subseteq topspace$   $X - \{a\}$  for  $K$ 
  proof (intro  $closedin\_subset\_topspace$ )
  show  $closedin$   $X$  ( $K \cap (S - \{a\})$ )
    using that
    by (metis  $IntD1$   $Int\_insert\_right\_if0$   $R$   $True$   $\langle compactin$   $X$   $S \rangle$ 
 $closed\_Int\_compactin$   $insert\_Diff$   $subset\_Diff\_insert$ )
  qed (use that in auto)
  moreover have  $k\_space$  (subtopology  $X$  (topspace  $X - \{a\}$ ))
    using  $R$  by blast
  moreover have  $S - \{a\} \subseteq topspace$   $X \wedge S - \{a\} \subseteq topspace$   $X - \{a\}$ 
    using  $\langle S \subseteq topspace$   $X \rangle$  by auto
  ultimately show  $closedin$  (subtopology  $X$  (topspace  $X - \{a\}$ )) ( $S - \{a\}$ )
    using  $\langle S \subseteq topspace$   $X \rangle$   $True$ 
  by (simp add:  $k\_space\_def$   $compactin\_subtopology$   $subtopology\_subtopology$ )
  qed
show  $openin$   $X$  ( $topspace$   $X - \{a\}$ )
  by (simp add:  $R$ )
qed
ultimately show ?thesis
  by (simp add:  $\langle S \subseteq topspace$   $X \rangle$   $closedin\_def$ )

```

1616

```
next
  case False
  then show ?thesis
  by (simp add: R ⟨compactin X S⟩)
qed
qed
qed
```

```
inductive Alexandroff_open for X where
  base: openin X U  $\implies$  Alexandroff_open X (Some ' U)
| ext: [compactin X C; closedin X C]  $\implies$  Alexandroff_open X (insert None (Some
' (topspace X - C)))
```

```
hide_fact (open) base ext
```

```
lemma Alexandroff_open_iff: Alexandroff_open X S  $\longleftrightarrow$ 
  ( $\exists U. (S = \text{Some ' } U \wedge \text{openin } X U) \vee (S = \text{insert None (Some ' (topspace } X
- U)) \wedge \text{compactin } X U \wedge \text{closedin } X U)$ )
  by (meson Alexandroff_open.cases Alexandroff_open.ext Alexandroff_open.base)
```

```
lemma Alexandroff_open_Un_aux:
  assumes U: openin X U and Alexandroff_open X T
  shows Alexandroff_open X (Some ' U  $\cup$  T)
  using ⟨Alexandroff_open X T⟩
proof (induction rule: Alexandroff_open.induct)
  case (base V)
  then show ?case
  by (metis Alexandroff_open.base U image_Un openin_Un)
next
  case (ext C)
  have  $U \subseteq \text{topspace } X$ 
  by (simp add: U openin_subset)
  then have eq:  $\text{Some ' } U \cup \text{insert None (Some ' (topspace } X - C)) = \text{insert
None (Some ' (topspace } X - (C \cap (\text{topspace } X - U))))$ 
  by force
  have closedin X ( $C \cap (\text{topspace } X - U)$ )
  using U ext.hyps(2) by blast
  moreover
  have compactin X ( $C \cap (\text{topspace } X - U)$ )
  using U compact_Int_closedin ext.hyps(1) by blast
  ultimately show ?case
  unfolding eq using Alexandroff_open.ext by blast
qed
```

```
lemma Alexandroff_open_Un:
  assumes Alexandroff_open X S and Alexandroff_open X T
  shows Alexandroff_open X (S  $\cup$  T)
  using assms
```

```

proof (induction rule: Alexandroff_open.induct)
  case (base U)
  then show ?case
    by (simp add: Alexandroff_open_Un_aux)
next
  case (ext C)
  then show ?case
    by (smt (verit, best) Alexandroff_open_Un_aux Alexandroff_open_iff Un_commute
  Un_insert_left closedin_def insert_absorb2)
qed

```

```

lemma Alexandroff_open_Int_aux:
  assumes U: openin X U and Alexandroff_open X T
  shows Alexandroff_open X (Some ' U  $\cap$  T)
  using ‹Alexandroff_open X T›
proof (induction rule: Alexandroff_open.induct)
  case (base V)
  then show ?case
    by (metis Alexandroff_open.base U image_Int inj_Some openin_Int)
next
  case (ext C)
  have eq: Some ' U  $\cap$  insert None (Some ' (topspace X - C)) = Some ' (topspace
  X - (C  $\cup$  (topspace X - U)))
  by force
  have openin X (topspace X - (C  $\cup$  (topspace X - U)))
  using U ext.hyps(2) by blast
  then show ?case
    unfolding eq using Alexandroff_open.base by blast
qed

```

```

proposition istopology_Alexandroff_open: istopology (Alexandroff_open X)
  unfolding istopology_def
proof (intro conjI strip)
  fix S T
  assume Alexandroff_open X S and Alexandroff_open X T
  then show Alexandroff_open X (S  $\cap$  T)
  proof (induction rule: Alexandroff_open.induct)
  case (base U)
  then show ?case
    using Alexandroff_open_Int_aux by blast
next
  case EC: (ext C)
  show ?case
    using ‹Alexandroff_open X T›
  proof (induction rule: Alexandroff_open.induct)
  case (base V)
  then show ?case
    by (metis Alexandroff_open.ext Alexandroff_open_Int_aux EC.hyps inf_commute)
next

```

```

    case (ext D)
      have eq: insert None (Some ' (topspace X - C)) ∩ insert None (Some '
(topspace X - D))
        = insert None (Some ' (topspace X - (C ∪ D)))
      by auto
      show ?case
      unfolding eq
      by (simp add: Alexandroff_open.ext EC.hyps closedin_Un compactin_Un
ext.hyps)
    qed
  qed
next
fix  $\mathcal{K}$ 
assume §:  $\forall K \in \mathcal{K}. \text{Alexandroff\_open } X K$ 
show Alexandroff_open X ( $\bigcup \mathcal{K}$ )
proof (cases None  $\in \bigcup \mathcal{K}$ )
  case True
    have  $\forall K \in \mathcal{K}. \exists U. (\text{openin } X U \wedge K = \text{Some ' } U) \vee (K = \text{insert None (Some '}$ 
' (topspace X - U))  $\wedge \text{compactin } X U \wedge \text{closedin } X U)$ 
      by (metis § Alexandroff_open_iff)
    then obtain U where U:
       $\bigwedge K. K \in \mathcal{K} \implies \text{openin } X (U K) \wedge K = \text{Some ' } (U K)$ 
       $\vee (K = \text{insert None (Some ' (topspace X - U K))} \wedge \text{compactin } X$ 
(U K)  $\wedge \text{closedin } X (U K))$ 
      by metis
    define  $\mathcal{KN}$  where  $\mathcal{KN} \equiv \{K \in \mathcal{K}. \text{None} \in K\}$ 
    define A where  $A \equiv \bigcup_{K \in \mathcal{K} - \mathcal{KN}} U K$ 
    define B where  $B \equiv \bigcap_{K \in \mathcal{KN}} U K$ 
    have U1:  $\bigwedge K. K \in \mathcal{K} - \mathcal{KN} \implies \text{openin } X (U K) \wedge K = \text{Some ' } (U K)$ 
      using U  $\mathcal{KN\_def}$  by auto
    have U2:  $\bigwedge K. K \in \mathcal{KN} \implies K = \text{insert None (Some ' (topspace X - U K))}$ 
 $\wedge \text{compactin } X (U K) \wedge \text{closedin } X (U K)$ 
      using U  $\mathcal{KN\_def}$  by auto
    have eqA:  $\bigcup (\mathcal{K} - \mathcal{KN}) = \text{Some ' } A$ 
    proof
      show  $\bigcup (\mathcal{K} - \mathcal{KN}) \subseteq \text{Some ' } A$ 
        by (metis A_def Sup_le_iff U1 UN_upper subset_image_iff)
      show  $\text{Some ' } A \subseteq \bigcup (\mathcal{K} - \mathcal{KN})$ 
        using A_def U1 by blast
    qed
    have eqB:  $\bigcup \mathcal{KN} = \text{insert None (Some ' (topspace X - B))}$ 
      using U2 True
      by (auto simp: B_def image_iff  $\mathcal{KN\_def}$ )
    have  $\bigcup \mathcal{K} = \bigcup \mathcal{KN} \cup \bigcup (\mathcal{K} - \mathcal{KN})$ 
      by (auto simp:  $\mathcal{KN\_def}$ )
    then have eq:  $\bigcup \mathcal{K} = (\text{Some ' } A) \cup (\text{insert None (Some ' (topspace X - B))})$ 
      by (simp add: eqA eqB Un_commute)
    show ?thesis
    unfolding eq

```

```

proof (intro Alexandroff_open_Un Alexandroff_open.intros)
  show openin X A
    using A_def U1 by blast
  show closedin X B
    unfolding B_def using U2 True KN_def by auto
  show compactin X B
  by (metis B_def U2 eqB Inf_lower Union_iff ‹closedin X B› closed_compactin
imageI insertI1)
  qed
next
case False
then have  $\forall K \in \mathcal{K}. \exists U. \text{openin } X \ U \wedge K = \text{Some } 'U$ 
  by (metis Alexandroff_open.simps UnionI § insertCI)
then obtain U where U:  $\forall K \in \mathcal{K}. \text{openin } X \ (U \ K) \wedge K = \text{Some } '(U \ K)$ 
  by metis
then have eq:  $\bigcup \mathcal{K} = \text{Some } '(\bigcup K \in \mathcal{K}. U \ K)$ 
  using image_iff by fastforce
show ?thesis
  unfolding eq by (simp add: U Alexandroff_open.base openin_clauses(3))
qed
qed

```

definition Alexandroff_compactification **where**

$\text{Alexandroff_compactification } X \equiv \text{topology } (\text{Alexandroff_open } X)$

lemma openin_Alexandroff_compactification:

```

openin(Alexandroff_compactification X) V  $\longleftrightarrow$ 
  ( $\exists U. \text{openin } X \ U \wedge V = \text{Some } 'U$ )  $\vee$ 
  ( $\exists C. \text{compactin } X \ C \wedge \text{closedin } X \ C \wedge V = \text{insert None } (\text{Some } '(topspace$ 
X - C)))
by (auto simp: Alexandroff_compactification_def istopology_Alexandroff_open
Alexandroff_open.simps)

```

lemma topspace_Alexandroff_compactification [simp]:

$\text{topspace}(\text{Alexandroff_compactification } X) = \text{insert None } (\text{Some } ' \text{topspace } X)$
(is ?lhs = ?rhs)

proof

show ?lhs \subseteq ?rhs

by (force simp: topspace_def openin_Alexandroff_compactification)

have None \in topspace (Alexandroff_compactification X)

by (meson closedin_empty compactin_empty insert_subset openin_Alexandroff_compactification
openin_subset)

moreover have Some $x \in$ topspace (Alexandroff_compactification X)

if $x \in$ topspace X **for** x

by (meson that imageI openin_Alexandroff_compactification openin_subset
openin_topspace subsetD)

ultimately show ?rhs \subseteq ?lhs

1620

by (auto simp: image_subset_iff)
qed

lemma *closedin_Alexandroff_compactification*:

$\text{closedin } (\text{Alexandroff_compactification } X) C \longleftrightarrow$
 $(\exists K. \text{compactin } X K \wedge \text{closedin } X K \wedge C = \text{Some } 'K) \vee$
 $(\exists U. \text{openin } X U \wedge C = \text{topspace}(\text{Alexandroff_compactification } X) - \text{Some}$
 $'U)$
(is ?lhs \longleftrightarrow ?rhs)

proof

show ?lhs \implies ?rhs

apply (clarsimp simp: closedin_def openin_Alexandroff_compactification)

by (smt (verit) Diff_Diff_Int_None_notin_image_Some_image_set_diff_inf.absorb_iff2
inj_Some_insert_Diff_if_subset_insert)

show ?rhs \implies ?lhs

using openin_subset

by (fastforce simp: closedin_def openin_Alexandroff_compactification)

qed

lemma *openin_Alexandroff_compactification_image_Some* [simp]:

$\text{openin}(\text{Alexandroff_compactification } X) (\text{Some } 'U) \longleftrightarrow \text{openin } X U$

by (auto simp: openin_Alexandroff_compactification inj_image_eq_iff)

lemma *closedin_Alexandroff_compactification_image_Some* [simp]:

$\text{closedin } (\text{Alexandroff_compactification } X) (\text{Some } 'K) \longleftrightarrow \text{compactin } X K \wedge$
 $\text{closedin } X K$

by (auto simp: closedin_Alexandroff_compactification inj_image_eq_iff)

lemma *open_map_Some*: $\text{open_map } X (\text{Alexandroff_compactification } X) \text{Some}$

using open_map_def openin_Alexandroff_compactification by blast

lemma *continuous_map_Some*: $\text{continuous_map } X (\text{Alexandroff_compactification } X) \text{Some}$

unfolding continuous_map_def

proof (intro conjI strip)

fix U

assume openin (Alexandroff_compactification X) U

then consider V where openin X V U = Some 'V

| C where compactin X C closedin X C U = insert None (Some ' (topspace X
- C))

by (auto simp: openin_Alexandroff_compactification)

then show openin X {x \in topspace X. Some x \in U}

proof cases

case 1

then show ?thesis

using openin_subopen openin_subset by fastforce

next

case 2

then show ?thesis


```

    by (simp add: closedin_def image_iff set_diff_eq)
  qed
qed auto

```

```

lemma embedding_map_Some: embedding_map X (Alexandroff_compactification
X) Some
  by (simp add: continuous_map_Some injective_open_imp_embedding_map open_map_Some)

```

```

lemma compact_space_Alexandroff_compactification [simp]:
  compact_space (Alexandroff_compactification X)
proof (clarsimp simp: compact_space_alt image_subset_iff)
  fix  $\mathcal{U}$  U
  assume ope [rule_format]:  $\forall U \in \mathcal{U}. \text{openin } (\text{Alexandroff\_compactification } X) U$ 
    and cover:  $\forall x \in \text{topspace } X. \exists X \in \mathcal{U}. \text{Some } x \in X$ 
    and  $U \in \mathcal{U} \text{ None} \in U$ 
  then have Usub:  $U \subseteq \text{insert None } (\text{Some } \text{'topspace } X)$ 
    by (metis openin_subset topspace_Alexandroff_compactification)
  with ope [OF  $\langle U \in \mathcal{U} \rangle$ ]  $\langle \text{None} \in U \rangle$ 
  obtain C where C: compactin X C  $\wedge$  closedin X C  $\wedge$ 
    insert None (Some 'topspace X) - U = Some ' C
  by (auto simp: openin_closedin closedin_Alexandroff_compactification)
  then have D: compactin (Alexandroff_compactification X) (insert None (Some
'topspace X) - U)
    by (metis continuous_map_Some image_compactin)
  consider V where openin X V U = Some ' V
    | C where compactin X C closedin X C U = insert None (Some ' (topspace X
- C))
  using ope [OF  $\langle U \in \mathcal{U} \rangle$ ] by (auto simp: openin_Alexandroff_compactification)
  then show  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge (\exists X \in \mathcal{F}. \text{None} \in X) \wedge (\forall x \in \text{topspace } X.
\exists X \in \mathcal{F}. \text{Some } x \in X)$ 
proof cases
  case 1
  then show ?thesis
    using  $\langle \text{None} \in U \rangle$  by blast
  next
  case 2
  obtain  $\mathcal{F}$  where finite  $\mathcal{F} \mathcal{F} \subseteq \mathcal{U}$  insert None (Some 'topspace X) - U  $\subseteq \bigcup \mathcal{F}$ 
    by (smt (verit, del_insts) C D Union_iff compactinD compactin_subset_topspace
cover_image_subset_iff ope subsetD)
  with  $\langle U \in \mathcal{U} \rangle$  show ?thesis
    by (rule_tac x=insert U  $\mathcal{F}$  in exI) auto
  qed
qed

```

```

lemma topspace_Alexandroff_compactification_delete:
  topspace (Alexandroff_compactification X) - {None} = Some 'topspace X
  by simp

```

lemma *Alexandroff_compactification_dense*:
assumes \neg *compact_space* X
shows (*Alexandroff_compactification* X) *closure_of* (*Some* ‘ *topspace* X) =
topspace(*Alexandroff_compactification* X)
unfolding *topspace_Alexandroff_compactification_delete* [*symmetric*]
proof (*intro one_point_compactification_dense*)
show \neg *compactin* (*Alexandroff_compactification* X) (*topspace* (*Alexandroff_compactification*
 X) - {*None*})
using *assms compact_space_proper_map_preimage compact_space_subtopology*
embedding_map_Some embedding_map_def homeomorphic_imp_proper_map by
fastforce
qed *auto*

lemma *t0_space_one_point_compactification*:
assumes *compact_space* $X \wedge$ *openin* X (*topspace* X - { a })
shows *t0_space* $X \longleftrightarrow$ *t0_space* (*subtopology* X (*topspace* X - { a }))
(is *?lhs* \longleftrightarrow *?rhs)
proof
show *?lhs* \implies *?rhs*
using *t0_space_subtopology by blast*
show *?rhs* \implies *?lhs*
using *assms*
unfolding *t0_space_def by (bestsimp simp flip: Int_Diff dest: openin_trans_full)*
qed*

lemma *t0_space_Alexandroff_compactification* [*simp*]:
t0_space (*Alexandroff_compactification* X) \longleftrightarrow *t0_space* X
using *t0_space_one_point_compactification* [*of Alexandroff_compactification X*
None]
using *embedding_map_Some embedding_map_imp_homeomorphic_space home-*
omorphic_t0_space by fastforce

lemma *t1_space_one_point_compactification*:
assumes Xa : *openin* X (*topspace* X - { a })
and \S : $\bigwedge K. \llbracket$ *compactin* (*subtopology* X (*topspace* X - { a })) K ; *closedin*
(*subtopology* X (*topspace* X - { a })) K $\rrbracket \implies$ *closedin* X K
shows *t1_space* $X \longleftrightarrow$ *t1_space* (*subtopology* X (*topspace* X - { a })) **(is** *?lhs*
 \longleftrightarrow *?rhs)
proof
show *?lhs* \implies *?rhs*
using *t1_space_subtopology by blast*
assume R : *?rhs*
show *?lhs*
unfolding *t1_space_closedin_singleton*
proof (*intro strip*)
fix x
assume $x \in$ *topspace* X
show *closedin* X { x }*

```

proof (cases x=a)
  case True
  then show ?thesis
    using ⟨x ∈ topspace X⟩ Xa closedin_def by blast
  next
  case False
  show ?thesis
    by (simp add: § False R ⟨x ∈ topspace X⟩ closedin_t1_singleton)
qed
qed
qed

```

```

lemma closedin_Alexandroff_I:
  assumes compactin (Alexandroff_compactification X) K K ⊆ Some ‘ topspace X
  shows closedin (Alexandroff_compactification X) T K = T ∩ Some ‘ topspace X
proof –
  obtain S where S: S ⊆ topspace X K = Some ‘ S
  by (meson ⟨K ⊆ Some ‘ topspace X⟩ subset_imageE)
  with assms have compactin X S
  by (metis compactin_subtopology embedding_map_Some embedding_map_def
  homeomorphic_map_compactness)
  moreover have closedin X S
  using assms S
  by (metis closedin_subtopology embedding_map_Some embedding_map_def
  homeomorphic_map_closedness)
  ultimately show ?thesis
  by (simp add: S)
qed

```

```

lemma t1_space_Alexandroff_compactification [simp]:
  t1_space (Alexandroff_compactification X) ↔ t1_space X
proof –
  have openin (Alexandroff_compactification X) (topspace (Alexandroff_compactification X) – {None})
  by auto
  then show ?thesis
  using t1_space_one_point_compactification [of Alexandroff_compactification X None]
  using embedding_map_Some embedding_map_imp_homeomorphic_space homeomorphic_t1_space
  by (fastforce simp: compactin_subtopology closedin_Alexandroff_I closedin_subtopology)
qed

```

```

lemma kc_space_one_point_compactification:
  assumes compact_space X

```

and *ope*: *openin* X (*topspace* $X - \{a\}$)
and \S : $\bigwedge K. \llbracket \text{compactin} (\text{subtopology } X (\text{topspace } X - \{a\})) K; \text{closedin} (\text{subtopology } X (\text{topspace } X - \{a\})) K \rrbracket$
 $\implies \text{closedin } X K$
shows *kc_space* $X \longleftrightarrow$
 $k_space (\text{subtopology } X (\text{topspace } X - \{a\})) \wedge kc_space (\text{subtopology } X (\text{topspace } X - \{a\}))$
proof –
have *closedin* $X K$
if *kc_space* (*subtopology* X (*topspace* $X - \{a\}$)) **and** *compactin* $X K$ $a \notin K$ **for** K
using that **unfolding** *kc_space_def*
by (*metis* \S *Diff_empty_compactin_subspace_compactin_subtopology_subset_Diff_insert*)
then show *?thesis*
by (*metis* $\langle \text{compact_space } X \rangle$ *kc_space_one_point_compactification_gen_ope*)
qed

lemma *kc_space_Alexandroff_compactification*:

kc_space(*Alexandroff_compactification* X) \longleftrightarrow (*k_space* $X \wedge kc_space$ X) (**is** *kc_space* $?Y = _$)

proof –

have *kc_space* (*Alexandroff_compactification* X) \longleftrightarrow
 $(k_space (\text{subtopology } ?Y (\text{topspace } ?Y - \{None\})) \wedge kc_space (\text{subtopology } ?Y (\text{topspace } ?Y - \{None\})))$
by (*rule* *kc_space_one_point_compactification*) (*auto simp: compactin_subtopology closedin_subtopology closedin_Alexandroff_I*)
also have $\dots \longleftrightarrow k_space$ $X \wedge kc_space$ X
using *embedding_map_Some_embedding_map_imp_homeomorphic_space_homeomorphic_k_space_homeomorphic_kc_space* **by** *simp blast*
finally show *?thesis* .
qed

proposition *regular_space_one_point_compactification*:

assumes *compact_space* X **and** *ope*: *openin* X (*topspace* $X - \{a\}$)
and \S : $\bigwedge K. \llbracket \text{compactin} (\text{subtopology } X (\text{topspace } X - \{a\})) K; \text{closedin} (\text{subtopology } X (\text{topspace } X - \{a\})) K \rrbracket \implies \text{closedin } X K$
shows *regular_space* $X \longleftrightarrow$
 $\text{regular_space} (\text{subtopology } X (\text{topspace } X - \{a\})) \wedge \text{locally_compact_space} (\text{subtopology } X (\text{topspace } X - \{a\}))$
(is *?lhs* \longleftrightarrow *?rhs)*

proof

show *?lhs* \implies *?rhs*
using *assms(1) compact_imp_locally_compact_space_locally_compact_space_open_subset_ope_regular_space_subtopology* **by** *blast*
assume R : *?rhs*
let $?Xa = \text{subtopology } X (\text{topspace } X - \{a\})$
show *?lhs*
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*

```

imp_conjL
proof (intro strip)
  fix W x
  assume openin X W and x ∈ W
  show ∃ U V. openin X U ∧ closedin X V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆ W
proof (cases x=a)
  case True
  have compactin ?Xa (topspace X - W)
  using ⟨openin X W⟩ assms(1) closedin_compact_space
  by (metis Diff_mono True ⟨x ∈ W⟩ compactin_subtopology_empty_subsetI
insert_subset openin_closedin_eq order_refl)
  then obtain V K where V: openin ?Xa V and K: compactin ?Xa K closedin
?Xa K and topspace X - W ⊆ V V ⊆ K
  by (metis locally_compact_space_compact_closed_compact R)
  show ?thesis
proof (intro exI conjI)
  show openin X (topspace X - K)
  using § K by blast
  show closedin X (topspace X - V)
  using V ope openin_trans_full by blast
  show x ∈ topspace X - K
proof (rule)
  show x ∈ topspace X
  using ⟨openin X W⟩ ⟨x ∈ W⟩ openin_subset by blast
  show x ∉ K
  using K True closedin_imp_subset by blast
qed
show topspace X - K ⊆ topspace X - V
  by (simp add: Diff_mono ⟨V ⊆ K⟩)
show topspace X - V ⊆ W
  using ⟨topspace X - W ⊆ V⟩ by auto
qed
next
case False
have openin ?Xa ((topspace X - {a}) ∩ W)
  using ⟨openin X W⟩ openin_subtopology_Int2 by blast
moreover have x ∈ (topspace X - {a}) ∩ W
  using ⟨openin X W⟩ ⟨x ∈ W⟩ openin_subset False by blast
ultimately obtain U V where openin ?Xa U compactin ?Xa V closedin ?Xa
V
  x ∈ U U ⊆ V V ⊆ (topspace X - {a}) ∩ W
  using R locally_compact_regular_space_neighbourhood_base neighbour-
hood_base_of
  by (metis (no_types, lifting))
then show ?thesis
  by (meson § le_infE ope openin_trans_full)
qed
qed
qed

```

lemma *regular_space_Alexandroff_compactification*:
 $\text{regular_space}(\text{Alexandroff_compactification } X) \longleftrightarrow \text{regular_space } X \wedge \text{locally_compact_space } X$
 (is *regular_space* ?Y = ?rhs)
proof –
have *regular_space* ?Y \longleftrightarrow
 $\text{regular_space}(\text{subtopology } ?Y (\text{topspace } ?Y - \{\text{None}\})) \wedge \text{locally_compact_space}(\text{subtopology } ?Y (\text{topspace } ?Y - \{\text{None}\}))$
by (rule *regular_space_one_point_compactification*) (auto simp: *compactin_subtopology* *closedin_subtopology* *closedin_Alexandroff_I*)
also have ... $\longleftrightarrow \text{regular_space } X \wedge \text{locally_compact_space } X$
by (metis *embedding_map_Some* *embedding_map_imp_homeomorphic_space* *homeomorphic_locally_compact_space* *homeomorphic_regular_space* *topspace_Alexandroff_compactification_delete*)
finally show ?thesis .
qed

lemma *Hausdorff_space_one_point_compactification*:
assumes *compact_space* X **and** *openin* X (topspace X - {a})
and $\bigwedge K. [\text{compactin}(\text{subtopology } X (\text{topspace } X - \{a\})) K; \text{closedin}(\text{subtopology } X (\text{topspace } X - \{a\})) K] \implies \text{closedin } X K$
shows *Hausdorff_space* X \longleftrightarrow
 $\text{Hausdorff_space}(\text{subtopology } X (\text{topspace } X - \{a\})) \wedge \text{locally_compact_space}(\text{subtopology } X (\text{topspace } X - \{a\}))$
 (is ?lhs \longleftrightarrow ?rhs)
proof
show ?rhs if ?lhs
proof –
have *locally_compact_space* (subtopology X (topspace X - {a}))
using *assms that compact_imp_locally_compact_space* *locally_compact_space_open_subset*

by *blast*
with that show ?rhs
by (simp add: *Hausdorff_space_subtopology*)
qed
next
show ?lhs \implies ?rhs
by (metis *assms locally_compact_Hausdorff_or_regular* *regular_space_one_point_compactification* *regular_t1_eq_Hausdorff_space* *t1_space_one_point_compactification*)
qed

lemma *Hausdorff_space_Alexandroff_compactification*:
 $\text{Hausdorff_space}(\text{Alexandroff_compactification } X) \longleftrightarrow \text{Hausdorff_space } X \wedge \text{locally_compact_space } X$
by (meson *compact_Hausdorff_imp_regular_space* *compact_space_Alexandroff_compactification*)

```

locally_compact_Hausdorff_or_regular regular_space_Alexandroff_compactification
regular_t1_eq_Hausdorff_space t1_space_Alexandroff_compactification)

lemma completely_regular_space_Alexandroff_compactification:
  completely_regular_space(Alexandroff_compactification X)  $\longleftrightarrow$ 
    completely_regular_space X  $\wedge$  locally_compact_space X
by (metis regular_space_Alexandroff_compactification completely_regular_eq_regular_space
  compact_imp_locally_compact_space compact_space_Alexandroff_compactification)

proposition Hausdorff_space_one_point_compactification_asymmetric_prod:
  assumes compact_space X
  shows Hausdorff_space X  $\longleftrightarrow$ 
    kc_space (prod_topology X (subtopology X (topspace X - {a})))  $\wedge$ 
    k_space (prod_topology X (subtopology X (topspace X - {a}))) (is ?lhs
 $\longleftrightarrow$  ?rhs)
proof (cases a  $\in$  topspace X)
  case True
  show ?thesis
  proof
    show ?rhs if ?lhs
    proof
      show kc_space (prod_topology X (subtopology X (topspace X - {a})))
      using Hausdorff_imp_kc_space kc_space_prod_topology_right kc_space_subtopology
that by blast
      show k_space (prod_topology X (subtopology X (topspace X - {a})))
      by (meson Hausdorff_imp_kc_space assms compact_imp_locally_compact_space
k_space_prod_topology_left
      kc_space_one_point_compactification_gen that)
    qed
  next
  assume R: ?rhs
  define D where D  $\equiv$  ( $\lambda x. (x,x)$ ) ' (topspace X - {a})
  show ?lhs
  proof (cases topspace X = {a})
  case True
  then show ?thesis
    by (simp add: Hausdorff_space_def)
  next
  case False
  with R True have kc_space X
  using kc_space_retraction_map_image [of prod_topology X (subtopology X
(topspace X - {a})) X fst]
  by (metis Diff_subset insert_Diff retraction_map_fst topspace_discrete_topology
topspace_subtopology_subset)
  have closedin (subtopology (prod_topology X (subtopology X (topspace X -
{a}))) K) (K  $\cap$  D)
  if compactin (prod_topology X (subtopology X (topspace X - {a}))) K for
K

```

```

proof (intro closedin_subtopology_Int_subset[where V=K] closedin_subset_topspace)
  show fst ' K × snd ' K ∩ D ⊆ fst ' K × snd ' K K ⊆ fst ' K × snd ' K
    by force+
  have eq: (fst ' K × snd ' K ∩ D) = ((λx. (x,x)) ' (fst ' K ∩ snd ' K))
    using compactin_subset_topspace that by (force simp: D_def image_iff)
  have compactin (prod_topology X (subtopology X (topspace X - {a}))) (fst
' K × snd ' K ∩ D)
    unfolding eq
  proof (rule image_compactin [of subtopology X (topspace X - {a})])
    have compactin X (fst ' K) compactin X (snd ' K)
    by (meson compactin_subtopology_continuous_map_fst continuous_map_snd
image_compactin that)+
    moreover have fst ' K ∩ snd ' K ⊆ topspace X - {a}
      using compactin_subset_topspace that by force
    ultimately
    show compactin (subtopology X (topspace X - {a})) (fst ' K ∩ snd ' K)
      unfolding compactin_subtopology
      by (meson ⟨kc_space X⟩ closed_Int_compactin kc_space_def)
    show continuous_map (subtopology X (topspace X - {a})) (prod_topology
X (subtopology X (topspace X - {a}))) (λx. (x,x))
      by (simp add: continuous_map_paired)
    qed
  then show closedin (prod_topology X (subtopology X (topspace X - {a})))
(fst ' K × snd ' K ∩ D)
    using R compactin_imp_closedin_gen by blast
  qed
  moreover have D ⊆ topspace X × (topspace X ∩ (topspace X - {a}))
    by (auto simp: D_def)
  ultimately have *: closedin (prod_topology X (subtopology X (topspace X -
{a}))) D
    using R by (auto simp: k_space)
  have x=y
  if x ∈ topspace X y ∈ topspace X
    and §: ∧ T. [(x,y) ∈ T; openin (prod_topology X X) T] ==> ∃ z ∈ topspace
X. (z,z) ∈ T for x y
  proof (cases x=a ∧ y=a)
  case False
  then consider x≠a | y≠a
    by blast
  then show ?thesis
  proof cases
  case 1
  have ∃ z ∈ topspace X - {a}. (z,z) ∈ T
    if (y,x) ∈ T openin (prod_topology X (subtopology X (topspace X -
{a}))) T for T
  proof -
  have (x,y) ∈ {z ∈ topspace (prod_topology X X). (snd z,fst z) ∈ T ∩
topspace X × (topspace X - {a})}
    by (simp add: 1 ⟨x ∈ topspace X⟩ ⟨y ∈ topspace X⟩ that)

```



```

moreover have openin (prod_topology X X) {z ∈ topspace (prod_topology
X X). (snd z, fst z) ∈ T ∩ topspace X × (topspace X - {a})}
  proof (rule openin_continuous_map_preimage)
    show continuous_map (prod_topology X X) (prod_topology X X) (λx.
(snd x, fst x))
      by (simp add: continuous_map_fst continuous_map_pairedI contin-
uous_map_snd)
    have openin (prod_topology X X) (topspace X × (topspace X - {a}))
    using ⟨kc_space X⟩ assms kc_space_one_point_compactification_gen
openin_prod_Times_iff by fastforce
    moreover have openin (prod_topology X X) T
    using kc_space_one_point_compactification_gen [OF ⟨compact_space
X⟩] ⟨kc_space X⟩ that
    by (metis openin_prod_Times_iff openin_topspace openin_trans_full
prod_topology_subtopology(2))
    ultimately show openin (prod_topology X X) (T ∩ topspace X ×
(topspace X - {a}))
      by blast
    qed
    ultimately show ?thesis
      by (smt (verit) § Int_iff fst_conv mem_Collect_eq mem_Sigma_iff
snd_conv)
    qed
    then have (y,x) ∈ prod_topology X (subtopology X (topspace X - {a}))
closure_of D
      by (simp add: 1 D_def in_closure_of that)
    then show ?thesis
      using that * D_def closure_of_closedin by fastforce
  next
  case 2
  have ∃ z ∈ topspace X - {a}. (z,z) ∈ T
    if (x,y) ∈ T openin (prod_topology X (subtopology X (topspace X -
{a}))) T for T
    proof -
      have openin (prod_topology X X) (topspace X × (topspace X - {a}))
      using ⟨kc_space X⟩ assms kc_space_one_point_compactification_gen
openin_prod_Times_iff by fastforce
      moreover have XXT: openin (prod_topology X X) T
      using kc_space_one_point_compactification_gen [OF ⟨compact_space
X⟩] ⟨kc_space X⟩ that
      by (metis openin_prod_Times_iff openin_topspace openin_trans_full
prod_topology_subtopology(2))
      ultimately have openin (prod_topology X X) (T ∩ topspace X ×
(topspace X - {a}))
        by blast
      then show ?thesis
        by (smt (verit) § Diff_subset XXT mem_Sigma_iff openin_subset
subsetD that topspace_prod_topology topspace_subtopology_subset)
    qed

```

```

      then have  $(x,y) \in \text{prod\_topology } X (\text{subtopology } X (\text{topspace } X - \{a\}))$ 
closure_of D
      by (simp add: 2 D_def in_closure_of that)
      then show ?thesis
      using that * D_def closure_of_closedin by fastforce
    qed
  qed auto
  then show ?thesis
  unfolding Hausdorff_space_closedin_diagonal closure_of_subset_eq [symmetric]

      by (force simp: closure_of_def)
    qed
  qed
next
  case False
  then show ?thesis
  by (simp add: assms compact_imp_k_space compact_space_prod_topology
kc_space_compact_prod_topology)
qed

```

lemma Hausdorff_space_Alexandroff_compactification_asymmetric_prod:

$$\text{Hausdorff_space}(\text{Alexandroff_compactification } X) \longleftrightarrow$$

$$kc_space(\text{prod_topology } (\text{Alexandroff_compactification } X) X) \wedge$$

$$k_space(\text{prod_topology } (\text{Alexandroff_compactification } X) X)$$

(is Hausdorff_space ?Y = ?rhs)

proof –

```

  have *: subtopology (Alexandroff_compactification X)
    (topspace (Alexandroff_compactification X) –
    {None}) homeomorphic_space X
  using embedding_map_Some embedding_map_imp_homeomorphic_space home-
omorphi_c_space_sym by fastforce
  have Hausdorff_space (Alexandroff_compactification X)  $\longleftrightarrow$ 
    (kc_space (prod_topology ?Y (subtopology ?Y (topspace ?Y – {None})))  $\wedge$ 
    k_space (prod_topology ?Y (subtopology ?Y (topspace ?Y – {None}))))
  by (rule Hausdorff_space_one_point_compactification_asymmetric_prod) (auto
simp: compactin_subtopology closedin_subtopology closedin_Alexandroff_I)
  also have ...  $\longleftrightarrow$  ?rhs
  using homeomorphic_k_space homeomorphic_kc_space homeomorphic_space_prod_topology

      homeomorphic_space_refl * by blast
  finally show ?thesis .
qed

```

lemma kc_space_as_compactification_unique:

assumes kc_space X compact_space X

shows openin X U \longleftrightarrow

$$(\text{if } a \in U \text{ then } U \subseteq \text{topspace } X \wedge \text{compactin } X (\text{topspace } X - U)$$

$$\text{else openin } (\text{subtopology } X (\text{topspace } X - \{a\})) U)$$

```

proof (cases a ∈ U)
  case True
  then show ?thesis
  by (meson assms closedin_compact_space compactin_imp_closedin_gen openin_closedin_eq)
next
  case False
  then show ?thesis
  by (metis Diff_empty kc_space_one_point_compactification_gen openin_open_subtopology
openin_subset subset_Diff_insert assms)
qed

```

lemma *kc_space_as_compactification_unique_explicit:*

```

assumes kc_space X compact_space X
shows openin X U  $\longleftrightarrow$ 
  (if a ∈ U then U ⊆ topspace X ∧
    compactin (subtopology X (topspace X - {a})) (topspace X - U)
  ∧
    closedin (subtopology X (topspace X - {a})) (topspace X - U)
  else openin (subtopology X (topspace X - {a})) U)
apply (simp add: kc_space_subtopology compactin_imp_closedin_gen assms
compactin_subtopology cong: conj_cong)
by (metis Diff_mono assms bot.extremum insert_subset kc_space_as_compactification_unique
subset_refl)

```

lemma *Alexandroff_compactification_unique:*

```

assumes kc_space X compact_space X and a: a ∈ topspace X
shows Alexandroff_compactification (subtopology X (topspace X - {a})) homeomorphic_space X
  (is ?Y homeomorphic_space X)
proof -
  have [simp]: topspace X ∩ (topspace X - {a}) = topspace X - {a}
  by auto
  have [simp]: insert None (Some ' A) = insert None (Some ' B)  $\longleftrightarrow$  A = B
    insert None (Some ' A)  $\neq$  Some ' B for A B
  by auto
  have quotient_map X ?Y (λx. if x = a then None else Some x)
  unfolding quotient_map_def
  proof (intro conjI strip)
  show (λx. if x = a then None else Some x) ' topspace X = topspace ?Y
  using ⟨a ∈ topspace X⟩ by force
  show openin X {x ∈ topspace X. (if x = a then None else Some x) ∈ U} =
openin ?Y U (is ?L = ?R)
  if U ⊆ topspace ?Y for U
  proof (cases None ∈ U)
  case True
  then obtain T where T[simp]: U = insert None (Some ' T)
  by (metis Int_insert_right UNIV_I UNIV_option_conv inf.orderE inf_le2
subsetI subset_imageE)
  have Tsub: T ⊆ topspace X - {a}

```

```

    using in_these_eq that by auto
  then have {x ∈ topspace X. (if x = a then None else Some x) ∈ U} = insert
a T
    by (auto simp: a_image_iff cong: conj_cong)
  then have ?L ↔ openin X (insert a T)
    by metis
  also have ... ↔ ?R
    using Tsub_assms
  apply (simp add: openin_Alexandroff_compactification kc_space_as_compactification_unique_exp
[where a=a])
    by (smt (verit, best) Diff_insert2 Diff_subset closedin_imp_subset dou-
ble_diff)
  finally show ?thesis .
next
case False
then obtain T where [simp]: U = Some ' T
  by (metis Int_insert_right UNIV_I UNIV_option_conv inf.orderE inf_le2
subsetI subset_imageE)
  have **: ∧ V. openin X V ⇒ openin X (V - {a})
    by (simp add: assms compactin_imp_closedin_gen openin_diff)
  have Tsub: T ⊆ topspace X - {a}
    using in_these_eq that by auto
  then have {x ∈ topspace X. (if x = a then None else Some x) ∈ U} = T
    by (auto simp: image_iff cong: conj_cong)
  then show ?thesis
    by (simp add: ** Tsub openin_open_subtopology)
qed
qed
moreover have inj_on (λx. if x = a then None else Some x) (topspace X)
  by (auto simp: inj_on_def)
ultimately show ?thesis
  using homeomorphic_space_sym homeomorphic_space homeomorphic_map_def
by blast
qed

```

6.11.7 Extending continuous maps "pointwise" in a regular space

```

lemma continuous_map_on_intermediate_closure_of:
  assumes Y: regular_space Y
    and T: T ⊆ X closure_of S
    and f: ∧ t. t ∈ T ⇒ limitin Y f (f t) (atin_within X t S)
  shows continuous_map (subtopology X T) Y f
proof (clarsimp simp add: continuous_map_atin)
  fix a
  assume a ∈ topspace X and a ∈ T
  have f ' T ⊆ topspace Y
    by (metis f_image_subsetI limitin_topspace)
  have ∀ F x in atin_within X a T. f x ∈ W

```

```

  if W: openin Y W f a ∈ W for W
proof -
  obtain V C where openin Y V closedin Y C f a ∈ V V ⊆ C C ⊆ W
  by (metis Y W neighbourhood_base_of neighbourhood_base_of_closedin)
  have ∀F x in atin_within X a S. f x ∈ V
  by (metis ⟨a ∈ T⟩ ⟨f a ∈ V⟩ ⟨openin Y V⟩ f limitin_def)
  then obtain U where openin X U a ∈ U and U: ∀x ∈ U - {a}. x ∈ S →
  f x ∈ V
  by (smt (verit) Diff_iff ⟨a ∈ topspace X⟩ eventually_atin_within insert_iff)
  moreover have f z ∈ W if z ∈ U z ≠ a z ∈ T for z
proof -
  have z ∈ topspace X
  using ⟨openin X U⟩ openin_subset ⟨z ∈ U⟩ by blast
  then have f z ∈ topspace Y
  using ⟨f ‘ T ⊆ topspace Y⟩ ⟨z ∈ T⟩ by blast
  { assume f z ∈ topspace Y f z ∉ C
  then have ∀F x in atin_within X z S. f x ∈ topspace Y - C
  by (metis Diff_iff ⟨closedin Y C⟩ closedin_def f limitinD ⟨z ∈ T⟩)
  then obtain U' where U': openin X U' z ∈ U'
  ∧ x. x ∈ U' - {z} ⇒ x ∈ S ⇒ f x ∉ C
  by (smt (verit) Diff_iff ⟨z ∈ topspace X⟩ eventually_atin_within insertCI)
  then have *: ∧D. z ∈ D ∧ openin X D ⇒ ∃y. y ∈ S ∧ y ∈ D
  by (meson T in_closure_of_subsetD ⟨z ∈ T⟩)
  have False
  using * [of U ∩ U'] U' U ⟨V ⊆ C⟩ ⟨f a ∈ V⟩ ⟨f z ∉ C⟩ ⟨openin X U⟩
  that
  by blast
  }
  then show ?thesis
  using ⟨C ⊆ W⟩ ⟨f z ∈ topspace Y⟩ by auto
qed
ultimately have ∃U. openin X U ∧ a ∈ U ∧ (∀x ∈ U - {a}. x ∈ T → f x
∈ W)
  by blast
  then show ?thesis
  using eventually_atin_within by fastforce
qed
then show limitin Y f (f a) (atin (subtopology X T) a)
  by (metis ⟨a ∈ T⟩ atin_subtopology_within f limitin_def)
qed

```

```

lemma continuous_map_on_intermediate_closure_of_eq:
  assumes regular_space Y S ⊆ T and Tsub: T ⊆ X closure_of S
  shows continuous_map (subtopology X T) Y f ↔ (∀t ∈ T. limitin Y f (f t)
(atin_within X t S))
  (is ?lhs ↔ ?rhs)
proof
  assume L: ?lhs

```

```

show ?rhs
proof (clarsimp simp add: continuous_map_atin)
  fix x
  assume  $x \in T$ 
  with  $L$   $T_{\text{sub}}$   $\text{closure\_of\_subset\_topspace}$ 
  have  $\text{limitin } Y f (f x) (\text{atin } (\text{subtopology } X T) x)$ 
    by (fastforce simp: continuous_map_atin)
  then show  $\text{limitin } Y f (f x) (\text{atin\_within } X x S)$ 
    using  $\langle x \in T \rangle \langle S \subseteq T \rangle$ 
    by (force simp: limitin_def atin_subtopology_within eventually_atin_within)
  qed
next
show ?rhs  $\implies$  ?lhs
  using assms continuous_map_on_intermediate_closure_of by blast
qed

```

```

lemma continuous_map_extension_pointwise_alt:
  assumes §: regular_space  $Y S \subseteq T T \subseteq X$   $\text{closure\_of } S$ 
  and  $f$ : continuous_map (subtopology  $X S$ )  $Y f$ 
  and  $\text{lim}$ :  $\bigwedge t. t \in T - S \implies \exists l. \text{limitin } Y f l (\text{atin\_within } X t S)$ 
  obtains  $g$  where continuous_map (subtopology  $X T$ )  $Y g \bigwedge x. x \in S \implies g x = f x$ 
proof -
  obtain  $g$  where  $g: \bigwedge t. t \in T \wedge t \notin S \implies \text{limitin } Y f (g t) (\text{atin\_within } X t S)$ 
    by (metis Diff_iff lim)
  let ?h =  $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } g x$ 
  show thesis
  proof
    have  $T: T \subseteq \text{topspace } X$ 
      using § closure_of_subset_topspace by fastforce
    have  $\text{limitin } Y ?h (f t) (\text{atin\_within } X t S)$  if  $t \in T t \in S$  for  $t$ 
    proof -
      have  $\text{limitin } Y f (f t) (\text{atin\_within } X t S)$ 
        by (meson  $T f$  limit_continuous_map_within subset_eq that)
      then show ?thesis
        by (simp add: eventually_atin_within limitin_def)
    qed
    moreover have  $\text{limitin } Y ?h (g t) (\text{atin\_within } X t S)$  if  $t \in T t \notin S$  for  $t$ 
      by (smt (verit, del_insts) eventually_atin_within g limitin_def that)
    ultimately show continuous_map (subtopology  $X T$ )  $Y ?h$ 
      unfolding continuous_map_on_intermediate_closure_of_eq [OF §]
      by (auto simp: § atin_subtopology_within)
    qed auto
  qed

```

```

lemma continuous_map_extension_pointwise:
  assumes regular_space  $Y S \subseteq T$  and  $T_{\text{sub}}: T \subseteq X$   $\text{closure\_of } S$ 
  and  $ex: \bigwedge x. x \in T \implies \exists g. \text{continuous\_map } (\text{subtopology } X (\text{insert } x S)) Y$ 

```

```

g ∧
  (∀ x ∈ S. g x = f x)
obtains g where continuous_map (subtopology X T) Y g ∧ x. x ∈ S ⇒ g x =
f x
proof (rule continuous_map_extension_pointwise_alt)
  show continuous_map (subtopology X S) Y f
  proof (clarsimp simp add: continuous_map_atin)
    fix t
    assume t ∈ topspace X and t ∈ S
    then obtain g where g: limitin Y g (g t) (atin (subtopology X (insert t S)) t)
and gf: ∀ x ∈ S. g x = f x
    by (metis Int_iff ‹S ⊆ T› continuous_map_atin ex inf.orderE insert_absorb
topspace_subtopology)
    with ‹t ∈ S› show limitin Y f (f t) (atin (subtopology X S) t)
    by (simp add: limitin_def atin_subtopology_within_if_eventually_atin_within
gf insert_absorb)
  qed
  show ∃ l. limitin Y f l (atin_within X t S) if t ∈ T - S for t
  proof -
    obtain g where g: continuous_map (subtopology X (insert t S)) Y g and gf:
∀ x ∈ S. g x = f x
    using ‹S ⊆ T› ex ‹t ∈ T - S› by force
    then have limitin Y g (g t) (atin_within X t (insert t S))
    using Tsub in_closure_of_limit_continuous_map_within that by fastforce
    then show ?thesis
    unfolding limitin_def
    by (smt (verit) eventually_atin_within gf subsetD subset_insertI)
  qed
qed (use assms in auto)

```

6.11.8 Extending Cauchy continuous functions to the closure

```

lemma Cauchy_continuous_map_extends_to_continuous_closure_of_aux:
  assumes m2: mcomplete_of m2 and f: Cauchy_continuous_map (submetric
m1 S) m2 f
  and S ⊆ mspace m1
  obtains g
  where continuous_map (subtopology (mtopology_of m1) (mtopology_of m1 clo-
sure_of S))
    (mtopology_of m2) g ∧ x. x ∈ S ⇒ g x = f x
proof (rule continuous_map_extension_pointwise_alt)
  interpret L: Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2
  by (simp add: Metric_space12_mspace_mdist)
  interpret S: Metric_space S ∩ mspace m1 mdist m1
  by (simp add: L.M1.subspace)
  show regular_space (mtopology_of m2)
  by (simp add: Metric_space.regular_space_mtopology_mtopology_of_def)
  show S ⊆ mtopology_of m1 closure_of S
  by (simp add: assms(3) closure_of_subset)

```

```

show continuous_map (subtopology (mtopology_of m1) S) (mtopology_of m2) f
  by (metis Cauchy_continuous_imp_continuous_map f mtopology_of_submetric)
fix a
assume a: a ∈ mtopology_of m1 closure_of S - S
then obtain σ where ranσ: range σ ⊆ S range σ ⊆ mspace m1
  and limσ: limitin L.M1.mtopology σ a sequentially
  by (force simp: mtopology_of_def L.M1.closure_of_sequentially)
then have L.M1.MCauchy σ
  by (simp add: L.M1.convergent_imp_MCauchy mtopology_of_def)
then have L.M2.MCauchy (f ∘ σ)
  using f ranσ by (simp add: Cauchy_continuous_map_def L.M1.subspace Metric_space.MCauchy_def)
then obtain l where l: limitin L.M2.mtopology (f ∘ σ) l sequentially
  by (meson L.M2.mcomplete_def m2 mcomplete_of_def)
have limitin L.M2.mtopology f l (atin_within L.M1.mtopology a S)
  unfolding L.limit_atin_sequentially_within_imp_conjL
proof (intro conjI strip)
  show l ∈ mspace m2
  using L.M2.limitin_mspace l by blast
fix ρ
assume range ρ ⊆ S ∩ mspace m1 - {a} and limρ: limitin L.M1.mtopology
ρ a sequentially
then have ranρ: range ρ ⊆ S range ρ ⊆ mspace m1 ∧ n. ρ n ≠ a
  by auto
have a ∈ mspace m1
  using L.M1.limitin_mspace limρ by auto
have S.MCauchy σ S.MCauchy ρ
  using L.M1.convergent_imp_MCauchy L.M1.MCauchy_def S.MCauchy_def
limσ ranσ limρ ranρ by force+
then have L.M2.MCauchy (f ∘ ρ) L.M2.MCauchy (f ∘ σ)
  using f by (auto simp: Cauchy_continuous_map_def)
then have ran_f: range (λx. f (ρ x)) ⊆ mspace m2 range (λx. f (σ x)) ⊆
mspace m2
  by (auto simp: L.M2.MCauchy_def)
have (λn. mdist m2 (f (ρ n)) l) ⟶ 0
proof (rule Lim_null_comparison)
  have mdist m2 (f (ρ n)) l ≤ mdist m2 (f (σ n)) l + mdist m2 (f (σ n)) (f
(ρ n)) for n
  using ⟨l ∈ mspace m2⟩ ran_f L.M2.triangle'' by (smt (verit, best)
range_subsetD)
  then show ∀F n in sequentially. norm (mdist m2 (f (ρ n)) l) ≤ mdist m2
(f (σ n)) l + mdist m2 (f (σ n)) (f (ρ n))
  by force
define ψ where ψ ≡ λn. if even n then σ (n div 2) else ρ (n div 2)
have (λn. mdist m1 (σ n) (ρ n)) ⟶ 0
proof (rule Lim_null_comparison)
  show ∀F n in sequentially. norm (mdist m1 (σ n) (ρ n)) ≤ mdist m1 (σ
n) a + mdist m1 (ρ n) a
  using L.M1.triangle' [of _ a] ranσ ranρ ⟨a ∈ mspace m1⟩ by (simp add:

```



```

range_subsetD)
  have  $(\lambda n. \text{mdist } m1 (\sigma n) a) \longrightarrow 0$ 
    using L.M1.limitin_metric_dist_null lim $\sigma$  by blast
  moreover have  $(\lambda n. \text{mdist } m1 (\varrho n) a) \longrightarrow 0$ 
    using L.M1.limitin_metric_dist_null lim $\varrho$  by blast
  ultimately show  $(\lambda n. \text{mdist } m1 (\sigma n) a + \text{mdist } m1 (\varrho n) a) \longrightarrow 0$ 
    by (simp add: tendsto_add_zero)
qed
with  $\langle S.MCauchy \sigma \rangle \langle S.MCauchy \varrho \rangle$  have S.MCauchy  $\psi$ 
  by (simp add: S.MCauchy_interleaving_gen  $\psi\_def$ )
then have L.M2.MCauchy  $(f \circ \psi)$ 
by (metis Cauchy_continuous_map_def mdist_submetric mspace_submetric)
then have  $(\lambda n. \text{mdist } m2 (f (\sigma n)) (f (\varrho n))) \longrightarrow 0$ 
  using L.M2.MCauchy_interleaving_gen [of  $f \circ \sigma f \circ \varrho$ ]
  by (simp add: if_distrib  $\psi\_def$  o_def cong: if_cong)
moreover have  $\forall_F n$  in sequentially.  $f (\sigma n) \in \text{mspace } m2 \wedge (\lambda x. \text{mdist } m2$ 
 $(f (\sigma x)) l) \longrightarrow 0$ 
  using l by (auto simp: L.M2.limitin_metric_dist_null  $\langle l \in \text{mspace } m2 \rangle$ )
ultimately show  $(\lambda n. \text{mdist } m2 (f (\sigma n)) l + \text{mdist } m2 (f (\sigma n)) (f (\varrho n)))$ 
 $\longrightarrow 0$ 
  by (metis (mono_tags) tendsto_add_zero eventually_sequentially order_refl)
qed
with ran_f show limitin L.M2.mtopology  $(f \circ \varrho) l$  sequentially
  by (auto simp: L.M2.limitin_metric_dist_null eventually_sequentially  $\langle l \in$ 
mspace  $m2 \rangle$ )
qed
then show  $\exists l. \text{limitin} (\text{mtopology\_of } m2) f l$  (atin_within (mtopology_of  $m1$ )
 $a S$ )
  by (force simp: mtopology_of_def)
qed auto

```

lemma *Cauchy_continuous_map_extends_to_continuous_closure_of:*

```

assumes mcomplete_of  $m2$ 
  and f: Cauchy_continuous_map (submetric  $m1 S$ )  $m2 f$ 
obtains g
  where continuous_map (subtopology (mtopology_of  $m1$ ) ((mtopology_of  $m1$ )
closure_of  $S$ ))
    (mtopology_of  $m2$ )  $g \wedge \lambda x. x \in S \implies g x = f x$ 

```

proof –

```

  obtain g where cmg:
    continuous_map (subtopology (mtopology_of  $m1$ ) ((mtopology_of  $m1$ )
closure_of (mspace  $m1 \cap S$ )))
      (mtopology_of  $m2$ )  $g$ 
  and gf:  $(\forall x \in \text{mspace } m1 \cap S. g x = f x)$ 
  using Cauchy_continuous_map_extends_to_continuous_closure_of_aux assms
  by (metis inf_commute inf_le2 submetric_restrict)
  define h where h  $\equiv \lambda x. \text{if } x \in \text{topspace}(\text{mtopology\_of } m1) \text{ then } g x \text{ else } f x$ 
  show thesis

```

```

proof
  show continuous_map (subtopology (mtopology_of m1) ((mtopology_of m1)
closure_of S))
    (mtopology_of m2) h
    unfolding h_def
    proof (rule continuous_map_eq)
      show continuous_map (subtopology (mtopology_of m1) (mtopology_of m1
closure_of S)) (mtopology_of m2) g
      by (metis closure_of_restrict cmg topspace_mtopology_of)
    qed auto
  qed (auto simp: gf h_def)
qed

```

```

lemma Cauchy_continuous_map_extends_to_continuous_intermediate_closure_of:
  assumes mcomplete_of m2
  and f: Cauchy_continuous_map (submetric m1 S) m2 f
  and T:  $T \subseteq \text{mtopology\_of } m1 \text{ closure\_of } S$ 
  obtains g
  where continuous_map (subtopology (mtopology_of m1) T) (mtopology_of m2)
g
    ( $\forall x \in S. g \ x = f \ x$ )
  by (metis Cauchy_continuous_map_extends_to_continuous_closure_of T assms(1)
continuous_map_from_subtopology_mono f)

```

Technical lemma helpful for porting particularly ugly HOL Light proofs

```

lemma all_in_closure_of:
  assumes P:  $\forall x \in S. P \ x$  and clo: closedin X  $\{x \in \text{topspace } X. P \ x\}$ 
  shows  $\forall x \in X \text{ closure\_of } S. P \ x$ 
proof –
  have  $*$ :  $\text{topspace } X \cap S \subseteq \{x \in \text{topspace } X. P \ x\}$ 
  using P by auto
  show ?thesis
  using closure_of_minimal [OF  $*$  clo] closure_of_restrict by fastforce
qed

```

```

lemma Lipschitz_continuous_map_on_intermediate_closure_aux:
  assumes lcf: Lipschitz_continuous_map (submetric m1 S) m2 f
  and  $S \subseteq T$  and Tsub:  $T \subseteq (\text{mtopology\_of } m1) \text{ closure\_of } S$ 
  and cmf: continuous_map (subtopology (mtopology_of m1) T) (mtopology_of
m2) f
  and  $S \subseteq \text{mspace } m1$ 
  shows Lipschitz_continuous_map (submetric m1 T) m2 f
proof –
  interpret L: Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2
  by (simp add: Metric_space12_mspace_mdist)
  interpret S: Metric_space  $S \cap \text{mspace } m1$  mdist m1
  by (simp add: L.M1.subspace)
  have  $T \subseteq \text{mspace } m1$ 

```

```

    using Tsub by (auto simp: mtopology_of_def closure_of_def)
  show ?thesis
    unfolding Lipschitz_continuous_map_pos
  proof
    show  $f \in \text{mspace (submetric } m1 \ T) \rightarrow \text{mspace } m2$ 
      by (metis cmf Metric_space.metric_continuous_map Metric_space_mspace_mdistspace_mtopology_of_def
          mtopology_of_submetric image_subset_iff_funcset)
    define X where  $X \equiv \text{prod\_topology (subtopology } L.M1.mtopology \ T) \text{ (subtopology } L.M1.mtopology \ T)$ 
    obtain  $B::\text{real}$  where  $B > 0$  and  $B: \forall (x,y) \in S \times S. \text{mdist } m2 \ (f \ x) \ (f \ y) \leq B * \text{mdist } m1 \ x \ y$ 
      using lcf  $\langle S \subseteq \text{mspace } m1 \rangle$  by (force simp: Lipschitz_continuous_map_pos)
    have eq:  $\{z \in A. \text{case } z \text{ of } (x,y) \Rightarrow p \ x \ y \leq B * q \ x \ y\} = \{z \in A. ((\lambda(x,y). B * q \ x \ y - p \ x \ y)z) \in \{0..\}\}$ 
      for  $p \ q$  and  $A::('a*'a)\text{set}$ 
      by auto
    have clo:  $\text{closedin } X \ \{z \in \text{topspace } X. \text{case } z \text{ of } (x, y) \Rightarrow \text{mdist } m2 \ (f \ x) \ (f \ y) \leq B * \text{mdist } m1 \ x \ y\}$ 
      unfolding eq
    proof (rule closedin_continuous_map_preimage)
      have *:  $\text{continuous\_map } X \ L.M2.mtopology \ (f \circ \text{fst}) \ \text{continuous\_map } X \ L.M2.mtopology \ (f \circ \text{snd})$ 
        using cmf by (auto simp: mtopology_of_def X_def intro: continuous_map_compose continuous_map_fst continuous_map_snd)
      then show  $\text{continuous\_map } X \ \text{euclidean} \ (\lambda x. \text{case } x \text{ of } (x, y) \Rightarrow B * \text{mdist } m1 \ x \ y - \text{mdist } m2 \ (f \ x) \ (f \ y))$ 
        unfolding case_prod_unfold
      proof (intro continuous_intros; simp add: mtopology_of_def o_def)
        show  $\text{continuous\_map } X \ L.M1.mtopology \ \text{fst} \ \text{continuous\_map } X \ L.M1.mtopology \ \text{snd}$ 
          by (simp_all add: X_def continuous_map_subtopology_fst continuous_map_subtopology_snd flip: subtopology_Times)
        qed
      qed auto
    have  $\text{mdist } m2 \ (f \ x) \ (f \ y) \leq B * \text{mdist } m1 \ x \ y$  if  $x \in T \ y \in T$  for  $x \ y$ 
      using all_in_closure_of [OF B clo]  $\langle S \subseteq T \rangle$  Tsub
    by (fastforce simp: X_def subset_iff closure_of_Times closure_of_subtopology inf.absorb2
        mtopology_of_def that)
    then show  $\exists B > 0. \forall x \in \text{mspace (submetric } m1 \ T). \forall y \in \text{mspace (submetric } m1 \ T). \text{mdist } m2 \ (f \ x) \ (f \ y) \leq B * \text{mdist (submetric } m1 \ T) \ x \ y$ 
      using  $\langle 0 < B \rangle$  by auto
    qed
  qed

```

lemma *Lipschitz_continuous_map_on_intermediate_closure:*

assumes *Lipschitz_continuous_map* (submetric $m1$ S) $m2$ f
and $S \subseteq T$ $T \subseteq$ (mtopology_of $m1$) closure_of S
and *continuous_map* (subtopology (mtopology_of $m1$) T) (mtopology_of $m2$)
 f
shows *Lipschitz_continuous_map* (submetric $m1$ T) $m2$ f
by (metis *Lipschitz_continuous_map_on_intermediate_closure_aux* *assms* closure_of_subset_topospace subset_trans topospace_mtopology_of)

lemma *Lipschitz_continuous_map_extends_to_closure_of*:
assumes $m2$: *mcomplete_of* $m2$
and f : *Lipschitz_continuous_map* (submetric $m1$ S) $m2$ f
obtains g
where *Lipschitz_continuous_map* (submetric $m1$ (mtopology_of $m1$ closure_of S)) $m2$ g
 $\bigwedge x. x \in S \implies g\ x = f\ x$
proof –
obtain g
where g : *continuous_map* (subtopology (mtopology_of $m1$) ((mtopology_of $m1$) closure_of S))
(mtopology_of $m2$) g ($\forall x \in S. g\ x = f\ x$)
by (metis *Cauchy_continuous_map_extends_to_continuous_closure_of_Lipschitz_imp_Cauchy_continuous_map* $f\ m2$)
have *Lipschitz_continuous_map* (submetric $m1$ (mtopology_of $m1$ closure_of S)) $m2$ g
proof (rule *Lipschitz_continuous_map_on_intermediate_closure*)
show *Lipschitz_continuous_map* (submetric $m1$ (mspace $m1 \cap S$)) $m2$ g
by (smt (verit, best) *IntD2_Lipschitz_continuous_map_eq_f_g(2)* *inf_commute* *mspace_submetric_submetric_restrict*)
show $mspace\ m1 \cap S \subseteq$ mtopology_of $m1$ closure_of S
using closure_of_subset_Int **by** force
show mtopology_of $m1$ closure_of $S \subseteq$ mtopology_of $m1$ closure_of (mspace $m1 \cap S$)
by (metis closure_of_restrict subset_refl topospace_mtopology_of)
show *continuous_map* (subtopology (mtopology_of $m1$) (mtopology_of $m1$ closure_of S)) (mtopology_of $m2$) g
by (simp add: g)
qed
with g **that** **show** *thesis*
by *metis*
qed

lemma *Lipschitz_continuous_map_extends_to_intermediate_closure_of*:
assumes $mcomplete_of$ $m2$
and *Lipschitz_continuous_map* (submetric $m1$ S) $m2$ f
and $T \subseteq$ mtopology_of $m1$ closure_of S
obtains g
where *Lipschitz_continuous_map* (submetric $m1$ T) $m2$ g $\bigwedge x. x \in S \implies g\ x$

```
= f x
  by (metis Lipschitz_continuous_map_extends_to_closure_of_Lipschitz_continuous_map_from_submetric_monotonicity)
  assms)
```

This proof uses the same trick to extend the function's domain to its closure

```
lemma uniformly_continuous_map_on_intermediate_closure_aux:
  assumes ucf: uniformly_continuous_map (submetric m1 S) m2 f
  and S ⊆ T and Tsub: T ⊆ (mtopology_of m1) closure_of S
  and cmf: continuous_map (subtopology (mtopology_of m1) T) (mtopology_of m2) f
  and S ⊆ mspace m1
  shows uniformly_continuous_map (submetric m1 T) m2 f
proof -
  interpret L: Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2
  by (simp add: Metric_space12_mspace_mdists)
  interpret S: Metric_space S ∩ mspace m1 mdist m1
  by (simp add: L.M1.subspace)
  have T ⊆ mspace m1
  using Tsub by (auto simp: mtopology_of_def closure_of_def)
  show ?thesis
  unfolding uniformly_continuous_map_def
  proof (intro conjI strip)
  show f ∈ mspace (submetric m1 T) → mspace m2
  by (metis cmf Metric_space.metric_continuous_map Metric_space_mspace_mdists

      mtopology_of_def mtopology_of_submetric image_subset_iff_funcset)
  fix ε::real
  assume ε > 0
  then obtain δ where δ > 0 and δ: ∀ (x,y) ∈ S × S. mdist m1 x y < δ → mdist
m2 (f x) (f y) ≤ ε/2
  using ucf ⟨S ⊆ mspace m1⟩ unfolding uniformly_continuous_map_def
mspace_submetric
  apply (simp add: Ball_def del: divide_const_simps)
  by (metis IntD2 half_gt_zero inf.orderE less_eq_real_def)
  define X where X ≡ prod_topology (subtopology L.M1.mtopology T) (subtopology
L.M1.mtopology T)
```

A clever construction involving the union of two closed sets

```
have eq: {z ∈ A. case z of (x,y) ⇒ p x y < d → q x y ≤ e}
  = {z ∈ A. ((λ(x,y). p x y - d)z) ∈ {0..}} ∪ {z ∈ A. ((λ(x,y). e - q x
y)z) ∈ {0..}}
  for p q and d e::real and A::('a*'a)set
  by auto
  have clo: closedin X {z ∈ topspace X. case z of (x, y) ⇒ mdist m1 x y < δ
→ mdist m2 (f x) (f y) ≤ ε/2}
  unfolding eq
  proof (intro closedin_Un closedin_continuous_map_preimage)
  have *: continuous_map X L.M1.mtopology fst continuous_map X L.M1.mtopology
snd
```

```

    by (metis X_def continuous_map_subtopology_fst subtopology_Times continuous_map_subtopology_snd)+
    show continuous_map X euclidean ( $\lambda x. \text{case } x \text{ of } (x, y) \Rightarrow \text{mdist } m1 \ x \ y - \delta$ )
      unfolding case_prod_unfold
      by (intro continuous_intros; simp add: mtopology_of_def *)
      have *: continuous_map X L.M2.mtopology (f  $\circ$  fst) continuous_map X L.M2.mtopology (f  $\circ$  snd)
      using cmf by (auto simp: mtopology_of_def X_def intro: continuous_map_compose continuous_map_fst continuous_map_snd)
      then show continuous_map X euclidean ( $\lambda x. \text{case } x \text{ of } (x, y) \Rightarrow \varepsilon / 2 - \text{mdist } m2 \ (f \ x) \ (f \ y)$ )
        unfolding case_prod_unfold
        by (intro continuous_intros; simp add: mtopology_of_def o_def)
      qed auto
      have  $\text{mdist } m2 \ (f \ x) \ (f \ y) \leq \varepsilon / 2$  if  $x \in T \ y \in T \ \text{mdist } m1 \ x \ y < \delta$  for  $x \ y$ 
        using all_in_closure_of [OF  $\delta \ \text{clo}$ ]  $\langle S \subseteq T \rangle \ Tsub$ 
      by (fastforce simp: X_def subset_iff closure_of_Times closure_of_subtopology inf.absorb2
          mtopology_of_def that)
      then show  $\exists \delta > 0. \forall x \in \text{mspace} \ (\text{submetric } m1 \ T). \forall y \in \text{mspace} \ (\text{submetric } m1 \ T). \text{mdist} \ (\text{submetric } m1 \ T) \ y \ x < \delta \longrightarrow \text{mdist } m2 \ (f \ y) \ (f \ x) < \varepsilon$ 
        using  $\langle 0 < \delta \rangle \ \langle 0 < \varepsilon \rangle$  by fastforce
      qed
    qed
  
```

```

lemma uniformly_continuous_map_on_intermediate_closure:
  assumes uniformly_continuous_map (submetric m1 S) m2 f
    and  $S \subseteq T$  and  $T \subseteq (\text{mtopology\_of } m1) \ \text{closure\_of } S$ 
    and continuous_map (subtopology (mtopology_of m1) T) (mtopology_of m2)
  f
  shows uniformly_continuous_map (submetric m1 T) m2 f
  by (metis assms closure_of_subset_topspace subset_trans topspace_mtopology_of
      uniformly_continuous_map_on_intermediate_closure_aux)
  
```

```

lemma uniformly_continuous_map_extends_to_closure_of:
  assumes m2: mcomplete_of m2
    and f: uniformly_continuous_map (submetric m1 S) m2 f
  obtains g
  where uniformly_continuous_map (submetric m1 (mtopology_of m1 closure_of S)) m2 g
     $\wedge x. x \in S \implies g \ x = f \ x$ 
  proof -
    obtain g
    where g: continuous_map (subtopology (mtopology_of m1) ((mtopology_of m1) closure_of S))
      (mtopology_of m2) g ( $\forall x \in S. g \ x = f \ x$ )
    by (metis Cauchy_continuous_map_extends_to_continuous_closure_of uni-
  
```

```

formly_imp_Cauchy_continuous_map f m2)
  have uniformly_continuous_map (submetric m1 (mtopology_of m1 closure_of
S)) m2 g
  proof (rule uniformly_continuous_map_on_intermediate_closure)
    show uniformly_continuous_map (submetric m1 (mspace m1  $\cap$  S)) m2 g
    by (smt (verit, best) IntD2 uniformly_continuous_map_eq f g(2) inf_commute
mspace_submetric submetric_restrict)
    show mspace m1  $\cap$  S  $\subseteq$  mtopology_of m1 closure_of S
    using closure_of_subset_Int by force
    show mtopology_of m1 closure_of S  $\subseteq$  mtopology_of m1 closure_of (mspace
m1  $\cap$  S)
    by (metis closure_of_restrict subset_refl topspace_mtopology_of)
    show continuous_map (subtopology (mtopology_of m1) (mtopology_of m1 clo-
sure_of S)) (mtopology_of m2) g
    by (simp add: g)
  qed
with g that show thesis
  by metis
qed

```

lemma *uniformly_continuous_map_extends_to_intermediate_closure_of:*

```

  assumes mcomplete_of m2
    and uniformly_continuous_map (submetric m1 S) m2 f
    and T  $\subseteq$  mtopology_of m1 closure_of S
  obtains g
  where uniformly_continuous_map (submetric m1 T) m2 g  $\wedge$   $x. x \in S \implies g x$ 
= f x
  by (metis uniformly_continuous_map_extends_to_closure_of uniformly_continuous_map_from_submetric_mor-
assms)

```

lemma *Cauchy_continuous_map_on_intermediate_closure_aux:*

```

  assumes ucf: Cauchy_continuous_map (submetric m1 S) m2 f
    and S  $\subseteq$  T and Tsub: T  $\subseteq$  (mtopology_of m1) closure_of S
    and cmf: continuous_map (subtopology (mtopology_of m1) T) (mtopology_of
m2) f
    and S  $\subseteq$  mspace m1
  shows Cauchy_continuous_map (submetric m1 T) m2 f
proof -
  interpret L: Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2
    by (simp add: Metric_space12_mspace_mdist)
  interpret S: Metric_space S  $\cap$  mspace m1 mdist m1
    by (simp add: L.M1.subspace)
  interpret T: Metric_space T mdist m1
    by (metis L.M1.subspace Tsub closure_of_subset_topspace dual_order.trans
topspace_mtopology_of)
  have T  $\subseteq$  mspace m1
    using Tsub by (auto simp: mtopology_of_def closure_of_def)

```

```

then show ?thesis
proof (clarsimp simp: Cauchy_continuous_map_def Int_absorb2)
  fix  $\sigma$ 
  assume  $\sigma: T.MCauchy \sigma$ 
  have  $\exists y \in S. mdist m1 (\sigma n) y < inverse (Suc n) \wedge mdist m2 (f (\sigma n)) (f y)$ 
   $< inverse (Suc n)$  for  $n$ 
  proof -
    have  $\sigma n \in T$ 
    using  $\sigma$  by (force simp: T.MCauchy_def)
    moreover have continuous_map (mtopology_of (submetric m1 T)) L.M2.mtopology
   $f$ 
    by (metis cmf mtopology_of_def mtopology_of_submetric)
    ultimately obtain  $\delta$  where  $\delta > 0$  and  $\delta: \forall x \in T. mdist m1 (\sigma n) x < \delta$ 
   $\longrightarrow mdist m2 (f(\sigma n)) (f x) < inverse (Suc n)$ 
    using  $\langle T \subseteq mspace m1 \rangle$ 
    apply (simp add: mtopology_of_def Metric_space.metric_continuous_map
  L.M1.subspace Int_absorb2)
    by (metis inverse_Suc of_nat_Suc)
    have  $\exists y \in S. mdist m1 (\sigma n) y < \min \delta (inverse (Suc n))$ 
    using  $\langle \sigma n \in T \rangle Tsub \langle \delta > 0 \rangle$ 
    unfolding mtopology_of_def L.M1.metric_closure_of_subset_iff mem_Collect_eq
  L.M1.in_mball
    by (smt (verit, del_insts) inverse_Suc )
    with  $\delta \langle S \subseteq T \rangle$  show ?thesis
    by auto
  qed
  then obtain  $\rho$  where  $\rho S: \bigwedge n. \rho n \in S$  and  $\rho 1: \bigwedge n. mdist m1 (\sigma n) (\rho n) <$ 
   $inverse (Suc n)$ 
    and  $\rho 2: \bigwedge n. mdist m2 (f (\sigma n)) (f (\rho n)) < inverse (Suc n)$ 
    by metis
  have S.MCauchy  $\rho$ 
  unfolding S.MCauchy_def
  proof (intro conjI strip)
    show range  $\rho \subseteq S \cap mspace m1$ 
    using  $\langle S \subseteq mspace m1 \rangle$  by (auto simp:  $\rho S$ )
    fix  $\varepsilon :: real$ 
    assume  $\varepsilon > 0$ 
    then obtain  $M$  where  $M: \bigwedge n n'. M \leq n \implies M \leq n' \implies mdist m1 (\sigma n)$ 
   $(\sigma n') < \varepsilon/2$ 
    using  $\sigma$  unfolding T.MCauchy_def by (meson half_gt_zero)
    have  $\forall_F n$  in sequentially.  $inverse (Suc n) < \varepsilon/4$ 
    using Archimedean_eventually_inverse  $\langle 0 < \varepsilon \rangle$  divide_pos_pos zero_less_numeral
  by blast
    then obtain  $N$  where  $N: \bigwedge n. N \leq n \implies inverse (Suc n) < \varepsilon/4$ 
    by (meson eventually_sequentially)
    have  $mdist m1 (\rho n) (\rho n') < \varepsilon$  if  $n \geq \max M N n' \geq \max M N$  for  $n n'$ 
    proof -
      have  $mdist m1 (\rho n) (\rho n') \leq mdist m1 (\rho n) (\sigma n) + mdist m1 (\sigma n) (\rho$ 
   $n')$ 

```



```

    by (meson T.MCauchy_def T.triangle ρS σ ⟨S ⊆ T⟩ rangeI subset_iff)
  also have ... ≤ mdist m1 (ρ n) (σ n) + mdist m1 (σ n) (σ n') + mdist
m1 (σ n') (ρ n')
    by (smt (verit, best) T.MCauchy_def T.triangle ρS σ ⟨S ⊆ T⟩ in_mono
rangeI)
  also have ... < ε/4 + ε/2 + ε/4
    using ρ1[of n] ρ1[of n'] N[of n] N[of n'] that M[of n n'] by (simp add:
T commute)
  also have ... ≤ ε
    by simp
  finally show ?thesis .
qed
then show ∃ N. ∀ n n'. N ≤ n ⟶ N ≤ n' ⟶ mdist m1 (ρ n) (ρ n') < ε
  by blast
qed
then have fρ: L.M2.MCauchy (f ∘ ρ)
  using ucf by (simp add: Cauchy_continuous_map_def)
show L.M2.MCauchy (f ∘ σ)
  unfolding L.M2.MCauchy_def
proof (intro conjI strip)
  show range (f ∘ σ) ⊆ mspace m2
    using ⟨T ⊆ mspace m1⟩ σ cmf
  apply (auto simp: )
  by (metis Metric_space.metric_continuous_map Metric_space_mspace_mdists
T.MCauchy_def image_eqI inf.absorb1 mspace_submetric mtopology_of_def mtopology_of_submetric
range_subsetD subset_iff)
  fix ε :: real
  assume ε > 0
  then obtain M where M: ∧ n n'. M ≤ n ⟹ M ≤ n' ⟹ mdist m2 ((f ∘
ρ) n) ((f ∘ ρ) n') < ε/2
    using fρ unfolding L.M2.MCauchy_def by (meson half_gt_zero)
  have ∀ F n in sequentially. inverse (Suc n) < ε/4
    using Archimedean_eventually_inverse ⟨0 < ε⟩ divide_pos_pos zero_less_numeral
  by blast
  then obtain N where N: ∧ n. N ≤ n ⟹ inverse (Suc n) < ε/4
    by (meson eventually_sequentially)
  have mdist m2 ((f ∘ σ) n) ((f ∘ σ) n') < ε if n ≥ max M N n' ≥ max M N
  for n n'
  proof -
    have mdist m2 ((f ∘ σ) n) ((f ∘ σ) n') ≤ mdist m2 ((f ∘ σ) n) ((f ∘ ρ) n)
+ mdist m2 ((f ∘ ρ) n) ((f ∘ σ) n')
      by (meson L.M2.MCauchy_def ⟨range (f ∘ σ) ⊆ mspace m2⟩ fρ
mdist_triangle rangeI subset_eq)
    also have ... ≤ mdist m2 ((f ∘ σ) n) ((f ∘ ρ) n) + mdist m2 ((f ∘ ρ) n)
((f ∘ ρ) n') + mdist m2 ((f ∘ ρ) n') ((f ∘ σ) n')
      by (smt (verit) L.M2.MCauchy_def L.M2.triangle ⟨range (f ∘ σ) ⊆ mspace
m2⟩ fρ range_subsetD)
    also have ... < ε/4 + ε/2 + ε/4
      using ρ2[of n] ρ2[of n'] N[of n] N[of n'] that M[of n n'] by (simp add:

```

```

L.M2.commute)
  also have ... ≤ ε
  by simp
  finally show ?thesis .
qed
then show ∃ N. ∀ n n'. N ≤ n → N ≤ n' → mdist m2 ((f ∘ σ) n) ((f ∘
σ) n') < ε
  by blast
qed
qed
qed

```

lemma *Cauchy_continuous_map_on_intermediate_closure:*
assumes *Cauchy_continuous_map* (submetric m1 S) m2 f
and $S \subseteq T$ **and** $T \subseteq (\text{mtopology_of } m1) \text{ closure_of } S$
and *continuous_map* (subtopology (mtopology_of m1) T) (mtopology_of m2)
f
shows *Cauchy_continuous_map* (submetric m1 T) m2 f
by (metis *Cauchy_continuous_map_on_intermediate_closure_aux* assms clo-
sure_of_subset_topspace order.trans topspace_mtopology_of)

lemma *Cauchy_continuous_map_extends_to_closure_of:*
assumes m2: *mcomplete_of* m2
and f: *Cauchy_continuous_map* (submetric m1 S) m2 f
obtains g
where *Cauchy_continuous_map* (submetric m1 (mtopology_of m1 closure_of
S)) m2 g
 $\bigwedge x. x \in S \implies g x = f x$
proof –
obtain g
where g: *continuous_map* (subtopology (mtopology_of m1) ((mtopology_of m1)
closure_of S))
(mtopology_of m2) g ($\forall x \in S. g x = f x$)
by (metis *Cauchy_continuous_map_extends_to_continuous_closure_of* f m2)
have *Cauchy_continuous_map* (submetric m1 (mtopology_of m1 closure_of S))
m2 g
proof (rule *Cauchy_continuous_map_on_intermediate_closure*)
show *Cauchy_continuous_map* (submetric m1 (mspace m1 ∩ S)) m2 g
by (smt (verit, best) IntD2 *Cauchy_continuous_map_eq* f g(2) inf_commute
mspace_submetric_submetric_restrict)
show $mspace\ m1 \cap S \subseteq \text{mtopology_of } m1 \text{ closure_of } S$
using *closure_of_subset_Int* **by** force
show $\text{mtopology_of } m1 \text{ closure_of } S \subseteq \text{mtopology_of } m1 \text{ closure_of } (mspace\ m1 \cap S)$
by (metis *closure_of_restrict_subset_refl* topspace_mtopology_of)
show *continuous_map* (subtopology (mtopology_of m1) (mtopology_of m1 clo-
sure_of S)) (mtopology_of m2) g
by (simp add: g)

```

qed
with g that show thesis
  by metis
qed

```

```

lemma Cauchy_continuous_map_extends_to_intermediate_closure_of:
  assumes mcomplete_of m2
  and Cauchy_continuous_map (submetric m1 S) m2 f
  and  $T \subseteq \text{m topology\_of } m1 \text{ closure\_of } S$ 
  obtains g
  where Cauchy_continuous_map (submetric m1 T) m2 g  $\wedge x. x \in S \implies g x =$ 
  f x
  by (metis Cauchy_continuous_map_extends_to_closure_of Cauchy_continuous_map_from_submetric_mono
  assms)

```

6.11.9 Metric space of bounded functions

```

context Metric_space
begin

```

```

definition fspace :: 'b set  $\Rightarrow$  ('b  $\Rightarrow$  'a) set where
  fspace  $\equiv \lambda S. \{f. f'S \subseteq M \wedge f \in \text{extensional } S \wedge \text{mbounded } (f'S)\}$ 

```

```

definition fdist :: ['b set, 'b  $\Rightarrow$  'a, 'b  $\Rightarrow$  'a]  $\Rightarrow$  real where
  fdist  $\equiv \lambda S f g. \text{if } f \in \text{fspace } S \wedge g \in \text{fspace } S \wedge S \neq \{\} \text{ then } \text{Sup } ((\lambda x. d (f x) (g x)) ' S) \text{ else } 0$ 

```

```

lemma fspace_empty [simp]: fspace {} =  $\{\lambda x. \text{undefined}\}$ 
  by (auto simp: fspace_def)

```

```

lemma fdist_empty [simp]: fdist {} =  $(\lambda x y. 0)$ 
  by (auto simp: fdist_def)

```

```

lemma fspace_in_M:  $\llbracket f \in \text{fspace } S; x \in S \rrbracket \implies f x \in M$ 
  by (auto simp: fspace_def)

```

```

lemma bdd_above_dist:
  assumes f:  $f \in \text{fspace } S$  and g:  $g \in \text{fspace } S$  and  $S \neq \{\}$ 
  shows bdd_above  $((\lambda u. d (f u) (g u)) ' S)$ 

```

```

proof -
  obtain a where  $a \in S$ 
  using  $\langle S \neq \{\} \rangle$  by blast
  obtain B x C y where  $B > 0$  and  $B: f'S \subseteq \text{mcball } x B$ 
  and  $C > 0$  and  $C: g'S \subseteq \text{mcball } y C$ 
  using f g mbounded_pos by (auto simp: fspace_def)
  have  $d (f u) (g u) \leq B + d x y + C$  if  $u \in S$  for u
  proof -
    have  $f u \in M$ 

```

```

    by (meson B image_eqI mbounded_mcball mbounded_subset_mspace subsetD
that)
    have  $g\ u \in M$ 
    by (meson C image_eqI mbounded_mcball mbounded_subset_mspace subsetD
that)
    have  $x \in M\ y \in M$ 
    using B C that by auto
    have  $d\ (f\ u)\ (g\ u) \leq d\ (f\ u)\ x + d\ x\ (g\ u)$ 
    by (simp add:  $\langle f\ u \in M \rangle \langle g\ u \in M \rangle \langle x \in M \rangle$  triangle)
    also have  $\dots \leq d\ (f\ u)\ x + d\ x\ y + d\ y\ (g\ u)$ 
    by (simp add:  $\langle f\ u \in M \rangle \langle g\ u \in M \rangle \langle x \in M \rangle \langle y \in M \rangle$  triangle)
    also have  $\dots \leq B + d\ x\ y + C$ 
    using B C commute that by fastforce
    finally show ?thesis .
qed
then show ?thesis
by (meson bdd_above.I2)
qed

```

lemma *Metric_space_funspace: Metric_space (fspace S) (fdist S)*

proof

show $*$: $0 \leq \text{fdist } S\ f\ g$ **for** $f\ g$

by (auto simp: fdist_def intro: cSUP_upper2 [OF bdd_above_dist])

show $\text{fdist } S\ f\ g = \text{fdist } S\ g\ f$ **for** $f\ g$

by (auto simp: fdist_def commute)

show $(\text{fdist } S\ f\ g = 0) = (f = g)$

if $fg: f \in \text{fspace } S\ g \in \text{fspace } S$ **for** $f\ g$

proof

assume $0: \text{fdist } S\ f\ g = 0$

show $f = g$

proof (cases $S = \{\}$)

case True

with 0 that **show** ?thesis

by (simp add: fdist_def fspace_def)

next

case False

with $0\ fg$ **have** $\text{Sup}0: (\text{SUP } x \in S. d\ (f\ x)\ (g\ x)) = 0$

by (simp add: fdist_def)

have $d\ (f\ x)\ (g\ x) = 0$ **if** $x \in S$ **for** x

by (smt (verit) False Sup0 $\langle x \in S \rangle$ bdd_above_dist [OF fg] less_cSUP_iff nonneg)

with fg **show** $f = g$

by (simp add: fspace_def extensionalityI image_subset_iff)

qed

next

show $f = g \implies \text{fdist } S\ f\ g = 0$

using fspace_in_M [OF $\langle g \in \text{fspace } S \rangle$] by (auto simp: fdist_def)

qed

```

show  $f\text{dist } S f h \leq f\text{dist } S f g + f\text{dist } S g h$ 
  if  $fgh: f \in \text{fspace } S \ g \in \text{fspace } S \ h \in \text{fspace } S$  for  $f g h$ 
proof (clarsimp simp add: fdist_def that)
  assume  $S \neq \{\}$ 
  have  $d\text{fh}: d (f x) (h x) \leq d (f x) (g x) + d (g x) (h x)$  if  $x \in S$  for  $x$ 
    by (meson fgh fspace_in_M that triangle)
  have  $bdd\_fgh: bdd\_above ((\lambda x. d (f x) (g x)) ' S) \ bdd\_above ((\lambda x. d (g x) (h x)) ' S)$ 
    by (simp_all add: \langle S \neq \{\} \rangle bdd_above_dist that)
  then obtain  $B C$  where  $B: \bigwedge x. x \in S \implies d (f x) (g x) \leq B$  and  $C: \bigwedge x. x \in S \implies d (g x) (h x) \leq C$ 
    by (auto simp: bdd_above_def)
  then have  $\bigwedge x. x \in S \implies d (f x) (g x) + d (g x) (h x) \leq B + C$ 
    by force
  then have  $bdd: bdd\_above ((\lambda x. d (f x) (g x) + d (g x) (h x)) ' S)$ 
    by (auto simp: bdd_above_def)
  then have  $(\text{SUP } x \in S. d (f x) (h x)) \leq (\text{SUP } x \in S. d (f x) (g x) + d (g x) (h x))$ 
    by (metis (mono_tags, lifting) cSUP_mono \langle S \neq \{\} \rangle dfh)
  also have  $\dots \leq (\text{SUP } x \in S. d (f x) (g x)) + (\text{SUP } x \in S. d (g x) (h x))$ 
    by (simp add: \langle S \neq \{\} \rangle bdd cSUP_le_iff bdd_fgh add_mono cSup_upper)
  finally show  $(\text{SUP } x \in S. d (f x) (h x)) \leq (\text{SUP } x \in S. d (f x) (g x)) + (\text{SUP } x \in S. d (g x) (h x))$  .
qed
qed
end

```

definition funspace where

$$\text{funspace } S m \equiv \text{metric } (\text{Metric_space.fspace } (m\text{space } m) (m\text{dist } m)) S, \\ \text{Metric_space.fdist } (m\text{space } m) (m\text{dist } m) S$$

lemma mspace_funspace [simp]:

$$m\text{space } (\text{funspace } S m) = \text{Metric_space.fspace } (m\text{space } m) (m\text{dist } m) S$$

by (*simp add: Metric_space.Metric_space_funspace Metric_space.mspace_metric funspace_def*)

lemma mdist_funspace [simp]:

$$m\text{dist } (\text{funspace } S m) = \text{Metric_space.fdist } (m\text{space } m) (m\text{dist } m) S$$

by (*simp add: Metric_space.Metric_space_funspace Metric_space.mdist_metric funspace_def*)

lemma funspace_imp_welldefined:

$$\llbracket f \in m\text{space } (\text{funspace } S m); x \in S \rrbracket \implies f x \in m\text{space } m$$

by (*simp add: Metric_space.fspace_def subset_iff*)

lemma funspace_imp_extensional:

$$f \in m\text{space } (\text{funspace } S m) \implies f \in \text{extensional } S$$

by (simp add: Metric_space.fspace_def)

lemma funspace_imp_bounded_image:

$f \in \text{mspace} (\text{funspace } S \ m) \implies \text{Metric_space.mbounded} (\text{mspace } m) (\text{mdist } m)$
 $(f \ ' \ S)$

by (simp add: Metric_space.fspace_def)

lemma funspace_imp_bounded:

$f \in \text{mspace} (\text{funspace } S \ m) \implies S = \{\} \vee (\exists c \ B. \forall x \in S. \text{mdist } m \ c \ (f \ x) \leq B)$

by (auto simp: Metric_space.fspace_def Metric_space.mbounded)

lemma (in Metric_space) funspace_imp_bounded2:

assumes $f \in \text{fspace } S \ g \in \text{fspace } S$

obtains B where $\bigwedge x. x \in S \implies d (f \ x) (g \ x) \leq B$

proof –

have $\text{mbounded} (f \ ' \ S \cup g \ ' \ S)$

using mbounded_Un assms by (force simp: fspace_def)

then show *thesis*

by (metis UnCI imageI mbounded_alt that)

qed

lemma funspace_imp_bounded2:

assumes $f \in \text{mspace} (\text{funspace } S \ m) \ g \in \text{mspace} (\text{funspace } S \ m)$

obtains B where $\bigwedge x. x \in S \implies \text{mdist } m (f \ x) (g \ x) \leq B$

by (metis Metric_space_mspace_mdists assms mspace_funspace Metric_space.funspace_imp_bounded2)

lemma (in Metric_space) funspace_mdists_le:

assumes $fg: f \in \text{fspace } S \ g \in \text{fspace } S$ and $S \neq \{\}$

shows $\text{fdist } S \ f \ g \leq a \iff (\forall x \in S. d (f \ x) (g \ x) \leq a)$

using *assms* bdd_above_dist [OF *fg*] by (simp add: fdist_def cSUP_le_iff)

lemma funspace_mdists_le:

assumes $f \in \text{mspace} (\text{funspace } S \ m) \ g \in \text{mspace} (\text{funspace } S \ m)$ and $S \neq \{\}$

shows $\text{mdist} (\text{funspace } S \ m) \ f \ g \leq a \iff (\forall x \in S. \text{mdist } m (f \ x) (g \ x) \leq a)$

using *assms* by (simp add: Metric_space.funspace_mdists_le)

lemma (in Metric_space) mcomplete_funspace:

assumes *mcomplete*

shows *mcomplete_of* (funspace *S* *Self*)

proof –

interpret *F*: Metric_space fspace *S* fdist *S*

by (simp add: Metric_space_funspace)

show *?thesis*

proof (cases $S = \{\}$)

case *True*

then show *?thesis*

by (simp add: mcomplete_of_def mcomplete_trivial_singleton)

```

next
  case False
  show ?thesis
  proof (clarsimp simp: mcomplete_of_def Metric_space.mcomplete_def)
    fix  $\sigma$ 
    assume  $\sigma$ : F.MCauchy  $\sigma$ 
    then have  $\sigma M$ :  $\bigwedge n x. x \in S \implies \sigma n x \in M$ 
      by (auto simp: F.MCauchy_def intro: fspace_in_M)
    have  $fdist\_less$ :  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow fdist S (\sigma n) (\sigma n') <$ 
 $\varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
      using  $\sigma$  that by (auto simp: F.MCauchy_def)
    have  $\sigma ext$ :  $\bigwedge n. \sigma n \in extensional S$ 
      using  $\sigma$  unfolding F.MCauchy_def by (auto simp: fspace_def)
    have  $\sigma bd$ :  $\bigwedge n. mbounded (\sigma n ' S)$ 
      using  $\sigma$  unfolding F.MCauchy_def by (simp add: fspace_def image_subset_iff)
    have  $\sigma in[simp]$ :  $\sigma n \in fspace S$  for  $n$ 
      using F.MCauchy_def  $\sigma$  by blast
    have  $bd2$ :  $\bigwedge n n'. \exists B. \forall x \in S. d (\sigma n x) (\sigma n' x) \leq B$ 
      using  $\sigma$  unfolding F.MCauchy_def by (metis range_subsetD funspace_imp_bounded2)
    have  $sup$ :  $\bigwedge n n' x0. x0 \in S \implies d (\sigma n x0) (\sigma n' x0) \leq Sup ((\lambda x. d (\sigma n x)$ 
 $(\sigma n' x)) ' S)$ 
      proof (rule cSup_upper)
        show  $bdd\_above ((\lambda x. d (\sigma n x) (\sigma n' x)) ' S)$  if  $x0 \in S$  for  $n n' x0$ 
          using that  $bd2$  by (meson  $bdd\_above.I2$ )
      qed auto
    have  $pcy$ : MCauchy  $(\lambda n. \sigma n x)$  if  $x \in S$  for  $x$ 
      unfolding MCauchy_def
    proof (intro conjI strip)
      show  $range (\lambda n. \sigma n x) \subseteq M$ 
        using  $\sigma M$  that by blast
      fix  $\varepsilon :: real$ 
      assume  $\varepsilon > 0$ 
      then obtain  $N$  where  $N$ :  $\bigwedge n n'. N \leq n \longrightarrow N \leq n' \longrightarrow fdist S (\sigma n) (\sigma$ 
 $n') < \varepsilon$ 
        using  $\sigma$  by (force simp: F.MCauchy_def)
      { fix  $n n'$ 
        assume  $n$ :  $N \leq n \wedge N \leq n'$ 
        have  $d (\sigma n x) (\sigma n' x) \leq (SUP x \in S. d (\sigma n x) (\sigma n' x))$ 
          using that  $sup$  by presburger
        then have  $d (\sigma n x) (\sigma n' x) \leq fdist S (\sigma n) (\sigma n')$ 
          by (simp add:  $fdist\_def \langle S \neq \{\} \rangle$ )
        with  $N n$  have  $d (\sigma n x) (\sigma n' x) < \varepsilon$ 
          by fastforce
      } then show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (\sigma n x) (\sigma n' x) < \varepsilon$ 
        by blast
    qed
    have  $\exists l. limitin mtopology (\lambda n. \sigma n x) l$  sequentially if  $x \in S$  for  $x$ 
      using  $assms$  mcomplete_def  $pcy \langle x \in S \rangle$  by presburger
    then obtain  $g0$  where  $g0$ :  $\bigwedge x. x \in S \implies limitin mtopology (\lambda n. \sigma n x) (g0)$ 

```

x) *sequentially*
by *metis*
define g **where** $g \equiv \text{restrict } g0 \ S$
have $gext$: $g \in \text{extensional } S$
and $glim$: $\bigwedge x. x \in S \implies \text{limitin } mtopology (\lambda n. \sigma \ n \ x) (g \ x) \text{ sequentially}$
by (*auto simp: g_def g0*)
have gwd : $g \ x \in M$ **if** $x \in S$ **for** x
using $glim \ \text{limitin_metric}$ **that** **by** *blast*
have $unif$: $\exists N. \forall x \ n. x \in S \longrightarrow N \leq n \longrightarrow d (\sigma \ n \ x) (g \ x) < \varepsilon$ **if** $\varepsilon > 0$ **for**

ε
proof –
obtain N **where** N : $\bigwedge n \ n'. N \leq n \wedge N \leq n' \implies \text{Sup } ((\lambda x. d (\sigma \ n \ x) (\sigma \ n' \ x)) \ 'S) < \varepsilon/2$
using $\langle S \neq \{\} \rangle \ \langle \varepsilon > 0 \rangle \ \text{fdist_less [of } \varepsilon/2]$
by (*metis (mono_tags) sigma_in_fdist_def half_gt_zero*)
show *?thesis*
proof (*intro exI strip*)
fix $x \ n$
assume $x \in S$ **and** $N \leq n$
obtain N' **where** N' : $\bigwedge n. N' \leq n \implies \sigma \ n \ x \in M \wedge d (\sigma \ n \ x) (g \ x) < \varepsilon/2$
by (*metis* $\langle 0 < \varepsilon \rangle \ \langle x \in S \rangle \ glim \ half_gt_zero \ limit_metric_sequentially$)
have $d (\sigma \ n \ x) (g \ x) \leq d (\sigma \ n \ x) (\sigma (\max N N') \ x) + d (\sigma (\max N N') \ x) (g \ x)$
using $\langle x \in S \rangle \ \sigma M \ gwd \ triangle$ **by** *presburger*
also **have** $\dots < \varepsilon/2 + \varepsilon/2$
by (*smt (verit) N N'* $\langle N \leq n \rangle \ \langle x \in S \rangle \ max.cobounded1 \ max.cobounded2$)
sup)
finally **show** $d (\sigma \ n \ x) (g \ x) < \varepsilon$ **by** *simp*
qed
qed
have *limitin F.mtopology sigma g sequentially*
unfolding *F.limit_metric_sequentially*
proof (*intro conjI strip*)
obtain N **where** N : $\bigwedge n \ n'. N \leq n \wedge N \leq n' \implies \text{Sup } ((\lambda x. d (\sigma \ n \ x) (\sigma \ n' \ x)) \ 'S) < 1$
using *fdist_less [of 1]* $\langle S \neq \{\} \rangle$ **by** (*auto simp: fdist_def*)
have $\bigwedge x. x \in \sigma \ N \ 'S \implies x \in M$
using σM **by** *blast*
obtain $a \ B$ **where** $a \in M$ **and** B : $\bigwedge x. x \in (\sigma \ N) \ 'S \implies d \ a \ x \leq B$
by (*metis False sigma sigma_bd ex_in_conv imageI mbounded_alt_pos*)
have $d \ a \ (g \ x) \leq B+1$ **if** $x \in S$ **for** x
proof –
have $d \ a \ (g \ x) \leq d \ a \ (\sigma \ N \ x) + d (\sigma \ N \ x) (g \ x)$
by (*simp add:* $\langle a \in M \rangle \ \sigma M \ gwd$ *that triangle*)
also **have** $\dots \leq B+1$
proof –
have $d \ a \ (\sigma \ N \ x) \leq B$
by (*simp add: B that*)


```

moreover
have False if  $1: d (\sigma N x) (g x) > 1$ 
proof -
  obtain  $r$  where  $1 < r$  and  $r: r < d (\sigma N x) (g x)$ 
  using 1 dense by blast
  then obtain  $N'$  where  $N': \bigwedge n. N' \leq n \implies \sigma n x \in M \wedge d (\sigma n x)$ 
 $(g x) < r-1$ 
  using glim [OF <x∈S>] by (fastforce simp: limit_metric_sequentially)
  have  $d (\sigma N x) (g x) \leq d (\sigma N x) (\sigma (\max N N') x) + d (\sigma (\max N$ 
 $N') x) (g x)$ 
  by (metis <x ∈ S> σM commute gwd triangle')
  also have  $\dots < 1 + (r-1)$ 
  by (smt (verit) N N' <x ∈ S> max.cobounded1 max.cobounded2
 $\max.idem sup$ )
  finally have  $d (\sigma N x) (g x) < r$ 
  by simp
  with  $r$  show False
  by linarith
qed
ultimately show ?thesis
by force
qed
finally show ?thesis .
qed
with  $gwd \langle a \in M \rangle$  have mbounded (g ' S)
  unfolding mbounded by blast
with  $gwd \text{ gext}$  show  $g \in \text{fspace } S$ 
  by (auto simp: fspace_def)
fix  $\varepsilon::\text{real}$ 
assume  $\varepsilon > 0$ 
then obtain  $N$  where  $\bigwedge x n. x \in S \implies N \leq n \implies d (\sigma n x) (g x) < \varepsilon/2$ 
  by (meson unif_half_gt_zero)
then have  $\text{fdist } S (\sigma n) g \leq \varepsilon/2$  if  $N \leq n$  for  $n$ 
  using  $\langle g \in \text{fspace } S \rangle$  False that
  by (force simp: funspace_mdist_le simp del: divide_const_simps)
then show  $\exists N. \forall n \geq N. \sigma n \in \text{fspace } S \wedge \text{fdist } S (\sigma n) g < \varepsilon$ 
  by (metis <0 < ε> σin add_strict_increasing field_sum_of_halves
 $\text{half\_gt\_zero}$ )
qed
then show  $\exists x. \text{limitin } F.\text{mtopology } \sigma x$  sequentially
  by blast
qed
qed
qed

```

6.11.10 Metric space of continuous bounded functions

definition *cfunspace* **where**

cfunspace $X m \equiv \text{submetric } (\text{funspace } (\text{topspace } X) m) \{f. \text{continuous_map } X$

$(\text{mtopology_of } m) f\}$

lemma *mspace_cfunspace [simp]*:

$\text{mspace } (\text{cfunspace } X m) =$
 $\{f. f \in \text{topspace } X \rightarrow \text{mspace } m \wedge f \in \text{extensional } (\text{topspace } X) \wedge$
 $\text{Metric_space.mbounded } (\text{mspace } m) (\text{mdist } m) (f \text{ ' } (\text{topspace } X)) \wedge$
 $\text{continuous_map } X (\text{mtopology_of } m) f\}$
by (*auto simp: cfunspace_def Metric_space.fspace_def*)

lemma *mdist_cfunspace_eq_mdistspace*:

$\text{mdist } (\text{cfunspace } X m) = \text{mdist } (\text{funspace } (\text{topspace } X) m)$
by (*auto simp: cfunspace_def*)

lemma *cfunspace_subset_funspace*:

$\text{mspace } (\text{cfunspace } X m) \subseteq \text{mspace } (\text{funspace } (\text{topspace } X) m)$
by (*simp add: cfunspace_def*)

lemma *cfunspace_mdistspace_le*:

$\llbracket f \in \text{mspace } (\text{cfunspace } X m); g \in \text{mspace } (\text{cfunspace } X m); \text{topspace } X \neq \{\}\rrbracket$
 $\implies \text{mdist } (\text{cfunspace } X m) f g \leq a \iff (\forall x \in \text{topspace } X. \text{mdist } m (f x) (g x) \leq a)$
by (*simp add: cfunspace_def Metric_space.funspace_mdistspace_le*)

lemma *cfunspace_imp_bounded2*:

assumes $f \in \text{mspace } (\text{cfunspace } X m) \ g \in \text{mspace } (\text{cfunspace } X m)$
obtains B **where** $\bigwedge x. x \in \text{topspace } X \implies \text{mdist } m (f x) (g x) \leq B$
by (*metis assms all_not_in_conv cfunspace_mdistspace_le nle_le*)

lemma *cfunspace_mdistspace_lt*:

$\llbracket \text{compactin } X (\text{topspace } X); f \in \text{mspace } (\text{cfunspace } X m);$
 $g \in \text{mspace } (\text{cfunspace } X m); \text{mdist } (\text{cfunspace } X m) f g < a;$
 $x \in \text{topspace } X \rrbracket$
 $\implies \text{mdist } m (f x) (g x) < a$

by (*metis (full_types) cfunspace_mdistspace_le empty_iff_less_eq_real_def less_le_not_le*)

lemma *mdistspace_cfunspace_le*:

assumes $0 \leq B$ **and** $B: \bigwedge x. x \in \text{topspace } X \implies \text{mdist } m (f x) (g x) \leq B$
shows $\text{mdist } (\text{cfunspace } X m) f g \leq B$

proof (*cases X = trivial_topology*)

case *True*

then show *?thesis*

by (*simp add: Metric_space.fdist_empty =0> cfunspace_def*)

next

case *False*

have *bdd*: $\text{bdd_above } ((\lambda u. \text{mdist } m (f u) (g u)) \text{ ' } \text{topspace } X)$

by (*meson B bdd_above.I2*)

with *assms bdd* **show** *?thesis*

by (*simp add: mdistspace_cfunspace_eq_mdistspace Metric_space.fdist_def cSUP_le_iff*)

qed

lemma *mdist_cfunspace_imp_mdists_le*:

$\llbracket f \in \text{mspace } (\text{cfunspace } X \ m); g \in \text{mspace } (\text{cfunspace } X \ m);$
 $\text{mdist } (\text{cfunspace } X \ m) \ f \ g \leq a; x \in \text{topspace } X \rrbracket \implies \text{mdist } m \ (f \ x) \ (g \ x) \leq a$
using *cfunspace_mdists_le* **by** *blast*

lemma *compactin_mspace_cfunspace*:

$\text{compactin } X \ (\text{topspace } X)$
 $\implies \text{mspace } (\text{cfunspace } X \ m) =$
 $\{f. (\forall x \in \text{topspace } X. f \ x \in \text{mspace } m) \wedge$
 $f \in \text{extensional } (\text{topspace } X) \wedge$
 $\text{continuous_map } X \ (\text{mtopology_of } m) \ f\}$

by (*auto simp: Metric_space.compactin_imp_mbounded image_compactin_mtopology_of_def*)

lemma (**in** *Metric_space*) *mcomplete_cfunspace*:

assumes *mcomplete*
shows *mcomplete_of* (*cfunspace* *X* *Self*)

proof –

interpret *F*: *Metric_space* *fspace* (*topspace* *X*) *fdist* (*topspace* *X*)

by (*simp add: Metric_space_funspace*)

interpret *S*: *Submetric* *fspace* (*topspace* *X*) *fdist* (*topspace* *X*) *mspace* (*cfunspace* *X* *Self*)

proof

show *mspace* (*cfunspace* *X* *Self*) \subseteq *fspace* (*topspace* *X*)

by (*metis cfunspace_subset_funspace mdists_Self mspace_Self mspace_funspace*)

qed

show *?thesis*

proof (*cases* *X* = *trivial_topology*)

case *True*

then show *?thesis*

by (*simp add: mcomplete_of_def mcomplete_trivial_singleton mdists_cfunspace_eq_mdists_funspace cong: conj_cong*)

next

case *False*

have $*$: *continuous_map* *X* *mtopology* *g*

if *range* $\sigma \subseteq \text{mspace } (\text{cfunspace } X \ \text{Self})$

and *g*: *limitin* *F*.*mtopology* σ *g* *sequentially* **for** σ *g*

unfolding *continuous_map_to_metric*

proof (*intro strip*)

have σ : $\bigwedge n. \text{continuous_map } X \ \text{mtopology} \ (\sigma \ n)$

using *that* **by** (*auto simp: mtopology_of_def*)

fix *x* **and** $\varepsilon :: \text{real}$

assume $x \in \text{topspace } X$ **and** $0 < \varepsilon$

then obtain *N* **where** $N: \bigwedge n. N \leq n \implies \sigma \ n \in \text{fspace } (\text{topspace } X) \wedge \text{fdist}$
 $(\text{topspace } X) \ (\sigma \ n) \ g < \varepsilon/3$

unfolding *mtopology_of_def* *F.limitin_metric*

```

by (metis F.limit_metric_sequentially divide_pos_pos g zero_less_numeral)

then obtain U where openin X U x ∈ U
  and U:  $\bigwedge y. y \in U \implies \sigma N y \in \text{mball } (\sigma N x) (\varepsilon/3)$ 
  by (metis Metric_space.continuous_map_to_metric Metric_space_axioms
  <0 < ε> <x ∈ topspace X> σ divide_pos_pos zero_less_numeral)
moreover
have g y ∈ mball (g x) ε if y ∈ U for y
proof -
  have U ⊆ topspace X
    using <openin X U> by (simp add: openin_subset)
  have gx: g x ∈ M
    by (meson F.limitin_mspace <x ∈ topspace X> fspace_in_M g)
  have y ∈ topspace X
    using <U ⊆ topspace X> that by auto
  have gy: g y ∈ M
    by (meson F.limitin_mspace[OF g] <U ⊆ topspace X> fspace_in_M subsetD
  that)
  have d (g x) (g y) < ε
  proof -
    have *: d (σ N x0) (g x0) ≤ ε/3 if x0 ∈ topspace X for x0
    proof -
      have g ∈ fspace (topspace X)
        using F.limit_metric_sequentially g by blast
      with N that have bdd_above ((λx. d (σ N x) (g x)) ‘ topspace X)
        by (force intro: bdd_above_dist)
      then have d (σ N x0) (g x0) ≤ Sup ((λx. d (σ N x) (g x)) ‘ topspace X)
        by (simp add: cSup_upper that)
      also have ... ≤ ε/3
        using g False N <g ∈ fspace (topspace X)>
        by (fastforce simp: F.limit_metric_sequentially fdist_def)
      finally show ?thesis .
    qed
  have d (g x) (g y) ≤ d (g x) (σ N x) + d (σ N x) (g y)
    using U gx gy that triangle by force
  also have ... < ε/3 + ε/3 + ε/3
    by (smt (verit) * U gy <x ∈ topspace X> <y ∈ topspace X> commute
  in_mball that triangle)
  finally show ?thesis by simp
  qed
  with gx gy show ?thesis by simp
  qed
ultimately show  $\exists U. \text{openin } X U \wedge x \in U \wedge (\forall y \in U. g y \in \text{mball } (g x) \varepsilon)$ 
  by blast
qed

have S.sub.mcomplete
proof (rule S.sequentially_closedin_mcomplete_imp_mcomplete)
show F.mcomplete

```

```

  by (metis assms mcomplete_funspace mcomplete_of_def mdist_Self mdist_funspace
      mspace_Self mspace_funspace)
  fix  $\sigma$   $g$ 
  assume  $g$ : range  $\sigma \subseteq$  mspace (cfunspace  $X$  Self)  $\wedge$  limitin  $F$ .mtopology  $\sigma$   $g$ 
  sequentially
  show  $g \in$  mspace (cfunspace  $X$  Self)
  proof (simp add: mtopology_of_def, intro conjI)
    show  $g \in$  topspace  $X \rightarrow M$   $g \in$  extensional (topspace  $X$ ) mbounded ( $g$  ‘
      topspace  $X$ )
    using  $g$   $F$ .limitin_mspace by (force simp: fspace_def)+
    show continuous_map  $X$  mtopology  $g$ 
    using *  $g$  by blast
  qed
  qed
  then show ?thesis
  by (simp add: mcomplete_of_def mdist_cfunspace_eq_md_dist_funspace)
  qed
  qed

```

6.11.11 Existence of completion for any metric space M as a subspace of $M \Rightarrow \mathbb{R}$

lemma (in Metric_space) metric_completion_explicit:

```

  obtains  $f :: [a, a] \Rightarrow$  real and  $S$  where
     $S \subseteq$  mspace(funspace  $M$  euclidean_metric)
    mcomplete_of (submetric (funspace  $M$  euclidean_metric)  $S$ )
     $f \in M \rightarrow S$ 
    mtopology_of(funspace  $M$  euclidean_metric) closure_of  $f$  ‘  $M = S$ 
     $\wedge x$   $y$ .  $\llbracket x \in M; y \in M \rrbracket$ 
       $\implies$  mdist (funspace  $M$  euclidean_metric) ( $f$   $x$ ) ( $f$   $y$ ) =  $d$   $x$   $y$ 

```

proof –

```

  define  $m'$ :: ('a $\Rightarrow$ real) metric where  $m' \equiv$  funspace  $M$  euclidean_metric

```

```

  show thesis

```

```

  proof (cases  $M = \{\}$ )

```

```

    case True

```

```

      then show ?thesis

```

```

        using that by (simp add: mcomplete_of_def mcomplete_trivial)

```

```

    next

```

```

      case False

```

```

        then obtain  $a$  where  $a \in M$ 

```

```

          by auto

```

```

        define  $f$  where  $f \equiv \lambda x. (\lambda u \in M. d$   $x$   $u - d$   $a$   $u)$ 

```

```

        define  $S$  where  $S \equiv$  mtopology_of(funspace  $M$  euclidean_metric) closure_of
      ( $f$  ‘  $M$ )

```

```

        interpret  $S$ : Submetric Met_TC.fspace  $M$  Met_TC.fdist  $M$   $S \cap$  Met_TC.fspace
       $M$ 

```

```

        by (simp add: Met_TC.Metric_space_funspace Submetric.intro Submet-
      ric_axioms_def)

```

```

have fim: f ' M ⊆ mspace m'
proof (clarsimp simp: m'_def Met_TC.fspace_def)
  fix b
  assume b ∈ M
  then have  $\bigwedge c. [c \in M] \implies |d b c - d a c| \leq d a b$ 
    by (smt (verit, best) ⟨a ∈ M⟩ commute_triangle'')
  then have  $(\lambda x. d b x - d a x) ' M \subseteq cball\ 0\ (d a b)$ 
    by force
  then show f b ∈ extensional M ∧ bounded (f b ' M)
    by (metis bounded_cball bounded_subset f_def image_restrict_eq re-
strict_extensional_subset_eq_subset)
qed
show thesis
proof
  show S ⊆ mspace (funspace M euclidean_metric)
    by (simp add: S_def in_closure_of_subset_iff)
  have closedin S.mtopology (S ∩ Met_TC.fspace M)
    by (simp add: S_def closedin_Int funspace_def)
  moreover have S.mcomplete
    using Metric_space.mcomplete_funspace Met_TC.Metric_space_axioms
by (fastforce simp: mcomplete_of_def)
  ultimately show mcomplete_of (submetric (funspace M euclidean_metric)
S)
    by (simp add: S.closedin_eq_mcomplete mcomplete_of_def)
  show f ∈ M → S
    using S_def fim in_closure_of m'_def by fastforce
  show mtopology_of (funspace M euclidean_metric) closure_of f ' M = S
    by (auto simp: f_def S_def mtopology_of_def)
  show mdist (funspace M euclidean_metric) (f x) (f y) = d x y
    if x ∈ M y ∈ M for x y
  proof -
    have  $\forall c \in M. dist (f x c) (f y c) \leq r \implies d x y \leq r$  for r
      using that by (auto simp: f_def dist_real_def)
    moreover have  $dist (f x z) (f y z) \leq r$  if  $d x y \leq r$  and  $z \in M$  for r z
      using that ⟨x ∈ M⟩ ⟨y ∈ M⟩
    apply (simp add: f_def Met_TC.fdist_def dist_real_def)
    by (smt (verit, best) commute_triangle')
    ultimately have  $(SUP c \in M. dist (f x c) (f y c)) = d x y$ 
      by (intro cSup_unique) auto
    with that fim show ?thesis
      using that fim by (simp add: Met_TC.fdist_def False m'_def im-
age_subset_iff)
    qed
  qed
qed
qed

```

lemma (in Metric_space) metric_completion:

```

obtains  $f :: [a, 'a] \Rightarrow \text{real}$  and  $m'$  where
   $mcomplete\_of\ m'$ 
   $f \in M \rightarrow mspace\ m'$ 
   $mtopology\_of\ m'\ closure\_of\ f\ 'M = mspace\ m'$ 
   $\bigwedge x\ y. \llbracket x \in M; y \in M \rrbracket \Longrightarrow mdist\ m'\ (f\ x)\ (f\ y) = d\ x\ y$ 
proof -
obtain  $f :: [a, 'a] \Rightarrow \text{real}$  and  $S$  where
   $Ssub: S \subseteq mspace(\text{funspace}\ M\ euclidean\_metric)$ 
  and  $mcom: mcomplete\_of\ (submetric\ (\text{funspace}\ M\ euclidean\_metric)\ S)$ 
  and  $fm: f \in M \rightarrow S$ 
  and  $eqS: mtopology\_of(\text{funspace}\ M\ euclidean\_metric)\ closure\_of\ f\ 'M = S$ 
  and  $eqd: \bigwedge x\ y. \llbracket x \in M; y \in M \rrbracket \Longrightarrow mdist\ (\text{funspace}\ M\ euclidean\_metric)\ (f\ x)\ (f\ y) = d\ x\ y$ 
using  $metric\_completion\_explicit$  by  $metis$ 
define  $m'$  where  $m' \equiv submetric\ (\text{funspace}\ M\ euclidean\_metric)\ S$ 
show  $thesis$ 
proof
  show  $mcomplete\_of\ m'$ 
    by  $(simp\ add: mcom\ m'\_def)$ 
  show  $f \in M \rightarrow mspace\ m'$ 
    using  $Ssub\ fm\ m'\_def$  by  $auto$ 
  show  $mtopology\_of\ m'\ closure\_of\ f\ 'M = mspace\ m'$ 
    using  $eqS\ fm\ Ssub$ 
    by  $(force\ simp: m'\_def\ mtopology\_of\_submetric\ closure\_of\_subtopology\ Int\_absorb1\ image\_subset\_iff\_funcset)$ 
  show  $mdist\ m'\ (f\ x)\ (f\ y) = d\ x\ y$  if  $x \in M$  and  $y \in M$  for  $x\ y$ 
    using  $that\ eqd\ m'\_def$  by  $force$ 
qed
qed

lemma  $metrizable\_space\_completion:$ 
assumes  $metrizable\_space\ X$ 
obtains  $f :: [a, 'a] \Rightarrow \text{real}$  and  $Y$  where
   $completely\_metrizable\_space\ Y\ embedding\_map\ X\ Y\ f$ 
   $Y\ closure\_of\ (f\ '(\text{topspace}\ X)) = \text{topspace}\ Y$ 
proof -
obtain  $M\ d$  where  $Metric\_space\ M\ d$  and  $Xeq: X = Metric\_space.mtopology\ M\ d$ 
using  $assms\ metrizable\_space\_def$  by  $blast$ 
then interpret  $Metric\_space\ M\ d$  by  $simp$ 
obtain  $f :: [a, 'a] \Rightarrow \text{real}$  and  $m'$  where
   $mcomplete\_of\ m'$ 
  and  $fm: f \in M \rightarrow mspace\ m'$ 
  and  $m': mtopology\_of\ m'\ closure\_of\ f\ 'M = mspace\ m'$ 
  and  $eqd: \bigwedge x\ y. \llbracket x \in M; y \in M \rrbracket \Longrightarrow mdist\ m'\ (f\ x)\ (f\ y) = d\ x\ y$ 
  by  $(metis\ metric\_completion)$ 
show  $thesis$ 
proof
  show  $completely\_metrizable\_space\ (mtopology\_of\ m')$ 

```

```

    using ⟨mcomplete_of m'⟩
  unfolding completely_metrizable_space_def mcomplete_of_def mtopology_of_def
    by (metis Metric_space_mspace_mdistr)
  show embedding_map X (mtopology_of m') f
    using Metric_space12.isometry_imp_embedding_map
  by (metis Metric_space12_def Metric_space_axioms Metric_space_mspace_mdistr
    Xeq eqd fim
      mtopology_of_def)
  show (mtopology_of m') closure_of f ' topspace X = topspace (mtopology_of
    m')
    by (simp add: Xeq m')
  qed
qed

```

6.11.12 Contractions

lemma (in *Metric_space*) *contraction_imp_unique_fixpoint*:

```

  assumes f x = x f y = y
    and f ∈ M → M
    and k < 1
    and ∧x y. [x ∈ M; y ∈ M] ⇒ d (f x) (f y) ≤ k * d x y
    and x ∈ M y ∈ M
  shows x = y
  by (smt (verit, ccfv_SIG) mdistr_pos_less mult_le_cancel_right1 assms)

```

Banach Fixed-Point Theorem (aka, Contraction Mapping Principle)

lemma (in *Metric_space*) *Banach_fixedpoint_thm*:

```

  assumes mcomplete and M ≠ {} and fim: f ∈ M → M
    and k < 1
    and con: ∧x y. [x ∈ M; y ∈ M] ⇒ d (f x) (f y) ≤ k * d x y
  obtains x where x ∈ M f x = x

```

proof –

```

  obtain a where a ∈ M
    using ⟨M ≠ {}⟩ by blast
  show thesis
  proof (cases ∀x ∈ M. f x = f a)
    case True
    then show ?thesis
      by (metis ⟨a ∈ M⟩ fim image_subset_iff image_subset_iff_funcset that)
  next
  case False
  then obtain b where b ∈ M and b: f b ≠ f a
    by blast
  have k>0
    using Lipschitz_coefficient_pos [where f=f]
    by (metis False ⟨a ∈ M⟩ con fim mdistr_Self mspace_Self)
  define σ where σ ≡ λn. (f~n) a
  have f_iter: σ n ∈ M for n
    unfolding σ_def by (induction n) (use ⟨a ∈ M⟩ fim in auto)

```



```

show ?thesis
proof (cases f a = a)
  case True
  then show ?thesis
    using ⟨a ∈ M⟩ that by blast
next
  case False
  have MCauchy σ
  proof –
    show ?thesis
      unfolding MCauchy_def
    proof (intro conjI strip)
      show range σ ⊆ M
        using f_iter by blast
      fix ε::real
      assume ε>0
      with ⟨k < 1⟩ ⟨f a ≠ a⟩ ⟨a ∈ M⟩ fix m have gt0: ((1 - k) * ε) / d a (f a)
> 0
        by (fastforce simp: divide_simps Pi_iff)
      obtain N where k^N < ((1-k) * ε) / d a (f a)
        using real_arch_pow_inv [OF gt0 ⟨k < 1⟩] by blast
      then have N: ∧n. n ≥ N ⇒ k^n < ((1-k) * ε) / d a (f a)
        by (smt (verit) ⟨0 < k⟩ assms(4) power_decreasing)
      have ∀n n'. n < n' ⇒ N ≤ n ⇒ N ≤ n' ⇒ d (σ n) (σ n') < ε
      proof (intro exI strip)
        fix n n'
        assume n < n' N ≤ n N ≤ n'
        have d (σ n) (σ n') ≤ (∑ i=n..<n'. d (σ i) (σ (Suc i)))
        proof –
          have n < m ⇒ d (σ n) (σ m) ≤ (∑ i=n..<m. d (σ i) (σ (Suc i)))
for m
          proof (induction m)
            case 0
            then show ?case
              by simp
            next
            case (Suc m)
            then consider n < m | m = n
              by linarith
            then show ?case
            proof cases
              case 1
              have d (σ n) (σ (Suc m)) ≤ d (σ n) (σ m) + d (σ m) (σ (Suc m))
                by (simp add: f_iter triangle)
              also have ... ≤ (∑ i=n..<m. d (σ i) (σ (Suc i))) + d (σ m) (σ
(Suc m))
                using Suc 1 by linarith
              also have ... = (∑ i = n..<Suc m. d (σ i) (σ (Suc i)))
                using 1 by force

```

```

      finally show ?thesis .
    qed auto
  qed
  with ⟨n < n'⟩ show ?thesis by blast
  qed
  also have ... ≤ (∑ i=n..

```

```

show  $l \in M$ 
  using  $l$  limitin_mspace by blast
show  $f l = l$ 
proof (rule limitin_metric_unique)
  have limitin_mtopology  $(f \circ \sigma)$   $(f l)$  sequentially
  proof (rule continuous_map_limit)
    have Lipschitz_continuous_map_Self_Self  $f$ 
    using con by (auto simp: Lipschitz_continuous_map_def fm)
    then show continuous_map_mtopology_mtopology  $f$ 
    using Lipschitz_continuous_imp_continuous_map_Self_def by force
  qed (use  $l$  in auto)
  moreover have  $(f \circ \sigma) = (\lambda i. \sigma(i+1))$ 
  by (auto simp: sigma_def)
  ultimately show limitin_mtopology  $(\lambda n. (f \overset{\sim}{n}) a)$   $(f l)$  sequentially
  using limitin_sequentially_offset_rev [of mtopology  $\sigma$  1]
  by (simp add: sigma_def)
qed (use  $l$  in  $\langle$ auto simp: sigma_def $\rangle$ )
qed
qed
qed
qed

```

6.11.13 The Baire Category Theorem

Possibly relevant to the theorem "Baire" in Elementary Normed Spaces

lemma (in *Metric_space*) *metric_Baire_category*:

assumes *mcomplete* *countable* \mathcal{G}

and $\bigwedge T. T \in \mathcal{G} \implies \text{openin_mtopology } T \wedge \text{mtopology_closure_of } T = M$

shows *mtopology_closure_of* $\bigcap \mathcal{G} = M$

proof (cases $\mathcal{G} = \{\}$)

case *False*

then obtain $U :: \text{nat} \Rightarrow 'a$ **set where** $U: \text{range } U = \mathcal{G}$

by (*metis* \langle countable \mathcal{G} \rangle *uncountable_def*)

with *assms* **have** $u_open: \bigwedge n. \text{openin_mtopology } (U\ n)$ **and** $u_dense: \bigwedge n. \text{mtopology_closure_of } (U\ n) = M$

by *auto*

have $\bigcap (\text{range } U) \cap W \neq \{\}$ **if** $W: \text{openin_mtopology } W$ $W \neq \{\}$ **for** W

proof –

have $W \subseteq M$

using *openin_mtopology* W **by** *blast*

have $\exists r' x'. 0 < r' \wedge r' < r/2 \wedge x' \in M \wedge \text{mcball } x' r' \subseteq \text{mball } x r \cap U\ n$

if $r > 0$ $x \in M$ **for** $x\ r\ n$

proof –

obtain z **where** $z: z \in U\ n \cap \text{mball } x r$

using u_dense [of n] \langle $r > 0$ \rangle \langle $x \in M$ \rangle

by (*metis* *dense_intersects_open_centre_in_mball_iff_empty_iff_openin_mball* *topspace_mtopology_equals0I*)

then have $z \in M$ **by** *auto*

have *openin_mtopology* $(U\ n \cap \text{mball } x r)$

```

    by (simp add: openin_Int u_open)
  with ⟨z ∈ M⟩ z obtain e where e>0 and e: mball z e ⊆ U n ∩ mball x r
    by (meson openin_mtopology_mball)
  define r' where r' ≡ min e (r/4)
  show ?thesis
  proof (intro exI conjI)
    show 0 < r' r' < r / 2 z ∈ M
      using ⟨e>0⟩ ⟨r>0⟩ ⟨z ∈ M⟩ by (auto simp: r'_def)
    show mball z r' ⊆ mball x r ∩ U n
      using Metric_space.mball_subset_concentric e r'_def by auto
  qed
  qed
  then obtain nextr nextx
    where nextr: ∧r x n. [r>0; x∈M] ⇒ 0 < nextr r x n ∧ nextr r x n < r/2
      and nextx: ∧r x n. [r>0; x∈M] ⇒ nextx r x n ∈ M
      and nextsub: ∧r x n. [r>0; x∈M] ⇒ mball (nextx r x n) (nextr r x n) ⊆
mball x r ∩ U n
    by metis
  obtain x0 where x0: x0 ∈ U 0 ∩ W
    by (metis W dense_intersects_open topspace_mtopology all_not_in_conv
u_dense)
  then have x0 ∈ M
    using ⟨W ⊆ M⟩ by fastforce
  obtain r0 where 0 < r0 r0 < 1 and sub: mball x0 r0 ⊆ U 0 ∩ W
  proof -
    have openin_mtopology (U 0 ∩ W)
      using W u_open by blast
    then obtain r where r>0 and r: mball x0 r ⊆ U 0 mball x0 r ⊆ W
      by (meson Int_subset_iff openin_mtopology x0)
    define r0 where r0 ≡ (min r 1) / 2
    show thesis
    proof
      show 0 < r0 r0 < 1
        using ⟨r>0⟩ by (auto simp: r0_def)
      show mball x0 r0 ⊆ U 0 ∩ W
        using r ⟨0 < r0⟩ r0_def by auto
    qed
  qed
  define b where b ≡ rec_nat (x0,r0) (λn (x,r). (nextx r x n, nextr r x n))
  have b0[simp]: b 0 = (x0,r0)
    by (simp add: b_def)
  have bSuc[simp]: b (Suc n) = (let (x,r) = b n in (nextx r x n, nextr r x n))
  for n
    by (simp add: b_def)
  define xf where xf ≡ fst ∘ b
  define rf where rf ≡ snd ∘ b
  have rxf: 0 < rf n ∧ xf n ∈ M for n
  proof (induction n)
    case 0

```

```

    with ⟨0 < r0⟩ ⟨x0 ∈ M⟩ show ?case
      by (auto simp: rf_def xf_def)
  next
    case (Suc n)
    then show ?case
      by (auto simp: rf_def xf_def case_prod_unfold nextx nextx Let_def)
  qed
  have mcball_sub: mcball (xf (Suc n)) (rf (Suc n)) ⊆ mball (xf n) (rf n) ∩ U
n for n
    using rxf nextsub by (auto simp: xf_def rf_def case_prod_unfold Let_def)
  have half: rf (Suc n) < rf n / 2 for n
    using rxf next by (auto simp: xf_def rf_def case_prod_unfold Let_def)
  then have decseq rf
    using rxf by (smt (verit, ccfv_threshold) decseq_SucI field_sum_of_halves)
  have nested: mball (xf n) (rf n) ⊆ mball (xf m) (rf m) if m ≤ n for m n
    using that
  proof (induction n)
    case (Suc n)
    then show ?case
      by (metis mcball_sub order.trans inf.boundedE le_Suc_eq mball_subset_mcball
order.refl)
  qed auto
  have MCauchy xf
    unfolding MCauchy_def
  proof (intro conjI strip)
    show range xf ⊆ M
      using rxf by blast
    fix ε :: real
    assume ε > 0
    then obtain N where N: inverse (2^N) < ε
      using real_arch_pow_inv by (force simp flip: power_inverse)
    have d (xf n) (xf n') < ε if n ≤ n' N ≤ n N ≤ n' for n n'
    proof -
      have *: rf n < inverse (2 ^ n) for n
      proof (induction n)
        case 0
        then show ?case
          by (simp add: ⟨r0 < 1⟩ rf_def)
      next
        case (Suc n)
        with half show ?case
          by simp (smt (verit))
      qed
    have rf n ≤ rf N
      using ⟨decseq rf⟩ ⟨N ≤ n⟩ by (simp add: decseqD)
    moreover
    have xf n' ∈ mball (xf n) (rf n)
      using nested rxf ⟨n ≤ n'⟩ by blast
    ultimately have d (xf n) (xf n') < rf N

```

```

    by auto
    also have ... < ε
      using * N order.strict_trans by blast
    finally show ?thesis .
  qed
  then show  $\exists N. \forall n n'. N \leq n \longrightarrow N \leq n' \longrightarrow d (xf n) (xf n') < \varepsilon$ 
    by (metis commute linorder_le_cases)
  qed
  then obtain l where l: limitin mtopology xf l sequentially
    using <mcomplete> mcomplete_alt by blast
  have l_in: l ∈ mcball (xf n) (rf n) for n
  proof -
    have  $\forall_F m$  in sequentially.  $xf m \in mcball (xf n) (rf n)$ 
      unfolding eventually_sequentially
      by (meson nested rxf centre_in_mball_iff mball_subset_mcball subset_iff)
    with l limitin_closedin show ?thesis
      by (metis closedin_mcball trivial_limit_sequentially)
  qed
  then have  $\bigwedge n. l \in U n$ 
    using mcball_sub by blast
  moreover have l ∈ W
    using l_in[of 0] sub by (auto simp: xf_def rf_def)
  ultimately show ?thesis by auto
  qed
  with U show ?thesis
    by (metis dense_intersects_open topspace_mtopology)
  qed auto

```

lemma (in Metric_space) metric_Baire_category_alt:

```

  assumes mcomplete countable G
    and empty:  $\bigwedge T. T \in \mathcal{G} \implies \text{closedin mtopology } T \wedge \text{mtopology interior\_of } T = \{\}$ 
  shows mtopology interior_of  $\bigcup \mathcal{G} = \{\}$ 
  proof -
    have *: mtopology closure_of  $\bigcap ((-) M \text{ ' } \mathcal{G}) = M$ 
  proof (intro metric_Baire_category conjI <mcomplete>)
    show countable ((-) M ' G)
      using <countable G> by blast
    fix T
    assume T ∈ (-) M ' G
    then obtain U where U: U ∈ G T = M - U U ⊆ M
      using empty metric_closedin_iff_sequentially_closed by force
    with empty show openin mtopology T by blast
    show mtopology closure_of T = M
      using U by (simp add: closure_of_interior_of double_diff_empty)
  qed
  with closure_of_eq show ?thesis

```

by (fastforce simp: interior_of_closure_of_split: if_split_asm)
qed

Since all locally compact Hausdorff spaces are regular, the disjunction in the HOL Light version is redundant.

lemma *Baire_category_aux*:

assumes *locally_compact_space* X *regular_space* X
and *countable* \mathcal{G}
and *empty*: $\bigwedge G. G \in \mathcal{G} \implies \text{closedin } X \ G \wedge X \text{ interior_of } G = \{\}$
shows $X \text{ interior_of } \bigcup \mathcal{G} = \{\}$
proof (cases $\mathcal{G} = \{\}$)
case *True*
then show ?thesis
by simp
next
case *False*
then obtain $T :: \text{nat} \Rightarrow 'a \text{ set}$ **where** $T: \mathcal{G} = \text{range } T$
by (metis <countable \mathcal{G} > uncountable_def)
with *empty* **have** *Tempty*: $\bigwedge n. X \text{ interior_of } (T \ n) = \{\}$
by auto
show ?thesis
proof (clarsimp simp: *T interior_of_def*)
fix $z \ U$
assume $z \in U$ **and** *opeA*: $\text{openin } X \ U$ **and** *Asub*: $U \subseteq \bigcup (\text{range } T)$
with *openin_subset* **have** $z \in \text{topspace } X$
by blast
have *neighbourhood_base_of* $(\lambda C. \text{compactin } X \ C \wedge \text{closedin } X \ C) \ X$
using *assms* *locally_compact_regular_space_neighbourhood_base* **by** auto
then obtain $V \ K$ **where** $\text{openin } X \ V \ \text{compactin } X \ K \ \text{closedin } X \ K \ z \in V \ V \subseteq K \ K \subseteq U$
by (metis (no_types, lifting) < $z \in U$ > neighbourhood_base_of opeA)
have *nb_closedin*: *neighbourhood_base_of* $(\text{closedin } X) \ X$
using <*regular_space* X > *neighbourhood_base_of_closedin* **by** auto
have $\exists \Phi. \forall n. (\Phi \ n \subseteq K \wedge \text{closedin } X \ (\Phi \ n) \wedge X \text{ interior_of } \Phi \ n \neq \{\}) \wedge \text{disjnt } (\Phi \ n) \ (T \ n) \wedge \Phi \ (\text{Suc } n) \subseteq \Phi \ n$
proof (rule *dependent_nat_choice*)
show $\exists x \subseteq K. \text{closedin } X \ x \wedge X \text{ interior_of } x \neq \{\} \wedge \text{disjnt } x \ (T \ 0)$
proof –
have *False* **if** $V \subseteq T \ 0$
using *Tempty* < $\text{openin } X \ V$ > < $z \in V$ > *interior_of_maximal_that* **by** fastforce
then obtain x **where** $\text{openin } X \ (V - T \ 0) \wedge x \in V - T \ 0$
using *T* < $\text{openin } X \ V$ > *empty* **by** blast
with *nb_closedin*
obtain $N \ C$ **where** $\text{openin } X \ N \ \text{closedin } X \ C \ x \in N \ N \subseteq C \ C \subseteq V - T \ 0$
unfolding *neighbourhood_base_of* **by** metis
show ?thesis
proof (intro *exI conjI*)
show $C \subseteq K$

```

    using ⟨C ⊆ V - T 0⟩ ⟨V ⊆ K⟩ by auto
    show X interior_of C ≠ {}
    by (metis ⟨N ⊆ C⟩ ⟨openin X N⟩ ⟨x ∈ N⟩ empty_iff interior_of_eq_empty)
    show disjnt C (T 0)
    using ⟨C ⊆ V - T 0⟩ disjnt_iff by fastforce
  qed (use ⟨closedin X C⟩ in auto)
qed
show ∃ L. (L ⊆ K ∧ closedin X L ∧ X interior_of L ≠ {} ∧ disjnt L (T (Suc
n))) ∧ L ⊆ C
  if §: C ⊆ K ∧ closedin X C ∧ X interior_of C ≠ {} ∧ disjnt C (T n)
  for C n
  proof -
    have False if X interior_of C ⊆ T (Suc n)
      by (metis Empty interior_of_eq_empty § openin_interior_of that)
    then obtain x where openin X (X interior_of C - T (Suc n)) ∧ x ∈ X
interior_of C - T (Suc n)
      using T empty by fastforce
    with nb_closedin
    obtain N D where openin X N closedin X D x ∈ N N ⊆ D and D: D ⊆
X interior_of C - T(Suc n)
      unfolding neighbourhood_base_of by metis
    show ?thesis
    proof (intro conjI exI)
      show D ⊆ K
      using D interior_of_subset § by fastforce
      show X interior_of D ≠ {}
      by (metis ⟨N ⊆ D⟩ ⟨openin X N⟩ ⟨x ∈ N⟩ empty_iff interior_of_eq_empty)
      show disjnt D (T (Suc n))
      using D disjnt_iff by fastforce
      show D ⊆ C
      using interior_of_subset [of X C] D by blast
    qed (use ⟨closedin X D⟩ in auto)
  qed
qed
then obtain Φ where Φ: ∧n. Φ n ⊆ K ∧ closedin X (Φ n) ∧ X interior_of
Φ n ≠ {} ∧ disjnt (Φ n) (T n)
  and ∧n. Φ (Suc n) ⊆ Φ n by metis
then have decseq Φ
  by (simp add: decseq_SucI)
moreover have ∧n. Φ n ≠ {}
  by (metis Φ bot.extremum_uniqueI interior_of_subset)
ultimately have ⋂(range Φ) ≠ {}
  by (metis Φ compact_space_imp_nest ⟨compactin X K⟩ compactin_subspace
closedin_subset_topspace)
moreover have U ⊆ {y. ∃x. y ∈ T x}
  using Asub by auto
with T have {a. ∀n. a ∈ Φ n} ⊆ {}
  by (smt (verit) Asub Φ Collect_empty_eq UN_iff ⟨K ⊆ U⟩ disjnt_iff sub-
set_iff)

```



```

ultimately show False
  by blast
qed
qed

```

lemma *Baire_category_alt*:

```

assumes completely_metrizable_space X  $\vee$  locally_compact_space X  $\wedge$  regu-
lar_space X
and countable  $\mathcal{G}$ 
and  $\bigwedge T. T \in \mathcal{G} \implies \text{closedin } X T \wedge X \text{ interior\_of } T = \{\}$ 
shows  $X \text{ interior\_of } \bigcup \mathcal{G} = \{\}$ 
using Baire_category_aux [of X  $\mathcal{G}$ ] Metric_space.metric_Baire_category_alt
by (metis assms completely_metrizable_space_def)

```

lemma *Baire_category*:

```

assumes completely_metrizable_space X  $\vee$  locally_compact_space X  $\wedge$  regu-
lar_space X
and countable  $\mathcal{G}$ 
and top:  $\bigwedge T. T \in \mathcal{G} \implies \text{openin } X T \wedge X \text{ closure\_of } T = \text{topspace } X$ 
shows  $X \text{ closure\_of } \bigcap \mathcal{G} = \text{topspace } X$ 
proof (cases  $\mathcal{G} = \{\}$ )
case False
have *:  $X \text{ interior\_of } \bigcup ((-)(\text{topspace } X) \text{ ' } \mathcal{G}) = \{\}$ 
proof (intro Baire_category_alt conjI assms)
show countable ((-) (topspace X) '  $\mathcal{G}$ )
using assms by blast
fix T
assume T  $\in$  (-) (topspace X) '  $\mathcal{G}$ 
then obtain U where U:  $U \in \mathcal{G} \wedge T = (\text{topspace } X) - U \wedge U \subseteq (\text{topspace } X)$ 
by (meson top image_iff openin_subset)
then show closedin X T
by (simp add: closedin_diff top)
show  $X \text{ interior\_of } T = \{\}$ 
using U top by (simp add: interior_of_closure_of double_diff)
qed
then show ?thesis
by (simp add: closure_of_eq_topspace interior_of_complement)
qed auto

```

6.11.14 Sierpinski-Hausdorff type results about countable closed unions

lemma *locally_connected_not_countable_closed_union*:

```

assumes topspace X  $\neq$   $\{\}$  and csX: connected_space X
and lcX: locally_connected_space X
and X: completely_metrizable_space X  $\vee$  locally_compact_space X  $\wedge$  Haus-
dorff_space X

```

```

    and countable  $\mathcal{U}$  and pwU: pairwise_disjnt  $\mathcal{U}$ 
    and clo:  $\bigwedge C. C \in \mathcal{U} \implies \text{closedin } X \ C \wedge C \neq \{\}$ 
    and UU_eq:  $\bigcup \mathcal{U} = \text{topspace } X$ 
  shows  $\mathcal{U} = \{\text{topspace } X\}$ 
proof -
  define  $\mathcal{V}$  where  $\mathcal{V} \equiv (\text{frontier\_of}) \ X \ ' \ \mathcal{U}$ 
  define  $B$  where  $B \equiv \bigcup \mathcal{V}$ 
  then have Bsub:  $B \subseteq \text{topspace } X$ 
    by (simp add: Sup_le_iff  $\mathcal{V}$ _def closedin_frontier_of closedin_subset)
  have allsub:  $A \subseteq \text{topspace } X$  if  $A \in \mathcal{U}$  for  $A$ 
    by (meson clo closedin_def that)
  show ?thesis
proof (rule ccontr)
  assume  $\mathcal{U} \neq \{\text{topspace } X\}$ 
  with assms have  $\exists A \in \mathcal{U}. \neg (\text{closedin } X \ A \wedge \text{openin } X \ A)$ 
    by (metis Union_empty connected_space_clopen_in singletonI subsetI subset_singleton_iff)
  then have  $B \neq \{\}$ 
    by (auto simp: B_def  $\mathcal{V}$ _def frontier_of_eq_empty allsub)
  moreover
  have subtopology  $X \ B$  interior_of  $B = B$ 
    by (simp add: Bsub interior_of_openin openin_subtopology_refl)
  ultimately have int_B_nonempty: subtopology  $X \ B$  interior_of  $B \neq \{\}$ 
    by auto
  have subtopology  $X \ B$  interior_of  $\bigcup \mathcal{V} = \{\}$ 
  proof (intro Baire_category_alt conjI)
    have  $\bigcup \mathcal{U} \subseteq B \cup \bigcup ((\text{interior\_of}) \ X \ ' \ \mathcal{U})$ 
      using clo closure_of_closedin by (fastforce simp: B_def  $\mathcal{V}$ _def frontier_of_def)
    moreover have  $B \cup \bigcup ((\text{interior\_of}) \ X \ ' \ \mathcal{U}) \subseteq \bigcup \mathcal{U}$ 
      using allsub clo frontier_of_subset_eq interior_of_subset by (fastforce simp: B_def  $\mathcal{V}$ _def)
    moreover have disjnt  $B \ (\bigcup ((\text{interior\_of}) \ X \ ' \ \mathcal{U}))$ 
      using pwU
    apply (clarsimp simp: B_def  $\mathcal{V}$ _def frontier_of_def pairwise_def disjnt_iff)
    by (metis clo closure_of_eq interior_of_subset subsetD)
  ultimately have  $B = \text{topspace } X - \bigcup ((\text{interior\_of}) \ X \ ' \ \mathcal{U})$ 
    by (auto simp: UU_eq disjnt_iff)
  then have closedin  $X \ B$ 
    by fastforce
  with  $X$  show completely_metrizable_space (subtopology  $X \ B$ )  $\vee$  locally_compact_space
    (subtopology  $X \ B$ )  $\wedge$  regular_space (subtopology  $X \ B$ )
    by (metis completely_metrizable_space_closedin locally_compact_Hausdorff_or_regular

        locally_compact_space_closed_subset regular_space_subtopology)
  show countable  $\mathcal{V}$ 
    by (simp add:  $\mathcal{V}$ _def  $\langle$ countable  $\mathcal{U}\rangle$ )
  fix  $V$ 
  assume  $V \in \mathcal{V}$ 

```

```

then obtain  $S$  where  $S: S \in \mathcal{U} \ V = X \text{ frontier\_of } S$ 
  by (auto simp:  $\mathcal{V}$ _def)
show  $\text{closedin } (\text{subtopology } X \ B) \ V$ 
by (metis  $B$ _def  $\text{Sup\_upper } \mathcal{V}$ _def  $\langle V \in \mathcal{V} \rangle \text{ closedin\_frontier\_of closedin\_subset\_topspace}$ 
image_iff)
have  $\text{subtopology } X \ B \text{ interior\_of } (X \text{ frontier\_of } S) = \{\}$ 
proof (clarsimp simp: interior_of_def openin_subtopology_alt)
  fix  $a \ U$ 
  assume  $a \in B \ a \in U$  and  $\text{ope}U: \text{openin } X \ U$  and  $B\text{Usub}: B \cap U \subseteq X$ 
frontier_of  $S$ 
  then have  $a \in S$ 
  by (meson IntI  $\langle S \in \mathcal{U} \rangle \text{ clo frontier\_of\_subset\_closedin subsetD}$ )
  then obtain  $W \ C$  where  $\text{openin } X \ W \ \text{connectedin } X \ C \ a \in W \ W \subseteq C \ C$ 
 $\subseteq U$ 
  by (metis  $\langle a \in U \rangle \text{ lc}X \text{ locally\_connected\_space } \text{ope}U$ )
have  $W \cap X \text{ frontier\_of } S \neq \{\}$ 
  using  $\langle B \cap U \subseteq X \text{ frontier\_of } S \rangle \langle a \in B \rangle \langle a \in U \rangle \langle a \in W \rangle$  by auto
with frontier_of_openin_straddle_Int
obtain  $W \cap S \neq \{\} \ W - S \neq \{\} \ W \subseteq \text{topspace } X$ 
  using  $\langle \text{openin } X \ W \rangle$  by (metis openin_subset)
then obtain  $b$  where  $b \in \text{topspace } X \ b \in W - S$ 
  by blast
with  $UU\_eq$  obtain  $T$  where  $T \in \mathcal{U} \ T \neq S \ W \cap T \neq \{\}$ 
  by auto
then have  $\text{disjnt } S \ T$ 
  by (metis  $\langle S \in \mathcal{U} \rangle \text{ pairwise\_def } pwU$ )
then have  $C - T \neq \{\}$ 
  by (meson Diff_eq_empty_iff  $\langle W \subseteq C \rangle \langle a \in S \rangle \langle a \in W \rangle \text{ disjnt\_iff}$ 
subsetD)
then have  $C \cap X \text{ frontier\_of } T \neq \{\}$ 
using  $\langle W \cap T \neq \{\} \rangle \langle W \subseteq C \rangle \langle \text{connectedin } X \ C \rangle \text{ connectedin\_Int\_frontier\_of}$ 
by blast
moreover have  $C \cap X \text{ frontier\_of } T = \{\}$ 
proof -
  have  $X \text{ frontier\_of } S \subseteq S \ X \text{ frontier\_of } T \subseteq T$ 
  using frontier_of_subset_closedin  $\langle S \in \mathcal{U} \rangle \langle T \in \mathcal{U} \rangle \text{ clo}$  by blast+
  moreover have  $X \text{ frontier\_of } T \cup B = B$ 
  using  $B$ _def  $\mathcal{V}$ _def  $\langle T \in \mathcal{U} \rangle$  by blast
  ultimately show ?thesis
  using  $B\text{Usub} \langle C \subseteq U \rangle \langle \text{disjnt } S \ T \rangle \text{ unfolding } \text{disjnt\_def}$  by blast
qed
ultimately show False
  by simp
qed
with  $S$  show  $\text{subtopology } X \ B \text{ interior\_of } V = \{\}$ 
  by meson
qed
then show False
  using  $B$ _def int_B_nonempty by blast

```

qed
qed

lemma *real_Sierpinski_lemma*:

fixes $a b :: \text{real}$
 assumes $a \leq b$
 and *countable* \mathcal{U} and *pwU*: *pairwise disjoint* \mathcal{U}
 and *clo*: $\bigwedge C. C \in \mathcal{U} \implies \text{closed } C \wedge C \neq \{\}$
 and $\bigcup \mathcal{U} = \{a..b\}$
 shows $\mathcal{U} = \{\{a..b\}\}$
 proof –
 have *locally_connected_space* (*top_of_set* $\{a..b\}$)
 by (*simp add*: *locally_connected_real_interval*)
 moreover
 have *completely metrizable_space* (*top_of_set* $\{a..b\}$)
 by (*metis* *box_real(2)* *completely_metrizable_space_cbox*)
 ultimately
 show ?thesis
 using *locally_connected_not_countable_closed_union* [*of subtopology euclidean*
 $\{a..b\}$] *assms*
 apply (*simp add*: *closedin_subtopology*)
 by (*metis* *Union_upper_inf.orderE*)
 qed

6.11.15 The Tychonoff embedding

lemma *completely_regular_space_cube_embedding_explicit*:

assumes *completely_regular_space* X *Hausdorff_space* X
 shows *embedding_map* X
 (*product_topology* ($\lambda f. \text{top_of_set } \{0..1 :: \text{real}\}$)
 (*mpace* (*submetric* (*cfunspace* X *euclidean_metric*)
 $\{f. f \in \text{topspace } X \rightarrow \{0..1\}\}$))))
 ($\lambda x. \lambda f \in \text{mpace} (\text{submetric} (\text{cfunspace } X \text{ euclidean_metric}) \{f. f \in$
 $\text{topspace } X \rightarrow \{0..1\}\})$
 $f x$)

proof –

define K where $K \equiv \text{mpace}(\text{submetric} (\text{cfunspace } X \text{ euclidean_metric}) \{f. f$
 $\in \text{topspace } X \rightarrow \{0..1 :: \text{real}\}\})$

define e where $e \equiv \lambda x. \lambda f \in K. f x$

have $e x \neq e y$ if $xy: x \neq y \ x \in \text{topspace } X \ y \in \text{topspace } X$ for $x y$

proof –

have *closedin* $X \ \{x\}$

by (*simp add*: $\langle \text{Hausdorff_space } X \rangle$ *closedin_Hausdorff_singleton* $\langle x \in$
 $\text{topspace } X \rangle$)

then obtain f where *contf*: *continuous_map* X *euclideanreal* f

and $f01: \bigwedge x. x \in \text{topspace } X \implies f x \in \{0..1\}$ and $fx01: f y = 0 \ f x = 1$

using $\langle \text{completely_regular_space } X \rangle$ *xy* **unfolding** *completely_regular_space_def*

by (*smt* (*verit*, *ccfv_threshold*) *Diff_iff_continuous_map_in_subtopology*
image_subset_iff_singleton_iff)

```

then have bounded (f ' topspace X)
  by (meson bounded_closed_interval bounded_subset_image_subset_iff)
with contf f01 have restrict f (topspace X) ∈ K
  by (auto simp: K_def)
with fxy xy show ?thesis
  unfolding e_def by (metis restrict_apply' zero_neq_one)
qed
then have inj_on e (topspace X)
  by (meson inj_onI)
then obtain e' where e':  $\bigwedge x. x \in \text{topspace } X \implies e' (e x) = x$ 
  by (metis inv_into_f_f)
have continuous_map (subtopology (product_topology ( $\lambda f. \text{top\_of\_set } \{0..1\}$ )
K) (e ' topspace X)) X e'
proof (clarsimp simp add: continuous_map_atin limitin_atin openin_subtopology_alt
e')
  fix x U
  assume e x ∈ K  $\rightarrow_E \{0..1\}$  and x ∈ topspace X and openin X U and x ∈ U
  then obtain g where contg: continuous_map X (top_of_set {0..1}) g and
g x = 0
    and gim: g ' (topspace X - U)  $\subseteq \{1::\text{real}\}$ 
  using <completely_regular_space X> unfolding completely_regular_space_def

  by (metis Diff_iff openin_closedin_eq)
then have bounded (g ' topspace X)
  by (meson bounded_closed_interval bounded_subset_continuous_map_in_subtopology)
moreover have g ∈ topspace X  $\rightarrow \{0..1\}$ 
  using contg by (simp add: continuous_map_def)
ultimately have g_in_K: restrict g (topspace X) ∈ K
  using contg by (force simp add: K_def continuous_map_in_subtopology)
have openin (top_of_set {0..1}) {0..<1::real}
  using open_real_greaterThanLessThan[of -1 1] by (force simp: openin_open)
moreover have e x ∈ ( $\Pi_E f \in K. \text{if } f = \text{restrict } g \text{ (topspace X) then } \{0..<1\}$ 
else {0..1})
  using <e x ∈ K  $\rightarrow_E \{0..1\}$ > by (simp add: e_def <g x = 0> <x ∈ topspace
X> PiE_iff)
moreover have e y = e x
  if y  $\notin U$  and ey: e y ∈ ( $\Pi_E f \in K. \text{if } f = \text{restrict } g \text{ (topspace X) then } \{0..<1\}$ 
else {0..1})
  and y: y ∈ topspace X for y
proof -
  have e y (restrict g (topspace X)) ∈ {0..<1}
    using ey by (smt (verit, ccfv_SIG) PiE_mem g_in_K)
  with gim g_in_K y <y  $\notin U$ > show ?thesis
    by (fastforce simp: e_def)
qed
ultimately
show  $\exists W. \text{openin (product\_topology } (\lambda f. \text{top\_of\_set } \{0..1\}) K) W \wedge e x \in$ 
W  $\wedge e' ' (e ' \text{topspace } X \cap W - \{e x\}) \subseteq U$ 
  apply (rule_tac x=PiE K ( $\lambda f. \text{if } f = \text{restrict } g \text{ (topspace X) then } \{0..<1\}$ 

```

```

else {0..1} in exI)
  by (auto simp: openin_PiE_gen e')
qed
with e' have embedding_map X (product_topology ( $\lambda f$ . top_of_set {0..1}) K)
e
  unfolding embedding_map_def homeomorphic_map_maps homeomorphic_maps_def
  by (fastforce simp: e_def K_def continuous_map_in_subtopology continuous_map_componentwise)
  then show ?thesis
  by (simp add: K_def e_def)
qed

```

```

lemma completely_regular_space_cube_embedding:
  fixes X :: 'a topology
  assumes completely_regular_space X Hausdorff_space X
  obtains K :: ('a  $\Rightarrow$  real) set and e
  where embedding_map X (product_topology ( $\lambda f$ . top_of_set {0..1::real}) K)
e
  using completely_regular_space_cube_embedding_explicit [OF assms] by metis

```

6.11.16 Urysohn and Tietze analogs for completely regular spaces

"Urysohn and Tietze analogs for completely regular spaces if (\cdot) set is assumed compact instead of closed. Note that Hausdorffness is *not* required: inside (\cdot) proof we factor through the Kolmogorov quotient." – John Harrison

```

lemma Urysohn_completely_regular_closed_compact:
  fixes a b :: real
  assumes a  $\leq$  b completely_regular_space X closedin X S compactin X T disjnt S T
  obtains f where continuous_map X (subtopology euclidean {a..b}) f f ' T  $\subseteq$ 
{a} f ' S  $\subseteq$  {b}
proof -
  obtain f where contf: continuous_map X (subtopology euclideanreal {0..1}) f
  and f0: f ' T  $\subseteq$  {0} and f1: f ' S  $\subseteq$  {1}
  proof (cases T={})
  case True
  show thesis
  proof
  show continuous_map X (top_of_set {0..1}) ( $\lambda x$ . 1::real) ( $\lambda x$ . 1::real) ' T
 $\subseteq$  {0} ( $\lambda x$ . 1::real) ' S  $\subseteq$  {1}
  using True by auto
  qed
  next
  case False
  have  $\bigwedge t$ . t  $\in$  T  $\implies$   $\exists f$ . continuous_map X (subtopology euclideanreal ({0..1}))
f  $\wedge$  f t = 0  $\wedge$  f ' S  $\subseteq$  {1}

```

```

using assms unfolding completely_regular_space_def
by (meson DiffI compactin_subset_topspace disjnt_iff subset_eq)
then obtain g where contg:  $\bigwedge t. t \in T \implies \text{continuous\_map } X \text{ (subtopology euclideanreal } \{0..1\}) (g \ t)$ 
and g0:  $\bigwedge t. t \in T \implies g \ t \ t = 0$ 
and g1:  $\bigwedge t. t \in T \implies g \ t \ ' S \subseteq \{1\}$ 
by metis
then have g01:  $\bigwedge t. t \in T \implies g \ t \ ' \text{topspace } X \subseteq \{0..1\}$ 
by (meson continuous_map_in_subtopology)
define G where G  $\equiv \lambda t. \{x \in \text{topspace } X. g \ t \ x \in \{..<1/2\}\}$ 
have Ball (G'T) (openin X)
using contg unfolding G_def continuous_map_in_subtopology
by (smt (verit, best) Collect_cong openin_continuous_map_preimage image_iff open_lessThan open_openin)
moreover have  $T \subseteq \bigcup (G' T)$ 
using  $\langle \text{compactin } X \ T \rangle$  g0 compactin_subset_topspace by (force simp: G_def)
ultimately have  $\exists \mathcal{F}. \text{finite } \mathcal{F} \wedge \mathcal{F} \subseteq G' T \wedge T \subseteq \bigcup \mathcal{F}$ 
using  $\langle \text{compactin } X \ T \rangle$  unfolding compactin_def by blast
then obtain K where K: finite K  $K \subseteq T \ T \subseteq \bigcup (G' K)$ 
by (metis finite_subset_image)
with False have  $K \neq \{\}$ 
by fastforce
define f where  $f \equiv \lambda x. 2 * \max 0 (\text{Inf } ((\lambda t. g \ t \ x) \ ' K) - 1/2)$ 
have [simp]:  $\max 0 (x - 1/2) = 0 \iff x \leq 1/2$  for  $x::\text{real}$ 
by force
have [simp]:  $2 * \max 0 (x - 1/2) = 1 \iff x = 1$  for  $x::\text{real}$ 
by (simp add: max_def_raw)
show thesis
proof
have  $g \ t \ s = 1$  if  $s \in S \ t \in K$  for  $s \ t$ 
using  $\langle K \subseteq T \rangle$  g1 that by auto
then show  $f \ ' S \subseteq \{1\}$ 
using  $\langle K \neq \{\} \rangle$  by (simp add: f_def image_subset_iff)
have (INF  $t \in K. g \ t \ x$ )  $\leq 1/2$  if  $x \in T$  for  $x$ 
proof -
obtain k where  $k \in K \ g \ k \ x < 1/2$ 
using  $K \ \langle x \in T \rangle$  by (auto simp: G_def)
then show ?thesis
by (meson  $\langle \text{finite } K \rangle$  cInf_le_finite dual_order.trans finite_imageI imageI less_le_not_le)
qed
then show  $f \ ' T \subseteq \{0\}$ 
by (force simp: f_def)
have  $\bigwedge t. t \in K \implies \text{continuous\_map } X \ \text{euclideanreal } (g \ t)$ 
using  $\langle K \subseteq T \rangle$  contg continuous_map_in_subtopology by blast
moreover have  $2 * \max 0 ((\text{INF } t \in K. g \ t \ x) - 1/2) \leq 1$  if  $x \in \text{topspace } X$  for  $x$ 
proof -
obtain k where  $k \in K \ g \ k \ x \leq 1$ 

```

```

    using K ⟨x ∈ topspace X⟩ ⟨K ≠ {}⟩ g01 by (fastforce simp: G_def)
  then have (INF t∈K. g t x) ≤ 1
  by (meson ⟨finite K⟩ cInf_le_finite dual_order.trans finite_imageI imageI)
  then show ?thesis
    by (simp add: max_def_raw)
  qed
  ultimately show continuous_map X (top_of_set {0..1}) f
    by (force simp: f_def continuous_map_in_subtopology intro!: ⟨finite K⟩
    continuous_intros)
  qed
  qed
  define g where g ≡ λx. a + (b - a) * f x
  show thesis
  proof
    have ∀ x∈topspace X. a + (b - a) * f x ≤ b
      using contf ⟨a ≤ b⟩ apply (simp add: continuous_map_in_subtopology
    image_subset_iff)
      by (smt (verit, best) mult_right_le_one_le)
    then show continuous_map X (top_of_set {a..b}) g
      using contf ⟨a ≤ b⟩ unfolding g_def continuous_map_in_subtopology im-
    age_subset_iff
      by (intro conjI continuous_intros; simp)
    show g ‘ T ⊆ {a} g ‘ S ⊆ {b}
      using f0 f1 by (auto simp: g_def)
  qed
  qed

```

lemma *Urysohn_completely_regular_compact_closed:*

```

  fixes a b::real
  assumes a ≤ b completely_regular_space X compactin X S closedin X T disjnt
  S T
  obtains f where continuous_map X (subtopology euclidean {a..b}) f f ‘ T ⊆
  {a} f ‘ S ⊆ {b}
  proof -
    obtain f where contf: continuous_map X (subtopology euclidean {-b..-a}) f
  and fim: f ‘ T ⊆ {-a} f ‘ S ⊆ {-b}
      by (meson Urysohn_completely_regular_compact assms disjnt_sym
    neg_le_iff_le)
    show thesis
    proof
      show continuous_map X (top_of_set {a..b}) (uminus o f)
        using contf by (auto simp: continuous_map_in_subtopology o_def)
      show (uminus o f) ‘ T ⊆ {a} (uminus o f) ‘ S ⊆ {b}
        using fim by fastforce+
    qed
  qed

```

lemma *Urysohn_completely_regular_compact_closed_alt:*


```

fixes  $a\ b::\text{real}$ 
assumes completely_regular_space  $X$  compactin  $X\ S$  closedin  $X\ T$  disjnt  $S\ T$ 
obtains  $f$  where continuous_map  $X$  euclideanreal  $f\ f' : T \subseteq \{a\}\ f' : S \subseteq \{b\}$ 
proof (cases a b rule: le_cases)
  case  $le$ 
    then show ?thesis
    by (meson Urysohn_completely_regular_compact_closed assms continuous_map_into_fulltopology
that)
  next
    case  $ge$ 
    then show ?thesis
    using Urysohn_completely_regular_closed_compact assms
    by (metis Urysohn_completely_regular_closed_compact assms continuous_map_into_fulltopology
disjnt_sym that)
qed

```

lemma *Tietze_extension_comp_reg_aux*:

```

fixes  $T :: \text{real set}$ 
assumes completely_regular_space  $X$  Hausdorff_space  $X$  compactin  $X\ S$ 
  and  $T$ : is_interval  $T\ T \neq \{\}$ 
  and contf: continuous_map (subtopology  $X\ S$ ) euclidean  $f$  and fm:  $f : S \subseteq T$ 
obtains  $g$  where continuous_map  $X$  euclidean  $g\ g' : \text{topspace } X \subseteq T \wedge x. x \in S$ 
 $\implies g\ x = f\ x$ 
proof –
  obtain  $K :: ('a \Rightarrow \text{real})\ \text{set}$  and  $e$ 
    where  $e0$ : embedding_map  $X$  (product_topology ( $\lambda f. \text{top\_of\_set } \{0..1::\text{real}\}$ )
 $K$ )  $e$ 
    using assms completely_regular_space_cube_embedding by blast
  define cube where  $\text{cube} \equiv \text{product\_topology } (\lambda f. \text{top\_of\_set } \{0..1::\text{real}\})\ K$ 
  have  $e$ : embedding_map  $X$  cube  $e$ 
    using  $e0$  by (simp add: cube_def)
  obtain  $e'$  where  $e'$ : homeomorphic_maps  $X$  (subtopology  $\text{cube } (e' : \text{topspace } X)$ )
 $e\ e'$ 
    using  $e$  by (force simp: cube_def embedding_map_def homeomorphic_map_maps)
  then have conte: continuous_map  $X$  (subtopology  $\text{cube } (e' : \text{topspace } X)$ )  $e$ 
    and conte': continuous_map (subtopology  $\text{cube } (e' : \text{topspace } X)$ )  $X\ e'$ 
    and  $e'e$ :  $\forall x \in \text{topspace } X. e' (e\ x) = x$ 
    by (auto simp: homeomorphic_maps_def)
  have Hausdorff_space  $\text{cube}$ 
    unfolding cube_def
    using Hausdorff_space_euclidean Hausdorff_space_product_topology Haus-
dorff_space_subtopology by blast
  have normal_space  $\text{cube}$ 
  proof (rule compact_Hausdorff_or_regular_imp_normal_space)
    show compact_space  $\text{cube}$ 
    unfolding cube_def
    using compact_space_product_topology compact_space_subtopology com-
pactin_euclidean_iff by blast

```

```

qed (use ⟨Hausdorff_space_cube⟩ in auto)
moreover
have comp: compactin_cube (e ' S)
  by (meson ⟨compactin_X_S⟩ conte continuous_map_in_subtopology_image_compactin)
then have closedin_cube (e ' S)
  by (intro compactin_imp_closedin ⟨Hausdorff_space_cube⟩)
moreover
have continuous_map (subtopology_cube (e ' S)) euclideanreal (f ∘ e')
proof (intro continuous_map_compose)
  show continuous_map (subtopology_cube (e ' S)) (subtopology_X_S) e'
    unfolding continuous_map_in_subtopology
  proof
    show continuous_map (subtopology_cube (e ' S)) X e'
      by (meson ⟨compactin_X_S⟩ compactin_subset_topspace conte' continuous_map_from_subtopology_mono image_mono)
    show e' ' topspace (subtopology_cube (e ' S)) ⊆ S
      using ⟨compactin_X_S⟩ compactin_subset_topspace e'e by fastforce
  qed
qed (simp add: contf)
moreover
have (f ∘ e') ' e ' S ⊆ T
  using ⟨compactin_X_S⟩ compactin_subset_topspace e'e fim by fastforce
ultimately
obtain g where contg: continuous_map_cube euclidean g and gsub: g ' topspace_cube ⊆ T
  and gf:  $\bigwedge x. x \in e'S \implies g x = (f \circ e') x$ 
  using Tietze_extension_realinterval T by metis
show thesis
proof
  show continuous_map X euclideanreal (g ∘ e)
  by (meson contg conte continuous_map_compose continuous_map_in_subtopology)
  show (g ∘ e) ' topspace_X ⊆ T
    using gsub conte continuous_map_image_subset_topspace by fastforce
  fix x
  assume x ∈ S
  then show (g ∘ e) x = f x
    using gf ⟨compactin_X_S⟩ compactin_subset_topspace e'e by fastforce
qed
qed

```

lemma Tietze_extension_completely_regular:

```

assumes completely_regular_space X compactin_X_S is_interval T T ≠ {}
  and contf: continuous_map (subtopology_X_S) euclidean f and fim: f'S ⊆ T
obtains g where continuous_map_X euclideanreal g g ' topspace_X ⊆ T
   $\bigwedge x. x \in S \implies g x = f x$ 

```

proof –

```

define Q where Q ≡ Kolmogorov_quotient X ' (topspace_X)
obtain g where contg: continuous_map (subtopology_X (Kolmogorov_quotient

```

```

X ' S)) euclidean g
  and gf:  $\bigwedge x. x \in S \implies g(\text{Kolmogorov\_quotient } X \ x) = f \ x$ 
  using Kolmogorov\_quotient\_lift\_exists
  by (metis <compactin X S> contf compactin_subset_topspace open_openin
t0\_space\_def t1\_space)
  have S  $\subseteq$  topspace X
  by (simp add: <compactin X S> compactin_subset_topspace)
  then have [simp]:  $Q \cap \text{Kolmogorov\_quotient } X \ ' \ S = \text{Kolmogorov\_quotient } X \ ' \ S$ 
  S
  using Q\_def by blast
  have creg: completely\_regular\_space (subtopology X Q)
  by (simp add: <completely\_regular\_space X> completely\_regular\_space\_subtopology)
  then have regular\_space (subtopology X Q)
  by (simp add: completely\_regular\_imp\_regular\_space)
  then have Hausdorff\_space (subtopology X Q)
  using Q\_def regular\_t0\_eq\_Hausdorff\_space t0\_space\_Kolmogorov\_quotient
  by blast
  moreover
  have compactin (subtopology X Q) (Kolmogorov\_quotient X ' S)
  by (metis Q\_def <compactin X S> image\_compactin quotient\_imp\_continuous\_map
quotient\_map\_Kolmogorov\_quotient)
  ultimately obtain h where conth: continuous\_map (subtopology X Q) euclidean
  h
      and him:  $h \ ' \ \text{topspace (subtopology X Q)} \subseteq T$ 
      and hg:  $\bigwedge x. x \in \text{Kolmogorov\_quotient } X \ ' \ S \implies h \ x = g \ x$ 
  using Tietze\_extension\_comp\_reg\_aux [of subtopology X Q Kolmogorov\_quotient
X ' S T g]
  apply (simp add: subtopology\_subtopology creg contg assms)
  using fim gf by blast
  show thesis
  proof
  show continuous\_map X euclideanreal (h  $\circ$  Kolmogorov\_quotient X)
  by (metis Q\_def conth continuous\_map\_compose quotient\_imp\_continuous\_map
quotient\_map\_Kolmogorov\_quotient)
  show (h  $\circ$  Kolmogorov\_quotient X) ' topspace X  $\subseteq$  T
  using Q\_def continuous\_map\_Kolmogorov\_quotient continuous\_map\_image\_subset\_topspace
  him by fastforce
  fix x
  assume  $x \in S$  then show (h  $\circ$  Kolmogorov\_quotient X) x = f x
  by (simp add: gf hg)
  qed
  qed

```

6.11.17 Size bounds on connected or path-connected spaces

lemma *connected_space_imp_card_ge_alt:*

assumes *connected_space X completely_regular_space X closedin X S S \neq {} S \neq topspace X*
shows *(UNIV::real set) \lesssim topspace X*

```

proof –
  have  $S \subseteq \text{topspace } X$ 
    using  $\langle \text{closedin } X \ S \rangle$  closedin_subset by blast
  then obtain  $a$  where  $a \in \text{topspace } X$   $a \notin S$ 
    using  $\langle S \neq \text{topspace } X \rangle$  by blast
  have  $(UNIV::\text{real set}) \lesssim \{0..1::\text{real}\}$ 
    using eqpoll_real_subset
    by (meson atLeastAtMost_iff eqpoll_imp_lepoll eqpoll_sym less_eq_real_def
zero_less_one)
  also have  $\dots \lesssim \text{topspace } X$ 
  proof –
    obtain  $f$  where contf: continuous_map X euclidean f
      and fm: f ∈ (topspace X) → {0..1::real}
      and f0: f a = 0 and f1: f ' S ⊆ {1}
      using  $\langle \text{completely\_regular\_space } X \rangle$ 
      unfolding completely_regular_space_def
      by (metis Diff_iff ‹a ∈ topspace X› ‹a ∉ S› ‹closedin X S› continu-
ous_map_in_subtopology image_subset_iff_funcset)
    have  $\exists y \in \text{topspace } X. x = f y$  if  $0 \leq x$  and  $x \leq 1$  for  $x$ 
    proof –
      have connectedin euclidean (f ' topspace X)
        using  $\langle \text{connected\_space } X \rangle$  connectedin_continuous_map_image connecte-
din_topspace contf by blast
      moreover have  $\exists y. 0 = f y \wedge y \in \text{topspace } X$ 
        using  $\langle a \in \text{topspace } X \rangle$  f0 by auto
      moreover have  $\exists y. 1 = f y \wedge y \in \text{topspace } X$ 
        using  $\langle S \subseteq \text{topspace } X \rangle$   $\langle S \neq \{\} \rangle$  f1 by fastforce
      ultimately show ?thesis
      using that by (fastforce simp: is_interval_1 simp flip: is_interval_connected_1)
    qed
    then show ?thesis
      unfolding lepoll_iff using atLeastAtMost_iff by blast
    qed
  finally show ?thesis .
qed

```

```

lemma connected_space_imp_card_ge_gen:
  assumes connected_space X normal_space X closedin X S closedin X T S ≠ {}
T ≠ {} disjnt S T
  shows  $(UNIV::\text{real set}) \lesssim \text{topspace } X$ 
proof –
  have  $(UNIV::\text{real set}) \lesssim \{0..1::\text{real}\}$ 
    by (metis atLeastAtMost_iff eqpoll_real_subset eqpoll_imp_lepoll eqpoll_sym
less_le_not_le zero_less_one)
  also have  $\dots \lesssim \text{topspace } X$ 
proof –
  obtain  $f$  where contf: continuous_map X euclidean f
    and fm: f ∈ (topspace X) → {0..1::real}

```

```

    and f0: f ' S ⊆ {0} and f1: f ' T ⊆ {1}
  using assms by (metis continuous_map_in_subtopology normal_space_iff_Urysohn
image_subset_iff_funcset)
  have ∃ y ∈ topspace X. x = f y if 0 ≤ x and x ≤ 1 for x
  proof -
    have connectedin euclidean (f ' topspace X)
      using ⟨connected_space X⟩ connectedin_continuous_map_image connecte-
din_topspace contf by blast
    moreover have ∃ y. 0 = f y ∧ y ∈ topspace X
      using ⟨closedin X S⟩ ⟨S ≠ {}⟩ closedin_subset f0 by fastforce
    moreover have ∃ y. 1 = f y ∧ y ∈ topspace X
      using ⟨closedin X T⟩ ⟨T ≠ {}⟩ closedin_subset f1 by fastforce
    ultimately show ?thesis
      using that by (fastforce simp: is_interval_1 simp flip: is_interval_connected_1)
  qed
  then show ?thesis
    unfolding lepoll_iff using atLeastAtMost_iff by blast
  qed
  finally show ?thesis .
  qed

```

lemma *connected_space_imp_card_ge*:

```

  assumes connected_space X normal_space X t1_space X and nosing: ¬ (∃ a.
topspace X ⊆ {a})
  shows (UNIV::real set) ≲ topspace X
  proof -
    obtain a b where a ∈ topspace X b ∈ topspace X a ≠ b
      by (metis nosing singletonI subset_iff)
    then have {a} ≠ topspace X
      by force
    with connected_space_imp_card_ge_alt assms show ?thesis
      by (metis ⟨a ∈ topspace X⟩ closedin_t1_singleton insert_not_empty nor-
mal_imp_completely_regular_space_A)
  qed

```

lemma *connected_space_imp_infinite_gen*:

```

  [[connected_space X; t1_space X; ∄ a. topspace X ⊆ {a}]] ⇒ infinite(topspace
X)
  by (metis connected_space_discrete_topology finite_t1_space_imp_discrete_topology)

```

lemma *connected_space_imp_infinite*:

```

  [[connected_space X; Hausdorff_space X; ∄ a. topspace X ⊆ {a}]] ⇒ infi-
nite(topspace X)
  by (simp add: Hausdorff_imp_t1_space connected_space_imp_infinite_gen)

```

lemma *connected_space_imp_infinite_alt*:

```

  assumes connected_space X regular_space X closedin X S S ≠ {} S ≠ topspace
X
  shows infinite(topspace X)

```

proof –

have $S \subseteq \text{topspace } X$
using $\langle \text{closedin } X \ S \rangle$ *closedin_subset* **by** *blast*
then obtain a **where** $a: a \in \text{topspace } X \ a \notin S$
using $\langle S \neq \text{topspace } X \rangle$ **by** *blast*
have $\exists \Phi. \forall n. (\text{disjnt } (\Phi \ n) \ S \wedge a \in \Phi \ n \wedge \text{openin } X \ (\Phi \ n)) \wedge \Phi(\text{Suc } n) \subset \Phi \ n$
proof (*rule dependent_nat_choice*)
show $\exists T. \text{disjnt } T \ S \wedge a \in T \wedge \text{openin } X \ T$
by (*metis Diff_iff a* $\langle \text{closedin } X \ S \rangle$ *closedin_def disjnt_iff*)
fix $V \ n$
assume $\S: \text{disjnt } V \ S \wedge a \in V \wedge \text{openin } X \ V$
then obtain $U \ C$ **where** $U: \text{openin } X \ U \ \text{closedin } X \ C \ a \in U \ U \subseteq C \ C \subseteq V$
using $\langle \text{regular_space } X \rangle$ **by** (*metis neighbourhood_base_of_neighbourhood_base_of_closedin*)
with *assms* **have** $U \subset V$
by (*metis* $\langle S \subseteq \text{topspace } X \rangle$ *connected_space_clopen_in disjnt_def empty_iff*
inf.absorb_iff2 inf.orderE psubsetI subset_trans)
with U **show** $\exists U. (\text{disjnt } U \ S \wedge a \in U \wedge \text{openin } X \ U) \wedge U \subset V$
using \S *disjnt_subset1* **by** *blast*
qed
then obtain Φ **where** $\Phi: \bigwedge n. \text{disjnt } (\Phi \ n) \ S \wedge a \in \Phi \ n \wedge \text{openin } X \ (\Phi \ n)$
and $\Phi \text{sub}: \bigwedge n. \Phi(\text{Suc } n) \subset \Phi \ n$ **by** *metis*
then have *decseq* Φ
by (*simp add: decseq_SucI psubset_eq*)
have $\forall n. \exists x. x \in \Phi \ n \wedge x \notin \Phi(\text{Suc } n)$
by (*meson* Φsub *psubsetE subsetI*)
then obtain f **where** $\text{fin}: \bigwedge n. f \ n \in \Phi \ n$ **and** $\text{fout}: \bigwedge n. f \ n \notin \Phi(\text{Suc } n)$
by *metis*
have $\text{range } f \subseteq \text{topspace } X$
by (*meson* Φ *fin image_subset_iff openin_subset subset_iff*)
moreover have *inj* f
by (*metis* *Suc_le_eq* $\langle \text{decseq } \Phi \rangle$ *decseq_def fin fout linorder_injI subsetD*)
ultimately show *?thesis*
using *infinite_iff_countable_subset* **by** *blast*
qed

lemma *path_connected_space_imp_card_ge*:

assumes *path_connected_space* X *Hausdorff_space* X **and** *nosing*: $\neg (\exists x. \text{topspace } X \subseteq \{x\})$

shows $(\text{UNIV}::\text{real set}) \lesssim \text{topspace } X$

proof –

obtain $a \ b$ **where** $a \in \text{topspace } X \ b \in \text{topspace } X \ a \neq b$

by (*metis* *nosing singletonI subset_iff*)

then obtain γ **where** $\gamma: \text{pathin } X \ \gamma \ \gamma \ 0 = a \ \gamma \ 1 = b$

by (*meson* $\langle a \in \text{topspace } X \rangle \langle b \in \text{topspace } X \rangle \langle \text{path_connected_space } X \rangle$
path_connected_space_def)

let $?Y = \text{subtopology } X \ (\gamma \ ` (\text{topspace } (\text{subtopology euclidean } \{0..1\})))$

have $(\text{UNIV}::\text{real set}) \lesssim \text{topspace } ?Y$

proof (*intro compact_Hausdorff_or_regular_imp_normal_space connected_space_imp_card_ge*)
show *connected_space* $?Y$

```

    using ‹pathin X  $\gamma$ › connectedin_def connectedin_path_image by auto
  show Hausdorff_space ?Y  $\vee$  regular_space ?Y
    using Hausdorff_space_subtopology ‹Hausdorff_space X› by blast
  show t1_space ?Y
    using Hausdorff_imp_t1_space ‹Hausdorff_space X› t1_space_subtopology
  by blast
  show compact_space ?Y
  by (simp add: ‹pathin X  $\gamma$ › compact_space_subtopology compactin_path_image)
  have a  $\in$  topspace ?Y b  $\in$  topspace ?Y
    using  $\gamma$  pathin_subtopology by fastforce+
  with ‹a  $\neq$  b› show  $\nexists x. \text{topspace } ?Y \subseteq \{x\}$ 
    by blast
  qed
  also have ...  $\lesssim \gamma \text{ ' } \{0..1\}$ 
    by (simp add: subset_imp_lepoll)
  also have ...  $\lesssim$  topspace X
    by (meson  $\gamma$  path_image_subset_topspace subset_imp_lepoll image_subset_iff_funcset)
  finally show ?thesis .
  qed

```

lemma *connected_space_imp_uncountable*:

```

  assumes connected_space X regular_space X Hausdorff_space X  $\neg$  ( $\exists a. \text{topspace } X \subseteq \{a\}$ )
  shows  $\neg$  countable(topspace X)

```

proof

```

  assume coX: countable (topspace X)
  with ‹regular_space X› have normal_space X
    using countable_imp_Lindelof_space regular_Lindelof_imp_normal_space by
  blast
  then have (UNIV::real set)  $\lesssim$  topspace X
    by (simp add: Hausdorff_imp_t1_space assms connected_space_imp_card_ge)
  with coX show False
    using countable_lepoll_uncountable_UNIV_real by blast
  qed

```

lemma *path_connected_space_imp_uncountable*:

```

  assumes path_connected_space X t1_space X and nosing:  $\neg$  ( $\exists a. \text{topspace } X \subseteq \{a\}$ )
  shows  $\neg$  countable(topspace X)

```

proof

```

  assume coX: countable (topspace X)
  obtain a b where a  $\in$  topspace X b  $\in$  topspace X a  $\neq$  b
    by (metis nosing singletonI subset_iff)
  then obtain  $\gamma$  where pathin X  $\gamma$   $\gamma$  0 = a  $\gamma$  1 = b
    by (meson ‹a  $\in$  topspace X› ‹b  $\in$  topspace X› ‹path_connected_space X›
  path_connected_space_def)
  then have  $\gamma \text{ ' } \{0..1\} \lesssim$  topspace X
    by (meson path_image_subset_topspace subset_imp_lepoll image_subset_iff_funcset)
  define  $\mathcal{A}$  where  $\mathcal{A} \equiv ((\lambda a. \{x \in \{0..1\}. \gamma x \in \{a\}\}) \text{ ' } \text{topspace } X) - \{\{\}\}$ 

```

```

have A01:  $\mathcal{A} = \{\{0..1\}\}$ 
proof (rule real_Sierpinski_lemma)
  show countable  $\mathcal{A}$ 
  using  $\mathcal{A\_def}$  coX by blast
  show disjoint  $\mathcal{A}$ 
  by (auto simp:  $\mathcal{A\_def}$  disjnt_iff pairwise_def)
  show  $\bigcup \mathcal{A} = \{0..1\}$ 
  using  $\langle \text{pathin } X \ \gamma \rangle$  path_image_subset_topspace by (fastforce simp:  $\mathcal{A\_def}$ 
  Bex_def)
  fix C
  assume  $C \in \mathcal{A}$ 
  then obtain a where  $a \in \text{topspace } X$  and  $C: C = \{x \in \{0..1\}. \gamma \ x \in \{a\}\}$ 
  C  $\neq \{\}$ 
  by (auto simp:  $\mathcal{A\_def}$ )
  then have closedin X {a}
  by (meson  $\langle \text{t1\_space } X \rangle$  closedin_t1_singleton)
  then have closedin (top_of_set {0..1}) C
  using C  $\langle \text{pathin } X \ \gamma \rangle$  closedin_continuous_map_preimage pathin_def by
  fastforce
  then show closed C  $\wedge$  C  $\neq \{\}$ 
  using C closedin_closed_trans by blast
qed auto
then have  $\{0..1\} \in \mathcal{A}$ 
  by blast
then have  $\exists a \in \text{topspace } X. \{0..1\} \subseteq \{x. \gamma \ x = a\}$ 
  using  $\mathcal{A\_def}$  image_iff by auto
then show False
  using  $\langle \gamma \ 0 = a \rangle \langle \gamma \ 1 = b \rangle \langle a \neq b \rangle$  atLeastAtMost_iff zero_less_one_class.zero_le_one
  by blast
qed

```

6.11.18 Lavrentiev extension etc

```

lemma (in Metric_space) convergent_eq_zero_oscillation_gen:
  assumes mcomplete and fim:  $f \in (\text{topspace } X \cap S) \rightarrow M$ 
  shows  $(\exists l. \text{limitin } m\text{topology } f \ l \ (\text{atin\_within } X \ a \ S)) \longleftrightarrow$ 
     $M \neq \{\} \wedge$ 
     $(a \in \text{topspace } X$ 
       $\longrightarrow (\forall \varepsilon > 0. \exists U. \text{openin } X \ U \wedge a \in U \wedge$ 
         $(\forall x \in (S \cap U) - \{a\}. \forall y \in (S \cap U) - \{a\}. d(f \ x) (f \ y) <$ 
           $\varepsilon)))$ 
proof (cases  $M = \{\}$ )
  case True
  with limitin_mspace show ?thesis
  by blast
next
  case False
  show ?thesis
  proof (cases  $a \in \text{topspace } X$ )

```



```

case True
let ?R =  $\forall \varepsilon > 0. \exists U. \text{openin } X \ U \wedge a \in U \wedge (\forall x \in S \cap U - \{a\}. \forall y \in S \cap U - \{a\}. d(f \ x) \ (f \ y) < \varepsilon)$ 
show ?thesis
proof (cases a  $\in$  X derived_set_of S)
  case True
  have ?R
    if limitin mtopology f l (atin_within X a S) for l
  proof (intro strip)
    fix  $\varepsilon :: \text{real}$ 
    assume  $\varepsilon > 0$ 
    with that  $\langle a \in \text{topspace } X \rangle$ 
    obtain U where U: openin X U a  $\in$  U l  $\in$  M
      and Unless:  $\forall x \in U - \{a\}. x \in S \longrightarrow f \ x \in M \wedge d(f \ x) \ l < \varepsilon / 2$ 
      unfolding limitin_metric_eventually_atin_within by (metis Diff_iff iff
insertI1 half_gt_zero [OF  $\langle \varepsilon > 0 \rangle$ ])
    show  $\exists U. \text{openin } X \ U \wedge a \in U \wedge (\forall x \in S \cap U - \{a\}. \forall y \in S \cap U - \{a\}. d(f \ x) \ (f \ y) < \varepsilon)$ 
    proof (intro exI strip conjI)
      fix x y
      assume x:  $x \in S \cap U - \{a\}$  and y:  $y \in S \cap U - \{a\}$ 
      then have  $d(f \ x) \ l < \varepsilon / 2 \ d(f \ y) \ l < \varepsilon / 2 \ f \ x \in M \ f \ y \in M$ 
        using Unless by auto
      then show  $d(f \ x) \ (f \ y) < \varepsilon$ 
        using triangle'  $\langle l \in M \rangle$  by fastforce
    qed (auto simp add: U)
  qed
moreover have  $\exists l. \text{limitin mtopology f l (atin_within X a S)}$ 
  if R [rule_format]: ?R
proof -
  define F where F  $\equiv \lambda U. \text{mtopology closure\_of } f \ ' (S \cap U - \{a\})$ 
  define C where C  $\equiv F \ ' \{U. \text{openin } X \ U \wedge a \in U\}$ 
  have C_clo:  $\forall C \in C. \text{closedin mtopology } C$ 
    by (force simp add: C_def F_def)
  moreover have sub_mcball:  $\exists C \ a. C \in C \wedge C \subseteq \text{mcball } a \ \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
  proof -
    obtain U where U: openin X U a  $\in$  U
      and Unless:  $\forall x \in S \cap U - \{a\}. \forall y \in S \cap U - \{a\}. d(f \ x) \ (f \ y) < \varepsilon$ 
      using R [OF  $\langle \varepsilon > 0 \rangle$ ] by blast
    then obtain b where b:  $b \neq a \ b \in S \ b \in U$ 
      using True by (auto simp add: in_derived_set_of)
    have U  $\subseteq$  topspace X
      by (simp add: U(1) openin_subset)
    have f b  $\in$  M
      using b  $\langle \text{openin } X \ U \rangle$  by (metis image_subset_iff_funcset Int_iff fim
image_eqI openin_subset subsetD)
    moreover
      have mtopology_closure_of f ' ((S  $\cap$  U) - {a})  $\subseteq$  mcball (f b)  $\varepsilon$ 
      proof (rule closure_of_minimal)

```

```

have f y ∈ M if y ∈ S and y ∈ U for y
  using ⟨U ⊆ topspace X⟩ fim that by (auto simp: Pi_iff)
moreover
have d (f b) (f y) ≤ ε if y ∈ S y ∈ U y ≠ a for y
  using that Uless b by force
ultimately show f ‘ (S ∩ U - {a}) ⊆ mcball (f b) ε
  by (force simp: ⟨f b ∈ M⟩)
qed auto
ultimately show ?thesis
  using U by (auto simp add: C_def F_def)
qed
moreover have ⋂ F ≠ {} if finite F F ⊆ C for F
proof -
  obtain G where sub: G ⊆ {U. openin X U ∧ a ∈ U} and eq: F = F ‘ G
and finite G
  by (metis (no_types, lifting) C_def ⟨F ⊆ C⟩ ⟨finite F⟩ finite_subset_image)
  then have U ⊆ topspace X if U ∈ G for U
    using openin_subset that by auto
  then have T ⊆ mtopology closure_of T
    if T ∈ (λU. f ‘ (S ∩ U - {a})) ‘ G for T
    using that fim by (fastforce simp add: intro!: closure_of_subset)
  moreover
  have ain: a ∈ ⋂ (insert (topspace X) G) openin X (⋂ (insert (topspace
X) G))
    using True in_derived_set_of sub ⟨finite G⟩ by (fastforce intro!:
openin_Inter)+
  then obtain y where y ≠ a y ∈ S and y: y ∈ ⋂ (insert (topspace X) G)
    by (meson ⟨a ∈ X derived_set_of S⟩ sub in_derived_set_of)
  then have f y ∈ ⋂ F
    using eq that ain fim by (auto simp add: F_def image_subset_iff
in_closure_of)
  then show ?thesis by blast
qed
ultimately have ⋂ C ≠ {}
  using ⟨mcomplete⟩ mcomplete_fip by metis
then obtain b where b ∈ ⋂ C
  by auto
then have b ∈ M
  using sub_mcball C_clo mbounded_alt_pos mbounded_empty met-
ric_closedin_iff_sequentially_closed by force
have limitin mtopology f b (atin_within X a S)
proof (clarsimp simp: limitin_metric ⟨b ∈ M⟩)
  fix ε :: real
  assume ε > 0
  then obtain U where U: openin X U a ∈ U and subU: U ⊆ topspace X
    and Uless: ∀ x ∈ S ∩ U - {a}. ∀ y ∈ S ∩ U - {a}. d (f x) (f y) < ε/2
    by (metis R_half_gt_zero openin_subset)
  then obtain x where x: x ∈ S x ∈ U x ≠ a and fx: f x ∈ mball b (ε/2)
    using ⟨b ∈ ⋂ C⟩

```

```

    apply (simp add: C_def F_def closure_of_def del: divide_const_simps)
    by (metis Diff_iff Int_iff centre_in_mball_iff in_mball openin_mball
singletonI zero_less_numeral)
  moreover
  have d (f y) b < ε if y ∈ U y ≠ a y ∈ S for y
  proof -
    have d (f x) (f y) < ε/2
    using Unless that x by force
    moreover have d b (f x) < ε/2
    using fx by simp
    ultimately show ?thesis
    using triangle [of b f x f y] subU that ⟨b ∈ M⟩ commute fim fx by
fastforce
  qed
  ultimately show ∀ F x in atin_within X a S. f x ∈ M ∧ d (f x) b < ε
  using fim U
  apply (simp add: eventually_atin_within del: divide_const_simps flip:
image_subset_iff_funcset)
  by (smt (verit, del_insts) Diff_iff Int_iff imageI insertI1 openin_subset
subsetD)
  qed
  then show ?thesis ..
  qed
  ultimately
  show ?thesis
  by (meson True ⟨M ≠ {}⟩ in_derived_set_of)
next
case False
have (∃ l. limitin_mtopology f l (atin_within X a S))
  by (metis ⟨M ≠ {}⟩ False derived_set_of_trivial_limit equalsOI lim-
itin_trivial topology_mtopology)
  moreover have ∀ e>0. ∃ U. openin X U ∧ a ∈ U ∧ (∀ x∈S ∩ U - {a}.
∀ y∈S ∩ U - {a}. d (f x) (f y) < e)
  by (metis Diff_iff False IntE True in_derived_set_of insert_iff)
  ultimately show ?thesis
  using limitin_mspace by blast
  qed
next
case False
then show ?thesis
  by (metis derived_set_of_trivial_limit ex_in_conv in_derived_set_of lim-
itin_mspace limitin_trivial topology_mtopology)
  qed
qed

```

The HOL Light proof uses some ugly tricks to share common parts of what are two separate proofs for the two cases

lemma (in *Metric_space*) *gdelta_in_points_of_convergence_within*:
assumes *mcomplete*

```

    and f: continuous_map (subtopology X S) mtopology f ∨ t1_space X ∧ f ∈ S
  → M
  shows gdelta_in X {x ∈ topspace X. ∃ l. limitin mtopology f l (atin_within X x
  S)}
  proof -
    have fim: f ∈ (topspace X ∩ S) → M
      using continuous_map_image_subset_topspace f by force
    show ?thesis
    proof (cases M={})
      case True
      then show ?thesis
        by (smt (verit) Collect_cong_empty_def empty_iff gdelta_in_empty lim-
        itin_mspace)
      next
      case False
      define A where A ≡ {a ∈ topspace X. ∀ ε > 0. ∃ U. openin X U ∧ a ∈ U ∧
      (∀ x ∈ S ∩ U - {a}. ∀ y ∈ S ∩ U - {a}. d (f x) (f y) < ε)}
      have gdelta_in X A
      using f
      proof (elim disjE conjE)
        assume cm: continuous_map (subtopology X S) mtopology f
        define C where C ≡ λ r. ⋃ {U. openin X U ∧ (∀ x ∈ S ∩ U. ∀ y ∈ S ∩ U. d
      (f x) (f y) < r)}
        define B where B ≡ (⋂ n. C (inverse (Suc n)))
        define D where D ≡ (⋂ (C ' {0 < ..}))
        have D=B
          unfolding B_def C_def D_def
          apply (intro Inter_eq Inter_inverse_Suc Sup_subset_mono)
          by (smt (verit, ccfv_threshold) Collect_mono_iff)
        have B ⊆ topspace X
          using openin_subset by (force simp add: B_def C_def)
        have (countable_intersection_of openin X) B
          unfolding B_def C_def
          by (intro relative_to_inc countable_intersection_of_Inter countable_intersection_of_inc)
        auto
        then have gdelta_in X B
          unfolding gdelta_in_def by (intro relative_to_subset_inc ⟨B ⊆ topspace
      X⟩)
        moreover have A=D
        proof (intro equalityI subsetI)
          fix a
          assume x: a ∈ A
          then have a ∈ topspace X
            using A_def by blast
          show a ∈ D
          proof (clarsimp simp: D_def C_def ⟨a ∈ topspace X⟩)
            fix ε::real assume ε > 0
            then obtain U where openin X U a ∈ U and U: (∀ x ∈ S ∩ U - {a}.
            ∀ y ∈ S ∩ U - {a}. d (f x) (f y) < ε)

```

```

    using x by (force simp: A_def)
  show  $\exists T. \text{openin } X \ T \wedge (\forall x \in S \cap T. \forall y \in S \cap T. d(f\ x) (f\ y) < \varepsilon) \wedge a \in T$ 
  proof (cases a  $\in S$ )
    case True
      then obtain V where  $\text{openin } X \ V \ a \in V$  and  $V: \forall x. x \in S \wedge x \in V \rightarrow f\ a \in M \wedge f\ x \in M \wedge d(f\ a) (f\ x) < \varepsilon$ 
        using  $\langle a \in \text{topspace } X \rangle \langle \varepsilon > 0 \rangle$  cm
        by (force simp add: continuous_map_to_metric_openin_subtopology_alt Ball_def)
      show ?thesis
        proof (intro exI conjI strip)
          show  $\text{openin } X \ (U \cap V)$ 
            using  $\langle \text{openin } X \ U \rangle \langle \text{openin } X \ V \rangle$  by blast
          show  $a \in U \cap V$ 
            using  $\langle a \in U \rangle \langle a \in V \rangle$  by blast
          show  $\bigwedge x\ y. \llbracket x \in S \cap (U \cap V); y \in S \cap (U \cap V) \rrbracket \implies d(f\ x) (f\ y) < \varepsilon$ 
            by (metis DiffI Int_iff U V commute singletonD)
        qed
      next
        case False then show ?thesis
          using  $U \langle a \in U \rangle \langle \text{openin } X \ U \rangle$  by auto
        qed
      next
        fix x
        assume  $x: x \in D$ 
        then have  $x \in \text{topspace } X$ 
          using  $\langle B \subseteq \text{topspace } X \rangle \langle D=B \rangle$  by blast
        with x show  $x \in A$ 
          apply (clarsimp simp: D_def C_def A_def)
          by (meson DiffD1 greaterThan_iff)
        qed
      ultimately show ?thesis
        by (simp add:  $\langle D=B \rangle$ )
    next
      assume  $t1\_space\ X\ f \in S \rightarrow M$ 
      define C where  $C \equiv \lambda r. \bigcup \{U. \text{openin } X \ U \wedge (\exists b \in \text{topspace } X. \forall x \in S \cap U - \{b\}. \forall y \in S \cap U - \{b\}. d(f\ x) (f\ y) < r)\}$ 
      define B where  $B \equiv (\bigcap n. C(\text{inverse}(Suc\ n)))$ 
      define D where  $D \equiv (\bigcap (C\ ' \{0<..\}))$ 
      have  $D=B$ 
        unfolding B_def C_def D_def
        apply (intro Inter_eq Inter_inverse_Suc Sup_subset_mono)
        by (smt (verit, ccfv_threshold) Collect_mono_iff)
      have  $B \subseteq \text{topspace } X$ 
        using openin_subset by (force simp add: B_def C_def)

```

```

have (countable_intersection_of_openin X) B
  unfolding B_def C_def
by (intro relative_to_inc countable_intersection_of_Inter countable_intersection_of_inc)
auto
then have gdelta_in X B
  unfolding gdelta_in_def by (intro relative_to_subset_inc ⟨B ⊆ topspace
X⟩)
moreover have A=D
proof (intro equalityI subsetI)
  fix x
  assume x: x ∈ D
  then have x ∈ topspace X
    using ⟨B ⊆ topspace X⟩ ⟨D=B⟩ by blast
  show x ∈ A
  proof (clarsimp simp: A_def ⟨x ∈ topspace X⟩)
    fix ε :: real
    assume ε>0
    then obtain U b where openin X U b ∈ topspace X
      and U: ∀ x∈S ∩ U - {b}. ∀ y∈S ∩ U - {b}. d (f x) (f y) < ε and
x ∈ U
      using x by (auto simp: D_def C_def A_def Ball_def)
    then have openin X (U - {b})
      by (meson ⟨t1_space X⟩ t1_space_openin_delete_alt)
    then show ∃ U. openin X U ∧ x ∈ U ∧ (∀ xa∈S ∩ U - {x}. ∀ y∈S ∩ U
- {x}. d (f xa) (f y) < ε)
      using U ⟨openin X U⟩ ⟨x ∈ U⟩ by auto
    qed
  next
  show ∧x. x ∈ A ⇒ x ∈ D
    unfolding A_def D_def C_def
    by clarsimp meson
  qed
  ultimately show ?thesis
    by (simp add: ⟨D=B⟩)
  qed
  then show ?thesis
    by (simp add: A_def convergent_eq_zero_oscillation_gen False fim ⟨mcom-
plete⟩ cong: conj_cong)
  qed
qed

```

lemma gdelta_in_points_of_convergence_within:

```

assumes Y: completely_metrizable_space Y
  and f: continuous_map (subtopology X S) Y f ∨ t1_space X ∧ f ∈ S →
topspace Y
shows gdelta_in X {x ∈ topspace X. ∃ l. limitin Y f l (atin_within X x S)}
using assms
unfolding completely_metrizable_space_def

```

using *Metric_space.gdelta_in_points_of_convergence_within Metric_space.topspace_mtopology*
by *fastforce*

lemma *Laurentiev_extension_gen:*

assumes $S \subseteq \text{topspace } X$ **and** Y : *completely_metrizable_space* Y
and *contf*: *continuous_map* (*subtopology* X S) Y f
obtains U g **where** *gdelta_in* X U $S \subseteq U$
continuous_map (*subtopology* X (X *closure_of* $S \cap U$)) Y g
 $\bigwedge x. x \in S \implies g\ x = f\ x$

proof –

define U **where** $U \equiv \{x \in \text{topspace } X. \exists l. \text{limitin } Y\ f\ l\ (\text{atin_within } X\ x\ S)\}$
have $S \subseteq U$
using *that contf limit_continuous_map_within subsetD [OF ‹ $S \subseteq \text{topspace } X$ ›]*

by (*fastforce simp: U_def*)

then have $S \subseteq X$ *closure_of* $S \cap U$

by (*simp add: ‹ $S \subseteq \text{topspace } X$ › closure_of_subset*)

moreover

have $\bigwedge t. t \in X$ *closure_of* $S \cap U - S \implies \exists l. \text{limitin } Y\ f\ l\ (\text{atin_within } X\ t\ S)$

using U_def **by** *blast*

moreover have *regular_space* Y

by (*simp add: Y completely_metrizable_imp_metrizable_space metrizable_imp_regular_space*)

ultimately

obtain g **where** g : *continuous_map* (*subtopology* X (X *closure_of* $S \cap U$)) Y g
and gf : $\bigwedge x. x \in S \implies g\ x = f\ x$

using *continuous_map_extension_pointwise_alt* **assms** **by** *blast*

show *thesis*

proof

show *gdelta_in* X U

by (*simp add: U_def Y contf gdelta_in_points_of_convergence_within*)

show *continuous_map* (*subtopology* X (X *closure_of* $S \cap U$)) Y g

by (*simp add: g*)

qed (*use ‹ $S \subseteq U$ › gf in auto*)

qed

lemma *Laurentiev_extension:*

assumes $S \subseteq \text{topspace } X$

and X : *metrizable_space* $X \vee \text{topspace } X \subseteq X$ *closure_of* S

and Y : *completely_metrizable_space* Y

and *contf*: *continuous_map* (*subtopology* X S) Y f

obtains U g **where** *gdelta_in* X U $S \subseteq U$ $U \subseteq X$ *closure_of* S

continuous_map (*subtopology* X U) Y g $\bigwedge x. x \in S \implies g\ x = f\ x$

proof –

obtain U g **where** *gdelta_in* X U $S \subseteq U$

and *contg*: *continuous_map* (*subtopology* X (X *closure_of* $S \cap U$)) Y g

and gf : $\bigwedge x. x \in S \implies g\ x = f\ x$

using *Laurentiev_extension_gen* Y *assms*(1) *contf* **by** *blast*

define V **where** $V \equiv X$ *closure_of* $S \cap U$

```

show thesis
proof
  show gdelta_in X V
  by (metis V_def X ⟨gdelta_in X U⟩ closed_imp_gdelta_in closedin_closure_of
closure_of_subset_topspace gdelta_in_Int gdelta_in_topspace subset_antisym)
  show  $S \subseteq V$ 
  by (simp add: V_def ⟨S ⊆ U⟩ assms(1) closure_of_subset)
  show continuous_map (subtopology X V) Y g
  by (simp add: V_def contg)
qed (auto simp: gf V_def)
qed

```

6.11.19 Embedding in products and hence more about completely metrizable spaces

```

lemma (in Metric_space) gdelta_homeomorphic_space_closedin_product:
  assumes  $S: \bigwedge i. i \in I \implies \text{openin } mtopology (S \ i)$ 
  obtains  $T$  where closedin (prod_topology mtopology (powertop_real I)) T
subtopology mtopology ( $\bigcap i \in I. S \ i$ ) homeomorphic_space
subtopology (prod_topology mtopology (powertop_real I)) T
proof (cases I={})
  case True
  then have top: topspace (prod_topology mtopology (powertop_real I)) = (M ×
{(λx. undefined)}))
  by simp
  show ?thesis
  proof
  show closedin (prod_topology mtopology (powertop_real I)) (M × {(λx. unde-
defined)})
  by (metis top closedin_topspace)
  have subtopology mtopology ( $\bigcap (S \ i)$ ) homeomorphic_space mtopology
by (simp add: True product_topology_empty_discrete)
  also have ... homeomorphic_space (prod_topology mtopology (discrete_topology
{λx. undefined}))
  by (meson homeomorphic_space_sym prod_topology_homeomorphic_space_left)
  finally
  show subtopology mtopology ( $\bigcap (S \ i)$ ) homeomorphic_space subtopology (prod_topology
mtopology (powertop_real I)) (M × {(λx. undefined)})
  by (smt (verit, ccfv_SIG) True product_topology_empty_discrete subtopol-
ogy_topspace top)
  qed
next
  case False
  have  $SM: \bigwedge i. i \in I \implies S \ i \subseteq M$ 
  using assms openin_mtopology by blast
  then have  $(\bigcap i \in I. S \ i) \subseteq M$ 
  using False by blast
  define dd where dd ≡ λi. if i∉I ∨ S i = M then λu. 1 else (λu. INF x ∈ M -
S i. d u x)

```



```

have [simp]: bdd_below (d u ' A) for u A
  by (meson bdd_belowI2 nonneg)
have cont_dd: continuous_map (subtopology mtopology (S i)) euclidean (dd i)
if i ∈ I for i
proof -
  have dist (Inf (d x ' (M - S i))) (Inf (d y ' (M - S i))) ≤ d x y
    if x ∈ S i x ∈ M y ∈ S i y ∈ M S i ≠ M for x y
  proof -
    have [simp]: ¬ M ⊆ S i
      using SM ⟨S i ≠ M⟩ ⟨i ∈ I⟩ by auto
    have ∧u. [u ∈ M; u ∉ S i] ⇒ Inf (d x ' (M - S i)) ≤ d x y + d y u
      apply (clarsimp simp add: cInf_le_iff_less)
      by (smt (verit) DiffI triangle ⟨x ∈ M⟩ ⟨y ∈ M⟩)
    then have Inf (d x ' (M - S i)) - d x y ≤ Inf (d y ' (M - S i))
      by (force simp add: le_cInf_iff)
    moreover
    have ∧u. [u ∈ M; u ∉ S i] ⇒ Inf (d y ' (M - S i)) ≤ d x u + d x y
      apply (clarsimp simp add: cInf_le_iff_less)
      by (smt (verit) DiffI triangle'' ⟨x ∈ M⟩ ⟨y ∈ M⟩)
    then have Inf (d y ' (M - S i)) - d x y ≤ Inf (d x ' (M - S i))
      by (force simp add: le_cInf_iff)
    ultimately show ?thesis
      by (simp add: dist_real_def abs_le_iff)
  qed
  then have *: Lipschitz_continuous_map (submetric Self (S i)) euclidean_metric
    (λu. Inf (d u ' (M - S i)))
    unfolding Lipschitz_continuous_map_def by (force intro!: exI [where x=1])
  then show ?thesis
    using Lipschitz_continuous_imp_continuous_map [OF *]
    by (simp add: dd_def Self_def mtopology_of_submetric)
qed
have dd_pos: 0 < dd i x if x ∈ S i for i x
proof (clarsimp simp add: dd_def)
  assume i ∈ I and S i ≠ M
  have opeS: openin mtopology (S i)
    by (simp add: ⟨i ∈ I⟩ assms)
  then obtain r where r>0 and r: ∧y. [y ∈ M; d x y < r] ⇒ y ∈ S i
    by (meson ⟨x ∈ S i⟩ in_mball openin_mtopology subsetD)
  then have ∧y. y ∈ M - S i ⇒ d x y ≥ r
    by (meson Diff_iff linorder_not_le)
  then have Inf (d x ' (M - S i)) ≥ r
    by (meson Diff_eq_empty_iff SM ⟨S i ≠ M⟩ ⟨i ∈ I⟩ cINF_greatest set_eq_subset)
  with ⟨r>0⟩ show 0 < Inf (d x ' (M - S i)) by simp
qed
define f where f ≡ λx. (x, λi∈I. inverse(dd i x))
define T where T ≡ f ' (∩ i ∈ I. S i)
show ?thesis
proof
  show closedin (prod_topology mtopology (powertop_real I)) T

```

```

unfolding closure_of_subset_eq [symmetric]
proof
  show  $T \subseteq \text{topspace } (\text{prod\_topology } \text{mtopology } (\text{powertop\_real } I))$ 
    using False SM by (auto simp: T_def f_def)

  have  $(x, ds) \in T$ 
    if  $\S: \bigwedge U. [(x, ds) \in U; \text{openin } (\text{prod\_topology } \text{mtopology } (\text{powertop\_real } I)) U] \implies \exists y \in T. y \in U$ 
      and  $x \in M$  and  $ds: ds \in I \rightarrow_E \text{UNIV}$  for  $x ds$ 
    proof -
      have ope:  $\exists x. x \in \bigcap (S \text{ ' } I) \wedge f x \in U \times V$ 
        if  $x \in U$  and  $ds \in V$  and  $\text{openin } \text{mtopology } U$  and  $\text{openin } (\text{powertop\_real } I) V$  for  $U V$ 
        using  $\S$  [of  $U \times V$ ] that by (force simp add: T_def openin_prod_Times_iff)
        have x_in_INT:  $x \in \bigcap (S \text{ ' } I)$ 
          proof clarify
            fix i
            assume  $i \in I$ 
            show  $x \in S i$ 
            proof (rule ccontr)
              assume  $x \notin S i$ 
              have  $\text{openin } (\text{powertop\_real } I) \{z \in \text{topspace } (\text{powertop\_real } I). z i \in \{ds i - 1 <..< ds i + 1\}\}$ 
                proof (rule openin_continuous_map_preimage)
                  show  $\text{continuous\_map } (\text{powertop\_real } I) \text{euclidean } (\lambda x. x i)$ 
                    by (metis  $\langle i \in I \rangle \text{continuous\_map\_product\_projection}$ )
                qed auto
              then obtain  $y$  where  $y \in S i$   $y \in M$  and  $dxy: d x y < \text{inverse } (|ds i| + 1)$ 
                and  $\text{intvl}: \text{inverse } (dd i y) \in \{ds i - 1 <..< ds i + 1\}$ 
                using ope [of  $\text{mball } x (\text{inverse } (\text{abs } (ds i) + 1)) \{z \in \text{topspace } (\text{powertop\_real } I). z i \in \{ds i - 1 <..< ds i + 1\}\}$ ]
                 $\langle x \in M \rangle \langle ds \in I \rightarrow_E \text{UNIV} \rangle \langle i \in I \rangle$ 
                by (fastforce simp add: f_def)
              have  $\neg M \subseteq S i$ 
                using  $\langle x \notin S i \rangle \langle x \in M \rangle$  by blast
              have  $\text{inverse } (|ds i| + 1) \leq dd i y$ 
                using  $\text{intvl } \langle y \in S i \rangle dd\_pos$  [of  $y i$ ]
              by (smt (verit, ccfv_threshold) greaterThanLessThan_iff inverse_inverse_eq le_imp_inverse_le)
              also have  $\dots \leq d x y$ 
                using  $\langle i \in I \rangle \langle \neg M \subseteq S i \rangle \langle x \notin S i \rangle \langle x \in M \rangle$ 
                apply (simp add: dd_def cInf_le_iff_less)
                using commute by force
              finally show False
                using  $dxy$  by linarith
            qed
          qed
        moreover have  $ds = (\lambda i \in I. \text{inverse } (dd i x))$ 

```

```

proof (rule PiE_ext [OF ds])
  fix i
  assume i ∈ I
  define e where e ≡ |ds i - inverse (dd i x)|
  { assume con: e > 0
  have continuous_map (subtopology mtopology (S i)) euclidean (λx. inverse
  (dd i x))
    using dd_pos cont_dd ⟨i ∈ I⟩
    by (fastforce simp: intro!: continuous_map_real_inverse)
  then have openin (subtopology mtopology (S i))
    {z ∈ topspace (subtopology mtopology (S i)).
    inverse (dd i z) ∈ {inverse(dd i x) - e/2 <..using openin_continuous_map_preimage open_greaterThanLessThan
open_openin by blast
  then obtain U where openin mtopology U
    and U: {z ∈ S i. inverse (dd i x) - e/2 < inverse (dd i z) ∧
    inverse (dd i z) < inverse (dd i x) + e/2}
    = U ∩ S i
    using SM ⟨i ∈ I⟩ by (auto simp: openin_subtopology)
  have x ∈ U
    using U_x_in_INT ⟨i ∈ I⟩ con by fastforce
  have ds ∈ {z ∈ topspace (powertop_real I). z i ∈ {ds i - e / 2 <..by (simp add: con ds)
  moreover
  have openin (powertop_real I) {z ∈ topspace (powertop_real I). z i ∈
{ds i - e / 2 <..proof (rule openin_continuous_map_preimage)
    show continuous_map (powertop_real I) euclidean (λx. x i)
      by (metis ⟨i ∈ I⟩ continuous_map_product_projection)
    qed auto
  ultimately obtain y where y ∈ ∩(S ' I) ∧ f y ∈ U × {z ∈ topspace
(powertop_real I). z i ∈ {ds i - e / 2 <..using ope ⟨x ∈ U⟩ ⟨openin mtopology U⟩ ⟨x ∈ U⟩
    by presburger
  with ⟨i ∈ I⟩ U
    have False unfolding set_eq_iff f_def e_def by simp (smt (verit)
field_sum_of_halves)
  }
  then show ds i = (λi∈I. inverse (dd i x)) i
    using ⟨i ∈ I⟩ by (force simp: e_def)
  qed auto
  ultimately show ?thesis
    by (auto simp: T_def f_def)
  qed
  then show prod_topology mtopology (powertop_real I) closure_of T ⊆ T
    by (auto simp: closure_of_def)
  qed

```

```

have eq:  $(\bigcap (S \text{ ' } I) \times (I \rightarrow_E UNIV) \cap f \text{ ' } (M \cap \bigcap (S \text{ ' } I))) = (f \text{ ' } \bigcap (S \text{ ' } I))$ 
using False SM by (force simp: f_def image_iff)
have continuous_map (subtopology mtopology  $(\bigcap (S \text{ ' } I))$ ) euclidean (dd i) if i
∈ I for i
by (meson INT_lower cont_dd continuous_map_from_subtopology_mono
that)
then have continuous_map (subtopology mtopology  $(\bigcap (S \text{ ' } I))$ ) (powertop_real
I)  $(\lambda x. \lambda i \in I. \text{inverse } (dd \ i \ x))$ 
using dd_pos by (fastforce simp: continuous_map_componentwise intro!:
continuous_map_real_inverse)
then have embedding_map (subtopology mtopology  $(\bigcap (S \text{ ' } I))$ ) (prod_topology
(subtopology mtopology  $(\bigcap (S \text{ ' } I))$ ) (powertop_real I)) f
by (simp add: embedding_map_graph f_def)
moreover have subtopology (prod_topology (subtopology mtopology  $(\bigcap (S \text{ ' } I))$ )
(powertop_real I))
 $(f \text{ ' } \text{topspace } (subtopology \ mtopology \ (\bigcap (S \text{ ' } I)))) =$ 
subtopology (prod_topology mtopology (powertop_real I)) T
by (simp add: prod_topology_subtopology_subtopology T_def eq)
ultimately
show subtopology mtopology  $(\bigcap (S \text{ ' } I))$  homeomorphic_space subtopology (prod_topology
mtopology (powertop_real I)) T
by (metis embedding_map_imp_homeomorphic_space)
qed
qed

```

```

lemma gdelta_homeomorphic_space_closedin_product:
assumes metrizable_space X and  $\bigwedge i. i \in I \implies \text{openin } X \ (S \ i)$ 
obtains T where closedin (prod_topology X (powertop_real I)) T
subtopology X  $(\bigcap i \in I. S \ i)$  homeomorphic_space
subtopology (prod_topology X (powertop_real I)) T
using Metric_space.gdelta_homeomorphic_space_closedin_product
by (metis assms metrizable_space_def)

```

```

lemma open_homeomorphic_space_closedin_product:
assumes metrizable_space X and openin X S
obtains T where closedin (prod_topology X euclideanreal) T
subtopology X S homeomorphic_space
subtopology (prod_topology X euclideanreal) T

```

proof –

```

obtain T where cloT: closedin (prod_topology X (powertop_real  $\{()\}$ )) T
and homT: subtopology X S homeomorphic_space
subtopology (prod_topology X (powertop_real  $\{()\}$ )) T
using gdelta_homeomorphic_space_closedin_product [of X  $\{()\}$   $\lambda i. S$ ] assms
by auto
have prod_topology X (powertop_real  $\{()\}$ ) homeomorphic_space prod_topology
X euclideanreal
by (meson homeomorphic_space_prod_topology_homeomorphic_space_refl homeomorphic_space_singleton_product)

```

```

then obtain  $f$  where  $f$ : homeomorphic_map (prod_topology X (powertop_real
{()})) (prod_topology X euclideanreal)  $f$ 
  unfolding homeomorphic_space by metis
  show thesis
proof
  show closedin (prod_topology X euclideanreal) ( $f$  ‘  $T$ )
    using cloT f homeomorphic_map_closedness_eq by blast
  moreover have  $T = \text{topspace}$  (subtopology (prod_topology X (powertop_real
{()}))  $T$ )
    by (metis cloT closedin_subset topspace_subtopology_subset)
  ultimately show subtopology X  $S$  homeomorphic_space subtopology (prod_topology
X euclideanreal) ( $f$  ‘  $T$ )
    by (smt (verit, best) closedin_subset f homT homeomorphic_map_subtopologies
homeomorphic_space
      homeomorphic_space_trans topspace_subtopology topspace_subtopology_subset)
  qed
qed

```

lemma *completely_metrizable_space_gdelta_in_alt*:

```

assumes  $X$ : completely_metrizable_space  $X$ 
  and  $S$ : (countable_intersection_of_openin  $X$ )  $S$ 
shows completely_metrizable_space (subtopology  $X$   $S$ )
proof –
  obtain  $\mathcal{U}$  where countable  $\mathcal{U}$   $S = \bigcap \mathcal{U}$  and ope:  $\bigwedge U. U \in \mathcal{U} \implies \text{openin } X U$ 
    using  $S$  by (force simp add: intersection_of_def)
  then have  $\mathcal{U}$ : completely_metrizable_space (powertop_real  $\mathcal{U}$ )
    by (simp add: completely_metrizable_space_euclidean completely_metrizable_space_product_topology)
  obtain  $C$  where closedin (prod_topology X (powertop_real  $\mathcal{U}$ ))  $C$ 
    and sub: subtopology X ( $\bigcap \mathcal{U}$ ) homeomorphic_space
      subtopology (prod_topology X (powertop_real  $\mathcal{U}$ ))  $C$ 
    by (metis gdelta_homeomorphic_space_closedin_product X completely_metrizable_imp_metrizable_space
ope INF_identity_eq)
  moreover have completely_metrizable_space (prod_topology X (powertop_real
 $\mathcal{U}$ ))
    by (simp add: completely_metrizable_space_prod_topology X  $\mathcal{U}$ )
  ultimately have completely_metrizable_space (subtopology (prod_topology X
(powertop_real  $\mathcal{U}$ ))  $C$ )
    using completely_metrizable_space_closedin by blast
  then show thesis
    using  $\langle S = \bigcap \mathcal{U} \rangle$  sub homeomorphic_completely_metrizable_space by blast
qed

```

lemma *completely_metrizable_space_gdelta_in*:

```

[[completely_metrizable_space  $X$ ; gdelta_in  $X$   $S$ ]]
   $\implies$  completely_metrizable_space (subtopology  $X$   $S$ )
by (simp add: completely_metrizable_space_gdelta_in_alt gdelta_in_alt)

```

lemma *completely_metrizable_space_openin*:

```

[[completely_metrizable_space  $X$ ; openin  $X$   $S$ ]]

```

\implies *completely metrizable space* (subtopology $X S$)
 by (simp add: *completely metrizable space gdelta in open imp gdelta in*)

lemma (in *Metric_space*) *locally compact imp completely metrizable space*:

assumes *locally compact space mtopology*
shows *completely metrizable space mtopology*

proof –

obtain $f :: [a, a] \Rightarrow \text{real}$ **and** m' **where**

mcomplete_of m' **and** *fim*: $f \in M \rightarrow \text{mspace } m'$

and *clo*: *mtopology_of m' closure_of f ' M = mspace m'*

and d : $\bigwedge x y. \llbracket x \in M; y \in M \rrbracket \implies \text{mdist } m' (f x) (f y) = d x y$

by (*metis metric completion*)

then have *embedding_map mtopology (mtopology_of m') f*

unfolding *mtopology_of_def*

by (*metis Metric_space12.isometry_imp_embedding_map Metric_space12.mspace_mdists*
mdist_metric mspace_metric)

then have *hom: mtopology homeomorphic_space subtopology (mtopology_of m')*
(f ' M)

by (*metis embedding_map_imp_homeomorphic_space topspace_mtopology*)

have *locally compact space (subtopology (mtopology_of m') (f ' M))*

using *assms hom homeomorphic_locally_compact_space* **by** *blast*

moreover have *Hausdorff_space (mtopology_of m')*

by (*simp add: Metric_space.Hausdorff_space_mtopology_of_def*)

ultimately have *openin (mtopology_of m') (f ' M)*

by (*simp add: clo_dense_locally_compact_openin_Hausdorff_space fim image_subset_iff_funcset*)

then

have *completely metrizable space (subtopology (mtopology_of m') (f ' M))*

using $\langle \text{mcomplete_of } m' \rangle$ **unfolding** *mcomplete_of_def mtopology_of_def*

by (*metis Metric_space.completely_metrizable_space_mtopology Metric_space.mspace_mdists*
completely_metrizable_space_openin)

then show *?thesis*

using *hom homeomorphic_completely_metrizable_space* **by** *blast*

qed

lemma *locally compact imp completely metrizable space*:

assumes *metrizable_space X* **and** *locally_compact_space X*

shows *completely metrizable_space X*

by (*metis Metric_space.locally_compact_imp_completely_metrizable_space assms*
metrizable_space_def)

lemma *completely metrizable space imp gdelta in*:

assumes X : *metrizable_space X* **and** $S \subseteq$ *topspace X*

and $X S$: *completely metrizable_space (subtopology X S)*

shows *gdelta_in X S*

proof –

obtain $U f$ **where** *gdelta_in X U S \subseteq U* **and** U : $U \subseteq X$ *closure_of S*

```

    and contf: continuous_map (subtopology X U) (subtopology X S) f
    and fid:  $\bigwedge x. x \in S \implies f x = x$ 
  using Lavrentiev_extension[of S X subtopology X S id] assms by auto
  then have f' : topspace (subtopology X U)  $\subseteq$  topspace (subtopology X S)
    using continuous_map_image_subset_topospace by blast
  then have f'U  $\subseteq$  S
    by (metis  $\langle$ gdelta_in X U $\rangle$   $\langle$ S  $\subseteq$  topspace X $\rangle$  gdelta_in_subset_topospace_subtopology_subset)
  moreover
  have Hausdorff_space (subtopology X U)
    by (simp add: Hausdorff_space_subtopology X metrizable_imp_Hausdorff_space)
  then have  $\bigwedge x. x \in U \implies f x = x$ 
    using U fid contf forall_in_closure_of_eq [of _ subtopology X U S subtopology
X U f id]
    by (metis  $\langle$ S  $\subseteq$  U $\rangle$  closure_of_subtopology_open_continuous_map_id continuous_map_in_subtopology_id_apply inf.orderE subtopology_subtopology)
  ultimately have U  $\subseteq$  S
    by auto
  then show ?thesis
    using  $\langle$ S  $\subseteq$  U $\rangle$   $\langle$ gdelta_in X U $\rangle$  by auto
qed

```

lemma *completely_metrizable_space_eq_gdelta_in*:
 \llbracket completely_metrizable_space X; S \subseteq topspace X \rrbracket
 \implies completely_metrizable_space (subtopology X S) \longleftrightarrow gdelta_in X S
using completely_metrizable_imp_metrizable_space completely_metrizable_space_gdelta_in completely_metrizable_space_imp_gdelta_in **by** blast

lemma *gdelta_in_eq_completely_metrizable_space*:
completely_metrizable_space X
 \implies gdelta_in X S \longleftrightarrow S \subseteq topspace X \wedge completely_metrizable_space (subtopology X S)
by (metis completely_metrizable_space_eq_gdelta_in gdelta_in_alt)

6.11.20 Theorems from Kuratowski

Kuratowski, Remark on an Invariance Theorem, *Fundamenta Mathematicae* **37** (1950), pp. 251-252. The idea is that in suitable spaces, to show "number of components of the complement" (without distinguishing orders of infinity) is a homeomorphic invariant, it suffices to show it for closed subsets. Kuratowski states the main result for a "locally connected continuum", and seems clearly to be implicitly assuming that means metrizable. We call out the general topological hypotheses more explicitly, which do not however include connectedness.

lemma *separation_by_closed_intermediates_count*:
assumes X: hereditarily_normal_space X
and finite U
and pwU: pairwise (separated_in X) U
and nonempty: $\{\} \notin$ U

```

    and UU:  $\bigcup \mathcal{U} = \text{topspace } X - S$ 
    obtains C where closedin X C  $C \subseteq S$ 
       $\wedge D. \llbracket \text{closedin } X D; C \subseteq D; D \subseteq S \rrbracket$ 
       $\implies \exists \mathcal{V}. \mathcal{V} \approx \mathcal{U} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{V} \wedge \{\} \notin \mathcal{V} \wedge \bigcup \mathcal{V} =$ 
topspace X - D
  proof -
    obtain F where F:  $\wedge S. S \in \mathcal{U} \implies \text{openin } X (F S) \wedge S \subseteq F S$ 
    and pwF: pairwise  $(\lambda S T. \text{disjnt } (F S) (F T)) \mathcal{U}$ 
    using assms by (smt (verit, best) Diff_subset Sup_le_iff hereditarily_normal_separation_pairwise)
    show thesis
  proof
    show closedin X (topspace X -  $\bigcup (F ` \mathcal{U})$ )
      using F by blast
    show topspace X -  $\bigcup (F ` \mathcal{U}) \subseteq S$ 
      using UU F by auto
    show  $\exists \mathcal{V}. \mathcal{V} \approx \mathcal{U} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{V} \wedge \{\} \notin \mathcal{V} \wedge \bigcup \mathcal{V} = \text{topspace } X$ 
- C
      if closedin X C  $C \subseteq S$  and C: topspace X -  $\bigcup (F ` \mathcal{U}) \subseteq C$  for C
    proof (intro exI conjI strip)
      have inj_on  $(\lambda S. F S - C) \mathcal{U}$ 
        using pwF F
      unfolding inj_on_def pairwise_def disjnt_iff
      by (metis Diff_iff UU UnionI nonempty_subset_empty_subset_eq  $\langle C \subseteq S \rangle$ )
      then show  $(\lambda S. F S - C) ` \mathcal{U} \approx \mathcal{U}$ 
        by simp
      show pairwise (separatedin X)  $((\lambda S. F S - C) ` \mathcal{U})$ 
        using  $\langle \text{closedin } X C \rangle F pwF$  by (force simp: pairwise_def openin_diff
separatedin_open_sets disjnt_iff)
      show  $\{\} \notin (\lambda S. F S - C) ` \mathcal{U}$ 
        using nonempty UU  $\langle C \subseteq S \rangle F$ 
      by clarify (metis DiffD2 Diff_eq_empty_iff F UnionI subset_empty_subset_eq)
      show  $(\bigcup S \in \mathcal{U}. F S - C) = \text{topspace } X - C$ 
        using UU F C openin_subset by fastforce
    qed
  qed
qed

```

lemma separation_by_closed_intermediates_gen:
assumes X: hereditarily_normal_space X
and discon: $\neg \text{connectedin } X (\text{topspace } X - S)$
obtains C where closedin X C $C \subseteq S$
 $\wedge D. \llbracket \text{closedin } X D; C \subseteq D; D \subseteq S \rrbracket \implies \neg \text{connectedin } X (\text{topspace } X - D)$
proof -
obtain C1 C2 where Ueq: $C1 \cup C2 = \text{topspace } X - S$ **and** C1 $\neq \{\}$ C2 $\neq \{\}$
and sep: separatedin X C1 C2 **and** C1 $\neq C2$
by (metis Diff_subset connectedin_eq_not_separated discon separatedin_refl)
then obtain C where closedin X C $C \subseteq S$


```

and C:  $\bigwedge D. \llbracket \text{closedin } X D; C \subseteq D; D \subseteq S \rrbracket$ 
            $\implies \exists \mathcal{V}. \mathcal{V} \approx \{C1, C2\} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{V} \wedge \{\} \notin \mathcal{V} \wedge$ 
 $\bigcup \mathcal{V} = \text{topspace } X - D$ 
using separation_by_closed_intermediates_count [of X {C1, C2} S] X
apply (simp add: pairwise_insert separatedin_sym)
by metis
have  $\neg \text{connectedin } X (\text{topspace } X - D)$ 
if D:  $\text{closedin } X D \wedge C \subseteq D \wedge D \subseteq S$  for D
proof -
obtain V1 V2 where *:  $\text{pairwise } (\text{separatedin } X) \{V1, V2\} \wedge \{\} \notin \{V1, V2\}$ 
            $\bigcup \{V1, V2\} = \text{topspace } X - D \wedge V1 \neq V2$ 
by (metis C [OF D]  $\langle C1 \neq C2 \rangle$  eqpoll_doubleton_iff)
then have disjoint V1 V2
by (metis pairwise_insert separatedin_imp_disjoint singleton_iff)
with * show ?thesis
by (auto simp add: connectedin_eq_not_separated pairwise_insert)
qed
then show thesis
using  $\langle C \subseteq S \rangle \langle \text{closedin } X C \rangle$  that by auto
qed

```

lemma separation_by_closed_intermediates_eq_count:

```

fixes n::nat
assumes lcX: locally_connected_space X and hnX: hereditarily_normal_space X
shows  $(\exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup \mathcal{U} = \text{topspace } X - S)$ 
 $\iff$ 
 $(\exists C. \text{closedin } X C \wedge C \subseteq S \wedge$ 
 $(\forall D. \text{closedin } X D \wedge C \subseteq D \wedge D \subseteq S$ 
 $\implies (\exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge$ 
 $\bigcup \mathcal{U} = \text{topspace } X - D)))$ 
(is ?lhs = ?rhs)

```

proof

```

assume ?lhs then show ?rhs
by (smt (verit, best) hnX separation_by_closed_intermediates_count eqpoll_iff_finite_card
eqpoll_trans)
next
assume R: ?rhs
show ?lhs
proof (cases n=0)
case True
with R show ?thesis
by fastforce
next
case False
obtain C where  $\text{closedin } X C \wedge C \subseteq S$ 
and C:  $\bigwedge D. \llbracket \text{closedin } X D; C \subseteq D; D \subseteq S \rrbracket$ 
            $\implies \exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge$ 
 $\bigcup \mathcal{U} = \text{topspace } X - D$ 

```

```

    using R by force
  then have C ⊆ topspace X
    by (simp add: closedin_subset)
  define U where U ≡ {D ∈ connected_components_of (subtopology X (topspace
  X - C)). D - S ≠ {}}
  have openU: openin X U if U ∈ U for U
    using that ⟨closedin X C⟩ lcX locally_connected_space_open_connected_components

    by (fastforce simp add: closedin_def U_def)
  have {} ∉ U
    by (auto simp: U_def)
  have pairwise_disjnt U
  using connected_components_of_disjoint by (fastforce simp add: pairwise_def
  U_def)
  show ?lhs
  proof (rule ccontr)
    assume con: ∃U. U ≈ {..<n} ∧ pairwise (separatedin X) U ∧ {} ∉ U ∧ ⋃U
  = topspace X - S
    have cardU: finite U ∧ card U < n
    proof (rule ccontr)
      assume ¬ (finite U ∧ card U < n)
      then obtain V where V ⊆ U finite V card V = n
      by (metis infinite_arbitrarily_large linorder_not_less obtain_subset_with_card_n)
      then obtain T where T ∈ V
      using False by force
      define W where W ≡ insert (topspace X - S - ⋃(V - {T})) ((λD. D
  - S) ` (V - {T}))
      have ⋃W = topspace X - S
      using ⟨∧U. U ∈ U ⟹ openin X U⟩ ⟨V ⊆ U⟩ topspace_def by (fastforce
  simp: W_def)
      moreover have {} ∉ W
      proof -
        obtain a where a ∈ T a ∉ S
          using U_def ⟨T ∈ V⟩ ⟨V ⊆ U⟩ by blast
        then have a ∈ topspace X
          using ⟨T ∈ V⟩ openU ⟨V ⊆ U⟩ openin_subset by blast
        moreover have a ∉ ⋃(V - {T})
          using diff_Union_pairwise_disjoint [of V {T}] ⟨disjoint U⟩ pair-
  wise_subset ⟨T ∈ V⟩ ⟨V ⊆ U⟩ ⟨a ∈ T⟩
          by auto
        ultimately have topspace X - S - ⋃(V - {T}) ≠ {}
          using ⟨a ∉ S⟩ by blast
        moreover have ∧V. V ∈ V - {T} ⟹ V - S ≠ {}
          using U_def ⟨V ⊆ U⟩ by blast
        ultimately show ?thesis
          by (metis (no_types, lifting) W_def image_iff insert_iff)
      qed
    moreover have disjoint V
      using ⟨V ⊆ U⟩ ⟨disjoint U⟩ pairwise_subset by blast

```

```

then have inj: inj_on ( $\lambda D. D - S$ ) ( $\mathcal{V} - \{T\}$ )
  unfolding inj_on_def using  $\langle \mathcal{V} \subseteq \mathcal{U} \rangle$  disjointD  $\mathcal{U}$ _def inf_commute by
blast
have finite  $\mathcal{W}$  card  $\mathcal{W} = n$ 
  using  $\langle \{\} \notin \mathcal{W} \rangle$   $\langle n \neq 0 \rangle$   $\langle T \in \mathcal{V} \rangle$ 
  by (auto simp add:  $\mathcal{W}$ _def  $\langle$ finite  $\mathcal{V} \rangle$  card_insert_if card_image inj  $\langle$ card
 $\mathcal{V} = n \rangle$ )
moreover have pairwise (separatedin  $X$ )  $\mathcal{W}$ 
proof -
  have disjoint  $\mathcal{W}$ 
    using  $\langle$ disjoint  $\mathcal{V} \rangle$  by (auto simp:  $\mathcal{W}$ _def pairwise_def disjnt_iff)
  have pairwise (separatedin (subtopology  $X$  (topspace  $X - S$ )))  $\mathcal{W}$ 
proof (intro pairwiseI)
  fix  $A B$ 
  assume  $\S$ :  $A \in \mathcal{W} B \in \mathcal{W} A \neq B$ 
  then have disjnt  $A B$ 
    by (meson  $\langle$ disjoint  $\mathcal{W} \rangle$  pairwiseD)
  have closedin (subtopology  $X$  (topspace  $X - C$ )) ( $\bigcup (\mathcal{V} - \{T\})$ )
    using  $\mathcal{U}$ _def  $\langle \mathcal{V} \subseteq \mathcal{U} \rangle$  closedin_connected_components_of  $\langle$ finite  $\mathcal{V} \rangle$ 
    by (force simp add: intro!: closedin_Union)
  with  $\langle C \subseteq S \rangle$  have openin (subtopology  $X$  (topspace  $X - S$ )) (topspace
 $X - S - \bigcup (\mathcal{V} - \{T\})$ )
    by (fastforce simp add: openin_closedin_eq closedin_subtopology
Int_absorb1)
  moreover have  $\bigwedge V. V \in \mathcal{V} \wedge V \neq T \implies$  openin (subtopology  $X$  (topspace
 $X - S$ )) ( $V - S$ )
    using  $\langle \mathcal{V} \subseteq \mathcal{U} \rangle$  openU
    by (metis IntD2 Int_Diff inf.orderE openin_subset openin_subtopology)

  ultimately have openin (subtopology  $X$  (topspace  $X - S$ ))  $A$  openin
(subtopology  $X$  (topspace  $X - S$ ))  $B$ 
    using  $\S$   $\mathcal{W}$ _def by blast+
  with  $\langle$ disjnt  $A B \rangle$  show separatedin (subtopology  $X$  (topspace  $X - S$ ))
 $A B$ 
    using separatedin_open_sets by blast
qed
then show ?thesis
  by (simp add: pairwise_def separatedin_subtopology)
qed
ultimately show False
  by (metis con eqpoll_iff_finite_card)
qed
obtain  $\mathcal{V}$  where  $\mathcal{V} \approx \{..<n\}$   $\{\} \notin \mathcal{V}$ 
  and pw $\mathcal{V}$ : pairwise (separatedin  $X$ )  $\mathcal{V}$  and  $UV: \bigcup \mathcal{V} =$  topspace  $X -$ 
(topspace  $X - \bigcup \mathcal{U}$ )
proof -
  have closedin  $X$  (topspace  $X - \bigcup \mathcal{U}$ )
    using openU by blast
  moreover have  $C \subseteq$  topspace  $X - \bigcup \mathcal{U}$ 

```

```

      using ⟨C ⊆ topspace X⟩ connected_components_of_subset by (fastforce
simp: U_def)
      moreover have topspace X - ⋃U ⊆ S
      using Union_connected_components_of [of subtopology X (topspace X -
C)] ⟨C ⊆ S⟩
      by (auto simp: U_def)
      ultimately show thesis
      by (metis C that)
qed
have V ≲ U
proof (rule lepoll_relational_full)
  have ⋃V = ⋃U
  by (simp add: Sup_le_iff UV_double_diff openU openin_subset)
  then show ∃ U. U ∈ U ∧ ¬ disjnt U V if V ∈ V for V
  using that
  by (metis ⟨{} ∉ V⟩ disjnt_Union1 disjnt_self_iff_empty)
  show C1 = C2
  if T ∈ U and C1 ∈ V and C2 ∈ V and ¬ disjnt T C1 and ¬ disjnt T
C2 for T C1 C2
  proof (cases C1=C2)
  case False
  then have connectedin X T
  using U_def connectedin_connected_components_of connectedin_subtopology
⟨T ∈ U⟩ by blast
  have T ⊆ C1 ∪ ⋃(V - {C1})
  using ⟨⋃V = ⋃U⟩ ⟨T ∈ U⟩ by auto
  with ⟨connectedin X T⟩
  have ¬ separatedin X C1 (⋃(V - {C1}))
  unfolding connectedin_eq_not_separated_subset
  by (smt (verit) that False disjnt_def UnionI disjnt_iff insertE insert_Diff)
  with that show ?thesis
  by (metis (no_types, lifting) ⟨V ≈ {..<n}⟩ eqpoll_iff_finite_card
finite_Diff pairwiseD pairwise_alt pwV separatedin_Union(1) separatedin_def)
  qed auto
  qed
  then show False
  by (metis ⟨V ≈ {..<n}⟩ cardU eqpoll_iff_finite_card leD lepoll_iff_card_le)
  qed
qed
qed

```

lemma *separation_by_closed_intermediates_eq_gen:*

assumes *locally_connected_space X hereditarily_normal_space X*

shows $\neg \text{connectedin } X \text{ (topspace } X - S) \longleftrightarrow$

$(\exists C. \text{closedin } X C \wedge C \subseteq S \wedge$

$(\forall D. \text{closedin } X D \wedge C \subseteq D \wedge D \subseteq S \longrightarrow \neg \text{connectedin } X \text{ (topspace$

$X - D)))$

(**is** ?lhs = ?rhs)

proof -

```

have *: ( $\exists U::'a \text{ set set. } U \approx \{..< \text{Suc } (\text{Suc } 0)\} \wedge P U$ )  $\longleftrightarrow$  ( $\exists A B. A \neq B \wedge P\{A,B\}$ ) for  $P$ 
by (metis One_nat_def eqpoll_doubleton_iff lessThan_Suc lessThan_empty_iff zero_neq_one)
have *: ( $\exists C1 C2. \text{separatedin } X C1 C2 \wedge C1 \neq C2 \wedge C1 \neq \{\} \wedge C2 \neq \{\} \wedge C1 \cup C2 = \text{topspace } X - S$ )  $\longleftrightarrow$ 
  ( $\exists C. \text{closedin } X C \wedge C \subseteq S \wedge$ 
    ( $\forall D. \text{closedin } X D \wedge C \subseteq D \wedge D \subseteq S$ 
       $\longrightarrow (\exists C1 C2. \text{separatedin } X C1 C2 \wedge C1 \neq C2 \wedge C1 \neq \{\} \wedge C2 \neq \{\})$ 
       $\wedge C1 \cup C2 = \text{topspace } X - D$ ))
using separation_by_closed_intermediates_eq_count [OF assms, of Suc(Suc 0) S]
apply (simp add: * pairwise_insert separatedin_sym cong: conj_cong)
apply (simp add: eq_sym_conv conj_ac)
done
with separatedin_refl
show ?thesis
apply (simp add: connectedin_eq_not_separated)
by (smt (verit, best) separatedin_refl)
qed

```

lemma *lepoll_connected_components_connectedin*:

```

assumes  $\bigwedge C. C \in \mathcal{U} \implies \text{connectedin } X C \cup \mathcal{U} = \text{topspace } X$ 
shows  $\text{connected\_components\_of } X \lesssim \mathcal{U}$ 
proof -
have  $\text{connected\_components\_of } X \lesssim \mathcal{U} - \{\{\}\}$ 
proof (rule lepoll_relational_full)
show  $\exists U. U \in \mathcal{U} - \{\{\}\} \wedge U \subseteq V$ 
if  $V \in \text{connected\_components\_of } X$  for  $V$ 
using that unfolding connected_components_of_def image_iff
by (metis Union_iff assms connected_component_of_maximal empty_iff
  insert_Diff_single insert_iff)
show  $V = V'$ 
if  $U \in \mathcal{U} - \{\{\}\} \wedge V \in \text{connected\_components\_of } X \wedge V' \in \text{connected\_components\_of } X$ 
 $U \subseteq V \wedge U \subseteq V'$ 
for  $U V V'$ 
by (metis DiffD2 disjointD insertCI le_inf_iff pairwise_disjoint_connected_components_of
  subset_empty that)
qed
also have  $\dots \lesssim \mathcal{U}$ 
by (simp add: subset_imp_lepoll)
finally show ?thesis .
qed

```

lemma *lepoll_connected_components_alt*:

```

 $\{..<n::\text{nat}\} \lesssim \text{connected\_components\_of } X \longleftrightarrow$ 
 $n = 0 \vee (\exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge \text{pairwise } (\text{separatedin } X) \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup \mathcal{U} =$ 

```

```

topspace X)
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof (cases n=0)
next
  case False
  show ?thesis
  proof
    assume L: ?lhs
    with False show ?rhs
    proof (induction n rule: less_induct)
      case (less n)
      show ?case
      proof (cases n $\leq$ 1)
        case True
        with less.prem1 have topspace X  $\neq$  {} n=1
          by (fastforce simp add: connected_components_of_def)+
        then have {}  $\notin$  {topspace X}
          by blast
        with <n=1> show ?thesis
          by (simp add: eqpoll_iff_finite_card card_Suc_eq flip: ex_simps)
      next
      case False
      then have n-1  $\neq$  0
        by linarith
      have n1_lespoll: {.. $n-1$ }  $\prec$  {.. $n$ }
        using False lesspoll_iff_finite_card by fastforce
      also have ...  $\lesssim$  connected_components_of X
        using less by blast
      finally have {.. $n-1$ }  $\lesssim$  connected_components_of X
        using lesspoll_imp_lespoll by blast
      then obtain  $\mathcal{U}$  where Ueq:  $\mathcal{U} \approx$  {.. $n-1$ } and {}  $\notin$   $\mathcal{U}$ 
        and pwU: pairwise (separatedin X)  $\mathcal{U}$  and UU:  $\bigcup \mathcal{U} =$  topspace X
        by (meson <n - 1  $\neq$  0> diff_less grOI less_zero_less_one)
      show ?thesis
      proof (cases  $\forall C \in \mathcal{U}. \text{connectedin } X \ C$ )
        case True
        then show ?thesis
          using lepoll_connected_components_connectedin [of  $\mathcal{U}$  X] less.prem1
          by (metis UU Ueq lepoll_antisym lepoll_trans lepoll_trans2 lesspoll_def
n1_lespoll)
        next
        case False
        with UU obtain C A B where ABC: C  $\in$   $\mathcal{U}$  A  $\cup$  B = C A  $\neq$  {} B  $\neq$ 
{} and sep: separatedin X A B
          by (fastforce simp add: connectedin_eq_not_separated)
        define  $\mathcal{V}$  where  $\mathcal{V} \equiv$  insert A (insert B ( $\mathcal{U} - \{C\}$ ))
        have  $\mathcal{V} \approx$  {.. $n$ }
        proof -
          have A  $\neq$  B

```

```

    using ‹ $B \neq \{\}$ › sep by auto
  moreover obtain  $A \notin \mathcal{U}$   $B \notin \mathcal{U}$ 
    using pwU unfolding pairwise_def
    by (metis ABC sep separatedin_Un(1) separatedin_refl separate-
din_sym)
  moreover have  $\text{card } \mathcal{U} = n-1$  finite  $\mathcal{U}$ 
    using Ueq eqpoll_iff_finite_card by blast+
  ultimately
  have  $\text{card } (\text{insert } A (\text{insert } B (\mathcal{U} - \{C\}))) = n$ 
    using ‹ $C \in \mathcal{U}$ › by (auto simp add: card_insert_if)
  then show ?thesis
    using  $\mathcal{V}_\text{def}$  ‹finite  $\mathcal{U}$ › eqpoll_iff_finite_card by blast
qed
moreover have  $\{\} \notin \mathcal{V}$ 
  using ABC  $\mathcal{V}_\text{def}$  ‹ $\{\} \notin \mathcal{U}$ › by blast
moreover have  $\bigcup \mathcal{V} = \text{topspace } X$ 
  using ABC UU  $\mathcal{V}_\text{def}$  by auto
moreover have pairwise (separatedin X)  $\mathcal{V}$ 
  using pwU sep ABC unfolding  $\mathcal{V}_\text{def}$ 
  apply (simp add: separatedin_sym pairwise_def)
  by (metis member_remove_remove_def separatedin_Un(1))
ultimately show ?thesis
  by blast
qed
qed
qed
next
  assume ?rhs
  then obtain  $\mathcal{U}$  where  $\mathcal{U} \approx \{..<n\}$   $\{\} \notin \mathcal{U}$  and pwU: pairwise (separatedin X)
  and UU:  $\bigcup \mathcal{U} = \text{topspace } X$ 
    using False by force
  have  $\text{card } (\text{connected\_components\_of } X) \geq n$  if finite (connected_components_of
X)
  proof -
    have  $\mathcal{U} \lesssim \text{connected\_components\_of } X$ 
    proof (rule lepoll_relational_full)
      show  $\exists T. T \in \text{connected\_components\_of } X \wedge \neg \text{disjnt } T C$  if  $C \in \mathcal{U}$  for
C
      by (metis that UU Union_connected_components_of Union_iff ‹ $\{\} \notin \mathcal{U}$ ›
disjnt_iff equals0I)
      show  $(C1::'a \text{ set}) = C2$ 
        if  $T \in \text{connected\_components\_of } X$  and  $C1 \in \mathcal{U}$   $C2 \in \mathcal{U}$   $\neg \text{disjnt } T C1$ 
 $\neg \text{disjnt } T C2$  for  $T C1 C2$ 
      proof (rule ccontr)
        assume  $C1 \neq C2$ 
        then have connectedin X T
          by (simp add: connectedin_connected_components_of that(1))
        moreover have  $\neg \text{separatedin } X C1 (\bigcup (\mathcal{U} - \{C1\}))$ 
          using ‹connectedin X T› pwU unfolding pairwise_def

```

```

    by (smt (verit) Sup_upper UU Union_connected_components_of ⟨C1 ≠
C2⟩ complete_lattice_class.Sup_insert_connectedin_subset_separated_union dis-
jnt_subset2 disjnt_sym insert_Diff separatedin_imp_disjoint that)
    ultimately show False
    using ⟨U ≈ {..<n}⟩
    apply (simp add: connectedin_eq_not_separated_subset eqpoll_iff_finite_card)
    by (metis Sup_upper UU finite_Diff pairwise_alt pwU separate-
din_Union(1) that(2))
    qed
    qed
    then show ?thesis
    by (metis ⟨U ≈ {..<n}⟩ eqpoll_iff_finite_card lepoll_iff_card_le that)
    qed
    then show ?lhs
    by (metis card_lessThan_finite_lepoll_infinite finite_lessThan lepoll_iff_card_le)
    qed
  qed auto

```

6.11.21 A perfect set in common cases must have at least the cardinality of the continuum

```

lemma (in Metric_space) lepoll_perfect_set:
  assumes mcomplete
  and mtopology_derived_set_of S = S S ≠ {}
  shows (UNIV::real set) ≲ S
proof -
  have S ⊆ M
  using assms(2) derived_set_of_infinite_mball by blast
  have (UNIV::real set) ≲ (UNIV::nat set set)
  using eqpoll_imp_lepoll eqpoll_sym nat_sets_eqpoll_reals by blast
  also have ... ≲ S
  proof -
    have ∃ y z δ. y ∈ S ∧ z ∈ S ∧ 0 < δ ∧ δ < ε/2 ∧
      mball y δ ⊆ mball x ε ∧ mball z δ ⊆ mball x ε ∧ disjnt (mball
y δ) (mball z δ)
    if x ∈ S 0 < ε for x ε
    proof -
      define S' where S' ≡ S ∩ mball x (ε/4)
      have infinite S'
      using derived_set_of_infinite_mball [of S] assms that S'_def
      by (smt (verit, ccfv_SIG) mem_Collect_eq zero_less_divide_iff)
      then have ∧ x y z. ¬ (S' ⊆ {x,y,z})
      using finite_subset by auto
      then obtain l r where lr: l ∈ S' r ∈ S' l ≠ r l ≠ x r ≠ x
      by (metis insert_iff subsetI)
      show ?thesis
    proof (intro exI conjI)
      show l ∈ S r ∈ S d l r / 3 > 0
      using lr by (auto simp: S'_def)
    end
  end
end

```



```

    show  $d \ l \ r / 3 < \varepsilon / 2$   $mcball \ l \ (d \ l \ r / 3) \subseteq mcball \ x \ \varepsilon \ mcball \ r \ (d \ l \ r / 3)$ 
 $\subseteq mcball \ x \ \varepsilon$ 
    using  $lr$  by (clarsimp simp:  $S'_def$ , smt (verit) commute triangle'')+
    show  $disjnt \ (mcball \ l \ (d \ l \ r / 3)) \ (mcball \ r \ (d \ l \ r / 3))$ 
    using  $lr$  by (simp add:  $S'_def \ disjnt\_iff$ ) (smt (verit, best)  $mdist\_pos\_less$ 
triangle')
    qed
  qed
  then obtain  $l \ r \ \delta$ 
    where  $lrS: \bigwedge x \ \varepsilon. \llbracket x \in S; 0 < \varepsilon \rrbracket \implies l \ x \ \varepsilon \in S \wedge r \ x \ \varepsilon \in S$ 
    and  $\delta: \bigwedge x \ \varepsilon. \llbracket x \in S; 0 < \varepsilon \rrbracket \implies 0 < \delta \ x \ \varepsilon \wedge \delta \ x \ \varepsilon < \varepsilon / 2$ 
    and  $\bigwedge x \ \varepsilon. \llbracket x \in S; 0 < \varepsilon \rrbracket \implies mcball \ (l \ x \ \varepsilon) \ (\delta \ x \ \varepsilon) \subseteq mcball \ x \ \varepsilon \wedge$ 
 $mcball \ (r \ x \ \varepsilon) \ (\delta \ x \ \varepsilon) \subseteq mcball \ x \ \varepsilon \wedge$ 
 $disjnt \ (mcball \ (l \ x \ \varepsilon) \ (\delta \ x \ \varepsilon)) \ (mcball \ (r \ x \ \varepsilon) \ (\delta \ x \ \varepsilon))$ 
    by metis
    then have  $lr\_mcball: \bigwedge x \ \varepsilon. \llbracket x \in S; 0 < \varepsilon \rrbracket \implies mcball \ (l \ x \ \varepsilon) \ (\delta \ x \ \varepsilon) \subseteq mcball$ 
 $x \ \varepsilon \wedge mcball \ (r \ x \ \varepsilon) \ (\delta \ x \ \varepsilon) \subseteq mcball \ x \ \varepsilon$ 
    and  $lr\_disjnt: \bigwedge x \ \varepsilon. \llbracket x \in S; 0 < \varepsilon \rrbracket \implies disjnt \ (mcball \ (l \ x \ \varepsilon) \ (\delta \ x \ \varepsilon))$ 
 $(mcball \ (r \ x \ \varepsilon) \ (\delta \ x \ \varepsilon))$ 
    by metis+
    obtain  $a$  where  $a \in S$ 
    using  $\langle S \neq \{\} \rangle$  by blast
    define  $xe$  where  $xe \equiv$ 
 $\lambda B. rec\_nat \ (a, 1) \ (\lambda n \ (x, \gamma). ((if \ n \in B \ then \ r \ else \ l) \ x \ \gamma, \ \delta \ x \ \gamma))$ 
    have [simp]:  $xe \ B \ 0 = (a, 1)$  for  $b$ 
    by (simp add:  $xe\_def$ )
    have  $xe \ B \ (Suc \ n) = (let \ (x, \gamma) = xe \ B \ n \ in \ ((if \ n \in B \ then \ r \ else \ l) \ x \ \gamma, \ \delta \ x \ \gamma))$ 
for  $B \ n$ 
    by (simp add:  $xe\_def$ )
    define  $x$  where  $x \equiv \lambda B \ n. fst \ (xe \ B \ n)$ 
    define  $\gamma$  where  $\gamma \equiv \lambda B \ n. snd \ (xe \ B \ n)$ 
    have [simp]:  $x \ B \ 0 = a \ \gamma \ B \ 0 = 1$  for  $B$ 
    by (simp_all add:  $x\_def \ \gamma\_def \ xe\_def$ )
    have  $x\_Suc[simp]: x \ B \ (Suc \ n) = ((if \ n \in B \ then \ r \ else \ l) \ (x \ B \ n) \ (\gamma \ B \ n))$ 
    and  $\gamma\_Suc[simp]: \gamma \ B \ (Suc \ n) = \delta \ (x \ B \ n) \ (\gamma \ B \ n)$  for  $B \ n$ 
    by (simp_all add:  $x\_def \ \gamma\_def \ xe\_def \ split: prod.split$ )
    interpret Submetric  $M \ d \ S$ 
    proof qed (use  $\langle S \subseteq M \rangle$  in metis)
    have closedin_mtopology  $S$ 
    by (metis assms(2) closure_of closure_of_eq inf.absorb_iff2 subset subset_Un_eq subset_refl topology_mtopology)
    with  $\langle mcomplete \rangle$ 
    have sub.mcomplete
    by (metis closedin_mcomplete_imp_mcomplete)
    have *:  $x \ B \ n \in S \wedge \gamma \ B \ n > 0$  for  $B \ n$ 
    by (induction  $n$ ) (auto simp:  $\langle a \in S \rangle lrS \ \delta$ )
    with subset have  $E: x \ B \ n \in M$  for  $B \ n$ 
    by blast
    have  $\gamma\_le: \gamma \ B \ n \leq (1/2)^{\wedge n}$  for  $B \ n$ 

```

```

proof(induction n)
  case 0 then show ?case by auto
next
  case (Suc n)
  then show ?case
    by simp (smt (verit) *  $\delta$  field_sum_of_halves)
qed
{ fix B
  have  $\bigwedge n. \text{sub.mcball } (x B (\text{Suc } n)) (\gamma B (\text{Suc } n)) \subseteq \text{sub.mcball } (x B n) (\gamma B$ 
n)
  by (smt (verit, best) * Int_iff  $\gamma$ _Suc  $x$ _Suc in_mono lr_mcball mcball_submetric_eq
subsetI)
  then have mon: monotone ( $\leq$ ) ( $\lambda x y. y \subseteq x$ ) ( $\lambda n. \text{sub.mcball } (x B n) (\gamma B$ 
n))
    by (simp add: decseq_SucI)
  have  $\exists n a. \text{sub.mcball } (x B n) (\gamma B n) \subseteq \text{sub.mcball } a \ \varepsilon$  if  $\varepsilon > 0$  for  $\varepsilon$ 
proof –
    obtain n where  $(1/2)^{\widehat{n}} < \varepsilon$ 
      using  $\langle 0 < \varepsilon \rangle$  real_arch_pow_inv by force
    with  $\gamma\_le$  have  $\varepsilon: \gamma B n \leq \varepsilon$ 
      by (smt (verit))
    show ?thesis
    proof (intro exI)
      show  $\text{sub.mcball } (x B n) (\gamma B n) \subseteq \text{sub.mcball } (x B n) \ \varepsilon$ 
        by (simp add:  $\varepsilon$  sub.mcball_subset_concentric)
      qed
    qed
    then have  $\exists l. l \in S \wedge (\bigcap n. \text{sub.mcball } (x B n) (\gamma B n)) = \{l\}$ 
      using  $\langle \text{sub.mcomplete} \rangle$  mon
      unfolding sub.mcomplete_nest_sing
      apply (drule_tac x= $\lambda n. \text{sub.mcball } (x B n) (\gamma B n)$  in spec)
      by (meson * order.asym sub.closedin_mcball sub.mcball_eq_empty)
    }
  then obtain z where  $z: \bigwedge B. z B \in S \wedge (\bigcap n. \text{sub.mcball } (x B n) (\gamma B n)) =$ 
 $\{z B\}$ 
    by metis
  show ?thesis
    unfolding lepoll_def
  proof (intro exI conjI)
    show inj z
    proof (rule inj_onCI)
      fix B C
      assume eq:  $z B = z C$  and  $B \neq C$ 
      then have ne:  $\text{sym\_diff } B C \neq \{\}$ 
        by blast
      define n where  $n \equiv \text{LEAST } k. k \in (\text{sym\_diff } B C)$ 
      with ne have n:  $n \in \text{sym\_diff } B C$ 
        by (metis Inf_nat_def1 LeastI)
      then have non:  $n \in B \longleftrightarrow n \notin C$ 

```

```

    by blast
  have H:  $z \in C \in \text{sub.mcball } (x \in B \text{ (Suc } n)) (\gamma \text{ } B \text{ (Suc } n)) \wedge z \in C \in \text{sub.mcball}$ 
    ( $x \in C \text{ (Suc } n)$ ) ( $\gamma \text{ } C \text{ (Suc } n)$ )
    using  $z \text{ [of } B] z \text{ [of } C]$  apply (simp add: lrS set_eq_iff non *)
    by (smt (verit, best)  $\gamma \text{\_Suc eq non } x \text{\_Suc}$ )
  have  $k \in B \longleftrightarrow k \in C$  if  $k < n$  for  $k$ 
    using that unfolding  $n \text{\_def}$  by (meson DiffI UnCI not_less_Least)
  moreover have  $(\forall m. m < p \longrightarrow (m \in B \longleftrightarrow m \in C)) \implies x \in B \text{ } p = x \in C$ 
 $p \wedge \gamma \text{ } B \text{ } p = \gamma \text{ } C \text{ } p$  for  $p$ 
    by (induction  $p$ ) auto
  ultimately have  $x \in B \text{ } n = x \in C \text{ } n \wedge B \text{ } n = \gamma \text{ } C \text{ } n$ 
    by blast+
  then show False
    using lr_disjnt * H non
    by (smt (verit) IntD2  $\gamma \text{\_Suc disjnt\_iff mcball\_submetric\_eq } x \text{\_Suc}$ )
qed
show  $\text{range } z \subseteq S$ 
  using  $z$  by blast
qed
qed
finally show ?thesis .
qed

```

lemma *lepoll_perfect_set_aux*:

assumes lcX : *locally_compact_space* X and hsX : *Hausdorff_space* X
 and eq : X *derived_set_of* *topspace* $X = \text{topspace } X$ and $\text{topspace } X \neq \{\}$
 shows $(UNIV::\text{real set}) \lesssim \text{topspace } X$

proof -

have $(UNIV::\text{real set}) \lesssim (UNIV::\text{nat set set})$
 using *eqpoll_imp_lepoll_eqpoll_sym* *nat_sets_eqpoll_reals* by blast
 also have $\dots \lesssim \text{topspace } X$

proof -

obtain z where $z: z \in \text{topspace } X$
 using *assms* by blast

then obtain $U \ K$ where *openin* $X \ U$ *compactin* $X \ K \ U \neq \{\}$ $U \subseteq K$
 by (*metis emptyE lcX locally_compact_space_def*)

then have *closedin* $X \ K$

by (*simp add: compactin_imp_closedin hsX*)

have *intK_ne*: $X \text{ interior_of } K \neq \{\}$

using $\langle U \neq \{\} \rangle \langle U \subseteq K \rangle \langle \text{openin } X \ U \rangle$ *interior_of_eq_empty* by blast

have $\exists D \ E. \text{closedin } X \ D \wedge D \subseteq K \wedge X \text{ interior_of } D \neq \{\} \wedge$
 $\text{closedin } X \ E \wedge E \subseteq K \wedge X \text{ interior_of } E \neq \{\} \wedge$
 $\text{disjnt } D \ E \wedge D \subseteq C \wedge E \subseteq C$

if *closedin* $X \ C \ C \subseteq K$ and $C: X \text{ interior_of } C \neq \{\}$ for C

proof -

obtain z where $z: z \in X \text{ interior_of } C \ z \in \text{topspace } X$
 using *C interior_of_subset_topspace* by fastforce
 obtain $x \ y$ where $x \in X \text{ interior_of } C \ y \in X \text{ interior_of } C \ x \neq y$
 by (*metis z eq in_derived_set_of openin_interior_of*)

```

then have  $x \in \text{topspace } X$   $y \in \text{topspace } X$ 
using interior_of_subset_topspace by force+
with hsX obtain  $V$   $W$  where  $\text{openin } X$   $V$   $\text{openin } X$   $W$   $x \in V$   $y \in W$  disjnt
 $V$   $W$ 
by (metis Hausdorff_space_def  $\langle x \neq y \rangle$ )
have  $*$ :  $\bigwedge W$   $x$ .  $\text{openin } X$   $W$   $\wedge x \in W$ 
 $\implies \exists U$   $V$ .  $\text{openin } X$   $U$   $\wedge \text{closedin } X$   $V$   $\wedge x \in U$   $\wedge U \subseteq V$   $\wedge V \subseteq W$ 
using lcX hsX locally_compact_Hausdorff_imp_regular_space neighbourhood_base_of_closedin neighbourhood_base_of
by metis
obtain  $M$   $D$  where  $MD$ :  $\text{openin } X$   $M$   $\text{closedin } X$   $D$   $y \in M$   $M \subseteq D$   $D \subseteq X$ 
interior_of  $C \cap W$ 
using  $*$  [of  $X$  interior_of  $C \cap W$   $y$ ]
using  $\langle \text{openin } X$   $W \rangle$   $\langle y \in W \rangle$   $\langle y \in X$  interior_of  $C \rangle$  by fastforce
obtain  $N$   $E$  where  $NE$ :  $\text{openin } X$   $N$   $\text{closedin } X$   $E$   $x \in N$   $N \subseteq E$   $E \subseteq X$ 
interior_of  $C \cap V$ 
using  $*$  [of  $X$  interior_of  $C \cap V$   $x$ ]
using  $\langle \text{openin } X$   $V \rangle$   $\langle x \in V \rangle$   $\langle x \in X$  interior_of  $C \rangle$  by fastforce
show ?thesis
proof (intro exI conjI)
show  $X$  interior_of  $D \neq \{\}$   $X$  interior_of  $E \neq \{\}$ 
using  $MD$   $NE$  by (fastforce simp: interior_of_def) $+$ 
show disjnt  $D$   $E$ 
by (meson  $MD(5)$   $NE(5)$   $\langle \text{disjnt } V$   $W \rangle$  disjnt_subset1 disjnt_sym
le_inf_iff)
qed (use  $MD$   $NE$   $\langle C \subseteq K \rangle$  interior_of_subset in force) $+$ 
qed
then obtain  $L$   $R$  where
 $LR$ :  $\bigwedge C$ .  $\llbracket \text{closedin } X$   $C$ ;  $C \subseteq K$ ;  $X$  interior_of  $C \neq \{\}$   $\rrbracket$ 
 $\implies \text{closedin } X$   $(L$   $C) \wedge (L$   $C) \subseteq K \wedge X$  interior_of  $(L$   $C) \neq \{\}$   $\wedge$ 
 $\text{closedin } X$   $(R$   $C) \wedge (R$   $C) \subseteq K \wedge X$  interior_of  $(R$   $C) \neq \{\}$ 
and disjLR:  $\bigwedge C$ .  $\llbracket \text{closedin } X$   $C$ ;  $C \subseteq K$ ;  $X$  interior_of  $C \neq \{\}$   $\rrbracket$ 
 $\implies \text{disjnt } (L$   $C) (R$   $C) \wedge (L$   $C) \subseteq C \wedge (R$   $C) \subseteq C$ 
by metis
define  $d$  where  $d \equiv \lambda B$ . rec_nat  $K$   $(\lambda n$ . if  $n \in B$  then  $R$  else  $L)$ 
have  $d0$ [simp]:  $d$   $B$   $0 = K$  for  $B$ 
by (simp add:  $d\_def$ )
have [simp]:  $d$   $B$   $(\text{Suc } n) = (\text{if } n \in B$  then  $R$  else  $L)$   $(d$   $B$   $n)$  for  $B$   $n$ 
by (simp add:  $d\_def$ )
have  $d\_correct$ :  $\text{closedin } X$   $(d$   $B$   $n) \wedge d$   $B$   $n \subseteq K \wedge X$  interior_of  $(d$   $B$   $n) \neq$ 
 $\{\}$  for  $B$   $n$ 
proof (induction  $n$ )
case  $0$ 
then show ?case by (auto simp:  $\langle \text{closedin } X$   $K \rangle$  intK_ne)
next
case  $(\text{Suc } n)$  with  $LR$  show ?case by auto
qed
have  $(\bigcap n$ .  $d$   $B$   $n) \neq \{\}$  for  $B$ 
proof (rule compact_space_imp_nest)

```

```

show compact_space (subtopology X K)
  by (simp add: ⟨compactin X K⟩ compact_space_subtopology)
show closedin (subtopology X K) (d B n) for n :: nat
  by (simp add: closedin_subset_topospace d_correct)
show d B n ≠ {} for n :: nat
  by (metis d_correct interior_of_empty)
show antimono (d B)
proof (rule antimonoI [OF transitive_stepwise_le])
  fix n
  show d B (Suc n) ⊆ d B n
  by (simp add: d_correct disjLR)
qed auto
qed
then obtain x where x: ∧B. x B ∈ (∩ n. d B n)
  unfolding set_eq_iff by (metis empty_iff)
show ?thesis
  unfolding lepoll_def
proof (intro exI conjI)
  show inj x
  proof (rule inj_onCI)
    fix B C
    assume eq: x B = x C and B≠C
    then have ne: sym_diff B C ≠ {}
      by blast
    define n where n ≡ LEAST k. k ∈ (sym_diff B C)
    with ne have n: n ∈ sym_diff B C
      by (metis Inf_nat_def1 LeastI)
    then have non: n ∈ B ⟷ n ∉ C
      by blast
    have k ∈ B ⟷ k ∈ C if k < n for k
      using that unfolding n_def by (meson DiffI UnCI not_less_Least)
    moreover have (∀ m. m < n ⟶ (m ∈ B ⟷ m ∈ C)) ⟹ d B n = d C n
  p for p
    by (induction p) auto
  ultimately have d B n = d C n
    by blast
  then have disjnt (d B (Suc n)) (d C (Suc n))
    by (simp add: d_correct disjLR disjnt_sym non)
  then show False
    by (metis InterE disjnt_iff eq_rangeI x)
  qed
  show range x ⊆ topspace X
    using x d0 ⟨compactin X K⟩ compactin_subset_topospace d_correct by
fastforce
  qed
  qed
  finally show ?thesis .
qed

```

1714

lemma *lepoll_perfect_set*:

assumes X : *completely_metrizable_space* $X \vee$ *locally_compact_space* $X \wedge$ *Hausdorff_space* X

and S : X *derived_set_of* $S = S$ $S \neq \{\}$

shows $(UNIV::real\ set) \lesssim S$

using X

proof

assume *completely_metrizable_space* X

with *assms* **show** $(UNIV::real\ set) \lesssim S$

by (*metis* *Metric_space.lepoll_perfect_set* *completely_metrizable_space_def*)

next

assume *locally_compact_space* $X \wedge$ *Hausdorff_space* X

then **show** $(UNIV::real\ set) \lesssim S$

using *lepoll_perfect_set_aux* [*of_subtopology* X S]

by (*metis* *Hausdorff_space_subtopology* S *closedin_derived_set_of* *closedin_subset* *derived_set_of_subtopology*

locally_compact_space_closed_subset *subtopology_topspace* *topspace_subtopology* *topspace_subtopology_subset*)

qed

lemma *Kuratowski_aux1*:

assumes $\bigwedge S\ T. R\ S\ T \implies R\ T\ S$

shows $(\forall S\ T\ n. R\ S\ T \implies (f\ S \approx \{..<n::nat\} \longleftrightarrow f\ T \approx \{..<n::nat\})) \longleftrightarrow$

$(\forall n\ S\ T. R\ S\ T \implies \{..<n::nat\} \lesssim f\ S \implies \{..<n::nat\} \lesssim f\ T)$

(**is** *?lhs* = *?rhs*)

proof

assume *?lhs* **then** **show** *?rhs*

by (*meson* *eqpoll_iff_finite_card* *eqpoll_sym* *finite_lepoll_infinite* *finite_lessThan* *lepoll_trans2*)

next

assume *?rhs* **then** **show** *?lhs*

by (*smt* (*verit*, *best*) *lepoll_iff_finite_card* *assms* *eqpoll_iff_finite_card* *finite_lepoll_infinite*

finite_lessThan *le_Suc_eq* *lepoll_antisym* *lepoll_iff_card_le* *not_less_eq_eq*)

qed

lemma *Kuratowski_aux2*:

pairwise (*separatedin* (*subtopology* X (*topspace* $X - S$))) $\mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge$

$\bigcup \mathcal{U} = \text{topspace}(\text{subtopology } X (\text{topspace } X - S)) \longleftrightarrow$

pairwise (*separatedin* X) $\mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup \mathcal{U} = \text{topspace } X - S$

by (*auto simp: pairwise_def* *separatedin_subtopology*)

proposition *Kuratowski_component_number_invariance_aux*:

assumes *compact_space* X **and** *HsX*: *Hausdorff_space* X

and *lcX*: *locally_connected_space* X **and** *hnX*: *hereditarily_normal_space* X

and *hom*: (*subtopology* X S) *homeomorphic_space* (*subtopology* X T)

```

and leXS: {.. $n$ ::nat}  $\lesssim$  connected_components_of (subtopology X (topspace
X - S))
assumes §:  $\bigwedge S T.$ 
   $\llbracket$ closedin X S; closedin X T; (subtopology X S) homeomorphic_space
(subtopology X T);
  {.. $n$ ::nat}  $\lesssim$  connected_components_of (subtopology X (topspace X
- S)) $\rrbracket$ 
   $\implies$  {.. $n$ ::nat}  $\lesssim$  connected_components_of (subtopology X (topspace
X - T))
shows {.. $n$ ::nat}  $\lesssim$  connected_components_of (subtopology X (topspace X -
T))
proof (cases  $n=0$ )
  case False
obtain f g where homf: homeomorphic_map (subtopology X S) (subtopology X
T) f
  and homg: homeomorphic_map (subtopology X T) (subtopology X S) g
  and gf:  $\bigwedge x. x \in \text{topspace (subtopology X S)} \implies g(f x) = x$ 
  and fg:  $\bigwedge y. y \in \text{topspace (subtopology X T)} \implies f(g y) = y$ 
  and f:  $f \in \text{topspace (subtopology X S)} \rightarrow \text{topspace (subtopology X T)}$ 
  and g:  $g \in \text{topspace (subtopology X T)} \rightarrow \text{topspace (subtopology X S)}$ 
  using homeomorphic_space_unfold hom by metis
obtain C where closedin X C  $C \subseteq S$ 
  and C:  $\bigwedge D. \llbracket$ closedin X D;  $C \subseteq D$ ;  $D \subseteq S$  $\rrbracket$ 
     $\implies \exists \mathcal{U}. \mathcal{U} \approx \{.. $n$ \} \wedge \text{pairwise (separatedin X) } \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup \mathcal{U} =$ 
topspace X - D
  using separation_by_closed_intermediates_eq_count [of X n S] assms
by (smt (verit, ccfv_threshold) False Kuratowski_aux2 lepoll_connected_components_alt)
have  $\exists C. \text{closedin X } C \wedge C \subseteq T \wedge$ 
   $(\forall D. \text{closedin X } D \wedge C \subseteq D \wedge D \subseteq T$ 
   $\longrightarrow (\exists \mathcal{U}. \mathcal{U} \approx \{.. $n$ \} \wedge \text{pairwise (separatedin X) } \mathcal{U} \wedge$ 
   $\{\} \notin \mathcal{U} \wedge \bigcup \mathcal{U} = \text{topspace X - D}))$ 
proof (intro exI, intro conjI strip)
  have compactin X (f ' C)
  by (meson  $\langle C \subseteq S \rangle \langle \text{closedin X } C \rangle$  assms(1) closedin_compact_space com-
compactin_subtopology homeomorphic_map_compactness_eq homf)
  then show closedin X (f ' C)
  using  $\langle \text{Hausdorff\_space X} \rangle$  compactin_imp_closedin by blast
  show f ' C  $\subseteq T$ 
  by (meson  $\langle C \subseteq S \rangle \langle \text{closedin X } C \rangle$  closedin_imp_subset closedin_subset_topspace
homeomorphic_map_closedness_eq homf)
  fix D'
  assume D': closedin X D'  $\wedge f ' C \subseteq D' \wedge D' \subseteq T$ 
  define D where D  $\equiv g ' D'$ 
  have compactin X D
  unfolding D_def
  by (meson D'  $\langle \text{compact\_space X} \rangle$  closedin_compact_space compactin_subtopology
homeomorphic_map_compactness_eq homg)
  then have closedin X D
  by (simp add: assms(2) compactin_imp_closedin)

```

moreover have $C \subseteq D$
using $D' D_def \langle C \subseteq S \rangle \langle closedin X C \rangle closedin_subset\ gf\ image_iff$ **by**
fastforce
moreover have $D \subseteq S$
by (*metis* $D' D_def\ assms(1)\ closedin_compact_space\ compactin_subtopology$
homeomorphic_map_compactness_eq\ homg)
ultimately obtain \mathcal{U} **where** $\mathcal{U} \approx \{..<n\}$ *pairwise* (*separatedin* X) $\mathcal{U} \wedge \{\} \notin \mathcal{U}$
 $\bigcup \mathcal{U} = \text{topspace } X - D$
using C **by** *meson*
moreover have (*subtopology* $X D$) *homeomorphic_space* (*subtopology* $X D'$)
unfolding *homeomorphic_space_def*
proof (*intro exI*)
have *subtopology* $X D = \text{subtopology} (\text{subtopology } X S) D$
by (*simp add:* $\langle D \subseteq S \rangle \text{inf.absorb2 } \text{subtopology_subtopology}$)
moreover have *subtopology* $X D' = \text{subtopology} (\text{subtopology } X T) D'$
by (*simp add:* $D' \text{inf.absorb2 } \text{subtopology_subtopology}$)
moreover have *homeomorphic_maps* (*subtopology* $X T$) (*subtopology* $X S$) g
f
by (*simp add:* $fg\ gf\ \text{homeomorphic_maps_map}\ homf\ homg$)
ultimately
have *homeomorphic_maps* (*subtopology* $X D'$) (*subtopology* $X D$) $g\ f$
by (*metis* $D' D_def \langle closedin X D \rangle closedin_subset\ \text{homeomorphic_maps_subtopologies}$
topspace_subtopology\ Int_absorb1)
then show *homeomorphic_maps* (*subtopology* $X D$) (*subtopology* $X D'$) $f\ g$
using *homeomorphic_maps_sym* **by** *blast*
qed
ultimately show $\exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge \text{pairwise} (\text{separatedin } X) \mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge$
 $\bigcup \mathcal{U} = \text{topspace } X - D'$
by (*smt* (*verit*, *ccfv_SIG*) § $D' \text{False} \langle closedin X D \rangle \text{Kuratowski_aux2}\ \text{lep-}$
oll_connected_components_alt)
qed
then have $\exists \mathcal{U}. \mathcal{U} \approx \{..<n\} \wedge$
pairwise (*separatedin* (*subtopology* $X (\text{topspace } X - T)$)) $\mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge$
 $\bigcup \mathcal{U} = \text{topspace } X - T$
using *separation_by_closed_intermediates_eq_count* [*of* $X\ n\ T$] *Kuratowski_aux2*
lcX hnX **by** *auto*
with False **show** *?thesis*
using *lepoll_connected_components_alt* **by** *fastforce*
qed *auto*

theorem *Kuratowski_component_number_invariance:*

assumes *compact_space* X *Hausdorff_space* X *locally_connected_space* X *hereditarily_normal_space* X

shows $(\forall S\ T\ n.$

closedin $X S \wedge \text{closedin } X T \wedge$

(subtopology $X S)$ *homeomorphic_space* (*subtopology* $X T$)

\longrightarrow (*connected_components_of*

(subtopology $X (\text{topspace } X - S)) \approx \{..<n::nat\} \longleftrightarrow$


```

      connected_components_of
      (subtopology X (topspace X - T)) ≈ {..n::nat})) ↔
    (∀ S T n.
      (subtopology X S) homeomorphic_space (subtopology X T)
      → (connected_components_of
          (subtopology X (topspace X - S)) ≈ {..n::nat} ↔
          connected_components_of
          (subtopology X (topspace X - T)) ≈ {..n::nat})))
    (is ?lhs = ?rhs)
  proof
    assume L: ?lhs
    then show ?rhs
      apply (subst (asm) Kuratowski_aux1, use homeomorphic_space_sym in blast)
      apply (subst Kuratowski_aux1, use homeomorphic_space_sym in blast)
      apply (blast intro: Kuratowski_component_number_invariance_aux assms)
      done
  qed blast

end
theory Isolated
  imports HOL-Analysis.Elementary_Metric_Spaces

begin

6.11.22 Isolate and discrete

definition (in topological_space) isolated_in:: 'a ⇒ 'a set ⇒ bool (infixr isolated' in 60)
  where x isolated_in S ↔ (x ∈ S ∧ (∃ T. open T ∧ T ∩ S = {x}))

definition (in topological_space) discrete:: 'a set ⇒ bool
  where discrete S ↔ (∀ x ∈ S. x isolated_in S)

definition (in metric_space) uniform_discrete :: 'a set ⇒ bool where
  uniform_discrete S ↔ (∃ e > 0. ∀ x ∈ S. ∀ y ∈ S. dist x y < e → x = y)

lemma discreteI: (∧ x. x ∈ X ⇒ x isolated_in X) ⇒ discrete X
  unfolding discrete_def by auto

lemma discreteD: discrete X ⇒ x ∈ X ⇒ x isolated_in X
  unfolding discrete_def by auto

lemma uniformI1:
  assumes e > 0 ∧ x y. [[x ∈ S; y ∈ S; dist x y < e] ⇒ x = y]
  shows uniform_discrete S
  unfolding uniform_discrete_def using assms by auto

lemma uniformI2:
  assumes e > 0 ∧ x y. [[x ∈ S; y ∈ S; x ≠ y] ⇒ dist x y ≥ e]

```

1718

shows *uniform_discrete* *S*
unfolding *uniform_discrete_def* **using** *assms not_less* **by** *blast*

lemma *isolated_in_islimpt_iff*: $(x \text{ isolated_in } S) \longleftrightarrow (\neg (x \text{ islimpt } S) \wedge x \in S)$
unfolding *isolated_in_def islimpt_def* **by** *auto*

lemma *isolated_in_dist_Ex_iff*:
fixes $x :: 'a :: \text{metric_space}$
shows $x \text{ isolated_in } S \longleftrightarrow (x \in S \wedge (\exists e > 0. \forall y \in S. \text{dist } x \ y < e \longrightarrow y = x))$
unfolding *isolated_in_islimpt_iff islimpt_approachable* **by** (*metis dist_commute*)

lemma *discrete_empty[simp]*: *discrete* $\{\}$
unfolding *discrete_def* **by** *auto*

lemma *uniform_discrete_empty[simp]*: *uniform_discrete* $\{\}$
unfolding *uniform_discrete_def* **by** (*simp add: gt_ex*)

lemma *isolated_in_insert*:
fixes $x :: 'a :: \text{t1_space}$
shows $x \text{ isolated_in } (\text{insert } a \ S) \longleftrightarrow x \text{ isolated_in } S \vee (x = a \wedge \neg (x \text{ islimpt } S))$
by (*meson insert_iff islimpt_insert isolated_in_islimpt_iff*)

lemma *isolated_inI*:
assumes $x \in S$ *open* T $T \cap S = \{x\}$
shows $x \text{ isolated_in } S$
using *assms* **unfolding** *isolated_in_def* **by** *auto*

lemma *isolated_inE*:
assumes $x \text{ isolated_in } S$
obtains T **where** $x \in S$ *open* T $T \cap S = \{x\}$
using *assms* **that** **unfolding** *isolated_in_def* **by** *force*

lemma *isolated_inE_dist*:
assumes $x \text{ isolated_in } S$
obtains d **where** $d > 0 \wedge y. y \in S \implies \text{dist } x \ y < d \implies y = x$
by (*meson assms isolated_in_dist_Ex_iff*)

lemma *isolated_in_altdef*:
 $x \text{ isolated_in } S \longleftrightarrow (x \in S \wedge \text{eventually } (\lambda y. y \notin S) \text{ (at } x))$

proof
assume $x \text{ isolated_in } S$
from *isolated_inE* [*OF this*]
obtain T **where** $x \in S$ **and** T : *open* T $T \cap S = \{x\}$
by *metis*
have $\forall_F y$ *in nhds* $x. y \in T$
apply (*rule eventually_nhds_in_open*)
using T **by** *auto*
then have *eventually* $(\lambda y. y \in T - \{x\}) \text{ (at } x)$
unfolding *eventually_at_filter* **by** *eventually_elim auto*

```

then have eventually ( $\lambda y. y \notin S$ ) (at  $x$ )
  by eventually_elim (use  $T$  in auto)
then show  $x \in S \wedge (\forall_F y \text{ in at } x. y \notin S)$  using  $\langle x \in S \rangle$  by auto
next
assume  $x \in S \wedge (\forall_F y \text{ in at } x. y \notin S)$ 
then have  $\forall_F y \text{ in at } x. y \notin S$   $x \in S$  by auto
from this(1) have eventually ( $\lambda y. y \notin S \vee y = x$ ) (nhds  $x$ )
  unfolding eventually_at_filter by eventually_elim auto
then obtain  $T$  where  $T:\text{open } T$   $x \in T$   $(\forall y \in T. y \notin S \vee y = x)$ 
  unfolding eventually_nhds by auto
with  $\langle x \in S \rangle$  have  $T \cap S = \{x\}$ 
  by fastforce
with  $\langle x \in S \rangle$   $\langle \text{open } T \rangle$ 
show  $x$  isolated_in  $S$ 
  unfolding isolated_in_def by auto
qed

```

```

lemma discrete_altdef:
  discrete  $S \iff (\forall x \in S. \forall_F y \text{ in at } x. y \notin S)$ 
  unfolding discrete_def isolated_in_altdef by auto

```

```

lemma uniform_discrete_imp_closed:
  uniform_discrete  $S \implies$  closed  $S$ 
  by (meson discrete_imp_closed uniform_discrete_def)

```

```

lemma uniform_discrete_imp_discrete:
  uniform_discrete  $S \implies$  discrete  $S$ 
  by (metis discrete_def isolated_in_dist_Ex_iff uniform_discrete_def)

```

```

lemma isolated_in_subset:  $x$  isolated_in  $S \implies T \subseteq S \implies x \in T \implies x$  isolated_in  $T$ 
  unfolding isolated_in_def by fastforce

```

```

lemma discrete_subset[elim]: discrete  $S \implies T \subseteq S \implies$  discrete  $T$ 
  unfolding discrete_def using islimpt_subset isolated_in_islimpt_iff by blast

```

```

lemma uniform_discrete_subset[elim]: uniform_discrete  $S \implies T \subseteq S \implies$  uniform_discrete  $T$ 
  by (meson subsetD uniform_discrete_def)

```

```

lemma continuous_on_discrete: discrete  $S \implies$  continuous_on  $S$   $f$ 
  unfolding continuous_on_topological by (metis discrete_def islimptI isolated_in_islimpt_iff)

```

```

lemma uniform_discrete_insert: uniform_discrete (insert  $a$   $S$ )  $\iff$  uniform_discrete  $S$ 

```

```

proof
  assume asm: uniform_discrete  $S$ 

```

1720

```
let ?thesis = uniform_discrete (insert a S)
have ?thesis when a ∈ S using that asm by (simp add: insert_absorb)
moreover have ?thesis when S = {} using that asm by (simp add: uniform_discrete_def)
moreover have ?thesis when a ∉ S S ≠ {}
proof -
  obtain e1 where e1 > 0 and e1_dist: ∀ x ∈ S. ∀ y ∈ S. dist y x < e1 → y = x
  using asm unfolding uniform_discrete_def by auto
  define e2 where e2 ≡ min (setdist {a} S) e1
  have closed S using asm uniform_discrete_imp_closed by auto
  then have e2 > 0
  by (smt (verit) ⟨0 < e1⟩ e2_def infdist_eq_setdist infdist_pos_not_in_closed
that)
  moreover have x = y if x ∈ insert a S y ∈ insert a S dist x y < e2 for x y
  proof (cases x = a ∨ y = a)
    case True then show ?thesis
      by (smt (verit, best) dist_commute e2_def infdist_eq_setdist infdist_le
insertE that)
    next
      case False then show ?thesis
        using e1_dist e2_def that by force
  qed
  ultimately show ?thesis unfolding uniform_discrete_def by meson
qed
ultimately show ?thesis by auto
qed (simp add: subset_insertI uniform_discrete_subset)

lemma discrete_compact_finite_iff:
  fixes S :: 'a::t1_space set
  shows discrete S ∧ compact S ↔ finite S
proof
  assume finite S
  then have compact S using finite_imp_compact by auto
  moreover have discrete S
  unfolding discrete_def using isolated_in_islimpt_iff islimpt_finite[OF ⟨finite
S⟩] by auto
  ultimately show discrete S ∧ compact S by auto
next
  assume discrete S ∧ compact S
  then show finite S
  by (meson discrete_def Heine_Borel_imp_Bolzano_Weierstrass isolated_in_islimpt_iff
order_refl)
qed

lemma uniform_discrete_finite_iff:
  fixes S :: 'a::heine_borel set
  shows uniform_discrete S ∧ bounded S ↔ finite S
proof
  assume uniform_discrete S ∧ bounded S
  then have discrete S compact S
```

```

    using uniform_discrete_imp_discrete uniform_discrete_imp_closed compact_eq_bounded_closed
    by auto
  then show finite S using discrete_compact_finite_iff by auto
next
assume asm:finite S
let ?thesis = uniform_discrete S ∧ bounded S
have ?thesis when S={ } using that by auto
moreover have ?thesis when S≠{ }
proof -
  have  $\forall x. \exists d>0. \forall y \in S. y \neq x \longrightarrow d \leq \text{dist } x \ y$ 
    using finite_set_avoid[OF ⟨finite S⟩] by auto
  then obtain f where f_pos:  $f \ x > 0$ 
    and f_dist:  $\forall y \in S. y \neq x \longrightarrow f \ x \leq \text{dist } x \ y$ 
    if  $x \in S$  for x
    by metis
  define f_min where f_min  $\equiv \text{Min } (f \ ` \ S)$ 
  have f_min > 0
    unfolding f_min_def
    by (simp add: asm f_pos that)
  moreover have  $\forall x \in S. \forall y \in S. f\_min > \text{dist } x \ y \longrightarrow x=y$ 
    using f_dist unfolding f_min_def
    by (metis Min_le asm finite_imageI imageI le_less_trans linorder_not_less)
  ultimately have uniform_discrete S
    unfolding uniform_discrete_def by auto
  moreover have bounded S using ⟨finite S⟩ by auto
  ultimately show ?thesis by auto
qed
ultimately show ?thesis by blast
qed

lemma uniform_discrete_image_scale:
  assumes uniform_discrete S and dist:  $\forall x \in S. \forall y \in S. \text{dist } x \ y = c * \text{dist } (f \ x) \ (f \ y)$ 
  shows uniform_discrete (f ` S)
proof -
  have ?thesis when S={ } using that by auto
  moreover have ?thesis when S≠{ }  $c \leq 0$ 
  proof -
    obtain x1 where x1  $\in S$  using ⟨S≠{ }⟩ by auto
    have ?thesis when  $S - \{x1\} = \{ \}$ 
      using ⟨x1  $\in S$ ⟩ subset_antisym that uniform_discrete_insert by fastforce
    moreover have ?thesis when  $S - \{x1\} \neq \{ \}$ 
    proof -
      obtain x2 where x2  $\in S - \{x1\}$  using ⟨S - {x1} ≠ { }⟩ by auto
      then have x2  $\in S$  x1 ≠ x2 by auto
      then have dist x1 x2 > 0 by auto
      moreover have dist x1 x2 = c * dist (f x1) (f x2)
        by (simp add: ⟨x1  $\in S$ ⟩ ⟨x2  $\in S$ ⟩ dist)
      moreover have dist (f x2) (f x2)  $\geq 0$  by auto
    qed
  qed

```

```

    ultimately have False using <c>0> by (simp add: zero_less_mult_iff)
    then show ?thesis by auto
  qed
  ultimately show ?thesis by auto
  qed
  moreover have ?thesis when  $S \neq \{\}$   $c > 0$ 
  proof -
    obtain  $e1$  where  $e1 > 0$  and  $e1\_dist: \forall x \in S. \forall y \in S. dist\ y\ x < e1 \implies y = x$ 
      using <uniform_discrete S> unfolding uniform_discrete_def by auto
    define  $e$  where  $e \equiv e1 / c$ 
    have  $x1 = x2$  when  $x1 \in f\ S\ x2 \in f\ S$  and  $d: dist\ x1\ x2 < e$  for  $x1\ x2$ 
      by (smt (verit) <0 < c> d dist divide_right_mono e1_dist e_def imageE
nonzero_mult_div_cancel_left that)
    moreover have  $e > 0$  using <e1 > 0> <c > 0> unfolding e_def by auto
    ultimately show ?thesis unfolding uniform_discrete_def by meson
  qed
  ultimately show ?thesis by fastforce
  qed

definition sparse :: real  $\Rightarrow$  'a :: metric_space set  $\Rightarrow$  bool
  where sparse  $\varepsilon\ X \iff (\forall x \in X. \forall y \in X - \{x\}. dist\ x\ y > \varepsilon)$ 

lemma sparse_empty [simp, intro]: sparse  $\varepsilon\ \{\}$ 
  by (auto simp: sparse_def)

lemma sparseI [intro?]:
   $(\bigwedge x\ y. x \in X \implies y \in X \implies x \neq y \implies dist\ x\ y > \varepsilon) \implies sparse\ \varepsilon\ X$ 
  unfolding sparse_def by auto

lemma sparseD:
  sparse  $\varepsilon\ X \implies x \in X \implies y \in X \implies x \neq y \implies dist\ x\ y > \varepsilon$ 
  unfolding sparse_def by auto

lemma sparseD':
  sparse  $\varepsilon\ X \implies x \in X \implies y \in X \implies dist\ x\ y \leq \varepsilon \implies x = y$ 
  unfolding sparse_def by force

lemma sparse_singleton [simp, intro]: sparse  $\varepsilon\ \{x\}$ 
  by (auto simp: sparse_def)

definition setdist_gt where setdist_gt  $\varepsilon\ X\ Y \iff (\forall x \in X. \forall y \in Y. dist\ x\ y > \varepsilon)$ 

lemma setdist_gt_empty [simp]: setdist_gt  $\varepsilon\ \{\}\ Y\ setdist\_gt\ \varepsilon\ X\ \{\}$ 
  by (auto simp: setdist_gt_def)

lemma setdist_gtI:  $(\bigwedge x\ y. x \in X \implies y \in Y \implies dist\ x\ y > \varepsilon) \implies setdist\_gt\ \varepsilon\ X\ Y$ 
  unfolding setdist_gt_def by auto

```

lemma *setdist_gtD*: $\text{setdist_gt } \varepsilon X Y \implies x \in X \implies y \in Y \implies \text{dist } x y > \varepsilon$
unfolding *setdist_gt_def* **by** *auto*

lemma *setdist_gt_setdist*: $\varepsilon < \text{setdist } A B \implies \text{setdist_gt } \varepsilon A B$
unfolding *setdist_gt_def* **using** *setdist_le_dist* **by** *fastforce*

lemma *setdist_gt_mono*: $\text{setdist_gt } \varepsilon' A B \implies \varepsilon \leq \varepsilon' \implies A' \subseteq A \implies B' \subseteq B \implies \text{setdist_gt } \varepsilon A' B'$
by (*force simp: setdist_gt_def*)

lemma *setdist_gt_Un_left*: $\text{setdist_gt } \varepsilon (A \cup B) C \longleftrightarrow \text{setdist_gt } \varepsilon A C \wedge \text{setdist_gt } \varepsilon B C$
by (*auto simp: setdist_gt_def*)

lemma *setdist_gt_Un_right*: $\text{setdist_gt } \varepsilon C (A \cup B) \longleftrightarrow \text{setdist_gt } \varepsilon C A \wedge \text{setdist_gt } \varepsilon C B$
by (*auto simp: setdist_gt_def*)

lemma *compact_closed_imp_eventually_setdist_gt_at_right_0*:
assumes *compact A closed B A ∩ B = {}*
shows *eventually* $(\lambda \varepsilon. \text{setdist_gt } \varepsilon A B)$ $(\text{at_right } 0)$
proof (*cases A = {} ∨ B = {}*)
case *False*
hence $\text{setdist } A B > 0$
by (*metis IntI assms empty_iff in_closed_iff infdist_zero order_less_le setdist_attains_inf setdist_pos_le setdist_sym*)
hence *eventually* $(\lambda \varepsilon. \varepsilon < \text{setdist } A B)$ $(\text{at_right } 0)$
using *eventually_at_right_field* **by** *blast*
thus *?thesis*
by *eventually_elim* (*auto intro: setdist_gt_setdist*)
qed *auto*

lemma *setdist_gt_symI*: $\text{setdist_gt } \varepsilon A B \implies \text{setdist_gt } \varepsilon B A$
by (*force simp: setdist_gt_def dist_commute*)

lemma *setdist_gt_sym*: $\text{setdist_gt } \varepsilon A B \longleftrightarrow \text{setdist_gt } \varepsilon B A$
by (*force simp: setdist_gt_def dist_commute*)

lemma *eventually_setdist_gt_at_right_0_mult_iff*:
assumes $c > 0$
shows *eventually* $(\lambda \varepsilon. \text{setdist_gt } (c * \varepsilon) A B)$ $(\text{at_right } 0) \longleftrightarrow$
eventually $(\lambda \varepsilon. \text{setdist_gt } \varepsilon A B)$ $(\text{at_right } 0)$
proof –
have *eventually* $(\lambda \varepsilon. \text{setdist_gt } (c * \varepsilon) A B)$ $(\text{at_right } 0) \longleftrightarrow$
eventually $(\lambda \varepsilon. \text{setdist_gt } \varepsilon A B)$ $(\text{filtermap } ((*) c) (\text{at_right } 0))$
by (*simp add: eventually_filtermap*)
also **have** $\text{filtermap } ((*) c) (\text{at_right } 0) = \text{at_right } 0$
by (*subst filtermap_times_pos_at_right*) (*use assms in auto*)
finally **show** *?thesis* .

1724

qed

end

theory *Sparse_In*
 imports *Homotopy*

begin

6.11.23 A set of points sparse in another set

definition *sparse_in*:: 'a :: topological_space set \Rightarrow 'a set \Rightarrow bool
 (infixl (*sparse'_in*) 50)

where

$pts\ sparse_in\ A = (\forall x \in A. \exists B. x \in B \wedge open\ B \wedge (\forall y \in B. \neg y\ islimpt\ pts))$

lemma *sparse_in_empty[simp]*: $\{\}$ *sparse_in* A

by (meson UNIV_I empty_iff islimpt_def open_UNIV sparse_in_def)

lemma *finite_imp_sparse*:

fixes *pts*::'a:: t1_space set

shows *finite pts* \implies *pts sparse_in* S

by (meson UNIV_I islimpt_finite open_UNIV sparse_in_def)

lemma *sparse_in_singleton[simp]*: $\{x\}$ *sparse_in* (A::'a:: t1_space set)

by (rule *finite_imp_sparse*) auto

lemma *sparse_in_ball_def*:

$pts\ sparse_in\ D \iff (\forall x \in D. \exists e > 0. \forall y \in ball\ x\ e. \neg y\ islimpt\ pts)$

unfolding *sparse_in_def*

by (meson Elementary_Metric_Spaces.open_ball open_contains_ball_eq subset_eq)

lemma *get_sparse_in_cover*:

assumes *pts sparse_in* A

obtains B where *open B* $A \subseteq B$ $\forall y \in B. \neg y\ islimpt\ pts$

proof –

obtain *getB* where *getB*: $x \in getB\ x\ open\ (getB\ x) \forall y \in getB\ x. \neg y\ islimpt\ pts$
 if $x \in A$ for *x*

using *assms(1)* **unfolding** *sparse_in_def* **by** *metis*

define B where $B = Union\ (getB\ ` A)$

have *open B* **unfolding** *B_def* **using** *getB(2)* **by** *blast*

moreover **have** $A \subseteq B$ **unfolding** *B_def* **using** *getB(1)* **by** *auto*

moreover **have** $\forall y \in B. \neg y\ islimpt\ pts$ **unfolding** *B_def* **by** (meson UN_iff *getB(3)*)

ultimately **show** *?thesis* **using** *that* **by** *blast*

qed

lemma *sparse_in_open*:

assumes *open* A

shows pts *sparse_in* $A \iff (\forall y \in A. \neg y \text{ islimpt } pts)$
using *assms unfolding sparse_in_def by auto*

lemma *sparse_in_not_in*:
assumes pts *sparse_in* A $x \in A$
obtains B **where** *open* B $x \in B \forall y \in B. y \neq x \implies y \notin pts$
using *assms unfolding sparse_in_def*
by (*metis islimptI*)

lemma *sparse_in_subset*:
assumes pts *sparse_in* A $B \subseteq A$
shows pts *sparse_in* B
using *assms unfolding sparse_in_def by auto*

lemma *sparse_in_subset2*:
assumes $pts1$ *sparse_in* D $pts2 \subseteq pts1$
shows $pts2$ *sparse_in* D
by (*meson assms(1) assms(2) islimpt_subset sparse_in_def*)

lemma *sparse_in_union*:
assumes $pts1$ *sparse_in* $D1$ $pts2$ *sparse_in* $D1$
shows $(pts1 \cup pts2)$ *sparse_in* $(D1 \cap D2)$
using *assms unfolding sparse_in_def islimpt_Un*
by (*metis Int_iff open_Int*)

lemma *sparse_in_compact_finite*:
assumes pts *sparse_in* A *compact* A
shows *finite* $(A \cap pts)$
apply (*rule finite_not_islimpt_in_compact[OF <compact A>]*)
using *assms unfolding sparse_in_def by blast*

lemma *sparse_imp_closedin_pts*:
assumes pts *sparse_in* D
shows *closedin* (*top_of_set* D) $(D \cap pts)$
using *assms islimpt_subset unfolding closedin_limpt sparse_in_def*
by *fastforce*

lemma *open_diff_sparse_pts*:
assumes *open* D pts *sparse_in* D
shows *open* $(D - pts)$
using *assms sparse_imp_closedin_pts*
by (*metis Diff_Diff_Int Diff_cancel Diff_eq_empty_iff Diff_subset*
closedin_def double_diff openin_open_eq topspace_euclidean_subtopology)

lemma *sparse_imp_countable*:
fixes $D::'a :: euclidean_space$ *set*
assumes *open* D pts *sparse_in* D
shows *countable* $(D \cap pts)$

proof –

```

obtain  $K :: \text{nat} \Rightarrow 'a :: \text{euclidean\_space set}$ 
  where  $K: D = (\bigcup n. K\ n) \wedge n. \text{compact}\ (K\ n)$ 
  using assms by (metis open_Union_compact_subsets)
then have  $D \cap \text{pts} = (\bigcup n. K\ n \cap \text{pts})$ 
  by blast
moreover have  $\bigwedge n. \text{finite}\ (K\ n \cap \text{pts})$ 
  by (metis  $K(1)\ K(2)\ \text{Union\_iff}\ \text{assms}(2)\ \text{rangeI}$ 
    sparse_in_compact_finite sparse_in_subset subsetI)
ultimately show ?thesis
  by (metis countableI_type countable_UN countable_finite)
qed

```

```

lemma sparse_imp_connected:
  fixes  $D::'a :: \text{euclidean\_space set}$ 
  assumes  $2 \leq \text{DIM}\ ('a)$  connected D open D pts sparse_in D
  shows connected (D - pts)
  using assms
  by (metis Diff_Cmpl Diff_Diff_Int Diff_eq connected_open_diff_countable
    sparse_imp_countable)

```

```

lemma sparse_in_eventually_iff:
  assumes open A
  shows  $\text{pts sparse\_in}\ A \longleftrightarrow (\forall y \in A. (\forall_F\ y\ \text{in}\ \text{at}\ y. y \notin \text{pts}))$ 
  unfolding sparse_in_open[OF <open A>] islimpt_iff_eventually
  by simp

```

```

lemma get_sparse_from_eventually:
  fixes  $A::'a::\text{topological\_space set}$ 
  assumes  $\forall x \in A. \forall_F\ z\ \text{in}\ \text{at}\ x. P\ z\ \text{open}\ A$ 
  obtains pts where  $\text{pts sparse\_in}\ A \wedge \forall x \in A - \text{pts}. P\ x$ 
  proof –
  define  $\text{pts}::'a\ \text{set}$  where  $\text{pts} = \{x. \neg P\ x\}$ 
  have  $\text{pts sparse\_in}\ A \wedge \forall x \in A - \text{pts}. P\ x$ 
  unfolding sparse_in_eventually_iff[OF <open A>] pts_def
  using assms(1) by simp_all
  then show ?thesis using that by blast
qed

```

```

lemma sparse_disjoint:
  assumes  $\text{pts} \cap A = \{\}$  open A
  shows pts sparse_in A
  using assms unfolding sparse_in_eventually_iff[OF <open A>]
    eventually_at_topological
  by blast

```

6.11.24 Co-sparseness filter

The co-sparseness filter allows us to talk about properties that hold on a given set except for an “insignificant” number of points that are sparse in

that set.

lemma *is_filter_cosparse*: *is_filter* ($\lambda P. \{x. \neg P x\}$ *sparse_in A*)

proof (*standard, goal_cases*)

case 1

thus ?*case* **by** *auto*

next

case (2 *P Q*)

from *sparse_in_union*[*OF this, of UNIV*] **show** ?*case*

by (*auto simp: Un_def*)

next

case (3 *P Q*)

from 3(2) **show** ?*case*

by (*rule sparse_in_subset2*) (*use 3(1) in auto*)

qed

definition *cosparse* :: 'a set \Rightarrow 'a :: *topological_space filter* **where**

cosparse A = Abs_filter ($\lambda P. \{x. \neg P x\}$ *sparse_in A*)

syntax

_eventually_cosparse :: *pttrn* \Rightarrow 'a set \Rightarrow bool \Rightarrow bool (($\exists \approx _ \in _ / _$) [0, 0, 10] 10)

translations

$\forall \approx x \in A. P == \text{CONST eventually } (\lambda x. P) (\text{CONST cosparse } A)$

syntax

_qeventually_cosparse :: *pttrn* \Rightarrow bool \Rightarrow 'a \Rightarrow 'a (($\exists \approx _ | (_) / _$) [0, 0, 10] 10)

translations

$\forall \approx x | P. t \Rightarrow \text{CONST eventually } (\lambda x. t) (\text{CONST cosparse } \{x. P\})$

print_translation <

let

fun *ev_cosparse_tr'* [*Abs* (*x, Tx, t*),

Const (**const_syntax** <*cosparse*>, *_*) \$ (*Const* (**const_syntax** <*Collect*>, *_*) \$ *Abs* (*y, Ty, P*))] =

if *x* <> *y* *then raise Match*

else

let

val *x'* = *Syntax_Trans.mark_bound_body* (*x, Tx*);

val *t'* = *subst_bound* (*x', t*);

val *P'* = *subst_bound* (*x', P*);

in

Syntax.const **syntax_const** <*_qeventually_cosparse*> \$

Syntax_Trans.mark_bound_abs (*x, Tx*) \$ *P'* \$ *t'*

end

 | *ev_cosparse_tr' _* = *raise Match*;

in [(**const_syntax** <*eventually*>, *K ev_cosparse_tr'*)] *end*

>

1728

lemma *eventually_cosparse*: $\text{eventually } P \text{ (cosparse } A) \longleftrightarrow \{x. \neg P x\} \text{ sparse_in } A$

unfolding *cosparse_def* **by** (*rule eventually_Abs_filter[OF is_filter_cosparse]*)

lemma *eventually_not_in_cosparse*:

assumes *X sparse_in A*

shows $\text{eventually } (\lambda x. x \notin X) \text{ (cosparse } A)$

using *assms* **by** (*auto simp: eventually_cosparse*)

lemma *eventually_cosparse_open_eq*:

$\text{open } A \implies \text{eventually } P \text{ (cosparse } A) \longleftrightarrow (\forall x \in A. \text{eventually } P \text{ (at } x))$

unfolding *eventually_cosparse*

by (*subst sparse_in_open*) (*auto simp: islimpt_conv_frequently_at frequently_def*)

lemma *eventually_cosparse_imp_eventually_at*:

$\text{eventually } P \text{ (cosparse } A) \implies x \in A \implies \text{eventually } P \text{ (at } x \text{ within } B)$

unfolding *eventually_cosparse sparse_in_def*

apply (*auto simp: islimpt_conv_frequently_at frequently_def*)

apply (*metis UNIV_I eventually_at_topological*)

done

lemma *eventually_in_cosparse*:

assumes $A \subseteq X$ *open A*

shows $\text{eventually } (\lambda x. x \in X) \text{ (cosparse } A)$

proof –

have $\text{eventually } (\lambda x. x \in A) \text{ (cosparse } A)$

using *assms* **by** (*auto simp: eventually_cosparse_open_eq intro: eventually_at_in_open'*)

thus *?thesis*

by *eventually_elim* (*use assms(1) in blast*)

qed

lemma *cosparse_eq_bot_iff*: $\text{cosparse } A = \text{bot} \longleftrightarrow (\forall x \in A. \text{open } \{x\})$

proof –

have $\text{cosparse } A = \text{bot} \longleftrightarrow \text{eventually } (\lambda_. \text{False}) \text{ (cosparse } A)$

by (*simp add: trivial_limit_def*)

also have $\dots \longleftrightarrow (\forall x \in A. \text{open } \{x\})$

unfolding *eventually_cosparse sparse_in_def*

by (*auto simp: islimpt_UNIV_iff*)

finally show *?thesis* .

qed

lemma *cosparse_empty* [*simp*]: $\text{cosparse } \{\} = \text{bot}$

by (*rule filter_eqI*) (*auto simp: eventually_cosparse sparse_in_def*)

lemma *cosparse_eq_bot_iff'* [*simp*]: $\text{cosparse } (A :: 'a :: \text{perfect_space set}) = \text{bot} \longleftrightarrow A = \{\}$

by (*auto simp: cosparse_eq_bot_iff not_open_singleton*)

end

6.12 Operator Norm

```
theory Operator_Norm
imports Complex_Main
begin
```

This formulation yields zero if $'a$ is the trivial vector space.

definition

```
onorm :: ('a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector)  $\Rightarrow$  real where
onorm f = (SUP x. norm (f x) / norm x)
```

proposition *onorm_bound*:

```
  assumes  $0 \leq b$  and  $\bigwedge x. \text{norm } (f x) \leq b * \text{norm } x$ 
  shows  $\text{onorm } f \leq b$ 
  unfolding onorm_def
proof (rule cSUP_least)
  fix x
  show  $\text{norm } (f x) / \text{norm } x \leq b$ 
    using assms by (cases  $x = 0$ ) (simp_all add: pos_divide_le_eq)
qed simp
```

In non-trivial vector spaces, the first assumption is redundant.

lemma *onorm_le*:

```
  fixes f :: 'a::{real_normed_vector, perfect_space}  $\Rightarrow$  'b::real_normed_vector
  assumes  $\bigwedge x. \text{norm } (f x) \leq b * \text{norm } x$ 
  shows  $\text{onorm } f \leq b$ 
proof (rule onorm_bound [OF _ assms])
  have  $\{0::'a\} \neq \text{UNIV}$  by (metis not_open_singleton open_UNIV)
  then obtain a :: 'a where  $a \neq 0$  by fast
  have  $0 \leq b * \text{norm } a$ 
    by (rule order_trans [OF norm_ge_zero assms])
  with  $\langle a \neq 0 \rangle$  show  $0 \leq b$ 
    by (simp add: zero_le_mult_iff)
qed
```

lemma *le_onorm*:

```
  assumes bounded_linear f
  shows  $\text{norm } (f x) / \text{norm } x \leq \text{onorm } f$ 
proof -
  interpret f: bounded_linear f by fact
  obtain b where  $0 \leq b$  and  $\forall x. \text{norm } (f x) \leq \text{norm } x * b$ 
    using f.nonneg_bounded by auto
  then have  $\forall x. \text{norm } (f x) / \text{norm } x \leq b$ 
    by (clarify, case_tac  $x = 0$ ,
      simp_all add: f.zero_pos_divide_le_eq mult.commute)
  then have bdd_above (range  $(\lambda x. \text{norm } (f x) / \text{norm } x)$ )
```

1730

```
  unfolding bdd_above_def by fast
  with UNIV_I show ?thesis
  unfolding onorm_def by (rule cSUP_upper)
qed
```

```
lemma onorm:
  assumes bounded_linear f
  shows norm (f x) ≤ onorm f * norm x
proof -
  interpret f: bounded_linear f by fact
  show ?thesis
  proof (cases)
    assume x = 0
    then show ?thesis by (simp add: f.zero)
  next
    assume x ≠ 0
    have norm (f x) / norm x ≤ onorm f
      by (rule le_onorm [OF assms])
    then show norm (f x) ≤ onorm f * norm x
      by (simp add: pos_divide_le_eq ⟨x ≠ 0⟩)
  qed
qed
```

```
lemma onorm_pos_le:
  assumes f: bounded_linear f
  shows 0 ≤ onorm f
  using le_onorm [OF f, where x=0] by simp
```

```
lemma onorm_zero: onorm (λx. 0) = 0
proof (rule order_antisym)
  show onorm (λx. 0) ≤ 0
    by (simp add: onorm_bound)
  show 0 ≤ onorm (λx. 0)
    using bounded_linear_zero by (rule onorm_pos_le)
qed
```

```
lemma onorm_eq_0:
  assumes f: bounded_linear f
  shows onorm f = 0 ↔ (∀x. f x = 0)
  using onorm [OF f] by (auto simp: fun_eq_iff [symmetric] onorm_zero)
```

```
lemma onorm_pos_lt:
  assumes f: bounded_linear f
  shows 0 < onorm f ↔ ¬ (∀x. f x = 0)
  by (simp add: less_le onorm_pos_le [OF f] onorm_eq_0 [OF f])
```

```
lemma onorm_id_le: onorm (λx. x) ≤ 1
  by (rule onorm_bound) simp_all
```

```

lemma onorm_id:  $\text{onorm } (\lambda x. x::'a::\{\text{real\_normed\_vector}, \text{perfect\_space}\}) = 1$ 
proof (rule antisym[OF onorm_id_le])
  have  $\{0::'a\} \neq \text{UNIV}$  by (metis not_open_singleton open_UNIV)
  then obtain  $x :: 'a$  where  $x \neq 0$  by fast
  hence  $1 \leq \text{norm } x / \text{norm } x$ 
    by simp
  also have  $\dots \leq \text{onorm } (\lambda x::'a. x)$ 
    by (rule le_onorm) (rule bounded_linear_ident)
  finally show  $1 \leq \text{onorm } (\lambda x::'a. x)$  .
qed

```

```

lemma onorm_compose:
  assumes  $f$ : bounded_linear  $f$ 
  assumes  $g$ : bounded_linear  $g$ 
  shows  $\text{onorm } (f \circ g) \leq \text{onorm } f * \text{onorm } g$ 
proof (rule onorm_bound)
  show  $0 \leq \text{onorm } f * \text{onorm } g$ 
    by (intro mult_nonneg_nonneg onorm_pos_le f g)
next
  fix  $x$ 
  have  $\text{norm } (f (g x)) \leq \text{onorm } f * \text{norm } (g x)$ 
    by (rule onorm [OF f])
  also have  $\text{onorm } f * \text{norm } (g x) \leq \text{onorm } f * (\text{onorm } g * \text{norm } x)$ 
    by (rule mult_left_mono [OF onorm [OF g] onorm_pos_le [OF f]])
  finally show  $\text{norm } ((f \circ g) x) \leq \text{onorm } f * \text{onorm } g * \text{norm } x$ 
    by (simp add: mult.assoc)
qed

```

```

lemma onorm_scaleR_lemma:
  assumes  $f$ : bounded_linear  $f$ 
  shows  $\text{onorm } (\lambda x. r *_R f x) \leq |r| * \text{onorm } f$ 
proof (rule onorm_bound)
  show  $0 \leq |r| * \text{onorm } f$ 
    by (intro mult_nonneg_nonneg onorm_pos_le abs_ge_zero f)
next
  fix  $x$ 
  have  $|r| * \text{norm } (f x) \leq |r| * (\text{onorm } f * \text{norm } x)$ 
    by (intro mult_left_mono onorm abs_ge_zero f)
  then show  $\text{norm } (r *_R f x) \leq |r| * \text{onorm } f * \text{norm } x$ 
    by (simp only: norm_scaleR mult.assoc)
qed

```

```

lemma onorm_scaleR:
  assumes  $f$ : bounded_linear  $f$ 
  shows  $\text{onorm } (\lambda x. r *_R f x) = |r| * \text{onorm } f$ 
proof (cases r = 0)
  assume  $r \neq 0$ 
  show ?thesis
  proof (rule order_antisym)

```

```

show  $onorm (\lambda x. r *_R f x) \leq |r| * onorm f$ 
  using  $f$  by (rule  $onorm\_scaleR\_lemma$ )
next
  have  $bounded\_linear (\lambda x. r *_R f x)$ 
    using  $bounded\_linear\_scaleR\_right f$  by (rule  $bounded\_linear\_compose$ )
  then have  $onorm (\lambda x. inverse r *_R r *_R f x) \leq |inverse r| * onorm (\lambda x. r *_R$ 
 $f x)$ 
    by (rule  $onorm\_scaleR\_lemma$ )
  with  $\langle r \neq 0 \rangle$  show  $|r| * onorm f \leq onorm (\lambda x. r *_R f x)$ 
    by (simp add:  $inverse\_eq\_divide$   $pos\_le\_divide\_eq$   $mult.commute$ )
qed
qed (simp add:  $onorm\_zero$ )

```

```

lemma  $onorm\_scaleR\_left\_lemma$ :
  assumes  $r$ :  $bounded\_linear r$ 
  shows  $onorm (\lambda x. r x *_R f) \leq onorm r * norm f$ 
proof (rule  $onorm\_bound$ )
  fix  $x$ 
  have  $norm (r x *_R f) = norm (r x) * norm f$ 
    by  $simp$ 
  also have  $\dots \leq onorm r * norm x * norm f$ 
    by (intro  $mult\_right\_mono$   $onorm r$   $norm\_ge\_zero$ )
  finally show  $norm (r x *_R f) \leq onorm r * norm f * norm x$ 
    by (simp add:  $ac\_simps$ )
qed (intro  $mult\_nonneg\_nonneg$   $norm\_ge\_zero$   $onorm\_pos\_le r$ )

```

```

lemma  $onorm\_scaleR\_left$ :
  assumes  $f$ :  $bounded\_linear r$ 
  shows  $onorm (\lambda x. r x *_R f) = onorm r * norm f$ 
proof (cases  $f = 0$ )
  assume  $f \neq 0$ 
  show  $?thesis$ 
proof (rule  $order\_antisym$ )
  show  $onorm (\lambda x. r x *_R f) \leq onorm r * norm f$ 
    using  $f$  by (rule  $onorm\_scaleR\_left\_lemma$ )
next
  have  $bl1$ :  $bounded\_linear (\lambda x. r x *_R f)$ 
    by (metis  $bounded\_linear\_scaleR\_const f$ )
  have  $bounded\_linear (\lambda x. r x * norm f)$ 
    by (metis  $bounded\_linear\_mult\_const f$ )
  from  $onorm\_scaleR\_left\_lemma$  [OF  $this$ , of  $inverse (norm f)$ ]
  have  $onorm r \leq onorm (\lambda x. r x * norm f) * inverse (norm f)$ 
    using  $\langle f \neq 0 \rangle$ 
    by (simp add:  $inverse\_eq\_divide$ )
  also have  $onorm (\lambda x. r x * norm f) \leq onorm (\lambda x. r x *_R f)$ 
    by (rule  $onorm\_bound$ )
    (auto simp:  $abs\_mult$   $bl1$   $onorm\_pos\_le$  intro!:  $order\_trans$  [OF  $\_ onorm$ ])
  finally show  $onorm r * norm f \leq onorm (\lambda x. r x *_R f)$ 
    using  $\langle f \neq 0 \rangle$ 

```



```

    by (simp add: inverse_eq_divide pos_le_divide_eq mult.commute)
  qed
qed (simp add: onorm_zero)

```

```

lemma onorm_neg:
  shows onorm ( $\lambda x. - f x$ ) = onorm f
  unfolding onorm_def by simp

```

```

lemma onorm_triangle:
  assumes f: bounded_linear f
  assumes g: bounded_linear g
  shows onorm ( $\lambda x. f x + g x$ )  $\leq$  onorm f + onorm g
proof (rule onorm_bound)
  show 0  $\leq$  onorm f + onorm g
    by (intro add_nonneg_nonneg onorm_pos_le f g)
next
  fix x
  have norm (f x + g x)  $\leq$  norm (f x) + norm (g x)
    by (rule norm_triangle_ineq)
  also have norm (f x) + norm (g x)  $\leq$  onorm f * norm x + onorm g * norm x
    by (intro add_mono onorm f g)
  finally show norm (f x + g x)  $\leq$  (onorm f + onorm g) * norm x
    by (simp only: distrib_right)
qed

```

```

lemma onorm_triangle_le:
  assumes bounded_linear f
  assumes bounded_linear g
  assumes onorm f + onorm g  $\leq$  e
  shows onorm ( $\lambda x. f x + g x$ )  $\leq$  e
  using assms by (rule onorm_triangle [THEN order_trans])

```

```

lemma onorm_triangle_lt:
  assumes bounded_linear f
  assumes bounded_linear g
  assumes onorm f + onorm g  $<$  e
  shows onorm ( $\lambda x. f x + g x$ )  $<$  e
  using assms by (rule onorm_triangle [THEN order_le_less_trans])

```

```

lemma onorm_sum:
  assumes finite S
  assumes  $\bigwedge s. s \in S \implies$  bounded_linear (f s)
  shows onorm ( $\lambda x. \text{sum } (\lambda s. f s x) S$ )  $\leq$  sum ( $\lambda s. \text{onorm } (f s)$ ) S
  using assms
  by (induction) (auto simp: onorm_zero intro!: onorm_triangle_le bounded_linear_sum)

```

```

lemmas onorm_sum_le = onorm_sum [THEN order_trans]

```

```

end

```

6.13 Limits on the Extended Real Number Line

theory *Extended_Real_Limits*

imports

Topology_Euclidean_Space

HOL-Library.Extended_Real

HOL-Library.Extended_Nonnegative_Real

HOL-Library.Indicator_Function

begin

lemma *compact_UNIV*:

compact (*UNIV* :: 'a::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set)

using *compact_complete_linorder*

by (*auto simp: seq_compact_eq_compact[symmetric] seq_compact_def*)

lemma *compact_eq_closed*:

fixes *S* :: 'a::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set

shows *compact S* \longleftrightarrow *closed S*

using *closed_Int_compact[of S, OF _ compact_UNIV] compact_imp_closed*

by *auto*

lemma *closed_contains_Sup_cl*:

fixes *S* :: 'a::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set

assumes *closed S*

and *S* \neq {}

shows *Sup S* \in *S*

proof –

from *compact_eq_closed[of S] compact_attains_sup[of S] assms*

obtain *s* **where** *S*: *s* \in *S* $\forall t \in S. t \leq s$

by *auto*

then have *Sup S* = *s*

by (*auto intro!: Sup_eqI*)

with *S* **show** *?thesis*

by *simp*

qed

lemma *closed_contains_Inf_cl*:

fixes *S* :: 'a::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set

assumes *closed S*

and *S* \neq {}

shows *Inf S* \in *S*

proof –

from *compact_eq_closed[of S] compact_attains_inf[of S] assms*

obtain *s* **where** *S*: *s* \in *S* $\forall t \in S. s \leq t$

by *auto*

```

then have  $\text{Inf } S = s$ 
  by (auto intro!: Inf_eqI)
with  $S$  show ?thesis
  by simp
qed

instance enat :: second_countable_topology
proof
  show  $\exists B::\text{enat set set. countable } B \wedge \text{open} = \text{generate\_topology } B$ 
  proof (intro exI conjI)
    show countable (range lessThan  $\cup$  range greaterThan::enat set set)
      by auto
    qed (simp add: open_enat_def)
qed

instance ereal :: second_countable_topology
proof (standard, intro exI conjI)
  let  $?B = (\bigcup r \in \mathbb{Q}. \{\dots < r\}, \{r < \dots\}) :: \text{ereal set set}$ 
  show countable  $?B$ 
    by (auto intro: countable_rat)
  show open = generate_topology  $?B$ 
  proof (intro ext iffI)
    fix  $S :: \text{ereal set}$ 
    assume open  $S$ 
    then show generate_topology  $?B S$ 
      unfolding open_generated_order
    proof induct
      case (Basis  $b$ )
      then obtain  $e$  where  $b = \{\dots < e\} \vee b = \{e < \dots\}$ 
      by auto
      moreover have  $\{\dots < e\} = \bigcup \{\{\dots < x\} \mid x. x \in \mathbb{Q} \wedge x < e\} \{e < \dots\} = \bigcup \{\{x < \dots\} \mid x.$ 
 $x \in \mathbb{Q} \wedge e < x\}$ 
      by (auto dest: ereal_dense3
        simp del: ex_simps
        simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
      ultimately show ?case
      by (auto intro: generate_topology.intros)
    qed (auto intro: generate_topology.intros)
  next
    fix  $S$ 
    assume generate_topology  $?B S$ 
    then show open  $S$ 
      by induct auto
    qed
qed

```

This is a copy from *ereal* :: *second_countable_topology*. Maybe find a common super class of topological spaces where the rational numbers are densely embedded ?

```

instance ennreal :: second_countable_topology
proof (standard, intro exI conjI)
  let ?B = ( $\bigcup r \in \mathbb{Q}. \{.. < r\}, \{r <..\}$ ) :: ennreal set set)
  show countable ?B
    by (auto intro: countable_rat)
  show open = generate_topology ?B
proof (intro ext iffI)
  fix S :: ennreal set
  assume open S
  then show generate_topology ?B S
    unfolding open_generated_order
proof induct
  case (Basis b)
  then obtain e where b = {.. $e$ }  $\vee$  b = { $e$ .. $x$ }
    by auto
  moreover have {.. $e$ } =  $\bigcup \{\{.. $x$ \} | x. x \in \mathbb{Q} \wedge x < e\}$  { $e$ .. $x$ } =  $\bigcup \{\{x<..\} | x.$ 
 $x \in \mathbb{Q} \wedge e < x\}$ 
    by (auto dest: ennreal_rat_dense
      simp del: ex_simps
      simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
  ultimately show ?case
    by (auto intro: generate_topology.intros)
qed (auto intro: generate_topology.intros)
next
  fix S
  assume generate_topology ?B S
  then show open S
    by induct auto
qed
qed

lemma ereal_open_closed_aux:
  fixes S :: ereal set
  assumes open S
  and closed S
  and S:  $(-\infty) \notin S$ 
  shows S = {}
proof (rule ccontr)
  assume  $\neg$  ?thesis
  then have *:  $\text{Inf } S \in S$ 

  by (metis assms(2) closed_contains_Inf_cl)
  {
    assume  $\text{Inf } S = -\infty$ 
    then have False
      using * assms(3) by auto
  }
moreover
  {

```

```

  assume  $\text{Inf } S = \infty$ 
  then have  $S = \{\infty\}$ 
    by (metis  $\text{Inf\_eq\_PInfty } \langle S \neq \{\} \rangle$ )
  then have False
    by (metis  $\text{assms}(1) \text{ not\_open\_singleton}$ )
}
moreover
{
  assume  $\text{fin}: |\text{Inf } S| \neq \infty$ 
  from  $\text{ereal\_open\_cont\_interval}[OF \text{ assms}(1) * \text{fin}]$ 
  obtain  $e$  where  $e: e > 0 \{ \text{Inf } S - e <..< \text{Inf } S + e \} \subseteq S$  .
  then obtain  $b$  where  $b: \text{Inf } S - e < b < \text{Inf } S$ 
    using  $\text{fin } \text{ereal\_between}[of \text{ Inf } S e] \text{ dense}[of \text{ Inf } S - e]$ 
    by auto
  then have  $b \in \{ \text{Inf } S - e <..< \text{Inf } S + e \}$ 
    using  $e \text{ fin } \text{ereal\_between}[of \text{ Inf } S e]$ 
    by auto
  then have  $b \in S$ 
    using  $e$  by auto
  then have False
    using  $b$  by (metis  $\text{complete\_lattice\_class.Inf\_lower } leD$ )
}
ultimately show False
  by auto
qed

```

```

lemma  $\text{ereal\_open\_closed}$ :
  fixes  $S :: \text{ereal set}$ 
  shows  $\text{open } S \wedge \text{closed } S \longleftrightarrow S = \{\} \vee S = UNIV$ 
  using  $\text{ereal\_open\_closed\_aux } \text{open\_closed}$  by auto

```

```

lemma  $\text{ereal\_open\_atLeast}$ :
  fixes  $x :: \text{ereal}$ 
  shows  $\text{open } \{x..\} \longleftrightarrow x = -\infty$ 
  by (metis  $\text{atLeast\_eq\_UNIV\_iff } \text{bot\_ereal\_def } \text{closed\_atLeast } \text{ereal\_open\_closed}$ 
   $\text{not\_Ici\_eq\_empty}$ )

```

```

lemma  $\text{mono\_closed\_real}$ :
  fixes  $S :: \text{real set}$ 
  assumes  $\text{mono}: \forall y z. y \in S \wedge y \leq z \longrightarrow z \in S$ 
  and  $\text{closed } S$ 
  shows  $S = \{\} \vee S = UNIV \vee (\exists a. S = \{a..\})$ 

```

```

proof -
  {
    assume  $S \neq \{\}$ 
    { assume  $\text{ex}: \exists B. \forall x \in S. B \leq x$ 
      then have  $*$ :  $\forall x \in S. \text{Inf } S \leq x$ 
        using  $\text{cInf\_lower}[of \_ S] \text{ ex}$  by (metis  $\text{bdd\_below\_def}$ )
      then have  $\text{Inf } S \in S$ 

```

```

    by (meson ⟨S ≠ {}⟩ assms(2) bdd_belowI closed_contains_Inf)
  then have  $\forall x. \text{Inf } S \leq x \longleftrightarrow x \in S$ 
    using mono[rule_format, of Inf S] *
    by auto
  then have  $S = \{\text{Inf } S \dots\}$ 
    by auto
  then have  $\exists a. S = \{a \dots\}$ 
    by auto
}
moreover
{
  assume  $\neg (\exists B. \forall x \in S. B \leq x)$ 
  then have nex:  $\forall B. \exists x \in S. x < B$ 
    by (simp add: not_le)
  {
    fix y
    obtain x where  $x \in S$  and  $x < y$ 
      using nex by auto
    then have  $y \in S$ 
      using mono[rule_format, of x y] by auto
  }
  then have  $S = \text{UNIV}$ 
    by auto
}
ultimately have  $S = \text{UNIV} \vee (\exists a. S = \{a \dots\})$ 
  by blast
}
then show ?thesis
  by blast
qed

lemma mono_closed_ereal:
  fixes S :: real set
  assumes mono:  $\forall y z. y \in S \wedge y \leq z \longrightarrow z \in S$ 
    and closed S
  shows  $\exists a. S = \{x. a \leq \text{ereal } x\}$ 
proof -
  consider  $S = \{\} \vee S = \text{UNIV} \mid (\exists a. S = \{a \dots\})$ 
  using assms(2) mono mono_closed_real by blast
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
      by (meson PInfy_neq_ereal(1) UNIV_eq_I bot.extremum empty_Collect_eq
        ereal_infy_less_eq(1) mem_Collect_eq)
    next
    case 2
    then show ?thesis
      by (metis atLeast_iff ereal_less_eq(3) mem_Collect_eq subsetI subset_antisym)
  end
end

```

qed
qed

lemma *Liminf_within*:

fixes $f :: 'a::metric_space \Rightarrow 'b::complete_lattice$

shows $Liminf\ (at\ x\ within\ S)\ f = (SUP\ e \in \{0 < ..\}. INF\ y \in (S \cap ball\ x\ e - \{x\}). f\ y)$

unfolding *Liminf_def eventually_at*

proof (*rule SUP_eq, simp_all add: Ball_def Bex_def, safe*)

fix $P\ d$

assume $0 < d$ **and** $\forall y. y \in S \longrightarrow y \neq x \wedge dist\ y\ x < d \longrightarrow P\ y$

then have $S \cap ball\ x\ d - \{x\} \subseteq \{x. P\ x\}$

by (*auto simp: dist_commute*)

then show $\exists r > 0. Inf\ (f\ ' (Collect\ P)) \leq Inf\ (f\ ' (S \cap ball\ x\ r - \{x\}))$

by (*intro exI[of _ d] INF_mono conjI <0 < d> auto*)

next

fix $d :: real$

assume $0 < d$

then show $\exists P. (\exists d > 0. \forall xa. xa \in S \longrightarrow xa \neq x \wedge dist\ xa\ x < d \longrightarrow P\ xa) \wedge Inf\ (f\ ' (S \cap ball\ x\ d - \{x\})) \leq Inf\ (f\ ' (Collect\ P))$

by (*intro exI[of _ $\lambda y. y \in S \cap ball\ x\ d - \{x\}$] auto*)

(*auto intro!: INF_mono exI[of _ d] simp: dist_commute*)

qed

lemma *Limsup_within*:

fixes $f :: 'a::metric_space \Rightarrow 'b::complete_lattice$

shows $Limsup\ (at\ x\ within\ S)\ f = (INF\ e \in \{0 < ..\}. SUP\ y \in (S \cap ball\ x\ e - \{x\}). f\ y)$

unfolding *Limsup_def eventually_at*

proof (*rule INF_eq, simp_all add: Ball_def Bex_def, safe*)

fix $P\ d$

assume $0 < d$ **and** $\forall y. y \in S \longrightarrow y \neq x \wedge dist\ y\ x < d \longrightarrow P\ y$

then have $S \cap ball\ x\ d - \{x\} \subseteq \{x. P\ x\}$

by (*auto simp: dist_commute*)

then show $\exists r > 0. Sup\ (f\ ' (S \cap ball\ x\ r - \{x\})) \leq Sup\ (f\ ' (Collect\ P))$

by (*intro exI[of _ d] SUP_mono conjI <0 < d> auto*)

next

fix $d :: real$

assume $0 < d$

then show $\exists P. (\exists d > 0. \forall xa. xa \in S \longrightarrow xa \neq x \wedge dist\ xa\ x < d \longrightarrow P\ xa) \wedge Sup\ (f\ ' (Collect\ P)) \leq Sup\ (f\ ' (S \cap ball\ x\ d - \{x\}))$

by (*intro exI[of _ $\lambda y. y \in S \cap ball\ x\ d - \{x\}$] auto*)

(*auto intro!: SUP_mono exI[of _ d] simp: dist_commute*)

qed

lemma *Liminf_at*:

fixes $f :: 'a::metric_space \Rightarrow 'b::complete_lattice$

shows $Liminf\ (at\ x)\ f = (SUP\ e \in \{0 < ..\}. INF\ y \in (ball\ x\ e - \{x\}). f\ y)$

using *Liminf_within*[of $x\ UNIV\ f$] **by** *simp*

lemma *Limsup_at*:
fixes $f :: 'a::metric_space \Rightarrow 'b::complete_lattice$
shows $Limsup\ (at\ x)\ f = (INF\ e \in \{0 < ..\}. SUP\ y \in (ball\ x\ e - \{x\}). f\ y)$
using *Limsup_within[of x UNIV f]* **by** *simp*

lemma *min_Liminf_at*:
fixes $f :: 'a::metric_space \Rightarrow 'b::complete_linorder$
shows $min\ (f\ x)\ (Liminf\ (at\ x)\ f) = (SUP\ e \in \{0 < ..\}. INF\ y \in ball\ x\ e.\ f\ y)$
apply (*simp add: inf_min [symmetric] Liminf_at inf_commute [of f] SUP_inf*)
apply (*metis (no_types, lifting) INF_insert centre_in_ball greaterThan_iff image_cong inf_commute insert_Diff*)
done

6.13.1 Extended-Real.thy

lemma *sum_constant_ereal*:
fixes $a::ereal$
shows $(\sum\ i \in I.\ a) = a * card\ I$
proof (*induction I rule: infinite_finite_induct*)
case (*insert i I*)
then show *?case*
by (*simp add: ereal_right_distrib flip: plus_ereal.simps*)
qed *auto*

lemma *real_lim_then_eventually_real*:
assumes $(u \longrightarrow\ ereal\ l)\ F$
shows *eventually* $(\lambda n.\ u\ n = ereal(real_of_ereal(u\ n)))\ F$
proof –
have $ereal\ l \in \{-\infty < .. < (\infty::ereal)\}$ **by** *simp*
moreover have *open* $\{-\infty < .. < (\infty::ereal)\}$ **by** *simp*
ultimately have *eventually* $(\lambda n.\ u\ n \in \{-\infty < .. < (\infty::ereal)\})\ F$ **using** *assms tendsto_def* **by** *blast*
moreover have $\bigwedge x.\ x \in \{-\infty < .. < (\infty::ereal)\} \implies x = ereal(real_of_ereal\ x)$
using *ereal_real* **by** *auto*
ultimately show *?thesis* **by** (*metis (mono_tags, lifting) eventually_mono*)
qed

lemma *ereal_Inf_cmult*:
assumes $c > (0::real)$
shows $Inf\ \{ereal\ c * x \mid x.\ P\ x\} = ereal\ c * Inf\ \{x.\ P\ x\}$
proof –
have *bij* $((*)\ (ereal\ c))$
apply (*rule bij_betw_byWitness[of _ $\lambda x.\ (x::ereal) / c]$, *auto simp: assms ereal_mult_divide*)
using *assms ereal_divide_eq* **by** *auto*
then have $ereal\ c * Inf\ \{x.\ P\ x\} = Inf\ ((*)\ (ereal\ c)\ \{x.\ P\ x\})$
by (*simp add: assms ereal_mult_left_mono less_imp_le mono_def mono_bij_Inf*)
then show *?thesis**


```

  by (simp add: setcompr_eq_image)
qed

```

Continuity of addition

The next few lemmas remove an unnecessary assumption in *tendsto_add_ereal*, culminating in *tendsto_add_ereal_general* which essentially says that the addition is continuous on *ereal* times *ereal*, except at $(-\infty, \infty)$ and $(\infty, -\infty)$. It is much more convenient in many situations, see for instance the proof of *tendsto_sum_ereal* below.

```

lemma tendsto_add_ereal_PInf:
  fixes y :: ereal
  assumes y: y  $\neq$   $-\infty$ 
  assumes f: (f  $\longrightarrow$   $\infty$ ) F and g: (g  $\longrightarrow$  y) F
  shows (( $\lambda$ x. f x + g x)  $\longrightarrow$   $\infty$ ) F
proof -
  have  $\exists$  C. eventually ( $\lambda$ x. g x > ereal C) F
  proof (cases y)
    case (real r)
      have y > y-1 using y real by (simp add: ereal_between(1))
      then have eventually ( $\lambda$ x. g x > y - 1) F using g y order_tendsto_iff by
        auto
      moreover have y-1 = ereal(real_of_ereal(y-1))
        by (metis real_ereal_eq_1(1) ereal_minus(1) real_of_ereal.simps(1))
      ultimately have eventually ( $\lambda$ x. g x > ereal(real_of_ereal(y - 1))) F by
        simp
      then show ?thesis by auto
    next
      case (PInf)
        have eventually ( $\lambda$ x. g x > ereal 0) F using g PInf by (simp add: tend-
          sto_PInfTy)
        then show ?thesis by auto
      qed (simp add: y)
    then obtain C::real where ge: eventually ( $\lambda$ x. g x > ereal C) F by auto

  {
    fix M::real
    have eventually ( $\lambda$ x. f x > ereal(M - C)) F using f by (simp add: tend-
      sto_PInfTy)
    then have eventually ( $\lambda$ x. (f x > ereal (M-C))  $\wedge$  (g x > ereal C)) F
      by (auto simp: ge eventually_conj_iff)
    moreover have  $\bigwedge$ x. ((f x > ereal (M-C))  $\wedge$  (g x > ereal C))  $\implies$  (f x + g x
      > ereal M)
      using ereal_add_strict_mono2 by fastforce
    ultimately have eventually ( $\lambda$ x. f x + g x > ereal M) F using eventu-
      ally_mono by force
  }
  then show ?thesis by (simp add: tendsto_PInfTy)

```

qed

One would like to deduce the next lemma from the previous one, but the fact that $-(x + y)$ is in general different from $(-x) + (-y)$ in ereal creates difficulties, so it is more efficient to copy the previous proof.

lemma *tendsto_add_ereal_MInf*:

fixes $y :: \text{ereal}$

assumes $y: y \neq \infty$

assumes $f: (f \longrightarrow -\infty) F$ **and** $g: (g \longrightarrow y) F$

shows $((\lambda x. f x + g x) \longrightarrow -\infty) F$

proof –

have $\exists C. \text{eventually } (\lambda x. g x < \text{ereal } C) F$

proof (*cases y*)

case (*real r*)

have $y < y+1$ **using** *y real* **by** (*simp add: ereal_between(1)*)

then have *eventually* $(\lambda x. g x < y + 1) F$ **using** *g y order_tendsto_iff* **by force**

moreover have $y+1 = \text{ereal}(\text{real_of_ereal } (y+1))$ **by** (*simp add: real*)

ultimately have *eventually* $(\lambda x. g x < \text{ereal}(\text{real_of_ereal}(y + 1))) F$ **by simp**

then show *?thesis* **by auto**

next

case (*MInf*)

have *eventually* $(\lambda x. g x < \text{ereal } 0) F$ **using** *g MInf* **by** (*simp add: tendsto_MInfnty*)

then show *?thesis* **by auto**

qed (*simp add: y*)

then obtain $C::\text{real}$ **where** *ge: eventually* $(\lambda x. g x < \text{ereal } C) F$ **by auto**

{

fix $M::\text{real}$

have *eventually* $(\lambda x. f x < \text{ereal}(M - C)) F$ **using** *f* **by** (*simp add: tendsto_MInfnty*)

then have *eventually* $(\lambda x. (f x < \text{ereal } (M - C)) \wedge (g x < \text{ereal } C)) F$

by (*auto simp: ge eventually_conj_iff*)

moreover have $\bigwedge x. ((f x < \text{ereal } (M - C)) \wedge (g x < \text{ereal } C)) \implies (f x + g x < \text{ereal } M)$

using *ereal_add_strict_mono2* **by fastforce**

ultimately have *eventually* $(\lambda x. f x + g x < \text{ereal } M) F$ **using** *eventually_mono* **by force**

}

then show *?thesis* **by** (*simp add: tendsto_MInfnty*)

qed

lemma *tendsto_add_ereal_general1*:

fixes $x y :: \text{ereal}$

assumes $y: |y| \neq \infty$

assumes $f: (f \longrightarrow x) F$ **and** $g: (g \longrightarrow y) F$

shows $((\lambda x. f x + g x) \longrightarrow x + y) F$

```

proof (cases x)
  case (real r)
    have a:  $|x| \neq \infty$  by (simp add: real)
    show ?thesis by (rule tendsto_add_ereal[OF a, OF y, OF f, OF g])
  next
    case PInf
    then show ?thesis using tendsto_add_ereal_PInf assms by force
  next
    case MInf
    then show ?thesis using tendsto_add_ereal_MInf assms
      by (metis abs_ereal.simps(3) ereal_MInfty_eq_plus)
qed

```

lemma *tendsto_add_ereal_general2*:

```

fixes x y :: ereal
assumes x:  $|x| \neq \infty$ 
  and f:  $(f \longrightarrow x) F$  and g:  $(g \longrightarrow y) F$ 
shows  $((\lambda x. f x + g x) \longrightarrow x + y) F$ 
proof -
  have  $((\lambda x. g x + f x) \longrightarrow x + y) F$ 
    by (metis (full_types) add commute f g tendsto_add_ereal_general1 x)
  moreover have  $\bigwedge x. g x + f x = f x + g x$  using add commute by auto
  ultimately show ?thesis by simp
qed

```

The next lemma says that the addition is continuous on *ereal*, except at the pairs $(-\infty, \infty)$ and $(\infty, -\infty)$.

lemma *tendsto_add_ereal_general* [*tendsto_intros*]:

```

fixes x y :: ereal
assumes  $\neg((x=\infty \wedge y=-\infty) \vee (x=-\infty \wedge y=\infty))$ 
  and f:  $(f \longrightarrow x) F$  and g:  $(g \longrightarrow y) F$ 
shows  $((\lambda x. f x + g x) \longrightarrow x + y) F$ 
proof (cases x)
  case (real r)
    show ?thesis
      using real assms
      by (intro tendsto_add_ereal_general2; auto)
  next
    case (PInf)
    then have  $y \neq -\infty$  using assms by simp
    then show ?thesis using tendsto_add_ereal_PInf PInf assms by auto
  next
    case (MInf)
    then have  $y \neq \infty$  using assms by simp
    then show ?thesis
      by (metis ereal_MInfty_eq_plus tendsto_add_ereal_MInf MInf f g)
qed

```

Continuity of multiplication

In the same way as for addition, we prove that the multiplication is continuous on ereal times ereal, except at $(\infty, 0)$ and $(-\infty, 0)$ and $(0, \infty)$ and $(0, -\infty)$, starting with specific situations.

lemma *tendsto_mult_real_ereal*:

assumes $(u \longrightarrow \text{ereal } l) F (v \longrightarrow \text{ereal } m) F$

shows $((\lambda n. u \ n * v \ n) \longrightarrow \text{ereal } l * \text{ereal } m) F$

proof –

have *ureal*: *eventually* $(\lambda n. u \ n = \text{ereal}(\text{real_of_ereal}(u \ n))) F$ **by** $(\text{rule } \text{real_lim_then_eventually_real}[\text{assms}(1)])$

then have $((\lambda n. \text{ereal}(\text{real_of_ereal}(u \ n))) \longrightarrow \text{ereal } l) F$ **using** *assms* **by** *auto*

then have *limu*: $((\lambda n. \text{real_of_ereal}(u \ n)) \longrightarrow l) F$ **by** *auto*

have *vreal*: *eventually* $(\lambda n. v \ n = \text{ereal}(\text{real_of_ereal}(v \ n))) F$ **by** $(\text{rule } \text{real_lim_then_eventually_real}[\text{assms}(2)])$

then have $((\lambda n. \text{ereal}(\text{real_of_ereal}(v \ n))) \longrightarrow \text{ereal } m) F$ **using** *assms* **by** *auto*

then have *limv*: $((\lambda n. \text{real_of_ereal}(v \ n)) \longrightarrow m) F$ **by** *auto*

{
fix *n* **assume** $u \ n = \text{ereal}(\text{real_of_ereal}(u \ n)) \ v \ n = \text{ereal}(\text{real_of_ereal}(v \ n))$
then have $\text{ereal}(\text{real_of_ereal}(u \ n) * \text{real_of_ereal}(v \ n)) = u \ n * v \ n$
by $(\text{metis } \text{times_ereal.simps}(1))$

}
then have ***: *eventually* $(\lambda n. \text{ereal}(\text{real_of_ereal}(u \ n) * \text{real_of_ereal}(v \ n)) = u \ n * v \ n) F$

using *eventually_elim2*[*OF ureal vreal*] **by** *auto*

have $((\lambda n. \text{real_of_ereal}(u \ n) * \text{real_of_ereal}(v \ n)) \longrightarrow l * m) F$

using *tendsto_mult*[*OF limu limv*] **by** *auto*

then have $((\lambda n. \text{ereal}(\text{real_of_ereal}(u \ n) * \text{real_of_ereal}(v \ n)) \longrightarrow \text{ereal}(l * m)) F$

by *auto*

then show *?thesis* **using** ** filterlim_cong* **by** *fastforce*

qed

lemma *tendsto_mult_ereal_PInf*:

fixes *f g*: $_ \Rightarrow \text{ereal}$

assumes $(f \longrightarrow l) F \ l > 0 \ (g \longrightarrow \infty) F$

shows $((\lambda x. f \ x * g \ x) \longrightarrow \infty) F$

proof –

obtain *a*:*real* **where** $0 < \text{ereal } a \ a < l$

using *assms*(2) **using** *ereal_dense2* **by** *blast*

have ***: *eventually* $(\lambda x. f \ x > a) F$

using $\langle a < l \rangle$ *assms*(1) **by** $(\text{simp } \text{add: } \text{order_tendsto_iff})$

{
fix *K*:*real*

define *M* **where** $M = \max K \ 1$

```

    then have  $M > 0$  by simp
    then have  $\text{ereal}(M/a) > 0$  using  $\langle \text{ereal } a > 0 \rangle$  by simp
    then have  $\bigwedge x. ((f x > a) \wedge (g x > M/a)) \implies (f x * g x > \text{ereal } a * \text{ereal}(M/a))$ 
      using  $\text{ereal\_mult\_mono\_strict}$  [where  $?c = M/a$ ,  $OF \langle 0 < \text{ereal } a \rangle$ ] by
    auto
    moreover have  $\text{ereal } a * \text{ereal}(M/a) = M$  using  $\langle \text{ereal } a > 0 \rangle$  by simp
    ultimately have  $\bigwedge x. ((f x > a) \wedge (g x > M/a)) \implies (f x * g x > M)$  by simp
    moreover have  $M \geq K$  unfolding  $M\_def$  by simp
    ultimately have  $\text{Imp}: \bigwedge x. ((f x > a) \wedge (g x > M/a)) \implies (f x * g x > K)$ 
      using  $\text{ereal\_less\_eq}(3)$   $\text{le\_less\_trans}$  by blast

    have eventually  $(\lambda x. g x > M/a)$   $F$  using  $\text{assms}(3)$  by ( $\text{simp add: tendsto\_PInfy}$ )
    then have eventually  $(\lambda x. (f x > a) \wedge (g x > M/a))$   $F$ 
      using * by ( $\text{auto simp: eventually\_conj\_iff}$ )
    then have eventually  $(\lambda x. f x * g x > K)$   $F$  using  $\text{eventually\_mono Imp}$  by
    force
  }
  then show  $?thesis$  by ( $\text{auto simp: tendsto\_PInfy}$ )
qed

lemma  $\text{tendsto\_mult\_ereal\_pos}$ :
  fixes  $f g: \_ \Rightarrow \text{ereal}$ 
  assumes  $(f \longrightarrow l)$   $F$   $(g \longrightarrow m)$   $F$   $l > 0$   $m > 0$ 
  shows  $((\lambda x. f x * g x) \longrightarrow l * m)$   $F$ 
proof (cases)
  assume *:  $l = \infty \vee m = \infty$ 
  then show  $?thesis$ 
  proof (cases)
    assume  $m = \infty$ 
    then show  $?thesis$  using  $\text{tendsto\_mult\_ereal\_PInf}$   $\text{assms}$  by auto
  next
    assume  $\neg(m = \infty)$ 
    then have  $l = \infty$  using * by simp
    then have  $((\lambda x. g x * f x) \longrightarrow l * m)$   $F$  using  $\text{tendsto\_mult\_ereal\_PInf}$ 
   $\text{assms}$  by auto
    moreover have  $\bigwedge x. g x * f x = f x * g x$  using  $\text{mult.commute}$  by auto
    ultimately show  $?thesis$  by simp
  qed
next
  assume  $\neg(l = \infty \vee m = \infty)$ 
  then have  $l < \infty$   $m < \infty$  by auto
  then obtain  $lr mr$  where  $l = \text{ereal } lr$   $m = \text{ereal } mr$ 
    using  $\langle l > 0 \rangle \langle m > 0 \rangle$  by ( $\text{metis } \text{ereal\_cases } \text{ereal\_less}(6) \text{ not\_less\_iff\_gr\_or\_eq}$ )
  then show  $?thesis$  using  $\text{tendsto\_mult\_real\_ereal}$   $\text{assms}$  by auto
qed

```

We reduce the general situation to the positive case by multiplying by suitable signs. Unfortunately, as ereal is not a ring, all the neat sign lemmas

are not available there. We give the bare minimum we need.

lemma *ereal_sgn_abs*:

fixes $l::ereal$

shows $sgn(l) * l = abs(l)$

by (*cases* l , *auto simp: sgn_if_ereal_less_uminus_reorder*)

lemma *sgn_squared_ereal*:

assumes $l \neq (0::ereal)$

shows $sgn(l) * sgn(l) = 1$

using *assms* **by** (*cases* l , *auto simp: one_ereal_def_sgn_if*)

lemma *tendsto_mult_ereal* [*tendsto_intros*]:

fixes $f g::_ \Rightarrow ereal$

assumes ($f \longrightarrow l$) F ($g \longrightarrow m$) $F \neg((l=0 \wedge abs(m) = \infty) \vee (m=0 \wedge abs(l) = \infty))$

shows ($(\lambda x. f x * g x) \longrightarrow l * m$) F

proof (*cases*)

assume $l=0 \vee m=0$

then have $abs(l) \neq \infty \wedge abs(m) \neq \infty$ **using** *assms(3)* **by** *auto*

then obtain $lr mr$ **where** $l = ereal lr$ $m = ereal mr$ **by** *auto*

then show *?thesis* **using** *tendsto_mult_real_ereal* *assms* **by** *auto*

next

have *sgn_finite*: $\bigwedge a::ereal. abs(sgn a) \neq \infty$

by (*metis* *MInfty_neq_ereal(2)* *PInfty_neq_ereal(2)* *abs_eq_infinity_cases* *ereal_times(1)* *ereal_times(3)* *ereal_uminus_eq_reorder* *sgn_ereal.elims*)

then have *sgn_finite2*: $\bigwedge a b::ereal. abs(sgn a * sgn b) \neq \infty$

by (*metis* *abs_eq_infinity_cases* *abs_ereal.simps(2)* *abs_ereal.simps(3)* *ereal_mult_eq_MInfty* *ereal_mult_eq_PInfty*)

assume $\neg(l=0 \vee m=0)$

then have $l \neq 0 \wedge m \neq 0$ **by** *auto*

then have $abs(l) > 0 \wedge abs(m) > 0$

by (*metis* *abs_ereal_ge0* *abs_ereal_less0* *abs_ereal_pos* *ereal_uminus_uminus* *ereal_uminus_zero_less_le_not_less*)**+**

then have $sgn(l) * l > 0 \wedge sgn(m) * m > 0$ **using** *ereal_sgn_abs* **by** *auto*

moreover have ($(\lambda x. sgn(l) * f x) \longrightarrow (sgn(l) * l)$) F

by (*rule* *tendsto_cmult_ereal*, *auto simp: sgn_finite* *assms(1)*)

moreover have ($(\lambda x. sgn(m) * g x) \longrightarrow (sgn(m) * m)$) F

by (*rule* *tendsto_cmult_ereal*, *auto simp: sgn_finite* *assms(2)*)

ultimately have $*$: ($(\lambda x. (sgn(l) * f x) * (sgn(m) * g x)) \longrightarrow (sgn(l) * l) * (sgn(m) * m)$) F

using *tendsto_mult_ereal_pos* **by** *force*

have ($(\lambda x. (sgn(l) * sgn(m)) * ((sgn(l) * f x) * (sgn(m) * g x))) \longrightarrow (sgn(l) * sgn(m)) * ((sgn(l) * l) * (sgn(m) * m))$) F

by (*rule* *tendsto_cmult_ereal*, *auto simp: sgn_finite2* $*$)

moreover have $\bigwedge x. (sgn(l) * sgn(m)) * ((sgn(l) * f x) * (sgn(m) * g x)) = f x * g x$

by (*metis* *mult.left_neutral* *sgn_squared_ereal[OF <l ≠ 0>]* *sgn_squared_ereal[OF <m ≠ 0>]* *mult.assoc* *mult.commute*)

moreover have $(sgn(l) * sgn(m)) * ((sgn(l) * l) * (sgn(m) * m)) = l * m$

by (metis mult.left_neutral sgn_squared_ereal[OF ‹ $l \neq 0$ ›] sgn_squared_ereal[OF ‹ $m \neq 0$ ›] mult.assoc mult.commute)
 ultimately show ?thesis by auto
 qed

lemma tendsto_cmult_ereal_general [tendsto_intros]:
 fixes $f::_ \Rightarrow \text{ereal}$ and $c::\text{ereal}$
 assumes $(f \longrightarrow l) F \neg (l=0 \wedge \text{abs}(c) = \infty)$
 shows $((\lambda x. c * f x) \longrightarrow c * l) F$
 by (cases $c = 0$, auto simp: assms tendsto_mult_ereal)

Continuity of division

lemma tendsto_inverse_ereal_PInf:
 fixes $u::_ \Rightarrow \text{ereal}$
 assumes $(u \longrightarrow \infty) F$
 shows $((\lambda x. 1 / u x) \longrightarrow 0) F$
 proof -
 {
 fix $e::\text{real}$ assume $e > 0$
 have $1/e < \infty$ by auto
 then have eventually $(\lambda n. u n > 1/e) F$ using assms(1) by (simp add: tendsto_PInf)
 moreover
 {
 fix $z::\text{ereal}$ assume $z > 1/e$
 then have $z > 0$ using ‹ $e > 0$ › using less_le_trans not_le by fastforce
 then have $1/z \geq 0$ by auto
 moreover have $1/z < e$
 proof (cases z)
 case (real r)
 then show ?thesis
 using ‹ $0 < e$ › ‹ $0 < z$ › ‹ $\text{ereal}(1/e) < z$ › divide_less_eq_ereal_divide_less_pos
 by fastforce
 qed (use ‹ $0 < e$ › ‹ $0 < z$ › in auto)
 ultimately have $1/z \geq 0 \wedge 1/z < e$ by auto
 }
 ultimately have eventually $(\lambda n. 1/u n < e) F$ eventually $(\lambda n. 1/u n \geq 0) F$ by (auto simp: eventually_mono)
 } note * = this
 show ?thesis
 proof (subst order_tendsto_iff, auto)
 fix $a::\text{ereal}$ assume $a < 0$
 then show eventually $(\lambda n. 1/u n > a) F$ using *(2) eventually_mono
 less_le_trans linordered_field_no_ub by fastforce
 next
 fix $a::\text{ereal}$ assume $a > 0$
 then obtain $e::\text{real}$ where $e > 0 \wedge a > e$ using ereal_dense2_ereal_less(2) by blast
 then have eventually $(\lambda n. 1/u n < e) F$ using *(1) by auto

then show *eventually* $(\lambda n. 1/u\ n < a)$ *F* **using** $\langle a \rangle e$ **by** $(metis\ (mono_tags,$
lifting) *eventually_mono less_trans)*
qed
qed

The next lemma deserves to exist by itself, as it is so common and useful.

lemma *tendsto_inverse_real* [*tendsto_intros*]:
fixes $u::_ \Rightarrow real$
shows $(u \longrightarrow l)$ *F* $\implies l \neq 0 \implies ((\lambda x. 1/ u\ x) \longrightarrow 1/l)$ *F*
using *tendsto_inverse unfolding inverse_eq_divide* .

lemma *tendsto_inverse_ereal* [*tendsto_intros*]:
fixes $u::_ \Rightarrow ereal$
assumes $(u \longrightarrow l)$ *F* $l \neq 0$
shows $((\lambda x. 1/ u\ x) \longrightarrow 1/l)$ *F*
proof (*cases l*)
case (*real r*)
then have $r \neq 0$ **using** *assms(2)* **by** *auto*
then have $1/l = ereal(1/r)$ **using** *real* **by** $(simp\ add: one_ereal_def)$
define *v* **where** $v = (\lambda n. real_of_ereal(u\ n))$
have *ureal*: *eventually* $(\lambda n. u\ n = ereal(v\ n))$ *F* **unfolding** *v_def* **using** *real_lim_then_eventually_real*
assms(1) *real* **by** *auto*
then have $((\lambda n. ereal(v\ n)) \longrightarrow ereal\ r)$ *F* **using** *assms* *real_v_def* **by** *auto*
then have $*$: $((\lambda n. v\ n) \longrightarrow r)$ *F* **by** *auto*
then have $((\lambda n. 1/v\ n) \longrightarrow 1/r)$ *F* **using** $\langle r \neq 0 \rangle$ *tendsto_inverse_real* **by**
auto
then have *lim*: $((\lambda n. ereal(1/v\ n)) \longrightarrow 1/l)$ *F* **using** $\langle 1/l = ereal(1/r) \rangle$ **by**
auto

have $r \in -\{0\}$ *open* $(-\{(0::real)\})$ **using** $\langle r \neq 0 \rangle$ **by** *auto*
then have *eventually* $(\lambda n. v\ n \in -\{0\})$ *F* **using** $*$ **using** *topological_tendstoD*
by *blast*
then have *eventually* $(\lambda n. v\ n \neq 0)$ *F* **by** *auto*
moreover
{
fix *n* **assume** *H*: $v\ n \neq 0$ $u\ n = ereal(v\ n)$
then have $ereal(1/v\ n) = 1/ereal(v\ n)$ **by** $(simp\ add: one_ereal_def)$
then have $ereal(1/v\ n) = 1/u\ n$ **using** *H(2)* **by** *simp*
}
ultimately have *eventually* $(\lambda n. ereal(1/v\ n) = 1/u\ n)$ *F* **using** *ureal* *eventually_elim2* **by** *force*
with *Lim_transform_eventually[OF lim this]* **show** *?thesis* **by** *simp*
next
case (*PInf*)
then have $1/l = 0$ **by** *auto*
then show *?thesis* **using** *tendsto_inverse_ereal_PInf* *assms* *PInf* **by** *auto*
next
case (*MInf*)
then have $1/l = 0$ **by** *auto*


```

have  $1/z = -1/ -z$  if  $z < 0$  for  $z::ereal$ 
  apply (cases  $z$ ) using divide_ereal_def ‹  $z < 0$  › by auto
moreover have eventually  $(\lambda n. u\ n < 0)$   $F$  by (metis (no_types) MInf_assms(1))
tendsto_MInfTy zero_ereal_def)
ultimately have  $*$ : eventually  $(\lambda n. -1/-u\ n = 1/u\ n)$   $F$  by (simp add: eventually_mono)

define  $v$  where  $v = (\lambda n. -u\ n)$ 
have  $(v \longrightarrow \infty)$   $F$  unfolding  $v\_def$  using MInf_assms(1) tendsto_uminus_ereal
by fastforce
then have  $((\lambda n. 1/v\ n) \longrightarrow 0)$   $F$  using tendsto_inverse_ereal_PInf by auto
then have  $((\lambda n. -1/v\ n) \longrightarrow 0)$   $F$  using tendsto_uminus_ereal by fastforce
then show ?thesis unfolding v_def using Lim_transform_eventually[OF _ *]
  ‹  $1/l = 0$  › by auto
qed

```

```

lemma tendsto_divide_ereal [tendsto_intros]:
  fixes  $f\ g::\_ \Rightarrow ereal$ 
  assumes  $(f \longrightarrow l)$   $F$   $(g \longrightarrow m)$   $F$   $m \neq 0$   $\neg(abs\ l = \infty \wedge abs\ m = \infty)$ 
  shows  $((\lambda x. f\ x / g\ x) \longrightarrow l / m)$   $F$ 
proof -
  define  $h$  where  $h = (\lambda x. 1 / g\ x)$ 
  have  $*$ :  $(h \longrightarrow 1/m)$   $F$  unfolding  $h\_def$  using assms(2) assms(3) tendsto_inverse_ereal by auto
  have  $((\lambda x. f\ x * h\ x) \longrightarrow l * (1/m))$   $F$ 
  apply (rule tendsto_mult_ereal[OF assms(1) *]) using assms(3) assms(4) by
    (auto simp: divide_ereal_def)
  moreover have  $f\ x * h\ x = f\ x / g\ x$  for  $x$  unfolding  $h\_def$  by (simp add: divide_ereal_def)
  moreover have  $l * (1/m) = l/m$  by (simp add: divide_ereal_def)
  ultimately show ?thesis unfolding h_def using Lim_transform_eventually
by auto
qed

```

Further limits

The assumptions of $\llbracket |?x| \neq \infty; |?y| \neq \infty; (?f \longrightarrow ?x) ?F; (?g \longrightarrow ?y) ?F \rrbracket \implies ((\lambda x. ?f\ x - ?g\ x) \longrightarrow ?x - ?y) ?F$ are too strong, we weaken them here.

```

lemma tendsto_diff_ereal_general [tendsto_intros]:
  fixes  $u\ v::'a \Rightarrow ereal$ 
  assumes  $(u \longrightarrow l)$   $F$   $(v \longrightarrow m)$   $F$   $\neg((l = \infty \wedge m = \infty) \vee (l = -\infty \wedge m = -\infty))$ 
  shows  $((\lambda n. u\ n - v\ n) \longrightarrow l - m)$   $F$ 
proof -
  have  $\infty = l \longrightarrow ((\lambda a. u\ a + -v\ a) \longrightarrow l + -m)$   $F$ 
  by (metis (no_types) assms_ereal_uminus_uminus tendsto_add_ereal_general tendsto_uminus_ereal)
  then have  $((\lambda n. u\ n + (-v\ n)) \longrightarrow l + (-m))$   $F$ 

```

1750

```

    by (simp add: assms ereal_uminus_eq_reorder tendsto_add_ereal_general)
  then show ?thesis
    by (simp add: minus_ereal_def)
qed

```

```

lemma id_nat_ereal_tendsto_PInf [tendsto_intros]:
  ( $\lambda n::\text{nat}. \text{ereal } n$ )  $\longrightarrow$   $\infty$ 
by (simp add: filterlim_real_sequentially tendsto_PInf_eq_at_top)

```

```

lemma tendsto_at_top_pseudo_inverse [tendsto_intros]:
  fixes  $u::\text{nat} \Rightarrow \text{nat}$ 
  assumes LIM  $n$  sequentially.  $u \ n \ \> \ \text{at\_top}$ 
  shows LIM  $n$  sequentially.  $\text{Inf } \{N. u \ N \geq n\} \ \> \ \text{at\_top}$ 
proof -
  {
    fix  $C::\text{nat}$ 
    define  $M$  where  $M = \text{Max } \{u \ n \mid n. n \leq C\} + 1$ 
    {
      fix  $n$  assume  $n \geq M$ 
      have eventually ( $\lambda N. u \ N \geq n$ ) sequentially using assms
        by (simp add: filterlim_at_top)
      then have *:  $\{N. u \ N \geq n\} \neq \{\}$  by force

      have  $N > C$  if  $u \ N \geq n$  for  $N$ 
      proof (rule ccontr)
        assume  $\neg(N > C)$ 
        then have  $u \ N \leq \text{Max } \{u \ n \mid n. n \leq C\}$ 
          using Max_ge by (simp add: setcompr_eq_image not_less)
        then show False using  $\langle u \ N \geq n \rangle \langle n \geq M \rangle$  unfolding M_def by auto
      qed
      then have **:  $\{N. u \ N \geq n\} \subseteq \{C..\}$  by fastforce
      have  $\text{Inf } \{N. u \ N \geq n\} \geq C$ 
        by (metis * ** Inf_nat_def1 atLeast_iff subset_eq)
    }
    then have eventually ( $\lambda n. \text{Inf } \{N. u \ N \geq n\} \geq C$ ) sequentially
      using eventually_sequentially by auto
  }
  then show ?thesis using filterlim_at_top by auto
qed

```

```

lemma pseudo_inverse_finite_set:
  fixes  $u::\text{nat} \Rightarrow \text{nat}$ 
  assumes LIM  $n$  sequentially.  $u \ n \ \> \ \text{at\_top}$ 
  shows finite  $\{N. u \ N \leq n\}$ 
proof -
  fix  $n$ 
  have eventually ( $\lambda N. u \ N \geq n+1$ ) sequentially using assms
    by (simp add: filterlim_at_top)
  then obtain  $N1$  where  $N1: \bigwedge N. N \geq N1 \implies u \ N \geq n + 1$ 

```

```

using eventually_sequentially by auto
have  $\{N. u N \leq n\} \subseteq \{..<N1\}$ 
  by (metis (no_types, lifting) N1 Suc_eq_plus1 lessThan_iff less_le_not_le
mem_Collect_eq nle_le not_less_eq_eq subset_eq)
then show finite  $\{N. u N \leq n\}$  by (simp add: finite_subset)
qed

```

```

lemma tendsto_at_top_pseudo_inverse2 [tendsto_intros]:
  fixes  $u::nat \Rightarrow nat$ 
  assumes LIM  $n$  sequentially.  $u n \rightarrow at\_top$ 
  shows LIM  $n$  sequentially.  $Max \{N. u N \leq n\} \rightarrow at\_top$ 
proof -
  {
    fix  $N0::nat$ 
    have  $N0 \leq Max \{N. u N \leq n\}$  if  $n \geq u N0$  for  $n$ 
      by (simp add: assms pseudo_inverse_finite_set that)
    then have eventually  $(\lambda n. N0 \leq Max \{N. u N \leq n\})$  sequentially
      using eventually_sequentially by blast
  }
  then show ?thesis using filterlim_at_top by auto
qed

```

```

lemma ereal_truncation_top [tendsto_intros]:
  fixes  $x::ereal$ 
  shows  $(\lambda n::nat. min x n) \longrightarrow x$ 
proof (cases  $x$ )
  case (real  $r$ )
    then obtain  $K::nat$  where  $K>0$   $K > abs(r)$  using reals_Archimedean2 gr0I
  by auto
    then have  $min x n = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real, auto)
  using that eq_iff by fastforce
    then have eventually  $(\lambda n. min x n = x)$  sequentially using eventually_at_top_linorder
  by blast
    then show ?thesis by (simp add: tendsto_eventually)
  next
    case (PInf)
    then have  $min x n = n$  for  $n::nat$  by (auto simp: min_def)
    then show ?thesis using id_nat_ereal_tendsto_PInf PInf by auto
  next
    case (MInf)
    then have  $min x n = x$  for  $n::nat$  by (auto simp: min_def)
    then show ?thesis by auto
qed

```

```

lemma ereal_truncation_real_top [tendsto_intros]:
  fixes  $x::ereal$ 
  assumes  $x \neq -\infty$ 
  shows  $(\lambda n::nat. real\_of\_ereal(min x n)) \longrightarrow x$ 
proof (cases  $x$ )

```

```

    case (real r)
    then obtain  $K::nat$  where  $K > 0$   $K > abs(r)$  using reals_Archimedean2 gr0I
  by auto
    then have  $min\ x\ n = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real, auto)
  using that eq_iff by fastforce
    then have  $real\_of\_ereal(min\ x\ n) = r$  if  $n \geq K$  for  $n$  using real that by auto
    then have eventually  $(\lambda n. real\_of\_ereal(min\ x\ n) = r)$  sequentially using eventually_at_top_linorder by blast
    then have  $(\lambda n. real\_of\_ereal(min\ x\ n)) \longrightarrow r$  by (simp add: tendsto_eventually)
    then show ?thesis using real by auto
  next
    case (PInf)
    then have  $real\_of\_ereal(min\ x\ n) = n$  for  $n::nat$  by (auto simp: min_def)
    then show ?thesis using id_nat_ereal_tendsto_PInf PInf by auto
  qed (simp add: assms)

```

lemma *ereal_truncation_bottom* [*tendsto_intros*]:

```

  fixes  $x::ereal$ 
  shows  $(\lambda n::nat. max\ x\ (-\ real\ n)) \longrightarrow x$ 
  proof (cases x)
    case (real r)
    then obtain  $K::nat$  where  $K > 0$   $K > abs(r)$  using reals_Archimedean2 gr0I
  by auto
    then have  $max\ x\ (-\ real\ n) = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real, auto)
  using that eq_iff by fastforce
    then have eventually  $(\lambda n. max\ x\ (-\ real\ n) = x)$  sequentially using eventually_at_top_linorder by blast
    then show ?thesis by (simp add: tendsto_eventually)
  next
    case (MInf)
    then have  $max\ x\ (-\ real\ n) = (-1)*\ eral(real\ n)$  for  $n::nat$  by (auto simp: max_def)
    moreover have  $(\lambda n. (-1)*\ eral(real\ n)) \longrightarrow -\infty$ 
    using tendsto_cmult_ereal[of  $-1$ , OF id_nat_ereal_tendsto_PInf] by
  (simp add: one_ereal_def)
    ultimately show ?thesis using MInf by auto
  next
    case (PInf)
    then have  $max\ x\ (-\ real\ n) = x$  for  $n::nat$  by (auto simp: max_def)
    then show ?thesis by auto
  qed

```

lemma *ereal_truncation_real_bottom* [*tendsto_intros*]:

```

  fixes  $x::ereal$ 
  assumes  $x \neq \infty$ 
  shows  $(\lambda n::nat. real\_of\_ereal(max\ x\ (-\ real\ n))) \longrightarrow x$ 
  proof (cases x)
    case (real r)
    then obtain  $K::nat$  where  $K > 0$   $K > abs(r)$  using reals_Archimedean2 gr0I

```

```

by auto
  then have  $\max x (-real\ n) = x$  if  $n \geq K$  for  $n$  apply (subst real, subst real,
  auto) using that eq_iff by fastforce
  then have  $real\_of\_ereal(\max x (-real\ n)) = r$  if  $n \geq K$  for  $n$  using real that
  by auto
  then have eventually  $(\lambda n. real\_of\_ereal(\max x (-real\ n)) = r)$  sequentially
  using eventually_at_top_linorder by blast
  then have  $(\lambda n. real\_of\_ereal(\max x (-real\ n))) \longrightarrow r$  by (simp add: tend-
  sto_eventually)
  then show ?thesis using real by auto
next
case (MInf)
  then have  $real\_of\_ereal(\max x (-real\ n)) = (-1)*\ ereal(real\ n)$  for  $n::nat$  by
  (auto simp: max_def)
  moreover have  $(\lambda n. (-1)*\ ereal(real\ n)) \longrightarrow -\infty$ 
  using tendsto_cmult_ereal[of -1, OF _ id_nat_ereal_tendsto_PInf] by
  (simp add: one_ereal_def)
  ultimately show ?thesis using MInf by auto
qed (simp add: assms)

```

the next one is copied from *tendsto_sum*.

```

lemma tendsto_sum_ereal [tendsto_intros]:
  fixes  $f :: 'a \Rightarrow 'b \Rightarrow ereal$ 
  assumes  $\bigwedge i. i \in S \implies (f\ i \longrightarrow a\ i)\ F$ 
  and  $\bigwedge i. abs(a\ i) \neq \infty$ 
  shows  $((\lambda x. \sum_{i \in S}. f\ i\ x) \longrightarrow (\sum_{i \in S}. a\ i))\ F$ 
proof (cases finite S)
  assume finite S then show ?thesis using assms
  by (induct, simp, simp add: tendsto_add_ereal_general2 assms)
qed (simp)

```

```

lemma continuous_ereal_abs:
  continuous_on (UNIV::ereal set) abs
proof -
  have continuous_on  $(\{..0\} \cup \{(0::ereal)..\})\ abs$ 
  proof (intro continuous_on_closed_Un continuous_intros)
    show continuous_on  $\{..0\}\ abs$ 
    by (metis abs_ereal_ge0 abs_ereal_less0 continuous_on_eq_antisym_conv1
    atMost_iff continuous_uminus_ereal_ereal_uminus_zero)
    show continuous_on  $\{0::ereal..\}\ abs$ 
    by (metis abs_ereal_ge0 atLeast_iff continuous_on_eq_continuous_on_id)
  qed
  moreover have  $(UNIV::ereal\ set) = \{..0\} \cup \{(0::ereal)..\}$  by auto
  ultimately show ?thesis by auto
qed

```

```

lemmas continuous_on_compose_ereal_abs [continuous_intros] =
  continuous_on_compose2 [OF continuous_ereal_abs _ subset_UNIV]

```

lemma *tendsto_abs_ereal* [*tendsto_intros*]:
assumes $(u \longrightarrow (l::ereal)) F$
shows $((\lambda n. \text{abs}(u\ n)) \longrightarrow \text{abs } l) F$
using *continuous_ereal_abs* *assms* **by** (*metis UNIV_I continuous_on tendsto_compose*)

lemma *ereal_minus_real_tendsto_MInf* [*tendsto_intros*]:
 $(\lambda x. \text{ereal } (- \text{real } x)) \longrightarrow - \infty$
by (*subst uminus_ereal.simps(1)[symmetric]*, *intro tendsto_intros*)

6.13.2 Extended-Nonnegative-Real.thy

lemma *tendsto_diff_ennreal_general* [*tendsto_intros*]:
fixes $u\ v::'a \Rightarrow \text{ennreal}$
assumes $(u \longrightarrow l) F (v \longrightarrow m) F \neg(l = \infty \wedge m = \infty)$
shows $((\lambda n. u\ n - v\ n) \longrightarrow l - m) F$
proof –
have $((\lambda n. \text{e2ennreal}(\text{enn2ereal}(u\ n) - \text{enn2ereal}(v\ n))) \longrightarrow \text{e2ennreal}(\text{enn2ereal } l - \text{enn2ereal } m)) F$
by (*intro tendsto_intros*) (*use assms in auto*)
then show *?thesis* **by** *auto*
qed

lemma *tendsto_mult_ennreal* [*tendsto_intros*]:
fixes $l\ m::\text{ennreal}$
assumes $(u \longrightarrow l) F (v \longrightarrow m) F \neg((l = 0 \wedge m = \infty) \vee (l = \infty \wedge m = 0))$
shows $((\lambda n. u\ n * v\ n) \longrightarrow l * m) F$
proof –
have $((\lambda n. \text{e2ennreal}(\text{enn2ereal}(u\ n) * \text{enn2ereal}(v\ n))) \longrightarrow \text{e2ennreal}(\text{enn2ereal } l * \text{enn2ereal } m)) F$
by (*intro tendsto_intros*) (*use assms enn2ereal_inject zero_ennreal.rep_eq in fastforce*)
moreover have $\text{e2ennreal}(\text{enn2ereal}(u\ n) * \text{enn2ereal}(v\ n)) = u\ n * v\ n$ **for** n
by (*subst times_ennreal.abs_eq[symmetric]*, *auto simp: eq_onp_same_args*)
moreover have $\text{e2ennreal}(\text{enn2ereal } l * \text{enn2ereal } m) = l * m$
by (*subst times_ennreal.abs_eq[symmetric]*, *auto simp: eq_onp_same_args*)
ultimately show *?thesis*
by *auto*
qed

6.13.3 monoset

definition (*in order*) *mono_set*:
 $\text{mono_set } S \iff (\forall x\ y. x \leq y \longrightarrow x \in S \longrightarrow y \in S)$

lemma (*in order*) *mono_greaterThan* [*intro*, *simp*]: *mono_set* $\{B<..\}$ **unfolding** *mono_set* **by** *auto*

lemma (*in order*) *mono_atLeast* [*intro*, *simp*]: *mono_set* $\{B..\}$ **unfolding** *mono_set* **by** *auto*

lemma (in order) *mono_UNIV* [intro, simp]: *mono_set UNIV unfolding mono_set*
by *auto*

lemma (in order) *mono_empty* [intro, simp]: *mono_set {} unfolding mono_set*
by *auto*

lemma (in *complete_linorder*) *mono_set_iff*:

fixes *S* :: 'a set

defines *a* \equiv *Inf S*

shows *mono_set S* \longleftrightarrow *S* = {*a* <..<} \vee *S* = {*a*..<} (**is** *_* = ?*c*)

proof

assume *mono_set S*

then have *mono*: $\bigwedge x y. x \leq y \implies x \in S \implies y \in S$

by (*auto simp: mono_set*)

show ?*c*

proof *cases*

assume *a* $\in S$

show ?*c*

using *mono*[*OF _* <*a* $\in S$ >]

by (*auto intro: Inf_lower simp: a_def*)

next

assume *a* $\notin S$

have *S* = {*a* <..<}

proof *safe*

fix *x* **assume** *x* $\in S$

then have *a* $\leq x$

unfolding *a_def* **by** (*rule Inf_lower*)

then show *a* < *x*

using <*x* $\in S$ > <*a* $\notin S$ > **by** (*cases a = x*) *auto*

next

fix *x* **assume** *a* < *x*

then obtain *y* **where** *y* < *x* *y* $\in S$

unfolding *a_def Inf_less_iff* ..

with *mono*[*of y x*] **show** *x* $\in S$

by *auto*

qed

then show ?*c* ..

qed

qed *auto*

lemma *ereal_open_mono_set*:

fixes *S* :: *ereal set*

shows *open S* \wedge *mono_set S* \longleftrightarrow *S* = *UNIV* \vee *S* = {*Inf S* <..<}

by (*metis Inf_UNIV atLeast_eq_UNIV_iff ereal_open_atLeast*
ereal_open_closed mono_set_iff open_ereal_greaterThan)

lemma *ereal_closed_mono_set*:

fixes *S* :: *ereal set*

shows *closed S* \wedge *mono_set S* \longleftrightarrow *S* = {} \vee *S* = {*Inf S* ..}

by (*metis Inf_UNIV atLeast_eq_UNIV_iff closed_ereal_atLeast*)

ereal_open_closed mono_empty mono_set_iff open_ereal_greaterThan)

lemma *ereal_Liminf_Sup_monoset:*

fixes $f :: 'a \Rightarrow \text{ereal}$

shows $\text{Liminf } \text{net } f =$

$\text{Sup } \{l. \forall S. \text{open } S \longrightarrow \text{mono_set } S \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f x \in S)$
net}

(is $_ = \text{Sup } ?A)$

proof (*safe intro!*: $\text{Liminf_eqI complete_lattice_class.Sup_upper complete_lattice_class.Sup_least}$)

fix P

assume $P: \text{eventually } P \text{ net}$

fix S

assume $S: \text{mono_set } S \text{ Inf } (f' (\text{Collect } P)) \in S$

{

fix x

assume $P x$

then have $\text{Inf } (f' (\text{Collect } P)) \leq f x$

by (*intro complete_lattice_class.INF_lower*) *simp*

with S **have** $f x \in S$

by (*simp add: mono_set*)

}

with P **show** $\text{eventually } (\lambda x. f x \in S) \text{ net}$

by (*auto elim: eventually_mono*)

next

fix $y l$

assume $S: \forall S. \text{open } S \longrightarrow \text{mono_set } S \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x. f x \in S)$

net

assume $P: \forall P. \text{eventually } P \text{ net} \longrightarrow \text{Inf } (f' (\text{Collect } P)) \leq y$

show $l \leq y$

proof (*rule dense_le*)

fix B

assume $B < l$

then have $\text{eventually } (\lambda x. f x \in \{B <..\}) \text{ net}$

by (*intro S[rule_format]*) *auto*

then have $\text{Inf } (f' \{x. B < f x\}) \leq y$

using P **by** *auto*

moreover have $B \leq \text{Inf } (f' \{x. B < f x\})$

by (*intro INF_greatest*) *auto*

ultimately show $B \leq y$

by *simp*

qed

qed

lemma *ereal_Limsup_Inf_monoset:*

fixes $f :: 'a \Rightarrow \text{ereal}$

shows $\text{Limsup } \text{net } f =$

$\text{Inf } \{l. \forall S. \text{open } S \longrightarrow \text{mono_set } (\text{uminus } ' S) \longrightarrow l \in S \longrightarrow \text{eventually } (\lambda x.$
f x \in S) net}

(is $_ = \text{Inf } ?A)$


```

proof (safe intro!: Limsup_eqI complete_lattice_class.Inf_lower complete_lattice_class.Inf_greatest)
  fix P
  assume P: eventually P net
  fix S
  assume S: mono_set (uminus' S) Sup (f ' (Collect P)) ∈ S
  {
    fix x
    assume P x
    then have f x ≤ Sup (f ' (Collect P))
      by (intro complete_lattice_class.SUP_upper) simp
    with S(1)[unfolded mono_set, rule_format, of - Sup (f ' (Collect P)) - f x]
  S(2)
    have f x ∈ S
      by (simp add: inj_image_mem_iff) }
  with P show eventually (λx. f x ∈ S) net
    by (auto elim: eventually_mono)
next
  fix y l
  assume S: ∀ S. open S → mono_set (uminus ' S) → l ∈ S → eventually
(λx. f x ∈ S) net
  assume P: ∀ P. eventually P net → y ≤ Sup (f ' (Collect P))
  show y ≤ l
  proof (rule dense_ge)
    fix B
    assume l < B
    then have eventually (λx. f x ∈ {..< B}) net
      by (intro S[rule_format]) auto
    then have y ≤ Sup (f ' {x. f x < B})
      using P by auto
    moreover have Sup (f ' {x. f x < B}) ≤ B
      by (intro SUP_least) auto
    ultimately show y ≤ B
      by simp
  qed
qed

lemma liminf_bounded_open:
  fixes x :: nat ⇒ ereal
  shows x0 ≤ liminf x ↔ (∀ S. open S → mono_set S → x0 ∈ S → (∃ N.
∀ n ≥ N. x n ∈ S))
  (is _ ↔ ?P x0)
proof
  assume ?P x0
  then show x0 ≤ liminf x
    unfolding ereal_Liminf_Sup_monoset eventually_sequentially
    by (intro complete_lattice_class.Sup_upper) auto
next
  assume x0 ≤ liminf x
  {

```

```

fix S :: ereal set
assume om: open S mono_set S x0 ∈ S
then have ∃ N. ∀ n ≥ N. x n ∈ S
  by (metis ‹x0 ≤ liminf x› ereal_open_mono_set greaterThan_iff lim-
inf_bounded_iff om UNIV_I)
}
then show ?P x0
  by auto
qed

```

lemma *limsup_finite_then_bounded*:

```

fixes u::nat ⇒ real
assumes limsup u < ∞
shows ∃ C. ∀ n. u n ≤ C
proof –
obtain C where C: limsup u < C C < ∞ using assms ereal_dense2 by blast
then have C = ereal(real_of_ereal C) using ereal_real by force
have eventually (λn. u n < C) sequentially
  using SUP_lessD eventually_mono C(1)
  by (fastforce simp: INF_less_iff Limsup_def)
then obtain N where N: ∧n. n ≥ N ⇒ u n < C using eventually_sequentially
by auto
define D where D = max (real_of_ereal C) (Max {u n | n. n ≤ N})
have ∧n. u n ≤ D
proof –
fix n show u n ≤ D
proof (cases)
  assume *: n ≤ N
  have u n ≤ Max {u n | n. n ≤ N} by (rule Max_ge, auto simp: *)
  then show u n ≤ D unfolding D_def by linarith
next
  assume ¬(n ≤ N)
  then have n ≥ N by simp
  then have u n < C using N by auto
  then have u n < real_of_ereal C using ‹C = ereal(real_of_ereal C)›
less_ereal.simps(1) by fastforce
  then show u n ≤ D unfolding D_def by linarith
qed
qed
then show ?thesis by blast
qed

```

lemma *liminf_finite_then_bounded_below*:

```

fixes u::nat ⇒ real
assumes liminf u > -∞
shows ∃ C. ∀ n. u n ≥ C
proof –
obtain C where C: liminf u > C C > -∞ using assms using ereal_dense2
by blast

```

```

then have  $C = \text{ereal}(\text{real\_of\_ereal } C)$  using ereal_real by force
have eventually  $(\lambda n. u\ n > C)$  sequentially
  using eventually_elim2 less_INF_D C(1)
  by (fastforce simp: less_SUP_iff Liminf_def)
then obtain  $N$  where  $N: \bigwedge n. n \geq N \implies u\ n > C$  using eventually_sequentially
by auto
define  $D$  where  $D = \min(\text{real\_of\_ereal } C) (\text{Min } \{u\ n \mid n. n \leq N\})$ 
have  $\bigwedge n. u\ n \geq D$ 
proof -
  fix  $n$  show  $u\ n \geq D$ 
  proof (cases)
    assume  $*$ :  $n \leq N$ 
    have  $u\ n \geq \text{Min } \{u\ n \mid n. n \leq N\}$  by (rule Min_le, auto simp: *)
    then show  $u\ n \geq D$  unfolding  $D\_def$  by linarith
  next
    assume  $\neg(n \leq N)$ 
    then have  $n \geq N$  by simp
    then have  $u\ n > C$  using  $N$  by auto
    then have  $u\ n > \text{real\_of\_ereal } C$ 
      using  $\langle C = \text{ereal}(\text{real\_of\_ereal } C) \rangle$  less_ereal.simps(1) by fastforce
    then show  $u\ n \geq D$  unfolding  $D\_def$  by linarith
  qed
qed
then show ?thesis by blast
qed

```

lemma *liminf_upper_bound*:

```

fixes  $u:: \text{nat} \Rightarrow \text{ereal}$ 
assumes liminf  $u < l$ 
shows  $\exists N > k. u\ N < l$ 

```

by (*metis assms gt_ex less_le_trans liminf_bounded_iff not_less*)

lemma *limsup_shift*:

```

limsup  $(\lambda n. u\ (n+1)) = \text{limsup } u$ 

```

proof -

```

have  $(\text{SUP } m \in \{n+1..\}. u\ m) = (\text{SUP } m \in \{n..\}. u\ (m+1))$  for  $n$ 

```

```

  by (rule SUP_eq) (use Suc_le_D in auto)

```

```

then have  $a: (\text{INF } n. \text{SUP } m \in \{n..\}. u\ (m+1)) = (\text{INF } n. (\text{SUP } m \in \{n+1..\}. u\ m))$  by auto

```

```

have  $b: (\text{INF } n. (\text{SUP } m \in \{n+1..\}. u\ m)) = (\text{INF } n \in \{1..\}. (\text{SUP } m \in \{n..\}. u\ m))$ 

```

```

  by (rule INF_eq) (use Suc_le_D in auto)

```

```

have  $(\text{INF } n \in \{1..\}. v\ n) = (\text{INF } n. v\ n)$  if decseq  $v$  for  $v:: \text{nat} \Rightarrow 'a$ 

```

```

  by (rule INF_eq) (use <decseq v> decseq_Suc_iff in auto)

```

```

moreover have decseq  $(\lambda n. (\text{SUP } m \in \{n..\}. u\ m))$  by (simp add: SUP_subset_mono decseq_def)

```

```

ultimately have  $c: (\text{INF } n \in \{1..\}. (\text{SUP } m \in \{n..\}. u\ m)) = (\text{INF } n. (\text{SUP } m \in \{n..\}. u\ m))$  by simp

```

```

have  $(\text{INF } n. \text{Sup } (u\ ' \{n..\})) = (\text{INF } n. \text{SUP } m \in \{n..\}. u\ (m+1))$  using  $a\ b$ 

```

1760

c by *simp*
then show *?thesis* by (auto cong: *limsup_INF_SUP*)
qed

lemma *limsup_shift_k*:
 $\text{limsup } (\lambda n. u (n+k)) = \text{limsup } u$
proof (induction *k*)
 case (Suc *k*)
 have $\text{limsup } (\lambda n. u (n+k+1)) = \text{limsup } (\lambda n. u (n+k))$ using *limsup_shift*[where
 ?u= $\lambda n. u(n+k)$] by *simp*
 then show *?case* using *Suc.IH* by *simp*
qed (auto)

lemma *liminf_shift*:
 $\text{liminf } (\lambda n. u (n+1)) = \text{liminf } u$
proof –
 have $(\text{INF } m \in \{n+1..\}. u m) = (\text{INF } m \in \{n..\}. u (m + 1))$ for *n*
 by (rule *INF_eq*) (use *Suc_le_D* in auto)
 then have *a*: $(\text{SUP } n. \text{INF } m \in \{n..\}. u (m + 1)) = (\text{SUP } n. (\text{INF } m \in \{n+1..\}. u m))$ by auto
 have *b*: $(\text{SUP } n. (\text{INF } m \in \{n+1..\}. u m)) = (\text{SUP } n \in \{1..\}. (\text{INF } m \in \{n..\}. u m))$
 by (rule *SUP_eq*) (use *Suc_le_D* in auto)
 have $(\text{SUP } n \in \{1..\}. v n) = (\text{SUP } n. v n)$ if *incseq* *v* for *v*:*nat* \Rightarrow '*a*
 by (rule *SUP_eq*) (use *<incseq v>* *incseq_Suc_iff* in auto)
 moreover have *incseq* $(\lambda n. (\text{INF } m \in \{n..\}. u m))$ by (simp add: *INF_superset_mono*
 mono_def)
 ultimately have *c*: $(\text{SUP } n \in \{1..\}. (\text{INF } m \in \{n..\}. u m)) = (\text{SUP } n. (\text{INF } m \in \{n..\}. u m))$ by *simp*
 have $(\text{SUP } n. \text{Inf } (u \text{ ' } \{n..\})) = (\text{SUP } n. \text{INF } m \in \{n..\}. u (m + 1))$ using *a b*
 c by *simp*
 then show *?thesis* by (auto cong: *liminf_SUP_INF*)
qed

lemma *liminf_shift_k*:
 $\text{liminf } (\lambda n. u (n+k)) = \text{liminf } u$
proof (induction *k*)
 case (Suc *k*)
 have $\text{liminf } (\lambda n. u (n+k+1)) = \text{liminf } (\lambda n. u (n+k))$
 using *liminf_shift*[where *?u*= $\lambda n. u(n+k)$] by *simp*
 then show *?case* using *Suc.IH* by *simp*
qed (auto)

lemma *Limsup_obtain*:
 fixes *u*: $_ \Rightarrow$ '*a* :: *complete_linorder*
 assumes *Limsup* *F* *u* > *c*
 shows $\exists i. u i > c$
proof –
 have $(\text{INF } P \in \{P. \text{eventually } P F\}. \text{SUP } x \in \{x. P x\}. u x) > c$ using *assms* by

```
(simp add: Limsup_def)
  then show ?thesis by (metis eventually_True mem_Collect_eq less_INF_D
less_SUP_iff)
qed
```

The next lemma is extremely useful, as it often makes it possible to reduce statements about limsups to statements about limits.

```
lemma limsup_subseq_lim:
  fixes  $u::nat \Rightarrow 'a :: \{complete\_linorder, linorder\_topology\}$ 
  shows  $\exists r::nat \Rightarrow nat. strict\_mono\ r \wedge (u\ o\ r) \longrightarrow limsup\ u$ 
proof (cases)
  assume  $\forall n. \exists p > n. \forall m \geq p. u\ m \leq u\ p$ 
  then have  $\exists r. \forall n. (\forall m \geq r\ n. u\ m \leq u\ (r\ n)) \wedge r\ n < r\ (Suc\ n)$ 
    by (intro dependent_nat_choice) (auto simp: conj_commute)
  then obtain  $r :: nat \Rightarrow nat$  where  $strict\_mono\ r$  and  $mono: \bigwedge n\ m. r\ n \leq m \implies u\ m \leq u\ (r\ n)$ 
    by (auto simp: strict_mono_Suc_iff)
  define  $umax$  where  $umax = (\lambda n. (SUP\ m \in \{n..\}. u\ m))$ 
  have decseq  $umax$  unfolding  $umax\_def$  by (simp add: SUP_subset_mono antimonono_def)
  then have  $umax \longrightarrow limsup\ u$  unfolding  $umax\_def$  by (metis LIMSEQ_INF limsup_INF_SUP)
  then have  $*$ :  $(umax\ o\ r) \longrightarrow limsup\ u$  by (simp add: LIMSEQ_subseq_LIMSEQ <strict_mono_r>)
  have  $\bigwedge n. umax\ (r\ n) = u\ (r\ n)$  unfolding  $umax\_def$  using  $mono$ 
    by (metis SUP_le_iff antisym_atLeast_def mem_Collect_eq order_refl)
  then have  $umax\ o\ r = u\ o\ r$  unfolding  $o\_def$  by simp
  then have  $(u\ o\ r) \longrightarrow limsup\ u$  using  $*$  by simp
  then show ?thesis using <strict_mono_r> by blast
next
  assume  $\neg (\forall n. \exists p > n. (\forall m \geq p. u\ m \leq u\ p))$ 
  then obtain  $N$  where  $N: \bigwedge p. p > N \implies \exists m > p. u\ p < u\ m$  by (force simp: not_le le_less)
  have  $\exists r. \forall n. N < r\ n \wedge r\ n < r\ (Suc\ n) \wedge (\forall i \in \{N <..r\ (Suc\ n)\}. u\ i \leq u\ (r\ (Suc\ n)))$ 
    proof (rule dependent_nat_choice)
      fix  $x$  assume  $N < x$ 
      then have  $a: finite\ \{N <..x\}\ \{N <..x\} \neq \{\}$  by simp_all
      have  $Max\ \{u\ i\ | i. i \in \{N <..x\}\} \in \{u\ i\ | i. i \in \{N <..x\}\}$  apply (rule Max_in)
    using  $a$  by (auto)
      then obtain  $p$  where  $p \in \{N <..x\}$  and  $upmax: u\ p = Max\ \{u\ i\ | i. i \in \{N <..x\}\}$  by auto
      define  $U$  where  $U = \{m. m > p \wedge u\ p < u\ m\}$ 
      have  $U \neq \{\}$  unfolding  $U\_def$  using  $N[of\ p]\ \langle p \in \{N <..x\}\rangle$  by auto
      define  $y$  where  $y = Inf\ U$ 
      then have  $y \in U$  using <math>U \neq \{\}> by (simp add: Inf_nat_def1)
      have  $a: \bigwedge i. i \in \{N <..x\} \implies u\ i \leq u\ p$ 
      proof -
        fix  $i$  assume  $i \in \{N <..x\}$ 
```

```

    then have  $u\ i \in \{u\ i \mid i. i \in \{N<..x\}\}$  by blast
    then show  $u\ i \leq u\ p$  using upmax by simp
qed
moreover have  $u\ p < u\ y$  using  $\langle y \in U \rangle U\_def$  by auto
ultimately have  $y \notin \{N<..x\}$  using not_le by blast
moreover have  $y > N$  using  $\langle y \in U \rangle U\_def \langle p \in \{N<..x\}\rangle$  by auto
ultimately have  $y > x$  by auto

have  $\bigwedge i. i \in \{N<..y\} \implies u\ i \leq u\ y$ 
proof -
  fix i assume  $i \in \{N<..y\}$  show  $u\ i \leq u\ y$ 
  proof (cases)
    assume  $i = y$ 
    then show ?thesis by simp
  next
    assume  $\neg(i=y)$ 
    then have  $i: i \in \{N<.. using  $\langle i \in \{N<..y\}\rangle$  by simp
    have  $u\ i \leq u\ p$ 
    proof (cases)
      assume  $i \leq x$ 
      then have  $i \in \{N<..x\}$  using i by simp
      then show ?thesis using a by simp
    next
      assume  $\neg(i \leq x)$ 
      then have  $i > x$  by simp
      then have  $*$ :  $i > p$  using  $\langle p \in \{N<..x\}\rangle$  by simp
      have  $i < \text{Inf } U$  using i y_def by simp
      then have  $i \notin U$  using Inf_nat_def not_less_Least by auto
      then show ?thesis using U_def * by auto
    qed
    then show  $u\ i \leq u\ y$  using  $\langle u\ p < u\ y \rangle$  by auto
  qed
qed
then have  $N < y \wedge x < y \wedge (\forall i \in \{N<..y\}. u\ i \leq u\ y)$  using  $\langle y > x \rangle \langle y > N \rangle$  by auto
then show  $\exists y > N. x < y \wedge (\forall i \in \{N<..y\}. u\ i \leq u\ y)$  by auto
qed (auto)
then obtain r where  $r: \forall n. N < r\ n \wedge r\ n < r\ (\text{Suc } n) \wedge (\forall i \in \{N<..r\ (\text{Suc } n)\}. u\ i \leq u\ (r\ (\text{Suc } n)))$  by auto
have strict_mono r using r by (auto simp: strict_mono_Suc_iff)
have incseq (u o r) unfolding o_def using r by (simp add: incseq_SucI order.strict_implies_order)
then have  $(u\ o\ r) \longrightarrow (\text{SUP } n. (u\ o\ r)\ n)$  using LIMSEQ_SUP by blast
then have  $\text{limsup } (u\ o\ r) = (\text{SUP } n. (u\ o\ r)\ n)$  by (simp add: lim_imp_Limsup)
moreover have  $\text{limsup } (u\ o\ r) \leq \text{limsup } u$  using  $\langle \text{strict\_mono } r \rangle$  by (simp add: limsup_subseq_mono)
ultimately have  $(\text{SUP } n. (u\ o\ r)\ n) \leq \text{limsup } u$  by simp
{$ 
```

```

fix  $i$  assume  $i: i \in \{N<..\}$ 
  obtain  $n$  where  $i < r$  ( $Suc\ n$ ) using  $\langle strict\_mono\ r \rangle$  using  $Suc\_le\_eq$ 
seq\_suble by  $blast$ 
  then have  $i \in \{N<..r(Suc\ n)\}$  using  $i$  by  $simp$ 
  then have  $u\ i \leq u\ (r(Suc\ n))$  using  $r$  by  $simp$ 
  then have  $u\ i \leq (SUP\ n. (u\ o\ r)\ n)$  unfolding  $o\_def$  by ( $meson\ SUP\_upper2$ 
UNIV_I)
  }
  then have  $(SUP\ i \in \{N<..\}. u\ i) \leq (SUP\ n. (u\ o\ r)\ n)$  using  $SUP\_least$  by
blast
  then have  $limsup\ u \leq (SUP\ n. (u\ o\ r)\ n)$  unfolding  $Limsup\_def$ 
  by ( $metis\ (mono\_tags,\ lifting)\ INF\_lower2\ atLeast\_Suc\_greaterThan\ atLeast\_def$ 
eventually\_ge\_at\_top\ mem\_Collect\_eq)
  then have  $limsup\ u = (SUP\ n. (u\ o\ r)\ n)$  using  $\langle (SUP\ n. (u\ o\ r)\ n) \leq limsup\ u \rangle$ 
by  $simp$ 
  then have  $(u\ o\ r) \longrightarrow limsup\ u$  using  $\langle (u\ o\ r) \longrightarrow (SUP\ n. (u\ o\ r)\ n) \rangle$ 
by  $simp$ 
  then show  $?thesis$  using  $\langle strict\_mono\ r \rangle$  by  $auto$ 
qed

```

lemma $liminf_subseq_lim$:

```

fixes  $u::nat \Rightarrow 'a :: \{complete\_linorder,\ linorder\_topology\}$ 
shows  $\exists r::nat \Rightarrow nat. strict\_mono\ r \wedge (u\ o\ r) \longrightarrow liminf\ u$ 
proof ( $cases$ )
  assume  $\forall n. \exists p > n. \forall m \geq p. u\ m \geq u\ p$ 
  then have  $\exists r. \forall n. (\forall m \geq r\ n. u\ m \geq u\ (r\ n)) \wedge r\ n < r$  ( $Suc\ n$ )
  by ( $intro\ dependent\_nat\_choice$ ) ( $auto\ simp: conj\_commute$ )
  then obtain  $r :: nat \Rightarrow nat$  where  $strict\_mono\ r$  and  $mono: \bigwedge n\ m. r\ n \leq m$ 
 $\implies u\ m \geq u\ (r\ n)$ 
  by ( $auto\ simp: strict\_mono\_Suc\_iff$ )
  define  $umin$  where  $umin = (\lambda n. (INF\ m \in \{n..\}. u\ m))$ 
  have  $incseq\ umin$  unfolding  $umin\_def$  by ( $simp\ add: INF\_superset\_mono$ 
incseq\_def)
  then have  $umin \longrightarrow liminf\ u$  unfolding  $umin\_def$  by ( $metis\ LIMSEQ\_SUP$ 
liminf\_SUP\_INF)
  then have  $*(umin\ o\ r) \longrightarrow liminf\ u$  by ( $simp\ add: LIMSEQ\_subseq\_LIMSEQ$ 
 $\langle strict\_mono\ r \rangle$ )
  have  $\bigwedge n. umin(r\ n) = u(r\ n)$  unfolding  $umin\_def$  using  $mono$ 
  by ( $metis\ le\_INF\_iff\ antisym\ atLeast\_def\ mem\_Collect\_eq\ order\_refl$ )
  then have  $umin\ o\ r = u\ o\ r$  unfolding  $o\_def$  by  $simp$ 
  then have  $(u\ o\ r) \longrightarrow liminf\ u$  using  $*$  by  $simp$ 
  then show  $?thesis$  using  $\langle strict\_mono\ r \rangle$  by  $blast$ 
next
  assume  $\neg (\forall n. \exists p > n. (\forall m \geq p. u\ m \geq u\ p))$ 
  then obtain  $N$  where  $N: \bigwedge p. p > N \implies \exists m > p. u\ p > u\ m$  by ( $force\ simp:$ 
not\_le\_less)
  have  $\exists r. \forall n. N < r\ n \wedge r\ n < r$  ( $Suc\ n$ )  $\wedge (\forall i \in \{N<..r(Suc\ n)\}. u\ i \geq u\ (r$ 
( $Suc\ n$ )))
  proof ( $rule\ dependent\_nat\_choice$ )

```

```

fix x assume N < x
then have a: finite {N<..x} {N<..x} ≠ {} by simp_all
have Min {u i | i. i ∈ {N<..x}} ∈ {u i | i. i ∈ {N<..x}} apply (rule Min_in)
using a by (auto)
then obtain p where p ∈ {N<..x} and upmin: u p = Min{u i | i. i ∈ {N<..x}}
by auto
define U where U = {m. m > p ∧ u p > u m}
have U ≠ {} unfolding U_def using N[of p] ⟨p ∈ {N<..x}⟩ by auto
define y where y = Inf U
then have y ∈ U using ⟨U ≠ {}⟩ by (simp add: Inf_nat_def1)
have a: ∧i. i ∈ {N<..x} ⇒ u i ≥ u p
proof -
  fix i assume i ∈ {N<..x}
  then have u i ∈ {u i | i. i ∈ {N<..x}} by blast
  then show u i ≥ u p using upmin by simp
qed
moreover have u p > u y using ⟨y ∈ U⟩ U_def by auto
ultimately have y ∉ {N<..x} using not_le by blast
moreover have y > N using ⟨y ∈ U⟩ U_def ⟨p ∈ {N<..x}⟩ by auto
ultimately have y > x by auto

have ∧i. i ∈ {N<..y} ⇒ u i ≥ u y
proof -
  fix i assume i ∈ {N<..y} show u i ≥ u y
  proof (cases)
    assume i = y
    then show ?thesis by simp
  next
    assume ¬(i=y)
    then have i: i ∈ {N<..

```



```

then obtain  $r :: \text{nat} \Rightarrow \text{nat}$ 
  where  $r: \forall n. N < r\ n \wedge r\ n < r\ (\text{Suc}\ n) \wedge (\forall i \in \{N <..r\ (\text{Suc}\ n)\}. u\ i \geq u\ (r\ (\text{Suc}\ n)))$  by auto
  have strict_mono  $r$  using  $r$  by (auto simp: strict_mono_Suc_iff)
  have decseq  $(u\ o\ r)$  unfolding o_def using  $r$  by (simp add: decseq_SucI order.strict_implies_order)
  then have  $(u\ o\ r) \longrightarrow (\text{INF}\ n. (u\ o\ r)\ n)$  using LIMSEQ_INF by blast
  then have  $\text{liminf}\ (u\ o\ r) = (\text{INF}\ n. (u\ o\ r)\ n)$  by (simp add: lim_imp_Liminf)
  moreover have  $\text{liminf}\ (u\ o\ r) \geq \text{liminf}\ u$  using  $\langle \text{strict\_mono}\ r \rangle$  by (simp add: liminf_subseq_mono)
  ultimately have  $(\text{INF}\ n. (u\ o\ r)\ n) \geq \text{liminf}\ u$  by simp

{
  fix  $i$  assume  $i: i \in \{N <.. \}$ 
  obtain  $n$  where  $i < r\ (\text{Suc}\ n)$  using  $\langle \text{strict\_mono}\ r \rangle$  using Suc_le_eq seq_suble by blast
  then have  $i \in \{N <..r\ (\text{Suc}\ n)\}$  using  $i$  by simp
  then have  $u\ i \geq u\ (r\ (\text{Suc}\ n))$  using  $r$  by simp
  then have  $u\ i \geq (\text{INF}\ n. (u\ o\ r)\ n)$  unfolding o_def by (meson INF_lower2 UNIV_I)
}
then have  $(\text{INF}\ i \in \{N <.. \}. u\ i) \geq (\text{INF}\ n. (u\ o\ r)\ n)$  using INF_greatest by blast
then have  $\text{liminf}\ u \geq (\text{INF}\ n. (u\ o\ r)\ n)$  unfolding Liminf_def
by (metis (mono_tags, lifting) SUP_upper2 atLeast_Suc_greaterThan atLeast_def eventually_ge_at_top mem_Collect_eq)
then have  $\text{liminf}\ u = (\text{INF}\ n. (u\ o\ r)\ n)$  using  $\langle (\text{INF}\ n. (u\ o\ r)\ n) \geq \text{liminf}\ u \rangle$ 
by simp
then have  $(u\ o\ r) \longrightarrow \text{liminf}\ u$  using  $\langle (u\ o\ r) \longrightarrow (\text{INF}\ n. (u\ o\ r)\ n) \rangle$ 
by simp
then show ?thesis using  $\langle \text{strict\_mono}\ r \rangle$  by auto
qed

```

The following statement about limsups is reduced to a statement about limits using subsequences thanks to *limsup_subseq_lim*. The statement for limits follows for instance from *tendsto_add_ereal_general*.

lemma *ereal_limsup_add_mono*:

```

fixes  $u\ v :: \text{nat} \Rightarrow \text{ereal}$ 
shows  $\text{limsup}\ (\lambda n. u\ n + v\ n) \leq \text{limsup}\ u + \text{limsup}\ v$ 

```

proof (*cases*)

```

assume  $(\text{limsup}\ u = \infty) \vee (\text{limsup}\ v = \infty)$ 

```

```

then have  $\text{limsup}\ u + \text{limsup}\ v = \infty$  by simp

```

```

then show ?thesis by auto

```

next

```

assume  $\neg((\text{limsup}\ u = \infty) \vee (\text{limsup}\ v = \infty))$ 

```

```

then have  $\text{limsup}\ u < \infty$   $\text{limsup}\ v < \infty$  by auto

```

```

define  $w$  where  $w = (\lambda n. u\ n + v\ n)$ 

```

```

obtain  $r$  where  $r: \text{strict\_mono}\ r\ (w\ o\ r) \longrightarrow \text{limsup}\ w$  using limsup_subseq_lim

```

```

by auto
  obtain s where s: strict_mono s (u o r o s) ⟶ limsup (u o r) using
limsup_subseq_lim by auto
  obtain t where t: strict_mono t (v o r o s o t) ⟶ limsup (v o r o s) using
limsup_subseq_lim by auto

  define a where a = r o s o t
  have strict_mono a using r s t by (simp add: a_def strict_mono_o)
  have l:(w o a) ⟶ limsup w
        (u o a) ⟶ limsup (u o r)
        (v o a) ⟶ limsup (v o r o s)
  apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
  apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
  apply (metis (no_types, lifting) t(2) a_def comp_assoc)
  done

  have limsup (u o r) ≤ limsup u by (simp add: limsup_subseq_mono r(1))
  then have a: limsup (u o r) ≠ ∞ using ⟨limsup u < ∞⟩ by auto
  have limsup (v o r o s) ≤ limsup v
  by (simp add: comp_assoc limsup_subseq_mono r(1) s(1) strict_mono_o)
  then have b: limsup (v o r o s) ≠ ∞ using ⟨limsup v < ∞⟩ by auto

  have (λn. (u o a) n + (v o a) n) ⟶ limsup (u o r) + limsup (v o r o s)
  using l tendsto_add_ereal_general a b by fastforce
  moreover have (λn. (u o a) n + (v o a) n) = (w o a) unfolding w_def by
auto
  ultimately have (w o a) ⟶ limsup (u o r) + limsup (v o r o s) by simp
  then have limsup w = limsup (u o r) + limsup (v o r o s) using l(1) LIM-
SEQ_unique by blast
  then have limsup w ≤ limsup u + limsup v
  using ⟨limsup (u o r) ≤ limsup u⟩ ⟨limsup (v o r o s) ≤ limsup v⟩ add_mono
by simp
  then show ?thesis unfolding w_def by simp
qed

```

There is an asymmetry between liminfs and limsups in *ereal*, as $\infty + (-\infty) = \infty$. This explains why there are more assumptions in the next lemma dealing with liminfs than in the previous one about limsups.

lemma *ereal_liminf_add_mono*:

```

  fixes u v::nat ⇒ ereal
  assumes ¬((liminf u = ∞ ∧ liminf v = -∞) ∨ (liminf u = -∞ ∧ liminf v =
∞))
  shows liminf (λn. u n + v n) ≥ liminf u + liminf v
proof (cases)
  assume (liminf u = -∞) ∨ (liminf v = -∞)
  then have *: liminf u + liminf v = -∞ using assms by auto
  show ?thesis by (simp add: *)

```

```

next
  assume  $\neg((\text{liminf } u = -\infty) \vee (\text{liminf } v = -\infty))$ 
  then have  $\text{liminf } u > -\infty$   $\text{liminf } v > -\infty$  by auto

  define w where  $w = (\lambda n. u\ n + v\ n)$ 
  obtain r where  $r: \text{strict\_mono } r (w\ o\ r) \longrightarrow \text{liminf } w$  using liminf_subseq_lim
  by auto
  obtain s where  $s: \text{strict\_mono } s (u\ o\ r\ o\ s) \longrightarrow \text{liminf } (u\ o\ r)$  using
  liminf_subseq_lim by auto
  obtain t where  $t: \text{strict\_mono } t (v\ o\ r\ o\ s\ o\ t) \longrightarrow \text{liminf } (v\ o\ r\ o\ s)$  using
  liminf_subseq_lim by auto

  define a where  $a = r\ o\ s\ o\ t$ 
  have strict_mono a using r s t by (simp add: a_def strict_mono_o)
  have  $l:(w\ o\ a) \longrightarrow \text{liminf } w$ 
     $(u\ o\ a) \longrightarrow \text{liminf } (u\ o\ r)$ 
     $(v\ o\ a) \longrightarrow \text{liminf } (v\ o\ r\ o\ s)$ 
  apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
  apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
  apply (metis (no_types, lifting) t(2) a_def comp_assoc)
  done

  have  $\text{liminf } (u\ o\ r) \geq \text{liminf } u$  by (simp add: liminf_subseq_mono r(1))
  then have  $a: \text{liminf } (u\ o\ r) \neq -\infty$  using  $\langle \text{liminf } u > -\infty \rangle$  by auto
  have  $\text{liminf } (v\ o\ r\ o\ s) \geq \text{liminf } v$ 
    by (simp add: comp_assoc liminf_subseq_mono r(1) s(1) strict_mono_o)
  then have  $b: \text{liminf } (v\ o\ r\ o\ s) \neq -\infty$  using  $\langle \text{liminf } v > -\infty \rangle$  by auto

  have  $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) \longrightarrow \text{liminf } (u\ o\ r) + \text{liminf } (v\ o\ r\ o\ s)$ 
    using l tendsto_add_ereal_general a b by fastforce
  moreover have  $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) = (w\ o\ a)$  unfolding w_def by
  auto
  ultimately have  $(w\ o\ a) \longrightarrow \text{liminf } (u\ o\ r) + \text{liminf } (v\ o\ r\ o\ s)$  by simp
  then have  $\text{liminf } w = \text{liminf } (u\ o\ r) + \text{liminf } (v\ o\ r\ o\ s)$  using l(1) LIMSEQ_unique by blast
  then have  $\text{liminf } w \geq \text{liminf } u + \text{liminf } v$ 
    using  $\langle \text{liminf } (u\ o\ r) \geq \text{liminf } u \rangle \langle \text{liminf } (v\ o\ r\ o\ s) \geq \text{liminf } v \rangle$  add_mono
  by simp
  then show ?thesis unfolding w_def by simp
qed

lemma ereal_limsup_lim_add:
  fixes  $u\ v::\text{nat} \Rightarrow \text{ereal}$ 
  assumes  $u \longrightarrow a$   $\text{abs}(a) \neq \infty$ 
  shows  $\text{limsup } (\lambda n. u\ n + v\ n) = a + \text{limsup } v$ 
proof -
  have  $\text{limsup } u = a$  using assms(1) using tendsto_iff_Liminf_eq_Limsup triv-

```

ial_limit_at_top_linorder **by** *blast*

have $(\lambda n. -u\ n) \longrightarrow -a$ **using** *assms(1)* **by** *auto*

then have $\limsup (\lambda n. -u\ n) = -a$ **using** *tendsto_iff_Liminf_eq_Limsup* *trivial_limit_at_top_linorder* **by** *blast*

have $\limsup (\lambda n. u\ n + v\ n) \leq \limsup u + \limsup v$

by (*rule ereal_limsup_add_mono*)

then have *up*: $\limsup (\lambda n. u\ n + v\ n) \leq a + \limsup v$ **using** $\langle \limsup u = a \rangle$ **by** *simp*

have *a*: $\limsup (\lambda n. (u\ n + v\ n) + (-u\ n)) \leq \limsup (\lambda n. u\ n + v\ n) + \limsup (\lambda n. -u\ n)$

by (*rule ereal_limsup_add_mono*)

have *eventually* $(\lambda n. u\ n = \text{ereal}(\text{real_of_ereal}(u\ n)))$ *sequentially* **using** *assms* *real_lim_then_eventually_real* **by** *auto*

moreover have $\bigwedge x. x = \text{ereal}(\text{real_of_ereal}(x)) \implies x + (-x) = 0$

by (*metis plus_ereal.simps(1) right_minus_uminus_ereal.simps(1) zero_ereal_def*)

ultimately have *eventually* $(\lambda n. u\ n + (-u\ n) = 0)$ *sequentially*

by (*metis (mono_tags, lifting) eventually_mono*)

moreover have $\bigwedge n. u\ n + (-u\ n) = 0 \implies u\ n + v\ n + (-u\ n) = v\ n$

by (*metis add.commute add.left_commute add.left_neutral*)

ultimately have *eventually* $(\lambda n. u\ n + v\ n + (-u\ n) = v\ n)$ *sequentially*

using *eventually_mono* **by** *force*

then have $\limsup v = \limsup (\lambda n. u\ n + v\ n + (-u\ n))$ **using** *Limsup_eq* **by** *force*

then have $\limsup v \leq \limsup (\lambda n. u\ n + v\ n) - a$ **using** $\langle \limsup (\lambda n. -u\ n) = -a \rangle$ **by** (*simp add: minus_ereal_def*)

then have $\limsup (\lambda n. u\ n + v\ n) \geq a + \limsup v$ **using** *assms(2)* **by** (*metis add.commute ereal_le_minus*)

then show *?thesis* **using** *up* **by** *simp*

qed

lemma *ereal_limsup_lim_mult*:

fixes $u\ v::\text{nat} \Rightarrow \text{ereal}$

assumes $u \longrightarrow a$ $a > 0$ $a \neq \infty$

shows $\limsup (\lambda n. u\ n * v\ n) = a * \limsup v$

proof $-$

define *w* **where** $w = (\lambda n. u\ n * v\ n)$

obtain *r* **where** r : *strict_mono* $r (v\ o\ r) \longrightarrow \limsup v$ **using** *limsup_subseq_lim* **by** *auto*

have $(u\ o\ r) \longrightarrow a$ **using** *assms(1)* *LIMSEQ_subseq_LIMSEQ* *r* **by** *auto*

with *tendsto_mult_ereal* [*OF this r(2)*] **have** $(\lambda n. (u\ o\ r)\ n * (v\ o\ r)\ n) \longrightarrow a * \limsup v$ **using** *assms(2)* *assms(3)* **by** *auto*

moreover have $\bigwedge n. (w\ o\ r)\ n = (u\ o\ r)\ n * (v\ o\ r)\ n$ **unfolding** *w_def* **by** *auto*

ultimately have $(w\ o\ r) \longrightarrow a * \limsup v$ **unfolding** *w_def* **by** *presburger*

then have $\limsup (w\ o\ r) = a * \limsup v$ **by** (*simp add: tendsto_iff_Liminf_eq_Limsup*)

then have *I*: $\limsup w \geq a * \limsup v$ **by** (*metis limsup_subseq_mono r(1)*)

obtain s **where** s : $strict_mono\ s\ (w\ o\ s) \longrightarrow limsup\ w$ **using** $limsup_subseq_lim$
by $auto$
have $*$: $(u\ o\ s) \longrightarrow a$ **using** $assms(1)\ LIMSEQ_subseq_LIMSEQ\ s$ **by** $auto$
have $eventually\ (\lambda n. (u\ o\ s)\ n > 0)$ **sequentially** **using** $assms(2)\ *\ order_tendsto_iff$
by $blast$
moreover **have** $eventually\ (\lambda n. (u\ o\ s)\ n < \infty)$ **sequentially** **using** $assms(3)\ *$
 $order_tendsto_iff$ **by** $blast$
moreover **have** $(w\ o\ s)\ n / (u\ o\ s)\ n = (v\ o\ s)\ n$ **if** $(u\ o\ s)\ n > 0$ $(u\ o\ s)\ n <$
 ∞ **for** n
unfolding w_def **using** $that$ **by** $(auto\ simp: ereal_divide_eq)$
ultimately **have** $eventually\ (\lambda n. (w\ o\ s)\ n / (u\ o\ s)\ n = (v\ o\ s)\ n)$ **sequentially**
using $eventually_elim2$ **by** $force$
moreover **have** $(\lambda n. (w\ o\ s)\ n / (u\ o\ s)\ n) \longrightarrow (limsup\ w) / a$
apply $(rule\ tendsto_divide_ereal[OF\ s(2)\ *])$ **using** $assms(2)\ assms(3)$ **by**
 $auto$
ultimately **have** $(v\ o\ s) \longrightarrow (limsup\ w) / a$ **using** $Lim_transform_eventually$
by $fastforce$
then **have** $limsup\ (v\ o\ s) = (limsup\ w) / a$ **by** $(simp\ add: tendsto_iff_Liminf_eq_Limsup)$
then **have** $limsup\ v \geq (limsup\ w) / a$ **by** $(metis\ limsup_subseq_mono\ s(1))$
then **have** $a * limsup\ v \geq limsup\ w$ **using** $assms(2)\ assms(3)$ **by** $(simp\ add:$
 $ereal_divide_le_pos)$
then **show** $?thesis$ **using** I **unfolding** w_def **by** $auto$
qed

lemma $ereal_liminf_lim_mult$:

fixes $u\ v::nat \Rightarrow ereal$
assumes $u \longrightarrow a$ $a > 0$ $a \neq \infty$
shows $liminf\ (\lambda n. u\ n * v\ n) = a * liminf\ v$
proof $-$
define w **where** $w = (\lambda n. u\ n * v\ n)$
obtain r **where** r : $strict_mono\ r\ (v\ o\ r) \longrightarrow liminf\ v$ **using** $liminf_subseq_lim$
by $auto$
have $(u\ o\ r) \longrightarrow a$ **using** $assms(1)\ LIMSEQ_subseq_LIMSEQ\ r$ **by** $auto$
with $tendsto_mult_ereal[OF\ this\ r(2)]$ **have** $(\lambda n. (u\ o\ r)\ n * (v\ o\ r)\ n) \longrightarrow$
 $a * liminf\ v$ **using** $assms(2)\ assms(3)$ **by** $auto$
moreover **have** $\bigwedge n. (w\ o\ r)\ n = (u\ o\ r)\ n * (v\ o\ r)\ n$ **unfolding** w_def **by**
 $auto$
ultimately **have** $(w\ o\ r) \longrightarrow a * liminf\ v$ **unfolding** w_def **by** $presburger$
then **have** $liminf\ (w\ o\ r) = a * liminf\ v$ **by** $(simp\ add: tendsto_iff_Liminf_eq_Limsup)$
then **have** $I: liminf\ w \leq a * liminf\ v$ **by** $(metis\ liminf_subseq_mono\ r(1))$

obtain s **where** s : $strict_mono\ s\ (w\ o\ s) \longrightarrow liminf\ w$ **using** $liminf_subseq_lim$
by $auto$
have $*$: $(u\ o\ s) \longrightarrow a$ **using** $assms(1)\ LIMSEQ_subseq_LIMSEQ\ s$ **by** $auto$
have $eventually\ (\lambda n. (u\ o\ s)\ n > 0)$ **sequentially** **using** $assms(2)\ *\ order_tendsto_iff$
by $blast$
moreover **have** $eventually\ (\lambda n. (u\ o\ s)\ n < \infty)$ **sequentially** **using** $assms(3)\ *$
 $order_tendsto_iff$ **by** $blast$
moreover **have** $(w\ o\ s)\ n / (u\ o\ s)\ n = (v\ o\ s)\ n$ **if** $(u\ o\ s)\ n > 0$ $(u\ o\ s)\ n <$

∞ for n
unfolding w_def **using** *that* **by** (*auto simp: ereal_divide_eq*)
ultimately have *eventually* $(\lambda n. (w \ o \ s) \ n / (u \ o \ s) \ n = (v \ o \ s) \ n)$ *sequentially*
using *eventually_elim2* **by** *force*
moreover have $(\lambda n. (w \ o \ s) \ n / (u \ o \ s) \ n) \longrightarrow (liminf \ w) / a$
using $*$ *assms s tendsto_divide_ereal* **by** *fastforce*
ultimately have $(v \ o \ s) \longrightarrow (liminf \ w) / a$ **using** *Lim_transform_eventually*
by *fastforce*
then have $liminf \ (v \ o \ s) = (liminf \ w) / a$ **by** (*simp add: tendsto_iff_Liminf_eq_Limsup*)
then have $liminf \ v \leq (liminf \ w) / a$ **by** (*metis liminf_subseq_mono s(1)*)
then have $a * liminf \ v \leq liminf \ w$ **using** *assms(2) assms(3)* **by** (*simp add: ereal_le_divide_pos*)
then show *?thesis* **using** *I unfolding w_def* **by** *auto*
qed

lemma *ereal_liminf_lim_add:*

fixes $u \ v :: nat \Rightarrow ereal$
assumes $u \longrightarrow a$ $abs(a) \neq \infty$
shows $liminf \ (\lambda n. u \ n + v \ n) = a + liminf \ v$
proof $-$
have $liminf \ u = a$ **using** *assms(1) tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder*
by *blast*
then have $*$: $abs(liminf \ u) \neq \infty$ **using** *assms(2)* **by** *auto*
have $(\lambda n. -u \ n) \longrightarrow -a$ **using** *assms(1)* **by** *auto*
then have $liminf \ (\lambda n. -u \ n) = -a$ **using** *tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder* **by** *blast*
then have $**$: $abs(liminf \ (\lambda n. -u \ n)) \neq \infty$ **using** *assms(2)* **by** *auto*

have $liminf \ (\lambda n. u \ n + v \ n) \geq liminf \ u + liminf \ v$
using *abs_ereal.simps* **by** (*metis (full_types) * ereal_liminf_add_mono*)
then have *up*: $liminf \ (\lambda n. u \ n + v \ n) \geq a + liminf \ v$ **using** $\langle liminf \ u = a \rangle$ **by**
simp

have a : $liminf \ (\lambda n. (u \ n + v \ n) + (-u \ n)) \geq liminf \ (\lambda n. u \ n + v \ n) + liminf \ (\lambda n. -u \ n)$
apply (*rule ereal_liminf_add_mono*) **using** $**$ **by** *auto*
have *eventually* $(\lambda n. u \ n = ereal(real_of_ereal(u \ n)))$ *sequentially* **using** *assms real_lim_then_eventually_real* **by** *auto*
moreover have $\bigwedge x. x = ereal(real_of_ereal(x)) \implies x + (-x) = 0$
by (*metis plus_ereal.simps(1) right_minus_uminus_ereal.simps(1) zero_ereal_def*)
ultimately have *eventually* $(\lambda n. u \ n + (-u \ n) = 0)$ *sequentially*
by (*metis (mono_tags, lifting) eventually_mono*)
moreover have $\bigwedge n. u \ n + (-u \ n) = 0 \implies u \ n + v \ n + (-u \ n) = v \ n$
by (*metis add commute add.left_commute add.left_neutral*)
ultimately have *eventually* $(\lambda n. u \ n + v \ n + (-u \ n) = v \ n)$ *sequentially*
using *eventually_mono* **by** *force*
then have $liminf \ v = liminf \ (\lambda n. u \ n + v \ n + (-u \ n))$ **using** *Liminf_eq* **by**
force
then have $liminf \ v \geq liminf \ (\lambda n. u \ n + v \ n) - a$ **using** $\langle liminf \ (\lambda n. -u \ n) =$

```

-a) by (simp add: minus_ereal_def)
  then have  $\liminf (\lambda n. u\ n + v\ n) \leq a + \liminf v$  using assms(2) by (metis
  add.commute ereal_minus_le)
  then show ?thesis using up by simp
qed

```

lemma *ereal_liminf_limsup_add*:

```

  fixes  $u\ v::\text{nat} \Rightarrow \text{ereal}$ 
  shows  $\liminf (\lambda n. u\ n + v\ n) \leq \liminf u + \limsup v$ 
proof (cases)
  assume  $\limsup v = \infty \vee \liminf u = \infty$ 
  then show ?thesis by auto
next
  assume  $\neg(\limsup v = \infty \vee \liminf u = \infty)$ 
  then have  $\limsup v < \infty$   $\liminf u < \infty$  by auto

```

```

  define  $w$  where  $w = (\lambda n. u\ n + v\ n)$ 
  obtain  $r$  where  $r: \text{strict\_mono } r (u\ o\ r) \longrightarrow \liminf u$  using liminf_subseq_lim
  by auto
  obtain  $s$  where  $s: \text{strict\_mono } s (w\ o\ r\ o\ s) \longrightarrow \liminf (w\ o\ r)$  using
  liminf_subseq_lim by auto
  obtain  $t$  where  $t: \text{strict\_mono } t (v\ o\ r\ o\ s\ o\ t) \longrightarrow \limsup (v\ o\ r\ o\ s)$  using
  limsup_subseq_lim by auto

```

```

  define  $a$  where  $a = r\ o\ s\ o\ t$ 
  have strict_mono a using r s t by (simp add: a_def strict_mono_o)
  have  $l:(u\ o\ a) \longrightarrow \liminf u$ 
     $(w\ o\ a) \longrightarrow \liminf (w\ o\ r)$ 
     $(v\ o\ a) \longrightarrow \limsup (v\ o\ r\ o\ s)$ 
  apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
  comp_assoc)
  apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
  comp_assoc)
  apply (metis (no_types, lifting) t(2) a_def comp_assoc)
  done

```

```

  have  $\liminf (w\ o\ r) \geq \liminf w$  by (simp add: liminf_subseq_mono r(1))
  have  $\limsup (v\ o\ r\ o\ s) \leq \limsup v$ 
  by (simp add: comp_assoc limsup_subseq_mono r(1) s(1) strict_mono_o)
  then have  $b: \limsup (v\ o\ r\ o\ s) < \infty$  using  $\langle \limsup v < \infty \rangle$  by auto

```

```

  have  $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) \longrightarrow \liminf u + \limsup (v\ o\ r\ o\ s)$ 
  apply (rule tendsto_add_ereal_general) using  $b$   $\langle \liminf u < \infty \rangle$   $l(1)$   $l(3)$  by
  force+
  moreover have  $(\lambda n. (u\ o\ a)\ n + (v\ o\ a)\ n) = (w\ o\ a)$  unfolding w_def by
  auto
  ultimately have  $(w\ o\ a) \longrightarrow \liminf u + \limsup (v\ o\ r\ o\ s)$  by simp
  then have  $\liminf (w\ o\ r) = \liminf u + \limsup (v\ o\ r\ o\ s)$  using  $l(2)$  using
  LIMSEQ_unique by blast

```

1772

then have $\liminf w \leq \liminf u + \limsup v$
using $\langle \liminf (w \circ r) \geq \liminf w \rangle \langle \limsup (v \circ r \circ s) \leq \limsup v \rangle$
by (*metis add_mono_thms_linordered_semiring(2) le_less_trans not_less*)
then show *?thesis unfolding w_def by simp*
qed

lemma *ereal_liminf_limsup_minus*:
fixes $u v :: \text{nat} \Rightarrow \text{ereal}$
shows $\liminf (\lambda n. u\ n - v\ n) \leq \limsup u - \limsup v$
unfolding *minus_ereal_def*
apply (*subst add.commute*)
apply (*rule order_trans[OF eréal_liminf_limsup_add]*)
using *ereal_Limsup_u_minus*[of *sequentially* $\lambda n. - v\ n$]
apply (*simp add: add.commute*)
done

lemma *liminf_minus_ennreal*:
fixes $u v :: \text{nat} \Rightarrow \text{ennreal}$
shows $(\bigwedge n. v\ n \leq u\ n) \implies \liminf (\lambda n. u\ n - v\ n) \leq \limsup u - \limsup v$
unfolding *liminf_SUP_INF limsup_INF_SUP*
including *ennreal.lifting*
proof (*transfer, clarsimp*)
fix $v u :: \text{nat} \Rightarrow \text{ereal}$ **assume** $*: \forall x. 0 \leq v\ x \forall x. 0 \leq u\ x \bigwedge n. v\ n \leq u\ n$
moreover have $0 \leq \limsup u - \limsup v$
using $*$ **by** (*intro eréal_diff_positive Limsup_mono always_eventually*) *simp*
moreover have $0 \leq \text{Sup } (u \text{ ' } \{x..\})$ **for** x
using $*$ **by** (*intro SUP_upper2*[of x]) *auto*
moreover have $0 \leq \text{Sup } (v \text{ ' } \{x..\})$ **for** x
using $*$ **by** (*intro SUP_upper2*[of x]) *auto*
ultimately show $(\text{SUP } n. \text{INF } n \in \{n..\}. \max 0 (u\ n - v\ n))$
 $\leq \max 0 ((\text{INF } x. \max 0 (\text{Sup } (u \text{ ' } \{x..\}))) - (\text{INF } x. \max 0 (\text{Sup } (v \text{ ' } \{x..\}))))$
by (*auto simp: * eréal_diff_positive max.absorb2 liminf_SUP_INF*[*symmetric*]
limsup_INF_SUP[*symmetric*] *ereal_liminf_limsup_minus*)
qed

6.13.4 Relate extended reals and the indicator function

lemma *ereal_indicator_le_0*: $(\text{indicator } S\ x :: \text{ereal}) \leq 0 \iff x \notin S$
by (*auto split: split_indicator simp: one_ereal_def*)

lemma *ereal_indicator*: $\text{ereal } (\text{indicator } A\ x) = \text{indicator } A\ x$
by (*auto simp: indicator_def one_ereal_def*)

lemma *ereal_mult_indicator*: $\text{ereal } (x * \text{indicator } A\ y) = \text{ereal } x * \text{indicator } A\ y$
by (*simp split: split_indicator*)

lemma *ereal_indicator_mult*: $\text{ereal } (\text{indicator } A\ y * x) = \text{indicator } A\ y * \text{ereal } x$

by (simp split: split_indicator)

lemma *ereal_indicator_nonneg*[simp, intro]: $0 \leq (\text{indicator } A \ x :: \text{ereal})$
unfolding *indicator_def* **by** *auto*

lemma *indicator_inter_arith_ereal*: $\text{indicator } A \ x * \text{indicator } B \ x = (\text{indicator } (A \cap B) \ x :: \text{ereal})$
by (simp split: split_indicator)

end

6.14 Radius of Convergence and Summation Tests

theory *Summation_Tests*

imports

Complex_Main

HOL-Library.Discrete

HOL-Library.Extended_Real

HOL-Library.Liminf_Limsup

Extended_Real_Limits

begin

The definition of the radius of convergence of a power series, various summability tests, lemmas to compute the radius of convergence etc.

6.14.1 Convergence tests for infinite sums

Root test

lemma *limsup_root_powser*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach}, \text{real_normed_div_algebra}\}$

shows $\text{limsup } (\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n * z \wedge n)))) =$
 $\text{limsup } (\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n)))) * \text{ereal } (\text{norm } z)$

proof –

have $A: (\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n * z \wedge n)))) =$

$(\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n)))) * \text{ereal } (\text{norm } z)$ (**is** $?g = ?h$)

proof

fix n **show** $?g \ n = ?h \ n$

by (*cases* $n = 0$) (*simp_all add: norm_mult real_root_mult real_root_pos2 norm_power*)

qed

show *?thesis* **by** (*subst* A , *subst* *limsup_ereal_mult_right*) *simp_all*

qed

lemma *limsup_root_limit*:

assumes $(\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n)))) \longrightarrow l$ (**is** $?g \longrightarrow _$)

shows $\text{limsup } (\lambda n. \text{ereal } (\text{root } n \ (\text{norm } (f \ n)))) = l$

proof –

from *assms* **have** *convergent ?g lim ?g = l*

unfolding *convergent_def* **by** (*blast intro: limI*)
with *convergent_limsup_cl* **show** *?thesis* **by force**
qed

lemma *limsup_root_limit'*:
assumes $(\lambda n. \text{root } n (\text{norm } (f \ n))) \longrightarrow l$
shows $\text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f \ n)))) = \text{ereal } l$
by (*intro limsup_root_limit tendsto_ereal assms*)

theorem *root_test_convergence'*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{banach}$
defines $l \equiv \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f \ n))))$
assumes $l < 1$
shows *summable* f
proof –
have $0 = \text{limsup } (\lambda n. 0)$ **by** (*simp add: Limsup_const*)
also have $\dots \leq l$ **unfolding** *l_def* **by** (*intro Limsup_mono*) (*simp_all add: real_root_ge_zero*)
finally have $l \geq 0$ **by** *simp*
with l **obtain** l' **where** $l' : l = \text{ereal } l'$ **by** (*cases l*) *simp_all*

define c **where** $c = (1 - l') / 2$
from l **and** $\langle l \geq 0 \rangle$ **have** $c : l + c > l' + c \geq 0 \ l' + c < 1$ **unfolding** *c_def*
by (*simp_all add: field_simps l'*)
have $\forall C > l. \text{eventually } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f \ n))) < C)$ *sequentially*
by (*subst Limsup_le_iff[symmetric]*) (*simp add: l_def*)
with c **have** $\text{eventually } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f \ n))) < l + \text{ereal } c)$ *sequentially*
by *simp*
with *eventually_gt_at_top*[*of 0::nat*]
have $\text{eventually } (\lambda n. \text{norm } (f \ n) \leq (l' + c) \wedge n)$ *sequentially*
proof *eventually_elim*
fix $n :: \text{nat}$ **assume** $n : n > 0$
assume $\text{ereal } (\text{root } n (\text{norm } (f \ n))) < l + \text{ereal } c$
hence $\text{root } n (\text{norm } (f \ n)) \leq l' + c$ **by** (*simp add: l'*)
with $c \ n$ **have** $\text{root } n (\text{norm } (f \ n)) \wedge n \leq (l' + c) \wedge n$
by (*intro power_mono*) (*simp_all add: real_root_ge_zero*)
also from n **have** $\text{root } n (\text{norm } (f \ n)) \wedge n = \text{norm } (f \ n)$ **by** *simp*
finally show $\text{norm } (f \ n) \leq (l' + c) \wedge n$ **by** *simp*
qed
thus *?thesis*
by (*rule summable_comparison_test_ev[OF _ summable_geometric]*) (*simp add: c*)
qed

theorem *root_test_divergence*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{banach}$
defines $l \equiv \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f \ n))))$
assumes $l > 1$
shows $\neg \text{summable } f$

proof

assume *summable* *f*
hence *bounded*: *Bseq* *f* **by** (*simp* *add*: *summable_imp_Bseq*)

have $0 = \text{limsup } (\lambda n. 0)$ **by** (*simp* *add*: *Limsup_const*)
also have $\dots \leq l$ **unfolding** *l_def* **by** (*intro* *Limsup_mono*) (*simp_all* *add*:
real_root_ge_zero)
finally have *l_nonneg*: $l \geq 0$ **by** *simp*

define *c* **where** $c = (\text{if } l = \infty \text{ then } 2 \text{ else } 1 + (\text{real_of_ereal } l - 1) / 2)$
from *l_nonneg* **consider** $l = \infty \mid \exists l'. l = \text{ereal } l'$ **by** (*cases* *l*) *simp_all*
hence *c*: $c > 1 \wedge \text{ereal } c < l$ **by** *cases* (*insert* *l*, *auto* *simp*: *c_def* *field_simps*)

have *unbounded*: $\neg \text{bdd_above } \{n. \text{root } n (\text{norm } (f \ n)) > c\}$

proof

assume *bdd_above* $\{n. \text{root } n (\text{norm } (f \ n)) > c\}$
then obtain *N* **where** $\forall n. \text{root } n (\text{norm } (f \ n)) > c \longrightarrow n \leq N$ **unfolding**
bdd_above_def **by** *blast*
hence $\exists N. \forall n \geq N. \text{root } n (\text{norm } (f \ n)) \leq c$
by (*intro* *exI*[*of* $N + 1$]) (*force* *simp*: *not_less_eq_eq*[*symmetric*])
hence *eventually* $(\lambda n. \text{root } n (\text{norm } (f \ n)) \leq c)$ *sequentially*
by (*auto* *simp*: *eventually_at_top_linorder*)
hence $l \leq c$ **unfolding** *l_def* **by** (*intro* *Limsup_bounded*) *simp_all*
with *c* **show** *False* **by** *auto*

qed

from *bounded* **obtain** *K* **where** $K: K > 0 \wedge n. \text{norm } (f \ n) \leq K$ **using** *BseqE*
by *blast*

define *n* **where** $n = \text{nat } \lceil \log c \ K \rceil$

from *unbounded* **have** $\exists m > n. c < \text{root } m (\text{norm } (f \ m))$ **unfolding** *bdd_above_def*
by (*auto* *simp*: *not_le*)

then obtain *m* **where** $m: n < m < \text{root } m (\text{norm } (f \ m))$ **by** *auto*
from *c* *K* **have** $K = c \text{ powr } \log c \ K$ **by** (*simp* *add*: *powr_def* *log_def*)

also from *c* **have** $c \text{ powr } \log c \ K \leq c \text{ powr } \text{real } n$ **unfolding** *n_def*
by (*intro* *powr_mono*, *linarith*, *simp*)

finally have $K \leq c \wedge n$ **using** *c* **by** (*simp* *add*: *powr_realpow*)

also from *c* *m* **have** $c \wedge n < c \wedge m$ **by** *simp*
also from *c* *m* **have** $c \wedge m < \text{root } m (\text{norm } (f \ m)) \wedge m$ **by** (*intro* *power_strict_mono*)
simp_all

also from *m* **have** $\dots = \text{norm } (f \ m)$ **by** *simp*

finally show *False* **using** $K(2)[\text{of } m]$ **by** *simp*

qed

Cauchy's condensation test

context

fixes *f* :: *nat* \Rightarrow *real*

begin

private lemma *condensation_inequality*:

assumes *mono*: $\bigwedge m n. 0 < m \implies m \leq n \implies f n \leq f m$

shows $(\sum_{k=1..<n}. f k) \geq (\sum_{k=1..<n}. f (2 * 2^{\text{Discrete.log } k}))$ (**is** *?thesis1*)
 $(\sum_{k=1..<n}. f k) \leq (\sum_{k=1..<n}. f (2^{\text{Discrete.log } k}))$ (**is** *?thesis2*)

by (*intro sum_mono mono Discrete.log_exp2_ge Discrete.log_exp2_le, simp, simp*)**+**

private lemma *condensation_condense1*: $(\sum_{k=1..<2^n}. f (2^{\text{Discrete.log } k})) = (\sum_{k < n}. 2^k * f (2^k))$

proof (*induction n*)

case (*Suc n*)

have $\{1..<2^{\text{Suc } n}\} = \{1..<2^n\} \cup \{2^n..<(2^{\text{Suc } n} :: \text{nat})\}$ **by** *auto*

also have $(\sum_{k \in \dots} f (2^{\text{Discrete.log } k})) = (\sum_{k < n}. 2^k * f (2^k)) + (\sum_{k = 2^n..<2^{\text{Suc } n}. f (2^{\text{Discrete.log } k}))$

by (*subst sum.union_disjoint*) (*insert Suc, auto*)

also have *Discrete.log k = n* **if** $k \in \{2^n..<2^{\text{Suc } n}\}$ **for** *k* **using** *that* **by** (*intro Discrete.log_eqI simp_all*)

hence $(\sum_{k = 2^n..<2^{\text{Suc } n}. f (2^{\text{Discrete.log } k})) = (\sum_{_ :: \text{nat}} = 2^n..<2^{\text{Suc } n}. f (2^n))$

by (*intro sum.cong simp_all*)

also have $\dots = 2^n * f (2^n)$ **by** (*simp*)

finally show *?case* **by** *simp*

qed *simp*

private lemma *condensation_condense2*: $(\sum_{k=1..<2^n}. f (2 * 2^{\text{Discrete.log } k})) = (\sum_{k < n}. 2^k * f (2^{\text{Suc } k}))$

proof (*induction n*)

case (*Suc n*)

have $\{1..<2^{\text{Suc } n}\} = \{1..<2^n\} \cup \{2^n..<(2^{\text{Suc } n} :: \text{nat})\}$ **by** *auto*

also have $(\sum_{k \in \dots} f (2 * 2^{\text{Discrete.log } k})) = (\sum_{k < n}. 2^k * f (2^{\text{Suc } k})) + (\sum_{k = 2^n..<2^{\text{Suc } n}. f (2 * 2^{\text{Discrete.log } k}))$

by (*subst sum.union_disjoint*) (*insert Suc, auto*)

also have *Discrete.log k = n* **if** $k \in \{2^n..<2^{\text{Suc } n}\}$ **for** *k* **using** *that* **by** (*intro Discrete.log_eqI simp_all*)

hence $(\sum_{k = 2^n..<2^{\text{Suc } n}. f (2 * 2^{\text{Discrete.log } k})) = (\sum_{_ :: \text{nat}} = 2^n..<2^{\text{Suc } n}. f (2^{\text{Suc } n}))$

by (*intro sum.cong simp_all*)

also have $\dots = 2^n * f (2^{\text{Suc } n})$ **by** (*simp*)

finally show *?case* **by** *simp*

qed *simp*

theorem *condensation_test*:

assumes *mono*: $\bigwedge m. 0 < m \implies f (\text{Suc } m) \leq f m$

assumes *nonneg*: $\bigwedge n. f n \geq 0$

shows *summable f* \longleftrightarrow *summable* $(\lambda n. 2^n * f (2^n))$

proof $-$

define *f'* **where** $f' n = (\text{if } n = 0 \text{ then } 0 \text{ else } f n)$ **for** *n*

```

from mono have mono': decseq ( $\lambda n. f (Suc\ n)$ ) by (intro decseq_SucI) simp
hence mono':  $f\ n \leq f\ m$  if  $m \leq n$   $m > 0$  for  $m\ n$ 
  using that decseqD[OF mono', of m - 1 n - 1] by simp

have ( $\lambda n. f (Suc\ n) = \lambda n. f' (Suc\ n)$ ) by (intro ext) (simp add: f'_def)
hence summable  $f \longleftrightarrow$  summable  $f'$ 
  by (subst (1 2) summable_Suc_iff [symmetric]) (simp only:)
also have  $\dots \longleftrightarrow$  convergent ( $\lambda n. \sum_{k < n}. f' k$ ) unfolding summable_iff_convergent
..
also have monoseq ( $\lambda n. \sum_{k < n}. f' k$ ) unfolding f'_def
  by (intro mono_SucI1) (auto intro!: mult_nonneg_nonneg nonneg)
hence convergent ( $\lambda n. \sum_{k < n}. f' k$ )  $\longleftrightarrow$  Bseq ( $\lambda n. \sum_{k < n}. f' k$ )
  by (rule monoseq_imp_convergent_iff_Bseq)
also have  $\dots \longleftrightarrow$  Bseq ( $\lambda n. \sum_{k=1..<n}. f' k$ ) unfolding One_nat_def
  by (subst sum_shift_lb_Suc0_0_upt) (simp_all add: f'_def atLeast0LessThan)
also have  $\dots \longleftrightarrow$  Bseq ( $\lambda n. \sum_{k=1..<n}. f k$ ) unfolding f'_def by simp
also have  $\dots \longleftrightarrow$  Bseq ( $\lambda n. \sum_{k=1..<2^n}. f k$ )
  by (rule nonneg_incseq_Bseq_subseq_iff [symmetric])
  (auto intro!: sum_nonneg incseq_SucI nonneg simp: strict_mono_def)
also have  $\dots \longleftrightarrow$  Bseq ( $\lambda n. \sum_{k < n}. 2^k * f (2^k)$ )
proof (intro iffI)
  assume  $A: Bseq (\lambda n. \sum_{k=1..<2^n}. f k)$ 
  have eventually ( $\lambda n. norm (\sum_{k < n}. 2^k * f (2^{Suc\ k})) \leq norm (\sum_{k=1..<2^n}. f k)$ ) sequentially
    proof (intro always_eventually allI)
      fix  $n :: nat$ 
      have  $norm (\sum_{k < n}. 2^k * f (2^{Suc\ k})) = (\sum_{k < n}. 2^k * f (2^{Suc\ k}))$ 
unfolding real_norm_def
      by (intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg)
simp_all
      also have  $\dots \leq (\sum_{k=1..<2^n}. f k)$ 
      by (subst condensation_condense2 [symmetric]) (intro condensation_inequality mono')
      also have  $\dots = norm \dots$  unfolding real_norm_def
      by (intro abs_of_nonneg [symmetric] sum_nonneg ballI mult_nonneg_nonneg nonneg)
      finally show  $norm (\sum_{k < n}. 2^k * f (2^{Suc\ k})) \leq norm (\sum_{k=1..<2^n}. f k)$  .
    qed
  from this and  $A$  have Bseq ( $\lambda n. \sum_{k < n}. 2^k * f (2^{Suc\ k})$ ) by (rule Bseq_eventually_mono)
  from Bseq_mult[OF Bfun_const[of 2] this] have Bseq ( $\lambda n. \sum_{k < n}. 2^{Suc\ k} * f (2^{Suc\ k})$ )
    by (simp add: sum_distrib_left sum_distrib_right mult_ac)
  hence Bseq ( $\lambda n. (\sum_{k=Suc\ 0..<Suc\ n}. 2^k * f (2^k)) + f 1$ )
  by (intro Bseq_add, subst sum.shift_bounds_Suc_ivl) (simp add: atLeast0LessThan)
  hence Bseq ( $\lambda n. (\sum_{k=0..<Suc\ n}. 2^k * f (2^k))$ )
    by (subst sum.atLeast_Suc_lessThan) (simp_all add: add_ac)
  thus Bseq ( $\lambda n. (\sum_{k < n}. 2^k * f (2^k))$ )

```

```

    by (subst (asm) Bseq_Suc_iff) (simp add: atLeast0LessThan)
  next
    assume A: Bseq (λn. (∑ k<n. 2^k * f (2^k)))
    have eventually (λn. norm (∑ k=1..<2^n. f k) ≤ norm (∑ k<n. 2^k * f
(2^k))) sequentially
    proof (intro always_eventually allI)
      fix n :: nat
      have norm (∑ k=1..<2^n. f k) = (∑ k=1..<2^n. f k) unfolding real_norm_def
        by (intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg)
      also have ... ≤ (∑ k<n. 2^k * f (2^k))
      by (subst condensation_condense1 [symmetric]) (intro condensation_inequality
mono')
      also have ... = norm ... unfolding real_norm_def
      by (intro abs_of_nonneg [symmetric] sum_nonneg ballI mult_nonneg_nonneg
nonneg) simp_all
      finally show norm (∑ k=1..<2^n. f k) ≤ norm (∑ k<n. 2^k * f (2^k)) .
    qed
  from this and A show Bseq (λn. ∑ k=1..<2^n. f k) by (rule Bseq_eventually_mono)
qed
  also have monoseq (λn. (∑ k<n. 2^k * f (2^k)))
  by (intro mono_SucI1) (auto intro!: mult_nonneg_nonneg nonneg)
  hence Bseq (λn. (∑ k<n. 2^k * f (2^k))) ↔ convergent (λn. (∑ k<n. 2^k *
f (2^k)))
  by (rule monoseq_imp_convergent_iff_Bseq [symmetric])
  also have ... ↔ summable (λk. 2^k * f (2^k)) by (simp only: summable_iff_convergent)
  finally show ?thesis .
qed
end

```

Summability of powers

```

lemma abs_summable_complex_powr_iff:
  summable (λn. norm (exp (of_real (ln (of_nat n)) * s))) ↔ Re s < -1
proof (cases Re s ≤ 0)
  let ?l = λn. complex_of_real (ln (of_nat n))
  case False
  have eventually (λn. norm (1 :: real) ≤ norm (exp (?l n * s))) sequentially
    apply (rule eventually_mono [OF eventually_gt_at_top[of 0::nat]])
    using False ge_one_powr_ge_zero by auto
  from summable_comparison_test_ev[OF this] False show ?thesis by (auto simp:
summable_const_iff)
  next
  let ?l = λn. complex_of_real (ln (of_nat n))
  case True
  hence summable (λn. norm (exp (?l n * s))) ↔ summable (λn. 2^n * norm
(exp (?l (2^n) * s)))
  by (intro condensation_test) (auto intro!: mult_right_mono_neg)
  also have (λn. 2^n * norm (exp (?l (2^n) * s))) = (λn. (2 powr (Re s + 1)) ^

```

n)
proof
 fix n :: nat
 have $2^{\wedge}n * \text{norm} (\exp (?l (2^{\wedge}n) * s)) = \exp (\text{real } n * \ln 2) * \exp (\text{real } n * \ln 2 * \text{Re } s)$
 using True by (subst exp_of_nat_mult) (simp add: ln_realpow algebra_simps)
 also have $\dots = \exp (\text{real } n * (\ln 2 * (\text{Re } s + 1)))$
 by (simp add: algebra_simps exp_add)
 also have $\dots = \exp (\ln 2 * (\text{Re } s + 1))^{\wedge}n$ by (subst exp_of_nat_mult) simp
 also have $\exp (\ln 2 * (\text{Re } s + 1)) = 2 \text{ powr } (\text{Re } s + 1)$ by (simp add: powr_def)
 finally show $2^{\wedge}n * \text{norm} (\exp (?l (2^{\wedge}n) * s)) = (2 \text{ powr } (\text{Re } s + 1))^{\wedge}n$.
qed
 also have $\text{summable } \dots \longleftrightarrow 2 \text{ powr } (\text{Re } s + 1) < 2 \text{ powr } 0$
 by (subst summable_geometric_iff) simp
 also have $\dots \longleftrightarrow \text{Re } s < -1$ by (subst powr_less_cancel_iff) (simp, linarith)
 finally show ?thesis .
qed

theorem summable_complex_powr_iff:
 assumes $\text{Re } s < -1$
 shows $\text{summable } (\lambda n. \exp (\text{of_real } (\ln (\text{of_nat } n)) * s))$
 by (rule summable_norm_cancel, subst abs_summable_complex_powr_iff) fact

lemma summable_real_powr_iff: $\text{summable } (\lambda n. \text{of_nat } n \text{ powr } s :: \text{real}) \longleftrightarrow s < -1$
proof –
 from eventually_gt_at_top[of 0::nat]
 have $\text{summable } (\lambda n. \text{of_nat } n \text{ powr } s) \longleftrightarrow \text{summable } (\lambda n. \exp (\ln (\text{of_nat } n) * s))$
 by (intro summable_cong) (auto elim!: eventually_mono simp: powr_def)
 also have $\dots \longleftrightarrow s < -1$ using abs_summable_complex_powr_iff[of of_real s] by simp
 finally show ?thesis .
qed

lemma inverse_power_summable:
 assumes $s: s \geq 2$
 shows $\text{summable } (\lambda n. \text{inverse } (\text{of_nat } n^{\wedge}s :: 'a :: \{\text{real_normed_div_algebra, banach}\}))$
proof (rule summable_norm_cancel, subst summable_cong)
 from eventually_gt_at_top[of 0::nat]
 show $\text{eventually } (\lambda n. \text{norm } (\text{inverse } (\text{of_nat } n^{\wedge}s :: 'a))) = \text{real_of_nat } n \text{ powr } (-\text{real } s)$ at_top
 by eventually_elim (simp add: norm_inverse norm_power powr_minus powr_realpow)
qed (insert s summable_real_powr_iff[of -s], simp_all)

lemma not_summable_harmonic: $\neg \text{summable } (\lambda n. \text{inverse } (\text{of_nat } n) :: 'a :: \text{real_normed_field})$
proof

```

assume summable ( $\lambda n. \text{inverse } (\text{of\_nat } n) :: 'a$ )
hence convergent ( $\lambda n. \text{norm } (\text{of\_real } (\sum k < n. \text{inverse } (\text{of\_nat } k))) :: 'a$ )
  by (simp add: summable_iff_convergent convergent_norm)
hence convergent ( $\lambda n. \text{abs } (\sum k < n. \text{inverse } (\text{of\_nat } k)) :: \text{real}$ ) by (simp only:
norm_of_real)
also have ( $\lambda n. \text{abs } (\sum k < n. \text{inverse } (\text{of\_nat } k)) :: \text{real}$ ) = ( $\lambda n. \sum k < n. \text{inverse}$ 
( $\text{of\_nat } k$ ))
  by (intro ext abs_of_nonneg sum_nonneg) auto
also have convergent ...  $\longleftrightarrow$  summable ( $\lambda k. \text{inverse } (\text{of\_nat } k) :: \text{real}$ )
  by (simp add: summable_iff_convergent)
finally show False using summable_real_pour_iff[of -1] by (simp add: pour_minus)
qed

```

Kummer's test

theorem *kummers_test_convergence*:

fixes $f p :: \text{nat} \Rightarrow \text{real}$

assumes *pos_f*: eventually ($\lambda n. f n > 0$) sequentially

assumes *nonneg_p*: eventually ($\lambda n. p n \geq 0$) sequentially

defines $l \equiv \text{liminf } (\lambda n. \text{ereal } (p n * f n / f (\text{Suc } n) - p (\text{Suc } n)))$

assumes $l: l > 0$

shows summable f

unfolding summable_iff_convergent'

proof –

define r **where** $r = (\text{if } l = \infty \text{ then } 1 \text{ else } \text{real_of_ereal } l / 2)$

from l **have** $r > 0 \wedge \text{of_real } r < l$ **by** (cases l) (simp_all add: r_def)

hence $r: r > 0 \text{ of_real } r < l$ **by** simp_all

hence eventually ($\lambda n. p n * f n / f (\text{Suc } n) - p (\text{Suc } n) > r$) sequentially

unfolding l_def **by** (force dest: less_LiminfD)

moreover from *pos_f* **have** eventually ($\lambda n. f (\text{Suc } n) > 0$) sequentially

by (subst eventually_sequentially_Suc)

ultimately have eventually ($\lambda n. p n * f n - p (\text{Suc } n) * f (\text{Suc } n) > r * f (\text{Suc } n)$) sequentially

by eventually_elim (simp add: field_simps)

from eventually_conj[OF *pos_f* eventually_conj[OF *nonneg_p* this]]

obtain m **where** $m: \bigwedge n. n \geq m \implies f n > 0 \wedge \bigwedge n. n \geq m \implies p n \geq 0$

$\bigwedge n. n \geq m \implies p n * f n - p (\text{Suc } n) * f (\text{Suc } n) > r * f (\text{Suc } n)$

unfolding eventually_at_top_linorder **by** blast

let $?c = (\text{norm } (\sum k \leq m. r * f k) + p m * f m) / r$

have Bseq ($\lambda n. (\sum k \leq n + \text{Suc } m. f k)$)

proof (rule BseqI')

fix $k :: \text{nat}$

define n **where** $n = k + \text{Suc } m$

have $n: n > m$ **by** (simp add: n_def)

from r **have** $r * \text{norm } (\sum k \leq n. f k) = \text{norm } (\sum k \leq n. r * f k)$

by (simp add: sum_distrib_left[symmetric] abs_mult)

also from n **have** $\{..n\} = \{..m\} \cup \{\text{Suc } m..n\}$ **by** auto

hence $(\sum k \leq n. r * f k) = (\sum k \in \{..m\} \cup \{Suc\ m..n\}. r * f k)$ **by** (*simp only*:)
also have $\dots = (\sum k \leq m. r * f k) + (\sum k = Suc\ m..n. r * f k)$
by (*subst sum.union_disjoint*) *auto*
also have $norm \dots \leq norm (\sum k \leq m. r * f k) + norm (\sum k = Suc\ m..n. r * f k)$
by (*rule norm_triangle_ineq*)
also from $r\ less_imp_le[OF\ m(1)]$ **have** $(\sum k = Suc\ m..n. r * f k) \geq 0$
by (*intro sum_nonneg*) *auto*
hence $norm (\sum k = Suc\ m..n. r * f k) = (\sum k = Suc\ m..n. r * f k)$ **by** *simp*
also have $(\sum k = Suc\ m..n. r * f k) = (\sum k = m..<n. r * f (Suc\ k))$
by (*subst sum.shift_bounds_Suc_ivl* [*symmetric*])
(simp only: atLeastLessThanSuc_atLeastAtMost)
also from m **have** $\dots \leq (\sum k = m..<n. p\ k * f k - p (Suc\ k) * f (Suc\ k))$
by (*intro sum_mono*[*OF less_imp_le*]) *simp_all*
also have $\dots = -(\sum k = m..<n. p (Suc\ k) * f (Suc\ k) - p\ k * f k)$
by (*simp add: sum_negf* [*symmetric*] *algebra_simps*)
also from n **have** $\dots = p\ m * f\ m - p\ n * f\ n$
by (*cases n, simp, simp only: atLeastLessThanSuc_atLeastAtMost, subst sum_Suc_diff*) *simp_all*
also from $less_imp_le[OF\ m(1)]\ m(2)\ n$ **have** $\dots \leq p\ m * f\ m$ **by** *simp*
finally show $norm (\sum k \leq n. f k) \leq (norm (\sum k \leq m. r * f k) + p\ m * f\ m) / r$
using r
by (*subst pos_le_divide_eq*[*OF r(1)*]) (*simp only: mult_ac*)
qed
moreover have $(\sum k \leq n. f k) \leq (\sum k \leq n'. f k)$ **if** $Suc\ m \leq n \leq n'$ **for** $n\ n'$
using $less_imp_le[OF\ m(1)]$ **that** **by** (*intro sum_mono2*) *auto*
ultimately show $convergent (\lambda n. \sum k \leq n. f k)$ **by** (*rule Bseq_monoseq_convergent'_inc*)
qed

theorem kummers_test_divergence:

fixes $f\ p :: nat \Rightarrow real$
assumes $pos_f: eventually (\lambda n. f\ n > 0)$ *sequentially*
assumes $pos_p: eventually (\lambda n. p\ n > 0)$ *sequentially*
assumes $divergent_p: \neg summable (\lambda n. inverse (p\ n))$
defines $l \equiv limsup (\lambda n. ereal (p\ n * f\ n / f (Suc\ n) - p (Suc\ n)))$
assumes $l: l < 0$
shows $\neg summable\ f$
proof
assume $summable\ f$
from $eventually_conj[OF\ pos_f\ eventually_conj[OF\ pos_p\ Limsup_lessD[OF\ l[unfolded\ l_def]]]]$
obtain N **where** $\bigwedge n. n \geq N \implies p\ n > 0 \wedge n. n \geq N \implies f\ n > 0$
 $\bigwedge n. n \geq N \implies p\ n * f\ n / f (Suc\ n) - p (Suc\ n) < 0$
by (*auto simp: eventually_at_top_linorder*)
hence $A: p\ n * f\ n < p (Suc\ n) * f (Suc\ n)$ **if** $n \geq N$ **for** n **using** $that\ N[of\ n]$
 $N[of\ Suc\ n]$
by (*simp add: field_simps*)
have $B: p\ n * f\ n \geq p\ N * f\ N$ **if** $n \geq N$ **for** n **using** $that$ **and** A

```

  by (induction n rule: dec_induct) (auto intro!: less_imp_le elim!: order.trans)
  have eventually ( $\lambda n. \text{norm } (p \ N * f \ N * \text{inverse } (p \ n)) \leq f \ n$ ) sequentially
  apply (rule eventually_mono [OF eventually_ge_at_top[of N]])
  using B N by (auto simp: field_simps abs_of_pos)
  from this and <summable f> have summable ( $\lambda n. p \ N * f \ N * \text{inverse } (p \ n)$ )
  by (rule summable_comparison_test_ev)
  from summable_mult[OF this, of inverse (p N * f N)] N[OF le_refl]
  have summable ( $\lambda n. \text{inverse } (p \ n)$ ) by (simp add: field_split_simps)
  with divergent_p show False by contradiction
qed

```

Ratio test

theorem ratio_test_convergence:

```

  fixes f :: nat  $\Rightarrow$  real
  assumes pos_f: eventually ( $\lambda n. f \ n > 0$ ) sequentially
  defines l  $\equiv$  liminf ( $\lambda n. \text{ereal } (f \ n / f \ (\text{Suc } n))$ )
  assumes l: l > 1
  shows summable f
proof (rule kummers_test_convergence[OF pos_f])
  note l
  also have l = liminf ( $\lambda n. \text{ereal } (f \ n / f \ (\text{Suc } n) - 1)) + 1$ 
  by (subst Liminf_add_ereal_right[symmetric]) (simp_all add: minus_ereal_def
l_def one_ereal_def)
  finally show liminf ( $\lambda n. \text{ereal } (1 * f \ n / f \ (\text{Suc } n) - 1)) > 0$ 
  by (cases liminf ( $\lambda n. \text{ereal } (1 * f \ n / f \ (\text{Suc } n) - 1))$ ) simp_all
qed simp

```

theorem ratio_test_divergence:

```

  fixes f :: nat  $\Rightarrow$  real
  assumes pos_f: eventually ( $\lambda n. f \ n > 0$ ) sequentially
  defines l  $\equiv$  limsup ( $\lambda n. \text{ereal } (f \ n / f \ (\text{Suc } n))$ )
  assumes l: l < 1
  shows  $\neg$ summable f
proof (rule kummers_test_divergence[OF pos_f])
  have limsup ( $\lambda n. \text{ereal } (f \ n / f \ (\text{Suc } n) - 1)) + 1 = l$ 
  by (subst Limsup_add_ereal_right[symmetric]) (simp_all add: minus_ereal_def
l_def one_ereal_def)
  also note l
  finally show limsup ( $\lambda n. \text{ereal } (1 * f \ n / f \ (\text{Suc } n) - 1)) < 0$ 
  by (cases limsup ( $\lambda n. \text{ereal } (1 * f \ n / f \ (\text{Suc } n) - 1)$ )) simp_all
qed (simp_all add: summable_const_iff)

```

Raabe's test

theorem raabes_test_convergence:

```

  fixes f :: nat  $\Rightarrow$  real
  assumes pos: eventually ( $\lambda n. f \ n > 0$ ) sequentially
  defines l  $\equiv$  liminf ( $\lambda n. \text{ereal } (\text{of\_nat } n * (f \ n / f \ (\text{Suc } n) - 1))$ )
  assumes l: l > 1

```

```

  shows summable f
proof (rule kummers_test_convergence)
  let ?l' = liminf (λn. ereal (of_nat n * f n / f (Suc n) - of_nat (Suc n)))
  have 1 < l by fact
  also have l = liminf (λn. ereal (of_nat n * f n / f (Suc n) - of_nat (Suc n))
+ 1)
  by (simp add: l_def algebra_simps)
  also have ... = ?l' + 1 by (subst Liminf_add_ereal_right) simp_all
  finally show ?l' > 0 by (cases ?l') (simp_all add: algebra_simps)
qed (simp_all add: pos)

```

theorem raabes_test_divergence:

```

fixes f :: nat ⇒ real
  assumes pos: eventually (λn. f n > 0) sequentially
  defines l ≡ limsup (λn. ereal (of_nat n * (f n / f (Suc n) - 1)))
  assumes l: l < 1
  shows ¬summable f
proof (rule kummers_test_divergence)
  let ?l' = limsup (λn. ereal (of_nat n * f n / f (Suc n) - of_nat (Suc n)))
  note l
  also have l = limsup (λn. ereal (of_nat n * f n / f (Suc n) - of_nat (Suc n))
+ 1)
  by (simp add: l_def algebra_simps)
  also have ... = ?l' + 1 by (subst Limsup_add_ereal_right) simp_all
  finally show ?l' < 0 by (cases ?l') (simp_all add: algebra_simps)
qed (insert pos eventually_gt_at_top[of 0::nat] not_summable_harmonic, simp_all)

```

6.14.2 Radius of convergence

The radius of convergence of a power series. This value always exists, ranges from 0 to ∞ , and the power series is guaranteed to converge for all inputs with a norm that is smaller than that radius and to diverge for all inputs with a norm that is greater.

definition *conv_radius* :: (nat ⇒ 'a :: banach) ⇒ ereal **where**
conv_radius f = inverse (limsup (λn. ereal (root n (norm (f n)))))

lemma *conv_radius_cong_weak* [cong]: (Λn. f n = g n) ⇒ *conv_radius* f = *conv_radius* g
 by (drule ext) simp_all

lemma *conv_radius_nonneg*: *conv_radius* f ≥ 0

```

proof -
  have 0 = limsup (λn. 0) by (subst Limsup_const) simp_all
  also have ... ≤ limsup (λn. ereal (root n (norm (f n))))
  by (intro Limsup_mono) (simp_all add: real_root_ge_zero)
  finally show ?thesis
  unfolding conv_radius_def by (auto simp: ereal_inverse_nonneg_iff)
qed

```

lemma *conv_radius_zero* [simp]: $\text{conv_radius } (\lambda_. 0) = \infty$
by (*auto simp: conv_radius_def zero_ereal_def [symmetric] Limsup_const*)

lemma *conv_radius_altdef*:
 $\text{conv_radius } f = \text{liminf } (\lambda n. \text{inverse } (\text{ereal } (\text{root } n (\text{norm } (f n))))))$
by (*subst Liminf_inverse_ereal*) (*simp_all add: real_root_ge_zero conv_radius_def*)

lemma *conv_radius_cong'*:
assumes *eventually* $(\lambda x. f x = g x)$ *sequentially*
shows $\text{conv_radius } f = \text{conv_radius } g$
unfolding *conv_radius_altdef* **by** (*intro Liminf_eq eventually_mono [OF assms]*)
auto

lemma *conv_radius_cong*:
assumes *eventually* $(\lambda x. \text{norm } (f x) = \text{norm } (g x))$ *sequentially*
shows $\text{conv_radius } f = \text{conv_radius } g$
unfolding *conv_radius_altdef* **by** (*intro Liminf_eq eventually_mono [OF assms]*)
auto

theorem *abs_summable_in_conv_radius*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach}, \text{real_normed_div_algebra}\}$
assumes $\text{ereal } (\text{norm } z) < \text{conv_radius } f$
shows *summable* $(\lambda n. \text{norm } (f n * z ^ n))$
proof (*rule root_test_convergence'*)
define l **where** $l = \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f n))))$
have $0 = \text{limsup } (\lambda n. 0)$ **by** (*simp add: Limsup_const*)
also have $\dots \leq l$ **unfolding** l_def **by** (*intro Limsup_mono*) (*simp_all add: real_root_ge_zero*)
finally have $l_nonneg: l \geq 0$.

have $\text{limsup } (\lambda n. \text{root } n (\text{norm } (f n * z ^ n))) = l * \text{ereal } (\text{norm } z)$ **unfolding**
 l_def

by (*rule limsup_root_powser*)

also from l_nonneg **consider** $l = 0 \mid l = \infty \mid \exists l'. l = \text{ereal } l' \wedge l' > 0$

by (*cases l*) (*auto simp: less_le*)

hence $l * \text{ereal } (\text{norm } z) < 1$

proof cases

assume $l = \infty$

hence $\text{conv_radius } f = 0$ **unfolding** *conv_radius_def* l_def **by** *simp*

with *assms* **show** *?thesis* **by** *simp*

next

assume $\exists l'. l = \text{ereal } l' \wedge l' > 0$

then obtain l' **where** $l' : l = \text{ereal } l' \ 0 < l'$ **by** *auto*

hence $l \neq \infty$ **by** *auto*

have $l * \text{ereal } (\text{norm } z) < l * \text{conv_radius } f$

by (*intro ereal_mult_strict_left_mono*) (*simp_all add: l' assms*)

also have $\text{conv_radius } f = \text{inverse } l$ **by** (*simp add: conv_radius_def l_def*)

also from l' **have** $l * \text{inverse } l = 1$ **by** *simp*

```

  finally show ?thesis .
qed simp_all
  finally show  $\limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm} (\text{norm} (f \ n * z \ ^n)))) < 1$  by
simp
qed

```

lemma *summable_in_conv_radius*:

```

  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\text{ereal} (\text{norm } z) < \text{conv\_radius } f$ 
  shows summable  $(\lambda n. f \ n * z \ ^n)$ 
  by (rule summable_norm_cancel, rule abs_summable_in_conv_radius) fact+

```

theorem *not_summable_outside_conv_radius*:

```

  fixes f :: nat  $\Rightarrow$  'a :: {banach, real_normed_div_algebra}
  assumes  $\text{ereal} (\text{norm } z) > \text{conv\_radius } f$ 
  shows  $\neg \text{summable} (\lambda n. f \ n * z \ ^n)$ 
proof (rule root_test_divergence)
  define l where  $l = \limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm} (f \ n))))$ 
  have  $0 = \limsup (\lambda n. 0)$  by (simp add: Limsup_const)
  also have  $\dots \leq l$  unfolding l_def by (intro Limsup_mono) (simp_all add:
real_root_ge_zero)
  finally have l_nonneg:  $l \geq 0$  .
  from assms have l_nz:  $l \neq 0$  unfolding conv_radius_def l_def by auto

```

```

  have  $\limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm} (f \ n * z \ ^n)))) = l * \text{ereal} (\text{norm } z)$ 
  unfolding l_def by (rule limsup_root_powser)

```

```

  also have  $\dots > 1$ 

```

```

proof (cases l)

```

```

  assume  $l = \infty$ 

```

```

  with assms conv_radius_nonneg[of f] show ?thesis

```

```

  by (auto simp: zero_ereal_def[symmetric])

```

```

next

```

```

  fix l' assume l':  $l = \text{ereal } l'$ 

```

```

  from l_nonneg l_nz have  $1 = l * \text{inverse } l$  by (auto simp: l' field_simps)

```

```

  also from l_nz have  $\text{inverse } l = \text{conv\_radius } f$ 

```

```

  unfolding l_def conv_radius_def by auto

```

```

  also from l' l_nz l_nonneg assms have  $l * \dots < l * \text{ereal} (\text{norm } z)$ 

```

```

  by (intro  $\text{ereal\_mult\_strict\_left\_mono}$ ) (auto simp: l')

```

```

  finally show ?thesis .

```

```

qed (insert l_nonneg, simp_all)

```

```

  finally show  $\limsup (\lambda n. \text{ereal} (\text{root } n (\text{norm} (f \ n * z \ ^n)))) > 1$  .

```

```

qed

```

lemma *conv_radius_geI*:

```

  assumes summable  $(\lambda n. f \ n * z \ ^n)$  :: 'a :: {banach, real_normed_div_algebra}

```

```

  shows  $\text{conv\_radius } f \geq \text{norm } z$ 

```

```

  using not_summable_outside_conv_radius[of f z] assms by (force simp: not_le[symmetric])

```

lemma *conv_radius_leI*:

assumes \neg summable $(\lambda n. \text{norm } (f n * z^{\wedge} n) :: 'a :: \{\text{banach, real_normed_div_algebra}\})$
shows $\text{conv_radius } f \leq \text{norm } z$
using *abs_summable_in_conv_radius*[of z] *assms* **by** (*force simp: not_le[symmetric]*)

lemma *conv_radius_leI'*:

assumes \neg summable $(\lambda n. f n * z^{\wedge} n :: 'a :: \{\text{banach, real_normed_div_algebra}\})$
shows $\text{conv_radius } f \leq \text{norm } z$
using *summable_in_conv_radius*[of z] *assms* **by** (*force simp: not_le[symmetric]*)

lemma *conv_radius_geI_ex*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f n * z^{\wedge} n)$
shows $\text{conv_radius } f \geq R$
proof (*rule linorder_cases*[of *conv_radius* f R])
assume $R: \text{conv_radius } f < R$
with *conv_radius_nonneg*[of f] **obtain** $\text{conv_radius}'$
where [*simp*]: $\text{conv_radius } f = \text{ereal } \text{conv_radius}'$
by (*cases conv_radius* f) *simp_all*
define r **where** $r = (\text{if } R = \infty \text{ then } \text{conv_radius}' + 1 \text{ else } (\text{real_of_ereal } R + \text{conv_radius}') / 2)$
from R *conv_radius_nonneg*[of f] **have** $0 < r \wedge \text{ereal } r < R \wedge \text{ereal } r > \text{conv_radius } f$
unfolding r_def **by** (*cases* R) (*auto simp: r_def field_simps*)
with *assms*(1)[of r] **obtain** z **where** $\text{norm } z > \text{conv_radius } f$ *summable* $(\lambda n. f n * z^{\wedge} n)$ **by** *auto*
with *not_summable_outside_conv_radius*[of f z] **show** *?thesis* **by** *simp*
qed *simp_all*

lemma *conv_radius_geI_ex'*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge r. 0 < r \implies \text{ereal } r < R \implies \text{summable } (\lambda n. f n * \text{of_real } r^{\wedge} n)$
shows $\text{conv_radius } f \geq R$
proof (*rule conv_radius_geI_ex*)
fix r **assume** $0 < r$ *ereal* $r < R$
with *assms*[of r] **show** $\exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f n * z^{\wedge} n)$
by (*intro exI*[of $_$ of *real* $r :: 'a$]) *auto*
qed

lemma *conv_radius_leI_ex*:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $R \geq 0$
assumes $\bigwedge r. 0 < r \implies \text{ereal } r > R \implies \exists z. \text{norm } z = r \wedge \neg \text{summable } (\lambda n. \text{norm } (f n * z^{\wedge} n))$
shows $\text{conv_radius } f \leq R$
proof (*rule linorder_cases*[of *conv_radius* f R])
assume $R: \text{conv_radius } f > R$
from R *assms*(1) **obtain** R' **where** $R': R = \text{ereal } R'$ **by** (*cases* R) *simp_all*

```

define r where
  r = (if conv_radius f = ∞ then R' + 1 else (R' + real_of_ereal (conv_radius
f)) / 2)
from R conv_radius_nonneg[of f] have r > R ∧ r < conv_radius f unfolding
r_def
by (cases conv_radius f) (auto simp: r_def field_simps R')
with assms(1) assms(2)[of r] R'
obtain z where norm z < conv_radius f ¬summable (λn. norm (f n * zn))
by auto
with abs_summable_in_conv_radius[of z f] show ?thesis by auto
qed simp_all

```

```

lemma conv_radius_leI_ex':
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes R ≥ 0
  assumes ∧r. 0 < r ⇒ ereal r > R ⇒ ¬summable (λn. f n * of_real rn)
  shows conv_radius f ≤ R
proof (rule conv_radius_leI_ex)
  fix r assume 0 < r ereal r > R
  with assms(2)[of r] show ∃z. norm z = r ∧ ¬summable (λn. norm (f n * zn
n))
  by (intro exI[of _ of_real r :: 'a]) (auto dest: summable_norm_cancel)
qed fact+

```

```

lemma conv_radius_eqI:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes R ≥ 0
  assumes ∧r. 0 < r ⇒ ereal r < R ⇒ ∃z. norm z = r ∧ summable (λn. f n
* zn)
  assumes ∧r. 0 < r ⇒ ereal r > R ⇒ ∃z. norm z = r ∧ ¬summable (λn.
norm (f n * zn))
  shows conv_radius f = R
  by (intro antisym conv_radius_geI_ex conv_radius_leI_ex assms)

```

```

lemma conv_radius_eqI':
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes R ≥ 0
  assumes ∧r. 0 < r ⇒ ereal r < R ⇒ summable (λn. f n * (of_real r)n)
  assumes ∧r. 0 < r ⇒ ereal r > R ⇒ ¬summable (λn. norm (f n * (of_real
r)n))
  shows conv_radius f = R
proof (intro conv_radius_eqI[OF assms(1)])
  fix r assume 0 < r ereal r < R with assms(2)[OF this]
  show ∃z. norm z = r ∧ summable (λn. f n * zn) by force
next
  fix r assume 0 < r ereal r > R with assms(3)[OF this]
  show ∃z. norm z = r ∧ ¬summable (λn. norm (f n * zn)) by force
qed

```

lemma *conv_radius_zeroI*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge z. z \neq 0 \implies \neg \text{summable } (\lambda n. f\ n * z^{\wedge} n)$
shows $\text{conv_radius } f = 0$
proof (rule *ccontr*)
assume $\text{conv_radius } f \neq 0$
with *conv_radius_nonneg*[of f] **have** $\text{pos: conv_radius } f > 0$ **by** *simp*
define r **where** $r = (\text{if } \text{conv_radius } f = \infty \text{ then } 1 \text{ else } \text{real_of_ereal } (\text{conv_radius } f) / 2)$
from pos **have** $r: \text{ereal } r > 0 \wedge \text{ereal } r < \text{conv_radius } f$
by (*cases conv_radius f*) (*simp_all add: r_def*)
hence $\text{summable } (\lambda n. f\ n * \text{of_real } r^{\wedge} n)$ **by** (*intro summable_in_conv_radius*)
simp
moreover from r **and** *assms*[of *of_real r*] **have** $\neg \text{summable } (\lambda n. f\ n * \text{of_real } r^{\wedge} n)$ **by** *simp*
ultimately show *False* **by** *contradiction*
qed

lemma *conv_radius_inftyI'*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge r. r > c \implies \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^{\wedge} n)$
shows $\text{conv_radius } f = \infty$
proof –
{
fix $r :: \text{real}$
have $\text{max } r (c + 1) > c$ **by** (*auto simp: max_def*)
from *assms*[OF *this*] **obtain** z **where** $\text{norm } z = \text{max } r (c + 1)$ $\text{summable } (\lambda n. f\ n * z^{\wedge} n)$ **by** *blast*
from *conv_radius_geI*[OF *this*(2)] *this*(1) **have** $\text{conv_radius } f \geq r$ **by** *simp*
}
from *this*[of *real_of_ereal (conv_radius f + 1)*] **show** $\text{conv_radius } f = \infty$
by (*cases conv_radius f*) *simp_all*
qed

lemma *conv_radius_inftyI*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge r. \exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^{\wedge} n)$
shows $\text{conv_radius } f = \infty$
using *assms* **by** (rule *conv_radius_inftyI'*)

lemma *conv_radius_inftyI''*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $\bigwedge z. \text{summable } (\lambda n. f\ n * z^{\wedge} n)$
shows $\text{conv_radius } f = \infty$
proof (rule *conv_radius_inftyI'*)
fix $r :: \text{real}$ **assume** $r > 0$
with *assms* **show** $\exists z. \text{norm } z = r \wedge \text{summable } (\lambda n. f\ n * z^{\wedge} n)$
by (*intro exI*[of *of_real r*]) *simp*
qed


```

lemma conv_radius_conv_Sup:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  shows conv_radius f = Sup {r. ∀z. ereal (norm z) < r ⟶ summable (λn. f n
  * z ^ n)}
proof (rule Sup_eqI [symmetric], goal_cases)
  case (1 r)
  thus ?case
    by (intro conv_radius_geI_ex') auto
next
  case (2 r)
  from 2[of 0] have r: r ≥ 0 by auto
  show ?case
  proof (intro conv_radius_leI_ex' r)
    fix R assume R: R > 0 ereal R > r
    with r obtain r' where [simp]: r = ereal r' by (cases r) auto
    show ¬summable (λn. f n * of_real R ^ n)
  proof
    assume *: summable (λn. f n * of_real R ^ n)
    define R' where R' = (R + r') / 2
    from R have R': R' > r' R' < R by (simp_all add: R'_def)
    hence ∀z. norm z < R' ⟶ summable (λn. f n * z ^ n)
      using powser_inside[OF *] by auto
    from 2[of R'] and this have R' ≤ r' by auto
    with ⟨R' > r'⟩ show False by simp
  qed
  qed
qed

```

```

lemma conv_radius_shift:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  shows conv_radius (λn. f (n + m)) = conv_radius f
  unfolding conv_radius_conv_Sup summable_powser_ignore_initial_segment ..

```

```

lemma conv_radius_norm [simp]: conv_radius (λx. norm (f x)) = conv_radius f
  by (simp add: conv_radius_def)

```

```

lemma conv_radius_ratio_limit_ereal:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes nz: eventually (λn. f n ≠ 0) sequentially
  assumes lim: (λn. ereal (norm (f n) / norm (f (Suc n)))) ⟶ c
  shows conv_radius f = c
proof (rule conv_radius_eqI')
  show c ≥ 0 by (intro Lim_bounded2[OF lim]) simp_all
next
  fix r assume r: 0 < r ereal r < c
  let ?l = liminf (λn. ereal (norm (f n * of_real r ^ n) / norm (f (Suc n) * of_real
  r ^ Suc n)))
  have ?l = liminf (λn. ereal (norm (f n) / (norm (f (Suc n)))) * ereal (inverse

```

```

r))
  using r by (simp add: norm_mult norm_power field_split_simps)
  also from r have ... = liminf (λn. ereal (norm (f n) / (norm (f (Suc n)))) *
ereal (inverse r)
    by (intro Liminf_ereal_mult_right) simp_all
  also have liminf (λn. ereal (norm (f n) / (norm (f (Suc n)))) = c
    by (intro lim_imp_Liminf lim) simp
  finally have l: ?l = c * ereal (inverse r) by simp
from r have l': c * ereal (inverse r) > 1 by (cases c) (simp_all add: field_simps)
show summable (λn. f n * of_real r ^ n)
  by (rule summable_norm_cancel, rule ratio_test_convergence)
    (insert r nz l l', auto elim!: eventually_mono)
next
  fix r assume r: 0 < r ereal r > c
  let ?l = limsup (λn. ereal (norm (f n * of_real r ^ n) / norm (f (Suc n) *
of_real r ^ Suc n)))
  have ?l = limsup (λn. ereal (norm (f n) / (norm (f (Suc n)))) * ereal (inverse
r))
    using r by (simp add: norm_mult norm_power field_split_simps)
  also from r have ... = limsup (λn. ereal (norm (f n) / (norm (f (Suc n)))) *
ereal (inverse r)
    by (intro Limsup_ereal_mult_right) simp_all
  also have limsup (λn. ereal (norm (f n) / (norm (f (Suc n)))) = c
    by (intro lim_imp_Limsup lim) simp
  finally have l: ?l = c * ereal (inverse r) by simp
from r have l': c * ereal (inverse r) < 1 by (cases c) (simp_all add: field_simps)
show ¬summable (λn. norm (f n * of_real r ^ n))
  by (rule ratio_test_divergence) (insert r nz l l', auto elim!: eventually_mono)
qed

```

```

lemma conv_radius_ratio_limit_ereal_nonzero:
  fixes f :: nat ⇒ 'a :: {banach,real_normed_div_algebra}
  assumes nz: c ≠ 0
  assumes lim: (λn. ereal (norm (f n) / norm (f (Suc n)))) ⟶ c
  shows conv_radius f = c
proof (rule conv_radius_ratio_limit_ereal[OF _ lim], rule ccontr)
  assume ¬eventually (λn. f n ≠ 0) sequentially
  hence frequently (λn. f n = 0) sequentially by (simp add: frequently_def)
  hence frequently (λn. ereal (norm (f n) / norm (f (Suc n))) = 0) sequentially
    by (force elim!: frequently_elim1)
  hence c = 0 by (intro limit_frequently_eq[OF _ _ lim]) auto
  with nz show False by contradiction
qed

```

```

lemma conv_radius_ratio_limit:
  fixes f :: nat ⇒ 'a :: {banach,real_normed_div_algebra}
  assumes c' = ereal c
  assumes nz: eventually (λn. f n ≠ 0) sequentially
  assumes lim: (λn. norm (f n) / norm (f (Suc n))) ⟶ c

```

shows $\text{conv_radius } f = c'$
using *assms* **by** (intro conv_radius_ratio_limit_ereal) simp_all

lemma conv_radius_ratio_limit_nonzero:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, real_normed_div_algebra}\}$
assumes $c' = \text{ereal } c$
assumes $\text{nz}: c \neq 0$
assumes $\text{lim}: (\lambda n. \text{norm } (f n) / \text{norm } (f (\text{Suc } n))) \longrightarrow c$
shows $\text{conv_radius } f = c'$
using *assms* **by** (intro conv_radius_ratio_limit_ereal_nonzero) simp_all

lemma conv_radius_cmult_left:

assumes $c \neq (0 :: 'a :: \{\text{banach, real_normed_div_algebra}\})$
shows $\text{conv_radius } (\lambda n. c * f n) = \text{conv_radius } f$
proof –
have $\text{conv_radius } (\lambda n. c * f n) =$
 $\text{inverse } (\text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (c * f n))))))$
unfolding conv_radius_def ..
also have $(\lambda n. \text{ereal } (\text{root } n (\text{norm } (c * f n)))) =$
 $(\lambda n. \text{ereal } (\text{root } n (\text{norm } c)) * \text{ereal } (\text{root } n (\text{norm } (f n))))$
by (rule ext) (auto simp: norm_mult real_root_mult)
also have $\text{limsup } \dots = \text{ereal } 1 * \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f n))))$
using *assms* **by** (intro ereal_limsup_lim_mult tendsto_ereal LIMSEQ_root_const)
auto
also have $\text{inverse } \dots = \text{conv_radius } f$ **by** (simp add: conv_radius_def)
finally show ?thesis .

qed

lemma conv_radius_cmult_right:

assumes $c \neq (0 :: 'a :: \{\text{banach, real_normed_div_algebra}\})$
shows $\text{conv_radius } (\lambda n. f n * c) = \text{conv_radius } f$
proof –
have $\text{conv_radius } (\lambda n. f n * c) = \text{conv_radius } (\lambda n. c * f n)$
by (simp add: conv_radius_def norm_mult mult.commute)
with conv_radius_cmult_left[OF *assms*, of f] **show** ?thesis **by** simp

qed

lemma conv_radius_mult_power:

assumes $c \neq (0 :: 'a :: \{\text{real_normed_div_algebra, banach}\})$
shows $\text{conv_radius } (\lambda n. c ^ n * f n) = \text{conv_radius } f / \text{ereal } (\text{norm } c)$
proof –
have $\text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (c ^ n * f n)))) =$
 $\text{limsup } (\lambda n. \text{ereal } (\text{norm } c) * \text{ereal } (\text{root } n (\text{norm } (f n))))$
by (intro Limsup_eq eventually_mono [OF eventually_gt_at_top[of 0::nat]])
 $(\text{auto simp: norm_mult norm_power real_root_mult real_root_power})$
also have $\dots = \text{ereal } (\text{norm } c) * \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f n))))$
using *assms* **by** (subst Limsup_ereal_mult_left[symmetric]) simp_all
finally have $A: \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (c ^ n * f n)))) =$
 $\text{ereal } (\text{norm } c) * \text{limsup } (\lambda n. \text{ereal } (\text{root } n (\text{norm } (f n)))) .$

```

show ?thesis using assms
  apply (cases limsup ( $\lambda n. \text{ereal} (\text{root } n (\text{norm } (f \ n))) = 0$ ))
  apply (simp add: A conv_radius_def)
  apply (unfold conv_radius_def A divide_ereal_def, simp add: mult.commute
ereal_inverse_mult)
  done
qed

```

```

lemma conv_radius_mult_power_right:
  assumes  $c \neq (0 :: 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\})$ 
  shows  $\text{conv\_radius } (\lambda n. f \ n * c ^ n) = \text{conv\_radius } f / \text{ereal } (\text{norm } c)$ 
  using conv_radius_mult_power[OF assms, of f]
  unfolding conv_radius_def by (simp add: mult.commute norm_mult)

```

```

lemma conv_radius_divide_power:
  assumes  $c \neq (0 :: 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\})$ 
  shows  $\text{conv\_radius } (\lambda n. f \ n / c ^ n) = \text{conv\_radius } f * \text{ereal } (\text{norm } c)$ 
proof -
  from assms have inverse c ≠ 0 by simp
  from conv_radius_mult_power_right[OF this, of f] show ?thesis
  by (simp add: divide_inverse divide_ereal_def assms norm_inverse power_inverse)
qed

```

```

lemma conv_radius_add_ge:
   $\min (\text{conv\_radius } f) (\text{conv\_radius } g) \leq$ 
   $\text{conv\_radius } (\lambda x. f \ x + g \ x :: 'a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\})$ 
  by (rule conv_radius_geI_ex')
  (auto simp: algebra_simps intro!: summable_add summable_in_conv_radius)

```

```

lemma conv_radius_mult_ge:
  fixes  $f \ g :: \text{nat} \Rightarrow ('a :: \{\text{banach}, \text{real\_normed\_div\_algebra}\})$ 
  shows  $\text{conv\_radius } (\lambda x. \sum_{i \leq x} f \ i * g \ (x - i)) \geq \min (\text{conv\_radius } f) (\text{conv\_radius } g)$ 
proof (rule conv_radius_geI_ex')
  fix  $r$  assume  $r > 0 \ \text{ereal } r < \min (\text{conv\_radius } f) (\text{conv\_radius } g)$ 
  from  $r$  have summable ( $\lambda n. (\sum_{i \leq n} (f \ i * \text{of\_real } r ^ i) * (g \ (n - i) * \text{of\_real } r ^ (n - i)))$ )
  by (intro summable_Cauchy_product abs_summable_in_conv_radius simp_all
  thus summable ( $\lambda n. (\sum_{i \leq n} f \ i * g \ (n - i)) * \text{of\_real } r ^ n$ ))
  by (simp add: algebra_simps of_real_def power_add [symmetric] scaleR_sum_right)
qed

```

```

lemma le_conv_radius_iff:
  fixes  $a :: \text{nat} \Rightarrow 'a :: \{\text{real\_normed\_div\_algebra}, \text{banach}\}$ 
  shows  $r \leq \text{conv\_radius } a \iff (\forall x. \text{norm } (x - \xi) < r \implies \text{summable } (\lambda i. a \ i * (x - \xi) ^ i))$ 
apply (intro iffI allI impI summable_in_conv_radius conv_radius_geI_ex)
apply (meson less_ereal.simps(1) not_le order_trans)

```

```

apply (rule_tac x=of_real ra in exI, simp)
apply (metis abs_of_nonneg add_diff_cancel_left' less_eq_real_def norm_of_real)
done

end

```

6.15 Uniform Limit and Uniform Convergence

```

theory Uniform_Limit
imports Connected_Summation_Tests Infinite_Sum
begin

```

6.15.1 Definition

```

definition uniformly_on :: 'a set  $\Rightarrow$  ('a  $\Rightarrow$  'b::metric_space)  $\Rightarrow$  ('a  $\Rightarrow$  'b) filter
  where uniformly_on S l = (INF e $\in$ {0 <..}. principal {f.  $\forall x \in S. \text{dist} (f x) (l x) < e$ })

```

abbreviation

```

uniform_limit S f l  $\equiv$  filterlim f (uniformly_on S l)

```

definition uniformly_convergent_on **where**

```

uniformly_convergent_on X f  $\longleftrightarrow$  ( $\exists l. \text{uniform\_limit } X f l \text{ sequentially}$ )

```

definition uniformly_Cauchy_on **where**

```

uniformly_Cauchy_on X f  $\longleftrightarrow$  ( $\forall e > 0. \exists M. \forall x \in X. \forall (m::nat) \geq M. \forall n \geq M. \text{dist} (f m x) (f n x) < e$ )

```

proposition uniform_limit_iff:

```

uniform_limit S f l F  $\longleftrightarrow$  ( $\forall e > 0. \forall_F n \text{ in } F. \forall x \in S. \text{dist} (f n x) (l x) < e$ )

```

```

unfolding filterlim_iff uniformly_on_def

```

```

by (subst eventually_INF_base)

```

```

  (fastforce

```

```

    simp: eventually_principal uniformly_on_def

```

```

    intro: be_xI[where x=min a b for a b]

```

```

    elim: eventually_mono)+

```

lemma uniform_limitD:

```

uniform_limit S f l F  $\Longrightarrow$   $e > 0 \Longrightarrow \forall_F n \text{ in } F. \forall x \in S. \text{dist} (f n x) (l x) < e$ 

```

```

by (simp add: uniform_limit_iff)

```

lemma uniform_limitI:

```

( $\bigwedge e. e > 0 \Longrightarrow \forall_F n \text{ in } F. \forall x \in S. \text{dist} (f n x) (l x) < e$ )  $\Longrightarrow$  uniform_limit S f l F

```

```

by (simp add: uniform_limit_iff)

```

lemma uniform_limit_sequentially_iff:

```

uniform_limit S f l sequentially  $\longleftrightarrow$  ( $\forall e > 0. \exists N. \forall n \geq N. \forall x \in S. \text{dist} (f n x) (l x) < e$ )

```

unfolding *uniform_limit_iff eventually_sequentially ..*

lemma *uniform_limit_at_iff:*

uniform_limit S f l (at x) \longleftrightarrow
 $(\forall e > 0. \exists d > 0. \forall z. 0 < \text{dist } z \ x \wedge \text{dist } z \ x < d \longrightarrow (\forall x \in S. \text{dist } (f \ z \ x) \ (l \ x) < e))$

unfolding *uniform_limit_iff eventually_at by simp*

lemma *uniform_limit_at_le_iff:*

uniform_limit S f l (at x) \longleftrightarrow
 $(\forall e > 0. \exists d > 0. \forall z. 0 < \text{dist } z \ x \wedge \text{dist } z \ x < d \longrightarrow (\forall x \in S. \text{dist } (f \ z \ x) \ (l \ x) \leq e))$

unfolding *uniform_limit_iff eventually_at*

by (*fastforce dest: spec[where $x = e / 2$ for e]*)

lemma *uniform_limit_compose:*

assumes *ul: uniform_limit X g l F*

and cont: *uniformly_continuous_on U f*

and g: $\forall_F n \text{ in } F. g \ n \ ' \ X \subseteq U$ **and l:** $l \ ' \ X \subseteq U$

shows *uniform_limit X ($\lambda a \ b. f \ (g \ a \ b)$) (f o l) F*

proof (*rule uniform_limitI*)

fix $\varepsilon :: \text{real}$

assume $0 < \varepsilon$

then obtain δ where $\delta > 0$ and $\delta: \bigwedge u \ v. \llbracket u \in U; v \in U; \text{dist } u \ v < \delta \rrbracket \implies \text{dist } (f \ u) \ (f \ v) < \varepsilon$

by (*metis cont uniformly_continuous_on_def*)

show $\forall_F n \text{ in } F. \forall x \in X. \text{dist } (f \ (g \ n \ x)) \ ((f \ o \ l) \ x) < \varepsilon$

using *uniform_limitD [OF ul $\langle \delta > 0 \rangle$] g unfolding o_def*

by *eventually_elim (use $\delta \ l$ in blast)*

qed

lemma *metric_uniform_limit_imp_uniform_limit:*

assumes *f: uniform_limit S f a F*

assumes *le: eventually ($\lambda x. \forall y \in S. \text{dist } (g \ x \ y) \ (b \ y) \leq \text{dist } (f \ x \ y) \ (a \ y)) F$*

shows *uniform_limit S g b F*

proof (*rule uniform_limitI*)

fix $e :: \text{real}$

assume $0 < e$

from *uniform_limitD[OF f this] le*

show $\forall_F x \text{ in } F. \forall y \in S. \text{dist } (g \ x \ y) \ (b \ y) < e$

by *eventually_elim force*

qed

6.15.2 Exchange limits

proposition *swap_uniform_limit:*

assumes *f: $\forall_F n \text{ in } F. (f \ n \longrightarrow g \ n)$ (at x within S)*

assumes *g: ($g \longrightarrow l$) F*

assumes *uc: uniform_limit S f h F*

```

  assumes  $\neg$ trivial_limit F
  shows  $(h \longrightarrow l)$  (at  $x$  within  $S$ )
proof (rule tendstoI)
  fix  $e :: \text{real}$ 
  define  $e'$  where  $e' = e/3$ 
  assume  $0 < e$ 
  then have  $0 < e'$  by (simp add:  $e'_\text{def}$ )
  from uniform_limitD[OF  $uc \langle 0 < e' \rangle$ ]
  have  $\forall_F n \text{ in } F. \forall x \in S. \text{dist } (h \ x) \ (f \ n \ x) < e'$ 
    by (simp add: dist_commute)
  moreover
  from f
  have  $\forall_F n \text{ in } F. \forall_F x \text{ in at } x \text{ within } S. \text{dist } (g \ n) \ (f \ n \ x) < e'$ 
    by eventually_elim (auto dest!: tendstoD[OF  $\_ \langle 0 < e' \rangle$ ] simp: dist_commute)
  moreover
  from tendstoD[OF  $g \langle 0 < e' \rangle$ ] have  $\forall_F x \text{ in } F. \text{dist } l \ (g \ x) < e'$ 
    by (simp add: dist_commute)
  ultimately
  have  $\forall_F \_ \text{ in } F. \forall_F x \text{ in at } x \text{ within } S. \text{dist } (h \ x) \ l < e$ 
  proof eventually_elim
    case (elim n)
    note fh = elim(1)
    note gl = elim(3)
    have  $\forall_F x \text{ in at } x \text{ within } S. x \in S$ 
      by (auto simp: eventually_at_filter)
    with elim(2)
    show ?case
  proof eventually_elim
    case (elim x)
    from fh[rule_format, OF  $\langle x \in S \rangle$ ] elim(1)
    have  $\text{dist } (h \ x) \ (g \ n) < e' + e'$ 
      by (rule dist_triangle_lt[OF add_strict_mono])
    from dist_triangle_lt[OF add_strict_mono, OF this gl]
    show ?case by (simp add:  $e'_\text{def}$ )
  qed
qed
thus  $\forall_F x \text{ in at } x \text{ within } S. \text{dist } (h \ x) \ l < e$ 
  using eventually_happens by (metis  $\langle \neg$ trivial_limit F  $\rangle$ )
qed

```

6.15.3 Uniform limit theorem

```

lemma tendsto_uniform_limitI:
  assumes uniform_limit S f l F
  assumes  $x \in S$ 
  shows  $((\lambda y. f \ y \ x) \longrightarrow l \ x) \ F$ 
  using assms
  by (auto intro!: tendstoI simp: eventually_mono dest!: uniform_limitD)

```

theorem *uniform_limit_theorem*:

assumes $c: \forall_F n \text{ in } F. \text{continuous_on } A (f n)$

assumes $ul: \text{uniform_limit } A f l F$

assumes $\neg \text{trivial_limit } F$

shows $\text{continuous_on } A l$

unfolding *continuous_on_def*

proof *safe*

fix x **assume** $x \in A$

then have $\forall_F n \text{ in } F. (f n \longrightarrow f n x) \text{ (at } x \text{ within } A) ((\lambda n. f n x) \longrightarrow l x) F$

using $c ul$

by (*auto simp: continuous_on_def eventually_mono tendsto_uniform_limitI*)

then show $(l \longrightarrow l x) \text{ (at } x \text{ within } A)$

by (*rule swap_uniform_limit*) *fact+*

qed

lemma *uniformly_Cauchy_onI*:

assumes $\bigwedge e. e > 0 \implies \exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f m x) (f n x) < e$

shows $\text{uniformly_Cauchy_on } X f$

using *assms* **unfolding** *uniformly_Cauchy_on_def* **by** *blast*

lemma *uniformly_Cauchy_onI'*:

assumes $\bigwedge e. e > 0 \implies \exists M. \forall x \in X. \forall m \geq M. \forall n > m. \text{dist } (f m x) (f n x) < e$

shows $\text{uniformly_Cauchy_on } X f$

proof (*rule uniformly_Cauchy_onI*)

fix $e :: \text{real}$ **assume** $e: e > 0$

from *assms*[*OF this*] **obtain** M

where $M: \bigwedge x m n. x \in X \implies m \geq M \implies n > m \implies \text{dist } (f m x) (f n x) < e$ **by** *fast*

{

fix $x m n$ **assume** $x: x \in X$ **and** $m: m \geq M$ **and** $n: n \geq M$

with M [*OF this*(1,2), *of n*] M [*OF this*(1,3), *of m*] e **have** $\text{dist } (f m x) (f n x) < e$

< e

by (*cases m n rule: linorder_cases*) (*simp_all add: dist_commute*)

}

thus $\exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f m x) (f n x) < e$ **by** *fast*

qed

lemma *uniformly_Cauchy_imp_Cauchy*:

$\text{uniformly_Cauchy_on } X f \implies x \in X \implies \text{Cauchy } (\lambda n. f n x)$

unfolding *Cauchy_def* *uniformly_Cauchy_on_def* **by** *fast*

lemma *uniform_limit_cong*:

fixes $f g :: 'a \Rightarrow 'b \Rightarrow ('c :: \text{metric_space})$ **and** $h i :: 'b \Rightarrow 'c$

assumes $\text{eventually } (\lambda y. \forall x \in X. f y x = g y x) F$

assumes $\bigwedge x. x \in X \implies h x = i x$

shows $\text{uniform_limit } X f h F \longleftrightarrow \text{uniform_limit } X g i F$

proof –

{

fix $f g :: 'a \Rightarrow 'b \Rightarrow 'c$ **and** $h i :: 'b \Rightarrow 'c$


```

  assume C: uniform_limit X f h F and A: eventually ( $\lambda y. \forall x \in X. f y x = g y x$ ) F
  and B:  $\bigwedge x. x \in X \implies h x = i x$ 
  {
    fix e :: real assume e > 0
    with C have eventually ( $\lambda y. \forall x \in X. dist (f y x) (h x) < e$ ) F
      unfolding uniform_limit_iff by blast
    with A have eventually ( $\lambda y. \forall x \in X. dist (g y x) (i x) < e$ ) F
      by eventually_elim (insert B, simp_all)
  }
  hence uniform_limit X g i F unfolding uniform_limit_iff by blast
} note A = this
show ?thesis by (rule iffI) (erule A; insert assms; simp add: eq_commute)+
qed

```

```

lemma uniform_limit_cong':
  fixes f g :: 'a  $\Rightarrow$  'b  $\Rightarrow$  ('c :: metric_space) and h i :: 'b  $\Rightarrow$  'c
  assumes  $\bigwedge y x. x \in X \implies f y x = g y x$ 
  assumes  $\bigwedge x. x \in X \implies h x = i x$ 
  shows uniform_limit X f h F  $\longleftrightarrow$  uniform_limit X g i F
  using assms by (intro uniform_limit_cong always_eventually) blast+

```

```

lemma uniformly_convergent_cong:
  assumes eventually ( $\lambda x. \forall y \in A. f x y = g x y$ ) sequentially A = B
  shows uniformly_convergent_on A f  $\longleftrightarrow$  uniformly_convergent_on B g
  unfolding uniformly_convergent_on_def assms(2) [symmetric]
  by (intro iff_exI uniform_limit_cong eventually_mono [OF assms(1)]) auto

```

```

lemma uniformly_convergent_on_compose:
  assumes uniformly_convergent_on A f
  assumes filterlim g sequentially sequentially
  shows uniformly_convergent_on A ( $\lambda n. f (g n)$ )
proof -
  from assms(1) obtain f' where uniform_limit A f f' sequentially
  by (auto simp: uniformly_convergent_on_def)
  hence uniform_limit A ( $\lambda n. f (g n)$ ) f' sequentially
  by (rule filterlim_compose) fact
  thus ?thesis
  by (auto simp: uniformly_convergent_on_def)
qed

```

```

lemma uniformly_convergent_uniform_limit_iff:
  uniformly_convergent_on X f  $\longleftrightarrow$  uniform_limit X f ( $\lambda x. \lim (\lambda n. f n x)$ )
  sequentially
proof
  assume uniformly_convergent_on X f
  then obtain l where l: uniform_limit X f l sequentially
  unfolding uniformly_convergent_on_def by blast
  from l have uniform_limit X f ( $\lambda x. \lim (\lambda n. f n x)$ ) sequentially  $\longleftrightarrow$ 

```

```

      uniform_limit X f l sequentially
    by (intro uniform_limit_cong' limI tendsto_uniform_limitI[of f X l]) simp_all
    also note l
    finally show uniform_limit X f (λx. lim (λn. f n x)) sequentially .
qed (auto simp: uniformly_convergent_on_def)

```

```

lemma uniformly_convergentI: uniform_limit X f l sequentially  $\implies$  uniformly_convergent_on
X f
  unfolding uniformly_convergent_on_def by blast

```

```

lemma uniformly_convergent_on_empty [iff]: uniformly_convergent_on {} f
  by (simp add: uniformly_convergent_on_def uniform_limit_sequentially_iff)

```

```

lemma uniformly_convergent_on_const [simp,intro]:
  uniformly_convergent_on A (λ_. c)
  by (auto simp: uniformly_convergent_on_def uniform_limit_iff intro!: exI[of _
c])

```

Cauchy-type criteria for uniform convergence.

```

lemma Cauchy_uniformly_convergent:
  fixes f :: nat  $\Rightarrow$  'a  $\Rightarrow$  'b :: complete_space
  assumes uniformly_Cauchy_on X f
  shows uniformly_convergent_on X f
unfolding uniformly_convergent_uniform_limit_iff uniform_limit_iff
proof safe
  let ?f = λx. lim (λn. f n x)
  fix e :: real assume e: e > 0
  hence e/2 > 0 by simp
  with assms obtain N where N:  $\bigwedge x m n. x \in X \implies m \geq N \implies n \geq N \implies$ 
dist (f m x) (f n x) < e/2
  unfolding uniformly_Cauchy_on_def by fast
  show eventually (λn.  $\forall x \in X. \text{dist} (f n x) (?f x) < e$ ) sequentially
  using eventually_ge_at_top[of N]
proof eventually_elim
  fix n assume n: n  $\geq$  N
  show  $\forall x \in X. \text{dist} (f n x) (?f x) < e$ 
proof
  fix x assume x: x  $\in$  X
  with assms have (λn. f n x)  $\longrightarrow$  ?f x
  by (auto dest!: Cauchy_convergent_uniformly_Cauchy_imp_Cauchy simp:
convergent_LIMSEQ_iff)
  with  $\langle e/2 > 0 \rangle$  have eventually (λm. m  $\geq$  N  $\wedge$  dist (f m x) (?f x) < e/2)
sequentially
  by (intro tendstoD eventually_conj eventually_ge_at_top)
  then obtain m where m: m  $\geq$  N dist (f m x) (?f x) < e/2
  unfolding eventually_at_top_linorder by blast
  have dist (f n x) (?f x)  $\leq$  dist (f n x) (f m x) + dist (f m x) (?f x)
  by (rule dist_triangle)
  also from x n have ... < e/2 + e/2 by (intro add_strict_mono N m)

```

```

    finally show  $\text{dist } (f \ n \ x) \ (?f \ x) < e$  by simp
  qed
qed

```

```

lemma uniformly_convergent_Cauchy:
  assumes uniformly_convergent_on X f
  shows uniformly_Cauchy_on X f
proof (rule uniformly_Cauchy_onI)
  fix  $e::\text{real}$  assume  $e > 0$ 
  then have  $0 < e / 2$  by simp
  with assms[unfolded uniformly_convergent_on_def uniformly_limit_sequentially_iff]
  obtain  $l \ N$  where  $l:x \in X \implies n \geq N \implies \text{dist } (f \ n \ x) \ (l \ x) < e / 2$  for  $n$ 
    by metis
  from  $l \ l$  have  $x \in X \implies n \geq N \implies m \geq N \implies \text{dist } (f \ n \ x) \ (f \ m \ x) < e$  for  $n$ 
     $m \ x$ 
    by (rule dist_triangle_half_l)
  then show  $\exists M. \forall x \in X. \forall m \geq M. \forall n \geq M. \text{dist } (f \ m \ x) \ (f \ n \ x) < e$  by blast
qed

```

```

lemma uniformly_convergent_eq_Cauchy:
  uniformly_convergent_on X f = uniformly_Cauchy_on X f for  $f::\text{nat} \Rightarrow 'b \Rightarrow$ 
   $'a::\text{complete\_space}$ 
  using Cauchy_uniformly_convergent uniformly_convergent_Cauchy by blast

```

```

lemma uniformly_convergent_eq_cauchy:
  fixes  $s::\text{nat} \Rightarrow 'b \Rightarrow 'a::\text{complete\_space}$ 
  shows
     $(\exists l. \forall e > 0. \exists N. \forall n \ x. N \leq n \wedge P \ x \longrightarrow \text{dist}(s \ n \ x)(l \ x) < e) \longleftrightarrow$ 
     $(\forall e > 0. \exists N. \forall m \ n \ x. N \leq m \wedge N \leq n \wedge P \ x \longrightarrow \text{dist } (s \ m \ x) \ (s \ n \ x) < e)$ 
proof -
  have *:  $(\forall n \geq N. \forall x. Q \ x \longrightarrow R \ n \ x) \longleftrightarrow (\forall n \ x. N \leq n \wedge Q \ x \longrightarrow R \ n \ x)$ 
     $(\forall x. Q \ x \longrightarrow (\forall m \geq N. \forall n \geq N. S \ n \ m \ x)) \longleftrightarrow (\forall m \ n \ x. N \leq m \wedge N \leq n \wedge$ 
     $Q \ x \longrightarrow S \ n \ m \ x)$ 
  for  $N::\text{nat}$  and  $Q::'b \Rightarrow \text{bool}$  and  $R \ S$ 
  by blast+
  show ?thesis
    using uniformly_convergent_eq_Cauchy[of Collect P s]
    unfolding uniformly_convergent_on_def uniformly_Cauchy_on_def uniformly_limit_sequentially_iff
    by (simp add: *)
qed

```

```

lemma uniformly_cauchy_imp_uniformly_convergent:
  fixes  $s :: \text{nat} \Rightarrow 'a \Rightarrow 'b::\text{complete\_space}$ 
  assumes  $\forall e > 0. \exists N. \forall m \ (n::\text{nat}) \ x. N \leq m \wedge N \leq n \wedge P \ x \longrightarrow \text{dist}(s \ m \ x)(s$ 
   $n \ x) < e$ 
  and  $\forall x. P \ x \longrightarrow (\forall e > 0. \exists N. \forall n. N \leq n \longrightarrow \text{dist}(s \ n \ x)(l \ x) < e)$ 
  shows  $\forall e > 0. \exists N. \forall n \ x. N \leq n \wedge P \ x \longrightarrow \text{dist}(s \ n \ x)(l \ x) < e$ 
proof -

```

obtain l' **where** $l:\forall \epsilon>0. \exists N. \forall n x. N \leq n \wedge P x \longrightarrow \text{dist } (s \ n \ x) \ (l' \ x) < \epsilon$
using $\text{assms}(1)$ **unfolding** $\text{uniformly_convergent_eq_cauchy}[\text{symmetric}]$ **by**
auto
moreover
{
 fix x
 assume $P \ x$
 then have $l \ x = l' \ x$
 using $\text{tendsto_unique}[\text{OF trivial_limit_sequentially, of } \lambda n. s \ n \ x \ l \ x \ l' \ x]$
 using l **and** $\text{assms}(2)$ **unfolding** lim_sequentially **by** blast
}
ultimately show $?thesis$ **by** *auto*
qed

lemma $\text{uniformly_convergent_on_sum_E}$:

fixes $\epsilon::\text{real}$ **and** $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b::\{\text{complete_space,real_normed_vector}\}$
assumes $\text{uconv}: \text{uniformly_convergent_on } K \ (\lambda n \ z. \sum_{k<n}. f \ k \ z)$ **and** $\epsilon>0$
obtains N **where** $\bigwedge m \ n \ z. \llbracket N \leq m; m \leq n; z \in K \rrbracket \Longrightarrow \text{norm}(\sum_{k=m..<n}. f \ k \ z) < \epsilon$
proof –
obtain N **where** $N: \bigwedge m \ n \ z. \llbracket N \leq m; N \leq n; z \in K \rrbracket \Longrightarrow \text{dist } (\sum_{k<m}. f \ k \ z) \ (\sum_{k<n}. f \ k \ z) < \epsilon$
using $\text{uconv } \langle \epsilon > 0 \rangle$ **unfolding** $\text{uniformly_Cauchy_on_def}$ $\text{uniformly_convergent_eq_Cauchy}$
by meson
show $thesis$
proof
 fix $m \ n \ z$
 assume $N \leq m \ m \leq n \ z \in K$
 moreover have $(\sum_{k=m..<n}. f \ k \ z) = (\sum_{k<n}. f \ k \ z) - (\sum_{k<m}. f \ k \ z)$
 by $(\text{metis } \text{atLeast0LessThan } \text{le0_sum_diff_nat_ivl } \langle m \leq n \rangle)$
 ultimately show $\text{norm}(\sum_{k=m..<n}. f \ k \ z) < \epsilon$
 using N **by** $(\text{simp } \text{add: } \text{dist_norm})$
qed
qed

lemma $\text{uniformly_convergent_on_sum_iff}$:

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b::\{\text{complete_space,real_normed_vector}\}$
shows $\text{uniformly_convergent_on } K \ (\lambda n \ z. \sum_{k<n}. f \ k \ z)$
 $\longleftrightarrow (\forall \epsilon>0. \exists N. \forall m \ n \ z. N \leq m \longrightarrow m \leq n \longrightarrow z \in K \longrightarrow \text{norm}(\sum_{k=m..<n}. f \ k \ z) < \epsilon)$ **(is ?lhs=?rhs)**
proof
 assume $R: ?rhs$
 show $?lhs$
 unfolding $\text{uniformly_Cauchy_on_def}$ $\text{uniformly_convergent_eq_Cauchy}$
 proof $(\text{intro } \text{strip})$
 fix $\epsilon::\text{real}$
 assume $\epsilon>0$
 then obtain N **where** $\bigwedge m \ n \ z. \llbracket N \leq m; m \leq n; z \in K \rrbracket \Longrightarrow \text{norm}(\sum_{k=m..<n}. f \ k \ z) < \epsilon$

```

    using R by blast
  then have  $\forall x \in K. \forall m \geq N. \forall n \geq m. \text{norm} ((\sum k < m. f k x) - (\sum k < n. f k x))$ 
  <  $\varepsilon$ 
    by (metis atLeast0LessThan le0 sum_diff_nat ivl norm_minus_commute)
  then have  $\forall x \in K. \forall m \geq N. \forall n \geq N. \text{norm} ((\sum k < m. f k x) - (\sum k < n. f k x))$ 
  <  $\varepsilon$ 
    by (metis linorder_le_cases norm_minus_commute)
  then show  $\exists M. \forall x \in K. \forall m \geq M. \forall n \geq M. \text{dist} (\sum k < m. f k x) (\sum k < n. f k x)$ 
  <  $\varepsilon$ 
    by (metis dist_norm)
  qed
qed (metis uniformly_convergent_on_sum_E)

```

lemma *uniform_limit_suminf*:

```

  fixes f :: nat  $\Rightarrow$  'a :: {metric_space, comm_monoid_add}  $\Rightarrow$  'a
  assumes uniformly_convergent_on X ( $\lambda n x. \sum k < n. f k x$ )
  shows uniform_limit X ( $\lambda n x. \sum k < n. f k x$ ) ( $\lambda x. \sum k. f k x$ ) sequentially
proof -
  obtain S where S: uniform_limit X ( $\lambda n x. \sum k < n. f k x$ ) S sequentially
  using assms uniformly_convergent_on_def by blast
  then have  $(\sum k. f k x) = S x$  if  $x \in X$  for x
  using that_sums_iff_sums_def by (blast intro: tendsto_uniform_limitI [OF S])
  with S show ?thesis
  by (simp cong: uniform_limit_cong')
qed

```

TODO: remove explicit formulations ($\exists l. \forall e > 0. \exists N. \forall n x. N \leq n \wedge ?P x \longrightarrow \text{dist} (?s n x) (l x) < e$) = ($\forall e > 0. \exists N. \forall m n x. N \leq m \wedge N \leq n \wedge ?P x \longrightarrow \text{dist} (?s m x) (?s n x) < e$)

$\llbracket \forall e > 0. \exists N. \forall m n x. N \leq m \wedge N \leq n \wedge ?P x \longrightarrow \text{dist} (?s m x) (?s n x) < e; \forall x. ?P x \longrightarrow (\forall e > 0. \exists N. \forall n \geq N. \text{dist} (?s n x) (?l x) < e) \rrbracket \Longrightarrow \forall e > 0. \exists N. \forall n x. N \leq n \wedge ?P x \longrightarrow \text{dist} (?s n x) (?l x) < e$!

lemma *uniformly_convergent_imp_convergent*:

```

  uniformly_convergent_on X f  $\Longrightarrow$  x  $\in$  X  $\Longrightarrow$  convergent ( $\lambda n. f n x$ )
  unfolding uniformly_convergent_on_def convergent_def
  by (auto dest: tendsto_uniform_limitI)

```

6.15.4 Comparison Test

lemma *uniformly_summable_comparison_test*:

```

  fixes f :: nat  $\Rightarrow$  'a  $\Rightarrow$  'b :: banach
  assumes uniformly_convergent_on A ( $\lambda N x. \sum n < N. g n x$ )
  assumes  $\bigwedge n x. x \in A \Longrightarrow \text{norm} (f n x) \leq g n x$ 
  shows uniformly_convergent_on A ( $\lambda N x. \sum n < N. f n x$ )
proof -
  have uniformly_Cauchy_on A ( $\lambda N x. \sum n < N. f n x$ )
  proof (rule uniformly_Cauchy_onI')
    fix e :: real assume e: e > 0

```

obtain M **where** $M: \bigwedge x m n. x \in A \implies m \geq M \implies n \geq M \implies \text{dist} (\sum k < m. g k x) (\sum k < n. g k x) < e$
using $\text{assms}(1)$ e **unfolding** $\text{uniformly_convergent_eq_Cauchy}$ $\text{uniformly_Cauchy_on_def}$
by metis
show $\exists M. \forall x \in A. \forall m \geq M. \forall n > m. \text{dist} (\sum k < m. f k x) (\sum k < n. f k x) < e$
proof (rule $\text{exI}[\text{of } M]$, safe)
fix $x m n$ **assume** $xmn: x \in A m \geq M m < n$
have $\text{nonneg}: g k x \geq 0$ **for** k
by (rule $\text{order.trans}[\text{OF } \text{assms}(2)]$) ($\text{use } xmn$ **in** auto)
have $\text{dist} (\sum k < m. f k x) (\sum k < n. f k x) = \text{norm} (\sum k \in \{..<n\} - \{..<m\}. f k x)$
using xmn **by** (subst sum_diff) ($\text{auto simp: dist_norm norm_minus_commute}$)
also have $\{..<n\} - \{..<m\} = \{m..<n\}$
by auto
also have $\text{norm} (\sum k \in \{m..<n\}. f k x) \leq (\sum k \in \{m..<n\}. \text{norm} (f k x))$
using norm_sum **by** blast
also have $\dots \leq (\sum k \in \{m..<n\}. g k x)$
by (intro sum_mono $\text{assms } xmn$)
also have $\dots = |\sum k \in \{m..<n\}. g k x|$
by (subst abs_of_nonneg) ($\text{auto simp: nonneg intro!: sum_nonneg}$)
also have $\{m..<n\} = \{..<n\} - \{..<m\}$
by auto
also have $|\sum k \in \dots g k x| = \text{dist} (\sum k < m. g k x) (\sum k < n. g k x)$
using xmn **by** (subst sum_diff) ($\text{auto simp: abs_minus_commute dist_norm}$)
also have $\dots < e$
by ($\text{rule } M$) ($\text{use } xmn$ **in** auto)
finally show $\text{dist} (\sum k < m. f k x) (\sum k < n. f k x) < e$.
qed
qed
thus $?thesis$
unfolding $\text{uniformly_convergent_eq_Cauchy}$.
qed

lemma $\text{uniform_limit_compose_uniformly_continuous_on}$:

fixes $f :: 'a :: \text{metric_space} \Rightarrow 'b :: \text{metric_space}$

assumes $\text{lim}: \text{uniform_limit } A g g' F$

assumes $\text{cont}: \text{uniformly_continuous_on } B f$

assumes $\text{ev}: \text{eventually } (\lambda x. \forall y \in A. g x y \in B) F$ **and** $\text{closed } B$

shows $\text{uniform_limit } A (\lambda x y. f (g x y)) (\lambda y. f (g' y)) F$

proof ($\text{cases } F = \text{bot}$)

case [simp]: False

show $?thesis$

unfolding uniform_limit_iff

proof safe

fix $e :: \text{real}$ **assume** $e: e > 0$

have $g' \text{ in } B: g' y \in B$ **if** $y \in A$ **for** y

proof (rule Lim_in_closed_set)

show $\text{eventually } (\lambda x. g x y \in B) F$

```

    using ev by eventually_elim (use that in auto)
  show (( $\lambda x. g\ x\ y$ )  $\longrightarrow$   $g'\ y$ ) F
    using lim that by (metis tendsto_uniform_limitI)
qed (use <closed B> in auto)

  obtain d where d:  $d > 0 \wedge x\ y. x \in B \implies y \in B \implies \text{dist } x\ y < d \implies \text{dist}$ 
  ( $f\ x$ ) ( $f\ y$ )  $< e$ 
    using e cont unfolding uniformly_continuous_on_def by metis
  from lim have eventually ( $\lambda x. \forall y \in A. \text{dist } (g\ x\ y) (g'\ y) < d$ ) F
    unfolding uniform_limit_iff using <d > 0> by meson
  thus eventually ( $\lambda x. \forall y \in A. \text{dist } (f\ (g\ x\ y)) (f\ (g'\ y)) < e$ ) F
    using assms(3)
  proof eventually_elim
    case (elim x)
    show  $\forall y \in A. \text{dist } (f\ (g\ x\ y)) (f\ (g'\ y)) < e$ 
      proof safe
        fix y assume y:  $y \in A$ 
        show  $\text{dist } (f\ (g\ x\ y)) (f\ (g'\ y)) < e$ 
          proof (rule d)
            show  $\text{dist } (g\ x\ y) (g'\ y) < d$ 
              using elim y by blast
            qed (use y elim g'_in_B in auto)
          qed
        qed
      qed
    qed
  qed (auto simp: filterlim_def)

```

```

lemma uniformly_convergent_on_compose_uniformly_continuous_on:
  fixes f :: 'a :: metric_space  $\Rightarrow$  'b :: metric_space
  assumes lim: uniformly_convergent_on A g
  assumes cont: uniformly_continuous_on B f
  assumes ev: eventually ( $\lambda x. \forall y \in A. g\ x\ y \in B$ ) sequentially and closed B
  shows uniformly_convergent_on A ( $\lambda x\ y. f\ (g\ x\ y)$ )
  proof -
    from lim obtain g' where g': uniform_limit A g g' sequentially
      by (auto simp: uniformly_convergent_on_def)
    thus ?thesis
      using uniform_limit_compose_uniformly_continuous_on[OF g' cont ev <closed B>]
      by (auto simp: uniformly_convergent_on_def)
    qed
  qed

```

6.15.5 Weierstrass M-Test

```

proposition Weierstrass_m_test_ev:
  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes eventually ( $\lambda n. \forall x \in A. \text{norm } (f\ n\ x) \leq M\ n$ ) sequentially
  assumes summable M
  shows uniform_limit A ( $\lambda n\ x. \sum_{i < n. f\ i\ x}$ ) ( $\lambda x. \text{suminf } (\lambda i. f\ i\ x)$ ) sequentially

```

```

proof (rule uniform_limitI)
  fix e :: real
  assume 0 < e
  from suminf_exist_split[OF ‹0 < e› ‹summable M›]
  have  $\forall_F k$  in sequentially. norm  $(\sum i. M (i + k)) < e$ 
    by (auto simp: eventually_sequentially)
  with eventually_all_ge_at_top[OF assms(1)]
    show  $\forall_F n$  in sequentially.  $\forall x \in A. \text{dist } (\sum i < n. f i x) (\sum i. f i x) < e$ 
  proof eventually_elim
    case (elim k)
    show ?case
    proof safe
      fix x assume x  $\in A$ 
      have  $\exists N. \forall n \geq N. \text{norm } (f n x) \leq M n$ 
        using assms(1) ‹x  $\in A$ › by (force simp: eventually_at_top_linorder)
      hence summable_norm_f: summable  $(\lambda n. \text{norm } (f n x))$ 
        by (rule summable_norm_comparison_test[OF _ ‹summable M›])
      have summable_f: summable  $(\lambda n. f n x)$ 
        using summable_norm_cancel[OF summable_norm_f] .
      have summable_norm_f_plus_k: summable  $(\lambda i. \text{norm } (f (i + k) x))$ 
        using summable_ignore_initial_segment[OF summable_norm_f]
        by auto
      have summable_M_plus_k: summable  $(\lambda i. M (i + k))$ 
        using summable_ignore_initial_segment[OF ‹summable M›]
        by auto

      have  $\text{dist } (\sum i < k. f i x) (\sum i. f i x) = \text{norm } ((\sum i. f i x) - (\sum i < k. f i x))$ 
        using dist_norm_dist_commute by (subst dist_commute)
      also have ... = norm  $(\sum i. f (i + k) x)$ 
        using suminf_minus_initial_segment[OF summable_f, where k=k] by
simp
      also have ...  $\leq (\sum i. \text{norm } (f (i + k) x))$ 
        using summable_norm[OF summable_norm_f_plus_k] .
      also have ...  $\leq (\sum i. M (i + k))$ 
        by (rule suminf_le[OF _ summable_norm_f_plus_k summable_M_plus_k])
        (insert elim(1) ‹x  $\in A$ ›, simp)
      finally show  $\text{dist } (\sum i < k. f i x) (\sum i. f i x) < e$ 
        using elim by auto
    qed
  qed
qed

```

Alternative version, formulated as in HOL Light

corollary series_comparison_uniform:

```

fixes f ::  $\_ \Rightarrow \text{nat} \Rightarrow \_ :: \text{banach}$ 
assumes g: summable g and le:  $\bigwedge n x. N \leq n \wedge x \in A \implies \text{norm}(f x n) \leq g n$ 
shows  $\exists l. \forall e. 0 < e \implies (\exists N. \forall n x. N \leq n \wedge x \in A \implies \text{dist}(\text{sum } (f x) \{..<n\}) (l x) < e)$ 
proof -

```



```

have 1:  $\forall_F n$  in sequentially.  $\forall x \in A. \text{norm } (f x n) \leq g n$ 
  using le eventually_sequentially by auto
show ?thesis
  apply (rule_tac  $x = (\lambda x. \sum i. f x i)$  in exI)
  apply (metis (no_types, lifting) eventually_sequentially uniform_limitD [OF
Weierstrass_m_test_ev [OF 1 g]])
  done
qed

```

```

corollary Weierstrass_m_test:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes  $\bigwedge n x. x \in A \implies \text{norm } (f n x) \leq M n$ 
  assumes summable M
  shows uniform_limit A  $(\lambda n x. \sum i < n. f i x)$   $(\lambda x. \text{suminf } (\lambda i. f i x))$  sequentially
  using assms by (intro Weierstrass_m_test_ev always_eventually) auto

```

```

corollary Weierstrass_m_test'_ev:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes eventually  $(\lambda n. \forall x \in A. \text{norm } (f n x) \leq M n)$  sequentially summable M
  shows uniformly_convergent_on A  $(\lambda n x. \sum i < n. f i x)$ 
  unfolding uniformly_convergent_on_def by (rule exI, rule Weierstrass_m_test_ev[OF
assms])

```

```

corollary Weierstrass_m_test':
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \text{banach}$ 
  assumes  $\bigwedge n x. x \in A \implies \text{norm } (f n x) \leq M n$  summable M
  shows uniformly_convergent_on A  $(\lambda n x. \sum i < n. f i x)$ 
  unfolding uniformly_convergent_on_def by (rule exI, rule Weierstrass_m_test[OF
assms])

```

```

lemma Weierstrass_m_test_general:
  fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \text{banach}$ 
  fixes  $M :: 'a \Rightarrow \text{real}$ 
  assumes norm_le:  $\bigwedge x y. x \in X \implies y \in Y \implies \text{norm } (f x y) \leq M x$ 
  assumes summable: M summable_on X
  shows uniform_limit Y  $(\lambda X y. \sum x \in X. f x y)$   $(\lambda y. \sum_{\infty} x \in X. f x y)$  (finite_subsets_at_top
X)
proof (rule uniform_limitI)
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  define S where  $S = (\lambda y. \sum_{\infty} x \in X. f x y)$ 
  have S:  $((\lambda x. f x y) \text{ has\_sum } S y) X$  if  $y: y \in Y$  for  $y$ 
  unfolding S_def
  proof (rule has_sum_infsun)
  have  $(\lambda x. \text{norm } (f x y))$  summable_on X
  by (rule abs_summable_on_comparison_test'[OF summable norm_le]) (use
y in auto)
  thus  $(\lambda x. f x y)$  summable_on X

```

```

    by (metis abs_summable_summable)
  qed

define T where T = (∑∞x∈X. M x)
have T: (M has_sum T) X
  unfolding T_def by (simp add: local_summable)
have M_summable: M summable_on X' if X' ⊆ X for X'
  using local_summable summable_on_subset_banach that by blast

have f_summable: (λx. f x y) summable_on X' if X' ⊆ X y ∈ Y for X' y
  using S summable_on_def summable_on_subset_banach that by blast
have eventually (λX'. dist (∑x∈X' M x) T < ε) (finite_subsets_at_top X)
  using T <ε > 0 unfolding T_def has_sum_def tendsto_iff by blast
moreover have eventually (λX'. finite X' ∧ X' ⊆ X) (finite_subsets_at_top
X)
  by (simp add: eventually_finite_subsets_at_top_weakI)
ultimately show ∀F X' in finite_subsets_at_top X. ∀ y ∈ Y. dist (∑x∈X' f x
y) (∑∞x∈X. f x y) < ε
  proof eventually_elim
    case X': (elim X')
    show ∀ y ∈ Y. dist (∑x∈X' f x y) (∑∞x∈X. f x y) < ε
    proof
      fix y assume y: y ∈ Y
      have 1: ((λx. f x y) has_sum (S y - (∑x∈X' f x y))) (X - X')
        using X' y by (intro has_sum_Diff S has_sum_finite[of X'] f_summable)
      auto
      have 2: (M has_sum (T - (∑x∈X' M x))) (X - X')
        using X' y by (intro has_sum_Diff T has_sum_finite[of X'] M_summable)
      auto
      have dist (∑x∈X' f x y) (∑∞x∈X. f x y) = norm (S y - (∑x∈X' f x y))
        by (simp add: dist_norm norm_minus_commute S_def)
      also have norm (S y - (∑x∈X' f x y)) ≤ (∑∞x∈X-X'. M x)
        using 2 y by (intro norm_infsum_le[OF 1 _ norm_le]) (auto simp: infsumI)
      also have ... = T - (∑x∈X' M x)
        using 2 by (auto simp: infsumI)
      also have ... < ε
        using X' y by (simp add: dist_norm)
      finally show dist (∑x∈X' f x y) (∑∞x∈X. f x y) < ε .
    qed
  qed
qed

```

6.15.6 Structural introduction rules

lemma *uniform_limit_eq_rhs*: *uniform_limit* X f l F \implies l = m \implies *uniform_limit* X f m F
 by *simp*

named_theorems *uniform_limit_intros* introduction rules for *uniform_limit*

```

setup <
  Global_Theory.add_thms_dynamic (binding <uniform_limit_eq_intros>,
    fn context =>
      Named_Theorems.get (Context.proof_of context) named_theorems <uniform_limit_intros>
        |> map_filter (try (fn thm => @ {thm uniform_limit_eq_rhs} OF [thm])))
  >

```

```

lemma (in bounded_linear) uniform_limit[uniform_limit_intros]:
  assumes uniform_limit X g l F
  shows uniform_limit X ( $\lambda a b. f (g a b)$ ) ( $\lambda a. f (l a)$ ) F
proof (rule uniform_limitI)
  fix e::real
  from pos_bounded obtain K
    where K:  $\bigwedge x y. \text{dist } (f x) (f y) \leq K * \text{dist } x y$  K > 0
    by (auto simp: ac_simps dist_norm diff[symmetric])
  assume 0 < e with <K > 0 have e / K > 0 by simp
  from uniform_limitD[OF assms this]
  show  $\forall_F n \text{ in } F. \forall x \in X. \text{dist } (f (g n x)) (f (l x)) < e$ 
    by eventually_elim (metis le_less_trans mult.commute pos_less_divide_eq K)
qed

```

```

lemma (in bounded_linear) uniformly_convergent_on:
  assumes uniformly_convergent_on A g
  shows uniformly_convergent_on A ( $\lambda x y. f (g x y)$ )
proof -
  from assms obtain l where uniform_limit A g l sequentially
  unfolding uniformly_convergent_on_def by blast
  hence uniform_limit A ( $\lambda x y. f (g x y)$ ) ( $\lambda x. f (l x)$ ) sequentially
  by (rule uniform_limit)
  thus ?thesis unfolding uniformly_convergent_on_def by blast
qed

```

```

lemmas bounded_linear_uniform_limit_intros[uniform_limit_intros] =
  bounded_linear.uniform_limit[OF bounded_linear_Im]
  bounded_linear.uniform_limit[OF bounded_linear_Re]
  bounded_linear.uniform_limit[OF bounded_linear_cnj]
  bounded_linear.uniform_limit[OF bounded_linear_fst]
  bounded_linear.uniform_limit[OF bounded_linear_snd]
  bounded_linear.uniform_limit[OF bounded_linear_zero]
  bounded_linear.uniform_limit[OF bounded_linear_of_real]
  bounded_linear.uniform_limit[OF bounded_linear_inner_left]
  bounded_linear.uniform_limit[OF bounded_linear_inner_right]
  bounded_linear.uniform_limit[OF bounded_linear_divide]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_right]
  bounded_linear.uniform_limit[OF bounded_linear_mult_left]
  bounded_linear.uniform_limit[OF bounded_linear_mult_right]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_left]

```

lemmas *uniform_limit_uminus*[*uniform_limit_intros*] =
bounded_linear.uniform_limit[*OF bounded_linear_minus*[*OF bounded_linear_ident*]]

lemma *uniform_limit_const*[*uniform_limit_intros*]: *uniform_limit* S $(\lambda x. c)$ c f
by (*auto intro!*: *uniform_limitI*)

lemma *uniform_limit_add*[*uniform_limit_intros*]:
fixes $f g::'a \Rightarrow 'b \Rightarrow 'c::\text{real_normed_vector}$
assumes *uniform_limit* X f l F
assumes *uniform_limit* X g m F
shows *uniform_limit* X $(\lambda a b. f a b + g a b)$ $(\lambda a. l a + m a)$ F
proof (*rule uniform_limitI*)
fix $e::\text{real}$
assume $0 < e$
hence $0 < e / 2$ **by** *simp*
from
uniform_limitD[*OF assms*(1) *this*]
uniform_limitD[*OF assms*(2) *this*]
show $\forall_F n$ *in* $F. \forall x \in X. \text{dist} (f n x + g n x) (l x + m x) < e$
by *eventually_elim* (*simp add: dist_triangle_add_half*)
qed

lemma *uniform_limit_minus*[*uniform_limit_intros*]:
fixes $f g::'a \Rightarrow 'b \Rightarrow 'c::\text{real_normed_vector}$
assumes *uniform_limit* X f l F
assumes *uniform_limit* X g m F
shows *uniform_limit* X $(\lambda a b. f a b - g a b)$ $(\lambda a. l a - m a)$ F
unfolding *diff_conv_add_uminus*
by (*rule uniform_limit_intros assms*)+

lemma *uniform_limit_norm*[*uniform_limit_intros*]:
assumes *uniform_limit* S g l f
shows *uniform_limit* S $(\lambda x y. \text{norm} (g x y))$ $(\lambda x. \text{norm} (l x))$ f
using *assms*
apply (*rule metric_uniform_limit_imp_uniform_limit*)
apply (*rule eventuallyI*)
by (*metis dist_norm_norm_triangle_ineq3 real_norm_def*)

lemma (*in bounded_bilinear*) *bounded_uniform_limit*[*uniform_limit_intros*]:
assumes *uniform_limit* X f l F
assumes *uniform_limit* X g m F
assumes *bounded* $(m \text{ ' } X)$
assumes *bounded* $(l \text{ ' } X)$
shows *uniform_limit* X $(\lambda a b. \text{prod} (f a b) (g a b))$ $(\lambda a. \text{prod} (l a) (m a))$ F
proof (*rule uniform_limitI*)
fix $e::\text{real}$
from *pos_bounded* **obtain** K **where** K :
 $0 < K \wedge a b. \text{norm} (\text{prod} a b) \leq \text{norm} a * \text{norm} b * K$
by *auto*

hence $\text{sqrt } (K * 4) > 0$ by *simp*

from *assms* obtain Km Kl

where Km : $Km > 0 \wedge x. x \in X \implies \text{norm } (m x) \leq Km$

and Kl : $Kl > 0 \wedge x. x \in X \implies \text{norm } (l x) \leq Kl$

by (*auto simp: bounded_pos*)

hence $K * Km * 4 > 0$ $K * Kl * 4 > 0$

using $\langle K > 0 \rangle$

by *simp_all*

assume $0 < e$

hence $\text{sqrt } e > 0$ by *simp*

from *uniform_limitD*[*OF* *assms*(1) *divide_pos_pos*[*OF* *this* $\langle \text{sqrt } (K * 4) > 0 \rangle$]]

uniform_limitD[*OF* *assms*(2) *divide_pos_pos*[*OF* *this* $\langle \text{sqrt } (K * 4) > 0 \rangle$]]

uniform_limitD[*OF* *assms*(1) *divide_pos_pos*[*OF* $\langle e > 0 \rangle$ $\langle K * Km * 4 > 0 \rangle$]]

uniform_limitD[*OF* *assms*(2) *divide_pos_pos*[*OF* $\langle e > 0 \rangle$ $\langle K * Kl * 4 > 0 \rangle$]]

show $\forall_F n$ in F . $\forall x \in X$. $\text{dist } (\text{prod } (f n x) (g n x)) (\text{prod } (l x) (m x)) < e$

proof *eventually_elim*

case (*elim* n)

show ?*case*

proof *safe*

fix x assume $x \in X$

have $\text{dist } (\text{prod } (f n x) (g n x)) (\text{prod } (l x) (m x)) \leq$

$\text{norm } (\text{prod } (f n x - l x) (g n x - m x)) +$

$\text{norm } (\text{prod } (f n x - l x) (m x)) +$

$\text{norm } (\text{prod } (l x) (g n x - m x))$

by (*auto simp: dist_norm prod_diff_prod intro: order_trans norm_triangle_ineq add_mono*)

also note $K(2)$ [*of* $f n x - l x$ $g n x - m x$]

also from *elim*(1)[*THEN* *bspec*, *OF* $\langle _ \in X \rangle$, *unfolded* *dist_norm*]

have $\text{norm } (f n x - l x) \leq \text{sqrt } e / \text{sqrt } (K * 4)$

by *simp*

also from *elim*(2)[*THEN* *bspec*, *OF* $\langle _ \in X \rangle$, *unfolded* *dist_norm*]

have $\text{norm } (g n x - m x) \leq \text{sqrt } e / \text{sqrt } (K * 4)$

by *simp*

also have $\text{sqrt } e / \text{sqrt } (K * 4) * (\text{sqrt } e / \text{sqrt } (K * 4)) * K = e / 4$

using $\langle K > 0 \rangle$ $\langle e > 0 \rangle$ by *auto*

also note $K(2)$ [*of* $f n x - l x$ $m x$]

also note $K(2)$ [*of* $l x$ $g n x - m x$]

also from *elim*(3)[*THEN* *bspec*, *OF* $\langle _ \in X \rangle$, *unfolded* *dist_norm*]

have $\text{norm } (f n x - l x) \leq e / (K * Km * 4)$

by *simp*

also from *elim*(4)[*THEN* *bspec*, *OF* $\langle _ \in X \rangle$, *unfolded* *dist_norm*]

have $\text{norm } (g n x - m x) \leq e / (K * Kl * 4)$

by *simp*

also note $Kl(2)$ [*OF* $\langle _ \in X \rangle$]

also note $Km(2)$ [*OF* $\langle _ \in X \rangle$]

also have $e / (K * Km * 4) * Km * K = e / 4$

using $\langle K > 0 \rangle$ $\langle Km > 0 \rangle$ by *simp*

1810

```

also have  $Kl * (e / (K * Kl * 4)) * K = e / 4$ 
  using  $\langle K > 0 \rangle \langle Kl > 0 \rangle$  by simp
also have  $e / 4 + e / 4 + e / 4 < e$  using  $\langle e > 0 \rangle$  by simp
finally show  $dist (prod (f n x) (g n x)) (prod (l x) (m x)) < e$ 
  using  $\langle K > 0 \rangle \langle Kl > 0 \rangle \langle Km > 0 \rangle \langle e > 0 \rangle$ 
  by (simp add: algebra_simps mult_right_mono divide_right_mono)
qed
qed
qed

lemmas bounded_bilinear_bounded_uniform_limit_intros[uniform_limit_intros]
=
  bounded_bilinear.bounded_uniform_limit[OF Inner_Product.bounded_bilinear_inner]
  bounded_bilinear.bounded_uniform_limit[OF Real_Vector_Spaces.bounded_bilinear_mult]
  bounded_bilinear.bounded_uniform_limit[OF Real_Vector_Spaces.bounded_bilinear_scaleR]

lemma uniform_lim_mult:
fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: real\_normed\_algebra$ 
assumes  $f: uniform\_limit\ S\ f\ l\ F$ 
  and  $g: uniform\_limit\ S\ g\ m\ F$ 
  and  $l: bounded\ (l\ 'S)$ 
  and  $m: bounded\ (m\ 'S)$ 
shows  $uniform\_limit\ S\ (\lambda a\ b. f\ a\ b * g\ a\ b)\ (\lambda a. l\ a * m\ a)\ F$ 
by (intro bounded_bilinear_bounded_uniform_limit_intros assms)

lemma uniform_lim_inverse:
fixes  $f :: 'a \Rightarrow 'b \Rightarrow 'c :: real\_normed\_field$ 
assumes  $f: uniform\_limit\ S\ f\ l\ F$ 
  and  $b: \bigwedge x. x \in S \implies b \leq norm(l\ x)$ 
  and  $b > 0$ 
shows  $uniform\_limit\ S\ (\lambda x\ y. inverse\ (f\ x\ y))\ (inverse\ o\ l)\ F$ 
proof (rule uniform_limitI)
fix  $e :: real$ 
assume  $e > 0$ 
have  $lte: dist\ (inverse\ (f\ x\ y))\ ((inverse\ o\ l)\ y) < e$ 
  if  $b/2 \leq norm\ (f\ x\ y)\ norm\ (f\ x\ y - l\ y) < e * b^2 / 2\ y \in S$ 
  for  $x\ y$ 
proof -
  have [simp]:  $l\ y \neq 0\ f\ x\ y \neq 0$ 
  using  $\langle b > 0 \rangle$  that  $b$  [OF  $\langle y \in S \rangle$ ] by fastforce+
  have  $norm\ (l\ y - f\ x\ y) < e * b^2 / 2$ 
  by (metis norm_minus_commute that(2))
  also have  $\dots \leq e * (norm\ (f\ x\ y) * norm\ (l\ y))$ 
  using  $\langle e > 0 \rangle$  that  $b$  [OF  $\langle y \in S \rangle$ ] apply (simp add: power2_eq_square)
  by (metis  $\langle b > 0 \rangle$  less_eq_real_def mult.left_commute mult_mono)
  finally show ?thesis
  by (auto simp: dist_norm field_split_simps norm_mult norm_divide)
qed
have  $\forall_F\ n\ in\ F. \forall x \in S. dist\ (f\ n\ x)\ (l\ x) < b/2$ 

```

```

  using uniform_limitD [OF f, of b/2] by (simp add: ‹b > 0›)
  then have  $\forall_F x \text{ in } F. \forall y \in S. b/2 \leq \text{norm } (f x y)$ 
  apply (rule eventually_mono)
  using b apply (simp only: dist_norm)
  by (metis (no_types, opaque_lifting) diff_zero dist_commute dist_norm norm_triangle_half_l
not_less)
  then have  $\forall_F x \text{ in } F. \forall y \in S. b/2 \leq \text{norm } (f x y) \wedge \text{norm } (f x y - l y) < e * b^2 / 2$ 
  apply (simp only: ball_conj_distrib dist_norm [symmetric])
  apply (rule eventually_conj, assumption)
  apply (rule uniform_limitD [OF f, of e * b ^ 2 / 2])
  using ‹b > 0› ‹e > 0› by auto
  then show  $\forall_F x \text{ in } F. \forall y \in S. \text{dist } (\text{inverse } (f x y)) ((\text{inverse} \circ l) y) < e$ 
  using lte by (force intro: eventually_mono)
qed

```

lemma *uniform_lim_divide*:

```

  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  'c::real_normed_field
  assumes f: uniform_limit S f l F
  and g: uniform_limit S g m F
  and l: bounded (l ' S)
  and b:  $\bigwedge x. x \in S \implies b \leq \text{norm}(m x)$ 
  and b > 0
  shows uniform_limit S ( $\lambda a b. f a b / g a b$ ) ( $\lambda a. l a / m a$ ) F
proof -
  have m: bounded ((inverse  $\circ$  m) ' S)
  using b ‹b > 0›
  apply (simp add: bounded_iff)
  by (metis le_imp_inverse_le norm_inverse)
  have uniform_limit S ( $\lambda a b. f a b * \text{inverse } (g a b)$ )
    ( $\lambda a. l a * (\text{inverse} \circ m) a$ ) F
  by (rule uniform_lim_mult [OF f uniform_lim_inverse [OF g b ‹b > 0›] l m])
  then show ?thesis
  by (simp add: field_class.field_divide_inverse)
qed

```

lemma *uniform_limit_null_comparison*:

```

  assumes  $\forall_F x \text{ in } F. \forall a \in S. \text{norm } (f x a) \leq g x a$ 
  assumes uniform_limit S g ( $\lambda_. 0$ ) F
  shows uniform_limit S f ( $\lambda_. 0$ ) F
  using assms(2)
proof (rule metric_uniform_limit_imp_uniform_limit)
  show  $\forall_F x \text{ in } F. \forall y \in S. \text{dist } (f x y) 0 \leq \text{dist } (g x y) 0$ 
  using assms(1) by (rule eventually_mono) (force simp add: dist_norm)
qed

```

lemma *uniform_limit_on_Un*:

```

  uniform_limit I f g F  $\implies$  uniform_limit J f g F  $\implies$  uniform_limit (I  $\cup$  J) f g F

```

1812

by (auto intro!: uniform_limitI dest!: uniform_limitD elim: eventually_elim2)

lemma uniform_limit_on_empty [iff]:

uniform_limit {} f g F

by (auto intro!: uniform_limitI)

lemma uniform_limit_on_UNION:

assumes finite S

assumes $\bigwedge s. s \in S \implies \text{uniform_limit } (h\ s) f\ g\ F$

shows uniform_limit ($\bigcup (h\ 'S)$) f g F

using assms

by induct (auto intro: uniform_limit_on_empty uniform_limit_on_Un)

lemma uniform_limit_on_Union:

assumes finite I

assumes $\bigwedge J. J \in I \implies \text{uniform_limit } J\ f\ g\ F$

shows uniform_limit (Union I) f g F

by (metis SUP_identity_eq assms uniform_limit_on_UNION)

lemma uniform_limit_on_subset:

uniform_limit J f g F $\implies I \subseteq J \implies \text{uniform_limit } I\ f\ g\ F$

by (auto intro!: uniform_limitI dest!: uniform_limitD intro: eventually_mono)

lemma uniform_limit_bounded:

fixes f::'i \Rightarrow 'a::topological_space \Rightarrow 'b::metric_space

assumes l: uniform_limit S f l F

assumes bnd: $\forall_F i \text{ in } F. \text{bounded } (f\ i\ 'S)$

assumes F \neq bot

shows bounded (l 'S)

proof -

from l **have** $\forall_F n \text{ in } F. \forall x \in S. \text{dist } (l\ x) (f\ n\ x) < 1$

by (auto simp: uniform_limit_iff dist_commute dest!: spec[where x=1])

with bnd

have $\forall_F n \text{ in } F. \exists M. \forall x \in S. \text{dist undefined } (l\ x) \leq M + 1$

by eventually_elim

(auto intro!: order_trans[OF dist_triangle2 add_mono] intro: less_imp_le

simp: bounded_any_center[where a=undefined])

then show ?thesis **using** assms

by (auto simp: bounded_any_center[where a=undefined] dest!: eventually_happens)

qed

lemma uniformly_convergent_add:

uniformly_convergent_on A f \implies uniformly_convergent_on A g \implies

uniformly_convergent_on A ($\lambda k\ x. f\ k\ x + g\ k\ x :: 'a :: \{\text{real_normed_algebra}\}$)

unfolding uniformly_convergent_on_def **by** (blast dest: uniform_limit_add)

lemma uniformly_convergent_minus:

uniformly_convergent_on A f \implies uniformly_convergent_on A g \implies

uniformly_convergent_on A ($\lambda k\ x. f\ k\ x - g\ k\ x :: 'a :: \{\text{real_normed_algebra}\}$)

unfolding *uniformly_convergent_on_def* **by** (*blast dest: uniform_limit_minus*)

lemma *uniformly_convergent_mult*:

uniformly_convergent_on A $f \implies$

uniformly_convergent_on A $(\lambda k x. c * f k x :: 'a :: \{\text{real_normed_algebra}\})$

unfolding *uniformly_convergent_on_def*

by (*blast dest: bounded_linear_uniform_limit_intros(13)*)

6.15.7 Power series and uniform convergence

proposition *power_uniformly_convergent*:

fixes $a :: \text{nat} \Rightarrow 'a :: \{\text{real_normed_div_algebra, banach}\}$

assumes $r < \text{conv_radius } a$

shows *uniformly_convergent_on* $(\text{cball } \xi r)$ $(\lambda n x. \sum i < n. a i * (x - \xi) ^ i)$

proof (*cases* $0 \leq r$)

case *True*

then have $*$: *summable* $(\lambda n. \text{norm } (a n) * r ^ n)$

using *abs_summable_in_conv_radius* [*of of_real r a*] *assms*

by (*simp add: norm_mult norm_power*)

show *?thesis*

by (*simp add: Weierstrass_m_test'_ev* [*OF **] *norm_mult norm_power*
mult_left_mono power_mono dist_norm norm_minus_commute)

next

case *False* **then show** *?thesis* **by** (*simp add: not_le*)

qed

lemma *power_uniform_limit*:

fixes $a :: \text{nat} \Rightarrow 'a :: \{\text{real_normed_div_algebra, banach}\}$

assumes $r < \text{conv_radius } a$

shows *uniform_limit* $(\text{cball } \xi r)$ $(\lambda n x. \sum i < n. a i * (x - \xi) ^ i)$ $(\lambda x. \text{suminf } (\lambda i. a i * (x - \xi) ^ i))$ *sequentially*

using *power_uniformly_convergent* [*OF assms*]

by (*simp add: Uniform_Limit.uniformly_convergent_uniform_limit_iff_Series.suminf_eq_lim*)

lemma *power_continuous_suminf*:

fixes $a :: \text{nat} \Rightarrow 'a :: \{\text{real_normed_div_algebra, banach}\}$

assumes $r < \text{conv_radius } a$

shows *continuous_on* $(\text{cball } \xi r)$ $(\lambda x. \text{suminf } (\lambda i. a i * (x - \xi) ^ i))$

apply (*rule uniform_limit_theorem* [*OF power_uniform_limit*])

apply (*rule eventuallyI continuous_intros assms*)**+**

apply (*simp add:*)

done

lemma *power_continuous_sums*:

fixes $a :: \text{nat} \Rightarrow 'a :: \{\text{real_normed_div_algebra, banach}\}$

assumes $r: r < \text{conv_radius } a$

and $sm: \bigwedge x. x \in \text{cball } \xi r \implies (\lambda n. a n * (x - \xi) ^ n)$ *sums* $(f x)$

shows *continuous_on* $(\text{cball } \xi r)$ f

apply (*rule continuous_on_cong* [*THEN iffD1, OF refl power_continuous_suminf*])

1814

```
[OF r]])  
using sm sums_unique by fastforce  
  
lemmas uniform_limit_subset_union = uniform_limit_on_subset[OF uniform_limit_on_Union]  
  
end
```

6.16 Bounded Linear Function

```
theory Bounded_Linear_Function  
imports  
  Topology_Euclidean_Space  
  Operator_Norm  
  Uniform_Limit  
  Function_Topology  
  
begin  
  
lemma onorm_componentwise:  
  assumes bounded_linear f  
  shows onorm f  $\leq$  ( $\sum i \in \text{Basis}. \text{norm } (f i)$ )  
proof -  
  {  
    fix i::'a  
    assume i  $\in$  Basis  
    hence onorm ( $\lambda x. (x \cdot i) *_R f i$ )  $\leq$  onorm ( $\lambda x. (x \cdot i)$ ) * norm (f i)  
      by (auto intro!: onorm_scaleR_left_lemma bounded_linear_inner_left)  
    also have ...  $\leq$  norm i * norm (f i)  
      by (rule mult_right_mono)  
      (auto simp: ac_simps Cauchy_Schwarz_ineq2 intro!: onorm_le)  
    finally have onorm ( $\lambda x. (x \cdot i) *_R f i$ )  $\leq$  norm (f i) using <i  $\in$  Basis>  
      by simp  
  } hence onorm ( $\lambda x. \sum i \in \text{Basis}. (x \cdot i) *_R f i$ )  $\leq$  ( $\sum i \in \text{Basis}. \text{norm } (f i)$ )  
    by (auto intro!: order_trans[OF onorm_sum_le] bounded_linear_scaleR_const  
      sum_mono bounded_linear_inner_left)  
  also have ( $\lambda x. \sum i \in \text{Basis}. (x \cdot i) *_R f i$ ) = ( $\lambda x. f (\sum i \in \text{Basis}. (x \cdot i) *_R i)$ )  
    by (simp add: linear_sum bounded_linear.linear assms linear_simps)  
  also have ... = f  
    by (simp add: euclidean_representation)  
  finally show ?thesis .  
qed  
  
lemmas onorm_componentwise_le = order_trans[OF onorm_componentwise]
```

6.16.1 Intro rules for bounded_linear

```
named_theorems bounded_linear_intros
```

```
lemma onorm_inner_left:
```

```

assumes bounded_linear r
shows  $\text{onorm } (\lambda x. r\ x \cdot f) \leq \text{onorm } r * \text{norm } f$ 
proof (rule onorm_bound)
  fix x
  have  $\text{norm } (r\ x \cdot f) \leq \text{norm } (r\ x) * \text{norm } f$ 
    by (simp add: Cauchy_Schwarz_ineq2)
  also have  $\dots \leq \text{onorm } r * \text{norm } x * \text{norm } f$ 
    by (intro mult_right_mono onorm assms norm_ge_zero)
  finally show  $\text{norm } (r\ x \cdot f) \leq \text{onorm } r * \text{norm } f * \text{norm } x$ 
    by (simp add: ac_simps)
qed (intro mult_nonneg_nonneg norm_ge_zero onorm_pos_le assms)

```

```

lemma onorm_inner_right:
assumes bounded_linear r
shows  $\text{onorm } (\lambda x. f \cdot r\ x) \leq \text{norm } f * \text{onorm } r$ 
apply (subst inner_commute)
apply (rule onorm_inner_left[OF assms, THEN order_trans])
apply simp
done

```

```

lemmas [bounded_linear_intros] =
  bounded_linear_zero
  bounded_linear_add
  bounded_linear_const_mult
  bounded_linear_mult_const
  bounded_linear_scaleR_const
  bounded_linear_const_scaleR
  bounded_linear_ident
  bounded_linear_sum
  bounded_linear_Pair
  bounded_linear_sub
  bounded_linear_fst_comp
  bounded_linear_snd_comp
  bounded_linear_inner_left_comp
  bounded_linear_inner_right_comp

```

6.16.2 declaration of derivative/continuous/tendsto introduction rules for bounded linear functions

```

attribute_setup bounded_linear =
  <Scan.succeed (Thm.declaration_attribute (fn thm =>
    fold (fn (r, s) => Named_Theorems.add_thm s (thm RS r))
      [
        (@{thm bounded_linear.has_derivative}, named_theorems <derivative_intros>),
        (@{thm bounded_linear.tendsto}, named_theorems <tendsto_intros>),
        (@{thm bounded_linear.continuous}, named_theorems <continuous_intros>),
        (@{thm bounded_linear.continuous_on}, named_theorems <continuous_intros>),
        (@{thm bounded_linear.uniformly_continuous_on}, named_theorems <continuous_intros>),
        (@{thm bounded_linear.compose}, named_theorems <bounded_linear_intros>)
      ]
  )

```

```
)))>
```

```
attribute_setup bounded_bilinear =
  <Scan.succeed (Thm.declaration_attribute (fn thm =>
    fold (fn (r, s) => Named_Theorems.add_thm s (thm RS r))
      [
        (@{thm bounded_bilinear.FDERIV}, named_theorems <derivative_intros>),
        (@{thm bounded_bilinear.tendsto}, named_theorems <tendsto_intros>),
        (@{thm bounded_bilinear.continuous}, named_theorems <continuous_intros>),
        (@{thm bounded_bilinear.continuous_on}, named_theorems <continuous_intros>),
        (@{thm bounded_linear_compose[OF bounded_bilinear.bounded_linear_left]},
          named_theorems <bounded_linear_intros>),
        (@{thm bounded_linear_compose[OF bounded_bilinear.bounded_linear_right]},
          named_theorems <bounded_linear_intros>),
        (@{thm bounded_linear.uniformly_continuous_on[OF bounded_bilinear.bounded_linear_left]},
          named_theorems <continuous_intros>),
        (@{thm bounded_linear.uniformly_continuous_on[OF bounded_bilinear.bounded_linear_right]},
          named_theorems <continuous_intros>)
      ]
    )))>
```

6.16.3 Type of bounded linear functions

```
typedef (overloaded) ('a, 'b) blinfun (( $\_ \Rightarrow_L \_$ ) [22, 21] 21) =
  {f::'a::real_normed_vector $\Rightarrow$ 'b::real_normed_vector. bounded_linear f}
morphisms blinfun_apply Blinfun
by (blast intro: bounded_linear_intros)
```

```
declare [[coercion
  blinfun_apply :: ('a::real_normed_vector  $\Rightarrow_L$  'b::real_normed_vector)  $\Rightarrow$  'a  $\Rightarrow$ 
  'b]]
```

```
lemma bounded_linear_blinfun_apply[bounded_linear_intros]:
  bounded_linear g  $\implies$  bounded_linear ( $\lambda x$ . blinfun_apply f (g x))
  by (metis blinfun_apply mem_Collect_eq bounded_linear_compose)
```

```
setup_lifting type_definition_blinfun
```

```
lemma blinfun_eqI: ( $\bigwedge i$ . blinfun_apply x i = blinfun_apply y i)  $\implies$  x = y
  by transfer auto
```

```
lemma bounded_linear_Blinfun_apply: bounded_linear f  $\implies$  blinfun_apply (Blinfun
f) = f
  by (auto simp: Blinfun_inverse)
```

6.16.4 Type class instantiations

```
instantiation blinfun :: (real_normed_vector, real_normed_vector) real_normed_vector
begin
```

```
lift_definition norm_blinfun :: 'a  $\Rightarrow_L$  'b  $\Rightarrow$  real is onorm .
```

lift_definition *minus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
is $\lambda f g x. f x - g x$
by (*rule bounded_linear_sub*)

definition *dist_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow real$
where $dist_blinfun\ a\ b = norm\ (a - b)$

definition [*code del*]:
(*uniformity* :: $(('a \Rightarrow_L 'b) \times ('a \Rightarrow_L 'b))\ filter) = (INF\ e \in \{0 < ..\}. principal\ \{(x, y). dist\ x\ y < e\})$)

definition *open_blinfun* :: $('a \Rightarrow_L 'b)\ set \Rightarrow bool$
where [*code del*]: $open_blinfun\ S = (\forall x \in S. \forall_F (x', y)\ in\ uniformity. x' = x \longrightarrow y \in S)$

lift_definition *uminus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$ **is** $\lambda f x. - f x$
by (*rule bounded_linear_minus*)

lift_definition *zero_blinfun* :: $'a \Rightarrow_L 'b$ **is** $\lambda x. 0$
by (*rule bounded_linear_zero*)

lift_definition *plus_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
is $\lambda f g x. f x + g x$
by (*metis bounded_linear_add*)

lift_definition *scaleR_blinfun* :: $real \Rightarrow 'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$ **is** $\lambda r f x. r *_{R} f x$
by (*metis bounded_linear_compose bounded_linear_scaleR_right*)

definition *sgn_blinfun* :: $'a \Rightarrow_L 'b \Rightarrow 'a \Rightarrow_L 'b$
where $sgn_blinfun\ x = scaleR\ (inverse\ (norm\ x))\ x$

instance

apply *standard*

unfolding *dist_blinfun_def open_blinfun_def sgn_blinfun_def uniformity_blinfun_def*

apply (*rule refl* | (*transfer, force simp: onorm_triangle onorm_scaleR onorm_eq_0 algebra_simps*))+

done

end

declare *uniformity_Abort*[**where** $'a = ('a :: real_normed_vector) \Rightarrow_L ('b :: real_normed_vector)$,
code]

lemma *norm_blinfun_eqI*:

assumes $n \leq norm\ (blinfun_apply\ f\ x) / norm\ x$

assumes $\bigwedge x. norm\ (blinfun_apply\ f\ x) \leq n * norm\ x$

assumes $0 \leq n$

shows $norm\ f = n$

1818

```
by (auto simp: norm_blinfun_def
    intro!: antisym onorm_bound assms order_trans[OF _ le_onorm]
    bounded_linear_intros)
```

```
lemma norm_blinfun: norm (blinfun_apply f x) ≤ norm f * norm x
  by transfer (rule onorm)
```

```
lemma norm_blinfun_bound: 0 ≤ b ⇒ (∧x. norm (blinfun_apply f x) ≤ b *
norm x) ⇒ norm f ≤ b
  by transfer (rule onorm_bound)
```

```
lemma bounded_bilinear_blinfun_apply[bounded_bilinear]: bounded_bilinear blin-
fun_apply
```

proof

```
fix f g::'a ⇒L 'b and a b::'a and r::real
show (f + g) a = f a + g a (r *R f) a = r *R f a
  by (transfer, simp)+
interpret bounded_linear f for f::'a ⇒L 'b
  by (auto intro!: bounded_linear_intros)
show f (a + b) = f a + f b f (r *R a) = r *R f a
  by (simp_all add: add scaleR)
show ∃ K. ∀ a b. norm (blinfun_apply a b) ≤ norm a * norm b * K
  by (auto intro!: exI[where x=1] norm_blinfun)
```

qed

```
interpretation blinfun: bounded_bilinear blinfun_apply
  by (rule bounded_bilinear_blinfun_apply)
```

```
lemmas bounded_linear_apply_blinfun[intro, simp] = blinfun.bounded_linear_left
```

```
declare blinfun.zero_left [simp] blinfun.zero_right [simp]
```

context *bounded_bilinear*

begin

named_theorems *bilinear_simps*

```
lemmas [bilinear_simps] =
```

```
  add_left
  add_right
  diff_left
  diff_right
  minus_left
  minus_right
  scaleR_left
  scaleR_right
  zero_left
  zero_right
```

```

    sum_left
    sum_right

end

instance blinfun :: (real_normed_vector, banach) banach
proof
  fix X::nat  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  assume Cauchy X
  {
    fix x::'a
    {
      fix x::'a
      assume norm x  $\leq$  1
      have Cauchy ( $\lambda n. X n x$ )
      proof (rule CauchyI)
        fix e::real
        assume 0 < e
        from CauchyD[OF  $\langle$ Cauchy X $\rangle$   $\langle$ 0 < e $\rangle$ ] obtain M
          where M:  $\bigwedge m n. m \geq M \implies n \geq M \implies \text{norm } (X m - X n) < e$ 
          by auto
        show  $\exists M. \forall m \geq M. \forall n \geq M. \text{norm } (X m x - X n x) < e$ 
        proof (safe intro!: exI[where x=M])
          fix m n
          assume le:  $M \leq m \leq n$ 
          have norm (X m x - X n x) = norm ((X m - X n) x)
            by (simp add: blinfun.bilinear_simps)
          also have  $\dots \leq \text{norm } (X m - X n) * \text{norm } x$ 
            by (rule norm_blinfun)
          also have  $\dots \leq \text{norm } (X m - X n) * 1$ 
            using  $\langle \text{norm } x \leq 1 \rangle$  norm_ge_zero by (rule mult_left_mono)
          also have  $\dots = \text{norm } (X m - X n)$  by simp
          also have  $\dots < e$  using le by fact
          finally show norm (X m x - X n x) < e .
        qed
      qed
    }
  }
  hence convergent ( $\lambda n. X n x$ )
    by (metis Cauchy_convergent_iff)
} note convergent_norm1 = this
define y where y = x /R norm x
have y: norm y  $\leq$  1 and xy: x = norm x *R y
  by (simp_all add: y_def inverse_eq_divide)
have convergent ( $\lambda n. \text{norm } x *R X n y$ )
  by (intro bounded_bilinear.convergent[OF bounded_bilinear_scaleR] convergent_norm1 y)
also have ( $\lambda n. \text{norm } x *R X n y$ ) = ( $\lambda n. X n x$ )
  by (subst xy) (simp add: blinfun.bilinear_simps)

```

```

    finally have convergent ( $\lambda n. X\ n\ x$ ) .
  }
  then obtain v where v:  $\bigwedge x. (\lambda n. X\ n\ x) \longrightarrow v\ x$ 
    unfolding convergent_def
    by metis

  have Cauchy ( $\lambda n. \text{norm}\ (X\ n)$ )
  proof (rule CauchyI)
    fix e::real
    assume e > 0
    from CauchyD[OF  $\langle \text{Cauchy}\ X \rangle \langle 0 < e \rangle$ ] obtain M
      where M:  $\bigwedge m\ n. m \geq M \implies n \geq M \implies \text{norm}\ (X\ m - X\ n) < e$ 
      by auto
    show  $\exists M. \forall m \geq M. \forall n \geq M. \text{norm}\ (\text{norm}\ (X\ m) - \text{norm}\ (X\ n)) < e$ 
    proof (safe intro!: exI[where x=M])
      fix m n assume mn:  $m \geq M\ n \geq M$ 
      have  $\text{norm}\ (\text{norm}\ (X\ m) - \text{norm}\ (X\ n)) \leq \text{norm}\ (X\ m - X\ n)$ 
        by (metis norm_triangle_ineq3 real_norm_def)
      also have  $\dots < e$  using mn by fact
      finally show  $\text{norm}\ (\text{norm}\ (X\ m) - \text{norm}\ (X\ n)) < e$  .
    qed
  qed
  then obtain K where K:  $(\lambda n. \text{norm}\ (X\ n)) \longrightarrow K$ 
    unfolding Cauchy_convergent_iff convergent_def
    by metis

  have bounded_linear v
  proof
    fix x y and r::real
    from tendsto_add[OF v[of x] v [of y]] v[of x + y, unfolded blinfun.bilinear_simps]
      tendsto_scaleR[OF tendsto_const[of r] v[of x]] v[of r *R x, unfolded blin-
fun.bilinear_simps]
    show  $v\ (x + y) = v\ x + v\ y\ v\ (r *_{\mathbb{R}} x) = r *_{\mathbb{R}} v\ x$ 
      by (metis (poly_guards_query) LIMSEQ_unique) +
    show  $\exists K. \forall x. \text{norm}\ (v\ x) \leq \text{norm}\ x * K$ 
    proof (safe intro!: exI[where x=K])
      fix x
      have  $\text{norm}\ (v\ x) \leq K * \text{norm}\ x$ 
      by (rule tendsto_le[OF tendsto_mult[OF K tendsto_const] tendsto_norm[OF
v]])
        (auto simp: norm_blinfun)
      thus  $\text{norm}\ (v\ x) \leq \text{norm}\ x * K$ 
        by (simp add: ac_simps)
    qed
  qed
  hence Bv:  $\bigwedge x. (\lambda n. X\ n\ x) \longrightarrow \text{Blinfun}\ v\ x$ 
    by (auto simp: bounded_linear_Blinfun_apply v)

  have X  $\longrightarrow \text{Blinfun}\ v$ 

```



```

proof (rule LIMSEQ_I)
  fix  $r::\text{real}$  assume  $r > 0$ 
  define  $r'$  where  $r' = r / 2$ 
  have  $0 < r' \wedge r' < r$  using  $\langle r > 0 \rangle$  by (simp_all add: r'_def)
  from CauchyD[OF  $\langle \text{Cauchy } X \rangle \langle r' > 0 \rangle$ ]
  obtain  $M$  where  $M: \bigwedge m n. m \geq M \implies n \geq M \implies \text{norm } (X m - X n) < r'$ 
    by metis
  show  $\exists \text{no. } \forall n \geq \text{no. } \text{norm } (X n - \text{Blinfun } v) < r$ 
  proof (safe intro!: exI[where  $x=M$ ])
    fix  $n$  assume  $n: M \leq n$ 
    have  $\text{norm } (X n - \text{Blinfun } v) \leq r'$ 
    proof (rule norm_blinfun_bound)
      fix  $x$ 
      have eventually  $(\lambda m. m \geq M)$  sequentially
        by (metis eventually_ge_at_top)
      hence ev_le: eventually  $(\lambda m. \text{norm } (X n x - X m x) \leq r' * \text{norm } x)$ 
        sequentially
    proof eventually_elim
      case (elim  $m$ )
      have  $\text{norm } (X n x - X m x) = \text{norm } ((X n - X m) x)$ 
        by (simp add: blinfun.bilinear_simps)
      also have  $\dots \leq \text{norm } ((X n - X m)) * \text{norm } x$ 
        by (rule norm_blinfun)
      also have  $\dots \leq r' * \text{norm } x$ 
        using  $M[\text{OF } n \text{ elim}]$  by (simp add: mult_right_mono)
      finally show ?case .
    qed
  have tendsto_v:  $(\lambda m. \text{norm } (X n x - X m x)) \longrightarrow \text{norm } (X n x - \text{Blinfun } v x)$ 
    by (auto intro!: tendsto_intros Bv)
  show  $\text{norm } ((X n - \text{Blinfun } v) x) \leq r' * \text{norm } x$ 
    by (auto intro!: tendsto_upperbound tendsto_v ev_le simp: blinfun.bilinear_simps)
  qed (simp add:  $\langle 0 < r' \rangle$  less_imp_le)
  thus  $\text{norm } (X n - \text{Blinfun } v) < r$ 
    by (metis  $\langle r' < r \rangle$  le_less_trans)
  qed
qed
thus convergent  $X$ 
  by (rule convergentI)
qed

```

6.16.5 On Euclidean Space

```

lemma Zfun_sum:
  assumes finite  $s$ 
  assumes  $f: \bigwedge i. i \in s \implies Zfun (f i) F$ 
  shows  $Zfun (\lambda x. \text{sum } (\lambda i. f i x) s) F$ 
  using  $\text{assms}$  by induct (auto intro!: Zfun_zero Zfun_add)

```

```

lemma norm_blinfun_euclidean_le:
  fixes a::'a::euclidean_space  $\Rightarrow_L$  'b::real_normed_vector
  shows norm a  $\leq$  sum ( $\lambda x$ . norm (a x)) Basis
  apply (rule norm_blinfun_bound)
  apply (simp add: sum_nonneg)
  apply (subst euclidean_representation[symmetric, where 'a='a])
  apply (simp only: blinfun.bilinear_simps sum_distrib_right)
  apply (rule order.trans[OF norm_sum sum_mono])
  apply (simp add: abs_mult mult_right_mono ac_simps Basis_le_norm)
done

```

```

lemma tendsto_componentwise1:
  fixes a::'a::euclidean_space  $\Rightarrow_L$  'b::real_normed_vector
  and b::'c  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  assumes ( $\bigwedge j$ .  $j \in$  Basis  $\implies$  (( $\lambda n$ . b n j)  $\longrightarrow$  a j) F)
  shows (b  $\longrightarrow$  a) F
proof -
  have  $\bigwedge j$ .  $j \in$  Basis  $\implies$  Zfun ( $\lambda x$ . norm (b x j - a j)) F
  using assms unfolding tendsto_Zfun_iff Zfun_norm_iff .
  hence Zfun ( $\lambda x$ .  $\sum j \in$  Basis. norm (b x j - a j)) F
  by (auto intro!: Zfun_sum)
  thus ?thesis
  unfolding tendsto_Zfun_iff
  by (rule Zfun_le)
  (auto intro!: order_trans[OF norm_blinfun_euclidean_le] simp: blinfun.bilinear_simps)
qed

```

lift_definition

```

blinfun_of_matrix::('b::euclidean_space  $\Rightarrow$  'a::euclidean_space  $\Rightarrow$  real)  $\Rightarrow$  'a  $\Rightarrow_L$ 
'b
is  $\lambda a$  x.  $\sum i \in$  Basis.  $\sum j \in$  Basis. ((x  $\cdot$  j) * a i j) *R i
by (intro bounded_linear_intros)

```

lemma blinfun_of_matrix_works:

```

fixes f::'a::euclidean_space  $\Rightarrow_L$  'b::euclidean_space
shows blinfun_of_matrix ( $\lambda i$  j. (f j)  $\cdot$  i) = f
proof (transfer, rule, rule euclidean_eqI)
  fix f::'a  $\Rightarrow$  'b and x::'a and b::'b assume bounded_linear f and b: b  $\in$  Basis
  then interpret bounded_linear f by simp
  have ( $\sum j \in$  Basis.  $\sum i \in$  Basis. (x  $\cdot$  i * (f i  $\cdot$  j)) *R j)  $\cdot$  b
  = ( $\sum j \in$  Basis. if j = b then ( $\sum i \in$  Basis. (x  $\cdot$  i * (f i  $\cdot$  j))) else 0)
  using b
  by (simp add: inner_sum_left inner_Basis if_distrib cong: if_cong) (simp add:
sum.swap)
  also have ... = ( $\sum i \in$  Basis. (x  $\cdot$  i * (f i  $\cdot$  b)))
  using b by (simp)
  also have ... = f x  $\cdot$  b
  by (metis (mono_tags, lifting) Linear_Algebra.linear_componentwise linear_axioms)
  finally show ( $\sum j \in$  Basis.  $\sum i \in$  Basis. (x  $\cdot$  i * (f i  $\cdot$  j)) *R j)  $\cdot$  b = f x  $\cdot$  b .

```

qed

lemma *blinfun_of_matrix_apply*:

blinfun_of_matrix a $x = (\sum i \in \text{Basis}. \sum j \in \text{Basis}. ((x \cdot j) * a \ i \ j) *_{\mathbb{R}} i)$
by *transfer simp*

lemma *blinfun_of_matrix_minus*: *blinfun_of_matrix* $x - \text{blinfun_of_matrix } y$
 $= \text{blinfun_of_matrix } (x - y)$

by *transfer (auto simp: algebra_simps sum_subtractf)*

lemma *norm_blinfun_of_matrix*:

norm (blinfun_of_matrix a) ≤ (∑ i ∈ Basis. ∑ j ∈ Basis. |a i j|)

apply (*rule norm_blinfun_bound*)

apply (*simp add: sum_nonneg*)

apply (*simp only: blinfun_of_matrix_apply sum_distrib_right*)

apply (*rule order_trans[OF norm_sum sum_mono]*)

apply (*rule order_trans[OF norm_sum sum_mono]*)

apply (*simp add: abs_mult mult_right_mono ac_simps Basis_le_norm*)

done

lemma *tendsto_blinfun_of_matrix*:

assumes $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies ((\lambda n. b \ n \ i \ j) \longrightarrow a \ i \ j) \ F$

shows $((\lambda n. \text{blinfun_of_matrix } (b \ n)) \longrightarrow \text{blinfun_of_matrix } a) \ F$

proof –

have $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies \text{Zfun } (\lambda x. \text{norm } (b \ x \ i \ j - a \ i \ j)) \ F$

using *assms unfolding tendsto_Zfun_iff Zfun_norm_iff* .

hence $\text{Zfun } (\lambda x. (\sum i \in \text{Basis}. \sum j \in \text{Basis}. |b \ x \ i \ j - a \ i \ j|)) \ F$

by (*auto intro!: Zfun_sum*)

thus *?thesis*

unfolding *tendsto_Zfun_iff blinfun_of_matrix_minus*

by (*rule Zfun_le*) (*auto intro!: order_trans[OF norm_blinfun_of_matrix]*)

qed

lemma *tendsto_componentwise*:

fixes $a::'a::\text{euclidean_space} \Rightarrow_L 'b::\text{euclidean_space}$

and $b::'c \Rightarrow 'a \Rightarrow_L 'b$

shows $(\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies ((\lambda n. b \ n \ j \cdot i) \longrightarrow a \ j \cdot i) \ F) \implies (b \longrightarrow a) \ F$

apply (*subst blinfun_of_matrix_works[of a, symmetric]*)

apply (*subst blinfun_of_matrix_works[of b x for x, symmetric, abs_def]*)

by (*rule tendsto_blinfun_of_matrix*)

lemma

continuous_blinfun_componentwiseI:

fixes $f::'b::t2_space \Rightarrow 'a::\text{euclidean_space} \Rightarrow_L 'c::\text{euclidean_space}$

assumes $\bigwedge i \ j. i \in \text{Basis} \implies j \in \text{Basis} \implies \text{continuous } F \ (\lambda x. (f \ x) \ j \cdot i)$

shows *continuous F f*

using *assms by (auto simp: continuous_def intro!: tendsto_componentwise)*

lemma

continuous_blinfun_componentwiseI1:
fixes $f :: 'b::t2_space \Rightarrow 'a::euclidean_space \Rightarrow_L 'c::real_normed_vector$
assumes $\bigwedge i. i \in Basis \implies continuous\ F\ (\lambda x. f\ x\ i)$
shows $continuous\ F\ f$
using *assms* **by** (*auto simp: continuous_def intro!: tendsto_componentwise1*)

lemma

continuous_on_blinfun_componentwise:
fixes $f :: 'd::t2_space \Rightarrow 'e::euclidean_space \Rightarrow_L 'f::real_normed_vector$
assumes $\bigwedge i. i \in Basis \implies continuous_on\ s\ (\lambda x. f\ x\ i)$
shows $continuous_on\ s\ f$
using *assms*
by (*auto intro!: continuous_at_imp_continuous_on intro!: tendsto_componentwise1 simp: continuous_on_eq_continuous_within continuous_def*)

lemma *bounded_linear_blinfun_matrix*: $bounded_linear\ (\lambda x. (x::_ \Rightarrow_L _) j \cdot i)$
by (*auto intro!: bounded_linearI' bounded_linear_intros*)

lemma *continuous_blinfun_matrix*:

fixes $f :: 'b::t2_space \Rightarrow 'a::real_normed_vector \Rightarrow_L 'c::real_inner$
assumes $continuous\ F\ f$
shows $continuous\ F\ (\lambda x. (f\ x)\ j \cdot i)$
by (*rule bounded_linear.continuous[OF bounded_linear_blinfun_matrix assms]*)

lemma *continuous_on_blinfun_matrix*:

fixes $f :: 'a::t2_space \Rightarrow 'b::real_normed_vector \Rightarrow_L 'c::real_inner$
assumes $continuous_on\ S\ f$
shows $continuous_on\ S\ (\lambda x. (f\ x)\ j \cdot i)$
using *assms*
by (*auto simp: continuous_on_eq_continuous_within continuous_blinfun_matrix*)

lemma *continuous_on_blinfun_of_matrix[continuous_intros]*:

assumes $\bigwedge i\ j. i \in Basis \implies j \in Basis \implies continuous_on\ S\ (\lambda s. g\ s\ i\ j)$
shows $continuous_on\ S\ (\lambda s. blinfun_of_matrix\ (g\ s))$
using *assms*
by (*auto simp: continuous_on intro!: tendsto_blinfun_of_matrix*)

lemma *mult_if_delta*:

(*if P then* $(1::'a::comm_semiring_1)\ \text{else}\ 0$) * $q = (\text{if } P \text{ then } q \text{ else } 0)$
by *auto*

lemma *compact_blinfun_lemma*:

fixes $f :: nat \Rightarrow 'a::euclidean_space \Rightarrow_L 'b::euclidean_space$
assumes $bounded\ (range\ f)$
shows $\forall d \subseteq Basis. \exists l :: 'a \Rightarrow_L 'b. \exists r :: nat \Rightarrow nat.$
 $strict_mono\ r \wedge (\forall e > 0. eventually\ (\lambda n. \forall i \in d. dist\ (f\ (r\ n)\ i)\ (l\ i) < e)$
sequentially)
by (*rule compact_lemma_general[where unproj = $\lambda e. blinfun_of_matrix\ (\lambda i\ j.$*

```

e j · i))
  (auto intro!: euclidean_eqI[where 'a='b] bounded_linear_image assms
   simp: blinfun_of_matrix_works blinfun_of_matrix_apply inner_Basis mult_if_delta
   sum.delta'
   scaleR_sum_left[symmetric])

```

```

lemma blinfun_euclidean_eqI: ( $\bigwedge i. i \in \text{Basis} \implies \text{blinfun\_apply } x \ i = \text{blinfun\_apply } y \ i$ )  $\implies x = y$ 
apply (auto intro!: blinfun_eqI)
apply (subst (2) euclidean_representation[symmetric, where 'a='a])
apply (subst (1) euclidean_representation[symmetric, where 'a='a])
apply (simp add: blinfun.bilinear_simps)
done

```

```

lemma Blinfun_eq_matrix: bounded_linear f  $\implies \text{Blinfun } f = \text{blinfun\_of\_matrix}$ 
( $\lambda i \ j. f \ j \cdot i$ )
by (intro blinfun_euclidean_eqI)
  (auto simp: blinfun_of_matrix_apply bounded_linear_Blinfun_apply inner_Basis
if_distrib
if_distribR sum.delta' euclidean_representation
cong: if_cong)

```

TODO: generalize (via *compact_cball*)?

```

instance blinfun :: (euclidean_space, euclidean_space) heine_borel
proof
  fix f :: nat  $\Rightarrow$  'a  $\Rightarrow_L$  'b
  assume f: bounded (range f)
  then obtain l::'a  $\Rightarrow_L$  'b and r where r: strict_mono r
  and l:  $\forall e > 0. \text{eventually } (\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e)$  sequentially
  using compact_blinfun_lemma [OF f] by blast
  {
    fix e::real
    let ?d = real_of_nat DIM('a) * real_of_nat DIM('b)
    assume e > 0
    hence e / ?d > 0 by (simp)
    with l have eventually ( $\lambda n. \forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e / ?d$ ) sequentially
      by simp
    moreover
    {
      fix n
      assume n:  $\forall i \in \text{Basis}. \text{dist } (f \ (r \ n) \ i) \ (l \ i) < e / ?d$ 
      have norm (f (r n) - l) = norm (blinfun_of_matrix ( $\lambda i \ j. (f \ (r \ n) \ - l) \ j \cdot$ 
i))
        unfolding blinfun_of_matrix_works ..
        also note norm_blinfun_of_matrix
        also have ( $\sum i \in \text{Basis}. \sum j \in \text{Basis}. |(f \ (r \ n) \ - l) \ j \cdot i|$ ) <
          ( $\sum i \in (\text{Basis}::'b \ \text{set}). e / \text{real\_of\_nat } \text{DIM}('b)$ )
        proof (rule sum_strict_mono)
          fix i::'b assume i: i  $\in$  Basis

```

```

have ( $\sum j::'a \in \text{Basis}. |(f (r n) - l) j \cdot i|$ ) < ( $\sum j::'a \in \text{Basis}. e / ?d$ )
proof (rule sum_strict_mono)
  fix j::'a assume j:  $j \in \text{Basis}$ 
  have  $|(f (r n) - l) j \cdot i| \leq \text{norm} ((f (r n) - l) j)$ 
    by (simp add: Basis_le_norm i)
  also have  $\dots < e / ?d$ 
    using n i j by (auto simp: dist_norm blinfun.bilinear_simps)
  finally show  $|(f (r n) - l) j \cdot i| < e / ?d$  by simp
qed simp_all
also have  $\dots \leq e / \text{real\_of\_nat } \text{DIM}(b)$ 
  by simp
finally show ( $\sum j \in \text{Basis}. |(f (r n) - l) j \cdot i|$ ) <  $e / \text{real\_of\_nat } \text{DIM}(b)$ 
  by simp
qed simp_all
also have  $\dots \leq e$  by simp
finally have  $\text{dist} (f (r n)) l < e$ 
  by (auto simp: dist_norm)
}
ultimately have eventually ( $\lambda n. \text{dist} (f (r n)) l < e$ ) sequentially
  using eventually_elim2 by force
}
then have *:  $((f \circ r) \longrightarrow l)$  sequentially
  unfolding o_def tendsto_iff by simp
with r show  $\exists l r. \text{strict\_mono } r \wedge ((f \circ r) \longrightarrow l)$  sequentially
  by auto
qed

```

6.16.6 concrete bounded linear functions

```

lemma transfer_bounded_bilinear_bounded_linearI:
  assumes  $g = (\lambda i x. (\text{blinfun\_apply } (f i) x))$ 
  shows  $\text{bounded\_bilinear } g = \text{bounded\_linear } f$ 
proof
  assume bounded_bilinear g
  then interpret bounded_bilinear f by (simp add: assms)
  show bounded_linear f
proof (unfold locales, safe intro!: blinfun_eqI)
  fix i
  show  $f (x + y) i = (f x + f y) i$ 
    by (auto intro!: blinfun_eqI simp: blinfun.bilinear_simps)
  from _ nonneg_bounded show  $\exists K. \forall x. \text{norm} (f x) \leq \text{norm } x * K$ 
    by (rule ex_reg) (auto intro!: onorm_bound simp: norm_blinfun.rep_eq ac_simps)
  qed
qed (auto simp: assms intro!: blinfun.comp)

lemma transfer_bounded_bilinear_bounded_linear[transfer_rule]:
  ( $\text{rel\_fun } (\text{rel\_fun } (=) (\text{pcr\_blinfun } (=) (=))) (=) \text{bounded\_bilinear bounded\_linear}$ )
  by (auto simp: pcr_blinfun_def cr_blinfun_def rel_fun_def OO_def)

```

```

    intro!: transfer_bounded_bilinear_bounded_linearI)

context bounded_bilinear
begin

lift_definition prod_left::'b  $\Rightarrow$  'a  $\Rightarrow_L$  'c is ( $\lambda b a.$  prod a b)
  by (rule bounded_linear_left)
declare prod_left.rep_eq[simp]

lemma bounded_linear_prod_left[bounded_linear]: bounded_linear prod_left
  by transfer (rule flip)

lift_definition prod_right::'a  $\Rightarrow$  'b  $\Rightarrow_L$  'c is ( $\lambda a b.$  prod a b)
  by (rule bounded_linear_right)
declare prod_right.rep_eq[simp]

lemma bounded_linear_prod_right[bounded_linear]: bounded_linear prod_right
  by transfer (rule bounded_bilinear_axioms)

end

lift_definition id_blinfun::'a::real_normed_vector  $\Rightarrow_L$  'a is  $\lambda x.$  x
  by (rule bounded_linear_ident)

lemmas blinfun_apply_id_blinfun[simp] = id_blinfun.rep_eq

lemma norm_blinfun_id[simp]:
  norm (id_blinfun::'a::{real_normed_vector, perfect_space}  $\Rightarrow_L$  'a) = 1
  by transfer (auto simp: onorm_id)

lemma norm_blinfun_id_le:
  norm (id_blinfun::'a::real_normed_vector  $\Rightarrow_L$  'a)  $\leq$  1
  by transfer (auto simp: onorm_id_le)

lift_definition fst_blinfun::('a::real_normed_vector  $\times$  'b::real_normed_vector)
 $\Rightarrow_L$  'a is fst
  by (rule bounded_linear_fst)

lemma blinfun_apply_fst_blinfun[simp]: blinfun_apply fst_blinfun = fst
  by transfer (rule refl)

lift_definition snd_blinfun::('a::real_normed_vector  $\times$  'b::real_normed_vector)
 $\Rightarrow_L$  'b is snd
  by (rule bounded_linear_snd)

lemma blinfun_apply_snd_blinfun[simp]: blinfun_apply snd_blinfun = snd
  by transfer (rule refl)

```

```

lift_definition blinfun_compose::
  'a::real_normed_vector  $\Rightarrow_L$  'b::real_normed_vector  $\Rightarrow$ 
  'c::real_normed_vector  $\Rightarrow_L$  'a  $\Rightarrow$ 
  'c  $\Rightarrow_L$  'b (infixl o_L 55) is (o)
parametric comp_transfer
unfolding o_def
by (rule bounded_linear_compose)

lemma blinfun_apply_blinfun_compose[simp]: (a o_L b) c = a (b c)
by (simp add: blinfun_compose.rep_eq)

lemma norm_blinfun_compose:
  norm (f o_L g)  $\leq$  norm f * norm g
by transfer (rule onorm_compose)

lemma bounded_bilinear_blinfun_compose[bounded_bilinear]: bounded_bilinear (o_L)
by unfold_locales
  (auto intro!: blinfun_eqI exI[where x=1] simp: blinfun.bilinear_simps norm_blinfun_compose)

lemma blinfun_compose_zero[simp]:
  blinfun_compose 0 = ( $\lambda$ _. 0)
  blinfun_compose x 0 = 0
by (auto simp: blinfun.bilinear_simps intro!: blinfun_eqI)

lemma blinfun_bij2:
  fixes f::'a  $\Rightarrow_L$  'a::euclidean_space
  assumes f o_L g = id_blinfun
  shows bij (blinfun_apply g)
proof (rule bijI)
  show inj g
    using assms
  by (metis blinfun_apply_id_blinfun blinfun_compose.rep_eq injI inj_on_imageI2)
  then show surj g
    using blinfun.bounded_linear_right bounded_linear_def linear_inj_imp_surj
by blast
qed

lemma blinfun_bij1:
  fixes f::'a  $\Rightarrow_L$  'a::euclidean_space
  assumes f o_L g = id_blinfun
  shows bij (blinfun_apply f)
proof (rule bijI)
  show surj (blinfun_apply f)
    by (metis assms blinfun_apply_blinfun_compose blinfun_apply_id_blinfun
surjI)
  then show inj (blinfun_apply f)
    using blinfun.bounded_linear_right bounded_linear_def linear_surj_imp_inj

```


by blast
qed

lift_definition *blinfun_inner_right*:: $'a::\text{real_inner} \Rightarrow 'a \Rightarrow_L \text{real}$ **is** (\cdot)
by (rule *bounded_linear_inner_right*)
declare *blinfun_inner_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_right*[*bounded_linear*]: *bounded_linear blinfun_inner_right*
by transfer (rule *bounded_bilinear_inner*)

lift_definition *blinfun_inner_left*:: $'a::\text{real_inner} \Rightarrow 'a \Rightarrow_L \text{real}$ **is** $\lambda x y. y \cdot x$
by (rule *bounded_linear_inner_left*)
declare *blinfun_inner_left.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_left*[*bounded_linear*]: *bounded_linear blinfun_inner_left*
by transfer (rule *bounded_bilinear_flip*[*OF* *bounded_bilinear_inner*])

lift_definition *blinfun_scaleR_right*:: $\text{real} \Rightarrow 'a \Rightarrow_L 'a::\text{real_normed_vector}$ **is** $(*_R)$
by (rule *bounded_linear_scaleR_right*)
declare *blinfun_scaleR_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_scaleR_right*[*bounded_linear*]: *bounded_linear blinfun_scaleR_right*
by transfer (rule *bounded_bilinear_scaleR*)

lift_definition *blinfun_scaleR_left*:: $'a::\text{real_normed_vector} \Rightarrow \text{real} \Rightarrow_L 'a$ **is** $\lambda x y. y *_R x$
by (rule *bounded_linear_scaleR_left*)
lemmas [*simp*] = *blinfun_scaleR_left.rep_eq*

lemma *bounded_linear_blinfun_scaleR_left*[*bounded_linear*]: *bounded_linear blinfun_scaleR_left*
by transfer (rule *bounded_bilinear_flip*[*OF* *bounded_bilinear_scaleR*])

lift_definition *blinfun_mult_right*:: $'a \Rightarrow 'a \Rightarrow_L 'a::\text{real_normed_algebra}$ **is** $(*)$
by (rule *bounded_linear_mult_right*)
declare *blinfun_mult_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_mult_right*[*bounded_linear*]: *bounded_linear blinfun_mult_right*
by transfer (rule *bounded_bilinear_mult*)

lift_definition *blinfun_mult_left*::*'a::real_normed_algebra* \Rightarrow *'a* \Rightarrow_L *'a* **is** $\lambda x y.$
 $y * x$
by (*rule bounded_linear_mult_left*)
lemmas [*simp*] = *blinfun_mult_left.rep_eq*

lemma *bounded_linear_blinfun_mult_left*[*bounded_linear*]: *bounded_linear blinfun_mult_left*
by *transfer (rule bounded_bilinear.flip[OF bounded_bilinear_mult])*

lemmas *bounded_linear_function_uniform_limit_intros*[*uniform_limit_intros*] =
bounded_linear.uniform_limit[OF bounded_linear_apply_blinfun]
bounded_linear.uniform_limit[OF bounded_linear_blinfun_apply]
bounded_linear.uniform_limit[OF bounded_linear_blinfun_matrix]

6.16.7 The strong operator topology on continuous linear operators

Let *'a* and *'b* be two normed real vector spaces. Then the space of linear continuous operators from *'a* to *'b* has a canonical norm, and therefore a canonical corresponding topology (the type classes instantiation are given in `Bounded_Linear_Function.thy`).

However, there is another topology on this space, the strong operator topology, where T_n tends to T iff, for all x in *'a*, then $T_n x$ tends to $T x$. This is precisely the product topology where the target space is endowed with the norm topology. It is especially useful when *'b* is the set of real numbers, since then this topology is compact.

We can not implement it using type classes as there is already a topology, but at least we can define it as a topology.

Note that there is yet another (common and useful) topology on operator spaces, the weak operator topology, defined analogously using the product topology, but where the target space is given the weak-* topology, i.e., the pullback of the weak topology on the bidual of the space under the canonical embedding of a space into its bidual. We do not define it there, although it could also be defined analogously.

definition *strong_operator_topology*::(*'a::real_normed_vector* \Rightarrow_L *'b::real_normed_vector*)
topology
where *strong_operator_topology* = *pullback_topology UNIV blinfun_apply euclidean*

lemma *strong_operator_topology_topspace*:
topspace strong_operator_topology = *UNIV*

unfolding *strong_operator_topology_def topspace_pullback_topology topspace_euclidean*
by *auto*

lemma *strong_operator_topology_basis*:
fixes *f*::(*'a::real_normed_vector* \Rightarrow_L *'b::real_normed_vector*) **and** *U*::*'i* \Rightarrow *'b* *set*

```

and x::'i  $\Rightarrow$  'a
  assumes finite I  $\wedge$  i. i  $\in$  I  $\implies$  open (U i)
  shows openin strong_operator_topology {f.  $\forall$  i $\in$ I. blinfun_apply f (x i)  $\in$  U i}
proof -
  have open {g::('a $\Rightarrow$ 'b).  $\forall$  i $\in$ I. g (x i)  $\in$  U i}
    by (rule product_topology_basis'[OF assms])
  moreover have {f.  $\forall$  i $\in$ I. blinfun_apply f (x i)  $\in$  U i}
    = blinfun_apply- '{g::('a $\Rightarrow$ 'b).  $\forall$  i $\in$ I. g (x i)  $\in$  U i}  $\cap$  UNIV
    by auto
  ultimately show ?thesis
    unfolding strong_operator_topology_def by (subst openin_pullback_topology)
auto
qed

```

```

lemma strong_operator_topology_continuous_evaluation:
  continuous_map strong_operator_topology euclidean ( $\lambda$ f. blinfun_apply f x)
proof -
  have continuous_map strong_operator_topology euclidean (( $\lambda$ f. f x) o blinfun_apply)
    unfolding strong_operator_topology_def apply (rule continuous_map_pullback)
    using continuous_on_product_coordinates by fastforce
  then show ?thesis unfolding comp_def by simp
qed

```

```

lemma continuous_on_strong_operator_topo_iff_coordinatewise:
  continuous_map T strong_operator_topology f
   $\iff$  ( $\forall$  x. continuous_map T euclidean ( $\lambda$ y. blinfun_apply (f y) x))
proof (auto)
  fix x::'b
  assume continuous_map T strong_operator_topology f
  with continuous_map_compose[OF this strong_operator_topology_continuous_evaluation]
  have continuous_map T euclidean (( $\lambda$ z. blinfun_apply z x) o f)
    by simp
  then show continuous_map T euclidean ( $\lambda$ y. blinfun_apply (f y) x)
    unfolding comp_def by auto
next
  assume *:  $\forall$  x. continuous_map T euclidean ( $\lambda$ y. blinfun_apply (f y) x)
  have  $\wedge$  i. continuous_map T euclidean ( $\lambda$ x. blinfun_apply (f x) i)
    using * unfolding comp_def by auto
  then have continuous_map T euclidean (blinfun_apply o f)
    unfolding o_def
    by (metis (no_types) continuous_map_componentwise_UNIV euclidean_product_topology)
  show continuous_map T strong_operator_topology f
    unfolding strong_operator_topology_def
    apply (rule continuous_map_pullback')
    by (auto simp add: <continuous_map T euclidean (blinfun_apply o f)>)
qed

```

```

lemma strong_operator_topology_weaker_than_euclidean:

```

1832

```
continuous_map euclidean_strong_operator_topology (λf. f)
by (subst continuous_on_strong_operator_topo_iff_coordinatewise,
    auto simp add: linear_continuous_on)
```

end

6.17 Derivative

theory Derivative

imports

Bounded_Linear_Function

Line_Segment

Convex_Euclidean_Space

begin

declare bounded_linear_inner_left [intro]

declare has_derivative_bounded_linear [dest]

6.17.1 Derivatives

lemma has_derivative_add_const:

$(f \text{ has_derivative } f') \text{ net} \implies ((\lambda x. f x + c) \text{ has_derivative } f') \text{ net}$

by (intro derivative_eq_intros) auto

6.17.2 Derivative with composed bilinear function

More explicit epsilon-delta forms.

proposition has_derivative_within':

$(f \text{ has_derivative } f') \text{ (at } x \text{ within } s) \iff$

$\text{bounded_linear } f' \wedge$

$(\forall e > 0. \exists d > 0. \forall x' \in s. 0 < \text{norm } (x' - x) \wedge \text{norm } (x' - x) < d \implies$

$\text{norm } (f x' - f x - f'(x' - x)) / \text{norm } (x' - x) < e)$

unfolding has_derivative_within Lim_within dist_norm

by (simp add: diff_diff_eq)

lemma has_derivative_at':

$(f \text{ has_derivative } f') \text{ (at } x)$

$\iff \text{bounded_linear } f' \wedge$

$(\forall e > 0. \exists d > 0. \forall x'. 0 < \text{norm } (x' - x) \wedge \text{norm } (x' - x) < d \implies$

$\text{norm } (f x' - f x - f'(x' - x)) / \text{norm } (x' - x) < e)$

using has_derivative_within' [of f f' x UNIV] **by** simp

lemma has_derivative_componentwise_within:

$(f \text{ has_derivative } f') \text{ (at } a \text{ within } S) \iff$

$(\forall i \in \text{Basis}. ((\lambda x. f x \cdot i) \text{ has_derivative } (\lambda x. f' x \cdot i)) \text{ (at } a \text{ within } S))$

apply (simp add: has_derivative_within)

apply (subst tendsto_componentwise_iff)

apply (*simp add: ball_conj_distrib inner_diff_left inner_left_distrib flip: bounded_linear_componentwise_iff*)
done

lemma *has_derivative_at_withinI*:

(*f has_derivative f'*) (at *x*) \implies (*f has_derivative f'*) (at *x* within *s*)
unfolding *has_derivative_within' has_derivative_at'*
by *blast*

lemma *has_derivative_right*:

fixes *f* :: *real* \Rightarrow *real*
and *y* :: *real*
shows (*f has_derivative* ((* *y*)) (at *x* within ($\{x <..\} \cap I$)) \longleftrightarrow
 $((\lambda t. (f x - f t) / (x - t)) \longrightarrow y)$ (at *x* within ($\{x <..\} \cap I$)))
proof –
have $((\lambda t. (f t - (f x + y * (t - x))) / |t - x|) \longrightarrow 0)$ (at *x* within ($\{x <..\} \cap I$)) \longleftrightarrow
 $((\lambda t. (f t - f x) / (t - x) - y) \longrightarrow 0)$ (at *x* within ($\{x <..\} \cap I$))
by (*intro Lim_cong_within*) (*auto simp add: diff_divide_distrib add_divide_distrib*)
also have ... $\longleftrightarrow ((\lambda t. (f t - f x) / (t - x)) \longrightarrow y)$ (at *x* within ($\{x <..\} \cap I$))
by (*simp add: Lim_null[symmetric]*)
also have ... $\longleftrightarrow ((\lambda t. (f x - f t) / (x - t)) \longrightarrow y)$ (at *x* within ($\{x <..\} \cap I$))
by (*intro Lim_cong_within*) (*simp_all add: field_simps*)
finally show *?thesis*
by (*simp add: bounded_linear_mult_right has_derivative_within*)
qed

Caratheodory characterization

lemma *DERIV_caratheodory_within*:

(*f has_field_derivative l*) (at *x* within *S*) \longleftrightarrow
 $(\exists g. (\forall z. f z - f x = g z * (z - x)) \wedge \text{continuous (at } x \text{ within } S) g \wedge g x = l)$
(is ?lhs = ?rhs)

proof

assume *?lhs*

show *?rhs*

proof (*intro exI conjI*)

let *?g* = ($\%z. \text{if } z = x \text{ then } l \text{ else } (f z - f x) / (z - x)$)

show $\forall z. f z - f x = ?g z * (z - x)$ **by** *simp*

show *continuous* (at *x* within *S*) *?g* **using** $\langle ?lhs \rangle$

by (*auto simp add: continuous_within has_field_derivative_iff cong: Lim_cong_within*)

show *?g x = l* **by** *simp*

qed

next

assume *?rhs*

then obtain *g* **where**

$(\forall z. f z - f x = g z * (z - x))$ **and** *continuous* (at *x* within *S*) *g* **and** *g x = l*

by *blast*

```

thus ?lhs
by (auto simp add: continuous_within has_field_derivative_iff cong: Lim_cong_within)
qed

```

6.17.3 Differentiability

definition

```

differentiable_on :: ('a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector)  $\Rightarrow$  'a set
 $\Rightarrow$  bool

```

```

(infix differentiable'_on 50)

```

```

where f differentiable_on s  $\longleftrightarrow$  ( $\forall x \in s. f$  differentiable (at x within s))

```

```

lemma differentiableI: (f has_derivative f') net  $\Longrightarrow$  f differentiable net

```

```

unfolding differentiable_def

```

```

by auto

```

```

lemma differentiable_onD:  $\llbracket f$  differentiable_on S;  $x \in S \rrbracket \Longrightarrow f$  differentiable (at
x within S)

```

```

using differentiable_on_def by blast

```

```

lemma differentiable_at_withinI: f differentiable (at x)  $\Longrightarrow$  f differentiable (at x
within s)

```

```

unfolding differentiable_def

```

```

using has_derivative_at_withinI

```

```

by blast

```

```

lemma differentiable_at_imp_differentiable_on:

```

```

( $\bigwedge x. x \in s \Longrightarrow f$  differentiable at x)  $\Longrightarrow$  f differentiable_on s

```

```

by (metis differentiable_at_withinI differentiable_on_def)

```

```

corollary differentiable_iff_scaleR:

```

```

fixes f :: real  $\Rightarrow$  'a::real_normed_vector

```

```

shows f differentiable F  $\longleftrightarrow$  ( $\exists d. (f$  has_derivative ( $\lambda x. x *_R d$ )) F)

```

```

by (auto simp: differentiable_def dest: has_derivative_linear linear_imp_scaleR)

```

```

lemma differentiable_on_eq_differentiable_at:

```

```

open s  $\Longrightarrow$  f differentiable_on s  $\longleftrightarrow$  ( $\forall x \in s. f$  differentiable at x)

```

```

unfolding differentiable_on_def

```

```

by (metis at_within_interior interior_open)

```

```

lemma differentiable_transform_within:

```

```

assumes f differentiable (at x within s)

```

```

and 0 < d

```

```

and x  $\in$  s

```

```

and  $\bigwedge x'. \llbracket x' \in s; \text{dist } x' x < d \rrbracket \Longrightarrow f x' = g x'$ 

```

```

shows g differentiable (at x within s)

```

```

using assms has_derivative_transform_within unfolding differentiable_def

```

```

by blast

```

lemma *differentiable_on_ident* [*simp*, *derivative_intros*]: $(\lambda x. x)$ *differentiable_on* S

by (*simp add: differentiable_at_imp_differentiable_on*)

lemma *differentiable_on_id* [*simp*, *derivative_intros*]: *id* *differentiable_on* S

by (*simp add: id_def*)

lemma *differentiable_on_const* [*simp*, *derivative_intros*]: $(\lambda z. c)$ *differentiable_on* S

by (*simp add: differentiable_on_def*)

lemma *differentiable_on_mult* [*simp*, *derivative_intros*]:

fixes $f :: 'M :: \text{real_normed_vector} \Rightarrow 'a :: \text{real_normed_algebra}$

shows $\llbracket f \text{ differentiable_on } S; g \text{ differentiable_on } S \rrbracket \Longrightarrow (\lambda z. f z * g z)$ *differentiable_on* S

unfolding *differentiable_on_def differentiable_def*

using *differentiable_def differentiable_mult* **by** *blast*

lemma *differentiable_on_compose*:

$\llbracket g \text{ differentiable_on } S; f \text{ differentiable_on } (g \text{ ` } S) \rrbracket \Longrightarrow (\lambda x. f (g x))$ *differentiable_on* S

by (*simp add: differentiable_in_compose differentiable_on_def*)

lemma *bounded_linear_imp_differentiable_on*: *bounded_linear* $f \Longrightarrow f$ *differentiable_on* S

by (*simp add: differentiable_on_def bounded_linear_imp_differentiable*)

lemma *linear_imp_differentiable_on*:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{real_normed_vector}$

shows *linear* $f \Longrightarrow f$ *differentiable_on* S

by (*simp add: differentiable_on_def linear_imp_differentiable*)

lemma *differentiable_on_minus* [*simp*, *derivative_intros*]:

f *differentiable_on* $S \Longrightarrow (\lambda z. -(f z))$ *differentiable_on* S

by (*simp add: differentiable_on_def*)

lemma *differentiable_on_add* [*simp*, *derivative_intros*]:

$\llbracket f \text{ differentiable_on } S; g \text{ differentiable_on } S \rrbracket \Longrightarrow (\lambda z. f z + g z)$ *differentiable_on* S

by (*simp add: differentiable_on_def*)

lemma *differentiable_on_diff* [*simp*, *derivative_intros*]:

$\llbracket f \text{ differentiable_on } S; g \text{ differentiable_on } S \rrbracket \Longrightarrow (\lambda z. f z - g z)$ *differentiable_on* S

by (*simp add: differentiable_on_def*)

lemma *differentiable_on_inverse* [*simp*, *derivative_intros*]:

fixes $f :: 'a :: \text{real_normed_vector} \Rightarrow 'b :: \text{real_normed_field}$

shows $f \text{ differentiable_on } S \Longrightarrow (\bigwedge x. x \in S \Longrightarrow f x \neq 0) \Longrightarrow (\lambda x. \text{inverse } (f$

x) *differentiable_on S*
by (*simp add: differentiable_on_def*)

lemma *differentiable_on_scaleR* [*derivative_intros, simp*]:
 $\llbracket f \text{ differentiable_on } S; g \text{ differentiable_on } S \rrbracket \implies (\lambda x. f x *_R g x) \text{ differentiable_on } S$
unfolding *differentiable_on_def*
by (*blast intro: differentiable_scaleR*)

lemma *has_derivative_sqnorm_at* [*derivative_intros, simp*]:
 $((\lambda x. (\text{norm } x)^2) \text{ has_derivative } (\lambda x. 2 *_R (a \cdot x))) \text{ (at } a)$
using *bounded_bilinear.FDERIV [of (\cdot) id id a _ id id]*
by (*auto simp: inner_commute dot_square_norm bounded_bilinear_inner*)

lemma *differentiable_sqnorm_at* [*derivative_intros, simp*]:
fixes $a :: 'a :: \{\text{real_normed_vector, real_inner}\}$
shows $(\lambda x. (\text{norm } x)^2) \text{ differentiable (at } a)$
by (*force simp add: differentiable_def intro: has_derivative_sqnorm_at*)

lemma *differentiable_on_sqnorm* [*derivative_intros, simp*]:
fixes $S :: 'a :: \{\text{real_normed_vector, real_inner}\} \text{ set}$
shows $(\lambda x. (\text{norm } x)^2) \text{ differentiable_on } S$
by (*simp add: differentiable_at_imp_differentiable_on*)

lemma *differentiable_norm_at* [*derivative_intros, simp*]:
fixes $a :: 'a :: \{\text{real_normed_vector, real_inner}\}$
shows $a \neq 0 \implies \text{norm differentiable (at } a)$
using *differentiableI has_derivative_norm* **by** *blast*

lemma *differentiable_on_norm* [*derivative_intros, simp*]:
fixes $S :: 'a :: \{\text{real_normed_vector, real_inner}\} \text{ set}$
shows $0 \notin S \implies \text{norm differentiable_on } S$
by (*metis differentiable_at_imp_differentiable_on differentiable_norm_at*)

6.17.4 Frechet derivative and Jacobian matrix

definition *frechet_derivative f net = (SOME f'. (f has_derivative f') net)*

proposition *frechet_derivative_works*:
 $f \text{ differentiable } net \iff (f \text{ has_derivative } (\text{frechet_derivative } f \text{ net})) \text{ net}$
unfolding *frechet_derivative_def differentiable_def*
unfolding *some_eq_ex[of $\lambda f'. (f \text{ has_derivative } f') \text{ net}$]* ..

lemma *linear_frechet_derivative*: $f \text{ differentiable } net \implies \text{linear } (\text{frechet_derivative } f \text{ net})$
unfolding *frechet_derivative_works has_derivative_def*
by (*auto intro: bounded_linear.linear*)

lemma *frechet_derivative_const* [*simp*]: $\text{frechet_derivative } (\lambda x. c) \text{ (at } a) = (\lambda x.$

0)
using *differentiable_const frechet_derivative_works has_derivative_const has_derivative_unique*
by *blast*

lemma *frechet_derivative_id* [*simp*]: *frechet_derivative id (at a) = id*
using *differentiable_def frechet_derivative_works has_derivative_id has_derivative_unique*
by *blast*

lemma *frechet_derivative_ident* [*simp*]: *frechet_derivative ($\lambda x. x$) (at a) = ($\lambda x. x$)*
by (*metis eq_id_iff frechet_derivative_id*)

6.17.5 Differentiability implies continuity

proposition *differentiable_imp_continuous_within*:
f differentiable (at x within s) \implies continuous (at x within s) f
by (*auto simp: differentiable_def intro: has_derivative_continuous*)

lemma *differentiable_imp_continuous_on*:
f differentiable_on s \implies continuous_on s f
unfolding *differentiable_on_def continuous_on_eq_continuous_within*
using *differentiable_imp_continuous_within* **by** *blast*

lemma *differentiable_on_subset*:
f differentiable_on t \implies s \subseteq t \implies f differentiable_on s
unfolding *differentiable_on_def*
using *differentiable_within_subset*
by *blast*

lemma *differentiable_on_empty*: *f differentiable_on {}*
unfolding *differentiable_on_def*
by *auto*

lemma *has_derivative_continuous_on*:
 $(\bigwedge x. x \in s \implies (f \text{ has_derivative } f' x) (at x \text{ within } s)) \implies \text{continuous_on } s f$
by (*auto intro!: differentiable_imp_continuous_on differentiableI simp: differentiable_on_def*)

Results about neighborhoods filter.

lemma *eventually_nhds_metric_le*:
eventually P (nhds a) = $(\exists d > 0. \forall x. \text{dist } x a \leq d \longrightarrow P x)$
unfolding *eventually_nhds_metric* **by** (*safe, rule_tac x=d / 2 in exI, auto*)

lemma *le_nhds*: $F \leq \text{nhds } a \iff (\forall S. \text{open } S \wedge a \in S \longrightarrow \text{eventually } (\lambda x. x \in S) F)$
unfolding *le_filter_def eventually_nhds* **by** (*fast elim: eventually_mono*)

lemma *le_nhds_metric*: $F \leq \text{nhds } a \iff (\forall e > 0. \text{eventually } (\lambda x. \text{dist } x a < e) F)$

unfolding *le_filter_def eventually_nhds_metric* **by** (*fast elim: eventually_mono*)

lemma *le_nhds_metric_le*: $F \leq \text{nhds } a \iff (\forall e > 0. \text{eventually } (\lambda x. \text{dist } x \ a \leq e) \ F)$

unfolding *le_filter_def eventually_nhds_metric_le* **by** (*fast elim: eventually_mono*)

Several results are easier using a "multiplied-out" variant. (I got this idea from Dieudonne's proof of the chain rule).

lemma *has_derivative_within_alt*:

$(f \text{ has_derivative } f') \text{ (at } x \text{ within } s) \iff \text{bounded_linear } f' \wedge$
 $(\forall e > 0. \exists d > 0. \forall y \in s. \text{norm}(y - x) < d \implies \text{norm}(f y - f x - f'(y - x)) \leq$
 $e * \text{norm}(y - x))$

unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*

by (*force simp add: linear_0 bounded_linear.linear_pos_divide_le_eq*)

lemma *has_derivative_within_alt2*:

$(f \text{ has_derivative } f') \text{ (at } x \text{ within } s) \iff \text{bounded_linear } f' \wedge$
 $(\forall e > 0. \text{eventually } (\lambda y. \text{norm}(f y - f x - f'(y - x)) \leq e * \text{norm}(y - x))$
 $\text{(at } x \text{ within } s))$

unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*

by (*force simp add: linear_0 bounded_linear.linear_pos_divide_le_eq*)

lemma *has_derivative_at_alt*:

$(f \text{ has_derivative } f') \text{ (at } x) \iff$
 $\text{bounded_linear } f' \wedge$
 $(\forall e > 0. \exists d > 0. \forall y. \text{norm}(y - x) < d \implies \text{norm}(f y - f x - f'(y - x)) \leq e * \text{norm}(y - x))$

using *has_derivative_within_alt* [**where** $s = \text{UNIV}$]

by *simp*

6.17.6 The chain rule

proposition *diff_chain_within* [*derivative_intros*]:

assumes $(f \text{ has_derivative } f') \text{ (at } x \text{ within } s)$

and $(g \text{ has_derivative } g') \text{ (at } (f x) \text{ within } (f' s))$

shows $((g \circ f) \text{ has_derivative } (g' \circ f')) \text{ (at } x \text{ within } s)$

using *has_derivative_in_compose* [*OF assms*]

by (*simp add: comp_def*)

lemma *diff_chain_at* [*derivative_intros*]:

$(f \text{ has_derivative } f') \text{ (at } x) \implies$

$(g \text{ has_derivative } g') \text{ (at } (f x)) \implies ((g \circ f) \text{ has_derivative } (g' \circ f')) \text{ (at } x)$

by (*meson diff_chain_within has_derivative_at_withinI*)

lemma *has_vector_derivative_shift*: $(f \text{ has_vector_derivative } D x) \text{ (at } x)$

$\implies ((+) d \circ f \text{ has_vector_derivative } D x) \text{ (at } x)$

using *diff_chain_at* [*OF _ shift_has_derivative_id*]

by (simp add: has_derivative_iff_Ex has_vector_derivative_def)

lemma *has_vector_derivative_within_open*:

$a \in S \implies \text{open } S \implies$

$(f \text{ has_vector_derivative } f') \text{ (at } a \text{ within } S) \longleftrightarrow (f \text{ has_vector_derivative } f') \text{ (at } a)$

by (simp only: at_within_interior interior_open)

lemma *field_vector_diff_chain_within*:

assumes Df : $(f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$

and Dg : $(g \text{ has_field_derivative } g') \text{ (at } (f \ x) \text{ within } f' \ S)$

shows $((g \circ f) \text{ has_vector_derivative } (f' * g')) \text{ (at } x \text{ within } S)$

using *diff_chain_within[OF Df[unfolded has_vector_derivative_def]*

Dg [unfolded has_field_derivative_def]]

by (auto simp: o_def mult.commute has_vector_derivative_def)

lemma *vector_derivative_diff_chain_within*:

assumes Df : $(f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$

and Dg : $(g \text{ has_derivative } g') \text{ (at } (f \ x) \text{ within } f' \ S)$

shows $((g \circ f) \text{ has_vector_derivative } (g' f')) \text{ (at } x \text{ within } S)$

using *diff_chain_within[OF Df[unfolded has_vector_derivative_def] Dg]*

linear.scaleR[OF has_derivative_linear[OF Dg]]

unfolding *has_vector_derivative_def o_def*

by (auto simp: o_def mult.commute has_vector_derivative_def)

6.17.7 Composition rules stated just for differentiability

lemma *differentiable_chain_at*:

$f \text{ differentiable (at } x) \implies$

$g \text{ differentiable (at } (f \ x)) \implies (g \circ f) \text{ differentiable (at } x)$

unfolding *differentiable_def*

by (meson *diff_chain_at*)

lemma *differentiable_chain_within*:

$f \text{ differentiable (at } x \text{ within } S) \implies$

$g \text{ differentiable (at } (f \ x) \text{ within } (f' \ S)) \implies (g \circ f) \text{ differentiable (at } x \text{ within } S)$

unfolding *differentiable_def*

by (meson *diff_chain_within*)

6.17.8 Uniqueness of derivative

The general result is a bit messy because we need approachability of the limit point from any direction. But OK for nontrivial intervals etc.

proposition *frechet_derivative_unique_within*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{real_normed_vector}$

assumes 1: $(f \text{ has_derivative } f') \text{ (at } x \text{ within } S)$

and 2: $(f \text{ has_derivative } f'') \text{ (at } x \text{ within } S)$

and $S: \bigwedge i \ e. \llbracket i \in \text{Basis}; e > 0 \rrbracket \implies \exists d. 0 < |d| \wedge |d| < e \wedge (x + d *_{\mathbb{R}} i) \in S$

shows $f' = f''$

```

proof –
  note as = assms(1,2)[unfolded has_derivative_def]
  then interpret f': bounded_linear f' by auto
  from as interpret f'': bounded_linear f'' by auto
  have x islimpt S unfolding islimpt_approachable
  proof (intro allI impI)
    fix e :: real
    assume e > 0
    obtain d where 0 < |d| and |d| < e and x + d *R (SOME i. i ∈ Basis) ∈ S
      using assms(3) SOME_Basis <e>0> by blast
    then show  $\exists x' \in S. x' \neq x \wedge \text{dist } x' x < e$ 
      by (rule_tac x=x + d *R (SOME i. i ∈ Basis) in bxI) (auto simp: dist_norm
SOME_Basis nonzero_Basis) qed
    then have *: netlimit (at x within S) = x
      by (simp add: Lim_ident_at trivial_limit_within)
    show ?thesis
    proof (rule linear_eq_stdbasis)
      show linear f' linear f''
        unfolding linear_conv_bounded_linear using as by auto
    next
      fix i :: 'a'
      assume i: i ∈ Basis
      define e where e = norm (f' i - f'' i)
      show f' i = f'' i
      proof (rule ccontr)
        assume f' i ≠ f'' i
        then have e > 0
          unfolding e_def by auto
        obtain d where d:
          0 < d
           $(\bigwedge y. y \in S \longrightarrow 0 < \text{dist } y x \wedge \text{dist } y x < d \longrightarrow$ 
             $\text{dist } ((f y - f x - f' (y - x)) /_R \text{norm } (y - x) -$ 
             $(f y - f x - f'' (y - x)) /_R \text{norm } (y - x)) (0 - 0) < e)$ 
          using tendsto_diff [OF as(1,2)[THEN conjunct2]]
          unfolding * Lim_within
          using <e>0> by blast
        obtain c where c: 0 < |c| |c| < d ∧ x + c *R i ∈ S
          using assms(3) i d(1) by blast
        have *: norm (- ((1 / |c|) *R f' (c *R i)) + (1 / |c|) *R f'' (c *R i)) =
          norm ((1 / |c|) *R (- (f' (c *R i)) + f'' (c *R i)))
          unfolding scaleR_right_distrib by auto
        also have  $\dots = \text{norm } ((1 / |c|) *_{\mathbb{R}} (c *_{\mathbb{R}} (- (f' i) + f'' i)))$ 
          unfolding f'.scaleR f''.scaleR
          unfolding scaleR_right_distrib scaleR_minus_right
          by auto
        also have  $\dots = e$ 
          unfolding e_def
          using c(1)
          using norm_minus_cancel[of f' i - f'' i]

```

```

    by auto
  finally show False
    using c
    using d(2)[of x + c *R i]
    unfolding dist_norm
    unfolding f'.scaleR f''.scaleR f'.add f''.add f'.diff f''.diff
      scaleR_scaleR scaleR_right_diff_distrib scaleR_right_distrib
    using i
    by (auto simp: inverse_eq_divide)
qed
qed
qed

proposition frechet_derivative_unique_within_closed_interval:
  fixes f::'a::euclidean_space  $\Rightarrow$  'b::real_normed_vector
  assumes ab:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    and x:  $x \in \text{cbox } a \ b$ 
    and (f has_derivative f') (at x within cbox a b)
    and (f has_derivative f'') (at x within cbox a b)
  shows f' = f''
proof (rule frechet_derivative_unique_within)
  fix e :: real
  fix i :: 'a
  assume e > 0 and i: i  $\in$  Basis
  then show  $\exists d. 0 < |d| \wedge |d| < e \wedge x + d *_{\mathbb{R}} i \in \text{cbox } a \ b$ 
  proof (cases x·i = a·i)
    case True
      with ab[of i] <e>0 x i show ?thesis
      by (rule_tac x=(min (b·i - a·i) e) / 2 in exI)
        (auto simp add: mem_box field_simps inner_simps inner_Basis)
    next
      case False
        moreover have a · i < x · i
          using False i mem_box(2) x by force
        moreover {
          have a · i * 2 + min (x · i - a · i) e  $\leq$  a · i * 2 + x · i - a · i
            by auto
          also have ... = a · i + x · i
            by auto
          also have ...  $\leq$  2 * (x · i)
            using <a · i < x · i> by auto
          finally have a · i * 2 + min (x · i - a · i) e  $\leq$  x · i * 2
            by auto
        }
        moreover have min (x · i - a · i) e  $\geq$  0
          by (simp add: <0 < e> <a · i < x · i> less_eq_real_def)
        then have x · i * 2  $\leq$  b · i * 2 + min (x · i - a · i) e
          using i mem_box(2) x by force
        ultimately show ?thesis

```

```

using ab[of i] ⟨e>0⟩ x i
by (rule_tac x=- (min (x·i - a·i) e) / 2 in exI)
    (auto simp add: mem_box field_simps inner_simps inner_Basis)
qed
qed (use assms in auto)

```

```

lemma frechet_derivative_unique_within_open_interval:
  fixes f::'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes x: x ∈ box a b
  and f: (f has_derivative f') (at x within box a b) (f has_derivative f'') (at x
  within box a b)
  shows f' = f''
  by (metis at_within_open assms has_derivative_unique open_box)

```

```

lemma frechet_derivative_at:
  (f has_derivative f') (at x) ⇒ f' = frechet_derivative f (at x)
  using differentiable_def frechet_derivative_works has_derivative_unique by blast

```

```

lemma frechet_derivative_compose:
  frechet_derivative (f o g) (at x) = frechet_derivative (f) (at (g x)) o frechet_derivative
  g (at x)
  if g differentiable at x f differentiable at (g x)
  by (metis diff_chain_at frechet_derivative_at frechet_derivative_works that)

```

```

lemma frechet_derivative_within_cbox:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes ∧i. i∈Basis ⇒ a·i < b·i
  and x ∈ cbox a b
  and (f has_derivative f') (at x within cbox a b)
  shows frechet_derivative f (at x within cbox a b) = f'
  using assms
  by (metis Derivative.differentiableI frechet_derivative_unique_within_closed_interval
  frechet_derivative_works)

```

```

lemma frechet_derivative_transform_within_open:
  frechet_derivative f (at x) = frechet_derivative g (at x)
  if f differentiable at x open X x ∈ X ∧x. x ∈ X ⇒ f x = g x
  by (meson frechet_derivative_at frechet_derivative_works has_derivative_transform_within_open
  that)

```

6.17.9 Derivatives of local minima and maxima are zero

```

lemma has_derivative_local_min:
  fixes f :: 'a::real_normed_vector ⇒ real
  assumes deriv: (f has_derivative f') (at x)
  assumes min: eventually (λy. f x ≤ f y) (at x)
  shows f' = (λh. 0)
proof
  fix h :: 'a

```

```

interpret f': bounded_linear f'
  using deriv by (rule has_derivative_bounded_linear)
show f' h = 0
proof (cases h = 0)
  case False
  from min obtain d where d1: 0 < d and d2:  $\forall y \in \text{ball } x \ d. f \ x \leq f \ y$ 
  unfolding eventually_at by (force simp: dist_commute)
  have FDERIV ( $\lambda r. x + r *_{\mathbb{R}} h$ ) 0 :> ( $\lambda r. r *_{\mathbb{R}} h$ )
  by (intro derivative_eq_intros) auto
  then have FDERIV ( $\lambda r. f (x + r *_{\mathbb{R}} h)$ ) 0 :> ( $\lambda k. f' (k *_{\mathbb{R}} h)$ )
  by (rule has_derivative_compose, simp add: deriv)
  then have DERIV ( $\lambda r. f (x + r *_{\mathbb{R}} h)$ ) 0 :> f' h
  unfolding has_field_derivative_def by (simp add: f'.scaleR_mult_commute_abs)
  moreover have 0 < d / norm h using d1 and  $\langle h \neq 0 \rangle$  by simp
  moreover have  $\forall y. |0 - y| < d / \text{norm } h \longrightarrow f (x + 0 *_{\mathbb{R}} h) \leq f (x + y *_{\mathbb{R}} h)$ 
  h)
  using  $\langle h \neq 0 \rangle$  by (auto simp add: d2 dist_norm pos_less_divide_eq)
  ultimately show f' h = 0
  by (rule DERIV_local_min)
qed simp
qed

```

```

lemma has_derivative_local_max:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  real
  assumes (f has_derivative f') (at x)
  assumes eventually ( $\lambda y. f \ y \leq f \ x$ ) (at x)
  shows f' = ( $\lambda h. 0$ )
  using has_derivative_local_min [of  $\lambda x. - f \ x \ \lambda h. - f' \ h \ x$ ]
  using assms unfolding fun_eq_iff by simp

```

```

lemma differential_zero_maxmin:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  real
  assumes x  $\in S$ 
  and open S
  and deriv: (f has_derivative f') (at x)
  and mono: ( $\forall y \in S. f \ y \leq f \ x$ )  $\vee$  ( $\forall y \in S. f \ x \leq f \ y$ )
  shows f' = ( $\lambda v. 0$ )
  using mono
proof
  assume  $\forall y \in S. f \ y \leq f \ x$ 
  with  $\langle x \in S \rangle$  and  $\langle \text{open } S \rangle$  have eventually ( $\lambda y. f \ y \leq f \ x$ ) (at x)
  unfolding eventually_at_topological by auto
  with deriv show ?thesis
  by (rule has_derivative_local_max)
next
  assume  $\forall y \in S. f \ x \leq f \ y$ 
  with  $\langle x \in S \rangle$  and  $\langle \text{open } S \rangle$  have eventually ( $\lambda y. f \ x \leq f \ y$ ) (at x)
  unfolding eventually_at_topological by auto
  with deriv show ?thesis

```

by (rule has_derivative_local_min)
qed

lemma differential_zero_maxmin_component:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes $k: k \in \text{Basis}$
and ball: $0 < e (\forall y \in \text{ball } x \ e. (f \ y) \cdot k \leq (f \ x) \cdot k) \vee (\forall y \in \text{ball } x \ e. (f \ x) \cdot k \leq (f \ y) \cdot k)$
and diff: $f \text{ differentiable } (at \ x)$
shows $(\sum_{j \in \text{Basis}} (\text{frechet_derivative } f \ (at \ x) \ j \cdot k) *_{\mathbb{R}} j) = (0::'a)$ **(is ?D k = 0)**
proof –
let $?f' = \text{frechet_derivative } f \ (at \ x)$
have $x \in \text{ball } x \ e$ **using** $\langle 0 < e \rangle$ **by** *simp*
moreover have $\text{open } (\text{ball } x \ e)$ **by** *simp*
moreover have $(\lambda x. f \ x \cdot k) \text{ has_derivative } (\lambda h. ?f' \ h \cdot k) \ (at \ x)$
using *bounded_linear_inner_left diff[unfolded frechet_derivative_works]*
by (rule *bounded_linear.has_derivative*)
ultimately have $(\lambda h. \text{frechet_derivative } f \ (at \ x) \ h \cdot k) = (\lambda v. 0)$
using *ball(2)* **by** (rule *differential_zero_maxmin*)
then show *?thesis*
unfolding *fun_eq_iff* **by** *simp*
qed

6.17.10 One-dimensional mean value theorem

lemma mvt_simple:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $a < b$
and derf: $\bigwedge x. [a \leq x; x \leq b] \implies (f \text{ has_derivative } f' \ x) \ (at \ x \ \text{within } \{a..b\})$
shows $\exists x \in \{a <..<b\}. f \ b - f \ a = f' \ x \ (b - a)$
proof (rule *mvt*)
have $f \text{ differentiable_on } \{a..b\}$
using *derf* **unfolding** *differentiable_on_def differentiable_def* **by** *force*
then show $\text{continuous_on } \{a..b\} \ f$
by (rule *differentiable_imp_continuous_on*)
show $(f \text{ has_derivative } f' \ x) \ (at \ x)$ **if** $a < x < b$ **for** x
by (metis *at_within_Icc_at derf leI order.asym that*)
qed (use *assms in auto*)

lemma mvt_very_simple:
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $a \leq b$
and derf: $\bigwedge x. [a \leq x; x \leq b] \implies (f \text{ has_derivative } f' \ x) \ (at \ x \ \text{within } \{a..b\})$
shows $\exists x \in \{a..b\}. f \ b - f \ a = f' \ x \ (b - a)$
proof (*cases a = b*)
interpret *bounded_linear* $f' \ b$
using *assms* **by** *auto*
case *True*


```

then show ?thesis
  by force
next
  case False
  then show ?thesis
    using mvt_simple[OF _ derf]
    by (metis ‹a ≤ b› atLeastAtMost_iff dual_order.order_iff_strict greaterThanLessThan_iff)
qed

```

A nice generalization (see Havin's proof of 5.19 from Rudin's book).

```

lemma mvt_general:
  fixes f :: real ⇒ 'a::real_inner
  assumes a < b
    and contf: continuous_on {a..b} f
    and derf: ∧x. ‹a < x; x < b› ⇒ (f has_derivative f' x) (at x)
  shows ∃x∈{a<..b}. norm (f b - f a) ≤ norm (f' x (b - a))
proof -
  have ∃x∈{a<..b}. (f b - f a) · f b - (f b - f a) · f a = (f b - f a) · f' x (b -
a)
    apply (rule mvt [OF ‹a < b›, where f = λx. (f b - f a) · f x])
    apply (intro continuous_intros contf)
    using derf apply (auto intro: has_derivative_inner_right)
    done
  then obtain x where x: x ∈ {a<..b}
    (f b - f a) · f b - (f b - f a) · f a = (f b - f a) · f' x (b - a) ..
  show ?thesis
proof (cases f a = f b)
  case False
  have norm (f b - f a) * norm (f b - f a) = (norm (f b - f a))2
    by (simp add: power2_eq_square)
  also have ... = (f b - f a) · (f b - f a)
    unfolding power2_norm_eq_inner ..
  also have ... = (f b - f a) · f' x (b - a)
    using x(2) by (simp only: inner_diff_right)
  also have ... ≤ norm (f b - f a) * norm (f' x (b - a))
    by (rule norm_cauchy_schwarz)
  finally show ?thesis
    using False x(1)
    by (auto simp add: mult_left_cancel)
next
  case True
  then show ?thesis
    using ‹a < b› by (rule_tac x=(a + b) / 2 in bexI) auto
qed
qed

```

6.17.11 More general bound theorems

proposition differentiable_bound_general:

```

fixes  $f :: \text{real} \Rightarrow 'a::\text{real\_normed\_vector}$ 
assumes  $a < b$ 
  and  $f\_cont: \text{continuous\_on } \{a..b\} f$ 
  and  $\phi\_cont: \text{continuous\_on } \{a..b\} \phi$ 
  and  $f': \bigwedge x. a < x \implies x < b \implies (f \text{ has\_vector\_derivative } f' x) (at x)$ 
  and  $\phi': \bigwedge x. a < x \implies x < b \implies (\phi \text{ has\_vector\_derivative } \phi' x) (at x)$ 
  and  $bnd: \bigwedge x. a < x \implies x < b \implies \text{norm } (f' x) \leq \phi' x$ 
shows  $\text{norm } (f b - f a) \leq \phi b - \phi a$ 
proof -
{
  fix  $x$  assume  $x: a < x < b$ 
  have  $0 \leq \text{norm } (f' x)$  by simp
  also have  $\dots \leq \phi' x$  using  $x$  by (auto intro!:  $bnd$ )
  finally have  $0 \leq \phi' x$  .
} note  $\phi'_{nonneg} = \text{this}$ 
note  $f\_tendsto = \text{assms}(2)[\text{simplified\_continuous\_on\_def}, \text{rule\_format}]$ 
note  $\phi\_tendsto = \text{assms}(3)[\text{simplified\_continuous\_on\_def}, \text{rule\_format}]$ 
{
  fix  $e::\text{real}$  assume  $e > 0$ 
  define  $e2$  where  $e2 = e / 2$ 
  with  $\langle e > 0 \rangle$  have  $e2 > 0$  by simp
  let  $?le = \lambda x1. \text{norm } (f x1 - f a) \leq \phi x1 - \phi a + e * (x1 - a) + e$ 
  define  $A$  where  $A = \{x2. a \leq x2 \wedge x2 \leq b \wedge (\forall x1 \in \{a .. < x2\}. ?le x1)\}$ 
  have  $A\_subset: A \subseteq \{a..b\}$  by (auto simp: A_def)
  {
    fix  $x2$ 
    assume  $a: a \leq x2$   $x2 \leq b$  and  $le: \forall x1 \in \{a..<x2\}. ?le x1$ 
    have  $?le x2$  using  $\langle e > 0 \rangle$ 
    proof cases
      assume  $x2 \neq a$  with  $a$  have  $a < x2$  by simp
      have at  $x2$  within  $\{a <..<x2\} \neq \text{bot}$ 
        using  $\langle a < x2 \rangle$ 
        by (auto simp: trivial\_limit\_within islimpt\_in\_closure)
      moreover
      have  $((\lambda x1. (\phi x1 - \phi a) + e * (x1 - a) + e) \longrightarrow (\phi x2 - \phi a) + e * (x2 - a) + e)$  (at  $x2$  within  $\{a <..<x2\}$ )
         $((\lambda x1. \text{norm } (f x1 - f a)) \longrightarrow \text{norm } (f x2 - f a))$  (at  $x2$  within  $\{a <..<x2\}$ )
        using  $a$ 
        by (auto intro!: tendsto_eq_intros f_tendsto phi_tendsto intro: tendsto_within_subset[where S={a..b}])
      moreover
      have eventually  $(\lambda x. x > a)$  (at  $x2$  within  $\{a <..<x2\}$ )
        by (auto simp: eventually_at_filter)
      hence eventually  $?le$  (at  $x2$  within  $\{a <..<x2\}$ )
        unfolding eventually_at_filter
        by eventually_elim (insert le, auto)
      ultimately
      show  $?thesis$ 
    }
  }
}

```

```

    by (rule tendsto_le)
  qed simp
} note le_cont = this
have a ∈ A
  using assms by (auto simp: A_def)
hence [simp]: A ≠ {} by auto
have A_ivl:  $\bigwedge x1 x2. x2 \in A \implies x1 \in \{a ..x2\} \implies x1 \in A$ 
  by (simp add: A_def)
have [simp]: bdd_above A by (auto simp: A_def)
define y where y = Sup A
have y ≤ b
  unfolding y_def
  by (simp add: cSup_le_iff) (simp add: A_def)
have leI:  $\bigwedge x x1. a \leq x1 \implies x \in A \implies x1 < x \implies ?le x1$ 
  by (auto simp: A_def intro!: le_cont)
have y_all_le:  $\forall x1 \in \{a ..<y\}. ?le x1$ 
  by (auto simp: y_def less_cSup_iff leI)
have a ≤ y
  by (metis ⟨a ∈ A⟩ ⟨bdd_above A⟩ cSup_upper y_def)
have y ∈ A
  using y_all_le ⟨a ≤ y⟩ ⟨y ≤ b⟩
  by (auto simp: A_def)
hence A = {a .. y}
  using A_subset by (auto simp: subset_iff y_def cSup_upper intro: A_ivl)
from le_cont[OF ⟨a ≤ y⟩ ⟨y ≤ b⟩ y_all_le] have le_y: ?le y .
have y = b
proof (cases a = y)
case True
  with ⟨a < b⟩ have y < b by simp
  with ⟨a = y⟩ f_cont phi_cont ⟨e2 > 0⟩
  have 1:  $\forall_F x \text{ in } \text{at } y \text{ within } \{y..b\}. \text{dist } (f x) (f y) < e2$ 
  and 2:  $\forall_F x \text{ in } \text{at } y \text{ within } \{y..b\}. \text{dist } (\varphi x) (\varphi y) < e2$ 
  by (auto simp: continuous_on_def tendsto_iff)
  have 3: eventually (λx. y < x) (at y within {y..b})
  by (auto simp: eventually_at_filter)
  have 4: eventually (λx::real. x < b) (at y within {y..b})
  using _ ⟨y < b⟩
  by (rule order_tendstoD) (auto intro!: tendsto_eq_intros)
  from 1 2 3 4
  have eventually_le: eventually (λx. ?le x) (at y within {y .. b})
proof eventually_elim
case (elim x1)
  have norm (f x1 - f a) = norm (f x1 - f y)
  by (simp add: ⟨a = y⟩)
  also have norm (f x1 - f y) ≤ e2
  using elim ⟨a = y⟩ by (auto simp: dist_norm intro!: less_imp_le)
  also have ... ≤ e2 + (φ x1 - φ a + e2 + e * (x1 - a))
  using ⟨0 < e⟩ elim
  by (intro add_increasing2[OF add_nonneg_nonneg order.refl])

```

```

      (auto simp: ⟨a = y⟩ dist_norm intro!: mult_nonneg_nonneg)
    also have ... = φ x1 - φ a + e * (x1 - a) + e
      by (simp add: e2_def)
    finally show ?le x1 .
qed
from this[unfolded eventually_at_topological] ⟨?le y⟩
obtain S where S: open S y ∈ S ∧ x. x ∈ S ⇒ x ∈ {y..b} ⇒ ?le x
  by metis
from ⟨open S⟩ obtain d where d: ∧x. dist x y < d ⇒ x ∈ S d > 0
  by (force simp: dist_commute open_dist ball_def dest!: bspec[OF _ ⟨y ∈
S⟩])
define d' where d' = min b (y + (d/2))
have d' ∈ A
  unfolding A_def
proof safe
  show a ≤ d' using ⟨a = y⟩ ⟨0 < d⟩ ⟨y < b⟩ by (simp add: d'_def)
  show d' ≤ b by (simp add: d'_def)
  fix x1
  assume x1 ∈ {a..

```

```

    by (auto simp: eventually_at_filter)
  ultimately
  have  $\forall_F x1 \text{ in } ?F. \text{norm } (f x1 - f y) \leq (\varphi x1 - \varphi y) + e * |x1 - y|$ 
    (is  $\forall_F x1 \text{ in } ?F. ?le' x1$ )
  proof eventually_elim
    case (elim x1)
    from norm_triangle_ineq2[THEN order_trans, OF elim(1)]
    have  $\text{norm } (f x1 - f y) \leq \text{norm } (f' y) * |x1 - y| + e2 * |x1 - y|$ 
      by (simp add: ac_simps)
    also have  $\text{norm } (f' y) \leq \varphi' y$  using bnd  $\langle a < y \rangle \langle y < b \rangle$  by simp
    also have  $\varphi' y * |x1 - y| \leq \varphi x1 - \varphi y + e2 * |x1 - y|$ 
      using elim by (simp add: ac_simps)
    finally
    have  $\text{norm } (f x1 - f y) \leq \varphi x1 - \varphi y + e2 * |x1 - y| + e2 * |x1 - y|$ 
      by (auto simp: mult_right_mono)
    thus ?case by (simp add: e2_def)
  qed
  moreover have ?le' y by simp
  ultimately obtain S
  where  $S: \text{open } S \ y \in S \ \wedge \ x. x \in S \implies x \in \{y..<b\} \implies ?le' x$ 
    unfolding eventually_at_topological
    by metis
  from  $\langle \text{open } S \rangle$  obtain d where  $d: \wedge x. \text{dist } x \ y < d \implies x \in S \ d > 0$ 
    by (force simp: dist_commute open_dist ball_def dest!: bspec[OF  $\langle y \in S \rangle$ ])
  define d' where  $d' = \min ((y + b)/2) (y + (d/2))$ 
  have  $d' \in A$ 
    unfolding A_def
  proof safe
    show  $a \leq d'$  using  $\langle a < y \rangle \langle 0 < d \rangle \langle y < b \rangle$  by (simp add: d'_def)
    show  $d' \leq b$  using  $\langle y < b \rangle$  by (simp add: d'_def min_def)
    fix x1
    assume  $x1: x1 \in \{a..<d'\}$ 
    show ?le x1
    proof (cases  $x1 < y$ )
      case True
      then show ?thesis
        using  $\langle y \in A \rangle \text{local.leI } x1$  by auto
    next
      case False
      hence  $x1': x1 \in S \ x1 \in \{y..<b\}$  using x1
        by (auto simp: d'_def dist_real_def intro!: d)
      have  $\text{norm } (f x1 - f a) \leq \text{norm } (f x1 - f y) + \text{norm } (f y - f a)$ 
        by (rule order_trans[OF norm_triangle_ineq]) simp
      also note  $S(3)[OF x1']$ 
      also note le_y
      finally show ?le x1
        using False by (auto simp: algebra_simps)
    qed
  qed

```

```

    qed
  hence  $d' \leq y$ 
    unfolding  $y\_def$  by (rule  $cSup\_upper$ ) simp
  thus False using  $\langle d > 0 \rangle \langle y < b \rangle$ 
    by (simp add:  $d'\_def$   $min\_def$   $split: if\_split\_asm$ )
  qed
  qed
  with  $le\_y$  have  $norm (f b - f a) \leq \varphi b - \varphi a + e * (b - a + 1)$ 
    by (simp add:  $algebra\_simps$ )
} note * = this
show ?thesis
proof (rule  $field\_le\_epsilon$ )
  fix  $e::real$  assume  $e > 0$ 
  then show  $norm (f b - f a) \leq \varphi b - \varphi a + e$ 
    using  $*[of e / (b - a + 1)] \langle a < b \rangle$  by simp
  qed
  qed
lemma differentiable_bound:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes convex  $S$ 
    and  $derf: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x)$  (at  $x$  within  $S$ )
    and  $B: \bigwedge x. x \in S \implies onorm (f' x) \leq B$ 
    and  $x: x \in S$ 
    and  $y: y \in S$ 
  shows  $norm (f x - f y) \leq B * norm (x - y)$ 
proof -
  let ?p =  $\lambda u. x + u *_R (y - x)$ 
  let ? $\varphi$  =  $\lambda h. h * B * norm (x - y)$ 
  have *:  $x + u *_R (y - x) \in S$  if  $u \in \{0..1\}$  for  $u$ 
  proof -
    have  $u *_R y = u *_R (y - x) + u *_R x$ 
    by (simp add:  $scale\_right\_diff\_distrib$ )
    then show  $x + u *_R (y - x) \in S$ 
    using that  $\langle convex\ S \rangle x\ y$  by (simp add:  $convex\_alt$ )
      (metis  $pth\_b(2)$   $pth\_c(1)$   $scaleR\_collapse$ )
  qed
  have  $\bigwedge z. z \in (\lambda u. x + u *_R (y - x)) \langle \{0..1\} \implies$ 
    ( $f \text{ has\_derivative } f' z$ ) (at  $z$  within  $(\lambda u. x + u *_R (y - x)) \langle \{0..1\}$ )
    by (auto intro: *  $has\_derivative\_subset$  [OF  $derf$ ])
  then have  $continuous\_on$  ( $?p \langle \{0..1\}$ )  $f$ 
    unfolding  $continuous\_on\_eq\_continuous\_within$ 
    by (meson  $has\_derivative\_continuous$ )
  with * have  $1: continuous\_on \{0 .. 1\} (f \circ ?p)$ 
    by (intro  $continuous\_intros$ )+
  {
    fix  $u::real$  assume  $u: u \in \{0 <..< 1\}$ 
    let ?u = ?p  $u$ 
    interpret linear ( $f' ?u$ )

```

```

    using u by (auto intro!: has_derivative_linear derf *)
  have (f ∘ ?p has_derivative (f' ?u) ∘ (λu. 0 + u *R (y - x))) (at u within box
0 1)
    by (intro derivative_intros has_derivative_subset [OF derf]) (use u * in auto)
  hence ((f ∘ ?p) has_vector_derivative f' ?u (y - x)) (at u)
    by (simp add: at_within_open[OF u open_greaterThanLessThan] scaleR
has_vector_derivative_def o_def)
} note 2 = this
have 3: continuous_on {0..1} ?φ
  by (rule continuous_intros)+
have 4: (?φ has_vector_derivative B * norm (x - y)) (at u) for u
  by (auto simp: has_vector_derivative_def intro!: derivative_eq_intros)
{
  fix u::real assume u: u ∈ {0 <..< 1}
  let ?u = ?p u
  interpret bounded_linear (f' ?u)
    using u by (auto intro!: has_derivative_bounded_linear derf *)
  have norm (f' ?u (y - x)) ≤ onorm (f' ?u) * norm (y - x)
    by (rule onorm) (rule bounded_linear)
  also have onorm (f' ?u) ≤ B
    using u by (auto intro!: asms(3)[rule_format] *)
  finally have norm ((f' ?u) (y - x)) ≤ B * norm (x - y)
    by (simp add: mult_right_mono norm_minus_commute)
} note 5 = this
have norm (f x - f y) = norm ((f ∘ (λu. x + u *R (y - x))) 1 - (f ∘ (λu. x
+ u *R (y - x))) 0)
  by (auto simp add: norm_minus_commute)
also
from differentiable_bound_general[OF zero_less_one 1, OF 3 2 4 5]
have norm ((f ∘ ?p) 1 - (f ∘ ?p) 0) ≤ B * norm (x - y)
  by simp
finally show ?thesis .
qed

```

lemma field_differentiable_bound:

```

fixes S :: 'a::real_normed_field set
assumes cvs: convex S
  and df:  $\bigwedge z. z \in S \implies (f \text{ has\_field\_derivative } f' z) \text{ (at } z \text{ within } S)$ 
  and dn:  $\bigwedge z. z \in S \implies \text{norm } (f' z) \leq B$ 
  and x ∈ S y ∈ S
shows norm(f x - f y) ≤ B * norm(x - y)
proof (rule differentiable_bound [OF cvs])
  show  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } (*) (f' x)) \text{ (at } x \text{ within } S)$ 
    by (simp add: df has_field_derivative_imp_has_derivative)
  show  $\bigwedge x. x \in S \implies \text{onorm } ((*) (f' x)) \leq B$ 
    by (metis (no_types, opaque_lifting) dn norm_mult onorm_le order_refl or-
der_trans)
qed (use asms in auto)

```

lemma

differentiable_bound_segment:

fixes $f::'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$

assumes $\bigwedge t. t \in \{0..1\} \Longrightarrow x0 + t *_R a \in G$

assumes $f': \bigwedge x. x \in G \Longrightarrow (f \text{ has_derivative } f' x)$ (at x within G)

assumes $B: \bigwedge x. x \in \{0..1\} \Longrightarrow \text{onorm } (f' (x0 + x *_R a)) \leq B$

shows $\text{norm } (f (x0 + a) - f x0) \leq \text{norm } a * B$

proof –

let $?G = (\lambda x. x0 + x *_R a) \text{ ` } \{0..1\}$

have $?G = (+) x0 \text{ ` } (\lambda x. x *_R a) \text{ ` } \{0..1\}$ **by** *auto*

also have *convex* ...

by (*intro convex_translation convex_scaled convex_real_interval*)

finally have *convex* $?G$.

moreover have $?G \subseteq G$ $x0 \in ?G$ $x0 + a \in ?G$ **using** *assms* **by** (*auto intro: image_eqI[where x=1]*)

ultimately show *?thesis*

using *has_derivative_subset[OF f' `?G ⊆ G]* B

differentiable_bound[of $(\lambda x. x0 + x *_R a) \text{ ` } \{0..1\}$] $f f' B x0 + a x0$]

by (*force simp: ac_simps*)

qed

lemma *differentiable_bound_linearization:*

fixes $f::'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$

assumes $S: \bigwedge t. t \in \{0..1\} \Longrightarrow a + t *_R (b - a) \in S$

assumes $f'[derivative_intros]: \bigwedge x. x \in S \Longrightarrow (f \text{ has_derivative } f' x)$ (at x within S)

assumes $B: \bigwedge x. x \in S \Longrightarrow \text{onorm } (f' x - f' x0) \leq B$

assumes $x0 \in S$

shows $\text{norm } (f b - f a - f' x0 (b - a)) \leq \text{norm } (b - a) * B$

proof –

define g **where** [*abs_def*]: $g x = f x - f' x0 x$ **for** x

have $g: \bigwedge x. x \in S \Longrightarrow (g \text{ has_derivative } (\lambda i. f' x i - f' x0 i))$ (at x within S)

unfolding g_def **using** *assms*

by (*auto intro!: derivative_eq_intros*)

bounded_linear.has_derivative[*OF has_derivative_bounded_linear, OF f'*]

from B **have** $\forall x \in \{0..1\}. \text{onorm } (\lambda i. f' (a + x *_R (b - a)) i - f' x0 i) \leq B$

using *assms* **by** (*auto simp: fun_diff_def*)

with *differentiable_bound_segment*[*OF S g*] $\langle x0 \in S \rangle$

show *?thesis*

by (*simp add: g_def field_simps linear_diff*[*OF has_derivative_linear*[*OF f'*]])

qed

lemma *vector_differentiable_bound_linearization:*

fixes $f::\text{real} \Rightarrow 'b::\text{real_normed_vector}$

assumes $f': \bigwedge x. x \in S \Longrightarrow (f \text{ has_vector_derivative } f' x)$ (at x within S)

assumes *closed_segment* $a b \subseteq S$

assumes $B: \bigwedge x. x \in S \Longrightarrow \text{norm } (f' x - f' x0) \leq B$

assumes $x0 \in S$

shows $\text{norm } (f b - f a - (b - a) *_R f' x0) \leq \text{norm } (b - a) * B$


```

using assms
by (intro differentiable_bound_linearization[of a b S f  $\lambda x h. h *_{\mathbb{R}} f' x x0 B$ ])
    (force simp: closed_segment_real_eq has_vector_derivative_def
      scaleR_diff_right[symmetric] mult.commute[of B]
      intro!: onorm_le mult_left_mono)+

```

In particular.

```

lemma has_derivative_zero_constant:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes convex s
  and  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
  shows  $\exists c. \forall x \in s. f x = c$ 
proof -
  { fix  $x y$  assume  $x \in s \ y \in s$ 
    then have  $\text{norm } (f x - f y) \leq 0 * \text{norm } (x - y)$ 
    using assms by (intro differentiable_bound[of s]) (auto simp: onorm_zero)
    then have  $f x = f y$ 
    by simp }
  then show ?thesis
  by metis
qed

```

```

lemma has_field_derivative_zero_constant:
  assumes convex s  $\bigwedge x. x \in s \implies (f \text{ has\_field\_derivative } 0) \text{ (at } x \text{ within } s)$ 
  shows  $\exists c. \forall x \in s. f(x) = (c :: 'a :: \text{real\_normed\_field})$ 
proof (rule has_derivative_zero_constant)
  have  $A: (*) \ 0 = (\lambda \_. 0 :: 'a) \text{ by (intro ext) simp}$ 
  fix  $x$  assume  $x \in s$  thus  $(f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
  using assms(2)[of x] by (simp add: has_field_derivative_def A)
qed fact

```

```

lemma
  has_vector_derivative_zero_constant:
  assumes convex s
  assumes  $\bigwedge x. x \in s \implies (f \text{ has\_vector\_derivative } 0) \text{ (at } x \text{ within } s)$ 
  obtains  $c$  where  $\bigwedge x. x \in s \implies f x = c$ 
  using has_derivative_zero_constant[of s f] assms
  by (auto simp: has_vector_derivative_def)

```

```

lemma has_derivative_zero_unique:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes convex s
  and  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda h. 0)) \text{ (at } x \text{ within } s)$ 
  and  $x \in s \ y \in s$ 
  shows  $f x = f y$ 
  using has_derivative_zero_constant[OF assms(1,2)] assms(3-) by force

```

```

lemma has_derivative_zero_unique_connected:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow 'b::\text{real\_normed\_vector}$ 

```

```

assumes open s connected s
assumes f:  $\bigwedge x. x \in s \implies (f \text{ has\_derivative } (\lambda x. 0))$  (at s)
assumes x ∈ s y ∈ s
shows f x = f y
proof (rule connected_local_const[where f=f, OF ‹connected s› ‹x∈s› ‹y∈s›])
show  $\forall a \in s. \text{eventually } (\lambda b. f a = f b)$  (at a within s)
proof
  fix a assume a ∈ s
  with ‹open s› obtain e where 0 < e ball a e ⊆ s
  by (rule openE)
  then have  $\exists c. \forall x \in \text{ball } a \ e. f x = c$ 
  by (intro has_derivative_zero_constant)
  (auto simp: at_within_open[OF _ open_ball] f)
  with ‹0 < e› have  $\forall x \in \text{ball } a \ e. f a = f x$ 
  by auto
  then show eventually  $(\lambda b. f a = f b)$  (at a within s)
  using ‹0 < e› unfolding eventually_at_topological
  by (intro exI[of _ ball a e]) auto
qed
qed

```

6.17.12 Differentiability of inverse function (most basic form)

```

lemma has_derivative_inverse_basic:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes derf: (f has_derivative f') (at (g y))
  and ling': bounded_linear g'
  and g' o f' = id
  and contg: continuous (at y) g
  and open T
  and y ∈ T
  and fg:  $\bigwedge z. z \in T \implies f (g z) = z$ 
  shows (g has_derivative g') (at y)
proof –
  interpret f': bounded_linear f'
  using assms unfolding has_derivative_def by auto
  interpret g': bounded_linear g'
  using assms by auto
  obtain C where C: 0 < C  $\bigwedge x. \text{norm } (g' x) \leq \text{norm } x * C$ 
  using bounded_linear.pos_bounded[OF assms(2)] by blast
  have lem1:  $\forall e > 0. \exists d > 0. \forall z. \text{norm } (z - y) < d \implies \text{norm } (g z - g y - g'(z - y)) \leq e * \text{norm } (g z - g y)$ 
proof (intro allI impI)
  fix e :: real
  assume e > 0
  with C(1) have *: e / C > 0 by auto
  obtain d0 where 0 < d0 and d0:
     $\bigwedge u. \text{norm } (u - g y) < d0 \implies \text{norm } (f u - f (g y) - f' (u - g y)) \leq e / C * \text{norm } (u - g y)$ 

```

```

    using derf * unfolding has_derivative_at_alt by blast
  obtain d1 where 0 < d1 and d1:  $\bigwedge x. \llbracket 0 < \text{dist } x \ y; \text{dist } x \ y < d1 \rrbracket \implies \text{dist}$ 
  (g x) (g y) < d0
    using contg <0 < d0> unfolding continuous_at_Lim_at by blast
  obtain d2 where 0 < d2 and d2:  $\bigwedge u. \text{dist } u \ y < d2 \implies u \in T$ 
    using <open T> <y ∈ T> unfolding open_dist by blast
  obtain d where d: 0 < d d < d1 d < d2
    using field_lbound_gt_zero[OF <0 < d1> <0 < d2>] by blast
  show  $\exists d > 0. \forall z. \text{norm } (z - y) < d \longrightarrow \text{norm } (g z - g y - g'(z - y)) \leq e * \text{norm } (g z - g y)$ 
  proof (intro exI allI impI conjI)
    fix z
    assume as:  $\text{norm } (z - y) < d$ 
    then have z ∈ T
      using d2 d unfolding dist_norm by auto
    have  $\text{norm } (g z - g y - g'(z - y)) \leq \text{norm } (g'(f(g z) - y - f'(g z - g y)))$ 
      unfolding g'.diff f'.diff
      unfolding assms(3)[unfolded o_def id_def, THEN fun_cong] fg[OF <z ∈ T>]
      by (simp add: norm_minus_commute)
    also have  $\dots \leq \text{norm } (f(g z) - y - f'(g z - g y)) * C$ 
      by (rule C(2))
    also have  $\dots \leq (e / C) * \text{norm } (g z - g y) * C$ 
    proof -
      have  $\text{norm } (g z - g y) < d0$ 
        by (metis as cancel_comm_monoid_add_class.diff_cancel d(2) <0 < d0>
d1 diff_gt_0_iff_gt diff_strict_mono dist_norm dist_self_zero_less_dist_iff)
      then show ?thesis
        by (metis C(1) <y ∈ T> d0 fg mult_le_cancel_right_pos)
    qed
    also have  $\dots \leq e * \text{norm } (g z - g y)$ 
      using C by (auto simp add: field_simps)
    finally show  $\text{norm } (g z - g y - g'(z - y)) \leq e * \text{norm } (g z - g y)$ 
      by simp
  qed (use d in auto)
qed
have *:  $(0::\text{real}) < 1 / 2$ 
  by auto
obtain d where 0 < d and d:
   $\bigwedge z. \text{norm } (z - y) < d \implies \text{norm } (g z - g y - g'(z - y)) \leq 1/2 * \text{norm } (g z - g y)$ 
  using lem1 * by blast
define B where B = C * 2
have B > 0
  unfolding B_def using C by auto
have lem2:  $\text{norm } (g z - g y) \leq B * \text{norm } (z - y)$  if z:  $\text{norm}(z - y) < d$  for z
proof -
  have  $\text{norm } (g z - g y) \leq \text{norm}(g'(z - y)) + \text{norm}((g z - g y) - g'(z - y))$ 
    by (rule norm_triangle_sub)

```

```

also have ...  $\leq \text{norm } (g' (z - y)) + 1 / 2 * \text{norm } (g z - g y)$ 
  by (rule add_left_mono) (use d z in auto)
also have ...  $\leq \text{norm } (z - y) * C + 1 / 2 * \text{norm } (g z - g y)$ 
  by (rule add_right_mono) (use C in auto)
finally show  $\text{norm } (g z - g y) \leq B * \text{norm } (z - y)$ 
  unfolding B_def
  by (auto simp add: field_simps)
qed
show ?thesis
  unfolding has_derivative_at_alt
proof (intro conjI assms allI impI)
  fix e :: real
  assume e > 0
  then have *: e / B > 0 by (metis ‹B > 0› divide_pos_pos)
  obtain d' where 0 < d' and d':
     $\bigwedge z. \text{norm } (z - y) < d' \implies \text{norm } (g z - g y - g' (z - y)) \leq e / B * \text{norm } (g z - g y)$ 
  using lem1 * by blast
  obtain k where k: 0 < k k < d k < d'
  using field_lbound_gt_zero[OF ‹0 < d› ‹0 < d'›] by blast
  show  $\exists d > 0. \forall ya. \text{norm } (ya - y) < d \implies \text{norm } (g ya - g y - g' (ya - y)) \leq e * \text{norm } (ya - y)$ 
proof (intro exI allI impI conjI)
  fix z
  assume as:  $\text{norm } (z - y) < k$ 
  then have  $\text{norm } (g z - g y - g' (z - y)) \leq e / B * \text{norm } (g z - g y)$ 
  using d' k by auto
  also have ...  $\leq e * \text{norm } (z - y)$ 
  unfolding times_divide_eq_left pos_divide_le_eq[OF ‹B > 0›]
  using lem2[of z] k as ‹e > 0›
  by (auto simp add: field_simps)
  finally show  $\text{norm } (g z - g y - g' (z - y)) \leq e * \text{norm } (z - y)$ 
  by simp
qed (use k in auto)
qed
qed

```

Inverse function theorem for complex derivatives

lemma has_field_derivative_inverse_basic:

shows $\text{DERIV } f (g y) :> f' \implies$

$f' \neq 0 \implies$

$\text{continuous } (\text{at } y) g \implies$

$\text{open } t \implies$

$y \in t \implies$

$(\bigwedge z. z \in t \implies f (g z) = z)$

$\implies \text{DERIV } g y :> \text{inverse } (f')$

unfolding has_field_derivative_def

by (rule has_derivative_inverse_basic) (auto simp: bounded_linear_mult_right)

Simply rewrite that based on the domain point x.

```

lemma has_derivative_inverse_basic_x:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes ( $f$  has_derivative  $f'$ ) (at  $x$ )
    and bounded_linear  $g'$ 
    and  $g' \circ f' = id$ 
    and continuous (at  $(f\ x)$ )  $g$ 
    and  $g (f\ x) = x$ 
    and open  $T$ 
    and  $f\ x \in T$ 
    and  $\bigwedge y. y \in T \implies f (g\ y) = y$ 
  shows ( $g$  has_derivative  $g'$ ) (at  $(f\ x)$ )
  by (rule has_derivative_inverse_basic) (use assms in auto)

```

This is the version in Dieudonne', assuming continuity of f and g .

```

lemma has_derivative_inverse_dieudonne:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes open  $S$ 
    and  $fS$ : open  $(f\ 'S)$ 
    and  $A$ : continuous_on  $S$   $f$  continuous_on  $(f\ 'S)$   $g$ 
       $\bigwedge x. x \in S \implies g (f\ x) = x\ x \in S$ 
    and  $B$ : ( $f$  has_derivative  $f'$ ) (at  $x$ ) bounded_linear  $g'$   $g' \circ f' = id$ 
  shows ( $g$  has_derivative  $g'$ ) (at  $(f\ x)$ )
  using  $A$   $fS$  continuous_on_eq_continuous_at
  by (intro has_derivative_inverse_basic_x[OF  $B$  _ _  $fS$ ]) force+

```

Here's the simplest way of not assuming much about g .

```

proposition has_derivative_inverse:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
  assumes compact  $S$ 
    and  $x \in S$ 
    and  $fx$ :  $f\ x \in interior (f\ 'S)$ 
    and continuous_on  $S$   $f$ 
    and  $gf$ :  $\bigwedge y. y \in S \implies g (f\ y) = y$ 
    and  $B$ : ( $f$  has_derivative  $f'$ ) (at  $x$ ) bounded_linear  $g'$   $g' \circ f' = id$ 
  shows ( $g$  has_derivative  $g'$ ) (at  $(f\ x)$ )
  proof -
  have *:  $\bigwedge y. y \in interior (f\ 'S) \implies f (g\ y) = y$ 
    by (metis  $gf$  image_iff interior_subset subsetCE)
  show ?thesis
    using assms * continuous_on_interior continuous_on_inv  $fx$ 
    by (intro has_derivative_inverse_basic_x[OF  $B$ , where  $T = interior (f\ 'S)$ ])
  blast+
  qed

```

Invertible derivative continuous at a point implies local injectivity. It's only for this we need continuity of the derivative, except of course if we want the fact that the inverse derivative is also continuous. So if we know for some other reason that the inverse function exists, it's OK.

```

proposition has_derivative_locally_injective:

```

```

fixes  $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$ 
assumes  $a \in S$ 
  and  $open\ S$ 
  and  $bling: bounded\_linear\ g'$ 
  and  $g' \circ f' a = id$ 
  and  $derf: \bigwedge x. x \in S \implies (f\ has\_derivative\ f'\ x)\ (at\ x)$ 
  and  $\bigwedge e. e > 0 \implies \exists d > 0. \forall x. dist\ a\ x < d \longrightarrow onorm\ (\lambda v. f'\ x\ v - f'\ a\ v)$ 
<  $e$ 
obtains  $r$  where  $r > 0\ ball\ a\ r \subseteq S\ inj\_on\ f\ (ball\ a\ r)$ 
proof -
interpret  $bounded\_linear\ g'$ 
  using  $assms\ by\ auto$ 
note  $f'g' = assms(4)[unfolded\ id\_def\ o\_def, THEN\ cong]$ 
have  $g'\ (f'\ a\ (\sum\ Basis)) = (\sum\ Basis)\ (\sum\ Basis) \neq (0::'n)$ 
  using  $f'g'$  by  $auto$ 
then have  $*$ :  $0 < onorm\ g'$ 
  unfolding  $onorm\_pos\_lt[OF\ assms(3)]$ 
  by  $fastforce$ 
define  $k$  where  $k = 1 / onorm\ g' / 2$ 
have  $*$ :  $k > 0$ 
  unfolding  $k\_def$  using  $*$  by  $auto$ 
obtain  $d1$  where  $d1$ :
   $0 < d1$ 
   $\bigwedge x. dist\ a\ x < d1 \implies onorm\ (\lambda v. f'\ x\ v - f'\ a\ v) < k$ 
  using  $assms(6)\ *\ by\ blast$ 
from  $\langle open\ S \rangle$  obtain  $d2$  where  $d2 > 0\ ball\ a\ d2 \subseteq S$ 
  using  $\langle a \in S \rangle ..$ 
obtain  $d2$  where  $d2: 0 < d2\ ball\ a\ d2 \subseteq S$ 
  using  $\langle 0 < d2 \rangle\ \langle ball\ a\ d2 \subseteq S \rangle$  by  $blast$ 
obtain  $d$  where  $d: 0 < d\ d < d1\ d < d2$ 
  using  $field\_lbound\_gt\_zero[OF\ d1(1)\ d2(1)]$  by  $blast$ 
show  $?thesis$ 
proof
  show  $0 < d$  by  $(fact\ d)$ 
  show  $ball\ a\ d \subseteq S$ 
  using  $\langle d < d2 \rangle\ \langle ball\ a\ d2 \subseteq S \rangle$  by  $auto$ 
  show  $inj\_on\ f\ (ball\ a\ d)$ 
  unfolding  $inj\_on\_def$ 
  proof  $(intro\ strip)$ 
    fix  $x\ y$ 
    assume  $as: x \in ball\ a\ d\ y \in ball\ a\ d\ f\ x = f\ y$ 
    define  $ph$  where  $[abs\_def]: ph\ w = w - g'\ (f\ w - f\ x)$  for  $w$ 
    have  $ph': ph = g' \circ (\lambda w. f'\ a\ w - (f\ w - f\ x))$ 
    unfolding  $ph\_def\ o\_def$  by  $(simp\ add: diff\ f'g')$ 
    have  $norm\ (ph\ x - ph\ y) \leq (1 / 2) * norm\ (x - y)$ 
    proof  $(rule\ differentiable\_bound[OF\ convex\_ball\ \_\_ as(1-2)])$ 
      fix  $u$ 
      assume  $u: u \in ball\ a\ d$ 
      then have  $u \in S$ 

```

```

    using d d2 by auto
  have *:  $(\lambda v. v - g' (f' u v)) = g' \circ (\lambda w. f' a w - f' u w)$ 
    unfolding o_def and diff
    using f'g' by auto
  have blin: bounded_linear (f' a)
    using  $\langle a \in S \rangle$  derf by blast
  show (ph has_derivative  $(\lambda v. v - g' (f' u v))$ ) (at u within ball a d)
    unfolding ph' * comp_def
    by (rule  $\langle u \in S \rangle$  derivative_eq_intros has_derivative_at_withinI [OF
derf] bounded_linear.has_derivative [OF blin] bounded_linear.has_derivative [OF
bling] |simp)+
    have **: bounded_linear  $(\lambda x. f' u x - f' a x)$  bounded_linear  $(\lambda x. f' a x - f' u x)$ 
      using  $\langle u \in S \rangle$  blin bounded_linear_sub derf by auto
    then have onorm  $(\lambda v. v - g' (f' u v)) \leq \text{onorm } g' * \text{onorm } (\lambda w. f' a w - f' u w)$ 
      by (simp add: * bounded_linear_axioms onorm_compose)
    also have  $\dots \leq \text{onorm } g' * k$ 
      apply (rule mult_left_mono)
      using d1(2)[of u]
      using onorm_neg[where  $f = \lambda x. f' u x - f' a x$ ] d u onorm_pos_le[OF
bling]
      apply (auto simp: algebra_simps)
    done
    also have  $\dots \leq 1 / 2$ 
      unfolding k_def by auto
    finally show  $\text{onorm } (\lambda v. v - g' (f' u v)) \leq 1 / 2$  .
  qed
  moreover have  $\text{norm } (ph y - ph x) = \text{norm } (y - x)$ 
    by (simp add: as(3) ph_def)
  ultimately show  $x = y$ 
    unfolding norm_minus_commute by auto
  qed
qed
qed

```

6.17.13 Uniformly convergent sequence of derivatives

lemma has_derivative_sequence_lipschitz_lemma:

fixes $f :: \text{nat} \Rightarrow 'a :: \text{real_normed_vector} \Rightarrow 'b :: \text{real_normed_vector}$

assumes convex S

and derf: $\bigwedge n x. x \in S \implies ((f n) \text{ has_derivative } (f' n x))$ (at x within S)

and nle: $\bigwedge n x h. \llbracket n \geq N; x \in S \rrbracket \implies \text{norm } (f' n x h - g' x h) \leq e * \text{norm } h$

and $0 \leq e$

shows $\forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } ((f m x - f n x) - (f m y - f n y)) \leq 2 * e * \text{norm } (x - y)$

proof clarify

fix m n x y

assume as: $N \leq m \ N \leq n \ x \in S \ y \in S$

```

show norm ((f m x - f n x) - (f m y - f n y)) ≤ 2 * e * norm (x - y)
proof (rule differentiable_bound[where f'=λx h. f' m x h - f' n x h, OF ⟨convex
S⟩ _ _ as(β-4)])
  fix x
  assume x ∈ S
  show ((λa. f m a - f n a) has_derivative (λh. f' m x h - f' n x h)) (at x
within S)
    by (rule derivative_intros derf ⟨x∈S⟩)+
  show onorm (λh. f' m x h - f' n x h) ≤ 2 * e
  proof (rule onorm_bound)
    fix h
    have norm (f' m x h - f' n x h) ≤ norm (f' m x h - g' x h) + norm (f' n
x h - g' x h)
      using norm_triangle_ineq[of f' m x h - g' x h - f' n x h + g' x h]
      by (auto simp add: algebra_simps norm_minus_commute)
    also have ... ≤ e * norm h + e * norm h
      using nle[OF ⟨N ≤ m⟩ ⟨x ∈ S⟩, of h] nle[OF ⟨N ≤ n⟩ ⟨x ∈ S⟩, of h]
      by (auto simp add: field_simps)
    finally show norm (f' m x h - f' n x h) ≤ 2 * e * norm h
      by auto
  qed (simp add: ⟨0 ≤ e⟩)
qed
qed

```

lemma has_derivative_sequence_Lipschitz:

```

fixes f :: nat ⇒ 'a::real_normed_vector ⇒ 'b::real_normed_vector
assumes convex S
  and ∧n x. x ∈ S ⇒ ((f n) has_derivative (f' n x)) (at x within S)
  and nle: ∧e. e > 0 ⇒ ∀F n in sequentially. ∀x∈S. ∀h. norm (f' n x h - g'
x h) ≤ e * norm h
  and e > 0
shows ∃N. ∀m≥N. ∀n≥N. ∀x∈S. ∀y∈S.
  norm ((f m x - f n x) - (f m y - f n y)) ≤ e * norm (x - y)
proof -
  have *: 2 * (e/2) = e
    using ⟨e > 0⟩ by auto
  obtain N where ∀n≥N. ∀x∈S. ∀h. norm (f' n x h - g' x h) ≤ (e/2) * norm
h
    using nle ⟨e > 0⟩
  unfolding eventually_sequentially
  by (metis less_divide_eq_numeral1(1) mult_zero_left)
  then show ∃N. ∀m≥N. ∀n≥N. ∀x∈S. ∀y∈S. norm (f m x - f n x - (f m y
- f n y)) ≤ e * norm (x - y)
    apply (rule_tac x=N in exI)
    apply (rule has_derivative_sequence_lipschitz_lemma[where e=e/2, unfolded
*])
    using assms ⟨e > 0⟩
    apply auto
  done

```


qed

proposition *has_derivative_sequence*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{real_normed_vector} \Rightarrow 'b::\text{banach}$
assumes *convex S*
and *derf*: $\bigwedge n x. x \in S \implies ((f\ n) \text{ has_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$
and *nle*: $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g'$
 $x\ h) \leq e * \text{norm } h$
and $x0 \in S$
and *lim*: $((\lambda n. f\ n\ x0) \longrightarrow l) \text{ sequentially}$
shows $\exists g. \forall x \in S. (\lambda n. f\ n\ x) \longrightarrow g\ x \wedge (g \text{ has_derivative } g'(x)) \text{ (at } x \text{ within } S)$

proof –

have *lem1*: $\bigwedge e. e > 0 \implies \exists N. \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S.$
 $\text{norm } ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq e * \text{norm } (x - y)$
using *assms(1,2,3)* **by** (*rule has_derivative_sequence_Lipschitz*)
have $\exists g. \forall x \in S. ((\lambda n. f\ n\ x) \longrightarrow g\ x) \text{ sequentially}$
proof (*intro ballI bchoice*)
fix x
assume $x \in S$
show $\exists y. (\lambda n. f\ n\ x) \longrightarrow y$
unfolding *convergent_eq_Cauchy*
proof (*cases x = x0*)
case *True*
then show *Cauchy* $(\lambda n. f\ n\ x)$
using *LIMSEQ_imp_Cauchy[OF lim]* **by** *auto*
next
case *False*
show *Cauchy* $(\lambda n. f\ n\ x)$
unfolding *Cauchy_def*
proof (*intro allI impI*)
fix $e :: \text{real}$
assume $e > 0$
hence $*$: $e / 2 > 0 \wedge e / 2 / \text{norm } (x - x0) > 0$ **using** *False* **by** *auto*
obtain M **where** $M: \forall m \geq M. \forall n \geq M. \text{dist } (f\ m\ x0) (f\ n\ x0) < e / 2$
using *LIMSEQ_imp_Cauchy[OF lim]* ***** **unfolding** *Cauchy_def* **by** *blast*
obtain N **where** $N:$
 $\forall m \geq N. \forall n \geq N.$
 $\forall u \in S. \forall y \in S. \text{norm } (f\ m\ u - f\ n\ u - (f\ m\ y - f\ n\ y)) \leq$
 $e / 2 / \text{norm } (x - x0) * \text{norm } (u - y)$
using *lem1* $*$ (2) **by** *blast*
show $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (f\ m\ x) (f\ n\ x) < e$
proof (*intro exI allI impI*)
fix $m\ n$
assume $as: \max M\ N \leq m \wedge \max M\ N \leq n$
have $\text{dist } (f\ m\ x) (f\ n\ x) \leq \text{norm } (f\ m\ x0 - f\ n\ x0) + \text{norm } (f\ m\ x - f\ n$
 $x - (f\ m\ x0 - f\ n\ x0))$
unfolding *dist_norm*
by (*rule norm_triangle_sub*)

```

also have ...  $\leq \text{norm } (f\ m\ x0 - f\ n\ x0) + e / 2$ 
  using  $N \langle x \in S \rangle \langle x0 \in S \rangle$  as False by fastforce
also have ...  $< e / 2 + e / 2$ 
  by (rule add_strict_right_mono) (use as M in  $\langle \text{auto simp: dist\_norm} \rangle$ )
finally show  $\text{dist } (f\ m\ x) (f\ n\ x) < e$ 
  by auto
qed
qed
qed
qed
then obtain  $g$  where  $g: \forall x \in S. (\lambda n. f\ n\ x) \longrightarrow g\ x ..$ 
have lem2:  $\exists N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } ((f\ n\ x - f\ n\ y) - (g\ x - g\ y)) \leq$ 
 $e * \text{norm } (x - y)$  if  $e > 0$  for  $e$ 
proof -
  obtain  $N$  where
     $N: \forall m \geq N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } (f\ m\ x - f\ n\ x - (f\ m\ y - f\ n\ y))$ 
 $\leq e * \text{norm } (x - y)$ 
    using lem1  $\langle e > 0 \rangle$  by blast
    show  $\exists N. \forall n \geq N. \forall x \in S. \forall y \in S. \text{norm } (f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e * \text{norm } (x - y)$ 
    proof (intro exI ballI allI impI)
      fix  $n\ x\ y$ 
      assume as:  $N \leq n\ x \in S\ y \in S$ 
      have  $((\lambda m. \text{norm } (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y))) \longrightarrow \text{norm } (f\ n\ x - f\ n\ y - (g\ x - g\ y)))$  sequentially
      by (intro tendsto_intros g[rule_format] as)
      moreover have eventually  $(\lambda m. \text{norm } (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y)) \leq e * \text{norm } (x - y))$  sequentially
      unfolding eventually_sequentially
      proof (intro exI allI impI)
        fix  $m$ 
        assume  $N \leq m$ 
        then show  $\text{norm } (f\ n\ x - f\ n\ y - (f\ m\ x - f\ m\ y)) \leq e * \text{norm } (x - y)$ 
          using  $N$  as by (auto simp add: algebra_simps)
        qed
      ultimately show  $\text{norm } (f\ n\ x - f\ n\ y - (g\ x - g\ y)) \leq e * \text{norm } (x - y)$ 
        by (simp add: tendsto_upperbound)
      qed
    qed
  have  $\forall x \in S. ((\lambda n. f\ n\ x) \longrightarrow g\ x)$  sequentially  $\wedge (g \text{ has\_derivative } g' x)$  (at x within S)
    unfolding has_derivative_within_alt2
    proof (intro ballI conjI allI impI)
      fix  $x$ 
      assume  $x \in S$ 
      then show  $(\lambda n. f\ n\ x) \longrightarrow g\ x$ 
        by (simp add: g)
      have tog':  $(\lambda n. f' n\ x\ u) \longrightarrow g' x\ u$  for  $u$ 
        unfolding filterlim_def le_nhds_metric_le eventually_filtermap dist_norm

```

```

proof (intro allI impI)
  fix e :: real
  assume e > 0
  show eventually ( $\lambda n. \text{norm } (f' n x u - g' x u) \leq e$ ) sequentially
  proof (cases u = 0)
    case True
      have eventually ( $\lambda n. \text{norm } (f' n x u - g' x u) \leq e * \text{norm } u$ ) sequentially
        using nle <0 < e> <x ∈ S> by (fast elim: eventually_mono)
      then show ?thesis
        using <u = 0> <0 < e> by (auto elim: eventually_mono)
    next
      case False
        with <0 < e> have 0 < e / norm u by simp
        then have eventually ( $\lambda n. \text{norm } (f' n x u - g' x u) \leq e / \text{norm } u * \text{norm } u$ ) sequentially
          using nle <x ∈ S> by (fast elim: eventually_mono)
        then show ?thesis
          using <u ≠ 0> by simp
    qed
  qed
show bounded_linear (g' x)
proof
  fix x' y z :: 'a
  fix c :: real
  note lin = assms(2)[rule_format, OF <x ∈ S>, THEN has_derivative_bounded_linear]
  have ( $\lambda n. f' n x (c *_{\mathbb{R}} x')$ )  $\longrightarrow$   $c *_{\mathbb{R}} g' x x'$ 
    unfolding lin[THEN bounded_linear.linear, THEN linear_cmul]
    by (intro tendsto_intros tog')
  then show  $g' x (c *_{\mathbb{R}} x') = c *_{\mathbb{R}} g' x x'$ 
    using LIMSEQ_unique tog' by blast
  have ( $\lambda n. f' n x (y + z)$ )  $\longrightarrow$   $g' x y + g' x z$ 
    unfolding lin[THEN bounded_linear.linear, THEN linear_add]
    by (simp add: tendsto_add tog')
  then show  $g' x (y + z) = g' x y + g' x z$ 
    using LIMSEQ_unique tog' by blast
  obtain N where N:  $\forall h. \text{norm } (f' N x h - g' x h) \leq 1 * \text{norm } h$ 
    using nle <x ∈ S> unfolding eventually_sequentially by (fast intro:
zero_less_one)
  have bounded_linear (f' N x)
    using derf <x ∈ S> by fast
  from bounded_linear.bounded [OF this]
  obtain K where K:  $\forall h. \text{norm } (f' N x h) \leq \text{norm } h * K ..$ 
  {
    fix h
    have  $\text{norm } (g' x h) = \text{norm } (f' N x h - (f' N x h - g' x h))$ 
      by simp
    also have ...  $\leq \text{norm } (f' N x h) + \text{norm } (f' N x h - g' x h)$ 
      by (rule norm_triangle_ineq4)
    also have ...  $\leq \text{norm } h * K + 1 * \text{norm } h$ 
  }

```

```

    using N K by (fast intro: add_mono)
    finally have norm (g' x h) ≤ norm h * (K + 1)
    by (simp add: ring_distrib)
  }
  then show ∃K. ∀h. norm (g' x h) ≤ norm h * K by fast
qed
show eventually (λy. norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x))
(at x within S)
  if e > 0 for e
proof -
  have *: e / 3 > 0
  using that by auto
  obtain N1 where N1: ∀n≥N1. ∀x∈S. ∀h. norm (f' n x h - g' x h) ≤ e /
3 * norm h
  using nle * unfolding eventually_sequentially by blast
  obtain N2 where
    N2[rule_format]: ∀n≥N2. ∀x∈S. ∀y∈S. norm (f n x - f n y - (g x - g
y)) ≤ e / 3 * norm (x - y)
  using lem2 * by blast
  let ?N = max N1 N2
  have eventually (λy. norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 *
norm (y - x)) (at x within S)
  using derf[unfolded has_derivative_within_alt2] and ⟨x ∈ S⟩ and * by
fast
  moreover have eventually (λy. y ∈ S) (at x within S)
  unfolding eventually_at by (fast intro: zero_less_one)
  ultimately show ∀F y in at x within S. norm (g y - g x - g' x (y - x)) ≤
e * norm (y - x)
  proof (rule eventually_elim2)
    fix y
    assume y ∈ S
    assume norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 * norm (y - x)
    moreover have norm (g y - g x - (f ?N y - f ?N x)) ≤ e / 3 * norm (y
- x)
    using N2[OF _ ⟨y ∈ S⟩ ⟨x ∈ S⟩]
    by (simp add: norm_minus_commute)
    ultimately have norm (g y - g x - f' ?N x (y - x)) ≤ 2 * e / 3 * norm
(y - x)
    using norm_triangle_le[of g y - g x - (f ?N y - f ?N x) f ?N y - f ?N
x - f' ?N x (y - x) 2 * e / 3 * norm (y - x)]
    by (auto simp add: algebra_simps)
    moreover
    have norm (f' ?N x (y - x) - g' x (y - x)) ≤ e / 3 * norm (y - x)
    using N1 ⟨x ∈ S⟩ by auto
    ultimately show norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x)
    using norm_triangle_le[of g y - g x - f' (max N1 N2) x (y - x) f' (max
N1 N2) x (y - x) - g' x (y - x)]
    by (auto simp add: algebra_simps)
  qed
qed

```

```

  qed
  qed
  then show ?thesis by fast
  qed

```

Can choose to line up antiderivatives if we want.

```

lemma has_antiderivative_sequence:
  fixes f :: nat => 'a::real_normed_vector => 'b::banach
  assumes convex S
    and der:  $\bigwedge n x. x \in S \implies ((f\ n) \text{ has\_derivative } (f'\ n\ x)) \text{ (at } x \text{ within } S)$ 
    and no:  $\bigwedge e. e > 0 \implies \forall_F n \text{ in sequentially.}$ 
       $\forall x \in S. \forall h. \text{norm } (f'\ n\ x\ h - g'\ x\ h) \leq e * \text{norm } h$ 
  shows  $\exists g. \forall x \in S. (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 
proof (cases S = {})
  case False
  then obtain a where a  $\in S$ 
    by auto
  have *:  $\bigwedge P Q. \exists g. \forall x \in S. P\ g\ x \wedge Q\ g\ x \implies \exists g. \forall x \in S. Q\ g\ x$ 
    by auto
  show ?thesis
    apply (rule *)
    apply (rule has_antiderivative_sequence [OF <convex S> _ no, of  $\lambda n x. f\ n\ x + (f\ 0\ a - f\ n\ a)$ ])
    apply (metis assms(2) has_antiderivative_add_const)
    using <a  $\in S$ >
    apply auto
  done
qed auto

```

```

lemma has_antiderivative_limit:
  fixes g' :: 'a::real_normed_vector => 'a => 'b::banach
  assumes convex S
    and  $\bigwedge e. e > 0 \implies \exists f f'. \forall x \in S.$ 
       $(f \text{ has\_derivative } (f'\ x)) \text{ (at } x \text{ within } S) \wedge (\forall h. \text{norm } (f'\ x\ h - g'\ x\ h) \leq$ 
 $e * \text{norm } h)$ 
  shows  $\exists g. \forall x \in S. (g \text{ has\_derivative } g'\ x) \text{ (at } x \text{ within } S)$ 
proof -
  have *:  $\forall n. \exists f f'. \forall x \in S.$ 
     $(f \text{ has\_derivative } (f'\ x)) \text{ (at } x \text{ within } S) \wedge$ 
     $(\forall h. \text{norm } (f'\ x\ h - g'\ x\ h) \leq \text{inverse } (\text{real } (\text{Suc } n)) * \text{norm } h)$ 
    by (simp add: assms(2))
  obtain f where
    *:  $\bigwedge x. \exists f'. \forall xa \in S. (f\ x \text{ has\_derivative } f'\ xa) \text{ (at } xa \text{ within } S) \wedge$ 
       $(\forall h. \text{norm } (f'\ xa\ h - g'\ xa\ h) \leq \text{inverse } (\text{real } (\text{Suc } x)) * \text{norm } h)$ 
    using * by metis
  obtain f' where
    f':  $\bigwedge x. \forall z \in S. (f\ x \text{ has\_derivative } f'\ x\ z) \text{ (at } z \text{ within } S) \wedge$ 
       $(\forall h. \text{norm } (f'\ x\ z\ h - g'\ z\ h) \leq \text{inverse } (\text{real } (\text{Suc } x)) * \text{norm } h)$ 
    using * by metis

```

```

show ?thesis
proof (rule has_antiderivative_sequence[OF ‹convex S›, of f f'])
  fix e :: real
  assume e > 0
  obtain N where N: inverse (real (Suc N)) < e
    using reals_Archimedean[OF ‹e>0›] ..
  show  $\forall_F n$  in sequentially.  $\forall x \in S. \forall h. \text{norm } (f' n x h - g' x h) \leq e * \text{norm } h$ 
    unfolding eventually_sequentially
  proof (intro exI allI ballI impI)
    fix n x h
    assume n:  $N \leq n$  and x:  $x \in S$ 
    have *: inverse (real (Suc n))  $\leq e$ 
      using n N
    by (smt (verit, best) le_imp_inverse_le of_nat_0_less_iff of_nat_Suc
of_nat_le_iff zero_less_Suc)
    show  $\text{norm } (f' n x h - g' x h) \leq e * \text{norm } h$ 
      by (meson * mult_right_mono norm_ge_zero order.trans x f')
  qed
qed (use f' in auto)
qed

```

6.17.14 Differentiation of a series

proposition *has_derivative_series*:

```

fixes f :: nat  $\Rightarrow$  'a::real_normed_vector  $\Rightarrow$  'b::banach
assumes convex S
  and  $\bigwedge n x. x \in S \implies ((f n) \text{ has\_derivative } (f' n x))$  (at x within S)
  and  $\bigwedge e. e > 0 \implies \forall_F n$  in sequentially.  $\forall x \in S. \forall h. \text{norm } (\text{sum } (\lambda i. f' i x h)
\{..<n\} - g' x h) \leq e * \text{norm } h$ 
  and  $x \in S$ 
  and  $(\lambda n. f n x)$  sums l
shows  $\exists g. \forall x \in S. (\lambda n. f n x)$  sums (g x)  $\wedge$  (g has_derivative g' x) (at x within
S)
  unfolding sums_def
  apply (rule has_derivative_sequence[OF assms(1) __ assms(3)])
  apply (metis assms(2) has_derivative_sum)
  using assms(4-5)
  unfolding sums_def
  apply auto
  done

```

lemma *has_field_derivative_series*:

```

fixes f :: nat  $\Rightarrow$  ('a :: {real_normed_field,banach})  $\Rightarrow$  'a
assumes convex S
assumes  $\bigwedge n x. x \in S \implies (f n \text{ has\_field\_derivative } f' n x)$  (at x within S)
assumes uniform_limit S  $(\lambda n x. \sum i < n. f' i x) g'$  sequentially
assumes  $x0 \in S$  summable  $(\lambda n. f n x0)$ 
shows  $\exists g. \forall x \in S. (\lambda n. f n x)$  sums g x  $\wedge$  (g has_field_derivative g' x) (at x
within S)

```

```

unfolding has_field_derivative_def
proof (rule has_derivative_series)
  show  $\forall_F n$  in sequentially.
     $\forall x \in S. \forall h. \text{norm} ((\sum_{i < n}. f' i x * h) - g' x * h) \leq e * \text{norm } h$  if  $e > 0$ 
for  $e$ 
  unfolding eventually_sequentially
  proof -
    from that assms(3) obtain  $N$  where  $N: \bigwedge n x. n \geq N \implies x \in S \implies \text{norm} ((\sum_{i < n}. f' i x) - g' x) < e$ 
    unfolding uniform_limit_iff eventually_at_top_linorder dist_norm by blast
    {
      fix  $n :: \text{nat}$  and  $x h :: 'a$  assume  $nx: n \geq N x \in S$ 
      have  $\text{norm} ((\sum_{i < n}. f' i x * h) - g' x * h) = \text{norm} ((\sum_{i < n}. f' i x) - g' x) * \text{norm } h$ 
      by (simp add: norm_mult [symmetric] ring_distrib sum_distrib_right)
      also from  $N[OF nx]$  have  $\text{norm} ((\sum_{i < n}. f' i x) - g' x) \leq e$  by simp
      hence  $\text{norm} ((\sum_{i < n}. f' i x) - g' x) * \text{norm } h \leq e * \text{norm } h$ 
      by (intro mult_right_mono) simp_all
      finally have  $\text{norm} ((\sum_{i < n}. f' i x * h) - g' x * h) \leq e * \text{norm } h .$ 
    }
    thus  $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{norm} ((\sum_{i < n}. f' i x * h) - g' x * h) \leq e * \text{norm } h$  by blast
  qed
qed (use assms in <auto simp: has_field_derivative_def>)

```

lemma has_field_derivative_series':

```

fixes  $f :: \text{nat} \Rightarrow ('a :: \{\text{real\_normed\_field, banach}\}) \Rightarrow 'a$ 
assumes convex  $S$ 
assumes  $\bigwedge n x. x \in S \implies (f n \text{ has\_field\_derivative } f' n x)$  (at  $x$  within  $S$ )
assumes uniformly_convergent_on  $S (\lambda n x. \sum_{i < n}. f' i x)$ 
assumes  $x0 \in S$  summable  $(\lambda n. f n x0)$   $x \in \text{interior } S$ 
shows summable  $(\lambda n. f n x)$   $((\lambda x. \sum n. f n x) \text{ has\_field\_derivative } (\sum n. f' n x))$  (at  $x$ )
proof -
  from  $\langle x \in \text{interior } S \rangle$  have  $x \in S$  using interior_subset by blast
  define  $g'$  where [abs_def]:  $g' x = (\sum i. f' i x)$  for  $x$ 
  from assms(3) have uniform_limit  $S (\lambda n x. \sum_{i < n}. f' i x)$   $g'$  sequentially
  by (simp add: uniformly_convergent_uniform_limit_iff suminf_eq_lim  $g'$ _def)
  from has_field_derivative_series[OF assms(1,2) this assms(4,5)] obtain  $g$ 
where  $g$ :
     $\bigwedge x. x \in S \implies (\lambda n. f n x)$  sums  $g x$ 
     $\bigwedge x. x \in S \implies (g \text{ has\_field\_derivative } g' x)$  (at  $x$  within  $S$ ) by blast
  from  $g(1)[OF \langle x \in S \rangle]$  show summable  $(\lambda n. f n x)$  by (simp add: sums_iff)
  from  $g(2)[OF \langle x \in S \rangle]$   $\langle x \in \text{interior } S \rangle$  have  $(g \text{ has\_field\_derivative } g' x)$  (at  $x$ )
  by (simp add: at_within_interior[of  $x S$ ])
  also have  $(g \text{ has\_field\_derivative } g' x)$  (at  $x$ )  $\longleftrightarrow$ 
     $((\lambda x. \sum n. f n x) \text{ has\_field\_derivative } g' x)$  (at  $x$ )
  using eventually_nhds_in_nhd[OF  $\langle x \in \text{interior } S \rangle$ ] interior_subset[of  $S$ ]  $g(1)$ 

```

by (intro DERIV_cong_ev) (auto elim!: eventually_mono simp: sums_iff)
 finally show $((\lambda x. \sum n. f n x) \text{ has_field_derivative } g' x) \text{ (at } x \text{)}$.
 qed

lemma differentiable_series:

fixes $f :: \text{nat} \Rightarrow ('a :: \{\text{real_normed_field, banach}\}) \Rightarrow 'a$
 assumes convex S open S
 assumes $\bigwedge n x. x \in S \implies (f n \text{ has_field_derivative } f' n x) \text{ (at } x \text{)}$
 assumes uniformly_convergent_on S $(\lambda n x. \sum i < n. f' i x)$
 assumes $x0 \in S$ summable $(\lambda n. f n x0)$ and $x: x \in S$
 shows summable $(\lambda n. f n x)$ and $(\lambda x. \sum n. f n x)$ differentiable (at x)
 proof -
 from assms(4) obtain g' where A: uniform_limit S $(\lambda n x. \sum i < n. f' i x)$ g'
 sequentially
 unfolding uniformly_convergent_on_def by blast
 from x and ⟨open S⟩ have S: at x within S = at x by (rule at_within_open)
 have $\exists g. \forall x \in S. (\lambda n. f n x) \text{ sums } g x \wedge (g \text{ has_field_derivative } g' x) \text{ (at } x \text{ within } S)$
 by (intro has_field_derivative_series[of S f f' g' x0] assms A has_field_derivative_at_within)
 then obtain g where $g: \bigwedge x. x \in S \implies (\lambda n. f n x) \text{ sums } g x$
 $\bigwedge x. x \in S \implies (g \text{ has_field_derivative } g' x) \text{ (at } x \text{ within } S)$ by blast
 from $g[OF x]$ show summable $(\lambda n. f n x)$ by (auto simp: summable_def)
 from $g(2)[OF x]$ have $g': (g \text{ has_derivative } (*)) (g' x) \text{ (at } x \text{)}$
 by (simp add: has_field_derivative_def S)
 have $(\lambda x. \sum n. f n x) \text{ has_derivative } (*)) (g' x) \text{ (at } x \text{)}$
 by (rule has_derivative_transform_within_open[OF g' ⟨open S⟩ x])
 (insert g, auto simp: sums_iff)
 thus $(\lambda x. \sum n. f n x)$ differentiable (at x) unfolding differentiable_def
 by (auto simp: summable_def differentiable_def has_field_derivative_def)
 qed

lemma differentiable_series':

fixes $f :: \text{nat} \Rightarrow ('a :: \{\text{real_normed_field, banach}\}) \Rightarrow 'a$
 assumes convex S open S
 assumes $\bigwedge n x. x \in S \implies (f n \text{ has_field_derivative } f' n x) \text{ (at } x \text{)}$
 assumes uniformly_convergent_on S $(\lambda n x. \sum i < n. f' i x)$
 assumes $x0 \in S$ summable $(\lambda n. f n x0)$
 shows $(\lambda x. \sum n. f n x)$ differentiable (at $x0$)
 using differentiable_series[OF assms, of $x0$] ⟨ $x0 \in S$ ⟩ by blast+

6.17.15 Derivative as a vector

Considering derivative $\text{real} \Rightarrow 'b$ as a vector.

definition vector_derivative $f \text{ net} = (\text{SOME } f'. (f \text{ has_vector_derivative } f') \text{ net})$

lemma vector_derivative_unique_within:

assumes not_bot: at x within S $\neq \text{bot}$
 and $f': (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$
 and $f'': (f \text{ has_vector_derivative } f'') \text{ (at } x \text{ within } S)$


```

  shows  $f' = f''$ 
proof -
  have  $(\lambda x. x *_R f') = (\lambda x. x *_R f'')$ 
proof (rule frechet_derivative_unique_within, simp_all)
  show  $\exists d. d \neq 0 \wedge |d| < e \wedge x + d \in S$  if  $0 < e$  for  $e$ 
  proof -
    from that
    obtain  $x'$  where  $x' \in S$   $x' \neq x$   $|x' - x| < e$ 
      using islimpt_approachable_real[of x S] not_bot
      by (auto simp add: trivial_limit_within)
    then show ?thesis
      using eq_iff_diff_eq_0 by fastforce
  qed
qed (use f' f'' in ⟨auto simp: has_vector_derivative_def⟩)
then show ?thesis
  unfolding fun_eq_iff by (metis scaleR_one)
qed

lemma vector_derivative_unique_at:
  (f has_vector_derivative f') (at x)  $\implies$  (f has_vector_derivative f'') (at x)  $\implies$ 
  f' = f''
  by (rule vector_derivative_unique_within) auto

lemma differentiableI_vector: (f has_vector_derivative y) F  $\implies$  f differentiable
  F
  by (auto simp: differentiable_def has_vector_derivative_def)

proposition vector_derivative_works:
  f differentiable net  $\longleftrightarrow$  (f has_vector_derivative (vector_derivative f net)) net
  (is ?l = ?r)
proof
  assume ?l
  obtain f' where f': (f has_derivative f') net
    using ⟨?l⟩ unfolding differentiable_def ..
  then interpret bounded_linear f'
    by auto
  show ?r
    unfolding vector_derivative_def has_vector_derivative_def
    by (rule someI[of _ f' 1]) (simp add: scaleR[symmetric] f')
qed (auto simp: vector_derivative_def has_vector_derivative_def differentiable_def)

lemma vector_derivative_within:
  assumes not_bot: at x within S  $\neq$  bot and y: (f has_vector_derivative y) (at x
  within S)
  shows vector_derivative f (at x within S) = y
  using y
  by (intro vector_derivative_unique_within[OF not_bot vector_derivative_works[THEN
  iffD1] y])
  (auto simp: differentiable_def has_vector_derivative_def)

```

1870

lemma *deriv_of_real* [*simp*]:

at x within A \neq *bot* \implies *vector_derivative_of_real* (*at x within A*) = 1

by (*auto intro!*: *vector_derivative_within derivative_eq_intros*)

lemma *frechet_derivative_eq_vector_derivative*:

assumes *f differentiable* (*at x*)

shows (*frechet_derivative f* (*at x*)) = ($\lambda r. r *_{\mathbb{R}}$ *vector_derivative f* (*at x*))

using *assms*

by (*auto simp: differentiable_iff_scaleR_vector_derivative_def has_vector_derivative_def*
intro: someI frechet_derivative_at [symmetric])

lemma *has_real_derivative*:

fixes *f* :: *real* \Rightarrow *real*

assumes (*f has_derivative f'*) *F*

obtains *c* **where** (*f has_real_derivative c*) *F*

proof –

obtain *c* **where** *f' =* ($\lambda x. x * c$)

by (*metis assms has_derivative_bounded_linear real_bounded_linear*)

then show *?thesis*

by (*metis assms that has_field_derivative_def mult_commute_abs*)

qed

lemma *has_real_derivative_iff*:

fixes *f* :: *real* \Rightarrow *real*

shows ($\exists c. (f \text{ has_real_derivative } c) F$) = ($\exists D. (f \text{ has_derivative } D) F$)

by (*metis has_field_derivative_def has_real_derivative*)

lemma *has_vector_derivative_cong_ev*:

assumes ***: *eventually* ($\lambda x. x \in S \longrightarrow f x = g x$) (*nhds x*) *f x = g x*

shows (*f has_vector_derivative f'*) (*at x within S*) = (*g has_vector_derivative*
f') (*at x within S*)

proof (*cases at x within S = bot*)

case *True*

then show *?thesis*

by (*simp add: has_derivative_def has_vector_derivative_def*)

next

case *False*

then show *?thesis*

unfolding *has_vector_derivative_def has_derivative_def*

using ***

apply (*intro refl conj_cong filterlim_cong*)

apply (*auto simp: Lim_ident_at eventually_at_filter elim: eventually_mono*)

done

qed

lemma *vector_derivative_cong_eq*:

assumes *eventually* ($\lambda x. x \in A \longrightarrow f x = g x$) (*nhds x*) *x = y* *A = B* *x \in A*

shows *vector_derivative f* (*at x within A*) = *vector_derivative g* (*at y within*

B)
proof –
 have $f x = g x$
 using *assms eventually_nhds_x_imp_x* **by** *blast*
 hence $(\lambda D. (f \text{ has_vector_derivative } D) \text{ (at } x \text{ within } A)) =$
 $(\lambda D. (g \text{ has_vector_derivative } D) \text{ (at } x \text{ within } A))$ **using** *assms*
by (*intro ext has_vector_derivative_cong_ev refl assms*) *simp_all*
thus *?thesis* **by** (*simp add: vector_derivative_def assms*)
qed

lemma *islimpt_closure_open*:
 fixes $s :: 'a::\text{perfect_space set}$
 assumes *open s* **and** $t: t = \text{closure } s \ x \in t$
 shows $x \text{ islimpt } t$
proof *cases*
 assume $x \in s$
 { **fix** T **assume** $x \in T$ *open T*
 then **have** *open* $(s \cap T)$
 using $\langle \text{open } s \rangle$ **by** *auto*
 then **have** $s \cap T \neq \{x\}$
 using *not_open_singleton[of x]* **by** *auto*
 with $\langle x \in T \rangle \langle x \in s \rangle$ **have** $\exists y \in t. y \in T \wedge y \neq x$
 using *closure_subset[of s]* **by** (*auto simp: t*) }
 then **show** *?thesis*
by (*auto intro!: islimptI*)
next
 assume $x \notin s$ **with** t **show** *?thesis*
unfolding *t closure_def* **by** (*auto intro: islimpt_subset*)
qed

lemma *vector_derivative_unique_within_closed_interval*:
 assumes $ab: a < b \ x \in \text{cbox } a \ b$
 assumes $D: (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } \text{cbox } a \ b) (f \text{ has_vector_derivative } f'') \text{ (at } x \text{ within } \text{cbox } a \ b)$
 shows $f' = f''$
 using *ab*
by (*intro vector_derivative_unique_within[OF _ D]*)
 (*auto simp: trivial_limit_within intro!: islimpt_closure_open[where $s = \{a <..< b\}$]*)

lemma *vector_derivative_at*:
 $(f \text{ has_vector_derivative } f') \text{ (at } x) \implies \text{vector_derivative } f \text{ (at } x) = f'$
by (*intro vector_derivative_within at_neq_bot*)

lemma *has_vector_derivative_id_at [simp]*: $\text{vector_derivative } (\lambda x. x) \text{ (at } a) = 1$
by (*simp add: vector_derivative_at*)

lemma *vector_derivative_minus_at [simp]*:
 f differentiable at a

1872

$\implies \text{vector_derivative } (\lambda x. - f x) \text{ (at } a) = - \text{vector_derivative } f \text{ (at } a)$
by (*simp add: vector_derivative_at has_vector_derivative_minus vector_derivative_works [symmetric]*)

lemma *vector_derivative_add_at [simp]:*
 $\llbracket f \text{ differentiable at } a; g \text{ differentiable at } a \rrbracket$
 $\implies \text{vector_derivative } (\lambda x. f x + g x) \text{ (at } a) = \text{vector_derivative } f \text{ (at } a) + \text{vector_derivative } g \text{ (at } a)$
by (*simp add: vector_derivative_at has_vector_derivative_add vector_derivative_works [symmetric]*)

lemma *vector_derivative_diff_at [simp, derivative_intros]:*
 $\llbracket f \text{ differentiable at } a; g \text{ differentiable at } a \rrbracket$
 $\implies \text{vector_derivative } (\lambda x. f x - g x) \text{ (at } a) = \text{vector_derivative } f \text{ (at } a) - \text{vector_derivative } g \text{ (at } a)$
by (*simp add: vector_derivative_at has_vector_derivative_diff vector_derivative_works [symmetric]*)

lemma *vector_derivative_mult_at [simp]:*
 fixes $f g :: \text{real} \Rightarrow 'a :: \text{real_normed_algebra}$
 shows $\llbracket f \text{ differentiable at } a; g \text{ differentiable at } a \rrbracket$
 $\implies \text{vector_derivative } (\lambda x. f x * g x) \text{ (at } a) = f a * \text{vector_derivative } g \text{ (at } a) + \text{vector_derivative } f \text{ (at } a) * g a$
by (*simp add: vector_derivative_at has_vector_derivative_mult vector_derivative_works [symmetric]*)

lemma *vector_derivative_scaleR_at [simp]:*
 $\llbracket f \text{ differentiable at } a; g \text{ differentiable at } a \rrbracket$
 $\implies \text{vector_derivative } (\lambda x. f x *_{\mathbb{R}} g x) \text{ (at } a) = f a *_{\mathbb{R}} \text{vector_derivative } g \text{ (at } a) + \text{vector_derivative } f \text{ (at } a) *_{\mathbb{R}} g a$
 apply (*intro vector_derivative_at has_vector_derivative_scaleR*)
 apply (*auto simp: vector_derivative_works has_vector_derivative_def has_field_derivative_def mult_commute_abs*)
 done

lemma *vector_derivative_within_cbox:*
 assumes $ab: a < b \ x \in \text{cbox } a \ b$
 assumes $f: (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } \text{cbox } a \ b)$
 shows $\text{vector_derivative } f \text{ (at } x \text{ within } \text{cbox } a \ b) = f'$
by (*metis assms box_real(2) f_islimpt_Icc trivial_limit_within vector_derivative_within*)

lemma *vector_derivative_within_closed_interval:*
 fixes $f :: \text{real} \Rightarrow 'a :: \text{euclidean_space}$
 assumes $a < b \ \text{and } x \in \{a..b\}$
 assumes $(f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } \{a..b\})$
 shows $\text{vector_derivative } f \text{ (at } x \text{ within } \{a..b\}) = f'$
 using *assms vector_derivative_within_cbox*
 by *fastforce*

lemma *has_vector_derivative_within_subset*:
 $(f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S) \implies T \subseteq S \implies (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } T)$
by (*auto simp: has_vector_derivative_def intro: has_derivative_subset*)

lemma *has_vector_derivative_at_within*:
 $(f \text{ has_vector_derivative } f') \text{ (at } x) \implies (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$
unfolding *has_vector_derivative_def*
by (*rule has_derivative_at_withinI*)

lemma *has_vector_derivative_weaken*:
fixes $x D$ **and** $f g S T$
assumes $f: (f \text{ has_vector_derivative } D) \text{ (at } x \text{ within } T)$
and $x \in S \ S \subseteq T$
and $\bigwedge x. x \in S \implies f x = g x$
shows $(g \text{ has_vector_derivative } D) \text{ (at } x \text{ within } S)$
proof –
have $(f \text{ has_vector_derivative } D) \text{ (at } x \text{ within } S) \longleftrightarrow (g \text{ has_vector_derivative } D) \text{ (at } x \text{ within } S)$
unfolding *has_vector_derivative_def has_derivative_iff_norm*
using *assms* **by** (*intro conj_cong Lim_cong_within refl*) *auto*
then show *?thesis*
using *has_vector_derivative_within_subset[OF f ‹S ⊆ T›]* **by** *simp*
qed

lemma *has_vector_derivative_transform_within*:
assumes $(f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$
and $0 < d$
and $x \in S$
and $\bigwedge x'. \llbracket x' \in S; \text{dist } x' x < d \rrbracket \implies f x' = g x'$
shows $(g \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform_within*)

lemma *has_vector_derivative_transform_within_open*:
assumes $(f \text{ has_vector_derivative } f') \text{ (at } x)$
and *open S*
and $x \in S$
and $\bigwedge y. y \in S \implies f y = g y$
shows $(g \text{ has_vector_derivative } f') \text{ (at } x)$
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform_within_open*)

lemma *has_vector_derivative_transform*:
assumes $x \in S \ \bigwedge x. x \in S \implies g x = f x$
assumes $f': (f \text{ has_vector_derivative } f') \text{ (at } x \text{ within } S)$

shows $(g \text{ has_vector_derivative } f')$ (at x within S)
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform*)

lemma *vector_diff_chain_at*:

assumes $(f \text{ has_vector_derivative } f')$ (at x)
and $(g \text{ has_vector_derivative } g')$ (at $(f x)$)
shows $((g \circ f) \text{ has_vector_derivative } (f' *_{\mathbb{R}} g'))$ (at x)
using *assms has_vector_derivative_at_within has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_diff_chain_within*:

assumes $(f \text{ has_vector_derivative } f')$ (at x within s)
and $(g \text{ has_vector_derivative } g')$ (at $(f x)$ within $f' \text{ ` } s$)
shows $((g \circ f) \text{ has_vector_derivative } (f' *_{\mathbb{R}} g'))$ (at x within s)
using *assms has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_derivative_const_at* [*simp*]: $\text{vector_derivative } (\lambda x. c)$ (at a) = 0
by (*simp add: vector_derivative_at*)

lemma *vector_derivative_at_within_ivl*:

$(f \text{ has_vector_derivative } f')$ (at x) \implies
 $a \leq x \implies x \leq b \implies a < b \implies \text{vector_derivative } f$ (at x within $\{a..b\}$) = f'
using *has_vector_derivative_at_within vector_derivative_within_cbox* **by** *fastforce*

lemma *vector_derivative_chain_at*:

assumes f differentiable at x (g differentiable at $(f x)$)
shows $\text{vector_derivative } (g \circ f)$ (at x) =
 $\text{vector_derivative } f$ (at x) $*_{\mathbb{R}}$ $\text{vector_derivative } g$ (at $(f x)$)
by (*metis vector_diff_chain_at vector_derivative_at vector_derivative_works assms*)

lemma *field_vector_diff_chain_at*:

assumes Df : $(f \text{ has_vector_derivative } f')$ (at x)
and Dg : $(g \text{ has_field_derivative } g')$ (at $(f x)$)
shows $((g \circ f) \text{ has_vector_derivative } (f' * g'))$ (at x)
using *diff_chain_at* [*OF Df* [*unfolded has_vector_derivative_def*]
 Dg [*unfolded has_field_derivative_def*]]
by (*auto simp: o_def mult.commute has_vector_derivative_def*)

lemma *vector_derivative_chain_within*:

assumes at x within $S \neq \text{bot}$ f differentiable (at x within S)
 $(g \text{ has_derivative } g')$ (at $(f x)$ within $f' \text{ ` } S$)
shows $\text{vector_derivative } (g \circ f)$ (at x within S) =
 g' ($\text{vector_derivative } f$ (at x within S))
apply (*rule vector_derivative_within* [*OF* $\langle \text{at } x \text{ within } S \neq \text{bot} \rangle$])
apply (*rule vector_derivative_diff_chain_within*)

using *assms*(2-3) *vector_derivative_works*
by *auto*

6.17.16 Field differentiability

definition *field_differentiable* :: [*'a* \Rightarrow *'a::real_normed_field*, *'a filter*] \Rightarrow *bool*
 (**infixr** (*field'_differentiable*) 50)
where *f field_differentiable F* $\equiv \exists f'$. (*f has_field_derivative f'*) *F*

lemma *field_differentiable_imp_differentiable*:
f field_differentiable F \Longrightarrow *f differentiable F*
unfolding *field_differentiable_def differentiable_def*
using *has_field_derivative_imp_has_derivative* **by** *auto*

lemma *field_differentiable_imp_continuous_at*:
f field_differentiable (at x within S) \Longrightarrow *continuous (at x within S) f*
by (*metis DERIV_continuous field_differentiable_def*)

lemma *field_differentiable_within_subset*:
 $\llbracket f \text{ field_differentiable (at } x \text{ within } S); T \subseteq S \rrbracket \Longrightarrow f \text{ field_differentiable (at } x \text{ within } T)$
by (*metis DERIV_subset field_differentiable_def*)

lemma *field_differentiable_at_within*:
 $\llbracket f \text{ field_differentiable (at } x \rrbracket$
 $\Longrightarrow f \text{ field_differentiable (at } x \text{ within } S)$
unfolding *field_differentiable_def*
by (*metis DERIV_subset top_greatest*)

lemma *field_differentiable_linear* [*simp,derivative_intros*]: ($(*)$ *c*) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def mult_commute_abs*
by (*force intro: has_derivative_mult_right*)

lemma *field_differentiable_const* [*simp,derivative_intros*]: ($\lambda z. c$) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def*
using *DERIV_const has_field_derivative_imp_has_derivative* **by** *blast*

lemma *field_differentiable_ident* [*simp,derivative_intros*]: ($\lambda z. z$) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def*
using *DERIV_ident has_field_derivative_def* **by** *blast*

lemma *field_differentiable_id* [*simp,derivative_intros*]: *id field_differentiable F*
unfolding *id_def* **by** (*rule field_differentiable_ident*)

lemma *field_differentiable_minus* [*derivative_intros*]:
f field_differentiable F \Longrightarrow ($\lambda z. - (f z)$) *field_differentiable F*

unfolding *field_differentiable_def* **by** (*metis field_differentiable_minus*)

lemma *field_differentiable_diff_const* [*simp, derivative_intros*]:

$(-)\ c$ *field_differentiable* F

unfolding *field_differentiable_def* **by** (*rule derivative_eq_intros exI | force*) $+$

lemma *field_differentiable_add* [*derivative_intros*]:

assumes f *field_differentiable* F g *field_differentiable* F

shows $(\lambda z. f\ z + g\ z)$ *field_differentiable* F

using *assms* **unfolding** *field_differentiable_def*

by (*metis field_differentiable_add*)

lemma *field_differentiable_add_const* [*simp, derivative_intros*]:

$(+)\ c$ *field_differentiable* F

by (*simp add: field_differentiable_add*)

lemma *field_differentiable_sum* [*derivative_intros*]:

$(\bigwedge i. i \in I \implies (f\ i)$ *field_differentiable* F) $\implies (\lambda z. \sum_{i \in I} f\ i\ z)$ *field_differentiable* F

by (*induct I rule: infinite_finite_induct*)

(auto intro: field_differentiable_add field_differentiable_const)

lemma *field_differentiable_diff* [*derivative_intros*]:

assumes f *field_differentiable* F g *field_differentiable* F

shows $(\lambda z. f\ z - g\ z)$ *field_differentiable* F

using *assms* **unfolding** *field_differentiable_def*

by (*metis field_differentiable_diff*)

lemma *field_differentiable_inverse* [*derivative_intros*]:

assumes f *field_differentiable* (*at a within S*) $f\ a \neq 0$

shows $(\lambda z. \text{inverse}\ (f\ z))$ *field_differentiable* (*at a within S*)

using *assms* **unfolding** *field_differentiable_def*

by (*metis DERIV_inverse_fun*)

lemma *field_differentiable_mult* [*derivative_intros*]:

assumes f *field_differentiable* (*at a within S*)

g *field_differentiable* (*at a within S*)

shows $(\lambda z. f\ z * g\ z)$ *field_differentiable* (*at a within S*)

using *assms* **unfolding** *field_differentiable_def*

by (*metis DERIV_mult [of f _ a S g]*)

lemma *field_differentiable_divide* [*derivative_intros*]:

assumes f *field_differentiable* (*at a within S*)

g *field_differentiable* (*at a within S*)

$g\ a \neq 0$

shows $(\lambda z. f\ z / g\ z)$ *field_differentiable* (*at a within S*)

using *assms* **unfolding** *field_differentiable_def*

by (*metis DERIV_divide [of f _ a S g]*)

lemma *field_differentiable_power* [derivative_intros]:

assumes f *field_differentiable* (at a within S)
shows $(\lambda z. f z ^ n)$ *field_differentiable* (at a within S)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_power*)

lemma *field_differentiable_cnj_cnj*:

assumes f *field_differentiable* (at $(cnj z)$)
shows $(cnj \circ f \circ cnj)$ *field_differentiable* (at z)
using *has_field_derivative_cnj_cnj* *assms*
by (*auto simp: field_differentiable_def*)

lemma *field_differentiable_transform_within*:

$0 < d \implies$
 $x \in S \implies$
 $(\wedge x'. x' \in S \implies dist\ x'\ x < d \implies f\ x' = g\ x') \implies$
 f *field_differentiable* (at x within S)
 $\implies g$ *field_differentiable* (at x within S)
unfolding *field_differentiable_def* *has_field_derivative_def*
by (*blast intro: has_derivative_transform_within*)

lemma *field_differentiable_compose_within*:

assumes f *field_differentiable* (at a within S)
 g *field_differentiable* (at $(f a)$ within $f'S$)
shows $(g \circ f)$ *field_differentiable* (at a within S)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_image_chain*)

lemma *field_differentiable_compose*:

f *field_differentiable* at $z \implies g$ *field_differentiable* at $(f z)$
 $\implies (g \circ f)$ *field_differentiable* at z

by (*metis field_differentiable_at_within field_differentiable_compose_within*)

lemma *field_differentiable_within_open*:

$\llbracket a \in S; \text{open } S \rrbracket \implies f$ *field_differentiable* at a within $S \longleftrightarrow$
 f *field_differentiable* at a

unfolding *field_differentiable_def*
by (*metis at_within_open*)

lemma *exp_scaleR_has_vector_derivative_right*:

$((\lambda t. \exp (t *_R A))$ *has_vector_derivative* $\exp (t *_R A) * A$) (at t within T)
unfolding *has_vector_derivative_def*

proof (*rule has_derivativeI*)

let $?F =$ at t within $(T \cap \{t - 1 <..< t + 1\})$

have $*$: at t within $T = ?F$

by (*rule at_within_nhd*[**where** $S = \{t - 1 <..< t + 1\}$]) *auto*

let $?e = \lambda i x. (\text{inverse } (1 + \text{real } i) * \text{inverse } (\text{fact } i) * (x - t) ^ i) *_R (A * A ^ i)$

have $\forall_F n$ in sequentially.

$\forall x \in T \cap \{t - 1 < \dots < t + 1\}. \text{norm } (?e \ n \ x) \leq \text{norm } (A \wedge (n + 1) /_R \text{fact } (n + 1))$
apply (*auto simp: algebra_split_simps intro!: eventuallyI*)
apply (*rule mult_left_mono*)
apply (*auto simp add: field_simps power_abs intro!: divide_right_mono power_le_one*)
done
then have *uniform_limit* ($T \cap \{t - 1 < \dots < t + 1\}$) ($\lambda n \ x. \sum_{i < n}. ?e \ i \ x$) ($\lambda x. \sum_{i < n}. ?e \ i \ x$) *sequentially*
by (*rule Weierstrass_m_test_ev*) (*intro summable_ignore_initial_segment summable_norm_exp*)
moreover
have $\forall_F \ x \text{ in sequentially. } x > 0$
by (*metis eventually_gt_at_top*)
then have
 $\forall_F \ n \text{ in sequentially. } ((\lambda x. \sum_{i < n}. ?e \ i \ x) \longrightarrow A) \ ?F$
by *eventually_elim*
(auto intro!: tendsto_eq_intros
simp: power_0_left if_distrib if_distribR
cong: if_cong)
ultimately
have [*tendsto_intros*]: $((\lambda x. \sum_{i < n}. ?e \ i \ x) \longrightarrow A) \ ?F$
by (*auto intro!: swap_uniform_limit[where f = $\lambda n \ x. \sum_{i < n}. ?e \ i \ x$ and F = sequentially]*)
have [*tendsto_intros*]: $((\lambda x. \text{if } x = t \text{ then } 0 \text{ else } 1) \longrightarrow 1) \ ?F$
by (*rule tendsto_eventually*) (*simp add: eventually_at_filter*)
have $((\lambda y. ((y - t) / \text{abs } (y - t)) *_R ((\sum_{n < n}. ?e \ n \ y) - A)) \longrightarrow 0)$ (*at t within T*)
unfolding *
by (*rule tendsto_norm_zero_cancel*) (*auto intro!: tendsto_eq_intros*)

moreover have $\forall_F \ x \text{ in at } t \text{ within } T. x \neq t$
by (*simp add: eventually_at_filter*)
then have $\forall_F \ x \text{ in at } t \text{ within } T. ((x - t) / |x - t|) *_R ((\sum_{n < n}. ?e \ n \ x) - A) =$
 $(\text{exp } ((x - t) *_R A) - 1 - (x - t) *_R A) /_R \text{norm } (x - t)$
proof *eventually_elim*
case (*elim x*)
have $(\text{exp } ((x - t) *_R A) - 1 - (x - t) *_R A) /_R \text{norm } (x - t) =$
 $((\sum_{n < n}. (x - t) *_R ?e \ n \ x) - (x - t) *_R A) /_R \text{norm } (x - t)$
unfolding *exp_first_term*
by (*simp add: ac_simps*)
also
have *summable* ($\lambda n. ?e \ n \ x$)
proof -
from *elim have* $?e \ n \ x = (((x - t) *_R A) \wedge (n + 1)) /_R \text{fact } (n + 1) /_R (x - t)$
for n
by *simp*
then show *?thesis*
by (*auto simp only:*)

intro!: summable_scaleR_right summable_ignore_initial_segment summable_exp_generic)
 qed
 then have $(\sum n. (x - t) *_{\mathbb{R}} ?e n x) = (x - t) *_{\mathbb{R}} (\sum n. ?e n x)$
 by (rule suminf_scaleR_right[symmetric])
 also have $(\dots - (x - t) *_{\mathbb{R}} A) /_{\mathbb{R}} \text{norm } (x - t) = (x - t) *_{\mathbb{R}} ((\sum n. ?e n x) - A) /_{\mathbb{R}} \text{norm } (x - t)$
 by (simp add: algebra_simps)
 finally show ?case
 by simp (simp add: field_simps)
 qed

ultimately have $((\lambda y. (\exp ((y - t) *_{\mathbb{R}} A) - 1 - (y - t) *_{\mathbb{R}} A) /_{\mathbb{R}} \text{norm } (y - t)) \longrightarrow 0)$ (at t within T)
 by (rule Lim_transform_eventually)
 from tendsto_mult_right_zero[OF this, where $c = \exp (t *_{\mathbb{R}} A)$]
 show $((\lambda y. (\exp (y *_{\mathbb{R}} A) - \exp (t *_{\mathbb{R}} A) - (y - t) *_{\mathbb{R}} (\exp (t *_{\mathbb{R}} A) * A)) /_{\mathbb{R}} \text{norm } (y - t)) \longrightarrow 0)$
 (at t within T)
 by (rule Lim_transform_eventually)
 (auto simp: field_split_simps exp_add_commuting[symmetric])
 qed (rule bounded_linear_scaleR_left)

lemma exp_times_scaleR_commute: $\exp (t *_{\mathbb{R}} A) * A = A * \exp (t *_{\mathbb{R}} A)$
 using exp_times_arg_commute[symmetric, of $t *_{\mathbb{R}} A$]
 by (auto simp: algebra_simps)

lemma exp_scaleR_has_vector_derivative_left: $((\lambda t. \exp (t *_{\mathbb{R}} A)) \text{ has_vector_derivative } A * \exp (t *_{\mathbb{R}} A))$ (at t)
 using exp_scaleR_has_vector_derivative_right[of $A t$]
 by (simp add: exp_times_scaleR_commute)

lemma field_differentiable_series:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{real_normed_field}, \text{banach}\} \Rightarrow 'a$
 assumes convex S open S
 assumes $\bigwedge n x. x \in S \implies (f n \text{ has_field_derivative } f' n x)$ (at x)
 assumes uniformly_convergent_on S $(\lambda n x. \sum i < n. f' i x)$
 assumes $x_0 \in S$ summable $(\lambda n. f n x_0)$ and $x: x \in S$
 shows $(\lambda x. \sum n. f n x)$ field_differentiable (at x)
proof –
 from assms(4) obtain g' where $A: \text{uniform_limit } S (\lambda n x. \sum i < n. f' i x) g'$
 sequentially
 unfolding uniformly_convergent_on_def by blast
 from x and $\langle \text{open } S \rangle$ have $S: \text{at } x \text{ within } S = \text{at } x$ by (rule at_within_open)
 have $\exists g. \forall x \in S. (\lambda n. f n x) \text{ sums } g x \wedge (g \text{ has_field_derivative } g' x)$ (at x within S)
 by (intro has_field_derivative_series[of $S f f' g' x_0$] assms A has_field_derivative_at_within)
 then obtain g where $g: \bigwedge x. x \in S \implies (\lambda n. f n x) \text{ sums } g x$
 $\bigwedge x. x \in S \implies (g \text{ has_field_derivative } g' x)$ (at x within S) by blast
 from $g(2)$ [OF x] have $g': (g \text{ has_derivative } (*) (g' x))$ (at x)

```

  by (simp add: has_field_derivative_def S)
  have (( $\lambda x. \sum n. f n x$ ) has_derivative (*)) (g' x) (at x)
  by (rule has_derivative_transform_within_open[OF g' ‹open S› x])
     (insert g, auto simp: sums_iff)
  thus ( $\lambda x. \sum n. f n x$ ) field_differentiable (at x) unfolding differentiable_def
  by (auto simp: summable_def field_differentiable_def has_field_derivative_def)
qed

```

Caratheodory characterization

```

lemma field_differentiable_caratheodory_at:
  f field_differentiable (at z)  $\longleftrightarrow$ 
    ( $\exists g. (\forall w. f(w) - f(z) = g(w) * (w - z)) \wedge$  continuous (at z) g)
  using CARAT_DERIV [of f]
  by (simp add: field_differentiable_def has_field_derivative_def)

```

```

lemma field_differentiable_caratheodory_within:
  f field_differentiable (at z within s)  $\longleftrightarrow$ 
    ( $\exists g. (\forall w. f(w) - f(z) = g(w) * (w - z)) \wedge$  continuous (at z within s) g)
  using DERIV_caratheodory_within [of f]
  by (simp add: field_differentiable_def has_field_derivative_def)

```

6.17.17 Field derivative

```

definition deriv :: ('a  $\Rightarrow$  'a::real_normed_field)  $\Rightarrow$  'a  $\Rightarrow$  'a where
  deriv f x  $\equiv$  SOME D. DERIV f x  $:\>$  D

```

```

lemma DERIV_imp_deriv: DERIV f x  $:\>$  f'  $\implies$  deriv f x = f'
  unfolding deriv_def by (metis some_equality DERIV_unique)

```

```

lemma DERIV_deriv_iff_has_field_derivative:
  DERIV f x  $:\>$  deriv f x  $\longleftrightarrow$  ( $\exists f'. (f$  has_field_derivative f') (at x))
  by (auto simp: has_field_derivative_def DERIV_imp_deriv)

```

```

lemma DERIV_deriv_iff_real_differentiable:
  fixes x :: real
  shows DERIV f x  $:\>$  deriv f x  $\longleftrightarrow$  f differentiable at x
  unfolding differentiable_def by (metis DERIV_imp_deriv has_real_derivative_iff)

```

```

lemma DERIV_deriv_iff_field_differentiable:
  DERIV f x  $:\>$  deriv f x  $\longleftrightarrow$  f field_differentiable at x
  unfolding field_differentiable_def by (metis DERIV_imp_deriv)

```

```

lemma vector_derivative_of_real_left:
  assumes f differentiable at x
  shows vector_derivative ( $\lambda x. of\_real (f x)$ ) (at x) = of_real (deriv f x)
  by (metis DERIV_deriv_iff_real_differentiable assms has_vector_derivative_of_real
  vector_derivative_at)

```

```

lemma vector_derivative_of_real_right:

```

assumes f *field_differentiable* *at* (*of_real* x)
shows *vector_derivative* $(\lambda x. f (\text{of_real } x))$ *at* $x = \text{deriv } f (\text{of_real } x)$
by (*metis DERIV_deriv_iff_field_differentiable* *assms* *has_vector_derivative_real_field* *vector_derivative_at*)

lemma *deriv_cong_ev*:

assumes *eventually* $(\lambda x. f x = g x)$ (*nhds* x) $x = y$
shows *deriv* $f x = \text{deriv } g y$
proof –
have $(\lambda D. (f \text{ has_field_derivative } D) (\text{at } x)) = (\lambda D. (g \text{ has_field_derivative } D) (\text{at } y))$
by (*intro ext DERIV_cong_ev refl assms*)
thus ?thesis **by** (*simp add: deriv_def assms*)
qed

lemma *higher_deriv_cong_ev*:

assumes *eventually* $(\lambda x. f x = g x)$ (*nhds* x) $x = y$
shows $(\text{deriv } \hat{\hat{ }} n) f x = (\text{deriv } \hat{\hat{ }} n) g y$
proof –
from *assms*(1) **have** *eventually* $(\lambda x. (\text{deriv } \hat{\hat{ }} n) f x = (\text{deriv } \hat{\hat{ }} n) g x)$ (*nhds* x)
proof (*induction n arbitrary: f g*)
case (*Suc n*)
from *Suc.prem*s **have** *eventually* $(\lambda y. \text{eventually } (\lambda z. f z = g z) (\text{nhds } y))$ (*nhds* x)
by (*simp add: eventually_eventually*)
hence *eventually* $(\lambda x. \text{deriv } f x = \text{deriv } g x)$ (*nhds* x)
by *eventually_elim* (*rule deriv_cong_ev, simp_all*)
thus ?case **by** (*auto intro!: deriv_cong_ev Suc simp: funpow_Suc_right simp* *del: funpow.simps*)
qed auto
with $\langle x = y \rangle$ *eventually_nhds_x_imp_x* **show** ?thesis **by** *blast*
qed

lemma *real_derivative_chain*:

fixes $x :: \text{real}$
shows f *differentiable* *at* $x \implies g$ *differentiable* *at* $(f x)$
 $\implies \text{deriv } (g \circ f) x = \text{deriv } g (f x) * \text{deriv } f x$
by (*metis DERIV_deriv_iff_real_differentiable DERIV_chain DERIV_imp_deriv*)

lemma *field_derivative_eq_vector_derivative*:

$(\text{deriv } f x) = \text{vector_derivative } f (\text{at } x)$
by (*simp add: mult.commute deriv_def vector_derivative_def has_vector_derivative_def* *has_field_derivative_def*)

proposition *field_differentiable_derivI*:

f *field_differentiable* *at* $x \implies (f \text{ has_field_derivative } \text{deriv } f x) (\text{at } x)$
by (*simp add: field_differentiable_def DERIV_deriv_iff_has_field_derivative*)

lemma *vector_derivative_chain_at_general*:

assumes f differentiable at x g field differentiable at $(f x)$
shows $\text{vector_derivative } (g \circ f) \text{ (at } x) = \text{vector_derivative } f \text{ (at } x) * \text{deriv } g \text{ (f } x)$
using *assms field_differentiable_derivI field_vector_diff_chain_at vector_derivative_at vector_derivative_works* **by** *blast*

lemma *deriv_chain*:
 f field differentiable at $x \implies g$ field differentiable at $(f x)$
 $\implies \text{deriv } (g \circ f) x = \text{deriv } g (f x) * \text{deriv } f x$
by (*metis DERIV_deriv_iff_field_differentiable DERIV_chain DERIV_imp_deriv*)

lemma *deriv_linear* [*simp*]: $\text{deriv } (\lambda w. c * w) = (\lambda z. c)$
by (*metis DERIV_imp_deriv DERIV_cmult_Id*)

lemma *deriv_uminus* [*simp*]: $\text{deriv } (\lambda w. -w) = (\lambda z. -1)$
using *deriv_linear[of -1]* **by** (*simp del: deriv_linear*)

lemma *deriv_ident* [*simp*]: $\text{deriv } (\lambda w. w) = (\lambda z. 1)$
by (*metis DERIV_imp_deriv DERIV_ident*)

lemma *deriv_id* [*simp*]: $\text{deriv } \text{id} = (\lambda z. 1)$
by (*simp add: id_def*)

lemma *deriv_const* [*simp*]: $\text{deriv } (\lambda w. c) = (\lambda z. 0)$
by (*metis DERIV_imp_deriv DERIV_const*)

lemma *deriv_add* [*simp*]:
 $\llbracket f \text{ field_differentiable at } z; g \text{ field_differentiable at } z \rrbracket$
 $\implies \text{deriv } (\lambda w. f w + g w) z = \text{deriv } f z + \text{deriv } g z$
unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*
by (*auto intro!: DERIV_imp_deriv derivative_intros*)

lemma *deriv_minus* [*simp*]:
 f field differentiable at $z \implies \text{deriv } (\lambda w. -f w) z = - \text{deriv } f z$
by (*simp add: DERIV_deriv_iff_field_differentiable DERIV_imp_deriv Deriv.field_differentiable_minus*)

lemma *deriv_diff* [*simp*]:
 $\llbracket f \text{ field_differentiable at } z; g \text{ field_differentiable at } z \rrbracket$
 $\implies \text{deriv } (\lambda w. f w - g w) z = \text{deriv } f z - \text{deriv } g z$
unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*
by (*auto intro!: DERIV_imp_deriv derivative_intros*)

lemma *deriv_mult* [*simp*]:
 $\llbracket f \text{ field_differentiable at } z; g \text{ field_differentiable at } z \rrbracket$
 $\implies \text{deriv } (\lambda w. f w * g w) z = f z * \text{deriv } g z + \text{deriv } f z * g z$
unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*
by (*auto intro!: DERIV_imp_deriv derivative_eq_intros*)

lemma *deriv_cmult*:

f field_differentiable at z \implies $\text{deriv } (\lambda w. c * f w) z = c * \text{deriv } f z$
by *simp*

lemma *deriv_cmult_right*:

f field_differentiable at z \implies $\text{deriv } (\lambda w. f w * c) z = \text{deriv } f z * c$
by *simp*

lemma *deriv_inverse* [*simp*]:

$\llbracket f \text{ field_differentiable at } z; f z \neq 0 \rrbracket$
 $\implies \text{deriv } (\lambda w. \text{inverse } (f w)) z = - \text{deriv } f z / f z ^ 2$
unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*
by (*safe intro!*: *DERIV_imp_deriv_derivative_eq_intros*) (*auto simp: field_split_simps power2_eq_square*)

lemma *deriv_divide* [*simp*]:

$\llbracket f \text{ field_differentiable at } z; g \text{ field_differentiable at } z; g z \neq 0 \rrbracket$
 $\implies \text{deriv } (\lambda w. f w / g w) z = (\text{deriv } f z * g z - f z * \text{deriv } g z) / g z ^ 2$
by (*simp add: field_class.field_divide_inverse field_differentiable_inverse*)
(*simp add: field_split_simps power2_eq_square*)

lemma *deriv_cdivide_right*:

f field_differentiable at z \implies $\text{deriv } (\lambda w. f w / c) z = \text{deriv } f z / c$
by (*simp add: field_class.field_divide_inverse*)

lemma *deriv_pow*: $\llbracket f \text{ field_differentiable at } z \rrbracket$

$\implies \text{deriv } (\lambda w. f w ^ n) z = (\text{if } n=0 \text{ then } 0 \text{ else } n * \text{deriv } f z * f z ^ (n - \text{Suc } 0))$

unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*

by (*auto intro!*: *DERIV_imp_deriv_derivative_eq_intros*)

lemma *deriv_sum* [*simp*]:

$\llbracket \bigwedge i. f i \text{ field_differentiable at } z \rrbracket$
 $\implies \text{deriv } (\lambda w. \text{sum } (\lambda i. f i w) S) z = \text{sum } (\lambda i. \text{deriv } (f i) z) S$
unfolding *DERIV_deriv_iff_field_differentiable[symmetric]*
by (*auto intro!*: *DERIV_imp_deriv_derivative_intros*)

lemma *deriv_compose_linear'*:

assumes *f field_differentiable at (c*w + a)*
shows $\text{deriv } (\lambda w. f (c*w + a)) z = c * \text{deriv } f (c*z + a)$
apply (*subst deriv_chain [where f= $\lambda w. c*w + a$,unfolded comp_def]*)
using *assms* **by** (*auto intro: derivative_intros*)

lemma *deriv_compose_linear*:

assumes *f field_differentiable at (c * z)*
shows $\text{deriv } (\lambda w. f (c * w)) z = c * \text{deriv } f (c * z)$

proof –

have $\text{deriv } (\lambda a. f (c * a)) z = \text{deriv } f (c * z) * c$
using *assms* **by** (*simp add: DERIV_chain2 DERIV_deriv_iff_field_differentiable*)

DERIV_imp_deriv)
then show *?thesis*
 by *simp*
qed

lemma nonzero_deriv_nonconstant:
assumes *df: DERIV f ξ :> df* **and** *S: open S ξ ∈ S* **and** *df ≠ 0*
shows $\neg f \text{ constant_on } S$
unfolding *constant_on_def*
by (*metis* $\langle df \neq 0 \rangle$ *has_field_derivative_transform_within_open* [*OF df S*] *DERIV_const DERIV_unique*)

6.17.18 Relation between convexity and derivative

proposition *convex_on_imp_above_tangent:*
assumes *convex: convex_on A f* **and** *connected: connected A*
assumes *c: c ∈ interior A* **and** *x : x ∈ A*
assumes *deriv: (f has_field_derivative f') (at c within A)*
shows $f x - f c \geq f' * (x - c)$
proof (*cases x c rule: linorder_cases*)
assume *xc: x > c*
let *?A' = interior A ∩ {c<..}*
from *c* **have** *c ∈ interior A ∩ closure {c<..}* **by** *auto*
also have $\dots \subseteq \text{closure}(\text{interior } A \cap \{c<..\})$ **by** (*intro open_Int_closure_subset*)
auto
finally have *at c within ?A' ≠ bot* **by** (*subst at_within_eq_bot_iff*) *auto*
moreover from *deriv* **have** $((\lambda y. (f y - f c) / (y - c)) \longrightarrow f')$ (*at c within ?A'*)
unfolding *has_field_derivative_iff* **using** *interior_subset[of A]* **by** (*blast intro: tendsto_mono at_le*)
moreover from *eventually_at_right_real[OF xc]*
have *eventually* $(\lambda y. (f y - f c) / (y - c) \leq (f x - f c) / (x - c))$ (*at_right c*)
proof *eventually_elim*
fix *y* **assume** *y: y ∈ {c<..*x*}*
with *convex* *connected x c* **have** $f y \leq (f x - f c) / (x - c) * (y - c) + f c$
using *interior_subset[of A]*
by (*intro convex_onD_Icc' convex_on_subset[OF convex] connected_contains_Icc*)
auto
hence $f y - f c \leq (f x - f c) / (x - c) * (y - c)$ **by** *simp*
thus $(f y - f c) / (y - c) \leq (f x - f c) / (x - c)$ **using** *y xc* **by** (*simp add: field_split_simps*)
qed
hence *eventually* $(\lambda y. (f y - f c) / (y - c) \leq (f x - f c) / (x - c))$ (*at c within ?A'*)
by (*blast intro: filter_leD at_le*)
ultimately have $f' \leq (f x - f c) / (x - c)$ **by** (*simp add: tendsto_upperbound*)
thus *?thesis* **using** *xc* **by** (*simp add: field_simps*)
next
assume *xc: x < c*


```

let ?A' = interior A ∩ {.. $c$ }
from c have c ∈ interior A ∩ closure {.. $c$ } by auto
also have ... ⊆ closure (interior A ∩ {.. $c$ }) by (intro open_Int_closure_subset)
auto
finally have at c within ?A' ≠ bot by (subst at_within_eq_bot_iff) auto
moreover from deriv have ((λy. (f y - f c) / (y - c)) → f') (at c within
?A')
unfolding has_field_derivative_iff using interior_subset[of A] by (blast intro:
tendsto_mono at_le)
moreover from eventually_at_left_real[OF xc]
have eventually (λy. (f y - f c) / (y - c) ≥ (f x - f c) / (x - c)) (at_left c)
proof eventually_elim
fix y assume y: y ∈ {x.. $c$ }
with convex_connected x c have f y ≤ (f x - f c) / (c - x) * (c - y) + f c
using interior_subset[of A]
by (intro convex_onD_Icc'' convex_on_subset[OF convex] connected_contains_Icc)
auto
hence f y - f c ≤ (f x - f c) * ((c - y) / (c - x)) by simp
also have (c - y) / (c - x) = (y - c) / (x - c) using y xc by (simp add:
field_simps)
finally show (f y - f c) / (y - c) ≥ (f x - f c) / (x - c) using y xc
by (simp add: field_split_simps)
qed
hence eventually (λy. (f y - f c) / (y - c) ≥ (f x - f c) / (x - c)) (at c within
?A')
by (blast intro: filter_leD at_le)
ultimately have f' ≥ (f x - f c) / (x - c) by (simp add: tendsto_lowerbound)
thus ?thesis using xc by (simp add: field_simps)
qed simp_all

```

6.17.19 Partial derivatives

```

lemma eventually_at_Pair_within_TimesI1:
fixes x::'a::metric_space
assumes ∀F x' in at x within X. P x'
assumes P x
shows ∀F (x', y') in at (x, y) within X × Y. P x'
proof -
from assms[unfolded eventually_at_topological]
obtain S where S: open S x ∈ S ∧ x'. x' ∈ X ⇒ x' ∈ S ⇒ P x'
by metis
show ∀F (x', y') in at (x, y) within X × Y. P x'
unfolding eventually_at_topological
by (auto intro!: exI[where x=S × UNIV] S open_Times)
qed

```

```

lemma eventually_at_Pair_within_TimesI2:
fixes x::'a::metric_space
assumes ∀F y' in at y within Y. P y' P y

```

shows $\forall_F (x', y')$ in at (x, y) within $X \times Y$. $P y'$
proof –
from *assms*[*unfolded eventually_at_topological*]
obtain S **where** S : *open* $S y \in S \wedge y'. y' \in Y \implies y' \in S \implies P y'$
by *metis*
show $\forall_F (x', y')$ in at (x, y) within $X \times Y$. $P y'$
unfolding *eventually_at_topological*
by (*auto intro!*: *exI*[**where** $x = UNIV \times S$] *S open_Times*)
qed

proposition *has_derivative_partialsI*:

fixes $f::'a::real_normed_vector \Rightarrow 'b::real_normed_vector \Rightarrow 'c::real_normed_vector$
assumes fx : $((\lambda x. f x y)$ *has_derivative* fx) (at x within X)
assumes fy : $\bigwedge x y. x \in X \implies y \in Y \implies ((\lambda y. f x y)$ *has_derivative* *blinfun_apply* $(fy x y)$) (at y within Y)
assumes fy_cont [*unfolded continuous_within*]: *continuous* (at (x, y) within $X \times Y$) $(\lambda(x, y). fy x y)$
assumes $y \in Y$ *convex* Y
shows $((\lambda(x, y). f x y)$ *has_derivative* $(\lambda(tx, ty). fx tx + fy x y ty)$) (at (x, y) within $X \times Y$)
proof (*safe intro!*: *has_derivativeI tendstoI, goal_cases*)
case (2 e')
interpret fx : *bounded_linear* fx **using** fx **by** (*rule has_derivative_bounded_linear*)
define e **where** $e = e' / 9$
have $e > 0$ **using** $\langle e' > 0 \rangle$ **by** (*simp add: e_def*)

from fy_cont [*THEN tendstoD, OF* $\langle e > 0 \rangle$]
have $\forall_F (x', y')$ in at (x, y) within $X \times Y$. $dist (fy x' y') (fy x y) < e$
by (*auto simp: split_beta'*)
from *this*[*unfolded eventually_at*] **obtain** d' **where**
 $d' > 0$
 $\bigwedge x' y'. x' \in X \implies y' \in Y \implies (x', y') \neq (x, y) \implies dist (x', y') (x, y) < d'$
 \implies
 $dist (fy x' y') (fy x y) < e$
by *auto*
then
have d' : $x' \in X \implies y' \in Y \implies dist (x', y') (x, y) < d' \implies dist (fy x' y') (fy x y) < e$
for $x' y'$
using $\langle 0 < e \rangle$
by (*cases* $(x', y') = (x, y)$) *auto*
define d **where** $d = d' / sqrt 2$
have $d > 0$ **using** $\langle 0 < d' \rangle$ **by** (*simp add: d_def*)
have d : $x' \in X \implies y' \in Y \implies dist x' x < d \implies dist y' y < d \implies dist (fy x' y') (fy x y) < e$
for $x' y'$
by (*auto simp: dist_prod_def d_def intro!*: *d' real_sqrt_sum_squares_less*)

let $?S = ball y d \cap Y$

```

have convex ?S
  by (auto intro!: convex_Int ‹convex Y›)
{
  fix x'::'a and y'::'b
  assume x': x' ∈ X and y': y' ∈ Y
  assume dx': dist x' x < d and dy': dist y' y < d
  have norm (fy x' y' - fy x' y) ≤ dist (fy x' y') (fy x y) + dist (fy x' y) (fy x y)
    by norm
  also have dist (fy x' y') (fy x y) < e
    by (rule d; fact)
  also have dist (fy x' y) (fy x y) < e
    by (auto intro!: d simp: dist_prod_def x' ‹d > 0› ‹y ∈ Y› dx')
  finally
  have norm (fy x' y' - fy x' y) < e + e
    by arith
  then have onorm (blinfun_apply (fy x' y') - blinfun_apply (fy x' y)) < e + e
    by (auto simp: norm_blinfun_rep_eq blinfun_diff_left[abs_def] fun_diff_def)
} note onorm = this

have ev_mem: ∀F (x', y') in at (x, y) within X × Y. (x', y') ∈ X × Y
  using ‹y ∈ Y›
  by (auto simp: eventually_at intro!: zero_less_one)
moreover
have ev_dist: ∀F xy in at (x, y) within X × Y. dist xy (x, y) < d if d > 0 for
d
  using eventually_at_ball[OF that]
  by (rule eventually_elim2) (auto simp: dist_commute intro!: eventually_True)
note ev_dist[OF ‹0 < d›]
ultimately
have ∀F (x', y') in at (x, y) within X × Y.
  norm (f x' y' - f x' y - (fy x' y) (y' - y)) ≤ norm (y' - y) * (e + e)
proof (eventually_elim, safe)
  fix x' y'
  assume x' ∈ X and y': y' ∈ Y
  assume dist: dist (x', y') (x, y) < d
  then have dx: dist x' x < d and dy: dist y' y < d
    unfolding dist_prod_def fst_conv snd_conv atomize_conj
  by (metis le_less_trans real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
  {
    fix t::real
    assume t ∈ {0 .. 1}
    then have y + t *R (y' - y) ∈ closed_segment y y'
      by (auto simp: closed_segment_def algebra_simps intro!: exI[where x=t])
    also
    have ... ⊆ ball y d ∩ Y
      using ‹y ∈ Y› ‹0 < d› dy y'
      by (intro ‹convex ?S›[unfolded convex_contains_segment, rule_format, of
y y'])
      (auto simp: dist_commute)
  }

```

```

finally have  $y + t *_{R} (y' - y) \in ?S$  .
} note  $seg = this$ 

have  $\bigwedge x. x \in ball\ y\ d \cap Y \implies onorm (blinfun\_apply (fy\ x'\ x) - blinfun\_apply (fy\ x'\ y)) \leq e + e$ 
by (safe intro!: onorm less_imp_le  $\langle x' \in X \rangle dx$ ) (auto simp: dist_commute  $\langle 0 < d \rangle \langle y \in Y \rangle$ )
with  $seg\ has\_derivative\_subset[OF\ assms(2)[OF\ \langle x' \in X \rangle]]$ 
show  $norm (f\ x'\ y' - f\ x'\ y - (fy\ x'\ y) (y' - y)) \leq norm (y' - y) * (e + e)$ 
by (rule differentiable_bound_linearization[where  $S=?S$ ])
(auto intro!:  $\langle 0 < d \rangle \langle y \in Y \rangle$ )
qed
moreover
let  $?le = \lambda x'. norm (f\ x'\ y - f\ x\ y - (fx) (x' - x)) \leq norm (x' - x) * e$ 
from  $fx[unfolding\ has\_derivative\_within, THEN\ conjunct2, THEN\ tendstoD, OF\ \langle 0 < e \rangle]$ 
have  $\forall_F\ x'\ in\ at\ x\ within\ X. ?le\ x'$ 
by eventually_elim (simp,
simp add: dist_norm field_split_simps split: if_split_asm)
then have  $\forall_F (x', y')\ in\ at\ (x, y)\ within\ X \times Y. ?le\ x'$ 
by (rule eventually_at_Pair_within_TimesI1)
(simp add: blinfun.bilinear_simps)
moreover have  $\forall_F (x', y')\ in\ at\ (x, y)\ within\ X \times Y. norm ((x', y') - (x, y)) \neq 0$ 
unfolding norm_eq_zero right_minus_eq
by (auto simp: eventually_at intro!: zero_less_one)
moreover
from  $fy\_cont[THEN\ tendstoD, OF\ \langle 0 < e \rangle]$ 
have  $\forall_F\ x'\ in\ at\ x\ within\ X. norm (fy\ x'\ y - fy\ x\ y) < e$ 
unfolding eventually_at
using  $\langle y \in Y \rangle$ 
by (auto simp: dist_prod_def dist_norm)
then have  $\forall_F (x', y')\ in\ at\ (x, y)\ within\ X \times Y. norm (fy\ x'\ y - fy\ x\ y) < e$ 
by (rule eventually_at_Pair_within_TimesI1)
(simp add: blinfun.bilinear_simps  $\langle 0 < e \rangle$ )
ultimately
have  $\forall_F (x', y')\ in\ at\ (x, y)\ within\ X \times Y.$ 

$$\frac{norm ((f\ x'\ y' - f\ x\ y - (fx (x' - x) + fy\ x\ y (y' - y))) / R}{norm ((x', y') - (x, y))} < e'$$

proof (eventually_elim, safe)
fix  $x'\ y'$ 
have  $norm (f\ x'\ y' - f\ x\ y - (fx (x' - x) + fy\ x\ y (y' - y))) \leq$ 

$$norm (f\ x'\ y' - f\ x'\ y - fy\ x'\ y (y' - y)) +$$


$$norm (fy\ x\ y (y' - y) - fy\ x'\ y (y' - y)) +$$


$$norm (f\ x'\ y - f\ x\ y - fx (x' - x))$$

by norm
also
assume  $nz: norm ((x', y') - (x, y)) \neq 0$ 

```

```

    and nfy: norm (fy x' y - fy x y) < e
  assume norm (f x' y' - f x' y - blinfun_apply (fy x' y) (y' - y)) ≤ norm (y'
- y) * (e + e)
  also assume norm (f x' y - f x y - (fx) (x' - x)) ≤ norm (x' - x) * e
  also
  have norm ((fy x y) (y' - y) - (fy x' y) (y' - y)) ≤ norm ((fy x y) - (fy x'
y)) * norm (y' - y)
  by (auto simp: blinfun.bilinear_simps[symmetric] intro!: norm_blinfun)
  also have ... ≤ (e + e) * norm (y' - y)
  using ‹e > 0› nfy
  by (auto simp: norm_minus_commute intro!: mult_right_mono)
  also have norm (x' - x) * e ≤ norm (x' - x) * (e + e)
  using ‹0 < e› by simp
  also have norm (y' - y) * (e + e) + (e + e) * norm (y' - y) + norm (x' -
x) * (e + e) ≤
    (norm (y' - y) + norm (x' - x)) * (4 * e)
  using ‹e > 0›
  by (simp add: algebra_simps)
  also have ... ≤ 2 * norm ((x', y') - (x, y)) * (4 * e)
  using ‹0 < e› real_sqrt_sum_squares_ge1[of norm (x' - x) norm (y' - y)]
    real_sqrt_sum_squares_ge2[of norm (y' - y) norm (x' - x)]
  by (auto intro!: mult_right_mono simp: norm_prod_def
    simp del: real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
  also have ... ≤ norm ((x', y') - (x, y)) * (8 * e)
  by simp
  also have ... < norm ((x', y') - (x, y)) * e'
  using ‹0 < e'› nz
  by (auto simp: e_def)
  finally show norm ((f x' y' - f x y - (fx (x' - x) + fy x y (y' - y))) /R norm
((x', y') - (x, y))) < e'
  by (simp add: dist_norm) (auto simp add: field_split_simps)
qed
then show ?case
  by eventually_elim (auto simp: dist_norm field_simps)
next
  from has_derivative_bounded_linear[OF fx]
  obtain fxb where fx = blinfun_apply fxb
  by (metis bounded_linear_Blinfun_apply)
  then show bounded_linear (λ(tx, ty). fx tx + blinfun_apply (fy x y) ty)
  by (auto intro!: bounded_linear_intros simp: split_beta')
qed

```

6.17.20 Differentiable case distinction

lemma *has_derivative_within>If_eq*:

```

((λx. if P x then f x else g x) has_derivative f') (at x within S) =
  (bounded_linear f' ∧
  ((λy.(if P y then (f y - ((if P x then f x else g x) + f' (y - x)))/R norm (y
- x)

```

```

      else (g y - ((if P x then f x else g x) + f' (y - x)))/R norm (y - x))
    → 0) (at x within S))
(is _ = (_ ∧ (?if → 0) _))
proof -
  have (λy. (1 / norm (y - x)) *R
    ((if P y then f y else g y) -
    ((if P x then f x else g x) + f' (y - x)))) = ?if
  by (auto simp: inverse_eq_divide)
  thus ?thesis by (auto simp: has_derivative_within)
qed

```

lemma *has_derivative_If_within_closures*:

```

assumes f': x ∈ S ∪ (closure S ∩ closure T) ⇒
  (f has_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
assumes g': x ∈ T ∪ (closure S ∩ closure T) ⇒
  (g has_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
assumes connect: x ∈ closure S ⇒ x ∈ closure T ⇒ f x = g x
assumes connect': x ∈ closure S ⇒ x ∈ closure T ⇒ f' x = g' x
assumes x_in: x ∈ S ∪ T
shows ((λx. if x ∈ S then f x else g x) has_derivative
  (if x ∈ S then f' x else g' x)) (at x within (S ∪ T))
proof -
  from f' x_in interpret f': bounded_linear if x ∈ S then f' x else (λx. 0)
  by (auto simp add: has_derivative_within)
  from g' interpret g': bounded_linear if x ∈ T then g' x else (λx. 0)
  by (auto simp add: has_derivative_within)
  have bl: bounded_linear (if x ∈ S then f' x else g' x)
  using f'.scaleR f'.bounded f'.add g'.scaleR g'.bounded g'.add x_in
  by (unfold locales; force)
  show ?thesis
  using f' g' closure_subset[of T] closure_subset[of S]
  unfolding has_derivative_within_If_eq
  by (intro conjI bl tendsto_If_within_closures x_in)
  (auto simp: has_derivative_within inverse_eq_divide connect connect' sub-
  setD)
qed

```

lemma *has_vector_derivative_If_within_closures*:

```

assumes x_in: x ∈ S ∪ T
assumes u = S ∪ T
assumes f': x ∈ S ∪ (closure S ∩ closure T) ⇒
  (f has_vector_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
assumes g': x ∈ T ∪ (closure S ∩ closure T) ⇒
  (g has_vector_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
assumes connect: x ∈ closure S ⇒ x ∈ closure T ⇒ f x = g x
assumes connect': x ∈ closure S ⇒ x ∈ closure T ⇒ f' x = g' x
shows ((λx. if x ∈ S then f x else g x) has_vector_derivative
  (if x ∈ S then f' x else g' x)) (at x within u)
unfolding has_vector_derivative_def assms

```

```

using  $x$  in  $f'$   $g'$ 
by (intro has_derivative_If_within_closures[where  $?f' = \lambda x a. a *_{\mathbb{R}} f' x$  and
 $?g' = \lambda x a. a *_{\mathbb{R}} g' x$ ,
  THEN has_derivative_eq_rhs]; force simp: assms has_vector_derivative_def)

```

6.17.21 The Inverse Function Theorem

```

lemma linear_injective_contraction:
  assumes linear  $f$   $c < 1$  and  $le: \bigwedge x. \text{norm } (f x - x) \leq c * \text{norm } x$ 
  shows inj  $f$ 
  unfolding linear_injective_0[OF  $\langle$ linear  $f$  $\rangle$ ]
proof safe
  fix  $x$ 
  assume  $f x = 0$ 
  with  $le$  [of  $x$ ] have  $\text{norm } x \leq c * \text{norm } x$ 
  by simp
  then show  $x = 0$ 
  using  $\langle c < 1 \rangle$  by (simp add: mult_le_cancel_right1)
qed

```

From an online proof by J. Michael Boardman, Department of Mathematics,
Johns Hopkins University

```

lemma inverse_function_theorem_scaled:
  fixes  $f::'a::\text{euclidean\_space} \Rightarrow 'a$ 
  and  $f'::'a \Rightarrow ('a \Rightarrow_L 'a)$ 
  assumes open  $U$ 
  and  $derf: \bigwedge x. x \in U \implies (f \text{ has\_derivative } \text{blinfun\_apply } (f' x)) (at x)$ 
  and  $contf: \text{continuous\_on } U f'$ 
  and  $0 \in U$  and [simp]:  $f 0 = 0$ 
  and  $id: f' 0 = id\_blinfun$ 
  obtains  $U' V g g'$  where open  $U'$   $U' \subseteq U$   $0 \in U'$  open  $V$   $0 \in V$  homeomorphism
 $U' V f g$ 
   $\bigwedge y. y \in V \implies (g \text{ has\_derivative } (g' y)) (at y)$ 
   $\bigwedge y. y \in V \implies g' y = inv (\text{blinfun\_apply } (f'(g y)))$ 
   $\bigwedge y. y \in V \implies bij (\text{blinfun\_apply } (f'(g y)))$ 

```

```

proof –
  obtain  $d1$  where  $cball\ 0\ d1 \subseteq U$   $d1 > 0$ 
  using  $\langle$ open  $U$  $\rangle$   $\langle 0 \in U \rangle$  open_contains_cball by blast
  obtain  $d2$  where  $d2: \bigwedge x. [x \in U; \text{dist } x\ 0 \leq d2] \implies \text{dist } (f' x) (f' 0) < 1/2$ 
 $0 < d2$ 
  using continuous_onE [OF  $contf$ , of  $0\ 1/2$ ] by (metis  $\langle 0 \in U \rangle$  half_gt_zero_iff
zero_less_one)
  obtain  $\delta$  where  $le: \bigwedge x. \text{norm } x \leq \delta \implies \text{dist } (f' x) id\_blinfun \leq 1/2$  and  $0 <$ 
 $\delta$ 
  and  $subU: cball\ 0\ \delta \subseteq U$ 
proof
  show  $\min\ d1\ d2 > 0$ 
  by (simp add:  $\langle 0 < d1 \rangle$   $\langle 0 < d2 \rangle$ )
  show  $cball\ 0 (\min\ d1\ d2) \subseteq U$ 

```

```

    using ‹cball 0 d1 ⊆ U› by auto
    show dist (f' x) id_blinfun ≤ 1/2 if norm x ≤ min d1 d2 for x
    using ‹cball 0 d1 ⊆ U› d2 that id by fastforce
  qed
  let ?D = cball 0 δ
  define V:: 'a set where V ≡ ball 0 (δ/2)
  have 4: norm (f (x + h) - f x - h) ≤ 1/2 * norm h
    if x ∈ ?D x+h ∈ ?D for x h
  proof -
    let ?w = λx. f x - x
    have B: ∧x. x ∈ ?D ⇒ onorm (blinfun_apply (f' x - id_blinfun)) ≤ 1/2
      by (metis dist_norm le mem_cball_0 norm_blinfun.rep_eq)
    have ∧x. x ∈ ?D ⇒ (?w has_derivative (blinfun_apply (f' x - id_blinfun)))
      (at x)
      by (rule derivative_eq_intros derf subsetD [OF subU] | force simp: blin-
        fun.diff_left)+
    then have Dw: ∧x. x ∈ ?D ⇒ (?w has_derivative (blinfun_apply (f' x -
      id_blinfun))) (at x within ?D)
      using has_derivative_at_withinI by blast
    have norm (?w (x+h) - ?w x) ≤ (1/2) * norm h
      using differentiable_bound [OF convex_cball Dw B] that by fastforce
    then show ?thesis
      by (auto simp: algebra_simps)
  qed
  have for_g: ∃!x. norm x < δ ∧ f x = y if y: norm y < δ/2 for y
  proof -
    let ?u = λx. x + (y - f x)
    have *: norm (?u x) < δ if x ∈ ?D for x
    proof -
      have fxx: norm (f x - x) ≤ δ/2
        using 4 [of 0 x] ‹0 < δ› ‹f 0 = 0› that by auto
      have norm (?u x) ≤ norm y + norm (f x - x)
      by (metis add.commute add_diff_eq norm_minus_commute norm_triangle_ineq)
      also have ... < δ/2 + δ/2
        using fxx y by auto
      finally show ?thesis
        by simp
    qed
    have ∃!x ∈ ?D. ?u x = x
  proof (rule banach_fix)
    show cball 0 δ ≠ {}
      using ‹0 < δ› by auto
    show (λx. x + (y - f x)) ' cball 0 δ ⊆ cball 0 δ
      using * by force
    have dist (x + (y - f x)) (xh + (y - f xh)) * 2 ≤ dist x xh
      if norm x ≤ δ and norm xh ≤ δ for x xh
      using that 4 [of x xh-x] by (auto simp: dist_norm norm_minus_commute
        algebra_simps)
    then show ∀x ∈ cball 0 δ. ∀y ∈ cball 0 δ. dist (x + (y - f x)) (y + (y - f

```



```

ya)) ≤ (1/2) * dist x ya
  by auto
qed (auto simp: complete_eq_closed)
then show ?thesis
  by (metis * add_cancel_right_right eq_iff_diff_eq_0 le_less mem_cball_0)
qed
define g where g ≡ λy. THE x. norm x < δ ∧ f x = y
have g: norm (g y) < δ ∧ f (g y) = y if norm y < δ/2 for y
  unfolding g_def using that theI' [OF for_g] by meson
then have fg[simp]: f (g y) = y if y ∈ V for y
  using that by (auto simp: V_def)
have 5: norm (g y' - g y) ≤ 2 * norm (y' - y) if y ∈ V y' ∈ V for y y'
proof -
  have no: norm (g y) ≤ δ norm (g y') ≤ δ and [simp]: f (g y) = y
    using that g unfolding V_def by force+
  have norm (g y' - g y) ≤ norm (g y' - g y - (y' - y)) + norm (y' - y)
    by (simp add: add.commute norm_triangle_sub)
  also have ... ≤ (1/2) * norm (g y' - g y) + norm (y' - y)
    using 4 [of g y g y' - g y] that no by (simp add: g_norm_minus_commute
V_def)
  finally show ?thesis
    by auto
qed
have contg: continuous_on V g
proof
  fix y::'a and e::real
  assume 0 < e and y: y ∈ V
  show ∃ d>0. ∀ x'∈V. dist x' y < d → dist (g x') (g y) ≤ e
  proof (intro exI conjI ballI impI)
    show 0 < e/2
      by (simp add: ‹0 < e›)
    qed (use 5 y in ‹force simp: dist_norm›)
qed
show thesis
proof
  define U' where U' ≡ (f - ` V) ∩ ball 0 δ
  have contf: continuous_on U f
  using derf has_derivative_at_withinI by (fast intro: has_derivative_continuous_on)
  then have continuous_on (ball 0 δ) f
    by (meson ball_subset_cball continuous_on_subset subU)
  then show open U'
    by (simp add: U'_def V_def Int_commute continuous_open_preimage)
  show 0 ∈ U' U' ⊆ U open V 0 ∈ V
    using ‹0 < δ› subU by (auto simp: U'_def V_def)
  show hom: homeomorphism U' V f g
proof
  show continuous_on U' f
    using ‹U' ⊆ U› contf continuous_on_subset by blast
  show continuous_on V g

```

```

    using contg by blast
  show f ' U' ⊆ V
    using U'_def by blast
  show g ' V ⊆ U'
    by (simp add: U'_def V_def g image_subset_iff)
  show g (f x) = x if x ∈ U' for x
    by (metis that fg Int_iff U'_def V_def for_g g mem_ball_0 vimage_eq)
  show f (g y) = y if y ∈ V for y
    using that by (simp add: g V_def)
qed
show bij: bij (blinfun_apply (f'(g y))) if y ∈ V for y
proof -
  have inj: inj (blinfun_apply (f' (g y)))
  proof (rule linear_injective_contraction)
    show linear (blinfun_apply (f' (g y)))
      using blinfun.bounded_linear_right bounded_linear_def by blast
  next
  fix x
  have norm (blinfun_apply (f' (g y)) x - x) = norm (blinfun_apply (f' (g
y) - id_blinfun) x)
    by (simp add: blinfun.diff_left)
  also have ... ≤ norm (f' (g y) - id_blinfun) * norm x
    by (rule norm_blinfun)
  also have ... ≤ (1/2) * norm x
  proof (rule mult_right_mono)
    show norm (f' (g y) - id_blinfun) ≤ 1/2
      using that g [of y] le by (auto simp: V_def dist_norm)
  qed auto
  finally show norm (blinfun_apply (f' (g y)) x - x) ≤ (1/2) * norm x .
qed auto
moreover
have surj (blinfun_apply (f' (g y)))
  using blinfun.bounded_linear_right bounded_linear_def
  by (blast intro!: linear_inj_imp_surj [OF inj])
ultimately show ?thesis
  using bijI by blast
qed
define g' where g' ≡ λy. inv (blinfun_apply (f'(g y)))
show (g has_derivative g' y) (at y) if y ∈ V for y
proof -
  have gy: g y ∈ U
    using g subU that unfolding V_def by fastforce
  obtain e where e: ∧h. f (g y + h) = y + blinfun_apply (f' (g y)) h + e h
  and e0: (λh. norm (e h) / norm h) -0→ 0
    using iffD1 [OF has_derivative_iff_Ex derf [OF gy]] ⟨y ∈ V⟩ by auto
  have [simp]: e 0 = 0
    using e [of 0] that by simp
  let ?INV = inv (blinfun_apply (f' (g y)))
  have inj: inj (blinfun_apply (f' (g y)))

```

```

    using bij bij_betw_def that by blast
  have (g has_derivative g' y) (at y within V)
    unfolding has_derivative_at_within_iff_Ex [OF ⟨y ∈ V⟩ ⟨open V⟩]
  proof
    show blinv: bounded_linear (g' y)
      unfolding g'_def using derf gy inj inj_linear_imp_inv_bounded_linear
  by blast
  define eg where eg ≡ λk. - ?INV (e (g (y+k) - g y))
  have g (y+k) = g y + g' y k + eg k if y + k ∈ V for k
  proof -
    have ?INV k = ?INV (blinfun_apply (f' (g y)) (g (y+k) - g y) + e (g
(y+k) - g y))
      using e [of g(y+k) - g y] that by simp
    then have g (y+k) = g y + ?INV k - ?INV (e (g (y+k) - g y))
      using inj blinv by (simp add: linear_simps g'_def)
    then show ?thesis
      by (auto simp: eg_def g'_def)
  qed
  moreover have (λk. norm (eg k) / norm k) -> 0
  proof (rule Lim_null_comparison)
    let ?g = λk. 2 * onorm ?INV * norm (e (g (y+k) - g y)) / norm (g
(y+k) - g y)
    show ∀_F k in at 0. norm (norm (eg k) / norm k) ≤ ?g k
      unfolding eventually_at_topological
    proof (intro exI conjI ballI impI)
      show open ((+)(-y) ' V)
        using ⟨open V⟩ open_translation by blast
      show 0 ∈ (+)(-y) ' V
        by (simp add: that)
      show norm (norm (eg k) / norm k) ≤ 2 * onorm (inv (blinfun_apply
(f' (g y)))) * norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)
        if k ∈ (+)(-y) ' V k ≠ 0 for k
      proof -
        have y+k ∈ V
          using that by auto
        have norm (norm (eg k) / norm k) ≤ onorm ?INV * norm (e (g (y+k)
- g y)) / norm k
          using blinv g'_def onorm by (force simp: eg_def divide_simps)
        also have ... = (norm (g (y+k) - g y) / norm k) * (onorm ?INV *
(norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)))
          by (simp add: divide_simps)
        also have ... ≤ 2 * (onorm ?INV * (norm (e (g (y+k) - g y)) /
norm (g (y+k) - g y)))
          apply (rule mult_right_mono)
          using 5 [of y y+k] ⟨y ∈ V⟩ ⟨y + k ∈ V⟩ onorm_pos_le [OF blinv]
          apply (auto simp: divide_simps zero_le_mult_iff zero_le_divide_iff
g'_def)
        done
      finally show norm (norm (eg k) / norm k) ≤ 2 * onorm ?INV * norm

```

```

(e (g (y+k) - g y)) / norm (g (y+k) - g y)
  by simp
qed
qed
have 1: (λh. norm (e h) / norm h) -0→ (norm (e 0) / norm 0)
  using e0 by auto
have 2: (λk. g (y+k) - g y) -0→ 0
  using contg ⟨open V⟩ ⟨y ∈ V⟩ LIM_offset_zero_iff LIM_zero_iff
at_within_open continuous_on_def by fastforce
from tendsto_compose [OF 1 2, simplified]
have (λk. norm (e (g (y+k) - g y)) / norm (g (y+k) - g y)) -0→ 0 .
from tendsto_mult_left [OF this] show ?g -0→ 0 by auto
qed
ultimately show ∃ e. (∀ k. y + k ∈ V → g (y+k) = g y + g' y k + e k)
∧ (λk. norm (e k) / norm k) -0→ 0
  by blast
qed
then show ?thesis
  by (metis ⟨open V⟩ at_within_open that)
qed
show g' y = inv (blinfun_apply (f' (g y)))
  if y ∈ V for y
  by (simp add: g'_def)
qed
qed

```

We need all this to justify the scaling and translations.

```

theorem inverse_function_theorem:
  fixes f::'a::euclidean_space ⇒ 'a
  and f'::'a ⇒ ('a ⇒L 'a)
  assumes open U
  and derf: Λx. x ∈ U ⇒ (f has_derivative (blinfun_apply (f' x))) (at x)
  and contf: continuous_on U f'
  and x0 ∈ U
  and invf: invf oL f' x0 = id_blinfun
  obtains U' V g g' where open U' U' ⊆ U x0 ∈ U' open V f x0 ∈ V homeo-
morphism U' V f g
  Λy. y ∈ V ⇒ (g has_derivative (g' y)) (at y)
  Λy. y ∈ V ⇒ g' y = inv (blinfun_apply (f'(g y)))
  Λy. y ∈ V ⇒ bij (blinfun_apply (f'(g y)))
proof -
  have apply1 [simp]: Λi. blinfun_apply invf (blinfun_apply (f' x0) i) = i
  by (metis blinfun_apply_blinfun_compose blinfun_apply_id_blinfun invf)
  have apply2 [simp]: Λi. blinfun_apply (f' x0) (blinfun_apply invf i) = i
  by (metis apply1 bij_inv_eq_iff blinfun_bij1 invf)
  have [simp]: (range (blinfun_apply invf)) = UNIV
  using apply1 surjI by blast
  let ?f = invf ∘ (λx. (f ∘ (+)x0)x - f x0)
  let ?f' = λx. invf oL (f' (x + x0))

```

```

obtain  $U' V g g'$  where  $open\ U'$  and  $U': U' \subseteq (+)(-x0) \text{ ' } U\ 0 \in U'$ 
and  $open\ V\ 0 \in V$  and  $hom: homeomorphism\ U'\ V\ ?f\ g$ 
and  $derg: \bigwedge y. y \in V \implies (g\ has\_derivative\ (g'\ y))\ (at\ y)$ 
and  $g': \bigwedge y. y \in V \implies g'\ y = inv\ (?f'(g\ y))$ 
and  $bij: \bigwedge y. y \in V \implies bij\ (?f'(g\ y))$ 
proof (rule inverse_function_theorem_scaled [of (+)(-x0) \text{ ' } U\ ?f\ ?f'])
show  $ope: open\ ((+)\ (-\ x0)\ \text{ ' } U)$ 
using  $\langle open\ U \rangle\ open\_translation\ by\ blast$ 
show ( $?f\ has\_derivative\ blinfun\_apply\ (?f'\ x)\ (at\ x)$ )
if  $x \in (+)\ (-\ x0)\ \text{ ' } U$  for  $x$ 
using that
apply clarify
apply (rule derf_derivative_eq_intros | simp add: blinfun_compose.rep_eq)+
done
have  $YY: (\lambda x. f'\ (x + x0))\ -u -x0 \rightarrow f'\ u$ 
if  $f'\ -u \rightarrow f'\ u\ u \in U$  for  $u$ 
using that LIM_offset [where k = x0] by (auto simp: algebra_simps)
then have  $continuous\_on\ ((+)\ (-\ x0)\ \text{ ' } U)\ (\lambda x. f'\ (x + x0))$ 
using  $\langle contf\ \langle open\ U \rangle\ Lim\_at\_imp\_Lim\_at\_within$ 
by (fastforce simp: continuous_on_def at_within_open_NO_MATCH ope)
then show  $continuous\_on\ ((+)\ (-\ x0)\ \text{ ' } U)\ ?f'$ 
by (intro continuous_intros simp)
qed (auto simp: invf \langle x0 \in U \rangle)
show thesis
proof
let  $?U' = (+)x0 \text{ ' } U'$ 
let  $?V = ((+)\ (f\ x0)) \circ f'\ x0 \text{ ' } V$ 
let  $?g = (+)x0 \circ g \circ invf \circ (+)(- f\ x0)$ 
let  $?g' = \lambda y. inv\ (blinfun\_apply\ (f'\ (?g\ y)))$ 
show  $oU': open\ ?U'$ 
by (simp add: \langle open\ U' \rangle\ open\_translation)
show  $subU: ?U' \subseteq U$ 
using CompII \langle U' \subseteq (+)\ (-\ x0)\ \text{ ' } U \rangle by auto
show  $x0 \in ?U'$ 
by (simp add: \langle 0 \in U' \rangle)
show  $open\ ?V$ 
using blinfun_bij2 [OF invf]
by (metis \langle open\ V \rangle bij_is_surj blinfun.bounded_linear_right bounded_linear_def
image_comp open_surjective_linear_image open_translation)
show  $f\ x0 \in ?V$ 
using  $\langle 0 \in V \rangle\ image\_iff$  by fastforce
show  $homeomorphism\ ?U'\ ?V\ f\ ?g$ 
proof
show  $continuous\_on\ ?U'\ f$ 
by (meson subU continuous_on_eq_continuous_at derf_has_derivative_continuous
oU' subsetD)
have  $?f \text{ ' } U' \subseteq V$ 
using hom homeomorphism_image1 by blast
then show  $f \text{ ' } ?U' \subseteq ?V$ 

```

```

    unfolding image_subset_iff
  by (clarsimp simp: image_def) (metis apply2 add.commute diff_add_cancel)
show ?g ' ?V  $\subseteq$  ?U'
  using hom_invf by (auto simp: image_def homeomorphism_def)
show ?g (f x) = x
  if x  $\in$  ?U' for x
  using that hom homeomorphism_apply1 by fastforce
have continuous_on V g
  using hom homeomorphism_def by blast
then show continuous_on ?V ?g
  by (intro continuous_intros) (auto elim!: continuous_on_subset)
have fg: ?f (g x) = x if x  $\in$  V for x
  using hom homeomorphism_apply2 that by blast
show f (?g y) = y
  if y  $\in$  ?V for y
  using that fg by (simp add: image_iff) (metis apply2 add.commute
diff_add_cancel)
qed
show (?g has_derivative ?g' y) (at y) bij (blinfun_apply (f' (?g y)))
  if y  $\in$  ?V for y
proof -
  have 1: bij (blinfun_apply invf)
    using blinfun_bij1 invf by blast
  then have 2: bij (blinfun_apply (f' (x0 + g x))) if x  $\in$  V for x
    by (metis add.commute bij_betw_comp_iff2 blinfun_compose.rep_eq
that top_greatest)
  then show bij (blinfun_apply (f' (?g y)))
    using that by auto
  have g' x  $\circ$  blinfun_apply invf = inv (blinfun_apply (f' (x0 + g x)))
    if x  $\in$  V for x
  using that
  by (simp add: g' o_inv_distrib blinfun_compose.rep_eq 1 2 add.commute
bij_is_inj_flip: o_assoc)
  then show (?g has_derivative ?g' y) (at y)
    using that invf
    by clarsimp (rule derg_derivative_eq_intros | simp flip: id_def)+
qed
qed auto
qed

```

6.17.22 Piecewise differentiable functions

definition *piecewise_differentiable_on*

(**infixr** *piecewise'_differentiable'_on* 50)

where *f piecewise_differentiable_on i* \equiv

continuous_on i f \wedge

($\exists S. \text{finite } S \wedge (\forall x \in i - S. f \text{ differentiable (at } x \text{ within } i))$)

lemma *piecewise_differentiable_on_imp_continuous_on*:

f *piecewise_differentiable_on* $S \implies$ *continuous_on* S f
by (*simp add: piecewise_differentiable_on_def*)

lemma *piecewise_differentiable_on_subset*:

f *piecewise_differentiable_on* $S \implies T \leq S \implies f$ *piecewise_differentiable_on*
 T

using *continuous_on_subset*

by (*smt (verit) Diff_iff_differentiable_within_subset in_mono piecewise_differentiable_on_def*)

lemma *differentiable_imp_piecewise_differentiable*:

fixes $a:: 'a::\{linorder_topology,real_normed_vector\}$

shows f *differentiable_on* $\{a..b\} \implies f$ *piecewise_differentiable_on* $\{a..b\}$

using *differentiable_imp_continuous_on differentiable_onD piecewise_differentiable_on_def*

by *fastforce*

lemma *differentiable_imp_piecewise_differentiable*:

$(\bigwedge x. x \in S \implies f$ *differentiable* (at x within S))

$\implies f$ *piecewise_differentiable_on* S

by (*auto simp: piecewise_differentiable_on_def differentiable_imp_continuous_on differentiable_on_def*

intro: differentiable_within_subset)

lemma *piecewise_differentiable_const [iff]*: $(\lambda x. z)$ *piecewise_differentiable_on* S

by (*simp add: differentiable_imp_piecewise_differentiable*)

lemma *piecewise_differentiable_compose*:

$\llbracket f$ *piecewise_differentiable_on* S ; g *piecewise_differentiable_on* $(f \text{ ` } S)$;

$\bigwedge x. \text{finite } (S \cap f \text{ ` } \{x\}) \rrbracket$

$\implies (g \circ f)$ *piecewise_differentiable_on* S

apply (*simp add: piecewise_differentiable_on_def, safe*)

apply (*blast intro: continuous_on_compose2*)

apply (*rename_tac A B*)

apply (*rule_tac x=A \cup ($\bigcup_{x \in B}. S \cap f \text{ ` } \{x\}$) in exI*)

apply (*blast intro!: differentiable_chain_within*)

done

lemma *piecewise_differentiable_affine*:

fixes $m::real$

assumes f *piecewise_differentiable_on* $((\lambda x. m *_{\mathbb{R}} x + c) \text{ ` } S)$

shows $(f \circ (\lambda x. m *_{\mathbb{R}} x + c))$ *piecewise_differentiable_on* S

proof (*cases m = 0*)

case *True*

then show *?thesis*

unfolding *o_def*

by (*force intro: differentiable_imp_piecewise_differentiable differentiable_const*)

next

case *False*

show *?thesis*

apply (*rule piecewise_differentiable_compose [OF differentiable_imp_piecewise_differentiable]*)

1900

apply (rule *assms derivative_intros* | simp add: *False vimage_def real_vector_affinity_eq*) +
done
qed

lemma *piecewise_differentiable_cases*:

fixes *c::real*

assumes *f piecewise_differentiable_on {a..c}*

g piecewise_differentiable_on {c..b}

a ≤ c c ≤ b f c = g c

shows $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$ *piecewise_differentiable_on {a..b}*

proof –

obtain *S T* **where** *st: finite S finite T*

and *fd: $\bigwedge x. x \in \{a..c\} - S \implies f$ differentiable at x within $\{a..c\}$*

and *gd: $\bigwedge x. x \in \{c..b\} - T \implies g$ differentiable at x within $\{c..b\}$*

using *assms*

by (auto simp: *piecewise_differentiable_on_def*)

have *finabc: finite ({a,b,c} \cup (S \cup T))*

by (*metis* \langle finite *S* \rangle \langle finite *T* \rangle *finite_Un finite_insert finite.emptyI*)

have *continuous_on {a..c} f continuous_on {c..b} g*

using *assms piecewise_differentiable_on_def* **by** auto

then have *continuous_on {a..b} $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$*

using *continuous_on_cases [OF closed_real_atLeastAtMost [of a c],*

OF closed_real_atLeastAtMost [of c b],

of f g $\lambda x. x \leq c$] assms

by (*force simp: ivl_disj_un_two_touch*)

moreover

{ **fix** *x*

assume *x: x \in {a..b} - ({a,b,c} \cup (S \cup T))*

have $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$ *differentiable at x within {a..b}* (**is** *?diff_fg*)

proof (*cases x c rule: le_cases*)

case *le* **show** *?diff_fg*

proof (rule *differentiable_transform_within* [**where** *d = dist x c*])

have *f differentiable at x*

using *x le fd [of x] at_within_interior [of x {a..c}]* **by** *simp*

then show *f differentiable at x within {a..b}*

by (*simp add: differentiable_at_withinI*)

qed (*use x le st dist_real_def in auto*)

next

case *ge* **show** *?diff_fg*

proof (rule *differentiable_transform_within* [**where** *d = dist x c*])

have *g differentiable at x*

using *x ge gd [of x] at_within_interior [of x {c..b}]* **by** *simp*

then show *g differentiable at x within {a..b}*

by (*simp add: differentiable_at_withinI*)

qed (*use x ge st dist_real_def in auto*)

qed

}

then have $\exists S. \text{finite } S \wedge$

$(\forall x \in \{a..b\} - S. (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$ *differentiable at x*


```

within {a..b}
  by (meson finabc)
  ultimately show ?thesis
  by (simp add: piecewise_differentiable_on_def)
qed

lemma piecewise_differentiable_neg:
  f piecewise_differentiable_on S  $\implies$  ( $\lambda x. -(f x)$ ) piecewise_differentiable_on S
  by (auto simp: piecewise_differentiable_on_def continuous_on_minus)

lemma piecewise_differentiable_add:
  assumes f piecewise_differentiable_on i
         g piecewise_differentiable_on i
  shows ( $\lambda x. f x + g x$ ) piecewise_differentiable_on i
proof -
  obtain S T where st: finite S finite T
     $\forall x \in i - S. f$  differentiable at  $x$  within  $i$ 
     $\forall x \in i - T. g$  differentiable at  $x$  within  $i$ 
  using assms by (auto simp: piecewise_differentiable_on_def)
  then have finite (S  $\cup$  T)  $\wedge$  ( $\forall x \in i - (S \cup T). (\lambda x. f x + g x)$  differentiable at
 $x$  within  $i$ )
  by auto
  moreover have continuous_on i f continuous_on i g
  using assms piecewise_differentiable_on_def by auto
  ultimately show ?thesis
  by (auto simp: piecewise_differentiable_on_def continuous_on_add)
qed

lemma piecewise_differentiable_diff:
   $\llbracket f$  piecewise_differentiable_on S;  $g$  piecewise_differentiable_on S  $\rrbracket$ 
 $\implies$  ( $\lambda x. f x - g x$ ) piecewise_differentiable_on S
  unfolding diff_conv_add_uminus
  by (metis piecewise_differentiable_add piecewise_differentiable_neg)

```

6.17.23 The concept of continuously differentiable

John Harrison writes as follows:

“The usual assumption in complex analysis texts is that a path γ should be piecewise continuously differentiable, which ensures that the path integral exists at least for any continuous f , since all piecewise continuous functions are integrable. However, our notion of validity is weaker, just piecewise differentiability. . . [namely] continuity plus differentiability except on a finite set. . . [Our] underlying theory of integration is the Kurzweil-Henstock theory. In contrast to the Riemann or Lebesgue theory (but in common with a simple notion based on antiderivatives), this can integrate all derivatives.”

"Formalizing basic complex analysis." From Insight to Proof: Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar and Rhetoric 10.23

(2007): 151-165.

And indeed he does not assume that his derivatives are continuous, but the penalty is unreasonably difficult proofs concerning winding numbers. We need a self-contained and straightforward theorem asserting that all derivatives can be integrated before we can adopt Harrison's choice.

definition $C1_differentiable_on :: (real \Rightarrow 'a::real_normed_vector) \Rightarrow real\ set \Rightarrow bool$

(**infix** $C1'_differentiable'_on$ 50)

where

$f\ C1_differentiable_on\ S \longleftrightarrow$

$(\exists D. (\forall x \in S. (f\ has_vector_derivative\ (D\ x))\ (at\ x)) \wedge continuous_on\ S\ D)$

lemma $C1_differentiable_on_eq:$

$f\ C1_differentiable_on\ S \longleftrightarrow$

$(\forall x \in S. f\ differentiable\ at\ x) \wedge continuous_on\ S\ (\lambda x. vector_derivative\ f\ (at\ x))$

(**is** $?lhs = ?rhs$)

proof

assume $?lhs$

then show $?rhs$

unfolding $C1_differentiable_on_def$

by ($metis$ (no_types , $lifting$) $continuous_on_eq\ differentiableI_vector\ vector_derivative_at$)

next

assume $?rhs$

then show $?lhs$

using $C1_differentiable_on_def\ vector_derivative_works$ **by** $fastforce$

qed

lemma $C1_differentiable_on_subset:$

$f\ C1_differentiable_on\ T \Longrightarrow S \subseteq T \Longrightarrow f\ C1_differentiable_on\ S$

unfolding $C1_differentiable_on_def\ continuous_on_eq\ continuous_within$

by ($blast\ intro:$ $continuous_within_subset$)

lemma $C1_differentiable_compose:$

assumes $fg: f\ C1_differentiable_on\ S\ g\ C1_differentiable_on\ (f\ 'S)$ **and** $fin:$
 $\bigwedge x. finite\ (S \cap f^{-1}\{x\})$

shows $(g \circ f)\ C1_differentiable_on\ S$

proof –

have $\bigwedge x. x \in S \Longrightarrow g \circ f\ differentiable\ at\ x$

by ($meson\ C1_differentiable_on_eq\ assms\ differentiable_chain_at\ imageI$)

moreover have $continuous_on\ S\ (\lambda x. vector_derivative\ (g \circ f)\ (at\ x))$

proof ($rule\ continuous_on_eq\ [of\ _ \lambda x. vector_derivative\ f\ (at\ x) *R\ vector_derivative\ g\ (at\ (f\ x))]$)

show $continuous_on\ S\ (\lambda x. vector_derivative\ f\ (at\ x) *R\ vector_derivative\ g\ (at\ (f\ x)))$

using fg

apply ($clarsimp\ simp\ add:$ $C1_differentiable_on_eq$)

```

apply (rule Limits.continuous_on_scaleR, assumption)
by (metis (mono_tags, lifting) continuous_at_imp_continuous_on con-
tinuous_on_compose continuous_on_cong differentiable_imp_continuous_within
o_def)
show  $\bigwedge x. x \in S \implies \text{vector\_derivative } f \text{ (at } x) *_{\mathbb{R}} \text{vector\_derivative } g \text{ (at } (f$ 
 $x)) = \text{vector\_derivative } (g \circ f) \text{ (at } x)$ 
by (metis (mono_tags, opaque_lifting) C1_differentiable_on_eq fg_imageI
vector_derivative_chain_at)
qed
ultimately show ?thesis
by (simp add: C1_differentiable_on_eq)
qed

```

```

lemma C1_diff_imp_diff:  $f \text{ C1\_differentiable\_on } S \implies f \text{ differentiable\_on } S$ 
by (simp add: C1_differentiable_on_eq differentiable_at_imp_differentiable_on)

```

```

lemma C1_differentiable_on_ident [simp, derivative_intros]:  $(\lambda x. x) \text{ C1\_differentiable\_on } S$ 
by (auto simp: C1_differentiable_on_eq)

```

```

lemma C1_differentiable_on_const [simp, derivative_intros]:  $(\lambda z. a) \text{ C1\_differentiable\_on } S$ 
by (auto simp: C1_differentiable_on_eq)

```

```

lemma C1_differentiable_on_add [simp, derivative_intros]:
 $f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f x + g x)$ 
 $\text{C1\_differentiable\_on } S$ 
unfolding C1_differentiable_on_eq by (auto intro: continuous_intros)

```

```

lemma C1_differentiable_on_minus [simp, derivative_intros]:
 $f \text{ C1\_differentiable\_on } S \implies (\lambda x. - f x) \text{ C1\_differentiable\_on } S$ 
unfolding C1_differentiable_on_eq by (auto intro: continuous_intros)

```

```

lemma C1_differentiable_on_diff [simp, derivative_intros]:
 $f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f x - g x)$ 
 $\text{C1\_differentiable\_on } S$ 
unfolding C1_differentiable_on_eq by (auto intro: continuous_intros)

```

```

lemma C1_differentiable_on_mult [simp, derivative_intros]:
fixes  $f g :: \text{real} \Rightarrow 'a :: \text{real\_normed\_algebra}$ 
shows  $f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f x * g$ 
 $x) \text{ C1\_differentiable\_on } S$ 
unfolding C1_differentiable_on_eq
by (auto simp: continuous_on_add continuous_on_mult continuous_at_imp_continuous_on
differentiable_imp_continuous_within)

```

```

lemma C1_differentiable_on_scaleR [simp, derivative_intros]:
 $f \text{ C1\_differentiable\_on } S \implies g \text{ C1\_differentiable\_on } S \implies (\lambda x. f x *_{\mathbb{R}} g x)$ 
 $\text{C1\_differentiable\_on } S$ 

```

unfolding $C1_differentiable_on_eq$
by (*rule continuous_intros | simp add: continuous_at_imp_continuous_on differentiable_imp_continuous_within*) $+$

lemma $C1_differentiable_on_of_real$ [*derivative_intros*]: $of_real\ C1_differentiable_on\ S$

unfolding $C1_differentiable_on_def$
using *vector_derivative_works* **by** *fastforce*

lemma $C1_differentiable_on_translation$:

$f\ C1_differentiable_on\ U - S \implies (+)\ d \circ f\ C1_differentiable_on\ U - S$
by (*metis C1_differentiable_on_def has_vector_derivative_shift*)

lemma $C1_differentiable_on_translation_eq$:

fixes $d :: 'a::real_normed_vector$
shows $(+)\ d \circ f\ C1_differentiable_on\ i - S \iff f\ C1_differentiable_on\ i - S$
by (*force simp: o_def intro: C1_differentiable_on_translation dest: C1_differentiable_on_translation [of concl: $-d$]*)

definition $piecewise_C1_differentiable_on$

(**infixr** $piecewise'\ C1'\ differentiable'\ on\ 50$)

where $f\ piecewise_C1_differentiable_on\ i \equiv$
 $continuous_on\ i\ f \wedge$
 $(\exists S. finite\ S \wedge (f\ C1_differentiable_on\ (i - S)))$

lemma $C1_differentiable_imp_piecewise$:

$f\ C1_differentiable_on\ S \implies f\ piecewise_C1_differentiable_on\ S$
by (*auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq continuous_at_imp_continuous_on differentiable_imp_continuous_within*)

lemma $piecewise_C1_imp_differentiable$:

$f\ piecewise_C1_differentiable_on\ i \implies f\ piecewise_differentiable_on\ i$
by (*auto simp: piecewise_C1_differentiable_on_def piecewise_differentiable_on_def C1_differentiable_on_def differentiable_def has_vector_derivative_def intro: has_derivative_at_withinI*)

lemma $piecewise_C1_differentiable_on_translation_eq$:

$((+)\ d \circ f\ piecewise_C1_differentiable_on\ i) \iff (f\ piecewise_C1_differentiable_on\ i)$

unfolding $piecewise_C1_differentiable_on_def\ continuous_on_translation_eq$
by (*metis C1_differentiable_on_translation_eq*)

lemma $piecewise_C1_differentiable_compose$ [*derivative_intros*]:

assumes $fg: f\ piecewise_C1_differentiable_on\ S\ g\ piecewise_C1_differentiable_on\ (f\ 'S)$ **and** $fin: \bigwedge x. finite\ (S \cap f\ -\{x\})$

shows $(g \circ f)\ piecewise_C1_differentiable_on\ S$

proof $-$

have $continuous_on\ S\ (\lambda x. g\ (f\ x))$

```

  by (metis continuous_on_compose2 fg order_refl piecewise_C1_differentiable_on_def)
  moreover have  $\exists T. \text{finite } T \wedge g \circ f \text{ C1\_differentiable\_on } S - T$ 
  proof -
    obtain F where finite F and  $F: f \text{ C1\_differentiable\_on } S - F$  and  $f: f \text{ piecewise\_C1\_differentiable\_on } S$ 
    using fg by (auto simp: piecewise_C1_differentiable_on_def)
    obtain G where finite G and  $G: g \text{ C1\_differentiable\_on } f^{-1} S - G$  and  $g: g \text{ piecewise\_C1\_differentiable\_on } f^{-1} S$ 
    using fg by (auto simp: piecewise_C1_differentiable_on_def)
    show ?thesis
    proof (intro exI conjI)
      show finite  $(F \cup (\bigcup_{x \in G}. S \cap f^{-1}\{x\}))$ 
      using fin by (auto simp only: Int_Union ⟨finite F⟩ ⟨finite G⟩ finite_UN finite_imageI)
      show  $g \circ f \text{ C1\_differentiable\_on } S - (F \cup (\bigcup_{x \in G}. S \cap f^{-1}\{x\}))$ 
      apply (rule C1_differentiable_compose)
      apply (blast intro: C1_differentiable_on_subset [OF F])
      apply (blast intro: C1_differentiable_on_subset [OF G])
      by (simp add: C1_differentiable_on_subset G Diff_Int_distrib2 fin)
    qed
  qed
  ultimately show ?thesis
  by (simp add: piecewise_C1_differentiable_on_def)
qed

```

```

lemma piecewise_C1_differentiable_on_subset:
   $f \text{ piecewise\_C1\_differentiable\_on } S \implies T \leq S \implies f \text{ piecewise\_C1\_differentiable\_on } T$ 
  by (auto simp: piecewise_C1_differentiable_on_def elim!: continuous_on_subset C1_differentiable_on_subset)

```

```

lemma C1_differentiable_imp_continuous_on:
   $f \text{ C1\_differentiable\_on } S \implies \text{continuous\_on } S f$ 
  unfolding C1_differentiable_on_eq continuous_on_eq continuous_within
  using differentiable_at_withinI differentiable_imp_continuous_within by blast

```

```

lemma C1_differentiable_on_empty [iff, derivative_intros]:  $f \text{ C1\_differentiable\_on } \{\}$ 
  unfolding C1_differentiable_on_def
  by auto

```

```

lemma piecewise_C1_differentiable_affine:
  fixes  $m::\text{real}$ 
  assumes  $f \text{ piecewise\_C1\_differentiable\_on } ((\lambda x. m * x + c) ^{-1} S)$ 
  shows  $(f \circ (\lambda x. m * x + c)) \text{ piecewise\_C1\_differentiable\_on } S$ 
  proof (cases  $m = 0$ )
  case True
  then show ?thesis
  unfolding o_def by (auto simp: piecewise_C1_differentiable_on_def)

```

```

next
  case False
  have *:  $\bigwedge x. \text{finite } (S \cap \{y. m * y + c = x\})$ 
    using False not_finite_existsD by fastforce
  show ?thesis
  apply (rule piecewise_C1_differentiable_compose [OF C1_differentiable_imp_piecewise])
  apply (rule * assms derivative_intros | simp add: False vimage_def) +
  done
qed

lemma piecewise_C1_differentiable_cases [derivative_intros]:
  fixes c::real
  assumes f piecewise_C1_differentiable_on {a..c}
    g piecewise_C1_differentiable_on {c..b}
     $a \leq c \leq b$   $f c = g c$ 
  shows  $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$  piecewise_C1_differentiable_on {a..b}
proof -
  obtain S T where st: f C1_differentiable_on ({a..c} - S)
    g C1_differentiable_on ({c..b} - T)
    finite S finite T
  using assms
  by (force simp: piecewise_C1_differentiable_on_def)
  then have f_diff: f differentiable_on {a.<c} - S
    and g_diff: g differentiable_on {c.<.b} - T
  by (simp_all add: C1_differentiable_on_eq differentiable_at_withinI differentiable_on_def)
  have continuous_on {a..c} f continuous_on {c..b} g
    using assms piecewise_C1_differentiable_on_def by auto
  then have cab: continuous_on {a..b}  $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$ 
    using continuous_on_cases [OF closed_real_atLeastAtMost [of a c],
      OF closed_real_atLeastAtMost [of c b],
      of f g  $\lambda x. x \leq c$ ] assms
  by (force simp: invl_disj_un_two_touch)
  { fix x
  assume x: x  $\in$  {a..b} - insert c (S  $\cup$  T)
  have  $(\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$  differentiable at x (is ?diff_fg)
  proof (cases x c rule: le_cases)
  case le show ?diff_fg
    apply (rule differentiable_transform_within [where f=f and d = dist x c])
    using x dist_real_def le st by (auto simp: C1_differentiable_on_eq)
  next
  case ge show ?diff_fg
    apply (rule differentiable_transform_within [where f=g and d = dist x
c])
    using dist_nz x dist_real_def ge st x by (auto simp: C1_differentiable_on_eq)
  qed
  }
  then have  $(\forall x \in \{a..b\} - \text{insert } c \ (S \cup T)). (\lambda x. \text{if } x \leq c \text{ then } f x \text{ else } g x)$ 
differentiable at x

```

```

    by auto
  moreover
  { assume fcon: continuous_on ({a<..

```

```

} note * = this
have continuous_on ({a<..<b} - insert c (S ∪ T)) (λx. vector_derivative (λx.
if x ≤ c then f x else g x) (at x))
using st
by (auto simp: C1_differentiable_on_eq elim!: continuous_on_subset intro: *)
ultimately have ∃ S. finite S ∧ ((λx. if x ≤ c then f x else g x) C1_differentiable_on
{a..b} - S)
apply (rule_tac x={a,b,c} ∪ S ∪ T in exI)
using st by (auto simp: C1_differentiable_on_eq elim!: continuous_on_subset)
with cab show ?thesis
by (simp add: piecewise_C1_differentiable_on_def)
qed

```

```

lemma piecewise_C1_differentiable_const [derivative_intros]:
(λx. c) piecewise_C1_differentiable_on S
by (simp add: C1_differentiable_imp_piecewise)

```

```

lemma piecewise_C1_differentiable_scaleR [derivative_intros]:
[[f piecewise_C1_differentiable_on S]]
⇒ (λx. c *R f x) piecewise_C1_differentiable_on S
by (force simp add: piecewise_C1_differentiable_on_def continuous_on_scaleR)

```

```

lemma piecewise_C1_differentiable_neg [derivative_intros]:
f piecewise_C1_differentiable_on S ⇒ (λx. -(f x)) piecewise_C1_differentiable_on
S
unfolding piecewise_C1_differentiable_on_def
by (auto intro!: continuous_on_minus C1_differentiable_on_minus)

```

```

lemma piecewise_C1_differentiable_add [derivative_intros]:
assumes f piecewise_C1_differentiable_on i
g piecewise_C1_differentiable_on i
shows (λx. f x + g x) piecewise_C1_differentiable_on i
proof -
obtain S t where st: finite S finite t
f C1_differentiable_on (i-S)
g C1_differentiable_on (i-t)
using assms by (auto simp: piecewise_C1_differentiable_on_def)
then have finite (S ∪ t) ∧ (λx. f x + g x) C1_differentiable_on i - (S ∪ t)
by (auto intro: C1_differentiable_on_add elim!: C1_differentiable_on_subset)
moreover have continuous_on i f continuous_on i g
using assms piecewise_C1_differentiable_on_def by auto
ultimately show ?thesis
by (auto simp: piecewise_C1_differentiable_on_def continuous_on_add)
qed

```

```

lemma piecewise_C1_differentiable_diff [derivative_intros]:
[[f piecewise_C1_differentiable_on S; g piecewise_C1_differentiable_on S]]
⇒ (λx. f x - g x) piecewise_C1_differentiable_on S
unfolding diff_conv_add_uminus

```


by (metis piecewise_C1_differentiable_add piecewise_C1_differentiable_neg)

lemma piecewise_C1_differentiable_cmult_right [derivative_intros]:

fixes $c::\text{complex}$

shows f piecewise_C1_differentiable_on S

$\implies (\lambda x. f x * c)$ piecewise_C1_differentiable_on S

by (force simp: piecewise_C1_differentiable_on_def continuous_on_mult_right)

lemma piecewise_C1_differentiable_cmult_left [derivative_intros]:

fixes $c::\text{complex}$

shows f piecewise_C1_differentiable_on S

$\implies (\lambda x. c * f x)$ piecewise_C1_differentiable_on S

using piecewise_C1_differentiable_cmult_right [of $f S c$] by (simp add: mult.commute)

lemma piecewise_C1_differentiable_on_of_real [derivative_intros]:

of_real piecewise_C1_differentiable_on S

by (simp add: C1_differentiable_imp_piecewise C1_differentiable_on_of_real)

end

6.18 Finite Cartesian Products of Euclidean Spaces

theory Cartesian_Euclidean_Space

imports Derivative

begin

lemma subspace_special_hyperplane: subspace $\{x. x \$ k = 0\}$

by (simp add: subspace_def)

lemma sum_mult_product:

$\text{sum } h \{..\langle A * B :: \text{nat} \} = (\sum i \in \{..\langle A\}. \sum j \in \{..\langle B\}. h (j + i * B))$

unfolding sum.nat_group [of $h B A$, unfolded atLeast0LessThan, symmetric]

proof (rule sum.cong, simp, rule sum.reindex_cong)

fix i

show $\text{inj_on } (\lambda j. j + i * B) \{..\langle B\}$ by (auto intro!: inj_onI)

show $\{i * B .. \langle i * B + B\} = (\lambda j. j + i * B) ' \{..\langle B\}$

proof safe

fix j assume $j \in \{i * B .. \langle i * B + B\}$

then show $j \in (\lambda j. j + i * B) ' \{..\langle B\}$

by (auto intro!: image_eqI [of $_ _ j - i * B$])

qed simp

qed simp

lemma interval_cbox_cart: $\{a::\text{real}^n..b\} = \text{cbox } a b$

by (auto simp add: less_eq_vec_def mem_box Basis_vec_def inner_axis)

lemma differentiable_vec:

fixes $S :: 'a::\text{euclidean_space}$ set

shows vec differentiable_on S

1910

by (simp add: linear_linear bounded_linear_imp_differentiable_on)

lemma *continuous_vec* [continuous_intros]:

fixes $x :: 'a::\text{euclidean_space}$

shows *isCont* *vec* x

apply (clarsimp simp add: continuous_def LIM_def dist_vec_def L2_set_def)

apply (rule_tac $x=r / \text{sqrt}(\text{real } \text{CARD}(b))$ in *exI*)

by (simp add: mult.commute pos_less_divide_eq real_sqrt_mult)

lemma *box_vec_eq_empty* [simp]:

shows $\text{cbox}(\text{vec } a)(\text{vec } b) = \{\}$ \longleftrightarrow $\text{cbox } a b = \{\}$

$\text{box}(\text{vec } a)(\text{vec } b) = \{\}$ \longleftrightarrow $\text{box } a b = \{\}$

by (auto simp: Basis_vec_def mem_box box_eq_empty inner_axis)

6.18.1 Closures and interiors of halfspaces

lemma *interior_halfspace_component_le* [simp]:

$\text{interior } \{x. x\$k \leq a\} = \{x :: (\text{real}^n). x\$k < a\}$ (is ?LE)

and *interior_halfspace_component_ge* [simp]:

$\text{interior } \{x. x\$k \geq a\} = \{x :: (\text{real}^n). x\$k > a\}$ (is ?GE)

proof –

have *axis* $k(1::\text{real}) \neq 0$

by (simp add: axis_def vec_eq_iff)

moreover have *axis* $k(1::\text{real}) \cdot x = x\k for x

by (simp add: cart_eq_inner_axis inner_commute)

ultimately show ?LE ?GE

using *interior_halfspace_le* [of *axis* $k(1::\text{real}) a$]

interior_halfspace_ge [of *axis* $k(1::\text{real}) a$] by auto

qed

lemma *closure_halfspace_component_lt* [simp]:

$\text{closure } \{x. x\$k < a\} = \{x :: (\text{real}^n). x\$k \leq a\}$ (is ?LE)

and *closure_halfspace_component_gt* [simp]:

$\text{closure } \{x. x\$k > a\} = \{x :: (\text{real}^n). x\$k \geq a\}$ (is ?GE)

proof –

have *axis* $k(1::\text{real}) \neq 0$

by (simp add: axis_def vec_eq_iff)

moreover have *axis* $k(1::\text{real}) \cdot x = x\k for x

by (simp add: cart_eq_inner_axis inner_commute)

ultimately show ?LE ?GE

using *closure_halfspace_lt* [of *axis* $k(1::\text{real}) a$]

closure_halfspace_gt [of *axis* $k(1::\text{real}) a$] by auto

qed

lemma *interior_standard_hyperplane*:

$\text{interior } \{x :: (\text{real}^n). x\$k = a\} = \{\}$

proof –

have *axis* $k(1::\text{real}) \neq 0$

by (simp add: axis_def vec_eq_iff)

moreover have $\text{axis } k (1::\text{real}) \cdot x = x\k **for** x
by (*simp add: cart_eq_inner_axis inner_commute*)
ultimately show *?thesis*
using *interior_hyperplane [of axis k (1::real) a]*
by force
qed

lemma *matrix_vector_mul_bounded_linear*[*intro, simp*]: *bounded_linear* $((*v) A)$
for $A :: 'a::\{\text{euclidean_space, real_algebra_1}\}^n{}^m$
using *matrix_vector_mul_linear*[*of A*]
by (*simp add: linear_conv_bounded_linear linear_matrix_vector_mul_eq*)

lemma
fixes $A :: 'a::\{\text{euclidean_space, real_algebra_1}\}^n{}^m$
shows *matrix_vector_mult_linear_continuous_at* [*continuous_intros*]: *isCont*
 $((*v) A) z$
and *matrix_vector_mult_linear_continuous_on* [*continuous_intros*]: *continuous_on* $S ((*v) A)$
by (*simp_all add: linear_continuous_at linear_continuous_on*)

6.18.2 Bounds on components etc. relative to operator norm

lemma *norm_column_le_onorm*:

fixes $A :: \text{real}^n{}^m$
shows $\text{norm}(\text{column } i A) \leq \text{onorm}((*v) A)$

proof –

have $\text{norm}(\chi j. A \$ j \$ i) \leq \text{norm}(A *v \text{axis } i 1)$
by (*simp add: matrix_mult_dot_cart_eq_inner_axis*)
also have $\dots \leq \text{onorm}((*v) A)$
using *onorm [OF matrix_vector_mul_bounded_linear, of A axis i 1]* **by auto**
finally have $\text{norm}(\chi j. A \$ j \$ i) \leq \text{onorm}((*v) A)$.
then show *?thesis*
unfolding *column_def* .

qed

lemma *matrix_component_le_onorm*:

fixes $A :: \text{real}^n{}^m$
shows $|A \$ i \$ j| \leq \text{onorm}((*v) A)$

proof –

have $|A \$ i \$ j| \leq \text{norm}(\chi n. (A \$ n \$ j))$
by (*metis (full_types, lifting) component_le_norm_cart_vec_lambda_beta*)
also have $\dots \leq \text{onorm}((*v) A)$
by (*metis (no_types) column_def norm_column_le_onorm*)
finally show *?thesis* .

qed

lemma *component_le_onorm*:

fixes $f :: \text{real}^m \Rightarrow \text{real}^n$
shows $\text{linear } f \implies |\text{matrix } f \$ i \$ j| \leq \text{onorm } f$

1912

by (metis matrix_component_le_onorm matrix_vector_mul(2))

lemma onorm_le_matrix_component_sum:

fixes $A :: \text{real}^n{}^m$

shows $\text{onorm}((\ast v) A) \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j|)$

proof (rule onorm_le)

fix x

have $\text{norm} (A \ast v x) \leq (\sum i \in \text{UNIV}. |(A \ast v x) \$ i|)$

by (rule norm_le_l1_cart)

also have $\dots \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$

proof (rule sum_mono)

fix i

have $|(A \ast v x) \$ i| \leq |\sum j \in \text{UNIV}. A \$ i \$ j \ast x \$ j|$

by (simp add: matrix_vector_mult_def)

also have $\dots \leq (\sum j \in \text{UNIV}. |A \$ i \$ j \ast x \$ j|)$

by (rule sum_abs)

also have $\dots \leq (\sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$

by (rule sum_mono) (simp add: abs_mult component_le_norm_cart mult_left_mono)

finally show $|(A \ast v x) \$ i| \leq (\sum j \in \text{UNIV}. |A \$ i \$ j| \ast \text{norm } x)$.

qed

finally show $\text{norm} (A \ast v x) \leq (\sum i \in \text{UNIV}. \sum j \in \text{UNIV}. |A \$ i \$ j|) \ast \text{norm } x$

by (simp add: sum_distrib_right)

qed

lemma onorm_le_matrix_component:

fixes $A :: \text{real}^n{}^m$

assumes $\bigwedge i j. \text{abs}(A \$ i \$ j) \leq B$

shows $\text{onorm}((\ast v) A) \leq \text{real} (\text{CARD}'m) \ast \text{real} (\text{CARD}'n) \ast B$

proof (rule onorm_le)

fix $x :: \text{real}^n$:

have $\text{norm} (A \ast v x) \leq (\sum i \in \text{UNIV}. |(A \ast v x) \$ i|)$

by (rule norm_le_l1_cart)

also have $\dots \leq (\sum i :: 'm \in \text{UNIV}. \text{real} (\text{CARD}'n) \ast B \ast \text{norm } x)$

proof (rule sum_mono)

fix i

have $|(A \ast v x) \$ i| \leq \text{norm}(A \$ i) \ast \text{norm } x$

by (simp add: matrix_mult_dot Cauchy_Schwarz_ineq2)

also have $\dots \leq (\sum j \in \text{UNIV}. |A \$ i \$ j|) \ast \text{norm } x$

by (simp add: mult_right_mono norm_le_l1_cart)

also have $\dots \leq \text{real} (\text{CARD}'n) \ast B \ast \text{norm } x$

by (simp add: assms sum_bounded_above mult_right_mono)

finally show $|(A \ast v x) \$ i| \leq \text{real} (\text{CARD}'n) \ast B \ast \text{norm } x$.

qed

also have $\dots \leq \text{CARD}'m \ast \text{real} (\text{CARD}'n) \ast B \ast \text{norm } x$

by simp

finally show $\text{norm} (A \ast v x) \leq \text{CARD}'m \ast \text{real} (\text{CARD}'n) \ast B \ast \text{norm } x$.

qed

lemma vector_sub_project_orthogonal_cart: $(b :: \text{real}^n) \cdot (x - ((b \cdot x) / (b \cdot b)))$

$*s\ b) = 0$

unfolding *inner_simps scalar_mult_eq_scaleR* **by** *auto*

lemma *infnorm_cart*: $\text{infnorm } (x::\text{real}^n) = \text{Sup } \{|x\ \$i| \mid i. i \in \text{UNIV}\}$

by (*simp add: infnorm_def inner_axis Basis_vec_def*) (*metis (lifting) inner_axis real_inner_1_right*)

lemma *component_le_infnorm_cart*: $|x\ \$i| \leq \text{infnorm } (x::\text{real}^n)$

using *Basis_le_infnorm*[*of axis i 1 x*]

by (*simp add: Basis_vec_def axis_eq_axis inner_axis*)

lemma *continuous_component*[*continuous_intros*]: $\text{continuous } F\ f \implies \text{continuous } F\ (\lambda x. f\ x\ \$i)$

unfolding *continuous_def* **by** (*rule tendsto_vec_nth*)

lemma *continuous_on_component*[*continuous_intros*]: $\text{continuous_on } s\ f \implies \text{continuous_on } s\ (\lambda x. f\ x\ \$i)$

unfolding *continuous_on_def* **by** (*fast intro: tendsto_vec_nth*)

lemma *continuous_on_vec_lambda*[*continuous_intros*]:

$(\bigwedge i. \text{continuous_on } S\ (f\ i)) \implies \text{continuous_on } S\ (\lambda x. \chi\ i. f\ i\ x)$

unfolding *continuous_on_def* **by** (*auto intro: tendsto_vec_lambda*)

lemma *closed_positive_orthant*: $\text{closed } \{x::\text{real}^n. \forall i. 0 \leq x\ \$i\}$

by (*simp add: Collect_all_eq closed_INT closed_Collect_le continuous_on_component*)

lemma *bounded_component_cart*: $\text{bounded } s \implies \text{bounded } ((\lambda x. x\ \$i)\ 's)$

unfolding *bounded_def*

apply *clarify*

apply (*rule_tac x=x\ \\$i in exI*)

apply (*rule_tac x=e in exI*)

apply *clarify*

apply (*rule order_trans [OF dist_vec_nth_le], simp*)

done

lemma *compact_lemma_cart*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{heine_borel } ^n$

assumes f : *bounded* (*range* f)

shows $\exists l\ r. \text{strict_mono } r \wedge$

$(\forall e>0. \text{eventually } (\lambda n. \forall i \in d. \text{dist } (f\ (r\ n)\ \$i)\ (l\ \$i) < e) \text{ sequentially})$

(*is ?th d*)

proof –

have $\forall d' \subseteq d. ?th\ d'$

by (*rule compact_lemma_general*[**where** *unproj=vec_lambda*])

(*auto intro!: f bounded_component_cart*)

then show *?th d* **by** *simp*

qed

instance *vec* :: (*heine_borel, finite*) *heine_borel*

proof

```

fix f :: nat ⇒ 'a ^ 'b
assume f: bounded (range f)
then obtain l r where r: strict_mono r
      and l: ∀ e>0. eventually (λn. ∀ i∈UNIV. dist (f (r n) $ i) (l $ i) < e)
sequentially
  using compact_lemma_cart [OF f] by blast
let ?d = UNIV::'b set
  { fix e::real assume e>0
    hence 0 < e / (real_of_nat (card ?d))
      using zero_less_card_finite divide_pos_pos[of e, of real_of_nat (card ?d)]
by auto
    with l have eventually (λn. ∀ i. dist (f (r n) $ i) (l $ i) < e / (real_of_nat
(card ?d))) sequentially
      by simp
    moreover
    { fix n
      assume n: ∀ i. dist (f (r n) $ i) (l $ i) < e / (real_of_nat (card ?d))
      have dist (f (r n)) l ≤ (∑ i∈?d. dist (f (r n) $ i) (l $ i))
        unfolding dist_vec_def using zero_le_dist by (rule L2_set_le_sum)
      also have ... < (∑ i∈?d. e / (real_of_nat (card ?d)))
        by (rule sum_strict_mono) (simp_all add: n)
      finally have dist (f (r n)) l < e by simp
    }
    ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
      by (rule eventually_mono)
  }
hence ((f ∘ r) ⟶ l) sequentially unfolding o_def tendsto_iff by simp
with r show ∃ l r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially by auto
qed

```

lemma interval_cart:

```

fixes a :: real^n
shows box a b = {x::real^n. ∀ i. a$i < x$i ∧ x$i < b$i}
and cbox a b = {x::real^n. ∀ i. a$i ≤ x$i ∧ x$i ≤ b$i}
by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def mem_box Basis_vec_def inner_axis)

```

lemma mem_box_cart:

```

fixes a :: real^n
shows x ∈ box a b ⟷ (∀ i. a$i < x$i ∧ x$i < b$i)
and x ∈ cbox a b ⟷ (∀ i. a$i ≤ x$i ∧ x$i ≤ b$i)
using interval_cart[of a b] by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def)

```

lemma interval_eq_empty_cart:

```

fixes a :: real^n
shows (box a b = {}) ⟷ (∃ i. b$i ≤ a$i) (is ?th1)
and (cbox a b = {}) ⟷ (∃ i. b$i < a$i) (is ?th2)

```

proof –

```

{ fix i x assume as: b$i ≤ a$i and x: x ∈ cbox a b
  hence a $ i < x $ i ∧ x $ i < b $ i unfolding mem_box_cart by auto
  hence a$i < b$i by auto
  hence False using as by auto }
moreover
{ assume as: ∀ i. ¬ (b$i ≤ a$i)
  let ?x = (1/2) *R (a + b)
  { fix i
    have a$i < b$i using as[THEN spec[where x=i]] by auto
    hence a$i < ((1/2) *R (a+b)) $ i ((1/2) *R (a+b)) $ i < b$i
      unfolding vector_smult_component and vector_add_component
      by auto }
  hence cbox a b ≠ {} using mem_box_cart(1)[of ?x a b] by auto }
ultimately show ?th1 by blast

```

```

{ fix i x assume as: b$i < a$i and x: x ∈ cbox a b
  hence a $ i ≤ x $ i ∧ x $ i ≤ b $ i unfolding mem_box_cart by auto
  hence a$i ≤ b$i by auto
  hence False using as by auto }
moreover
{ assume as: ∀ i. ¬ (b$i < a$i)
  let ?x = (1/2) *R (a + b)
  { fix i
    have a$i ≤ b$i using as[THEN spec[where x=i]] by auto
    hence a$i ≤ ((1/2) *R (a+b)) $ i ((1/2) *R (a+b)) $ i ≤ b$i
      unfolding vector_smult_component and vector_add_component
      by auto }
  hence cbox a b ≠ {} using mem_box_cart(2)[of ?x a b] by auto }
ultimately show ?th2 by blast
qed

```

lemma interval_ne_empty_cart:

```

fixes a :: realn
shows cbox a b ≠ {} ↔ (∀ i. a$i ≤ b$i)
  and cbox a b ≠ {} ↔ (∀ i. a$i < b$i)
unfolding interval_eq_empty_cart[of a b] by (auto simp add: not_less not_le)

```

lemma subset_interval_imp_cart:

```

fixes a :: realn
shows (∀ i. a$i ≤ c$i ∧ d$i ≤ b$i) ⇒ cbox c d ⊆ cbox a b
  and (∀ i. a$i < c$i ∧ d$i < b$i) ⇒ cbox c d ⊆ box a b
  and (∀ i. a$i ≤ c$i ∧ d$i ≤ b$i) ⇒ box c d ⊆ cbox a b
  and (∀ i. a$i < c$i ∧ d$i < b$i) ⇒ box c d ⊆ box a b
unfolding subset_eq[unfolded Ball_def] unfolding mem_box_cart
by (auto intro: order_trans less_le_trans le_less_trans less_imp_le)

```

lemma interval_sing:

```

fixes a :: 'a::linordern

```

```

shows {a .. a} = {a} ∧ {a <..<a} = {}
apply (auto simp add: set_eq_iff less_vec_def less_eq_vec_def vec_eq_iff)
done

```

lemma *subset_interval_cart*:

```

fixes a :: realn
shows cbox c d ⊆ cbox a b ↔ (∀ i. ci ≤ di) --> (∀ i. ai ≤ ci ∧ di ≤ bi) (is ?th1)
and cbox c d ⊆ box a b ↔ (∀ i. ci ≤ di) --> (∀ i. ai < ci ∧ di < bi) (is ?th2)
and box c d ⊆ cbox a b ↔ (∀ i. ci < di) --> (∀ i. ai ≤ ci ∧ di ≤ bi) (is ?th3)
and box c d ⊆ box a b ↔ (∀ i. ci < di) --> (∀ i. ai ≤ ci ∧ di ≤ bi) (is ?th4)
using subset_box[of c d a b] by (simp_all add: Basis_vec_def inner_axis)

```

lemma *disjoint_interval_cart*:

```

fixes a :: realn
shows cbox a b ∩ cbox c d = {} ↔ (∃ i. (bi < ai ∨ di < ci ∨ bi < ci ∨ di < ai)) (is ?th1)
and cbox a b ∩ box c d = {} ↔ (∃ i. (bi < ai ∨ di ≤ ci ∨ bi ≤ ci ∨ di ≤ ai)) (is ?th2)
and box a b ∩ cbox c d = {} ↔ (∃ i. (bi ≤ ai ∨ di < ci ∨ bi ≤ ci ∨ di ≤ ai)) (is ?th3)
and box a b ∩ box c d = {} ↔ (∃ i. (bi ≤ ai ∨ di ≤ ci ∨ bi ≤ ci ∨ di ≤ ai)) (is ?th4)
using disjoint_interval[of a b c d] by (simp_all add: Basis_vec_def inner_axis)

```

lemma *Int_interval_cart*:

```

fixes a :: realn
shows cbox a b ∩ cbox c d = {(χ i. max (ai) (ci)) .. (χ i. min (bi) (di))}
unfolding Int_interval
by (auto simp: mem_box less_eq_vec_def)
(auto simp: Basis_vec_def inner_axis)

```

lemma *closed_interval_left_cart*:

```

fixes b :: realn
shows closed {x :: realn. ∀ i. xi ≤ bi}
by (simp add: Collect_all_eq closed_INT closed_Collect_le continuous_on_component)

```

lemma *closed_interval_right_cart*:

```

fixes a :: realn
shows closed {x :: realn. ∀ i. ai ≤ xi}
by (simp add: Collect_all_eq closed_INT closed_Collect_le continuous_on_component)

```

lemma *is_interval_cart*:

```

is_interval (s :: (realn) set) ↔
(∀ a ∈ s. ∀ b ∈ s. ∀ x. (∀ i. ((ai ≤ xi ∧ xi ≤ bi) ∨ (bi ≤ xi ∧ xi ≤ ai)))
→ x ∈ s)

```


by (simp add: is_interval_def Ball_def Basis_vec_def inner_axis imp_ex)

lemma *closed_halfspace_component_le_cart*: *closed* $\{x::\text{real}^n. x\$i \leq a\}$
 by (simp add: closed_Collect_le continuous_on_component)

lemma *closed_halfspace_component_ge_cart*: *closed* $\{x::\text{real}^n. x\$i \geq a\}$
 by (simp add: closed_Collect_le continuous_on_component)

lemma *open_halfspace_component_lt_cart*: *open* $\{x::\text{real}^n. x\$i < a\}$
 by (simp add: open_Collect_less continuous_on_component)

lemma *open_halfspace_component_gt_cart*: *open* $\{x::\text{real}^n. x\$i > a\}$
 by (simp add: open_Collect_less continuous_on_component)

lemma *Lim_component_le_cart*:
 fixes $f :: 'a \Rightarrow \text{real}^n$
 assumes $(f \longrightarrow l)$ *net* \neg (*trivial_limit net*) *eventually* $(\lambda x. f x \$i \leq b)$ *net*
 shows $l\$i \leq b$
 by (rule tendsto_le[OF assms(2) tendsto_const tendsto_vec_nth, OF assms(1, 3)])

lemma *Lim_component_ge_cart*:
 fixes $f :: 'a \Rightarrow \text{real}^n$
 assumes $(f \longrightarrow l)$ *net* \neg (*trivial_limit net*) *eventually* $(\lambda x. b \leq (f x)\$i)$ *net*
 shows $b \leq l\$i$
 by (rule tendsto_le[OF assms(2) tendsto_vec_nth tendsto_const, OF assms(1, 3)])

lemma *Lim_component_eq_cart*:
 fixes $f :: 'a \Rightarrow \text{real}^n$
 assumes *net*: $(f \longrightarrow l)$ *net* \neg *trivial_limit net* **and** *ev*:*eventually* $(\lambda x. f(x)\$i = b)$ *net*
 shows $l\$i = b$
 using *ev*[*unfolded order_eq_iff eventually_conj_iff*] **and**
Lim_component_ge_cart[*OF net, of b i*] **and**
Lim_component_le_cart[*OF net, of i b*] **by auto**

lemma *connected_ivt_component_cart*:
 fixes $x :: \text{real}^n$
 shows *connected* $s \implies x \in s \implies y \in s \implies x\$k \leq a \implies a \leq y\$k \implies (\exists z \in s. z\$k = a)$
 using *connected_ivt_hyperplane*[*of s x y axis k 1 a*]
 by (*auto simp add: inner_axis inner_commute*)

lemma *subspace_substandard_cart*: *vec.subspace* $\{x. (\forall i. P i \longrightarrow x\$i = 0)\}$
 unfolding *vec.subspace_def* **by auto**

lemma *closed_substandard_cart*:
closed $\{x::'a::\text{real_normed_vector}^n. \forall i. P i \longrightarrow x\$i = 0\}$

1918

```
proof –
  { fix  $i::'n$ 
    have  $closed \{x::'a \hat{=} 'n. P\ i \longrightarrow x\ \$i = 0\}$ 
      by  $(cases\ P\ i)\ (simp\_all\ add:\ closed\_Collect\_eq\ continuous\_on\_component)$ 
  }
  thus  $?thesis$ 
    unfolding  $Collect\_all\_eq$  by  $(simp\ add:\ closed\_INT)$ 
qed
```

6.18.3 Convex Euclidean Space

```
lemma  $Cart\_1:(1::real^{'n}) = \sum\ Basis$ 
  using  $const\_vector\_cart[of\ 1]$  by  $(simp\ add:\ one\_vec\_def)$ 
```

```
declare  $vector\_add\_ldistrib[simp]\ vector\_ssub\_ldistrib[simp]\ vector\_smult\_assoc[simp]$ 
 $vector\_smult\_rneg[simp]$ 
declare  $vector\_sadd\_rdistrib[simp]\ vector\_sub\_rdistrib[simp]$ 
```

```
lemmas  $vector\_component\_simps = vector\_minus\_component\ vector\_smult\_component$ 
 $vector\_add\_component\ less\_eq\_vec\_def\ vec\_lambda\_beta\ vector\_uminus\_component$ 
```

```
lemma  $convex\_box\_cart:$ 
  assumes  $\bigwedge i. convex \{x. P\ i\ x\}$ 
  shows  $convex \{x. \forall i. P\ i\ (x\ \$i)\}$ 
  using  $assms$  unfolding  $convex\_def$  by  $auto$ 
```

6.18.4 Arbitrarily good rational approximations

```
lemma  $rational\_approximation:$ 
  assumes  $e > 0$ 
  obtains  $r::real$  where  $r \in \mathbb{Q} \mid r - x < e$ 
  using  $Rats\_dense\_in\_real [of\ x - e/2\ x + e/2]$   $assms$  by  $auto$ 
```

```
lemma  $Rats\_closure\_real: closure\ \mathbb{Q} = (UNIV::real\ set)$ 
```

```
proof –
  have  $\bigwedge x::real. x \in closure\ \mathbb{Q}$ 
    by  $(metis\ closure\_approachable\ dist\_real\_def\ rational\_approximation)$ 
  then show  $?thesis$  by  $auto$ 
qed
```

```
proposition  $matrix\_rational\_approximation:$ 
```

```
  fixes  $A :: real^{'n\ 'm}$ 
```

```
  assumes  $e > 0$ 
```

```
  obtains  $B$  where  $\bigwedge i\ j. B\ \$i\ \$j \in \mathbb{Q}$   $onorm(\lambda x. (A - B) * v\ x) < e$ 
```

```
proof –
```

```
  have  $\forall i\ j. \exists q \in \mathbb{Q}. |q - A\ \$i\ \$j| < e / (2 * CARD('m) * CARD('n))$ 
```

```
    using  $assms$  by  $(force\ intro:\ rational\_approximation [of\ e / (2 * CARD('m) * CARD('n))])$ 
```

```
  then obtain  $B$  where  $B: \bigwedge i\ j. B\ \$i\ \$j \in \mathbb{Q}$  and  $Bclo: \bigwedge i\ j. |B\ \$i\ \$j - A\ \$i\ \$j| < e / (2 * CARD('m) * CARD('n))$ 
```

```

  by (auto simp: lambda_skolem Bex_def)
show ?thesis
proof
  have onorm ((*v) (A - B)) ≤ real CARD('m) * real CARD('n) *
    (e / (2 * real CARD('m) * real CARD('n)))
    apply (rule onorm_le_matrix_component)
    using Bclo by (simp add: abs_minus_commute less_imp_le)
  also have ... < e
    using ‹0 < e› by (simp add: field_split_simps)
  finally show onorm ((*v) (A - B)) < e .
qed (use B in auto)
qed

```

6.18.5 Derivative

definition *jacobian* f $net = matrix(frechet_derivative\ f\ net)$

proposition *jacobian_works*:

$(f::(real^a) \Rightarrow (real^b))$ differentiable $net \longleftrightarrow$
 $(f$ has_derivative $(\lambda h. (jacobian\ f\ net) * v\ h))\ net$ (is ?lhs = ?rhs)

proof

assume ?lhs then show ?rhs

by (simp add: frechet_derivative_works has_derivative_linear jacobian_def)

next

assume ?rhs then show ?lhs

by (rule differentiableI)

qed

Component of the differential must be zero if it exists at a local maximum or minimum for that corresponding component

proposition *differential_zero_maxmin_cart*:

fixes $f::real^a \Rightarrow real^b$

assumes $0 < e$ ($(\forall y \in ball\ x\ e. (f\ y)\$k \leq (f\ x)\$k) \vee (\forall y \in ball\ x\ e. (f\ x)\$k \leq (f\ y)\$k)$)

f differentiable (at x)

shows $jacobian\ f\ (at\ x)\ \$k = 0$

using *differential_zero_maxmin_component*[of axis k 1 $e\ x\ f$] *assms*

vector_cart[of $\lambda j. frechet_derivative\ f\ (at\ x)\ j\ \k]

by (simp add: *Basis_vec_def axis_eq_axis inner_axis jacobian_def matrix_def*)

6.18.6 Routine results connecting the types $(real, 1)$ *vec* and *real*

lemma *vec_cbox_1_eq* [simp]:

shows $vec\ 'cbox\ u\ v = cbox\ (vec\ u)\ (vec\ v::real^1)$

by (force simp: *Basis_vec_def cart_eq_inner_axis [symmetric] mem_box*)

lemma *vec_nth_cbox_1_eq* [simp]:

fixes $u\ v::'a::euclidean_space^1$

1920

shows $(\lambda x. x \$ 1) \text{ ' } cbox\ u\ v = cbox\ (u\$1)\ (v\$1)$
by $(auto\ simp: Basis_vec_def\ cart_eq_inner_axis\ [symmetric]\ mem_box\ image_iff\ Bex_def\ inner_axis)\ (metis\ vec_component)$

lemma $vec_nth_1_iff_cbox\ [simp]:$

fixes $a\ b :: 'a::euclidean_space$

shows $(\lambda x::'a^1. x \$ 1) \text{ ' } S = cbox\ a\ b \longleftrightarrow S = cbox\ (vec\ a)\ (vec\ b)$

(is $?lhs = ?rhs$ **)**

proof

assume $L: ?lhs\ show\ ?rhs$

proof $(intro\ equalityI\ subsetI)$

fix x

assume $x \in S$

then have $x \$ 1 \in (\lambda v. v \$ (1::1)) \text{ ' } cbox\ (vec\ a)\ (vec\ b)$

using L **by** $auto$

then show $x \in cbox\ (vec\ a)\ (vec\ b)$

by $(metis\ (no_types,\ lifting)\ imageE\ vector_one_nth)$

next

fix $x :: 'a^1$

assume $x \in cbox\ (vec\ a)\ (vec\ b)$

then show $x \in S$

by $(metis\ (no_types,\ lifting)\ L\ imageE\ imageI\ vec_component\ vec_nth_cbox_1_eq\ vector_one_nth)$

qed

qed $simp$

lemma $vec_nth_real_1_iff_cbox\ [simp]:$

fixes $a\ b :: real$

shows $(\lambda x::real^1. x \$ 1) \text{ ' } S = \{a..b\} \longleftrightarrow S = cbox\ (vec\ a)\ (vec\ b)$

using $vec_nth_1_iff_cbox[of\ S\ a\ b]$

by $simp$

lemma $interval_split_cart:$

$\{a..b::real^n\} \cap \{x. x\$k \leq c\} = \{a..(\chi\ i. \text{if } i = k \text{ then } \min\ (b\$k)\ c \text{ else } b\$i)\}$

$cbox\ a\ b \cap \{x. x\$k \geq c\} = \{(\chi\ i. \text{if } i = k \text{ then } \max\ (a\$k)\ c \text{ else } a\$i)..b\}$

unfolding $Int_iff\ mem_box_cart\ mem_Collect_eq\ interval_cbox_cart\ set_eq_iff$

unfolding vec_lambda_beta

by $auto$

lemmas $cartesian_euclidean_space_uniform_limit_intros[uniform_limit_intros]$

=

$bounded_linear.uniform_limit[OF\ blinfun.bounded_linear_right]$

$bounded_linear.uniform_limit[OF\ bounded_linear_vec_nth]$

end

6.19 Complex Analysis Basics

Definitions of analytic and holomorphic functions, limit theorems, complex differentiation

```
theory Complex_Analysis_Basics
  imports Derivative HOL-Library.Nonpos_Ints Uncountable_Sets
begin
```

6.19.1 General lemmas

```
lemma nonneg_Reals_cmod_eq_Re:  $z \in \mathbb{R}_{\geq 0} \implies \text{norm } z = \text{Re } z$ 
  by (simp add: complex_nonneg_Reals_iff_cmod_eq_Re)
```

```
lemma fact_cancel:
  fixes  $c :: 'a :: \text{real\_field}$ 
  shows  $\text{of\_nat } (\text{Suc } n) * c / (\text{fact } (\text{Suc } n)) = c / (\text{fact } n)$ 
  using  $\text{of\_nat\_neq\_0}$  by force
```

```
lemma vector_derivative_cnj_within:
  assumes  $\text{at } x \text{ within } A \neq \text{bot}$  and  $f \text{ differentiable at } x \text{ within } A$ 
  shows  $\text{vector\_derivative } (\lambda z. \text{cnj } (f z)) \text{ (at } x \text{ within } A) =$ 
 $\text{cnj } (\text{vector\_derivative } f \text{ (at } x \text{ within } A))$  (is  $\_ = \text{cnj } ?D$ )
```

```
proof -
  let  $?D = \text{vector\_derivative } f \text{ (at } x \text{ within } A)$ 
  from  $\text{assms}$  have  $(f \text{ has\_vector\_derivative } ?D) \text{ (at } x \text{ within } A)$ 
    by (subst (asm) vector_derivative_works)
  hence  $(\lambda x. \text{cnj } (f x)) \text{ has\_vector\_derivative } \text{cnj } ?D \text{ (at } x \text{ within } A)$ 
    by (rule has_vector_derivative_cnj)
  thus  $?thesis$  using  $\text{assms}$  by (auto dest: vector_derivative_within)
qed
```

```
lemma vector_derivative_cnj:
  assumes  $f \text{ differentiable at } x$ 
  shows  $\text{vector\_derivative } (\lambda z. \text{cnj } (f z)) \text{ (at } x) = \text{cnj } (\text{vector\_derivative } f \text{ (at } x))$ 
  using  $\text{assms}$  by (intro vector_derivative_cnj_within) auto
```

```
lemma
  shows  $\text{open\_halfspace\_Re\_lt}: \text{open } \{z. \text{Re}(z) < b\}$ 
    and  $\text{open\_halfspace\_Re\_gt}: \text{open } \{z. \text{Re}(z) > b\}$ 
    and  $\text{closed\_halfspace\_Re\_ge}: \text{closed } \{z. \text{Re}(z) \geq b\}$ 
    and  $\text{closed\_halfspace\_Re\_le}: \text{closed } \{z. \text{Re}(z) \leq b\}$ 
    and  $\text{closed\_halfspace\_Re\_eq}: \text{closed } \{z. \text{Re}(z) = b\}$ 
    and  $\text{open\_halfspace\_Im\_lt}: \text{open } \{z. \text{Im}(z) < b\}$ 
    and  $\text{open\_halfspace\_Im\_gt}: \text{open } \{z. \text{Im}(z) > b\}$ 
    and  $\text{closed\_halfspace\_Im\_ge}: \text{closed } \{z. \text{Im}(z) \geq b\}$ 
    and  $\text{closed\_halfspace\_Im\_le}: \text{closed } \{z. \text{Im}(z) \leq b\}$ 
    and  $\text{closed\_halfspace\_Im\_eq}: \text{closed } \{z. \text{Im}(z) = b\}$ 
  by (intro open_Collect_less closed_Collect_le closed_Collect_eq continuous_on_Re)
```

continuous_on_Im continuous_on_id continuous_on_const)+

lemma *uncountable_halfspace_Im_gt*: *uncountable* {*z*. *Im z* > *c*}
proof –
obtain *r* **where** *r*: *r* > 0 *ball* ((*c* + 1) *_{*R*} *i*) *r* ⊆ {*z*. *Im z* > *c*}
using *open_halfspace_Im_gt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show ?*thesis*
using *countable_subset_uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Im_lt*: *uncountable* {*z*. *Im z* < *c*}
proof –
obtain *r* **where** *r*: *r* > 0 *ball* ((*c* – 1) *_{*R*} *i*) *r* ⊆ {*z*. *Im z* < *c*}
using *open_halfspace_Im_lt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show ?*thesis*
using *countable_subset_uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Re_gt*: *uncountable* {*z*. *Re z* > *c*}
proof –
obtain *r* **where** *r*: *r* > 0 *ball* (*of_real*(*c* + 1)) *r* ⊆ {*z*. *Re z* > *c*}
using *open_halfspace_Re_gt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show ?*thesis*
using *countable_subset_uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Re_lt*: *uncountable* {*z*. *Re z* < *c*}
proof –
obtain *r* **where** *r*: *r* > 0 *ball* (*of_real*(*c* – 1)) *r* ⊆ {*z*. *Re z* < *c*}
using *open_halfspace_Re_lt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show ?*thesis*
using *countable_subset_uncountable_ball* **by** *blast*
qed

lemma *connected_halfspace_Im_gt* [*intro*]: *connected* {*z*. *c* < *Im z*}
by (*intro convex_connected convex_halfspace_Im_gt*)

lemma *connected_halfspace_Im_lt* [*intro*]: *connected* {*z*. *c* > *Im z*}
by (*intro convex_connected convex_halfspace_Im_lt*)

lemma *connected_halfspace_Re_gt* [*intro*]: *connected* {*z*. *c* < *Re z*}
by (*intro convex_connected convex_halfspace_Re_gt*)

lemma *connected_halfspace_Re_lt* [*intro*]: *connected* {*z*. *c* > *Re z*}
by (*intro convex_connected convex_halfspace_Re_lt*)

lemma *closed_complex_Reals*: *closed* (**R** :: *complex set*)
proof –
have (**R** :: *complex set*) = {*z*. *Im z* = 0}

```

  by (auto simp: complex_is_Real_iff)
  then show ?thesis
  by (metis closed_halfspace_Im_eq)
qed

```

```

lemma closed_Real_halfspace_Re_le: closed ( $\mathbb{R} \cap \{w. \operatorname{Re} w \leq x\}$ )
  by (simp add: closed_Int closed_complex_Reals closed_halfspace_Re_le)

```

```

lemma closed_nonpos_Reals_complex [simp]: closed ( $\mathbb{R}_{\leq 0} :: \text{complex set}$ )
proof -
  have  $\mathbb{R}_{\leq 0} = \mathbb{R} \cap \{z. \operatorname{Re}(z) \leq 0\}$ 
  using complex_nonpos_Reals_iff complex_is_Real_iff by auto
  then show ?thesis
  by (metis closed_Real_halfspace_Re_le)
qed

```

```

lemma closed_Real_halfspace_Re_ge: closed ( $\mathbb{R} \cap \{w. x \leq \operatorname{Re}(w)\}$ )
  using closed_halfspace_Re_ge
  by (simp add: closed_Int closed_complex_Reals)

```

```

lemma closed_nonneg_Reals_complex [simp]: closed ( $\mathbb{R}_{\geq 0} :: \text{complex set}$ )
proof -
  have  $\mathbb{R}_{\geq 0} = \mathbb{R} \cap \{z. \operatorname{Re}(z) \geq 0\}$ 
  using complex_nonneg_Reals_iff complex_is_Real_iff by auto
  then show ?thesis
  by (metis closed_Real_halfspace_Re_ge)
qed

```

```

lemma closed_real_abs_le: closed  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\}$ 
proof -
  have  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\} = (\mathbb{R} \cap \{w. \operatorname{Re} w \leq r\}) \cap (\mathbb{R} \cap \{w. \operatorname{Re} w \geq -r\})$ 
  by auto
  then show closed  $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq r\}$ 
  by (simp add: closed_Int closed_Real_halfspace_Re_ge closed_Real_halfspace_Re_le)
qed

```

```

lemma real_lim:
  fixes  $l::\text{complex}$ 
  assumes  $(f \longrightarrow l) F$  and  $\neg \text{trivial\_limit } F$  and eventually  $P F$  and  $\bigwedge a. P a$ 
 $\implies f a \in \mathbb{R}$ 
  shows  $l \in \mathbb{R}$ 
  using Lim_in_closed_set[OF closed_complex_Reals] assms
  by (smt (verit) eventually_mono)

```

```

lemma real_lim_sequentially:
  fixes  $l::\text{complex}$ 
  shows  $(f \longrightarrow l) \text{ sequentially} \implies (\exists N. \forall n \geq N. f n \in \mathbb{R}) \implies l \in \mathbb{R}$ 
  by (rule real_lim [where  $F=\text{sequentially}$ ]) (auto simp: eventually_sequentially)

```

lemma *real_series*:
fixes $l::\text{complex}$
shows $f \text{ sums } l \implies (\bigwedge n. f\ n \in \mathbb{R}) \implies l \in \mathbb{R}$
unfolding *sums_def*
by (*metis real_lim_sequentially_sum_in_Reals*)

lemma *Lim_null_comparison_Re*:
assumes *eventually* $(\lambda x. \text{norm}(f\ x) \leq \text{Re}(g\ x))\ F\ (g \longrightarrow 0)\ F$ **shows** $(f \longrightarrow 0)\ F$
using *Lim_null_comparison_assms_tendsto_Re* **by** *fastforce*

6.19.2 Holomorphic functions

definition *holomorphic_on* :: $[\text{complex} \Rightarrow \text{complex}, \text{complex set}] \Rightarrow \text{bool}$
(infixl (holomorphic'_on) 50)
where $f \text{ holomorphic_on } s \equiv \forall x \in s. f \text{ field_differentiable (at } x \text{ within } s)$

named_theorems *holomorphic_intros* *structural introduction rules for holomorphic_on*

lemma *holomorphic_onI* [*intro?*]: $(\bigwedge x. x \in s \implies f \text{ field_differentiable (at } x \text{ within } s)) \implies f \text{ holomorphic_on } s$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_onD* [*dest?*]: $\llbracket f \text{ holomorphic_on } s; x \in s \rrbracket \implies f \text{ field_differentiable (at } x \text{ within } s)$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_on_imp_differentiable_on*:
 $f \text{ holomorphic_on } s \implies f \text{ differentiable_on } s$
unfolding *holomorphic_on_def differentiable_on_def*
by (*simp add: field_differentiable_imp_differentiable*)

lemma *holomorphic_on_imp_differentiable_at*:
 $\llbracket f \text{ holomorphic_on } s; \text{open } s; x \in s \rrbracket \implies f \text{ field_differentiable (at } x)$
using *at_within_open_holomorphic_on_def* **by** *fastforce*

lemma *holomorphic_on_empty* [*holomorphic_intros*]: $f \text{ holomorphic_on } \{\}$
by (*simp add: holomorphic_on_def*)

lemma *holomorphic_on_open*:
 $\text{open } s \implies f \text{ holomorphic_on } s \longleftrightarrow (\forall x \in s. \exists f'. \text{DERIV } f\ x \text{ :> } f')$
by (*auto simp: holomorphic_on_def field_differentiable_def has_field_derivative_def at_within_open [of _ s]*)

lemma *holomorphic_on_UN_open*:
assumes $\bigwedge n. n \in I \implies f \text{ holomorphic_on } A\ n \ \bigwedge n. n \in I \implies \text{open } (A\ n)$
shows $f \text{ holomorphic_on } (\bigcup n \in I. A\ n)$
by (*metis UN_E assms_holomorphic_on_open open_UN*)

lemma *holomorphic_on_imp_continuous_on*:

f holomorphic_on s \implies continuous_on s f

using *differentiable_imp_continuous_on holomorphic_on_imp_differentiable_on*
by *blast*

lemma *holomorphic_closedin_preimage_constant*:

assumes *f holomorphic_on D*

shows *closedin (top_of_set D) {z ∈ D. f z = a}*

by (*simp add: assms continuous_closedin_preimage_constant holomorphic_on_imp_continuous_on*)

lemma *holomorphic_closed_preimage_constant*:

assumes *f holomorphic_on UNIV*

shows *closed {z. f z = a}*

using *holomorphic_closedin_preimage_constant [OF assms]* **by** *simp*

lemma *holomorphic_on_subset [elim]*:

f holomorphic_on s \implies t \subseteq s \implies f holomorphic_on t

unfolding *holomorphic_on_def*

by (*metis field_differentiable_within_subset subsetD*)

lemma *holomorphic_transform*: $\llbracket f \text{ holomorphic_on } s; \bigwedge x. x \in s \implies f x = g x \rrbracket$
 $\implies g \text{ holomorphic_on } s$

by (*metis field_differentiable_transform_within linordered_field_no_ub holomorphic_on_def*)

lemma *holomorphic_cong*: $s = t \implies (\bigwedge x. x \in s \implies f x = g x) \implies f \text{ holomorphic_on } s \iff g \text{ holomorphic_on } t$

by (*metis holomorphic_transform*)

lemma *holomorphic_on_linear [simp, holomorphic_intros]*: $((*) c) \text{ holomorphic_on } s$

unfolding *holomorphic_on_def* **by** (*metis field_differentiable_linear*)

lemma *holomorphic_on_const [simp, holomorphic_intros]*: $(\lambda z. c) \text{ holomorphic_on } s$

unfolding *holomorphic_on_def* **by** (*metis field_differentiable_const*)

lemma *holomorphic_on_ident [simp, holomorphic_intros]*: $(\lambda x. x) \text{ holomorphic_on } s$

unfolding *holomorphic_on_def* **by** (*metis field_differentiable_ident*)

lemma *holomorphic_on_id [simp, holomorphic_intros]*: *id holomorphic_on s*

unfolding *id_def* **by** (*rule holomorphic_on_ident*)

lemma *constant_on_imp_holomorphic_on*:

assumes *f constant_on A*

shows *f holomorphic_on A*

by (*metis assms constant_on_def holomorphic_on_const holomorphic_transform*)

lemma *holomorphic_on_compose*:
 $f \text{ holomorphic_on } s \implies g \text{ holomorphic_on } (f' s) \implies (g \circ f) \text{ holomorphic_on } s$
using *field_differentiable_compose_within*[of $f _ s \ g$]
by (*auto simp: holomorphic_on_def*)

lemma *holomorphic_on_compose_gen*:
 $f \text{ holomorphic_on } s \implies g \text{ holomorphic_on } t \implies f' s \subseteq t \implies (g \circ f) \text{ holomorphic_on } s$
by (*metis holomorphic_on_compose holomorphic_on_subset*)

lemma *holomorphic_on_balls_imp_entire*:
assumes $\neg \text{bdd_above } A \wedge r. r \in A \implies f \text{ holomorphic_on ball } c \ r$
shows $f \text{ holomorphic_on } B$
proof (*rule holomorphic_on_subset*)
show $f \text{ holomorphic_on UNIV}$ **unfolding** *holomorphic_on_def*
proof
fix $z :: \text{complex}$
from $\langle \neg \text{bdd_above } A \rangle$ **obtain** r **where** $r: r \in A \ r > \text{norm } (z - c)$
by (*meson bdd_aboveI not_le*)
with *assms*(2) **have** $f \text{ holomorphic_on ball } c \ r$ **by** *blast*
moreover from r **have** $z \in \text{ball } c \ r$ **by** (*auto simp: dist_norm norm_minus_commute*)
ultimately show $f \text{ field_differentiable at } z$
by (*auto simp: holomorphic_on_def at_within_open*[of $_ \text{ball } c \ r$])
qed
qed *auto*

lemma *holomorphic_on_balls_imp_entire'*:
assumes $\wedge r. r > 0 \implies f \text{ holomorphic_on ball } c \ r$
shows $f \text{ holomorphic_on } B$
proof (*rule holomorphic_on_balls_imp_entire*)
show $\neg \text{bdd_above } \{(0::\text{real}) <..\}$ **unfolding** *bdd_above_def*
by (*meson greaterThan_iff gt_ex less_le_not_le order_le_less_trans*)
qed (*use assms in auto*)

lemma *holomorphic_on_minus* [*holomorphic_intros*]: $f \text{ holomorphic_on } A \implies (\lambda z. -(f z)) \text{ holomorphic_on } A$
by (*metis field_differentiable_minus holomorphic_on_def*)

lemma *holomorphic_on_add* [*holomorphic_intros*]:
 $\llbracket f \text{ holomorphic_on } A; g \text{ holomorphic_on } A \rrbracket \implies (\lambda z. f z + g z) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_add*)

lemma *holomorphic_on_diff* [*holomorphic_intros*]:
 $\llbracket f \text{ holomorphic_on } A; g \text{ holomorphic_on } A \rrbracket \implies (\lambda z. f z - g z) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_diff*)

lemma *holomorphic_on_mult* [*holomorphic_intros*]:
 $\llbracket f \text{ holomorphic_on } A; g \text{ holomorphic_on } A \rrbracket \implies (\lambda z. f z * g z) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_mult*)

lemma *holomorphic_on_inverse* [*holomorphic_intros*]:
 $\llbracket f \text{ holomorphic_on } A; \bigwedge z. z \in A \implies f z \neq 0 \rrbracket \implies (\lambda z. \text{inverse } (f z)) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_inverse*)

lemma *holomorphic_on_divide* [*holomorphic_intros*]:
 $\llbracket f \text{ holomorphic_on } A; g \text{ holomorphic_on } A; \bigwedge z. z \in A \implies g z \neq 0 \rrbracket \implies (\lambda z. f z / g z) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_divide*)

lemma *holomorphic_on_power* [*holomorphic_intros*]:
 $f \text{ holomorphic_on } A \implies (\lambda z. (f z) ^ n) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_power*)

lemma *holomorphic_on_power_int* [*holomorphic_intros*]:
assumes *nz*: $n \geq 0 \vee (\forall x \in A. f x \neq 0)$ **and** *f*: $f \text{ holomorphic_on } A$
shows $(\lambda x. f x \text{ powi } n) \text{ holomorphic_on } A$
proof (*cases n ≥ 0*)
case *True*
have $(\lambda x. f x ^ \text{nat } n) \text{ holomorphic_on } A$
by (*simp add: f holomorphic_on_power*)
with *True* **show** *?thesis*
by (*simp add: power_int_def*)
next
case *False*
hence $(\lambda x. \text{inverse } (f x ^ \text{nat } (-n))) \text{ holomorphic_on } A$
using *nz* **by** (*auto intro!: holomorphic_intros f*)
with *False* **show** *?thesis*
by (*simp add: power_int_def power_inverse*)
qed

lemma *holomorphic_on_sum* [*holomorphic_intros*]:
 $(\bigwedge i. i \in I \implies (f i) \text{ holomorphic_on } A) \implies (\lambda x. \text{sum } (\lambda i. f i x) I) \text{ holomorphic_on } A$
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_sum*)

lemma *holomorphic_on_prod* [*holomorphic_intros*]:
 $(\bigwedge i. i \in I \implies (f i) \text{ holomorphic_on } A) \implies (\lambda x. \text{prod } (\lambda i. f i x) I) \text{ holomorphic_on } A$
by (*induction I rule: infinite_finite_induct*) (*auto intro: holomorphic_intros*)

lemma *holomorphic_pochhammer* [*holomorphic_intros*]:
 $f \text{ holomorphic_on } A \implies (\lambda s. \text{pochhammer } (f s) n) \text{ holomorphic_on } A$
by (*induction n*) (*auto intro!: holomorphic_intros simp: pochhammer_Suc*)

lemma *holomorphic_on_scaleR* [*holomorphic_intros*]:
 f *holomorphic_on* $A \implies (\lambda x. c *_{\mathbb{R}} f x)$ *holomorphic_on* A
by (*auto simp: scaleR_conv_of_real intro!: holomorphic_intros*)

lemma *holomorphic_on_Un* [*holomorphic_intros*]:
assumes f *holomorphic_on* A f *holomorphic_on* B *open* A *open* B
shows f *holomorphic_on* $(A \cup B)$
by (*metis Un_iff assms holomorphic_on_open open_Un*)

lemma *holomorphic_on>If_Un* [*holomorphic_intros*]:
assumes f *holomorphic_on* A g *holomorphic_on* B *open* A *open* B
assumes $\bigwedge z. z \in A \implies z \in B \implies f z = g z$
shows $(\lambda z. \text{if } z \in A \text{ then } f z \text{ else } g z)$ *holomorphic_on* $(A \cup B)$ (**is** *?h holomorphic_on* $_$)
proof (*intro holomorphic_on_Un*)
note $\langle f \text{ holomorphic_on } A \rangle$
also have f *holomorphic_on* $A \iff ?h \text{ holomorphic_on } A$
by (*intro holomorphic_cong*) *auto*
finally show \dots .
next
note $\langle g \text{ holomorphic_on } B \rangle$
also have g *holomorphic_on* $B \iff ?h \text{ holomorphic_on } B$
using *assms* **by** (*intro holomorphic_cong*) *auto*
finally show \dots .
qed (*use assms in auto*)

lemma *holomorphic_derivI*:
 $\llbracket f \text{ holomorphic_on } S; \text{ open } S; x \in S \rrbracket \implies (f \text{ has_field_derivative } \text{deriv } f x)$ (*at* x *within* T)
by (*metis DERIV_deriv_iff_field_differentiable_at_within_open holomorphic_on_def has_field_derivative_at_within*)

lemma *complex_derivative_transform_within_open*:
 $\llbracket f \text{ holomorphic_on } s; g \text{ holomorphic_on } s; \text{ open } s; z \in s; \bigwedge w. w \in s \implies f w = g w \rrbracket$
 $\implies \text{deriv } f z = \text{deriv } g z$
by (*smt (verit) DERIV_imp_deriv has_field_derivative_transform_within_open holomorphic_on_open*)

lemma *holomorphic_on_compose_cnj_cnj*:
assumes f *holomorphic_on* $\text{cnj } A$ *open* A
shows $\text{cnj} \circ f \circ \text{cnj}$ *holomorphic_on* A
proof –
have [*simp*]: *open* $(\text{cnj } A)$
unfolding *image_cnj_conv_vimage_cnj* **using** *assms* **by** (*intro open_vimage*)
auto
show *?thesis*
using *assms* **unfolding** *holomorphic_on_def*

by (auto intro!: field_differentiable_cnj_cnj simp: at_within_open_NO_MATCH)
qed

lemma *holomorphic_nonconstant*:
assumes *holf*: f holomorphic_on S **and** open S $\xi \in S$ deriv f $\xi \neq 0$
shows \neg f constant_on S
by (rule nonzero_deriv_nonconstant [of f deriv f ξ ξ S])
 (use *assms* in \langle auto simp: holomorphic_derivI \rangle)

6.19.3 Analyticity on a set

definition *analytic_on* (infixl (*analytic'_on*) 50)
where f analytic_on $S \equiv \forall x \in S. \exists \varepsilon. 0 < \varepsilon \wedge f$ holomorphic_on (ball x ε)

named_theorems *analytic_intros* introduction rules for proving analyticity

lemma *analytic_imp_holomorphic*: f analytic_on $S \implies f$ holomorphic_on S
unfolding *analytic_on_def* *holomorphic_on_def*
using *centre_in_ball* *field_differentiable_at_within* *field_differentiable_within_open*
by *blast*

lemma *analytic_on_open*: open $S \implies f$ analytic_on $S \iff f$ holomorphic_on S
by (meson *analytic_imp_holomorphic* *analytic_on_def* *holomorphic_on_subset* *openE*)

lemma *constant_on_imp_analytic_on*:
assumes f constant_on A open A
shows f analytic_on A
by (simp add: *analytic_on_open* *assms* *constant_on_imp_holomorphic_on*)

lemma *analytic_on_imp_differentiable_at*:
 f analytic_on $S \implies x \in S \implies f$ field_differentiable (at x)
using *analytic_on_def* *holomorphic_on_imp_differentiable_at* **by** *auto*

lemma *analytic_at_imp_isCont*:
assumes f analytic_on $\{z\}$
shows *isCont* f z
by (meson *analytic_on_imp_differentiable_at* *assms* *field_differentiable_imp_continuous_at* *insertCI*)

lemma *analytic_at_neq_imp_eventually_neq*:
assumes f analytic_on $\{x\}$ f $x \neq c$
shows *eventually* ($\lambda y. f$ $y \neq c$) (at x)
using *analytic_at_imp_isCont* *assms* *isContD* *tendsto_imp_eventually_ne* **by**
blast

lemma *analytic_on_subset*: f analytic_on $S \implies T \subseteq S \implies f$ analytic_on T
by (auto simp: *analytic_on_def*)

1930

lemma *analytic_on_Un*: f *analytic_on* $(S \cup T) \longleftrightarrow f$ *analytic_on* $S \wedge f$ *analytic_on* T

by (*auto simp: analytic_on_def*)

lemma *analytic_on_Union*: f *analytic_on* $(\bigcup \mathcal{T}) \longleftrightarrow (\forall T \in \mathcal{T}. f$ *analytic_on* $T)$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_UN*: f *analytic_on* $(\bigcup_{i \in I} S i) \longleftrightarrow (\forall i \in I. f$ *analytic_on* $(S i))$

by (*auto simp: analytic_on_def*)

lemma *analytic_on_holomorphic*:

f *analytic_on* $S \longleftrightarrow (\exists T. \text{open } T \wedge S \subseteq T \wedge f$ *holomorphic_on* $T)$

(*is ?lhs = ?rhs*)

proof –

have $?lhs \longleftrightarrow (\exists T. \text{open } T \wedge S \subseteq T \wedge f$ *analytic_on* $T)$

proof *safe*

assume f *analytic_on* S

then have $\forall x \in \bigcup \{U. \text{open } U \wedge f$ *analytic_on* $U\}. \exists \varepsilon > 0. f$ *holomorphic_on* *ball* $x \ \varepsilon$

using *analytic_on_def* **by** *force*

moreover have $S \subseteq \bigcup \{U. \text{open } U \wedge f$ *analytic_on* $U\}$

using $\langle f$ *analytic_on* $S \rangle$

by (*smt (verit, best) open_ball_Union_iff_analytic_on_def analytic_on_open_centre_in_ball mem_Collect_eq subsetI*)

ultimately show $\exists T. \text{open } T \wedge S \subseteq T \wedge f$ *analytic_on* T

unfolding *analytic_on_def*

by (*metis (mono_tags, lifting) mem_Collect_eq open_Union*)

next

fix T

assume $\text{open } T \ S \subseteq T \ f$ *analytic_on* T

then show f *analytic_on* S

by (*metis analytic_on_subset*)

qed

also have $\dots \longleftrightarrow ?rhs$

by (*auto simp: analytic_on_open*)

finally show $?thesis$.

qed

lemma *analytic_on_linear* [*analytic_intros,simp*]: $((*) \ c)$ *analytic_on* S

by (*auto simp add: analytic_on_holomorphic*)

lemma *analytic_on_const* [*analytic_intros,simp*]: $(\lambda z. \ c)$ *analytic_on* S

by (*metis analytic_on_def holomorphic_on_const zero_less_one*)

lemma *analytic_on_ident* [*analytic_intros,simp*]: $(\lambda x. \ x)$ *analytic_on* S

by (*simp add: analytic_on_def gt_ex*)

```

lemma analytic_on_id [analytic_intros]: id analytic_on S
  unfolding id_def by (rule analytic_on_ident)

lemma analytic_on_scaleR [analytic_intros]: f analytic_on A  $\implies$   $(\lambda w. x *_R f w)$ 
analytic_on A
  by (metis analytic_on_holomorphic holomorphic_on_scaleR)

lemma analytic_on_compose:
  assumes f: f analytic_on S
    and g: g analytic_on (f ' S)
  shows  $(g \circ f)$  analytic_on S
unfolding analytic_on_def
proof (intro ballI)
  fix x
  assume x: x  $\in S$ 
  then obtain e where e: 0 < e and fh: f holomorphic_on ball x e using f
    by (metis analytic_on_def)
  obtain e' where e': 0 < e' and gh: g holomorphic_on ball (f x) e' using g
    by (metis analytic_on_def g image_eqI x)
  have isCont f x
    by (metis analytic_on_imp_differentiable_at field_differentiable_imp_continuous_at
f x)
  with e' obtain d where d: 0 < d and fd: f ' ball x d  $\subseteq$  ball (f x) e'
    by (auto simp: continuous_at_ball)
  have g  $\circ$  f holomorphic_on ball x (min d e)
    by (meson fd fh gh holomorphic_on_compose_gen holomorphic_on_subset
image_mono min.cobounded1 min.cobounded2 subset_ball)
  then show  $\exists e > 0. g \circ f$  holomorphic_on ball x e
    by (metis d e min_less_iff_conj)
qed

lemma analytic_on_compose_gen:
  f analytic_on S  $\implies$  g analytic_on T  $\implies$   $(\bigwedge z. z \in S \implies f z \in T)$ 
 $\implies$  g  $\circ$  f analytic_on S
  by (metis analytic_on_compose analytic_on_subset image_subset_iff)

lemma analytic_on_neg [analytic_intros]:
  f analytic_on S  $\implies$   $(\lambda z. -(f z))$  analytic_on S
  by (metis analytic_on_holomorphic holomorphic_on_minus)

lemma analytic_on_add [analytic_intros]:
  assumes f: f analytic_on S
    and g: g analytic_on S
  shows  $(\lambda z. f z + g z)$  analytic_on S
unfolding analytic_on_def
proof (intro ballI)
  fix z
  assume z: z  $\in S$ 
  then obtain e where e: 0 < e and fh: f holomorphic_on ball z e using f

```

```

    by (metis analytic_on_def)
  obtain e' where e': 0 < e' and gh: g holomorphic_on ball z e' using g
  by (metis analytic_on_def g z)
  have (λz. f z + g z) holomorphic_on ball z (min e e')
  by (metis fh gh holomorphic_on_add holomorphic_on_subset linorder_linear
    min_def subset_ball)
  then show ∃ e>0. (λz. f z + g z) holomorphic_on ball z e
  by (metis e e' min_less_iff_conj)
qed

```

```

lemma analytic_on_mult [analytic_intros]:
  assumes f: f analytic_on S
  and g: g analytic_on S
  shows (λz. f z * g z) analytic_on S
unfolding analytic_on_def
proof (intro ballI)
  fix z
  assume z: z ∈ S
  then obtain e where e: 0 < e and fh: f holomorphic_on ball z e using f
  by (metis analytic_on_def)
  obtain e' where e': 0 < e' and gh: g holomorphic_on ball z e' using g
  by (metis analytic_on_def g z)
  have (λz. f z * g z) holomorphic_on ball z (min e e')
  by (metis fh gh holomorphic_on_mult holomorphic_on_subset min.absorb_iff2
    min_def subset_ball)
  then show ∃ e>0. (λz. f z * g z) holomorphic_on ball z e
  by (metis e e' min_less_iff_conj)
qed

```

```

lemma analytic_on_diff [analytic_intros]:
  assumes f: f analytic_on S and g: g analytic_on S
  shows (λz. f z - g z) analytic_on S
proof -
  have (λz. - g z) analytic_on S
  by (simp add: analytic_on_neg g)
  then have (λz. f z + - g z) analytic_on S
  using analytic_on_add f by blast
  then show ?thesis
  by fastforce
qed

```

```

lemma analytic_on_inverse [analytic_intros]:
  assumes f: f analytic_on S
  and nz: (λz. z ∈ S ⇒ f z ≠ 0)
  shows (λz. inverse (f z)) analytic_on S
unfolding analytic_on_def
proof (intro ballI)
  fix z
  assume z: z ∈ S

```


then obtain e where $e: 0 < e$ and $fh: f$ holomorphic_on ball $z e$ using f
by (metis analytic_on_def)
have continuous_on (ball $z e$) f
by (metis fh holomorphic_on_imp_continuous_on)
then obtain e' where $e': 0 < e'$ and $nz': \bigwedge y. \text{dist } z y < e' \implies f y \neq 0$
by (metis open_ball centre_in_ball continuous_on_open_avoid $e z nz'$)
have ($\lambda z. \text{inverse } (f z)$) holomorphic_on ball $z (\text{min } e e')$
using fh holomorphic_on_inverse holomorphic_on_open nz' **by** fastforce
then show $\exists e > 0. (\lambda z. \text{inverse } (f z))$ holomorphic_on ball $z e$
by (metis $e e' \text{ min_less_iff_conj}$)
qed

lemma analytic_on_divide [analytic_intros]:
assumes $f: f$ analytic_on S **and** $g: g$ analytic_on S
and $nz: (\bigwedge z. z \in S \implies g z \neq 0)$
shows ($\lambda z. f z / g z$) analytic_on S
unfolding divide_inverse **by** (metis analytic_on_inverse analytic_on_mult $f g nz$)

lemma analytic_on_power [analytic_intros]:
 f analytic_on $S \implies (\lambda z. (f z) ^ n)$ analytic_on S
by (induct n) (auto simp: analytic_on_mult)

lemma analytic_on_power_int [analytic_intros]:
assumes $nz: n \geq 0 \vee (\forall x \in A. f x \neq 0)$ **and** $f: f$ analytic_on A
shows ($\lambda x. f x \text{ powi } n$) analytic_on A
proof (cases $n \geq 0$)
case True
have ($\lambda x. f x ^ \text{nat } n$) analytic_on A
using analytic_on_power f **by** blast
with True **show** ?thesis
by (simp add: power_int_def)
next
case False
hence ($\lambda x. \text{inverse } (f x ^ \text{nat } (-n))$) analytic_on A
using nz **by** (auto intro!: analytic_intros f)
with False **show** ?thesis
by (simp add: power_int_def power_inverse)

qed

lemma analytic_on_sum [analytic_intros]:
 $(\bigwedge i. i \in I \implies (f i)$ analytic_on $S) \implies (\lambda x. \text{sum } (\lambda i. f i x) I)$ analytic_on S
by (induct I rule: infinite_finite_induct) (auto simp: analytic_on_add)

lemma analytic_on_prod [analytic_intros]:
 $(\bigwedge i. i \in I \implies (f i)$ analytic_on $S) \implies (\lambda x. \text{prod } (\lambda i. f i x) I)$ analytic_on S
by (induct I rule: infinite_finite_induct) (auto simp: analytic_on_mult)

lemma analytic_on_gbinomial [analytic_intros]:

f analytic_on $A \implies (\lambda w. f w \text{ gchoose } n)$ analytic_on A
unfolding *gbinomial_prod_rev* **by** (*intro analytic_intros*) *auto*

lemma *deriv_left_inverse*:

assumes f holomorphic_on S **and** g holomorphic_on T
and open S **and** open T
and $f' S \subseteq T$
and [*simp*]: $\bigwedge z. z \in S \implies g (f z) = z$
and $w \in S$
shows $\text{deriv } f w * \text{deriv } g (f w) = 1$

proof –

have $\text{deriv } f w * \text{deriv } g (f w) = \text{deriv } g (f w) * \text{deriv } f w$
by (*simp add: algebra_simps*)
also have $\dots = \text{deriv } (g \circ f) w$
using *assms*

by (*metis analytic_on_imp_differentiable_at analytic_on_open deriv_chain image_subset_iff*)

also have $\dots = \text{deriv } id w$

proof (*rule complex_derivative_transform_within_open [where s=S]*)

show $g \circ f$ holomorphic_on S

by (*rule assms holomorphic_on_compose_gen holomorphic_intros*)+

qed (*use assms in auto*)

also have $\dots = 1$

by *simp*

finally show *?thesis* .

qed

6.19.4 Analyticity at a point

lemma *analytic_at_ball*:

f analytic_on $\{z\} \iff (\exists e. 0 < e \wedge f$ holomorphic_on ball $z e)$

by (*metis analytic_on_def singleton_iff*)

lemma *analytic_at*:

f analytic_on $\{z\} \iff (\exists s. \text{open } s \wedge z \in s \wedge f$ holomorphic_on $s)$

by (*metis analytic_on_holomorphic empty_subsetI insert_subset*)

lemma *holomorphic_on_imp_analytic_at*:

assumes f holomorphic_on A open A $z \in A$

shows f analytic_on $\{z\}$

using *assms* **by** (*meson analytic_at*)

lemma *analytic_on_analytic_at*:

f analytic_on $s \iff (\forall z \in s. f$ analytic_on $\{z\})$

by (*metis analytic_at_ball analytic_on_def*)

lemma *analytic_at_two*:

f analytic_on $\{z\} \wedge g$ analytic_on $\{z\} \iff$

$(\exists S. \text{open } S \wedge z \in S \wedge f$ holomorphic_on $S \wedge g$ holomorphic_on $S)$

```

(is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain S T
    where st: open S z ∈ S f holomorphic_on S
              open T z ∈ T g holomorphic_on T
    by (auto simp: analytic_at)
  then show ?rhs
    by (metis Int_iff holomorphic_on_subset inf_le1 inf_le2 open_Int)
next
  assume ?rhs
  then show ?lhs
    by (force simp add: analytic_at)
qed

```

6.19.5 Combining theorems for derivative with “analytic at” hypotheses

```

lemma
  assumes f analytic_on {z} g analytic_on {z}
  shows complex_derivative_add_at: deriv (λw. f w + g w) z = deriv f z + deriv
  g z
    and complex_derivative_diff_at: deriv (λw. f w - g w) z = deriv f z - deriv
  g z
    and complex_derivative_mult_at: deriv (λw. f w * g w) z =
    f z * deriv g z + deriv f z * g z
proof -
  show deriv (λw. f w + g w) z = deriv f z + deriv g z
    using analytic_on_imp_differentiable_at assms by auto
  show deriv (λw. f w - g w) z = deriv f z - deriv g z
    using analytic_on_imp_differentiable_at assms by force
  obtain S where open S z ∈ S f holomorphic_on S g holomorphic_on S
    using assms by (metis analytic_at_two)
  then show deriv (λw. f w * g w) z = f z * deriv g z + deriv f z * g z
    by (simp add: DERIV_imp_deriv [OF DERIV_mult!] holomorphic_derivI)
qed

```

```

lemma deriv_cmult_at:
  f analytic_on {z} ⇒ deriv (λw. c * f w) z = c * deriv f z
  by (auto simp: complex_derivative_mult_at)

```

```

lemma deriv_cmult_right_at:
  f analytic_on {z} ⇒ deriv (λw. f w * c) z = deriv f z * c
  by (auto simp: complex_derivative_mult_at)

```

6.19.6 Complex differentiation of sequences and series

```

lemma has_complex_derivative_sequence:
  fixes S :: complex set

```

assumes *cvs*: *convex S*
and *df*: $\bigwedge n x. x \in S \implies (f\ n\ \text{has_field_derivative}\ f'\ n\ x)$ (at *x* within *S*)
and *conv*: $\bigwedge e. 0 < e \implies \exists N. \forall n x. n \geq N \longrightarrow x \in S \longrightarrow \text{norm}\ (f'\ n\ x - g'\ x) \leq e$
and $\exists x\ l. x \in S \wedge ((\lambda n. f\ n\ x) \longrightarrow l)$ *sequentially*
shows $\exists g. \forall x \in S. ((\lambda n. f\ n\ x) \longrightarrow g\ x)$ *sequentially* \wedge
 $(g\ \text{has_field_derivative}\ (g'\ x))$ (at *x* within *S*)
proof –
from *assms* **obtain** *x l* **where** *x*: $x \in S$ **and** *tf*: $((\lambda n. f\ n\ x) \longrightarrow l)$ *sequentially*
by *blast*
show *?thesis*
unfolding *has_field_derivative_def*
proof (*rule has_derivative_sequence [OF cvs _ _ x]*)
show $(\lambda n. f\ n\ x) \longrightarrow l$
by (*rule tf*)
next
have **: $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{cmod}\ (f'\ n\ x * h - g'\ x * h) \leq \varepsilon * \text{cmod}\ h$
if $\varepsilon > 0$ **for** $\varepsilon :: \text{real}$
by (*metis that left_diff_distrib mult_right_mono norm_ge_zero norm_mult conv*)
show $\bigwedge e. e > 0 \implies \forall_F\ n\ \text{in}\ \text{sequentially}. \forall x \in S. \forall h. \text{cmod}\ (f'\ n\ x * h - g'\ x * h) \leq e * \text{cmod}\ h$
unfolding *eventually_sequentially* **by** (*blast intro: ***)
qed (*metis has_field_derivative_def df*)
qed

lemma *has_complex_derivative_series*:

fixes *S* :: *complex set*
assumes *cvs*: *convex S*
and *df*: $\bigwedge n x. x \in S \implies (f\ n\ \text{has_field_derivative}\ f'\ n\ x)$ (at *x* within *S*)
and *conv*: $\bigwedge e. 0 < e \implies \exists N. \forall n x. n \geq N \longrightarrow x \in S$
 $\longrightarrow \text{cmod}\ ((\sum i < n. f'\ i\ x) - g'\ x) \leq e$
and $\exists x\ l. x \in S \wedge ((\lambda n. f\ n\ x)\ \text{sums}\ l)$
shows $\exists g. \forall x \in S. ((\lambda n. f\ n\ x)\ \text{sums}\ g\ x) \wedge ((g\ \text{has_field_derivative}\ g'\ x)$ (at *x* within *S*)
proof –
from *assms* **obtain** *x l* **where** *x*: $x \in S$ **and** *sf*: $((\lambda n. f\ n\ x)\ \text{sums}\ l)$
by *blast*
{ fix $\varepsilon :: \text{real}$ **assume** $e: \varepsilon > 0$
then obtain *N* **where** *N*: $\forall n x. n \geq N \longrightarrow x \in S$
 $\longrightarrow \text{cmod}\ ((\sum i < n. f'\ i\ x) - g'\ x) \leq \varepsilon$
by (*metis conv*)
have $\exists N. \forall n \geq N. \forall x \in S. \forall h. \text{cmod}\ ((\sum i < n. h * f'\ i\ x) - g'\ x * h) \leq \varepsilon * \text{cmod}\ h$
proof (*rule exI [of _ N], clarify*)
fix *n y h*
assume $N \leq n\ y \in S$
have $\text{cmod}\ h * \text{cmod}\ ((\sum i < n. f'\ i\ y) - g'\ y) \leq \text{cmod}\ h * \varepsilon$
by (*simp add: N <N ≤ n> <y ∈ S> mult_le_cancel_left*)

```

    then show cmod (( $\sum_{i < n} h * f' i y$ ) -  $g' y * h$ )  $\leq \varepsilon * cmod h$ 
      by (simp add: norm_mult [symmetric] field_simps sum_distrib_left)
  qed
} note ** = this
show ?thesis
unfolding has_field_derivative_def
proof (rule has_derivative_series [OF cvs _ _ x])
  fix n x
  assume x  $\in S$ 
  then show ((f n) has_derivative ( $\lambda z. z * f' n x$ )) (at x within S)
    by (metis df has_field_derivative_def mult_commute_abs)
  next show (( $\lambda n. f n x$ ) sums l)
    by (rule sf)
  next show  $\bigwedge e. e > 0 \implies \forall_F n$  in sequentially.  $\forall x \in S. \forall h. cmod ((\sum_{i < n} h * f' i x) - g' x * h) \leq e * cmod h$ 
    unfolding eventually_sequentially by (blast intro: **)
  qed
qed

```

6.19.7 Taylor on Complex Numbers

lemma sum_Suc_reindex:

```

  fixes f :: nat  $\Rightarrow$  'a::ab_group_add
  shows sum f {0..n} = f 0 - f (Suc n) + sum ( $\lambda i. f (Suc i)$ ) {0..n}
  by (induct n) auto

```

lemma field_Taylor:

```

  assumes S: convex S
  and f:  $\bigwedge i x. x \in S \implies i \leq n \implies (f i \text{ has\_field\_derivative } f (Suc i) x)$  (at x within S)
  and B:  $\bigwedge x. x \in S \implies norm (f (Suc n) x) \leq B$ 
  and w:  $w \in S$ 
  and z:  $z \in S$ 
  shows norm(f 0 z - ( $\sum_{i \leq n} f i w * (z-w)^i / (fact i)$ ))
     $\leq B * norm(z - w)^{Suc n} / fact n$ 

```

proof -

```

  have wzs: closed_segment w z  $\subseteq S$  using assms
  by (metis convex_contains_segment)
  { fix u
    assume u  $\in$  closed_segment w z
    then have u  $\in S$ 
      by (metis wzs subsetD)
    have *: ( $\sum_{i \leq n} f i u * (- of_nat i * (z-u)^{i-1}) / (fact i) + f (Suc i) u * (z-u)^i / (fact i)$ ) =
      f (Suc n) u * (z-u)^n / (fact n)
    proof (induction n)
      case 0 show ?case by simp
    next
      case (Suc n)

```

```

have ( $\sum_{i \leq \text{Suc } n} f \ i \ u * (- \text{of\_nat } i * (z-u) ^ (i - 1)) / (\text{fact } i) +$ 
 $f (\text{Suc } i) \ u * (z-u) ^ i / (\text{fact } i) =$ 
 $f (\text{Suc } n) \ u * (z-u) ^ n / (\text{fact } n) +$ 
 $f (\text{Suc } (\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n) / (\text{fact } (\text{Suc } n)) -$ 
 $f (\text{Suc } n) \ u * ((1 + \text{of\_nat } n) * (z-u) ^ n) / (\text{fact } (\text{Suc } n))$ 
using Suc by simp
also have  $\dots = f (\text{Suc } (\text{Suc } n)) \ u * (z-u) ^ \text{Suc } n / (\text{fact } (\text{Suc } n))$ 
proof -
  have ( $\text{fact}(\text{Suc } n) *$ 
 $(f(\text{Suc } n) \ u * (z-u) ^ n / (\text{fact } n) +$ 
 $f(\text{Suc}(\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n) / (\text{fact}(\text{Suc } n)) -$ 
 $f(\text{Suc } n) \ u * ((1 + \text{of\_nat } n) * (z-u) ^ n) / (\text{fact}(\text{Suc } n))) =$ 
 $((\text{fact}(\text{Suc } n)) * (f(\text{Suc } n) \ u * (z-u) ^ n)) / (\text{fact } n) +$ 
 $((\text{fact}(\text{Suc } n)) * (f(\text{Suc}(\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n)) / (\text{fact}(\text{Suc } n)))$ 
  -
   $((\text{fact}(\text{Suc } n)) * (f(\text{Suc } n) \ u * (\text{of\_nat}(\text{Suc } n) * (z-u) ^ n))) / (\text{fact}(\text{Suc } n))$ 
  -
  by (simp add: algebra_simps del: fact_Suc)
also have  $\dots = ((\text{fact } (\text{Suc } n)) * (f (\text{Suc } n) \ u * (z-u) ^ n)) / (\text{fact } n) +$ 
 $(f (\text{Suc } (\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n)) -$ 
 $(f (\text{Suc } n) \ u * ((1 + \text{of\_nat } n) * (z-u) ^ n))$ 
by (simp del: fact_Suc)
also have  $\dots = (\text{of\_nat } (\text{Suc } n) * (f (\text{Suc } n) \ u * (z-u) ^ n)) +$ 
 $(f (\text{Suc } (\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n)) -$ 
 $(f (\text{Suc } n) \ u * ((1 + \text{of\_nat } n) * (z-u) ^ n))$ 
by (simp only: fact_Suc of_nat_mult ac_simps) simp
also have  $\dots = f (\text{Suc } (\text{Suc } n)) \ u * ((z-u) * (z-u) ^ n)$ 
by (simp add: algebra_simps)
finally show ?thesis
by (simp add: mult_left_cancel [where c = (fact (Suc n)), THEN iffD1])
del: fact_Suc
qed
finally show ?case .
qed
have ( $(\lambda v. (\sum_{i \leq n} f \ i \ v * (z - v) ^ i / (\text{fact } i)))$ 
 $\text{has\_field\_derivative } f (\text{Suc } n) \ u * (z-u) ^ n / (\text{fact } n)$ 
 $(\text{at } u \ \text{within } S)$ 
unfolding * [symmetric]
by (rule derivative_eq_intros assms ⟨u ∈ S⟩ refl | auto simp: field_simps)
note sum_deriv = this
{ fix u
assume u: u ∈ closed_segment w z
then have us: u ∈ S
by (metis wzs subsetD)
have  $\text{norm } (f (\text{Suc } n) \ u) * \text{norm } (z - u) ^ n \leq \text{norm } (f (\text{Suc } n) \ u) * \text{norm}$ 
 $(u - z) ^ n$ 
by (metis norm_minus_commute order_refl)
also have  $\dots \leq \text{norm } (f (\text{Suc } n) \ u) * \text{norm } (z - w) ^ n$ 
by (metis mult_left_mono norm_ge_zero power_mono segment_bound [OF

```

```

u])
  also have ... ≤ B * norm (z - w) ^ n
    by (metis norm_ge_zero zero_le_power mult_right_mono B [OF us])
  finally have norm (f (Suc n) u) * norm (z - u) ^ n ≤ B * norm (z - w) ^
n .
} note cmod_bound = this
have (∑ i≤n. f i z * (z - z) ^ i / (fact i)) = (∑ i≤n. (f i z / (fact i)) * 0 ^ i)
  by simp
also have ... = f 0 z / (fact 0)
  by (subst sum_zero_power) simp
finally have norm (f 0 z - (∑ i≤n. f i w * (z - w) ^ i / (fact i)))
  ≤ norm ((∑ i≤n. f i w * (z - w) ^ i / (fact i)) -
    (∑ i≤n. f i z * (z - z) ^ i / (fact i)))
  by (simp add: norm_minus_commute)
also have ... ≤ B * norm (z - w) ^ n / (fact n) * norm (w - z)
proof (rule field_differentiable_bound)
  show ∧x. x ∈ closed_segment w z ⇒
    ((λξ. ∑ i≤n. f i ξ * (z - ξ) ^ i / fact i) has_field_derivative f (Suc n) x
  * (z - x) ^ n / fact n)
    (at x within closed_segment w z)
  using DERIV_subset sum_deriv wzs by blast
qed (auto simp: norm_divide norm_mult norm_power divide_le_cancel cmod_bound)
also have ... ≤ B * norm (z - w) ^ Suc n / (fact n)
  by (simp add: algebra_simps norm_minus_commute)
finally show ?thesis .
qed

```

lemma *complex_Taylor*:

```

assumes S: convex S
  and f: ∧i x. x ∈ S ⇒ i ≤ n ⇒ (f i has_field_derivative f (Suc i) x) (at
x within S)
  and B: ∧x. x ∈ S ⇒ cmod (f (Suc n) x) ≤ B
  and w: w ∈ S
  and z: z ∈ S
  shows cmod(f 0 z - (∑ i≤n. f i w * (z-w) ^ i / (fact i))) ≤ B * cmod(z -
w) ^ (Suc n) / fact n
  using assms by (rule field_Taylor)

```

Something more like the traditional MVT for real components

lemma *complex_mvt_line*:

```

assumes ∧u. u ∈ closed_segment w z ⇒ (f has_field_derivative f'(u)) (at u)
  shows ∃u. u ∈ closed_segment w z ∧ Re(f z) - Re(f w) = Re(f'(u) * (z - w))
proof -
  define φ where φ ≡ λt. (1 - t) *R w + t *R z
  have twz: ∧t. φ t = w + t *R (z - w)
  by (simp add: φ_def real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib)
  note assms[unfolded has_field_derivative_def, derivative_intros]
  have *: ∧x. [0 ≤ x; x ≤ 1]
    ⇒ (Re ∘ f ∘ φ has_derivative Re ∘ (*) (f' (φ x)) ∘ (λt. t *R (z - w)))

```

(at x within {0..1})
unfolding φ_def
by (intro derivative_eq_intros has_derivative_at_withinI)
 (auto simp: in_segment scaleR_right_diff_distrib)
obtain x where $0 < x < 1$ (Re $\circ f \circ \varphi$) 1 -
 (Re $\circ f \circ \varphi$) 0 = (Re $\circ (*)$ (f' (φ x)) $\circ (\lambda t. t *_{\mathbb{R}} (z - w))$) (1 - 0)
using mvt_simple [OF zero_less_one *] **by** force
then show ?thesis
unfolding φ_def
by (smt (verit) comp_apply in_segment(1) scaleR_left_distrib scaleR_one
 scaleR_zero_left)
qed

lemma complex_Taylor_mvt:

assumes $\bigwedge i x. \llbracket x \in \text{closed_segment } w z; i \leq n \rrbracket \implies ((f\ i) \text{ has_field_derivative } f\ (\text{Suc } i)\ x)$ (at x)

shows $\exists u. u \in \text{closed_segment } w z \wedge$

$$\begin{aligned} & \text{Re } (f\ 0\ z) = \\ & \text{Re } \left(\left(\sum_{i=0..n} f\ i\ w * (z-w)^{\wedge i} / (\text{fact } i) \right) + \right. \\ & \left. (f\ (\text{Suc } n)\ u * (z-u)^{\wedge n} / (\text{fact } n)) * (z-w) \right) \end{aligned}$$

proof -

{ **fix** u

assume u: u \in closed_segment w z

have $(\sum_{i=0..n} f\ (\text{Suc } i)\ u * (z-u)^{\wedge i} - \text{of_nat } i * (f\ i\ u * (z-u)^{\wedge (i - \text{Suc } 0)})) /$

$$\frac{(f\ (\text{Suc } i)\ u * (z-u)^{\wedge i} - \text{of_nat } i * (f\ i\ u * (z-u)^{\wedge (i - \text{Suc } 0)}))}{(\text{fact } i)} =$$

$$f\ (\text{Suc } 0)\ u -$$

$$\frac{(f\ (\text{Suc } (\text{Suc } n))\ u * ((z-u)^{\wedge \text{Suc } n} - \text{of_nat } (\text{Suc } n)) * (z-u)^{\wedge n} * f\ (\text{Suc } n)\ u)}{(\text{fact } (\text{Suc } n)) +$$

$$(\sum_{i=0..n} f\ (\text{Suc } (\text{Suc } i))\ u * ((z-u)^{\wedge \text{Suc } i} - \text{of_nat } (\text{Suc } i)) * (f\ (\text{Suc } i)$$

$$u * (z-u)^{\wedge i}) / (\text{fact } (\text{Suc } i))$$

$$\frac{(f\ (\text{Suc } (\text{Suc } i))\ u * ((z-u)^{\wedge \text{Suc } i} - \text{of_nat } (\text{Suc } i)) * (f\ (\text{Suc } i)$$

by (subst sum_Suc_reindex) simp

also have ... = f (Suc 0) u -

$$\frac{(f\ (\text{Suc } (\text{Suc } n))\ u * ((z-u)^{\wedge \text{Suc } n} - \text{of_nat } (\text{Suc } n)) * (z-u)^{\wedge n} * f\ (\text{Suc } n)\ u)}{(\text{fact } (\text{Suc } n)) +$$

$$(\sum_{i=0..n} f\ (\text{Suc } (\text{Suc } i))\ u * ((z-u)^{\wedge \text{Suc } i} / (\text{fact } (\text{Suc } i)) -$$

$$f\ (\text{Suc } i)\ u * (z-u)^{\wedge i} / (\text{fact } i))$$

by (simp only: diff_divide_distrib fact_cancel ac_simps)

also have ... = f (Suc 0) u -

$$\frac{(f\ (\text{Suc } (\text{Suc } n))\ u * (z-u)^{\wedge \text{Suc } n} - \text{of_nat } (\text{Suc } n) * (z-u)^{\wedge n} * f\ (\text{Suc } n)\ u)}{(\text{fact } (\text{Suc } n)) +$$

$$f\ (\text{Suc } (\text{Suc } n))\ u * (z-u)^{\wedge \text{Suc } n} / (\text{fact } (\text{Suc } n)) - f\ (\text{Suc } 0)\ u$$

by (subst sum_Suc_diff) auto


```

also have ... = f (Suc n) u * (z-u) ^ n / (fact n)
  by (simp only: algebra_simps diff_divide_distrib fact_cancel)
finally have *: (∑ i = 0..n. (f (Suc i) u * (z-u) ^ i
  - of_nat i * (f i u * (z-u) ^ (i - Suc 0)))) / (fact i) =
  f (Suc n) u * (z-u) ^ n / (fact n) .
have ((λu. ∑ i = 0..n. f i u * (z-u) ^ i / (fact i)) has_field_derivative
  f (Suc n) u * (z-u) ^ n / (fact n)) (at u)
  unfolding * [symmetric]
  by (rule derivative_eq_intros assms u refl | auto simp: field_simps)+
}
then show ?thesis
  apply (cut_tac complex_mvz_line [of w z λu. ∑ i = 0..n. f i u * (z-u) ^ i /
(fact i)
    λu. (f (Suc n) u * (z-u) ^ n / (fact n))])
  apply (auto simp add: intro: open_closed_segment)
  done
qed

end

```

6.20 Complex Transcendental Functions

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

```

theory Complex_Transcendental
imports
  Complex_Analysis_Basics Summation_Tests HOL-Library.Periodic_Fun
begin

```

6.20.1 Möbius transformations

definition *moebius* $a\ b\ c\ d \equiv (\lambda z. (a*z+b) / (c*z+d :: 'a :: field))$

theorem *moebius_inverse*:

assumes $a * d \neq b * c$ $c * z + d \neq 0$

shows $moebius\ d\ (-b)\ (-c)\ a\ (moebius\ a\ b\ c\ d\ z) = z$

proof -

from *assms* **have** $(-c) * moebius\ a\ b\ c\ d\ z + a \neq 0$ **unfolding** *moebius_def*

by (*simp add: field_simps*)

with *assms* **show** ?thesis

unfolding *moebius_def* **by** (*simp add: moebius_def divide_simps*) (*simp add: algebra_simps*)?

qed

lemma *moebius_inverse'*:

assumes $a * d \neq b * c$ $c * z - a \neq 0$

shows $moebius\ a\ b\ c\ d\ (moebius\ d\ (-b)\ (-c)\ a\ z) = z$

using *assms moebius_inverse* [*of d a -b -c z*]

1942

by (auto simp: algebra_simps)

lemma *cmod_add_real_less*:

assumes $Im\ z \neq 0$ $r \neq 0$

shows $cmod\ (z + r) < cmod\ z + |r|$

proof (cases z)

case (Complex $x\ y$)

then have $0 < y * y$

using *assms mult_neg_neg* by force

with *assms* have $r * x / |r| < sqrt\ (x*x + y*y)$

by (simp add: *real_less_sqrt power2_eq_square*)

then show *?thesis* using *assms Complex*

apply (simp add: *cmod_def*)

apply (rule *power2_less_imp_less*, auto)

apply (simp add: *power2_eq_square field_simps*)

done

qed

lemma *cmod_diff_real_less*: $Im\ z \neq 0 \implies x \neq 0 \implies cmod\ (z - x) < cmod\ z + |x|$

using *cmod_add_real_less* [of $z - x$]

by *simp*

lemma *cmod_square_less_1_plus*:

assumes $Im\ z = 0 \implies |Re\ z| < 1$

shows $(cmod\ z)^2 < 1 + cmod\ (1 - z^2)$

proof (cases $Im\ z = 0 \vee Re\ z = 0$)

case True

with *assms abs_square_less_1* show *?thesis*

by (force simp add: *Re_power2 Im_power2 cmod_def*)

next

case False

with *cmod_diff_real_less* [of $1 - z^2\ 1$] show *?thesis*

by (simp add: *norm_power Im_power2*)

qed

6.20.2 The Exponential Function

lemma *norm_exp_i_times* [simp]: $norm\ (exp(i * of_real\ y)) = 1$

by *simp*

lemma *norm_exp_imaginary*: $norm(exp\ z) = 1 \implies Re\ z = 0$

by *simp*

lemma *field_differentiable_within_exp*: *exp* field_differentiable (at z within s)

using *DERIV_exp field_differentiable_at_within field_differentiable_def* by *blast*

lemma *continuous_within_exp*:

```

fixes z::'a::{real_normed_field,banach}
shows continuous (at z within s) exp
by (simp add: continuous_at_imp_continuous_within)

```

```

lemma holomorphic_on_exp [holomorphic_intros]: exp holomorphic_on s
by (simp add: field_differentiable_within_exp holomorphic_on_def)

```

```

lemma holomorphic_on_exp' [holomorphic_intros]:
  f holomorphic_on s  $\implies$  ( $\lambda x$ . exp (f x)) holomorphic_on s
using holomorphic_on_compose[OF _ holomorphic_on_exp] by (simp add:
o_def)

```

```

lemma exp_analytic_on [analytic_intros]:
assumes f analytic_on A
shows ( $\lambda z$ . exp (f z)) analytic_on A
by (metis analytic_on_holomorphic assms holomorphic_on_exp')

```

```

lemma
assumes  $\bigwedge w$ .  $w \in A \implies \exp (f w) = w$ 
assumes f holomorphic_on A  $z \in A$  open A
shows deriv_complex_logarithm: deriv f z = 1 / z
and has_field_derivative_complex_logarithm: (f has_field_derivative 1 / z)
(at z)
proof -
have [simp]:  $z \neq 0$ 
using assms(1)[of z] assms(3) by auto
have deriv [derivative_intros]: (f has_field_derivative deriv f z) (at z)
using assms holomorphic_derivI by blast
have (( $\lambda w$ . w) has_field_derivative 1) (at z)
by (intro derivative_intros)
also have ?this  $\longleftrightarrow$  (( $\lambda w$ . exp (f w)) has_field_derivative 1) (at z)
by (smt (verit, best) assms has_field_derivative_transform_within_open)
finally have (( $\lambda w$ . exp (f w)) has_field_derivative 1) (at z) .
moreover have (( $\lambda w$ . exp (f w)) has_field_derivative exp (f z) * deriv f z) (at
z)
by (rule derivative_eq_intros refl)+
ultimately have exp (f z) * deriv f z = 1
using DERIV_unique by blast
with assms show deriv f z = 1 / z
by (simp add: field_simps)
with deriv show (f has_field_derivative 1 / z) (at z)
by simp
qed

```

6.20.3 Euler and de Moivre formulas

The sine series times i

```

lemma sin_i_eq: ( $\lambda n$ . (i * sin_coeff n) *  $z^n$ ) sums (i * sin z)
proof -

```

have $(\lambda n. i * \sin_coeff\ n * {}_R z^{\wedge} n)$ *sums* $(i * \sin\ z)$
using *sin_converges sums_mult* **by** *blast*
then show *?thesis*
by *(simp add: scaleR_conv_of_real_field_simps)*
qed

theorem *exp_Euler*: $\exp(i * z) = \cos(z) + i * \sin(z)$

proof –

have $(\lambda n. (\cos_coeff\ n + i * \sin_coeff\ n) * z^{\wedge} n) = (\lambda n. (i * z)^{\wedge} n / {}_R (fact\ n))$
by *(force simp: cos_coeff_def sin_coeff_def scaleR_conv_of_real_field_simps elim!: evenE oddE)*
also have \dots *sums* $(\exp\ (i * z))$
by *(rule exp_converges)*
finally have $(\lambda n. (\cos_coeff\ n + i * \sin_coeff\ n) * z^{\wedge} n)$ *sums* $(\exp\ (i * z))$.
moreover have $(\lambda n. (\cos_coeff\ n + i * \sin_coeff\ n) * z^{\wedge} n)$ *sums* $(\cos\ z + i * \sin\ z)$
using *sums_add [OF cos_converges [of z] sin_i_eq [of z]]*
by *(simp add: field_simps scaleR_conv_of_real)*
ultimately show *?thesis*
using *sums_unique2* **by** *blast*
qed

corollary *exp_minus_Euler*: $\exp(-(i * z)) = \cos(z) - i * \sin(z)$
using *exp_Euler [of -z]* **by** *simp*

lemma *sin_exp_eq*: $\sin\ z = (\exp(i * z) - \exp(-(i * z))) / (2 * i)$
by *(simp add: exp_Euler exp_minus_Euler)*

lemma *sin_exp_eq'*: $\sin\ z = i * (\exp(-(i * z)) - \exp(i * z)) / 2$
by *(simp add: exp_Euler exp_minus_Euler)*

lemma *cos_exp_eq*: $\cos\ z = (\exp(i * z) + \exp(-(i * z))) / 2$
by *(simp add: exp_Euler exp_minus_Euler)*

theorem *Euler*: $\exp(z) = \text{of_real}(\exp(\text{Re}\ z)) * (\text{of_real}(\cos(\text{Im}\ z)) + i * \text{of_real}(\sin(\text{Im}\ z)))$
by *(simp add: Complex_eq cis.code exp_eq_polar)*

lemma *Re_sin*: $\text{Re}(\sin\ z) = \sin(\text{Re}\ z) * (\exp(\text{Im}\ z) + \exp(-(\text{Im}\ z))) / 2$
by *(simp add: sin_exp_eq field_simps Re_divide Im_exp)*

lemma *Im_sin*: $\text{Im}(\sin\ z) = \cos(\text{Re}\ z) * (\exp(\text{Im}\ z) - \exp(-(\text{Im}\ z))) / 2$
by *(simp add: sin_exp_eq field_simps Im_divide Re_exp)*

lemma *Re_cos*: $\text{Re}(\cos\ z) = \cos(\text{Re}\ z) * (\exp(\text{Im}\ z) + \exp(-(\text{Im}\ z))) / 2$
by *(simp add: cos_exp_eq field_simps Re_divide Re_exp)*

lemma *Im_cos*: $\text{Im}(\cos\ z) = \sin(\text{Re}\ z) * (\exp(-(\text{Im}\ z)) - \exp(\text{Im}\ z)) / 2$
by *(simp add: cos_exp_eq field_simps Im_divide Im_exp)*

lemma *Re_sin_pos*: $0 < \operatorname{Re} z \implies \operatorname{Re} z < \pi \implies \operatorname{Re} (\sin z) > 0$
by (*auto simp: Re_sin Im_sin add_pos_pos sin_gt_zero*)

lemma *Im_sin_nonneg*: $\operatorname{Re} z = 0 \implies 0 \leq \operatorname{Im} z \implies 0 \leq \operatorname{Im} (\sin z)$
by (*simp add: Re_sin Im_sin algebra_simps*)

lemma *Im_sin_nonneg2*: $\operatorname{Re} z = \pi \implies \operatorname{Im} z \leq 0 \implies 0 \leq \operatorname{Im} (\sin z)$
by (*simp add: Re_sin Im_sin algebra_simps*)

6.20.4 Relationships between real and complex trigonometric and hyperbolic functions

lemma *real_sin_eq* [*simp*]: $\operatorname{Re}(\sin(\operatorname{of_real} x)) = \sin x$
by (*simp add: sin_of_real*)

lemma *real_cos_eq* [*simp*]: $\operatorname{Re}(\cos(\operatorname{of_real} x)) = \cos x$
by (*simp add: cos_of_real*)

lemma *DeMoivre*: $(\cos z + i * \sin z) ^ n = \cos(n * z) + i * \sin(n * z)$
by (*metis exp_Euler [symmetric] exp_of_nat_mult mult.left_commute*)

lemma *exp_cnj*: $\operatorname{cnj}(\exp z) = \exp(\operatorname{cnj} z)$
by (*simp add: cis_cnj exp_eq_polar*)

lemma *cnj_sin*: $\operatorname{cnj}(\sin z) = \sin(\operatorname{cnj} z)$
by (*simp add: sin_exp_eq exp_cnj field_simps*)

lemma *cnj_cos*: $\operatorname{cnj}(\cos z) = \cos(\operatorname{cnj} z)$
by (*simp add: cos_exp_eq exp_cnj field_simps*)

lemma *field_differentiable_at_sin*: *sin* *field_differentiable* *at* *z*
using *DERIV_sin field_differentiable_def* **by** *blast*

lemma *field_differentiable_within_sin*: *sin* *field_differentiable* (*at* *z* *within* *S*)
by (*simp add: field_differentiable_at_sin field_differentiable_at_within*)

lemma *field_differentiable_at_cos*: *cos* *field_differentiable* *at* *z*
using *DERIV_cos field_differentiable_def* **by** *blast*

lemma *field_differentiable_within_cos*: *cos* *field_differentiable* (*at* *z* *within* *S*)
by (*simp add: field_differentiable_at_cos field_differentiable_at_within*)

lemma *holomorphic_on_sin*: *sin* *holomorphic_on* *S*
by (*simp add: field_differentiable_within_sin holomorphic_on_def*)

lemma *holomorphic_on_cos*: *cos* *holomorphic_on* *S*
by (*simp add: field_differentiable_within_cos holomorphic_on_def*)

lemma *holomorphic_on_sin'* [*holomorphic_intros*]:
assumes *f holomorphic_on A*
shows $(\lambda x. \sin (f x)) \text{ holomorphic_on } A$
using *holomorphic_on_compose[OF assms holomorphic_on_sin]* **by** (*simp add: o_def*)

lemma *holomorphic_on_cos'* [*holomorphic_intros*]:
assumes *f holomorphic_on A*
shows $(\lambda x. \cos (f x)) \text{ holomorphic_on } A$
using *holomorphic_on_compose[OF assms holomorphic_on_cos]* **by** (*simp add: o_def*)

lemma *analytic_on_sin* [*analytic_intros*]: *f analytic_on A* $\implies (\lambda w. \sin (f w)) \text{ analytic_on } A$
and *analytic_on_cos* [*analytic_intros*]: *f analytic_on A* $\implies (\lambda w. \cos (f w)) \text{ analytic_on } A$
and *analytic_on_sinh* [*analytic_intros*]: *f analytic_on A* $\implies (\lambda w. \sinh (f w)) \text{ analytic_on } A$
and *analytic_on_cosh* [*analytic_intros*]: *f analytic_on A* $\implies (\lambda w. \cosh (f w)) \text{ analytic_on } A$
unfolding *sin_exp_eq cos_exp_eq sinh_def cosh_def*
by (*auto intro!: analytic_intros*)

lemma *analytic_on_tan* [*analytic_intros*]:
f analytic_on A $\implies (\bigwedge z. z \in A \implies \cos (f z) \neq 0) \implies (\lambda w. \tan (f w)) \text{ analytic_on } A$
and *analytic_on_cot* [*analytic_intros*]:
f analytic_on A $\implies (\bigwedge z. z \in A \implies \sin (f z) \neq 0) \implies (\lambda w. \cot (f w)) \text{ analytic_on } A$
and *analytic_on_tanh* [*analytic_intros*]:
f analytic_on A $\implies (\bigwedge z. z \in A \implies \cosh (f z) \neq 0) \implies (\lambda w. \tanh (f w)) \text{ analytic_on } A$
unfolding *tan_def cot_def tanh_def* **by** (*auto intro!: analytic_intros*)

6.20.5 More on the Polar Representation of Complex Numbers

lemma *exp_Complex*: $\exp(\text{Complex } r \ t) = \text{of_real}(\exp r) * \text{Complex}(\cos t) (\sin t)$
using *Complex_eq Euler complex.sel* **by** *presburger*

lemma *exp_eq_1*: $\exp z = 1 \iff \text{Re}(z) = 0 \wedge (\exists n::\text{int}. \text{Im}(z) = \text{of_int}(2 * n) * \pi)$
(is ?lhs = ?rhs)

proof

assume *exp z = 1*
then have *Re z = 0*
by (*metis exp_eq_one_iff norm_exp_eq_Re norm_one*)
with $\langle ?lhs \rangle$ **show** *?rhs*

by (metis Re_exp cos_one_2pi_int exp_zero mult.commute mult_1 of_int_mult of_int_numeral one_complex.simps(1))

next

assume ?rhs then show ?lhs

using Im_exp Re_exp complex_eq_iff

by (simp add: cos_one_2pi_int cos_one_sin_zero mult.commute)

qed

lemma exp_eq: $\exp w = \exp z \iff (\exists n::int. w = z + (\text{of_int } (2 * n) * \text{pi}) * i)$
 (is ?lhs = ?rhs)

proof -

have $\exp w = \exp z \iff \exp (w-z) = 1$

by (simp add: exp_diff)

also have $\dots \iff (\text{Re } w = \text{Re } z \wedge (\exists n::int. \text{Im } w - \text{Im } z = \text{of_int } (2 * n) * \text{pi}))$

by (simp add: exp_eq_1)

also have $\dots \iff ?rhs$

by (auto simp: algebra_simps intro!: complex_eqI)

finally show ?thesis .

qed

lemma exp_complex_eqI: $|\text{Im } w - \text{Im } z| < 2 * \text{pi} \implies \exp w = \exp z \implies w = z$
 by (auto simp: exp_eq abs_mult)

lemma exp_integer_2pi:

assumes $n \in \mathbb{Z}$

shows $\exp((2 * n * \text{pi}) * i) = 1$

by (metis assms cis_conv_exp cis_multiple_2pi mult.assoc mult.commute)

lemma exp_plus_2pin [simp]: $\exp (z + i * (\text{of_int } n * (\text{of_real } \text{pi} * 2))) = \exp z$

by (simp add: exp_eq)

lemma exp_integer_2pi_plus1:

assumes $n \in \mathbb{Z}$

shows $\exp(((2 * n + 1) * \text{pi}) * i) = -1$

using exp_integer_2pi [OF assms]

by (metis cis_conv_exp cis_mult cis_pi distrib_left mult.commute mult.right_neutral)

lemma inj_on_exp_pi:

fixes $z::\text{complex}$ shows inj_on exp (ball z pi)

proof (clarsimp simp: inj_on_def exp_eq)

fix $y \ n$

assume $\text{dist } z (y + 2 * \text{of_int } n * \text{of_real } \text{pi} * i) < \text{pi}$

$\text{dist } z \ y < \text{pi}$

then have $\text{dist } y (y + 2 * \text{of_int } n * \text{of_real } \text{pi} * i) < \text{pi} + \text{pi}$

using dist_commute_lessI dist_triangle_less_add by blast

then have $\text{norm } (2 * \text{of_int } n * \text{of_real } \text{pi} * i) < 2 * \text{pi}$

by (simp add: dist_norm)

then show $n = 0$

by (auto simp: norm_mult)
qed

lemma cmod_add_squared:

fixes r1 r2::real
shows (cmod (r1 * exp (i * ϑ 1) + r2 * exp (i * ϑ 2)))² = r1² + r2² + 2 * r1 * r2 * cos (ϑ 1 - ϑ 2) (is (cmod (?z1 + ?z2))² = ?rhs)
proof -
have (cmod (?z1 + ?z2))² = (?z1 + ?z2) * cnj (?z1 + ?z2)
by (rule complex_norm_square)
also have ... = (?z1 * cnj ?z1 + ?z2 * cnj ?z2) + (?z1 * cnj ?z2 + cnj ?z1 * ?z2)
by (simp add: algebra_simps)
also have ... = (norm ?z1)² + (norm ?z2)² + 2 * Re (?z1 * cnj ?z2)
unfolding complex_norm_square [symmetric] cnj_add_mult_eq_Re by simp
also have ... = ?rhs
by (simp add: norm_mult) (simp add: exp_Euler complex_is_Real_iff [THEN iffD1] cos_diff algebra_simps)
finally show ?thesis
using of_real_eq_iff by blast
qed

lemma cmod_diff_squared:

fixes r1 r2::real
shows (cmod (r1 * exp (i * ϑ 1) - r2 * exp (i * ϑ 2)))² = r1² + r2² - 2*r1*r2*cos (ϑ 1 - ϑ 2)
using cmod_add_squared [of r1 _ -r2] by simp

lemma polar_convergence:

fixes R::real
assumes $\bigwedge j. r\ j > 0\ R > 0$
shows (($\lambda j. r\ j * \exp (i * \vartheta\ j)$) \longrightarrow (R * exp (i * Θ))) \longleftrightarrow
(r \longrightarrow R) \wedge ($\exists k. (\lambda j. \vartheta\ j - \text{of_int } (k\ j) * (2 * \pi)) \longrightarrow \Theta$) (is
(?z \longrightarrow ?Z) = ?rhs)
proof
assume L: ?z \longrightarrow ?Z
have rR: r \longrightarrow R
using tendsto_norm [OF L] assms by (auto simp: norm_mult abs_of_pos)
moreover obtain k where ($\lambda j. \vartheta\ j - \text{of_int } (k\ j) * (2 * \pi)$) \longrightarrow Θ
proof -
have cos ($\vartheta\ j - \Theta$) = ((r j)² + R² - (norm(?z j - ?Z))²) / (2 * R * r j) for
j
using assms by (auto simp: cmod_diff_squared less_le)
moreover have ($\lambda j. ((r\ j)^2 + R^2 - (\text{norm}(\vartheta\ j - \Theta))^2) / (2 * R * r\ j)$)
 \longrightarrow ((R² + R² - (norm(?Z - ?Z))²) / (2 * R * R))
by (intro L rR tendsto_intros) (use <R > 0> in force)
moreover have ((R² + R² - (norm(?Z - ?Z))²) / (2 * R * R)) = 1
using <R > 0> by (simp add: power2_eq_square field_split_simps)
ultimately have ($\lambda j. \cos (\vartheta\ j - \Theta)$) \longrightarrow 1


```

    by auto
  then show ?thesis
    using that cos_diff_limit_1 by blast
qed
ultimately show ?rhs
  by metis
next
assume R: ?rhs
show ?z  $\longrightarrow$  ?Z
proof (rule tendsto_mult)
  show  $(\lambda x. \text{complex\_of\_real } (r x)) \longrightarrow \text{of\_real } R$ 
    using R by (auto simp: tendsto_of_real_iff)
  obtain k where  $(\lambda j. \vartheta j - \text{of\_int } (k j) * (2 * \pi)) \longrightarrow \Theta$ 
    using R by metis
  then have  $(\lambda j. \text{complex\_of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi))) \longrightarrow \text{of\_real } \Theta$ 
    using tendsto_of_real_iff by force
  then have  $(\lambda j. \text{exp } (i * \text{of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi)))) \longrightarrow \text{exp } (i * \Theta)$ 
    using tendsto_mult [OF tendsto_const] isCont_exp isCont_tendsto_compose
  by blast
  moreover have  $\text{exp } (i * \text{of\_real } (\vartheta j - \text{of\_int } (k j) * (2 * \pi))) = \text{exp } (i * \vartheta j)$ 
    for j
    unfolding exp_eq
    by (rule_tac x=- k j in exI) (auto simp: algebra_simps)
  ultimately show  $(\lambda j. \text{exp } (i * \vartheta j)) \longrightarrow \text{exp } (i * \Theta)$ 
    by auto
qed
qed

lemma exp_i_ne_1:
  assumes  $0 < x < 2 * \pi$ 
  shows  $\text{exp}(i * \text{of\_real } x) \neq 1$ 
  by (smt (verit) Im_i_times_Re_complex_of_real assms exp_complex_eqI exp_zero
  zero_complex.sel(2))

lemma sin_eq_0:
  fixes z::complex
  shows  $\sin z = 0 \iff (\exists n::\text{int}. z = \text{of\_real}(n * \pi))$ 
  by (simp add: sin_exp_eq exp_eq)

lemma cos_eq_0:
  fixes z::complex
  shows  $\cos z = 0 \iff (\exists n::\text{int}. z = \text{complex\_of\_real}(n * \pi) + \text{of\_real } \pi/2)$ 
  using sin_eq_0 [of z - of_real pi/2]
  by (simp add: sin_diff algebra_simps)

lemma cos_eq_1:
  fixes z::complex

```

1950

shows $\cos z = 1 \iff (\exists n::int. z = \text{complex_of_real}(2 * n * \pi))$
by (*metis Re_complex_of_real cos_of_real cos_one_2pi_int cos_one_sin_zero mult.commute of_real_1 sin_eq_0*)

lemma *csin_eq_1*:

fixes $z::\text{complex}$

shows $\sin z = 1 \iff (\exists n::int. z = \text{of_real}(2 * n * \pi) + \text{of_real } \pi/2)$

using *cos_eq_1 [of z - of_real pi/2]*

by (*simp add: cos_diff algebra_simps*)

lemma *csin_eq_minus1*:

fixes $z::\text{complex}$

shows $\sin z = -1 \iff (\exists n::int. z = \text{complex_of_real}(2 * n * \pi) + 3/2*\pi)$

(**is** $_ = ?rhs$)

proof -

have $\sin z = -1 \iff \sin (-z) = 1$

by (*simp add: equation_minus_iff*)

also have $\dots \iff (\exists n::int. z = -\text{of_real}(2 * n * \pi) - \text{of_real } \pi/2)$

by (*metis (mono_tags, lifting) add_uminus_conv_diff csin_eq_1 equation_minus_iff minus_add_distrib*)

also have $\dots = ?rhs$

apply *safe*

apply (*rule_tac [2] x=-(x+1) in exI*)

apply (*rule_tac x=-(x+1) in exI*)

apply (*simp_all add: algebra_simps*)

done

finally show *?thesis* .

qed

lemma *ccos_eq_minus1*:

fixes $z::\text{complex}$

shows $\cos z = -1 \iff (\exists n::int. z = \text{complex_of_real}(2 * n * \pi) + \pi)$

using *csin_eq_1 [of z - of_real pi/2]*

by (*simp add: sin_diff algebra_simps equation_minus_iff*)

lemma *sin_eq_1*: $\sin x = 1 \iff (\exists n::int. x = (2 * n + 1 / 2) * \pi)$

(**is** $_ = ?rhs$)

proof -

have $\sin x = 1 \iff \sin (\text{complex_of_real } x) = 1$

by (*metis of_real_1 one_complex_simps(1) real_sin_eq sin_of_real*)

also have $\dots \iff (\exists n::int. x = \text{of_real}(2 * n * \pi) + \text{of_real } \pi/2)$

by (*metis csin_eq_1 Re_complex_of_real id_apply of_real_add of_real_divide of_real_eq_id of_real_numeral*)

also have $\dots = ?rhs$

by (*auto simp: algebra_simps*)

finally show *?thesis* .

qed

lemma *sin_eq_minus1*: $\sin x = -1 \iff (\exists n::int. x = (2*n + 3/2) * \pi)$ (**is** $_ = ?rhs$)

= ?rhs)

proof -

have $\sin x = -1 \iff \sin(\text{complex_of_real } x) = -1$

by (metis Re_complex_of_real_of_real_def scaleR_minus1_left sin_of_real)

also have $\dots \iff (\exists n::\text{int}. x = \text{of_real}(2 * n * \pi) + 3/2 * \pi)$

by (metis Re_complex_of_real csin_eq_minus1_id_apply_of_real_add_of_real_eq_id)

also have $\dots = ?rhs$

by (auto simp: algebra_simps)

finally show ?thesis .

qed

lemma *cos_eq_minus1*: $\cos x = -1 \iff (\exists n::\text{int}. x = (2 * n + 1) * \pi)$
 (is _ = ?rhs)

proof -

have $\cos x = -1 \iff \cos(\text{complex_of_real } x) = -1$

by (metis Re_complex_of_real_of_real_def scaleR_minus1_left cos_of_real)

also have $\dots \iff (\exists n::\text{int}. x = \text{of_real}(2 * n * \pi) + \pi)$

by (metis ccos_eq_minus1_id_apply_of_real_add_of_real_eq_id_of_real_eq_iff)

also have $\dots = ?rhs$

by (auto simp: algebra_simps)

finally show ?thesis .

qed

lemma *cos_gt_neg1*:

assumes $(t::\text{real}) \in \{-\pi < .. < \pi\}$

shows $\cos t > -1$

using *assms*

by *simp* (metis cos_minus cos_monotone_0_pi cos_monotone_minus_pi_0
 cos_pi linorder_le_cases)

lemma *dist_exp_i_1*: $\text{norm}(\exp(i * \text{of_real } t) - 1) = 2 * |\sin(t / 2)|$

proof -

have $\text{sqrt}(2 - \cos t * 2) = 2 * |\sin(t / 2)|$

using *cos_double_sin* [of $t/2$] **by** (*simp add: real_sqrt_mult*)

then show ?thesis

by (*simp add: exp_Euler cmod_def power2_diff sin_of_real cos_of_real algebra_simps*)

qed

lemma *sin_cx_2pi* [*simp*]: $\llbracket z = \text{of_int } m; \text{ even } m \rrbracket \implies \sin(z * \text{complex_of_real } \pi) = 0$

by (*simp add: sin_eq_0*)

lemma *cos_cx_2pi* [*simp*]: $\llbracket z = \text{of_int } m; \text{ even } m \rrbracket \implies \cos(z * \text{complex_of_real } \pi) = 1$

using *cos_eq_1* **by** *auto*

lemma *complex_sin_eq*:

fixes $w :: \text{complex}$

1952

shows $\sin w = \sin z \iff (\exists n \in \mathbb{Z}. w = z + \text{of_real}(2*n*\pi) \vee w = -z + \text{of_real}((2*n + 1)*\pi))$
(**is** ?lhs = ?rhs)

proof

assume ?lhs

then consider $\sin((w - z) / 2) = 0 \mid \cos((w + z) / 2) = 0$

by (metis divide_eq_0_iff nonzero_eq_divide_eq right_minus_eq sin_diff_sin zero_neq_numeral)

then show ?rhs

proof cases

case 1

then show ?thesis

by (simp add: sin_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)

next

case 2

then show ?thesis

by (simp add: cos_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)

qed

next

assume ?rhs

then consider $n::\text{int}$ **where** $w = z + \text{of_real}(2 * \text{of_int } n * \pi)$

$\mid n::\text{int}$ **where** $w = -z + \text{of_real}((2 * \text{of_int } n + 1) * \pi)$

using Ints_cases **by blast**

then show ?lhs

proof cases

case 1

then show ?thesis

using Periodic_Fun.sin.plus_of_int [of z n]

by (auto simp: algebra_simps)

next

case 2

then show ?thesis

using Periodic_Fun.sin.plus_of_int [of -z n]

apply (simp add: algebra_simps)

by (metis add.commute add.inverse_inverse add_diff_cancel_left diff_add_cancel sin_plus_pi)

qed

qed

lemma complex_cos_eq:

fixes $w :: \text{complex}$

shows $\cos w = \cos z \iff (\exists n \in \mathbb{Z}. w = z + \text{of_real}(2*n*\pi) \vee w = -z + \text{of_real}(2*n*\pi))$

(**is** ?lhs = ?rhs)

proof

assume ?lhs

then consider $\sin((w + z) / 2) = 0 \mid \sin((z - w) / 2) = 0$

by (metis mult_eq_0_iff cos_diff_cos right_minus_eq zero_neq_numeral)

then show ?rhs

```

proof cases
  case 1
    then obtain  $n$  where  $w + z = \text{of\_int } n * (\text{complex\_of\_real } \pi * 2)$ 
    by (auto simp: sin_eq_0 algebra_simps)
    then have  $w = -z + \text{of\_real}(2 * \text{of\_int } n * \pi)$ 
    by (auto simp: algebra_simps)
    then show ?thesis
    using Ints_of_int by blast
  next
    case 2
    then obtain  $n$  where  $z = w + \text{of\_int } n * (\text{complex\_of\_real } \pi * 2)$ 
    by (auto simp: sin_eq_0 algebra_simps)
    then have  $w = z + \text{complex\_of\_real}(2 * \text{of\_int}(-n) * \pi)$ 
    by (auto simp: algebra_simps)
    then show ?thesis
    using Ints_of_int by blast
  qed
next
  assume ?rhs
  then obtain  $n::\text{int}$  where  $w = z + \text{of\_real}(2 * \text{of\_int } n * \pi) \vee$ 
     $w = -z + \text{of\_real}(2 * n * \pi)$ 
  using Ints_cases by (metis of_int_mult of_int_numeral)
  then show ?lhs
  using Periodic_Fun.cos_plus_of_int [of  $z$   $n$ ]
  apply (simp add: algebra_simps)
  by (metis cos_plus_of_int cos_minus minus_add_cancel mult.commute)
qed

lemma sin_eq:
   $\sin x = \sin y \iff (\exists n \in \mathbb{Z}. x = y + 2 * n * \pi \vee x = -y + (2 * n + 1) * \pi)$ 
  using complex_sin_eq [of  $x$   $y$ ]
  by (simp only: sin_of_real Re_complex_of_real of_real_add [symmetric] of_real_minus
[symmetric] of_real_mult [symmetric] of_real_eq_iff)

lemma cos_eq:
   $\cos x = \cos y \iff (\exists n \in \mathbb{Z}. x = y + 2 * n * \pi \vee x = -y + 2 * n * \pi)$ 
  using complex_cos_eq [of  $x$   $y$ ] unfolding cos_of_real
  by (metis Re_complex_of_real of_real_add of_real_minus)

lemma sinh_complex:
  fixes  $z :: \text{complex}$ 
  shows  $(\exp z - \text{inverse}(\exp z)) / 2 = -i * \sin(i * z)$ 
  by (simp add: sin_exp_eq field_split_simps exp_minus)

lemma sin_i_times:
  fixes  $z :: \text{complex}$ 
  shows  $\sin(i * z) = i * ((\exp z - \text{inverse}(\exp z)) / 2)$ 
  using sinh_complex by auto

```

lemma *sinh_real*:

fixes $x :: \text{real}$

shows $\text{of_real}((\exp x - \text{inverse}(\exp x)) / 2) = -i * \sin(i * \text{of_real } x)$

by (*simp add: exp_of_real sin_i_times*)

lemma *cosh_complex*:

fixes $z :: \text{complex}$

shows $(\exp z + \text{inverse}(\exp z)) / 2 = \cos(i * z)$

by (*simp add: cos_exp_eq field_split_simps exp_minus exp_of_real*)

lemma *cosh_real*:

fixes $x :: \text{real}$

shows $\text{of_real}((\exp x + \text{inverse}(\exp x)) / 2) = \cos(i * \text{of_real } x)$

by (*simp add: cos_exp_eq field_split_simps exp_minus exp_of_real*)

lemmas *cos_i_times = cosh_complex [symmetric]*

lemma *norm_cos_squared*:

$\text{norm}(\cos z) ^ 2 = \cos(\text{Re } z) ^ 2 + (\exp(\text{Im } z) - \text{inverse}(\exp(\text{Im } z))) ^ 2 / 4$

proof (*cases z*)

case (*Complex x1 x2*)

then show *?thesis*

apply (*simp only: cos_add cmod_power2 cos_of_real sin_of_real Complex_eq*)

apply (*simp add: cos_exp_eq sin_exp_eq exp_minus exp_of_real Re_divide Im_divide power_divide*)

apply (*simp only: left_diff_distrib [symmetric] power_mult_distrib sin_squared_eq*)

apply (*simp add: power2_eq_square field_split_simps*)

done

qed

lemma *norm_sin_squared*:

$\text{norm}(\sin z) ^ 2 = (\exp(2 * \text{Im } z) + \text{inverse}(\exp(2 * \text{Im } z)) - 2 * \cos(2 * \text{Re } z)) / 4$

using *cos_double_sin [of Re z]*

apply (*simp add: sin_cos_eq norm_cos_squared exp_minus mult.commute [of _ 2] exp_double*)

apply (*simp add: algebra_simps power2_eq_square*)

done

lemma *exp_uminus_Im*: $\exp(-\text{Im } z) \leq \exp(\text{cmod } z)$

using *abs_Im_le_cmod linear_order_trans* **by** *fastforce*

lemma *norm_cos_le*:

fixes $z :: \text{complex}$

shows $\text{norm}(\cos z) \leq \exp(\text{norm } z)$

proof –

have $\text{Im } z \leq \text{cmod } z$

using *abs_Im_le_cmod abs_le_D1* **by** *auto*

then have $\exp(-\text{Im } z) + \exp(\text{Im } z) \leq \exp(\text{cmod } z) * 2$

```

  by (metis exp_uminus_Im add_mono exp_le_cancel_iff mult_2_right)
  then show ?thesis
  by (force simp add: cos_exp_eq norm_divide intro: order_trans [OF norm_triangle_ineq])
qed

```

lemma *norm_cos_plus1_le*:

```

  fixes z::complex
  shows  $\text{norm}(1 + \cos z) \leq 2 * \exp(\text{norm } z)$ 
  by (metis mult_2 norm_cos_le norm_ge_zero norm_one norm_triangle_mono
  one_le_exp_iff)

```

lemma *sinh_conv_sin*: $\sinh z = -i * \sin(i*z)$

```

  by (simp add: sinh_field_def sin_i_times exp_minus)

```

lemma *cosh_conv_cos*: $\cosh z = \cos(i*z)$

```

  by (simp add: cosh_field_def cos_i_times exp_minus)

```

lemma *tanh_conv_tan*: $\tanh z = -i * \tan(i*z)$

```

  by (simp add: tanh_def sinh_conv_sin cosh_conv_cos tan_def)

```

lemma *sin_conv_sinh*: $\sin z = -i * \sinh(i*z)$

```

  by (simp add: sinh_conv_sin)

```

lemma *cos_conv_cosh*: $\cos z = \cosh(i*z)$

```

  by (simp add: cosh_conv_cos)

```

lemma *tan_conv_tanh*: $\tan z = -i * \tanh(i*z)$

```

  by (simp add: tan_def sin_conv_sinh cos_conv_cosh tanh_def)

```

lemma *sinh_complex_eq_iff*:

```

  sinh (z :: complex) = sinh w  $\longleftrightarrow$ 
    ( $\exists n \in \mathbb{Z}. z = w - 2 * i * \text{of\_real } n * \text{of\_real } \pi \vee$ 
      $z = -(2 * \text{complex\_of\_real } n + 1) * i * \text{complex\_of\_real } \pi - w$ ) (is
  _ = ?rhs)

```

proof -

```

  have  $\sinh z = \sinh w \longleftrightarrow \sin(i * z) = \sin(i * w)$ 

```

```

  by (simp add: sinh_conv_sin)

```

```

  also have ...  $\longleftrightarrow$  ?rhs

```

```

  by (subst complex_sin_eq) (force simp: field_simps complex_eq_iff)

```

```

  finally show ?thesis .

```

qed

6.20.6 Taylor series for complex exponential, sine and cosine

```

declare power_Suc [simp del]

```

lemma *Taylor_exp_field*:

```

  fixes z::'a::{banach,real_normed_field}

```

```

  shows  $\text{norm}(\exp z - (\sum_{i \leq n} z^i / \text{fact } i)) \leq \exp(\text{norm } z) * (\text{norm } z \wedge \text{Suc } n)$ 

```

```

n) / fact n
proof (rule field_Taylor [of _ n  $\lambda k. \exp \exp (\text{norm } z) 0 z, \text{simplified}$ ])
  show convex (closed_segment 0 z)
    by (rule convex_closed_segment [of 0 z])
next
  fix k x
  assume  $x \in \text{closed\_segment } 0 z \text{ } k \leq n$ 
  show (exp_has_field_derivative exp x) (at x within closed_segment 0 z)
    using DERIV_exp DERIV_subset by blast
next
  fix x
  assume  $x: x \in \text{closed\_segment } 0 z$ 
  have norm (exp x)  $\leq \exp (\text{norm } x)$ 
    by (rule norm_exp)
  also have norm x  $\leq \text{norm } z$ 
    using x by (auto simp: closed_segment_def intro!: mult_left_le_one_le)
  finally show norm (exp x)  $\leq \exp (\text{norm } z)$ 
    by simp
qed auto

```

For complex z , a tighter bound than in the previous result

```

lemma Taylor_exp:
  norm (exp z - ( $\sum_{k \leq n} z^k / (\text{fact } k)$ ))  $\leq \exp |\text{Re } z| * (\text{norm } z)^{\wedge (\text{Suc } n)} / (\text{fact } n)$ 
proof (rule complex_Taylor [of _ n  $\lambda k. \exp \exp |\text{Re } z| 0 z, \text{simplified}$ ])
  show convex (closed_segment 0 z)
    by (rule convex_closed_segment [of 0 z])
next
  fix k x
  assume  $x \in \text{closed\_segment } 0 z \text{ } k \leq n$ 
  show (exp_has_field_derivative exp x) (at x within closed_segment 0 z)
    using DERIV_exp DERIV_subset by blast
next
  fix x
  assume  $x \in \text{closed\_segment } 0 z$ 
  then obtain u where  $x = \text{complex\_of\_real } u * z \text{ } 0 \leq u \text{ } u \leq 1$ 
    by (auto simp: closed_segment_def scaleR_conv_of_real)
  then have  $u * \text{Re } z \leq |\text{Re } z|$ 
    by (metis abs_ge_self abs_ge_zero mult.commute mult.right_neutral mult_mono)
  then show  $\text{Re } x \leq |\text{Re } z|$ 
    by (simp add: u)
qed auto

```

```

lemma
  assumes  $0 \leq u \text{ } u \leq 1$ 
  shows cmod_sin_le_exp: cmod (sin (u *R z))  $\leq \exp |\text{Im } z|$ 
    and cmod_cos_le_exp: cmod (cos (u *R z))  $\leq \exp |\text{Im } z|$ 
proof -
  have mono:  $\bigwedge u w z :: \text{real}. w \leq u \implies z \leq u \implies (w + z) / 2 \leq u$ 

```



```

  by simp
  have *: (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z)))) ) / 2 ≤ exp
|Im z|
  proof (rule mono)
    show cmod (exp (i * (u * z))) ≤ exp |Im z|
      using assms
      by (auto simp: abs_if mult_left_le_one_le not_less intro: order_trans [of _
0])
    show cmod (exp (- (i * (u * z)))) ≤ exp |Im z|
      using assms
      by (auto simp: abs_if mult_left_le_one_le mult_nonneg_nonpos intro: or-
der_trans [of _ 0])
  qed
  have cmod (sin (u *R z)) = cmod (exp (i * (u * z)) - exp (- (i * (u * z)))) / 2
    by (auto simp: scaleR_conv_of_real norm_mult norm_power sin_exp_eq
norm_divide)
  also have ... ≤ (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z)))) ) / 2
    by (intro divide_right_mono norm_triangle_ineq4) simp
  also have ... ≤ exp |Im z|
    by (rule *)
  finally show cmod (sin (u *R z)) ≤ exp |Im z| .
  have cmod (cos (u *R z)) = cmod (exp (i * (u * z)) + exp (- (i * (u * z)))) / 2
    by (auto simp: scaleR_conv_of_real norm_mult norm_power cos_exp_eq
norm_divide)
  also have ... ≤ (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z)))) ) / 2
    by (intro divide_right_mono norm_triangle_ineq) simp
  also have ... ≤ exp |Im z|
    by (rule *)
  finally show cmod (cos (u *R z)) ≤ exp |Im z| .
qed

```

lemma *Taylor_sin*:

$$\begin{aligned} & \text{norm}(\sin z - (\sum_{k \leq n}. \text{complex_of_real}(\text{sin_coeff } k) * z^k)) \\ & \leq \exp |Im z| * (\text{norm } z)^{\wedge} (\text{Suc } n) / (\text{fact } n) \end{aligned}$$

proof –

$$\text{have mono: } \bigwedge u \ w \ z :: \text{real. } w \leq u \implies z \leq u \implies w + z \leq u * 2$$

by arith

$$\begin{aligned} \text{have } *: & \text{cmod}(\sin z - \\ & (\sum_{i \leq n}. (-1)^{\wedge} (i \text{ div } 2) * (\text{if even } i \text{ then } \sin 0 \text{ else } \cos 0) * z^i / \\ & (\text{fact } i))) \\ & \leq \exp |Im z| * \text{cmod } z^{\wedge} \text{Suc } n / (\text{fact } n) \end{aligned}$$

proof (rule *complex_Taylor* [of *closed_segment 0 z n*

$$\lambda k \ x. (-1)^{\wedge} (k \text{ div } 2) * (\text{if even } k \text{ then } \sin x \text{ else } \cos x) \exp |Im z| \ 0 \ z, \text{ simplified}]$$

fix *k x*

$$\begin{aligned} \text{show } & ((\lambda x. (-1)^{\wedge} (k \text{ div } 2) * (\text{if even } k \text{ then } \sin x \text{ else } \cos x)) \text{ has_field_derivative} \\ & (-1)^{\wedge} (\text{Suc } k \text{ div } 2) * (\text{if odd } k \text{ then } \sin x \text{ else } \cos x)) \\ & (\text{at } x \text{ within } \text{closed_segment } 0 \ z) \end{aligned}$$

by (cases even *k*) (intro *derivative_eq_intros* | simp add: *power_Suc*) +

```

next
  fix  $x$ 
    assume  $x \in \text{closed\_segment } 0 z$ 
    then show  $\text{cmod } ((-1) \wedge (\text{Suc } n \text{ div } 2)) * (\text{if odd } n \text{ then } \sin x \text{ else } \cos x) \leq$ 
 $\text{exp } |\text{Im } z|$ 
    by (auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp
 $\text{cmod\_cos\_le\_exp}$ )
    qed
    have **:  $\bigwedge k. \text{complex\_of\_real } (\text{sin\_coeff } k) * z \wedge k$ 
 $= (-1) \wedge (k \text{ div } 2) * (\text{if even } k \text{ then } \sin 0 \text{ else } \cos 0) * z \wedge k / \text{of\_nat } (\text{fact}$ 
 $k)$ 
    by (auto simp: sin_coeff_def elim!: oddE)
    show ?thesis
    by (simp add: ** order_trans [OF _ *])
qed

```

lemma *Taylor_cos*:

```

 $\text{norm}(\cos z - (\sum_{k \leq n} \text{complex\_of\_real } (\text{cos\_coeff } k) * z \wedge k))$ 
 $\leq \text{exp } |\text{Im } z| * (\text{norm } z) \wedge \text{Suc } n / (\text{fact } n)$ 
proof -
  have mono:  $\bigwedge u w z :: \text{real}. w \leq u \implies z \leq u \implies w + z \leq u * 2$ 
    by arith
  have *:  $\text{cmod } (\cos z -$ 
 $(\sum_{i \leq n} (-1) \wedge (\text{Suc } i \text{ div } 2) * (\text{if even } i \text{ then } \cos 0 \text{ else } \sin 0) * z$ 
 $\wedge i / (\text{fact } i)))$ 
 $\leq \text{exp } |\text{Im } z| * \text{cmod } z \wedge \text{Suc } n / (\text{fact } n)$ 
  proof (rule complex_Taylor [of closed_segment 0 z n
 $\lambda k x. (-1) \wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos x \text{ else } \sin$ 
 $x)$ 
 $\text{exp } |\text{Im } z| 0 z, \text{simplified}]$ )
    fix  $k x$ 
    assume  $x \in \text{closed\_segment } 0 z$   $k \leq n$ 
    show  $((\lambda x. (-1) \wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos x \text{ else } \sin x))$ 
 $\text{has\_field\_derivative}$ 
 $(-1) \wedge \text{Suc } (k \text{ div } 2) * (\text{if odd } k \text{ then } \cos x \text{ else } \sin x))$ 
 $(\text{at } x \text{ within } \text{closed\_segment } 0 z)$ 
    by (cases even k) (intro derivative_eq_intros | simp add: power_Suc) +
  next
    fix  $x$ 
    assume  $x \in \text{closed\_segment } 0 z$ 
    then show  $\text{cmod } ((-1) \wedge \text{Suc } (n \text{ div } 2)) * (\text{if odd } n \text{ then } \cos x \text{ else } \sin x) \leq$ 
 $\text{exp } |\text{Im } z|$ 
    by (auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp
 $\text{cmod\_cos\_le\_exp}$ )
    qed
    have **:  $\bigwedge k. \text{complex\_of\_real } (\text{cos\_coeff } k) * z \wedge k$ 
 $= (-1) \wedge (\text{Suc } k \text{ div } 2) * (\text{if even } k \text{ then } \cos 0 \text{ else } \sin 0) * z \wedge k / \text{of\_nat}$ 
 $(\text{fact } k)$ 
    by (auto simp: cos_coeff_def elim!: evenE)

```

```

show ?thesis
  by (simp add: ** order_trans [OF _ *])
qed

declare power_Suc [simp]

32-bit Approximation to e
lemma e_approx_32:  $|exp(1) - 5837465777 / 2147483648| \leq (inverse(2 ^ 32)::real)$ 
  using Taylor_exp [of 1 14] exp_le
  apply (simp add: sum_distrib_right in_Reals_norm Re_exp atMost_nat_numeral
fact_numeral)
  apply (simp only: pos_le_divide_eq [symmetric])
  done

lemma e_less_272:  $exp\ 1 < (272/100::real)$ 
  using e_approx_32
  by (simp add: abs_if_split: if_split_asm)

lemma ln_272_gt_1:  $ln\ (272/100) > (1::real)$ 
  by (metis e_less_272 exp_less_cancel_iff exp_ln_iff less_trans ln_exp)

```

Apparently redundant. But many arguments involve integers.

```

lemma ln3_gt_1:  $ln\ 3 > (1::real)$ 
  by (simp add: less_trans [OF ln_272_gt_1])

```

6.20.7 The argument of a complex number (HOL Light version)

```

definition is_Arg :: [complex,real]  $\Rightarrow$  bool
  where is_Arg z r  $\equiv z = of\_real(norm\ z) * exp(i * of\_real\ r)$ 

definition Arg2pi :: complex  $\Rightarrow$  real
  where Arg2pi z  $\equiv$  if z = 0 then 0 else THE t. 0  $\leq$  t  $\wedge$  t < 2*pi  $\wedge$  is_Arg z t

lemma is_Arg_2pi_iff:  $is\_Arg\ z\ (r + of\_int\ k * (2 * pi)) \longleftrightarrow is\_Arg\ z\ r$ 
  by (simp add: algebra_simps is_Arg_def)

lemma is_Arg_eqI:
  assumes is_Arg z r and is_Arg z s and  $abs\ (r - s) < 2 * pi$  and  $z \neq 0$ 
  shows  $r = s$ 
  using assms unfolding is_Arg_def
  by (metis Im_i_times Re_complex_of_real exp_complex_eqI mult_cancel_left
mult_eq_0_iff)

```

This function returns the angle of a complex number from its representation in polar coordinates. Due to periodicity, its range is arbitrary. *Arg2pi* follows HOL Light in adopting the interval $[0, 2\pi)$. But we have the same periodicity issue with logarithms, and it is usual to adopt the same interval for the complex logarithm and argument functions. Further on down, we shall define

1960

both functions for the interval $(-\pi, \pi]$. The present version is provided for compatibility.

lemma *Arg2pi_0* [*simp*]: $\text{Arg2pi}(0) = 0$
by (*simp add: Arg2pi_def*)

lemma *Arg2pi_unique_lemma*:
assumes *is_Arg z t*
and *is_Arg z t'*
and $0 \leq t < 2\pi$
and $0 \leq t' < 2\pi$
and $z \neq 0$
shows $t' = t$
using *is_Arg_eqI* *assms* **by force**

lemma *Arg2pi*: $0 \leq \text{Arg2pi } z \wedge \text{Arg2pi } z < 2\pi \wedge \text{is_Arg } z (\text{Arg2pi } z)$

proof (*cases z=0*)

case True then show *?thesis*
by (*simp add: Arg2pi_def is_Arg_def*)

next

case False

obtain *t* **where** $0 \leq t < 2\pi$
and *ReIm*: $\text{Re } z / \text{cmod } z = \cos t \text{ Im } z / \text{cmod } z = \sin t$
using *sincos_total_2pi* [*OF complex_unit_circle*] [*OF False*]
by blast

then have *z*: *is_Arg z t*
unfolding *is_Arg_def*
using *t False ReIm*

by (*intro complex_eqI*) (*auto simp: exp_Euler sin_of_real cos_of_real field_split_simps*)
show *?thesis*

apply (*simp add: Arg2pi_def False*)
apply (*rule theI* [**where** $a=t$])
using *t z False*
apply (*auto intro: Arg2pi_unique_lemma*)
done

qed

corollary

shows *Arg2pi_ge_0*: $0 \leq \text{Arg2pi } z$
and *Arg2pi_lt_2pi*: $\text{Arg2pi } z < 2\pi$
and *Arg2pi_eq*: $z = \text{of_real}(\text{norm } z) * \exp(i * \text{of_real}(\text{Arg2pi } z))$
using *Arg2pi is_Arg_def* **by auto**

lemma *complex_norm_eq_1_exp*: $\text{norm } z = 1 \iff \exp(i * \text{of_real} (\text{Arg2pi } z)) = z$

by (*metis Arg2pi_eq cis_conv_exp mult.left_neutral norm_cis of_real_1*)

lemma *Arg2pi_unique*: $\llbracket \text{of_real } r * \exp(i * \text{of_real } a) = z; 0 < r; 0 \leq a; a < 2\pi \rrbracket \implies \text{Arg2pi } z = a$

by (*rule Arg2pi_unique_lemma* [*unfolded is_Arg_def, OF _ Arg2pi_eq*]) (*use*

Arg2pi [of z] in $\langle \text{auto simp: norm_mult} \rangle$

lemma cos_Arg2pi : $\text{cmod } z * \cos (\text{Arg2pi } z) = \text{Re } z$ **and** sin_Arg2pi : $\text{cmod } z * \sin (\text{Arg2pi } z) = \text{Im } z$
using Arg2pi_eq [of z] cis_conv_exp Re_rcis Im_rcis **unfolding** rcis_def **by** metis+

lemma Arg2pi_minus :
assumes $z \neq 0$ **shows** $\text{Arg2pi } (-z) = (\text{if } \text{Arg2pi } z < \pi \text{ then } \text{Arg2pi } z + \pi \text{ else } \text{Arg2pi } z - \pi)$
apply ($\text{rule } \text{Arg2pi_unique}$ [of $\text{norm } z$, OF complex_eqI])
using cos_Arg2pi sin_Arg2pi Arg2pi_ge_0 Arg2pi_lt_2pi [of z] assms
by ($\text{auto simp: Re_exp Im_exp}$)

lemma $\text{Arg2pi_times_of_real}$ [simp]:
assumes $0 < r$ **shows** $\text{Arg2pi } (\text{of_real } r * z) = \text{Arg2pi } z$
by ($\text{metis } (\text{no_types, lifting}) \text{Arg2pi Arg2pi_eq Arg2pi_unique assms mult.assoc}$
 $\text{mult_eq_0_iff mult_pos_pos of_real_mult zero_less_norm_iff}$)

lemma $\text{Arg2pi_times_of_real2}$ [simp]: $0 < r \implies \text{Arg2pi } (z * \text{of_real } r) = \text{Arg2pi } z$
by ($\text{metis Arg2pi_times_of_real mult.commute}$)

lemma $\text{Arg2pi_divide_of_real}$ [simp]: $0 < r \implies \text{Arg2pi } (z / \text{of_real } r) = \text{Arg2pi } z$
by ($\text{metis Arg2pi_times_of_real2 less_irrefl nonzero_eq_divide_eq of_real_eq_0_iff}$)

lemma Arg2pi_le_pi : $\text{Arg2pi } z \leq \pi \iff 0 \leq \text{Im } z$
proof ($\text{cases } z=0$)
case False
have $0 \leq \text{Im } z \iff 0 \leq \text{Im } (\text{of_real } (\text{cmod } z) * \exp (i * \text{complex_of_real } (\text{Arg2pi } z)))$
by (metis Arg2pi_eq)
also have $\dots = (0 \leq \text{Im } (\exp (i * \text{complex_of_real } (\text{Arg2pi } z))))$
using False **by** ($\text{simp add: zero_le_mult_iff}$)
also have $\dots \iff \text{Arg2pi } z \leq \pi$
by (simp add: Im_exp) ($\text{metis Arg2pi_ge_0 Arg2pi_lt_2pi sin_lt_zero sin_ge_zero not_le}$)
finally show $?thesis$
by blast
qed auto

lemma Arg2pi_lt_pi : $0 < \text{Arg2pi } z \wedge \text{Arg2pi } z < \pi \iff 0 < \text{Im } z$
using Arg2pi_le_pi [of z]
by ($\text{smt } (\text{verit, del_insts}) \text{Arg2pi_0 Arg2pi_le_pi Arg2pi_minus uminus_complex.simps(2)} \text{zero_complex.simps(2)}$)

lemma Arg2pi_eq_0 : $\text{Arg2pi } z = 0 \iff z \in \mathbf{R} \wedge 0 \leq \text{Re } z$

1962

proof (cases z=0)
 case False
 then have $z \in \mathbb{R} \wedge 0 \leq \text{Re } z \iff z \in \mathbb{R} \wedge 0 \leq \text{Re } (\exp (i * \text{complex_of_real } (\text{Arg2pi } z)))$
 by (metis cis.sel(1) cis_conv_exp cos_Arg2pi norm_ge_zero norm_le_zero_iff zero_le_mult_iff)
 also have $\dots \iff \text{Arg2pi } z = 0$
 proof -
 have [simp]: $\text{Arg2pi } z = 0 \implies z \in \mathbb{R}$
 using Arg2pi_eq [of z] **by** (auto simp: Reals_def)
 moreover have $\llbracket z \in \mathbb{R}; 0 \leq \cos (\text{Arg2pi } z) \rrbracket \implies \text{Arg2pi } z = 0$
 by (smt (verit, ccfv_SIG) Arg2pi_ge_0 Arg2pi_le_pi Arg2pi_lt_pi complex_is_Real_iff cos_pi)
 ultimately show ?thesis
 by (auto simp: Re_exp)
 qed
 finally show ?thesis
 by blast
qed auto

corollary Arg2pi_gt_0:
 assumes $z \notin \mathbb{R}_{\geq 0}$
 shows $\text{Arg2pi } z > 0$
 using Arg2pi_eq_0 Arg2pi_ge_0 assms dual_order.strict_iff_order
 unfolding nonneg_Reals_def **by** fastforce

lemma Arg2pi_eq_pi: $\text{Arg2pi } z = \pi \iff z \in \mathbb{R} \wedge \text{Re } z < 0$
 using Arg2pi_le_pi [of z] Arg2pi_lt_pi [of z] Arg2pi_eq_0 [of z] Arg2pi_ge_0 [of z]
 by (fastforce simp: complex_is_Real_iff)

lemma Arg2pi_eq_0_pi: $\text{Arg2pi } z = 0 \vee \text{Arg2pi } z = \pi \iff z \in \mathbb{R}$
 using Arg2pi_eq_0 Arg2pi_eq_pi not_le **by** auto

lemma Arg2pi_of_real: $\text{Arg2pi } (\text{of_real } r) = (\text{if } r < 0 \text{ then } \pi \text{ else } 0)$
 using Arg2pi_eq_0_pi Arg2pi_eq_pi **by** fastforce

lemma Arg2pi_real: $z \in \mathbb{R} \implies \text{Arg2pi } z = (\text{if } 0 \leq \text{Re } z \text{ then } 0 \text{ else } \pi)$
 using Arg2pi_eq_0 Arg2pi_eq_0_pi **by** auto

lemma Arg2pi_inverse: $\text{Arg2pi}(\text{inverse } z) = (\text{if } z \in \mathbb{R} \text{ then } \text{Arg2pi } z \text{ else } 2*\pi - \text{Arg2pi } z)$

proof (cases z=0)
 case False
 show ?thesis
 apply (rule Arg2pi_unique [of inverse (norm z)])
 using Arg2pi_eq False Arg2pi_ge_0 [of z] Arg2pi_lt_2pi [of z] Arg2pi_eq_0 [of z]
 by (auto simp: Arg2pi_real in_Reals_norm exp_diff field_simps)

qed auto

lemma Arg2pi_eq_iff:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Arg2pi } w = \text{Arg2pi } z \iff (\exists x. 0 < x \wedge w = \text{of_real } x * z)$ (is ?lhs = ?rhs)

proof

assume ?lhs

then have $(\text{cmod } w) * (z / \text{cmod } z) = w$

by (metis Arg2pi_eq assms(2) mult_eq_0_iff nonzero_mult_div_cancel_left)

then show ?rhs

by (metis assms divide_pos_pos of_real_divide times_divide_eq_left times_divide_eq_right zero_less_norm_iff)

qed auto

lemma Arg2pi_inverse_eq_0: $\text{Arg2pi}(\text{inverse } z) = 0 \iff \text{Arg2pi } z = 0$

by (metis Arg2pi_eq_0 Arg2pi_inverse inverse_inverse_eq)

lemma Arg2pi_divide:

assumes $w \neq 0$ $z \neq 0$ $\text{Arg2pi } w \leq \text{Arg2pi } z$

shows $\text{Arg2pi}(z / w) = \text{Arg2pi } z - \text{Arg2pi } w$

apply (rule Arg2pi_unique [of norm(z / w)])

using assms Arg2pi_eq Arg2pi_ge_0 [of w] Arg2pi_lt_2pi [of z]

apply (auto simp: exp_diff norm_divide field_simps)

done

lemma Arg2pi_le_div_sum:

assumes $w \neq 0$ $z \neq 0$ $\text{Arg2pi } w \leq \text{Arg2pi } z$

shows $\text{Arg2pi } z = \text{Arg2pi } w + \text{Arg2pi}(z / w)$

by (simp add: Arg2pi_divide assms)

lemma Arg2pi_le_div_sum_eq:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Arg2pi } w \leq \text{Arg2pi } z \iff \text{Arg2pi } z = \text{Arg2pi } w + \text{Arg2pi}(z / w)$

using assms by (auto simp: Arg2pi_ge_0 intro: Arg2pi_le_div_sum)

lemma Arg2pi_diff:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Arg2pi } w - \text{Arg2pi } z = (\text{if } \text{Arg2pi } z \leq \text{Arg2pi } w \text{ then } \text{Arg2pi}(w / z) \text{ else } \text{Arg2pi}(w/z) - 2*\pi)$

using assms Arg2pi_divide Arg2pi_inverse [of w/z] Arg2pi_eq_0_pi

by (force simp add: Arg2pi_ge_0 Arg2pi_divide not_le split: if_split_asm)

lemma Arg2pi_add:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Arg2pi } w + \text{Arg2pi } z = (\text{if } \text{Arg2pi } w + \text{Arg2pi } z < 2*\pi \text{ then } \text{Arg2pi}(w * z) \text{ else } \text{Arg2pi}(w * z) + 2*\pi)$

using assms Arg2pi_diff [of w*z z] Arg2pi_le_div_sum_eq [of z w*z] Arg2pi [of w * z]

1964

by *auto*

lemma *Arg2pi_times*:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Arg2pi}(w * z) = (\text{if } \text{Arg2pi } w + \text{Arg2pi } z < 2 * \pi \text{ then } \text{Arg2pi } w + \text{Arg2pi } z$

$\text{else } (\text{Arg2pi } w + \text{Arg2pi } z) - 2 * \pi)$

using *Arg2pi_add* [*OF assms*] **by** *auto*

lemma *Arg2pi_cnj_eq_inverse*:

assumes $z \neq 0$ **shows** $\text{Arg2pi}(\text{cnj } z) = \text{Arg2pi}(\text{inverse } z)$

proof –

have $\exists r > 0. \text{of_real } r / z = \text{cnj } z$

by (*metis assms complex_norm_square nonzero_mult_div_cancel_left zero_less_norm_iff zero_less_power*)

then show *?thesis*

by (*metis Arg2pi_times_of_real2 divide_inverse_commute*)

qed

lemma *Arg2pi_cnj*: $\text{Arg2pi}(\text{cnj } z) = (\text{if } z \in \mathbb{R} \text{ then } \text{Arg2pi } z \text{ else } 2 * \pi - \text{Arg2pi } z)$

by (*metis Arg2pi_cnj_eq_inverse Arg2pi_inverse Reals_cnj_iff complex_cnj_zero*)

lemma *Arg2pi_exp*: $0 \leq \text{Im } z \implies \text{Im } z < 2 * \pi \implies \text{Arg2pi}(\text{exp } z) = \text{Im } z$

by (*simp add: Arg2pi_unique exp_eq_polar*)

lemma *complex_split_polar*:

obtains $r a :: \text{real}$ **where** $z = \text{complex_of_real } r * (\cos a + i * \sin a)$ $0 \leq r$ $0 \leq a < 2 * \pi$

using *Arg2pi_cis.ctr cis_conv_exp unfolding Complex_eq_is_Arg_def* **by** *fastforce*

lemma *Re_Im_le_cmod*: $\text{Im } w * \sin \varphi + \text{Re } w * \cos \varphi \leq \text{cmod } w$

proof (*cases w rule: complex_split_polar*)

case ($1 r a$) **with** *sin_cos_le1* [*of a* φ] **show** *?thesis*

apply (*simp add: norm_mult cmod_unit_one*)

by (*metis (no_types, opaque_lifting) abs_le_D1 distrib_left mult.commute mult.left_commute mult_left_le*)

qed

6.20.8 Analytic properties of tangent function

lemma *cnj_tan*: $\text{cnj}(\tan z) = \tan(\text{cnj } z)$

by (*simp add: cnj_cos cnj_sin tan_def*)

lemma *field_differentiable_at_tan*: $\cos z \neq 0 \implies \tan \text{ field_differentiable at } z$

unfolding *field_differentiable_def*

using *DERIV_tan* **by** *blast*


```

lemma field_differentiable_within_tan:  $\cos z \neq 0$ 
   $\implies \tan$  field_differentiable (at  $z$  within  $s$ )
  using field_differentiable_at_tan field_differentiable_at_within by blast

lemma continuous_within_tan:  $\cos z \neq 0 \implies \text{continuous}$  (at  $z$  within  $s$ ) tan
  using continuous_at_imp_continuous_within isCont_tan by blast

lemma continuous_on_tan [continuous_intros]:  $(\bigwedge z. z \in s \implies \cos z \neq 0) \implies$ 
continuous_on  $s$  tan
  by (simp add: continuous_at_imp_continuous_on)

lemma holomorphic_on_tan:  $(\bigwedge z. z \in s \implies \cos z \neq 0) \implies \tan$  holomorphic_on
 $s$ 
  by (simp add: field_differentiable_within_tan holomorphic_on_def)

```

6.20.9 The principal branch of the Complex logarithm

```

instantiation complex :: ln
begin

```

```

definition ln_complex :: complex  $\Rightarrow$  complex
  where ln_complex  $\equiv \lambda z. \text{THE } w. \exp w = z \ \& \ -\pi < \text{Im}(w) \ \& \ \text{Im}(w) \leq \pi$ 

```

NOTE: within this scope, the constant Ln is not yet available!

```

lemma
  assumes  $z \neq 0$ 
  shows exp_Ln [simp]:  $\exp(\ln z) = z$ 
  and mpi_less_Im_Ln:  $-\pi < \text{Im}(\ln z)$ 
  and Im_Ln_le_pi:  $\text{Im}(\ln z) \leq \pi$ 
proof -
  obtain  $\psi$  where  $z / (\text{cmod } z) = \text{Complex } (\cos \psi) (\sin \psi)$ 
  using complex_unimodular_polar [of  $z / (\text{norm } z)$ ] assms
  by (auto simp: norm_divide field_split_simps)
  obtain  $\varphi$  where  $-\pi < \varphi \leq \pi \ \sin \varphi = \sin \psi \ \cos \varphi = \cos \psi$ 
  using sincos_principal_value [of  $\psi$ ] assms
  by (auto simp: norm_divide field_split_simps)
  have  $\exp(\ln z) = z \ \& \ -\pi < \text{Im}(\ln z) \ \& \ \text{Im}(\ln z) \leq \pi$  unfolding ln_complex_def
  apply (rule theI [where  $a = \text{Complex } (\ln(\text{norm } z)) \ \varphi$ ])
  using  $z$  assms  $\varphi$ 
  apply (auto simp: field_simps exp_complex_eqI exp_eq_polar cis.code)
  done
  then show  $\exp(\ln z) = z \ -\pi < \text{Im}(\ln z) \ \text{Im}(\ln z) \leq \pi$ 
  by auto
qed

```

```

lemma Ln_exp [simp]:
  assumes  $-\pi < \text{Im}(z) \ \text{Im}(z) \leq \pi$ 
  shows  $\ln(\exp z) = z$ 
proof (rule exp_complex_eqI)

```

1966

show $|Im (ln (exp z)) - Im z| < 2 * pi$
using *assms mpi_less_Im_Ln [of exp z] Im_Ln_le_pi [of exp z]* **by** *auto*
qed *auto*

6.20.10 Relation to Real Logarithm

lemma *Ln_of_real*:

assumes $0 < z$

shows $ln(of_real\ z::complex) = of_real(ln\ z)$

by (*smt (verit) Im_complex_of_real Ln_exp assms exp_ln of_real_exp pi_ge_two*)

corollary *Ln_in_Reals [simp]*: $z \in \mathbf{R} \implies Re\ z > 0 \implies ln\ z \in \mathbf{R}$

by (*auto simp: Ln_of_real elim: Reals_cases*)

corollary *Im_Ln_of_real [simp]*: $r > 0 \implies Im (ln (of_real\ r)) = 0$

by (*simp add: Ln_of_real*)

lemma *cmod_Ln_Reals [simp]*: $z \in \mathbf{R} \implies 0 < Re\ z \implies cmod (ln\ z) = norm (ln (Re\ z))$

using *Ln_of_real* **by** *force*

lemma *Ln_Reals_eq*: $[x \in \mathbf{R}; Re\ x > 0] \implies ln\ x = of_real (ln (Re\ x))$

using *Ln_of_real* **by** *force*

lemma *Ln_1 [simp]*: $ln\ 1 = (0::complex)$

by (*smt (verit, best) Ln_of_real ln_one of_real_0 of_real_1*)

lemma *Ln_eq_zero_iff [simp]*: $x \notin \mathbf{R}_{\leq 0} \implies ln\ x = 0 \iff x = 1$ **for** $x::complex$

by (*metis (mono_tags, lifting) Ln_1 exp_Ln exp_zero nonpos_Reals_zero_I*)

instance

by *intro_classes (rule ln_complex_def Ln_1)*

end

abbreviation *Ln :: complex \Rightarrow complex*

where $Ln \equiv ln$

lemma *Ln_eq_iff*: $w \neq 0 \implies z \neq 0 \implies (Ln\ w = Ln\ z \iff w = z)$

by (*metis exp_Ln*)

lemma *Ln_unique*: $exp(z) = w \implies -pi < Im(z) \implies Im(z) \leq pi \implies Ln\ w = z$

using *Ln_exp* **by** *blast*

lemma *Re_Ln [simp]*: $z \neq 0 \implies Re(Ln\ z) = ln(norm\ z)$

by (*metis exp_Ln ln_exp norm_exp_eq_Re*)

corollary *ln_cmod_le*:

assumes $z: z \neq 0$

shows $\ln (\text{cmod } z) \leq \text{cmod } (\text{Ln } z)$
by (*metis Re_Ln complex_Re_le_cmod z*)

proposition *exists_complex_root*:

fixes $z :: \text{complex}$
assumes $n \neq 0$ **obtains** w **where** $z = w \wedge n$
by (*metis assms exp_Ln exp_of_nat_mult nonzero_mult_div_cancel_left of_nat_eq_0_iff power_0_left times_divide_eq_right*)

corollary *exists_complex_root_nonzero*:

fixes $z :: \text{complex}$
assumes $z \neq 0$ $n \neq 0$
obtains w **where** $w \neq 0$ $z = w \wedge n$
by (*metis exists_complex_root [of n z] assms power_0_left*)

6.20.11 Derivative of Ln away from the branch cut

lemma *Im_Ln_less_pi*:

assumes $z \notin \mathbb{R}_{\leq 0}$ **shows** $\text{Im } (\text{Ln } z) < \pi$

proof –

have $z \neq 0$ [simp]: $z \neq 0$

using *assms* **by** *auto*

with *Im_Ln_le_pi [of z]* **show** *?thesis*

by (*smt (verit, best) Arg2pi_eq_0_pi Arg2pi_exp_Ln_in_Reals assms complex_is_Real_iff complex_nonpos_Reals_iff exp_Ln_pi_ge_two*)

qed

lemma *has_field_derivative_Ln*:

assumes $z \notin \mathbb{R}_{\leq 0}$

shows $(\text{Ln } \text{has_field_derivative } \text{inverse}(z))$ (at z)

proof –

have $z \neq 0$ [simp]: $z \neq 0$

using *assms* **by** *auto*

then have $\text{Im } (\text{Ln } z) \neq \pi$

by (*smt (verit, best) Arg2pi_eq_0_pi Arg2pi_exp_Ln_in_Reals assms complex_is_Real_iff complex_nonpos_Reals_iff exp_Ln_pi_ge_two*)

let $?U = \{w. -\pi < \text{Im}(w) \wedge \text{Im}(w) < \pi\}$

have 1: *open ?U*

by (*simp add: open_Collect_conj open_halfspace_Im_gt open_halfspace_Im_lt*)

have 2: $\bigwedge x. x \in ?U \implies (\text{exp } \text{has_derivative } \text{blinfun_apply}(\text{Blinfun } ((*)) (\text{exp } x)))$ (at x)

by (*simp add: bounded_linear_Blinfun_apply bounded_linear_mult_right has_field_derivative_imp_has_derivative*)

have 3: *continuous_on ?U* $(\lambda x. \text{Blinfun } ((*)) (\text{exp } x))$

unfolding *blinfun_mult_right.abs_eq [symmetric]* **by** (*intro continuous_intros*)

have 4: $\text{Ln } z \in ?U$

by (*simp add: Im_Ln_less_pi assms mpi_less_Im_Ln*)

have 5: $\text{Blinfun } ((*)) (\text{inverse } z) \circ_L \text{Blinfun } ((*)) (\text{exp } (\text{Ln } z)) = \text{id_blinfun}$

by (*rule blinfun_eqI*) (*simp add: bounded_linear_mult_right bounded_linear_Blinfun_apply*)

obtain $U' V g g'$ **where** *open U'* **and** *sub: U' \subseteq ?U*

```

and  $Ln\ z \in U'$  open  $V\ z \in V$ 
and hom: homeomorphism  $U' V$  exp  $g$ 
and  $g: \bigwedge y. y \in V \implies (g\ \text{has\_derivative}\ (g'\ y))\ (\text{at}\ y)$ 
and  $g': \bigwedge y. y \in V \implies g'\ y = \text{inv}\ ((*)\ (\text{exp}\ (g\ y)))$ 
and  $bij: \bigwedge y. y \in V \implies \text{bij}\ ((*)\ (\text{exp}\ (g\ y)))$ 
using inverse_function_theorem [OF 1 2 3 4 5]
by (simp add: bounded_linear_Blinfun_apply bounded_linear_mult_right)
blast
show ( $Ln\ \text{has\_field\_derivative}\ \text{inverse}(z))\ (\text{at}\ z)$ 
unfolding has_field_derivative_def
proof (rule has_derivative_transform_within_open)
show  $g\_eq\_Ln: g\ y = Ln\ y$  if  $y \in V$  for  $y$ 
by (smt (verit, ccfv_threshold)  $Ln\_exp\ \text{hom}\ \text{homeomorphism\_def}\ \text{imageI}$ 
mem_Collect_eq sub subset_iff that)
have  $0 \notin V$ 
by (meson exp_not_eq_zero hom homeomorphism_def)
then have  $\bigwedge y. y \in V \implies g'\ y = \text{inv}\ ((*)\ y)$ 
by (metis exp_Ln  $g'\ g\_eq\_Ln$ )
then have  $g': g'\ z = (\lambda x. x/z)$ 
by (metis  $\langle z \in V \rangle$  bij bij_inv_eq_iff_exp_Ln g_eq_Ln nonzero_mult_div_cancel_left
 $znz$ )
show ( $g\ \text{has\_derivative}\ (*)\ (\text{inverse}\ z))\ (\text{at}\ z)$ 
using  $g$  [OF  $\langle z \in V \rangle$ ]  $g'$  by (simp add: divide_inverse_commute)
qed (auto simp:  $\langle z \in V \rangle$   $\langle \text{open}\ V \rangle$ )
qed

declare has_field_derivative_Ln [derivative_intros]
declare has_field_derivative_Ln [THEN DERIV_chain2, derivative_intros]

lemma field_differentiable_at_Ln:  $z \notin \mathbf{R}_{\leq 0} \implies Ln\ \text{field\_differentiable}\ \text{at}\ z$ 
using field_differentiable_def has_field_derivative_Ln by blast

lemma field_differentiable_within_Ln:  $z \notin \mathbf{R}_{\leq 0}$ 
 $\implies Ln\ \text{field\_differentiable}\ (\text{at}\ z\ \text{within}\ S)$ 
using field_differentiable_at_Ln field_differentiable_within_subset by blast

lemma continuous_at_Ln:  $z \notin \mathbf{R}_{\leq 0} \implies \text{continuous}\ (\text{at}\ z)\ Ln$ 
by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_Ln)

lemma isCont_Ln' [simp, continuous_intros]:
 $\llbracket \text{isCont}\ f\ z; f\ z \notin \mathbf{R}_{\leq 0} \rrbracket \implies \text{isCont}\ (\lambda x. Ln\ (f\ x))\ z$ 
by (blast intro: isCont_o2 [OF continuous_at_Ln])

lemma continuous_within_Ln [continuous_intros]:  $z \notin \mathbf{R}_{\leq 0} \implies \text{continuous}\ (\text{at}\ z\ \text{within}\ S)\ Ln$ 
using continuous_at_Ln continuous_at_imp_continuous_within by blast

lemma continuous_on_Ln [continuous_intros]:  $(\bigwedge z. z \in S \implies z \notin \mathbf{R}_{\leq 0}) \implies$ 
 $\text{continuous\_on}\ S\ Ln$ 

```

by (simp add: continuous_at_imp_continuous_on continuous_within_Ln)

lemma continuous_on_Ln' [continuous_intros]:

continuous_on S f $\implies (\bigwedge z. z \in S \implies f z \notin \mathbb{R}_{\leq 0}) \implies \text{continuous_on } S (\lambda x. \text{Ln } (f x))$

by (rule continuous_on_compose2[OF continuous_on_Ln, of UNIV - non-pos_Reals S f]) auto

lemma holomorphic_on_Ln [holomorphic_intros]: $S \cap \mathbb{R}_{\leq 0} = \{\}$ $\implies \text{Ln holomorphic_on } S$

by (simp add: disjoint_iff_field_differentiable_within_Ln holomorphic_on_def)

lemma holomorphic_on_Ln' [holomorphic_intros]:

$(\bigwedge z. z \in A \implies f z \notin \mathbb{R}_{\leq 0}) \implies f \text{ holomorphic_on } A \implies (\lambda z. \text{Ln } (f z)) \text{ holomorphic_on } A$

using holomorphic_on_compose_gen[OF holomorphic_on_Ln, of f A - $\mathbb{R}_{\leq 0}$]
by (auto simp: o_def)

lemma analytic_on_Ln [analytic_intros]:

assumes f analytic_on A f' A $\cap \mathbb{R}_{\leq 0} = \{\}$

shows $(\lambda w. \text{Ln } (f w)) \text{ analytic_on } A$

proof -

have *: ln analytic_on $(-\mathbb{R}_{\leq 0})$

by (subst analytic_on_open) (auto intro!: holomorphic_intros)

have $(\text{Ln} \circ f) \text{ analytic_on } A$

by (rule analytic_on_compose_gen[OF assms(1) *]) (use assms(2) in auto)

thus ?thesis

by (simp add: o_def)

qed

lemma tendsto_Ln [tendsto_intros]:

assumes $(f \longrightarrow L) F L \notin \mathbb{R}_{\leq 0}$

shows $((\lambda x. \text{Ln } (f x)) \longrightarrow \text{Ln } L) F$

by (simp add: assms isCont_tendsto_compose)

lemma divide_Ln_mono:

fixes x y::real

assumes $0 < x \leq y$

shows $x / \text{Ln } x \leq y / \text{Ln } y$

proof -

have $\bigwedge u. [x \leq u; u \leq y] \implies ((\lambda z. z / \text{Ln } z) \text{ has_field_derivative } 1 / \text{Ln } u - 1 / (\text{Ln } u)^2) (\text{at } u)$

using $\langle 0 < x \rangle$ by (force intro!: derivative_eq_intros simp: field_simps power_eq_if)

moreover

have $x / \text{Ln } x \leq y / \text{Ln } y$

if $\text{Re } (y / \text{Ln } y) - \text{Re } (x / \text{Ln } x) = (\text{Re } (1 / \text{Ln } u) - \text{Re } (1 / (\text{Ln } u)^2)) * (y - x)$

and $x: x \leq u \leq y$ for u

proof -

1970

have $eq: y / \ln y = (1 / \ln u - 1 / (\ln u)^2) * (y - x) + x / \ln x$
using *that* $\langle 3 \leq x \rangle$ **by** (*auto simp: Ln_Reals_eq in_Reals_norm_group_add_class.diff_eq_eq*)
show *?thesis*
using *exp_le* $\langle 3 \leq x \rangle$ **by** (*simp add: eq*) (*simp add: power_eq_if divide_simps ln_ge_iff*)
qed
ultimately show *?thesis*
using *complex_mvt_line* [*of* $x\ y\ \lambda z. z / \ln z\ \lambda z. 1 / (\ln z) - 1 / (\ln z)^2$]
assms
by (*force simp add: closed_segment_Reals closed_segment_eq_real_ivl*)
qed

theorem *Ln_series:*

fixes $z :: \text{complex}$
assumes $\text{norm } z < 1$
shows $(\lambda n. (-1)^{\text{Suc } n} / \text{of_nat } n * z^{\wedge} n) \text{ sums } \ln(1 + z)$ (**is** $(\lambda n. ?f\ n * z^{\wedge} n) \text{ sums } _$)
proof $-$
let $?F = \lambda z. \sum n. ?f\ n * z^{\wedge} n$ **and** $?F' = \lambda z. \sum n. \text{diffs } ?f\ n * z^{\wedge} n$
have $r: \text{conv_radius } ?f = 1$
by (*intro conv_radius_ratio_limit_nonzero*[*of* $_ 1$])
(simp_all add: norm_divide LIMSEQ_Suc_n_over_n del: of_nat_Suc)

have $\exists c. \forall z \in \text{ball } 0\ 1. \ln(1 + z) - ?F\ z = c$

proof (*rule has_field_derivative_zero_constant*)

fix $z :: \text{complex}$ **assume** $z': z \in \text{ball } 0\ 1$

hence $z: \text{norm } z < 1$ **by** *simp*

define $t :: \text{complex}$ **where** $t = \text{of_real } (1 + \text{norm } z) / 2$

from z **have** $t: \text{norm } z < \text{norm } t$ $\text{norm } t < 1$ **unfolding** t_def

by (*simp_all add: field_simps norm_divide del: of_real_add*)

have $\text{Re } (-z) \leq \text{norm } (-z)$ **by** (*rule complex_Re_le_cmod*)

also from z **have** $\dots < 1$ **by** *simp*

finally have $((\lambda z. \ln(1 + z)) \text{ has_field_derivative } \text{inverse } (1 + z))$ (*at* z)

by (*auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff*)

moreover have $(?F \text{ has_field_derivative } ?F' z)$ (*at* z) **using** $t\ r$

by (*intro termdiffs_strong*[*of* $_ t$] *summable_in_conv_radius*) *simp_all*

ultimately have $((\lambda z. \ln(1 + z) - ?F\ z) \text{ has_field_derivative } (\text{inverse } (1 + z) - ?F' z))$

(*at* z *within* $\text{ball } 0\ 1$)

by (*intro derivative_intros*) (*simp_all add: at_within_open*[*OF* z'])

also have $(\lambda n. \text{of_nat } n * ?f\ n * z^{\wedge} (n - \text{Suc } 0)) \text{ sums } ?F' z$ **using** $t\ r$

by (*intro diffs_equiv termdiff_converges*[*OF* $t(1)$] *summable_in_conv_radius*) *simp_all*

from *sums_split_initial_segment*[*OF* *this*, *of* 1]

have $(\lambda i. (-z)^{\wedge} i) \text{ sums } ?F' z$ **by** (*simp add: power_minus*[*of* z] *del: of_nat_Suc*)

hence $?F' z = \text{inverse } (1 + z)$ **using** z **by** (*simp add: sums_iff sum-inf_geometric divide_inverse*)

also have $\text{inverse } (1 + z) - \text{inverse } (1 + z) = 0$ by *simp*
 finally show $((\lambda z. \ln (1 + z) - ?F z) \text{ has_field_derivative } 0)$ (at z within ball 0 1) .
 qed *simp_all*
 then obtain c where $c: \bigwedge z. z \in \text{ball } 0 \ 1 \implies \ln (1 + z) - ?F z = c$ by *blast*
 from $c[\text{of } 0]$ have $c = 0$ by (*simp only: powser_zero*) *simp*
 with $c[\text{of } z]$ *assms* have $\ln (1 + z) = ?F z$ by *simp*
 moreover have *summable* $(\lambda n. ?f n * z^n)$ using *assms r*
 by (*intro summable_in_conv_radius*) *simp_all*
 ultimately show *?thesis* by (*simp add: sums_iff*)
 qed

lemma *Ln_series'*: $\text{cmod } z < 1 \implies (\lambda n. -((-z)^n) / \text{of_nat } n) \text{ sums } \ln (1 + z)$
 by (*drule Ln_series*) (*simp add: power_minus'*)

lemma *ln_series'*:
 fixes $x::\text{real}$
 assumes $|x| < 1$
 shows $(\lambda n. -((-x)^n) / \text{of_nat } n) \text{ sums } \ln (1 + x)$
proof -
 from *assms* have $(\lambda n. -((-of_real x)^n) / \text{of_nat } n) \text{ sums } \ln (1 + \text{complex_of_real } x)$
 by (*intro Ln_series'*) *simp_all*
 also have $(\lambda n. -((-of_real x)^n) / \text{of_nat } n) = (\lambda n. \text{complex_of_real } (-((-x)^n) / \text{of_nat } n))$
 by (*rule ext*) *simp*
 also from *assms* have $\ln (1 + \text{complex_of_real } x) = \text{of_real } (\ln (1 + x))$
 by (*smt (verit) Ln_of_real of_real_1 of_real_add*)
 finally show *?thesis* by (*subst (asm) sums_of_real_iff*)
 qed

lemma *Ln_approx_linear*:
 fixes $z :: \text{complex}$
 assumes $\text{norm } z < 1$
 shows $\text{norm } (\ln (1 + z) - z) \leq \text{norm } z^2 / (1 - \text{norm } z)$
proof -
 let $?f = \lambda n. (-1)^{Suc n} / \text{of_nat } n$
 from *assms* have $(\lambda n. ?f n * z^n) \text{ sums } \ln (1 + z)$ using *Ln_series* by *simp*
 moreover have $(\lambda n. (\text{if } n = 1 \text{ then } 1 \text{ else } 0) * z^n) \text{ sums } z$ using *powser_sums_if*[*of 1*] by *simp*
 ultimately have $(\lambda n. (?f n - (\text{if } n = 1 \text{ then } 1 \text{ else } 0)) * z^n) \text{ sums } (\ln (1 + z) - z)$
 by (*subst left_diff_distrib, intro sums_diff*) *simp_all*
 from *sums_split_initial_segment*[*OF this, of Suc 1*]
 have $(\lambda i. (-z^2)) * \text{inverse } (2 + \text{of_nat } i) * (-z)^i \text{ sums } (\ln (1 + z) - z)$
 by (*simp add: power2_eq_square mult_ac power_minus[of z] divide_inverse*)
 hence $(\ln (1 + z) - z) = (\sum i. (-z^2)) * \text{inverse } (\text{of_nat } (i+2)) * (-z)^i$

```

    by (simp add: sums_iff)
    also have A: summable ( $\lambda n. \text{norm } z^2 * (\text{inverse } (\text{real\_of\_nat } (\text{Suc } (\text{Suc } n))))$ 
    *  $\text{cmod } z^n$ )
    by (rule summable_mult, rule summable_comparison_test_ev[OF summable_geometric[of
    norm z]])
      (auto simp: assms field_simps intro!: always_eventually)
    hence norm ( $\sum i. (-z^2) * \text{inverse } (\text{of\_nat } (i+2)) * (-z)^i$ )
       $\leq (\sum i. \text{norm } (-z^2) * \text{inverse } (\text{of\_nat } (i+2)) * (-z)^i)$ 
    by (intro summable_norm)
      (auto simp: norm_power norm_inverse norm_mult mult_ac simp del:
    of_nat_add of_nat_Suc)
    also have norm  $((-z)^2 * (-z)^i * \text{inverse } (\text{of\_nat } (i+2))) \leq \text{norm } ((-z)^2$ 
    *  $(-z)^i) * 1$  for i
    by (intro mult_left_mono) (simp_all add: field_split_simps)
    hence  $(\sum i. \text{norm } (-z^2) * \text{inverse } (\text{of\_nat } (i+2)) * (-z)^i)$ 
       $\leq (\sum i. \text{norm } (-z^2) * (-z)^i)$ 
    using A assms
    unfolding norm_power norm_inverse norm_divide norm_mult
    apply (intro suminf_le summable_mult summable_geometric)
    apply (auto simp: norm_power field_simps simp del: of_nat_add of_nat_Suc)
    done
    also have ... = norm  $z^2 * (\sum i. \text{norm } z^i)$  using assms
    by (subst suminf_mult [symmetric]) (auto intro!: summable_geometric simp:
    norm_mult norm_power)
    also have  $(\sum i. \text{norm } z^i) = \text{inverse } (1 - \text{norm } z)$  using assms
    by (subst suminf_geometric) (simp_all add: divide_inverse)
    also have norm  $z^2 * \dots = \text{norm } z^2 / (1 - \text{norm } z)$  by (simp add: di-
    vide_inverse)
    finally show ?thesis .
  qed

```

lemma norm_Ln_le:

fixes $z :: \text{complex}$

assumes $\text{norm } z < 1/2$

shows $\text{norm } (\text{Ln}(1+z)) \leq 2 * \text{norm } z$

proof -

have sums: $(\lambda n. -((-z)^n) / \text{of_nat } n)$ sums $\text{ln } (1 + z)$

by (intro Ln_series') (use assms in auto)

have summable: summable $(\lambda n. \text{norm } (-((-z)^n) / \text{of_nat } n))$

using ln_series'[of $-\text{norm } z$] assms

by (simp add: sums_iff summable_minus_iff norm_power norm_divide)

have norm $(\text{ln } (1 + z)) = \text{norm } (\sum n. -((-z)^n) / \text{of_nat } n)$

using sums by (simp add: sums_iff)

also have ... $\leq (\sum n. \text{norm } (-((-z)^n) / \text{of_nat } n))$

using summable by (rule summable_norm)

also have ... = $(\sum n. \text{norm } (-((-z)^{\text{Suc } n}) / \text{of_nat } (\text{Suc } n)))$

using summable by (subst suminf_split_head) auto


```

also have ... ≤ (∑ n. norm z * (1 / 2) ^ n)
proof (rule suminf_le)
  show summable (λn. norm z * (1 / 2) ^ n)
    by (intro summable_mult summable_geometric) auto
next
  show summable (λn. norm (- ((- z) ^ Suc n / of_nat (Suc n))))
    using summable by (subst summable_Suc_iff)
next
  fix n
  have norm (-((-z) ^ Suc n / of_nat (Suc n))) = norm z * (norm z ^ n /
real (Suc n))
    by (simp add: norm_power norm_divide norm_mult del: of_nat_Suc)
  also have ... ≤ norm z * ((1 / 2) ^ n / 1)
    using assms by (intro mult_left_mono frac_le power_mono) auto
  finally show norm (- ((- z) ^ Suc n / of_nat (Suc n))) ≤ norm z * (1 / 2)
^ n
    by simp
qed
also have ... = norm z * (∑ n. (1 / 2) ^ n)
  by (subst suminf_mult) (auto intro: summable_geometric)
also have (∑ n. (1 / 2 :: real) ^ n) = 2
  using geometric_sums[of 1 / 2 :: real] by (simp add: sums_iff)
finally show ?thesis
  by (simp add: mult_ac)
qed

```

6.20.12 Quadrant-type results for Ln

```

lemma cos_lt_zero_pi: pi/2 < x ⟹ x < 3*pi/2 ⟹ cos x < 0
  using cos_minus_pi cos_gt_zero_pi [of x-pi]
  by simp

```

```

lemma Re_Ln_pos_le:

```

```

  assumes z ≠ 0

```

```

  shows |Im(Ln z)| ≤ pi/2 ⟷ 0 ≤ Re(z)

```

```

proof -

```

```

  { fix w

```

```

    assume w = Ln z

```

```

    then have w: Im w ≤ pi - pi < Im w

```

```

      using Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms

```

```

      by auto

```

```

    then have |Im w| ≤ pi/2 ⟷ 0 ≤ Re(exp w)

```

```

      using cos_lt_zero_pi [of - (Im w)] cos_lt_zero_pi [of (Im w)] not_le

```

```

      by (auto simp: Re_exp zero_le_mult_iff abs_if intro: cos_ge_zero)

```

```

  }

```

```

  then show ?thesis using assms

```

```

    by auto

```

```

qed

```

1974

lemma *Re_Ln_pos_lt*:
assumes $z \neq 0$
shows $|Im(Ln\ z)| < pi/2 \iff 0 < Re(z)$
using *Re_Ln_pos_le assms*
by (*smt (verit) Re_exp arccos_cos_minus cos_pi_half exp_Ln exp_gt_zero field_sum_of_halves mult_eq_0_iff*)

lemma *Im_Ln_pos_le*:
assumes $z \neq 0$
shows $0 \leq Im(Ln\ z) \wedge Im(Ln\ z) \leq pi \iff 0 \leq Im(z)$
proof –
{ **fix** w
assume $w = Ln\ z$
then have $w: Im\ w \leq pi - pi < Im\ w$
using *Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms*
by auto
then have $0 \leq Im\ w \wedge Im\ w \leq pi \iff 0 \leq Im(exp\ w)$
using *sin_ge_zero [of - (Im w)] sin_ge_zero [of abs(Im w)] sin_zero_pi_iff [of Im w]*
by (*force simp: Im_exp_zero_le_mult_iff sin_ge_zero*) }
then show *?thesis using assms*
by auto
qed

lemma *Im_Ln_pos_lt*:
assumes $z \neq 0$
shows $0 < Im(Ln\ z) \wedge Im(Ln\ z) < pi \iff 0 < Im(z)$
using *Im_Ln_pos_le [OF assms] assms*
by (*smt (verit, best) Arg2pi_exp Arg2pi_lt_pi exp_Ln*)

lemma *Re_Ln_pos_lt_imp*: $0 < Re(z) \implies |Im(Ln\ z)| < pi/2$
by (*metis Re_Ln_pos_lt less_irrefl zero_complex.simps(1)*)

lemma *Im_Ln_pos_lt_imp*: $0 < Im(z) \implies 0 < Im(Ln\ z) \wedge Im(Ln\ z) < pi$
by (*metis Im_Ln_pos_lt not_le order_refl zero_complex.simps(2)*)

A reference to the set of positive real numbers

lemma *Im_Ln_eq_0*: $z \neq 0 \implies (Im(Ln\ z) = 0 \iff 0 < Re(z) \wedge Im(z) = 0)$
using *Im_Ln_pos_le Im_Ln_pos_lt Re_Ln_pos_lt by fastforce*

lemma *Im_Ln_eq_pi*: $z \neq 0 \implies (Im(Ln\ z) = pi \iff Re(z) < 0 \wedge Im(z) = 0)$
using *Im_Ln_eq_0 Im_Ln_pos_le Im_Ln_pos_lt complex.expand by fastforce*

6.20.13 More Properties of Ln

lemma *cnj_Ln*: **assumes** $z \notin \mathbb{R}_{\leq 0}$ **shows** $cnj(Ln\ z) = Ln(cnj\ z)$
proof (*cases z=0*)
case *False*
show *?thesis*

by (smt (verit) False Im_Ln_less_pi Ln_exp assms conj.sel(2) exp_Ln exp_cnj
mpi_less_Im_Ln)
qed (use assms in auto)

lemma Ln_inverse: assumes $z \notin \mathbb{R}_{\leq 0}$ shows $\text{Ln}(\text{inverse } z) = -(\text{Ln } z)$

proof (cases $z=0$)

case False

show ?thesis

by (smt (verit) False Im_Ln_less_pi Ln_exp assms exp_Ln exp_minus mpi_less_Im_Ln
uminus_complex.sel(2))

qed (use assms in auto)

lemma Ln_minus1 [simp]: $\text{Ln}(-1) = i * \pi$

proof (rule exp_complex_eqI)

show $|\text{Im}(\text{Ln}(-1)) - \text{Im}(i * \pi)| < 2 * \pi$

using Im_Ln_le_pi [of -1] mpi_less_Im_Ln [of -1] by auto

qed auto

lemma Ln_ii [simp]: $\text{Ln } i = i * \text{of_real } \pi/2$

using Ln_exp [of $i * (\text{of_real } \pi/2)$]

unfolding exp_Euler

by simp

lemma Ln_minus_ii [simp]: $\text{Ln}(-i) = -(i * \pi/2)$

using Ln_inverse by fastforce

lemma Ln_times:

assumes $w \neq 0$ $z \neq 0$

shows $\text{Ln}(w * z) =$

(if $\text{Im}(\text{Ln } w + \text{Ln } z) \leq -\pi$ then $(\text{Ln}(w) + \text{Ln}(z)) + i * \text{of_real}(2 * \pi)$

else if $\text{Im}(\text{Ln } w + \text{Ln } z) > \pi$ then $(\text{Ln}(w) + \text{Ln}(z)) - i * \text{of_real}(2 * \pi)$

else $\text{Ln}(w) + \text{Ln}(z)$)

using pi_ge_zero Im_Ln_le_pi [of w] Im_Ln_le_pi [of z]

using assms mpi_less_Im_Ln [of w] mpi_less_Im_Ln [of z]

by (auto simp: exp_add exp_diff sin_double cos_double exp_Euler intro!: Ln_unique)

corollary Ln_times_simple:

$\llbracket w \neq 0; z \neq 0; -\pi < \text{Im}(\text{Ln } w) + \text{Im}(\text{Ln } z); \text{Im}(\text{Ln } w) + \text{Im}(\text{Ln } z) \leq \pi \rrbracket$

$\implies \text{Ln}(w * z) = \text{Ln}(w) + \text{Ln}(z)$

by (simp add: Ln_times)

corollary Ln_times_of_real:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(\text{of_real } r * z) = \ln r + \text{Ln}(z)$

using mpi_less_Im_Ln Im_Ln_le_pi

by (force simp: Ln_times)

corollary Ln_times_of_nat:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(\text{of_nat } r * z :: \text{complex}) = \ln (\text{of_nat } r) + \text{Ln}(z)$

1976

using *Ln_times_of_real*[*of_of_nat r z*] by *simp*

corollary *Ln_times_Reals*:

$\llbracket r \in \text{Reals}; \text{Re } r > 0; z \neq 0 \rrbracket \implies \text{Ln}(r * z) = \text{ln } (\text{Re } r) + \text{Ln}(z)$

using *Ln_Reals_eq Ln_times_of_real* by *fastforce*

corollary *Ln_divide_of_real*:

$\llbracket r > 0; z \neq 0 \rrbracket \implies \text{Ln}(z / \text{of_real } r) = \text{Ln}(z) - \text{ln } r$

using *Ln_times_of_real* [*of_inverse r z*]

by (*simp add: ln_inverse Ln_of_real mult.commute divide_inverse flip: of_real_inverse*)

corollary *Ln_prod*:

fixes $f :: 'a \Rightarrow \text{complex}$

assumes $\text{finite } A \wedge x. x \in A \implies f x \neq 0$

shows $\exists n. \text{Ln}(\text{prod } f A) = (\sum x \in A. \text{Ln}(f x) + (\text{of_int } (n x) * (2 * \text{pi})) * i)$

using *assms*

proof (*induction A*)

case (*insert x A*)

then obtain n where $n: \text{Ln}(\text{prod } f A) = (\sum x \in A. \text{Ln}(f x) + \text{of_real}(\text{of_int}(n x) * (2 * \text{pi})) * i)$

by *auto*

define D where $D \equiv \text{Im}(\text{Ln}(f x)) + \text{Im}(\text{Ln}(\text{prod } f A))$

define $q::\text{int}$ where $q \equiv (\text{if } D \leq -\text{pi} \text{ then } 1 \text{ else if } D > \text{pi} \text{ then } -1 \text{ else } 0)$

have $\text{prod } f A \neq 0 \wedge f x \neq 0$

by (*auto simp: insert.hyps insert.prem*s)

with *insert.hyps pi_ge_zero* show *?case*

by (*rule_tac x=n(x:=q) in exI*) (*force simp: Ln_times q_def D_def n intro!*:

sum.cong)

qed *auto*

lemma *Ln_minus*:

assumes $z \neq 0$

shows $\text{Ln}(-z) = (\text{if } \text{Im}(z) \leq 0 \wedge \neg(\text{Re}(z) < 0 \wedge \text{Im}(z) = 0) \text{ then } \text{Ln}(z) + i * \text{pi} \text{ else } \text{Ln}(z) - i * \text{pi})$

using *Im_Ln_le_pi* [*of z*] *mpi_less_Im_Ln* [*of z*] *assms*

Im_Ln_eq_pi [*of z*] *Im_Ln_pos_lt* [*of z*]

by (*intro Ln_unique*) (*auto simp: exp_add exp_diff*)

lemma *Ln_inverse_if*:

assumes $z \neq 0$

shows $\text{Ln}(\text{inverse } z) = (\text{if } z \in \mathbb{R}_{\leq 0} \text{ then } -(\text{Ln } z) + i * 2 * \text{complex_of_real } \text{pi} \text{ else } -(\text{Ln } z))$

proof (*cases z ∈ ℝ_{≤0}*)

case *False* then show *?thesis*

by (*simp add: Ln_inverse*)

next

case *True*

then have $z: \text{Im } z = 0 \wedge \text{Re } z < 0 \wedge -z \notin \mathbb{R}_{\leq 0}$

using *assms complex_eq_iff complex_nonpos_Reals_iff* **by** *auto*
have $\text{Ln}(\text{inverse } z) = \text{Ln}(-(\text{inverse } (-z)))$
by *simp*
also have $\dots = \text{Ln}(\text{inverse } (-z)) + i * \text{complex_of_real } \pi$
using *assms z* **by** (*simp add: Ln_minus divide_less_0_iff*)
also have $\dots = -\text{Ln}(-z) + i * \text{complex_of_real } \pi$
using *z Ln_inverse* **by** *presburger*
also have $\dots = -(\text{Ln } z) + i * 2 * \text{complex_of_real } \pi$
using *Ln_minus assms z* **by** *auto*
finally show *?thesis* **by** (*simp add: True*)
qed

lemma *Ln_times_ii*:

assumes $z \neq 0$
shows $\text{Ln}(i * z) = (\text{if } 0 \leq \text{Re}(z) \mid \text{Im}(z) < 0$
 $\quad \text{then } \text{Ln}(z) + i * \text{of_real } \pi/2$
 $\quad \text{else } \text{Ln}(z) - i * \text{of_real}(3 * \pi/2))$
using *Im_Ln_le_pi [of z] mpi_less_Im_Ln [of z] assms*
 $\quad \text{Im_Ln_eq_pi [of z] Im_Ln_pos_lt [of z] Re_Ln_pos_le [of z]}$
by (*simp add: Ln_times*) *auto*

lemma *Ln_of_nat [simp]*: $0 < n \implies \text{Ln}(\text{of_nat } n) = \text{of_real}(\ln(\text{of_nat } n))$

by (*metis Ln_of_real_of_nat_0_less_iff_of_real_of_nat_eq*)

lemma *Ln_of_nat_over_of_nat*:

assumes $m > 0 \ n > 0$

shows $\text{Ln}(\text{of_nat } m / \text{of_nat } n) = \text{of_real}(\ln(\text{of_nat } m) - \ln(\text{of_nat } n))$

proof -

have $\text{of_nat } m / \text{of_nat } n = (\text{of_real}(\text{of_nat } m / \text{of_nat } n) :: \text{complex})$ **by**
simp

also from *assms* **have** $\text{Ln } \dots = \text{of_real}(\ln(\text{of_nat } m / \text{of_nat } n))$

by (*simp add: Ln_of_real[symmetric]*)

also from *assms* **have** $\dots = \text{of_real}(\ln(\text{of_nat } m) - \ln(\text{of_nat } n))$

by (*simp add: ln_div*)

finally show *?thesis* .

qed

lemma *norm_Ln_times_le*:

assumes $w \neq 0 \ z \neq 0$

shows $\text{cmod}(\text{Ln}(w * z)) \leq \text{cmod}(\text{Ln}(w) + \text{Ln}(z))$

proof (*cases - pi < Im(Ln w + Ln z) \wedge Im(Ln w + Ln z) \leq pi*)

case *True*

then show *?thesis*

by (*simp add: Ln_times_simple assms*)

next

case *False*

then show *?thesis*

by (*smt (verit) Im_Ln_le_pi assms cmod_Im_le_iff_exp_Ln_exp_add ln_unique*
 $\text{mpi_less_Im_Ln_mult_eq_0_iff_norm_exp_eq_Re}$)

1978

qed

corollary *norm_Ln_prod_le*:

fixes $f :: 'a \Rightarrow \text{complex}$

assumes $\bigwedge x. x \in A \implies f x \neq 0$

shows $\text{cmod } (Ln (\text{prod } f A)) \leq (\sum x \in A. \text{cmod } (Ln (f x)))$

using *assms*

proof (*induction A rule: infinite_finite_induct*)

case (*insert x A*)

then show *?case*

by *simp (smt (verit) norm_Ln_times_le norm_triangle_ineq prod_zero_iff)*

qed *auto*

lemma *norm_Ln_exp_le*: $\text{norm } (Ln (\text{exp } z)) \leq \text{norm } z$

by (*smt (verit) Im_Ln_le_pi Ln_exp Re_Ln cmod_Im_le_iff exp_not_eq_zero ln_exp_mpi_less_Im_Ln norm_exp_eq_Re*)

6.20.14 Uniform convergence and products

lemma *uniformly_convergent_on_prod_aux*:

fixes $f :: \text{nat} \Rightarrow \text{complex} \Rightarrow \text{complex}$

assumes $\text{norm}_f: \bigwedge n x. x \in A \implies \text{norm } (f n x) < 1$

assumes *cont*: $\bigwedge n. \text{continuous_on } A (f n)$

assumes *conv*: *uniformly_convergent_on* $A (\lambda N x. \sum n < N. \text{ln } (1 + f n x))$

assumes A : *compact* A

shows *uniformly_convergent_on* $A (\lambda N x. \prod n < N. 1 + f n x)$

proof –

from *conv* **obtain** S **where** S : *uniform_limit* $A (\lambda N x. \sum n < N. \text{ln } (1 + f n x))$ S *sequentially*

by (*auto simp: uniformly_convergent_on_def*)

have *cont'*: *continuous_on* $A S$

proof (*rule uniform_limit_theorem[OF _ S]*)

have $f n x + 1 \notin \mathbb{R}_{\leq 0}$ **if** $x \in A$ **for** $n x$

proof

assume $f n x + 1 \in \mathbb{R}_{\leq 0}$

then obtain t **where** $t \leq 0$ $f n x = \text{of_real } (t - 1)$

by (*metis add_diff_cancel nonpos_Reals_cases of_real_1 of_real_diff*)

moreover have $\text{norm } \dots \geq 1$

using t **by** (*subst norm_of_real*) *auto*

ultimately show *False*

using norm_f [*of x n*] **that** *by* *auto*

qed

thus $\forall_F n$ *in sequentially. continuous_on* $A (\lambda x. \sum n < n. Ln (1 + f n x))$

by (*auto intro!: always_eventually_continuous_intros cont simp: add_ac*)

qed *auto*

define B **where** $B = \{x + y \mid x y. x \in S \wedge y \in \text{cball } 0 \ 1\}$

have *compact* B

unfolding B_def **by** (*intro compact_sums compact_continuous_image cont'*)

A) *auto*

```

have uniformly_convergent_on A ( $\lambda N x. \exp ((\sum n < N. \ln (1 + f n x)))$ )
  using conv
proof (rule uniformly_convergent_on_compose_uniformly_continuous_on)
  show closed B
    using <compact B> by (auto dest: compact_imp_closed)
  show uniformly_continuous_on B exp
    by (intro compact_uniformly_continuous_continuous_intros <compact B>)

  have eventually ( $\lambda N. \forall x \in A. \text{dist } (\sum n < N. \text{Ln } (1 + f n x)) (S x) < 1$ )
    sequentially
    using S unfolding uniform_limit_iff by simp
  thus eventually ( $\lambda N. \forall x \in A. (\sum n < N. \text{Ln } (1 + f n x)) \in B$ ) sequentially
  proof eventually_elim
    case (elim N)
    show  $\forall x \in A. (\sum n < N. \text{Ln } (1 + f n x)) \in B$ 
    proof safe
      fix x assume x: x  $\in A$ 
      have  $(\sum n < N. \text{Ln } (1 + f n x)) = S x + ((\sum n < N. \text{Ln } (1 + f n x)) - S x)$ 
        by auto
      moreover have  $((\sum n < N. \text{Ln } (1 + f n x)) - S x) \in \text{ball } 0 1$ 
        using elim x by (auto simp: dist_norm norm_minus_commute)
      ultimately show  $(\sum n < N. \text{Ln } (1 + f n x)) \in B$ 
        unfolding B_def using x by fastforce
    qed
  qed
qed
qed
also have ?this  $\longleftrightarrow$  uniformly_convergent_on A ( $\lambda N x. \prod n < N. 1 + f n x$ )
proof (intro uniformly_convergent_cong_refl_always_eventually_allI_ballI)
  fix N :: nat and x assume x: x  $\in A$ 
  have  $\exp (\sum n < N. \ln (1 + f n x)) = (\prod n < N. \exp (\ln (1 + f n x)))$ 
    by (simp add: exp_sum)
  also have ... =  $(\prod n < N. 1 + f n x)$ 
    using norm_f[of x] x
  by (smt (verit, best) add.right_neutral add_diff_cancel exp_Ln norm_minus_commute
    norm_one prod.cong)
  finally show  $\exp (\sum n < N. \ln (1 + f n x)) = (\prod n < N. 1 + f n x)$  .
qed
finally show ?thesis .
qed

```

Theorem 17.6 by Bak and Newman, Complex Analysis [roughly]

```

lemma uniformly_convergent_on_prod:
  fixes f :: nat  $\Rightarrow$  complex  $\Rightarrow$  complex
  assumes cont:  $\bigwedge n. \text{continuous\_on } A (f n)$ 
  assumes A: compact A
  assumes conv_sum: uniformly_convergent_on A ( $\lambda N x. \sum n < N. \text{norm } (f n x)$ )
  shows uniformly_convergent_on A ( $\lambda N x. \prod n < N. 1 + f n x$ )

```

proof –

obtain M **where** $M: \bigwedge n x. n \geq M \implies x \in A \implies \text{norm } (f n x) < 1 / 2$

proof –

from conv_sum **have** $\text{uniformly_Cauchy_on } A (\lambda N x. \sum n < N. \text{norm } (f n x))$

using $\text{uniformly_convergent_Cauchy}$ **by** blast

moreover **have** $(1 / 2 :: \text{real}) > 0$

by simp

ultimately obtain M **where** $M:$

$\bigwedge x m n. x \in A \implies m \geq M \implies n \geq M \implies \text{dist } (\sum k < m. \text{norm } (f k x)) < 1 / 2$

unfolding $\text{uniformly_Cauchy_on_def}$ **by** fast

show $?thesis$

proof ($\text{rule that[of } M]$)

fix $n x$ **assume** $nx: n \geq M x \in A$

have $\text{dist } (\sum k < \text{Suc } n. \text{norm } (f k x)) (\sum k < n. \text{norm } (f k x)) < 1 / 2$

by ($\text{rule } M$) ($\text{use } nx$ **in** auto)

also have $\text{dist } (\sum k < \text{Suc } n. \text{norm } (f k x)) (\sum k < n. \text{norm } (f k x)) = \text{norm } (f n x)$

by ($\text{simp add: dist_norm}$)

finally show $\text{norm } (f n x) < 1 / 2$.

qed

qed

have $\text{conv: uniformly_convergent_on } A (\lambda N x. \sum n < N. \ln (1 + f (n + M) x))$

proof ($\text{rule uniformly_summable_comparison_test}$)

show $\text{norm } (\ln (1 + f (n + M) x)) \leq 2 * \text{norm } (f (n + M) x)$ **if** $x \in A$ **for** $n x$

by (rule norm_Ln_le) ($\text{use } M[\text{of } n + M x]$ **that** **in** auto)

have $*$: $\text{filterlim } (\lambda n. n + M)$ at_top at_top

by ($\text{rule filterlim_add_const_nat_at_top}$)

have $\text{uniformly_convergent_on } A (\lambda N x. 2 * ((\sum n < N + M. \text{norm } (f n x)) - (\sum n < M. \text{norm } (f n x))))$

by ($\text{intro uniformly_convergent_mult uniformly_convergent_minus}$

$\text{uniformly_convergent_on_compose[OF conv_sum *] uniformly_convergent_on_const}$)

also have $(\lambda N x. 2 * ((\sum n < N + M. \text{norm } (f n x)) - (\sum n < M. \text{norm } (f n x)))) =$

$(\lambda N x. \sum n < N. 2 * \text{norm } (f (n + M) x))$ (**is** $?lhs = ?rhs$)

proof (intro ext)

fix $N x$

have $(\sum n < N + M. \text{norm } (f n x)) - (\sum n < M. \text{norm } (f n x)) = (\sum n \in \{.. < N + M\} - \{.. < M\}. \text{norm } (f n x))$

by (subst sum_diff) auto

also have $\dots = (\sum n < N. \text{norm } (f (n + M) x))$

by ($\text{intro sum.reindex_bij_witness[of } _ \lambda n. n + M \lambda n. n - M]$) auto

finally show $?lhs N x = ?rhs N x$

by ($\text{simp add: sum_distrib_left}$)

qed

finally show $\text{uniformly_convergent_on } A (\lambda N x. \sum n < N. 2 * \text{cmod } (f (n + M) x))$.

qed

have *conv'*: *uniformly_convergent_on* A $(\lambda N x. \prod_{n < N}. 1 + f (n + M) x)$
proof (rule *uniformly_convergent_on_prod_aux*)
show *norm* $(f (n + M) x) < 1$ **if** $x \in A$ **for** $n x$
using M [of $n + M x$] **that by** *simp*
qed (use M *assms conv in auto*)

then obtain S **where** S : *uniform_limit* A $(\lambda N x. \prod_{n < N}. 1 + f (n + M) x)$
 S *sequentially*

by (*auto simp: uniformly_convergent_on_def*)

have *cont'*: *continuous_on* $A S$

by (*intro uniform_limit_theorem[OF _ S] always_eventually_ballI allI continuous_intros cont*) *auto*

have *uniform_limit* A $(\lambda N x. (\prod_{n < M}. 1 + f n x) * (\prod_{n < N}. 1 + f (n + M) x))$ $(\lambda x. (\prod_{n < M}. 1 + f n x) * S x)$ *sequentially*

proof (rule *uniform_lim_mult[OF uniform_limit_const S]*)

show *bounded* $(S ' A)$

by (*intro compact_imp_bounded compact_continuous_image A cont'*)

show *bounded* $((\lambda x. \prod_{n < M}. 1 + f n x) ' A)$

by (*intro compact_imp_bounded compact_continuous_image A continuous_intros cont*)

qed

hence *uniformly_convergent_on* A $(\lambda N x. (\prod_{n < M}. 1 + f n x) * (\prod_{n < N}. 1 + f (n + M) x))$

by (*auto simp: uniformly_convergent_on_def*)

also have $(\lambda N x. (\prod_{n < M}. 1 + f n x) * (\prod_{n < N}. 1 + f (n + M) x)) = (\lambda N x. \prod_{n < M+N}. 1 + f n x)$

proof (*intro ext*)

fix $N :: nat$ **and** $x :: complex$

have $(\prod_{n < N}. 1 + f (n + M) x) = (\prod_{n \in \{M..<M+N\}}. 1 + f n x)$

by (*intro prod_reindex_bij_witness[of _ $\lambda n. n - M$ $\lambda n. n + M]$) *auto**

also have $(\prod_{n < M}. 1 + f n x) * \dots = (\prod_{n \in \{..<M\} \cup \{M..<M+N\}}. 1 + f n x)$

by (*subst prod_union_disjoint*) *auto*

also have $\{..<M\} \cup \{M..<M+N\} = \{..<M+N\}$

by *auto*

finally show $(\prod_{n < M}. 1 + f n x) * (\prod_{n < N}. 1 + f (n + M) x) = (\prod_{n < M+N}. 1 + f n x)$.

qed

finally have *uniformly_convergent_on* A $(\lambda N x. \prod_{n < M + N}. 1 + f n x)$

by *simp*

hence *uniformly_convergent_on* A $(\lambda N x. \prod_{n < M + (N - M)}. 1 + f n x)$

by (rule *uniformly_convergent_on_compose*) (rule *filterlim_minus_const_nat_at_top*)

also have *?this* \longleftrightarrow *?thesis*

proof (rule *uniformly_convergent_cong*)

show *eventually* $(\lambda x. \forall y \in A. (\prod_{n < M + (x - M)}. 1 + f n y) = (\prod_{n < x}. 1 + f n y))$ *at_top*

1982

```

    using eventually_ge_at_top[of M] by eventually_elim auto
  qed auto
  finally show ?thesis .
qed

```

```

lemma uniformly_convergent_on_prod':
  fixes f :: nat  $\Rightarrow$  complex  $\Rightarrow$  complex
  assumes cont:  $\bigwedge n. \text{continuous\_on } A (f n)$ 
  assumes A: compact A
  assumes conv_sum: uniformly_convergent_on A ( $\lambda N x. \sum n < N. \text{norm } (f n x - 1)$ )
  shows uniformly_convergent_on A ( $\lambda N x. \prod n < N. f n x$ )
proof -
  have uniformly_convergent_on A ( $\lambda N x. \prod n < N. 1 + (f n x - 1)$ )
    by (rule uniformly_convergent_on_prod) (use assms in <auto intros>)
  thus ?thesis
    by simp
qed

```

Prop 17.6 of Bak and Newman, Complex Analysis, p. 243. Only this version is for the reals. Can the two proofs be consolidated?

```

lemma uniformly_convergent_on_prod_real:
  fixes u :: nat  $\Rightarrow$  real  $\Rightarrow$  real
  assumes contu:  $\bigwedge k. \text{continuous\_on } K (u k)$ 
  and uconv: uniformly_convergent_on K ( $\lambda n x. \sum k < n. |u k x|$ )
  and K: compact K
  shows uniformly_convergent_on K ( $\lambda n x. \prod k < n. 1 + u k x$ )
proof -
  define f where f  $\equiv \lambda k. \text{complex\_of\_real } \circ u k \circ \text{Re}$ 
  define L where L  $\equiv \text{complex\_of\_real } \text{' } K$ 
  have compact L
    by (simp add: <compact K> L_def compact_continuous_image)
  have Re ' complex_of_real ' X = X for X
    by (auto simp: image_iff)
  with contu have contf:  $\bigwedge k. \text{continuous\_on } L (f k)$ 
  unfolding f_def L_def by (intro continuous_intros) auto
  obtain S where S:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \forall_F n \text{ in sequentially. } \forall x \in K. \text{dist } (\sum k < n. |u k x|) (S x) < \varepsilon$ 
  using uconv unfolding uniformly_convergent_on_def uniform_limit_iff by presburger
  have  $\forall_F n \text{ in sequentially. } \forall z \in L. \text{dist } (\sum k < n. \text{cmod } (f k z)) ((\text{of\_real } \circ S \circ \text{Re}) z) < \varepsilon$ 
    if  $\varepsilon > 0$  for  $\varepsilon$ 
    using S [OF that] by eventually_elim (simp add: L_def f_def)
  then have uconvf: uniformly_convergent_on L ( $\lambda n z. \sum k < n. \text{norm } (f k z)$ )
    unfolding uniformly_convergent_on_def uniform_limit_iff by blast
  obtain P where P:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \forall_F n \text{ in sequentially. } \forall z \in L. \text{dist } (\prod k < n. 1 + f k z) (P z) < \varepsilon$ 

```

```

  using uniformly_convergent_on_prod [OF contf ‹compact L› uconvf]
  unfolding uniformly_convergent_on_def uniform_limit_iff by blast
  have §:  $|\left(\prod_{k < n. 1 + u\ k\ x}\right) - \text{Re}(P\ x)| \leq \text{cmod}\left(\left(\prod_{k < n. 1 + \text{of\_real}(u\ k\ x)}\right) - P\ x\right)$  for  $n\ x$ 
  proof -
    have  $\left(\prod_{k \in N. \text{of\_real}(1 + u\ k\ x)}\right) = \left(\prod_{k \in N. 1 + \text{of\_real}(u\ k\ x)}\right)$  for  $N$ 
      by force
    then show ?thesis
      by (metis Re_complex_of_real abs_Re_le_cmod minus_complex.sel(1) of_real_prod)
  qed
  have  $\forall_F\ n\ \text{in}\ \text{sequentially. } \forall x \in K. \text{dist}\left(\prod_{k < n. 1 + u\ k\ x}\right)\left(\text{Re} \circ P \circ \text{of\_real}\right)\ x < \varepsilon$ 
    if  $\varepsilon > 0$  for  $\varepsilon$ 
    using P [OF that] by eventually_elim (simp add: L_def f_def dist_norm le_less_trans [OF §])
  then show ?thesis
    unfolding uniformly_convergent_on_def uniform_limit_iff by blast
  qed

```

6.20.15 The Argument of a Complex Number

Unlike in HOL Light, it's defined for the same interval as the complex logarithm: $(-\pi, \pi]$.

lemma *Arg_eq_Im_Ln*:

assumes $z \neq 0$ **shows** $\text{Arg}\ z = \text{Im}\ (\text{Ln}\ z)$

proof (rule *cis_Arg_unique*)

show $\text{sgn}\ z = \text{cis}\ (\text{Im}\ (\text{Ln}\ z))$

by (metis *assms exp_Ln exp_eq_polar nonzero_mult_div_cancel_left norm_eq_zero norm_exp_eq_Re of_real_eq_0_iff sgn_eq*)

show $-\pi < \text{Im}\ (\text{Ln}\ z)$

by (simp add: *assms mpi_less_Im_Ln*)

show $\text{Im}\ (\text{Ln}\ z) \leq \pi$

by (simp add: *Im_Ln_le_pi assms*)

qed

The 1990s definition of argument coincides with the more recent one

lemma *Arg_def*:

shows $\text{Arg}\ z = (\text{if}\ z = 0\ \text{then}\ 0\ \text{else}\ \text{Im}\ (\text{Ln}\ z))$

by (simp add: *Arg_eq_Im_Ln Arg_zero*)

lemma *Arg_of_real* [simp]: $\text{Arg}\ (\text{of_real}\ r) = (\text{if}\ r < 0\ \text{then}\ \pi\ \text{else}\ 0)$

by (simp add: *Im_Ln_eq_pi Arg_def*)

lemma *mpi_less_Arg*: $-\pi < \text{Arg}\ z$ **and** *Arg_le_pi*: $\text{Arg}\ z \leq \pi$

by (auto simp: *Arg_def mpi_less_Im_Ln Im_Ln_le_pi*)

lemma *Arg_eq*:

1984

assumes $z \neq 0$
shows $z = \text{of_real}(\text{norm } z) * \exp(i * \text{Arg } z)$
using *cis_conv_exp rcis_cmod_Arg rcis_def* **by force**

lemma *is_Arg_Arg*: $z \neq 0 \implies \text{is_Arg } z (\text{Arg } z)$
by (*simp add: Arg_eq is_Arg_def*)

lemma *Argument_exists*:

assumes $z \neq 0$ **and** $R: R = \{r - \pi < .. r + \pi\}$
obtains s **where** $\text{is_Arg } z s \ s \in R$

proof –

let $?rp = r - \text{Arg } z + \pi$

define k **where** $k \equiv \lfloor ?rp / (2 * \pi) \rfloor$

have $(\text{Arg } z + \text{of_int } k * (2 * \pi)) \in R$

using *floor_divide_lower [of 2*pi ?rp] floor_divide_upper [of 2*pi ?rp]*

by (*auto simp: k_def algebra_simps R*)

then show *?thesis*

using *Arg_eq <z ≠ 0> is_Arg_2pi_iff is_Arg_def that* **by blast**

qed

lemma *Argument_exists_unique*:

assumes $z \neq 0$ **and** $R: R = \{r - \pi < .. r + \pi\}$

obtains s **where** $\text{is_Arg } z s \ s \in R \ \wedge t. \llbracket \text{is_Arg } z t; t \in R \rrbracket \implies s = t$

proof –

obtain s **where** $s: \text{is_Arg } z s \ s \in R$

using *Argument_exists [OF assms]* .

moreover have $\wedge t. \llbracket \text{is_Arg } z t; t \in R \rrbracket \implies s = t$

using *assms s* **by** (*auto simp: is_Arg_eqI*)

ultimately show *thesis*

using *that* **by blast**

qed

lemma *Argument_Ex1*:

assumes $z \neq 0$ **and** $R: R = \{r - \pi < .. r + \pi\}$

shows $\exists! s. \text{is_Arg } z s \ \wedge s \in R$

using *Argument_exists_unique [OF assms]* **by metis**

lemma *Arg_divide*:

assumes $w \neq 0 \ z \neq 0$

shows $\text{is_Arg } (z / w) (\text{Arg } z - \text{Arg } w)$

using *Arg_eq [of z] Arg_eq [of w] Arg_eq [of norm(z / w)] assms*

by (*auto simp: is_Arg_def norm_divide field_simps exp_diff Arg_of_real*)

lemma *Arg_unique_lemma*:

assumes $\text{is_Arg } z t \ \text{is_Arg } z t'$

and $-\pi < t \ t \leq \pi$

and $-\pi < t' \ t' \leq \pi$

and $z \neq 0$

shows $t' = t$

using *is_Arg_eqI* *assms* by force

lemma *complex_norm_eq_1_exp_eq*: $\text{norm } z = 1 \longleftrightarrow \exp(i * (\text{Arg } z)) = z$
 by (*metis* *Arg2pi_eq* *Arg_eq* *complex_norm_eq_1_exp* *norm_eq_zero* *norm_exp_i_times*)

lemma *Arg_unique*: $\llbracket \text{of_real } r * \exp(i * a) = z; 0 < r; -\pi < a; a \leq \pi \rrbracket \implies$
 $\text{Arg } z = a$

by (*rule* *Arg_unique_lemma* [*unfolded is_Arg_def*, *OF _ Arg_eq*])
 (use *mpi_less_Arg* *Arg_le_pi* in $\langle \text{auto simp: norm_mult} \rangle$)

lemma *Arg_minus*:

assumes $z \neq 0$

shows $\text{Arg } (-z) = (\text{if } \text{Arg } z \leq 0 \text{ then } \text{Arg } z + \pi \text{ else } \text{Arg } z - \pi)$

proof -

have [*simp*]: $\text{cmod } z * \cos (\text{Arg } z) = \text{Re } z$

using *assms* *Arg_eq* [*of z*] by (*metis* *Re_exp* *exp_Ln* *norm_exp_eq_Re* *Arg_def*)

have [*simp*]: $\text{cmod } z * \sin (\text{Arg } z) = \text{Im } z$

using *assms* *Arg_eq* [*of z*] by (*metis* *Im_exp* *exp_Ln* *norm_exp_eq_Re* *Arg_def*)

show ?*thesis*

using *mpi_less_Arg* [*of z*] *Arg_le_pi* [*of z*] *assms*

by (*intro* *Arg_unique* [*of norm z*, *OF complex_eqI*]) (*auto simp: Re_exp* *Im_exp*)

qed

lemma *Arg_1* [*simp*]: $\text{Arg } 1 = 0$

by (*rule* *Arg_unique*[*of 1*]) *auto*

lemma *Arg_numeral* [*simp*]: $\text{Arg } (\text{numeral } n) = 0$

by (*rule* *Arg_unique*[*of numeral n*]) *auto*

lemma *Arg_times_of_real* [*simp*]:

assumes $0 < r$ shows $\text{Arg } (\text{of_real } r * z) = \text{Arg } z$

using *Arg_def* *Ln_times_of_real* *assms* by *auto*

lemma *Arg_times_of_real2* [*simp*]: $0 < r \implies \text{Arg } (z * \text{of_real } r) = \text{Arg } z$

by (*metis* *Arg_times_of_real* *mult.commute*)

lemma *Arg_divide_of_real* [*simp*]: $0 < r \implies \text{Arg } (z / \text{of_real } r) = \text{Arg } z$

by (*metis* *Arg_times_of_real2* *less_irrefl* *nonzero_eq_divide_eq_of_real_eq_0_iff*)

lemma *Arg_less_0*: $0 \leq \text{Arg } z \longleftrightarrow 0 \leq \text{Im } z$

using *Im_Ln_le_pi* *Im_Ln_pos_le*

by (*simp* *add: Arg_def*)

converse fails because the argument can equal π .

lemma *Arg_uminus*: $\text{Arg } z < 0 \implies \text{Arg } (-z) > 0$

by (*smt* (*verit*) *Arg_bounded* *Arg_minus* *Complex.Arg_def*)

lemma *Arg_eq_pi*: $Arg\ z = \pi \iff Re\ z < 0 \wedge Im\ z = 0$
by (*auto simp: Arg_def Im_Ln_eq_pi*)

lemma *Arg_lt_pi*: $0 < Arg\ z \wedge Arg\ z < \pi \iff 0 < Im\ z$
using *Arg_less_0 [of z] Im_Ln_pos_lt*
by (*auto simp: order.order_iff_strict Arg_def*)

lemma *Arg_eq_0*: $Arg\ z = 0 \iff z \in \mathbb{R} \wedge 0 \leq Re\ z$
using *Arg_def Im_Ln_eq_0 complex_eq_iff complex_is_Real_iff* **by** *auto*

corollary *Arg_ne_0*: **assumes** $z \notin \mathbb{R}_{\geq 0}$ **shows** $Arg\ z \neq 0$
using *assms* **by** (*auto simp: nonneg_Reals_def Arg_eq_0*)

lemma *Arg_eq_pi_iff*: $Arg\ z = \pi \iff z \in \mathbb{R} \wedge Re\ z < 0$
using *Arg_eq_pi complex_is_Real_iff* **by** *blast*

lemma *Arg_eq_0_pi*: $Arg\ z = 0 \vee Arg\ z = \pi \iff z \in \mathbb{R}$
using *Arg_eq_pi_iff Arg_eq_0* **by** *force*

lemma *Arg_real*: $z \in \mathbb{R} \implies Arg\ z = (if\ 0 \leq Re\ z\ then\ 0\ else\ \pi)$
using *Arg_eq_0 Arg_eq_0_pi* **by** *auto*

lemma *Arg_inverse*: $Arg(inverse\ z) = (if\ z \in \mathbb{R}\ then\ Arg\ z\ else\ -\ Arg\ z)$

proof (*cases z \in \mathbb{R}*)

case *False*

then show *?thesis*

by (*simp add: Arg_def Ln_inverse complex_is_Real_iff complex_nonpos_Reals_iff*)

qed (*use Arg_real Re_inverse in auto*)

lemma *Arg_eq_iff*:

assumes $w \neq 0\ z \neq 0$

shows $Arg\ w = Arg\ z \iff (\exists x. 0 < x \wedge w = of_real\ x * z)$ (**is** *?lhs = ?rhs*)

proof

assume *?lhs*

then have $w = (cmod\ w / cmod\ z) * z$

by (*metis Arg_eq assms divide_divide_eq_right eq_divide_eq exp_not_eq_zero of_real_divide*)

then show *?rhs*

using *assms divide_pos_pos zero_less_norm_iff* **by** *blast*

qed *auto*

lemma *Arg_inverse_eq_0*: $Arg(inverse\ z) = 0 \iff Arg\ z = 0$
by (*metis Arg_eq_0 Arg_inverse inverse_inverse_eq*)

lemma *Arg_cnj_eq_inverse*: $z \neq 0 \implies Arg\ (cnj\ z) = Arg\ (inverse\ z)$
using *Arg2pi_cnj_eq_inverse Arg2pi_eq_iff Arg_eq_iff* **by** *auto*

lemma *Arg_cnj*: $Arg(cnj\ z) = (if\ z \in \mathbb{R}\ then\ Arg\ z\ else\ -\ Arg\ z)$

by (metis Arg_cnj_eq_inverse Arg_inverse Reals_0 complex_cnj_zero)

lemma Arg_exp: $-pi < Im\ z \implies Im\ z \leq pi \implies Arg(exp\ z) = Im\ z$
 by (simp add: Arg_eq_Im_Ln)

lemma Arg_cis: $x \in \{-pi < .. pi\} \implies Arg(cis\ x) = x$
 unfolding cis_conv_exp by (subst Arg_exp) auto

lemma Arg_rcis: $x \in \{-pi < .. pi\} \implies r > 0 \implies Arg(rcis\ r\ x) = x$
 unfolding rcis_def by (subst Arg_times_of_real) (auto simp: Arg_cis)

lemma Ln_Arg: $z \neq 0 \implies Ln(z) = ln(norm\ z) + i * Arg(z)$
 by (metis Arg_def Re_Ln complex_eq)

lemma continuous_at_Arg:
 assumes $z \notin \mathbb{R}_{\leq 0}$
 shows continuous (at z) Arg
proof –
 have $(\lambda z. Im(Ln\ z)) -z \rightarrow Arg\ z$
 using Arg_def assms continuous_at by fastforce
 then show ?thesis
 unfolding continuous_at
 by (smt (verit, del_insts) Arg_eq_Im_Ln Lim_transform_away_at assms
 nonpos_Reals_zero_I)
qed

lemma continuous_within_Arg: $z \notin \mathbb{R}_{\leq 0} \implies continuous\ (at\ z\ within\ S)\ Arg$
 using continuous_at_Arg continuous_at_imp_continuous_within by blast

lemma Arg_Re_pos: $|Arg\ z| < pi / 2 \iff Re\ z > 0 \vee z = 0$
 using Arg_def Re_Ln_pos_lt by auto

lemma Arg_Re_nonneg: $|Arg\ z| \leq pi / 2 \iff Re\ z \geq 0$
 using Re_Ln_pos_le[of z] by (cases z = 0) (auto simp: Arg_eq_Im_Ln Arg_zero)

lemma Arg_times:
 assumes $Arg\ z + Arg\ w \in \{-pi < .. pi\}$ $z \neq 0$ $w \neq 0$
 shows $Arg(z * w) = Arg\ z + Arg\ w$
 using Arg_eq_Im_Ln Ln_times_simple assms by auto

6.20.16 The Unwinding Number and the Ln product Formula

Note that in this special case the unwinding number is -1, 0 or 1. But it's always an integer.

lemma is_Arg_exp_Im: $is_Arg(exp\ z)(Im\ z)$
 using exp_eq_polar is_Arg_def norm_exp_eq_Re by auto

lemma *is_Arg_exp_diff_2pi*:
assumes *is_Arg* (*exp z*) ϑ
shows $\exists k. \text{Im } z - \text{of_int } k * (2 * \text{pi}) = \vartheta$
proof (*intro exI is_Arg_eqI*)
let $?k = \lfloor (\text{Im } z - \vartheta) / (2 * \text{pi}) \rfloor$
show *is_Arg* (*exp z*) ($\text{Im } z - \text{real_of_int } ?k * (2 * \text{pi})$)
by (*metis diff_add_cancel is_Arg_2pi_iff is_Arg_exp_Im*)
show $|\text{Im } z - \text{real_of_int } ?k * (2 * \text{pi}) - \vartheta| < 2 * \text{pi}$
using *floor_divide_upper* [*of 2*pi Im z - \vartheta*] *floor_divide_lower* [*of 2*pi Im z - \vartheta*]
by (*auto simp: algebra_simps abs_if*)
qed (*auto simp: is_Arg_exp_Im assms*)

lemma *Arg_exp_diff_2pi*: $\exists k. \text{Im } z - \text{of_int } k * (2 * \text{pi}) = \text{Arg} (\text{exp } z)$
using *is_Arg_exp_diff_2pi* [*OF is_Arg_Arg*] **by** *auto*

lemma *unwinding_in_Ints*: $(z - \text{Ln}(\text{exp } z)) / (\text{of_real}(2 * \text{pi}) * i) \in \mathbb{Z}$
using *Arg_exp_diff_2pi* [*of z*]
by (*force simp: Ints_def image_def field_simps Arg_def intro!: complex_eqI*)

definition *unwinding* :: *complex* \Rightarrow *int* **where**
unwinding $z \equiv \text{THE } k. \text{of_int } k = (z - \text{Ln}(\text{exp } z)) / (\text{of_real}(2 * \text{pi}) * i)$

lemma *unwinding*: $\text{of_int} (\text{unwinding } z) = (z - \text{Ln}(\text{exp } z)) / (\text{of_real}(2 * \text{pi}) * i)$
using *unwinding_in_Ints* [*of z*]
unfolding *unwinding_def Ints_def* **by** *force*

lemma *unwinding_2pi*: $(2 * \text{pi}) * i * \text{unwinding}(z) = z - \text{Ln}(\text{exp } z)$
by (*simp add: unwinding*)

lemma *Ln_times_unwinding*:
 $w \neq 0 \implies z \neq 0 \implies \text{Ln}(w * z) = \text{Ln}(w) + \text{Ln}(z) - (2 * \text{pi}) * i * \text{unwinding}(\text{Ln } w + \text{Ln } z)$
using *unwinding_2pi* **by** (*simp add: exp_add*)

lemma *arg_conv_arctan*:
assumes $\text{Re } z > 0$
shows $\text{Arg } z = \arctan (\text{Im } z / \text{Re } z)$
proof (*rule cis_Arg_unique*)
show $\text{sgn } z = \text{cis} (\arctan (\text{Im } z / \text{Re } z))$
proof (*rule complex_eqI*)
have $\text{Re} (\text{cis} (\arctan (\text{Im } z / \text{Re } z))) = 1 / \text{sqrt} (1 + (\text{Im } z)^2 / (\text{Re } z)^2)$
by (*simp add: cos_arctan power_divide*)
also have $1 + \text{Im } z ^ 2 / \text{Re } z ^ 2 = \text{norm } z ^ 2 / \text{Re } z ^ 2$
using *assms* **by** (*simp add: cmod_def field_simps*)
also have $1 / \text{sqrt} \dots = \text{Re } z / \text{norm } z$
using *assms* **by** (*simp add: real_sqrt_divide*)
finally show $\text{Re} (\text{sgn } z) = \text{Re} (\text{cis} (\arctan (\text{Im } z / \text{Re } z)))$


```

    by simp
  next
    have  $Im (cis (arctan (Im z / Re z))) = Im z / (Re z * sqrt (1 + (Im z)^2 / (Re z)^2))$ 
      by (simp add: sin_arctan_field_simps)
    also have  $1 + Im z ^ 2 / Re z ^ 2 = norm z ^ 2 / Re z ^ 2$ 
      using assms by (simp add: cmod_def field_simps)
    also have  $Im z / (Re z * sqrt \dots) = Im z / norm z$ 
      using assms by (simp add: real_sqrt_divide)
    finally show  $Im (sgn z) = Im (cis (arctan (Im z / Re z)))$ 
      by simp
  qed
next
  show  $arctan (Im z / Re z) > -pi$ 
    by (smt (verit, ccfv_SIG) arctan_field_sum_of_halves)
next
  show  $arctan (Im z / Re z) \leq pi$ 
    by (smt (verit, best) arctan_field_sum_of_halves)
qed

```

6.20.17 Characterisation of $Im (Ln z)$ (Wenda Li)

lemma *Im_Ln_eq_pi_half*:

$z \neq 0 \implies (Im(Ln z) = pi/2 \iff 0 < Im(z) \wedge Re(z) = 0)$

$z \neq 0 \implies (Im(Ln z) = -pi/2 \iff Im(z) < 0 \wedge Re(z) = 0)$

using *Im_Ln_pos_lt Im_Ln_pos_le Re_Ln_pos_le Re_Ln_pos_lt pi_ge_two*
by *fastforce+*

lemma *Im_Ln_eq*:

assumes $z \neq 0$

shows $Im (Ln z) =$ (if $Re z \neq 0$ then
 if $Re z > 0$ then
 $arctan (Im z / Re z)$
 else if $Im z \geq 0$ then
 $arctan (Im z / Re z) + pi$
 else
 $arctan (Im z / Re z) - pi$
 else
 if $Im z > 0$ then $pi/2$ else $-pi/2$)

proof -

have *eq_arctan_pos*: $Im (Ln z) = arctan (Im z / Re z)$ when $Re z > 0$ for z

by (*metis Arg_eq Im_Ln_arg_conv_arctan order_less_irrefl that zero_complex_simps(1)*)

have *thesis* when $Re z = 0$

using *Im_Ln_eq_pi_half*[OF $\langle z \neq 0 \rangle$] that

using *assms complex_eq_iff* by *auto*

moreover have *thesis* when $Re z > 0$

using *eq_arctan_pos*[OF that] that by *auto*

moreover have *thesis* when $Re z < 0$ $Im z \geq 0$

proof -

```

have  $\text{Im} (\text{Ln} (-z)) = \arctan (\text{Im} (-z) / \text{Re} (-z))$ 
  by (simp add: eq_arctan_pos that(1))
moreover have  $\text{Ln} (-z) = \text{Ln} z - i * \text{complex\_of\_real } \pi$ 
  using Ln_minus assms that by fastforce
ultimately show ?thesis using that by auto
qed
moreover have ?thesis when  $\text{Re } z < 0 \ \text{Im } z < 0$ 
proof -
  have  $\text{Im} (\text{Ln} (-z)) = \arctan (\text{Im} (-z) / \text{Re} (-z))$ 
    by (simp add: eq_arctan_pos that(1))
  moreover have  $\text{Ln} (-z) = \text{Ln} z + i * \text{complex\_of\_real } \pi$ 
    using Ln_minus assms that by auto
  ultimately show ?thesis using that by auto
qed
ultimately show ?thesis by linarith
qed

```

6.20.18 Relation between Ln and Arg2pi, and hence continuity of Arg2pi

```

lemma Arg2pi_Ln:  $0 < \text{Arg2pi } z \implies \text{Arg2pi } z = \text{Im}(\text{Ln}(-z)) + \pi$ 
  by (smt (verit, best) Arg2pi_0 Arg2pi_exp Arg2pi_minus Arg_exp Arg_minus
  Im_Ln_le_pi
  exp_Ln_mpi_less_Im_Ln_neg_equal_0_iff_equal)

```

```

lemma continuous_at_Arg2pi:
  assumes  $z \notin \mathbb{R}_{\geq 0}$ 
  shows continuous (at z) Arg2pi
proof -
  have isCont  $(\lambda z. \text{Im} (\text{Ln} (-z)) + \pi) z$ 
    by (rule Complex.isCont_Im isCont_Ln' continuous_intros | simp add: assms
  complex_is_Real_iff)+
  moreover consider  $\text{Re } z < 0 \mid \text{Im } z \neq 0$  using assms
  using complex_nonneg_Reals_iff_not_le by blast
  ultimately have  $(\lambda z. \text{Im} (\text{Ln} (-z)) + \pi) -z \rightarrow \text{Arg2pi } z$ 
    by (simp add: Arg2pi_Ln Arg2pi_gt_0 assms continuous_within)
  then show ?thesis
    unfolding continuous_at
    by (metis (mono_tags, lifting) Arg2pi_Ln Arg2pi_gt_0 Compl_iff Lim_transform_within_open
  assms
  closed_nonneg_Reals_complex open_Compl)
qed

```

Relation between Arg2pi and arctangent in upper halfplane

```

lemma Arg2pi_arctan_upperhalf:
  assumes  $0 < \text{Im } z$ 
  shows  $\text{Arg2pi } z = \pi/2 - \arctan(\text{Re } z / \text{Im } z)$ 
proof (cases  $z = 0$ )
  case False

```

```

show ?thesis
proof (rule Arg2pi_unique [of norm z])
  show (cmod z) * exp (i * (pi / 2 - arctan (Re z / Im z))) = z
  apply (rule complex_eqI)
  using assms norm_complex_def [of z, symmetric]
  unfolding exp_Euler cos_diff sin_diff sin_of_real cos_of_real
  by (simp_all add: field_simps real_sqrt_divide sin_arctan cos_arctan)
qed (use False arctan [of Re z / Im z] in auto)
qed (use assms in auto)

lemma Arg2pi_eq_Im_Ln:
  assumes 0 ≤ Im z 0 < Re z
  shows Arg2pi z = Im (Ln z)
  by (smt (verit, ccfv_SIG) Arg2pi_exp Im_Ln_pos_le assms exp_Ln_pi_neq_zero
  zero_complex.simps(1))

lemma continuous_within_upperhalf_Arg2pi:
  assumes z ≠ 0
  shows continuous (at z within {z. 0 ≤ Im z}) Arg2pi
proof (cases z ∈ ℝ≥0)
  case False then show ?thesis
    using continuous_at_Arg2pi continuous_at_imp_continuous_within by auto
  next
  case True
  then have z: z ∈ ℝ 0 < Re z
    using assms by (auto simp: complex_nonneg_Reals_iff complex_is_Real_iff
  complex_neq_0)
  then have [simp]: Arg2pi z = 0 Im (Ln z) = 0
    by (auto simp: Arg2pi_eq_0 Im_Ln_eq_0 assms complex_is_Real_iff)
  show ?thesis
proof (clarsimp simp add: continuous_within Lim_within dist_norm)
  fix e::real
  assume 0 < e
  moreover have continuous (at z) (λx. Im (Ln x))
    using z by (simp add: continuous_at_Ln complex_nonpos_Reals_iff)
  ultimately
  obtain d where d: d>0 ∧ x. x ≠ z ⇒ cmod (x - z) < d ⇒ |Im (Ln x)| <
  e
  by (auto simp: continuous_within Lim_within dist_norm)
  { fix x
    assume cmod (x - z) < Re z / 2
    then have |Re x - Re z| < Re z / 2
      by (metis le_less_trans abs_Re_le_cmod minus_complex.simps(1))
    then have 0 < Re x
      using z by linarith
    }
  then show ∃ d>0. ∀ x. 0 ≤ Im x → x ≠ z ∧ cmod (x - z) < d → |Arg2pi
  x| < e
    apply (rule_tac x=min d (Re z / 2) in exI)

```

1992

```
      using z d by (auto simp: Arg2pi_eq_Im_Ln)
    qed
  qed
```

```
lemma continuous_on_upperhalf_Arg2pi: continuous_on ({z. 0 ≤ Im z} - {0})
  Arg2pi
```

```
  unfolding continuous_on_eq_continuous_within
  by (metis DiffE Diff_subset continuous_within_subset continuous_within_upperhalf_Arg2pi
    insertCI)
```

```
lemma open_Arg2pi2pi_less_Int:
```

```
  assumes 0 ≤ s t ≤ 2*pi
    shows open ({y. s < Arg2pi y} ∩ {y. Arg2pi y < t})
  proof -
    have 1: continuous_on (UNIV - ℝ≥0) Arg2pi
      using continuous_at_Arg2pi continuous_at_imp_continuous_within
      by (auto simp: continuous_on_eq_continuous_within)
    have 2: open (UNIV - ℝ≥0 :: complex set) by (simp add: open_Diff)
    have open ({z. s < z} ∩ {z. z < t})
      using open_lessThan [of t] open_greaterThan [of s]
      by (metis greaterThan_def lessThan_def open_Int)
    moreover have {y. s < Arg2pi y} ∩ {y. Arg2pi y < t} ⊆ - ℝ≥0
      using assms by (auto simp: Arg2pi_real complex_nonneg_Reals_iff complex_is_Real_iff)
    ultimately show ?thesis
      using continuous_imp_open_vimage [OF 1 2, of {z. Re z > s} ∩ {z. Re z < t}]
      by auto
  qed
```

```
lemma open_Arg2pi2pi_gt: open {z. t < Arg2pi z}
```

```
proof (cases t < 0)
  case True then have {z. t < Arg2pi z} = UNIV
    using Arg2pi_ge_0 less_le_trans by auto
  then show ?thesis
    by simp
next
  case False then show ?thesis
    using open_Arg2pi2pi_less_Int [of t 2*pi] Arg2pi_lt_2pi
    by auto
qed
```

```
lemma closed_Arg2pi2pi_le: closed {z. Arg2pi z ≤ t}
```

```
  using open_Arg2pi2pi_gt [of t]
  by (simp add: closed_def Set.Collect_neg_eq [symmetric] not_le)
```

6.20.19 Complex Powers

```
lemma powr_to_1 [simp]: z powr 1 = (z::complex)
```

by (simp add: powr_def)

lemma powr_nat:

fixes $n::\text{nat}$ and $z::\text{complex}$ shows $z \text{ powr } n = (\text{if } z = 0 \text{ then } 0 \text{ else } z^n)$
by (simp add: exp_of_nat_mult powr_def)

lemma powr_nat': $(z :: \text{complex}) \neq 0 \vee n \neq 0 \implies z \text{ powr of_nat } n = z^n$
by (cases $z = 0$) (auto simp: powr_nat)

lemma norm_powr_real: $w \in \mathbb{R} \implies 0 < \text{Re } w \implies \text{norm}(w \text{ powr } z) = \text{exp}(\text{Re } z * \ln(\text{Re } w))$
using Ln_Reals_eq norm_exp_eq_Re by (auto simp: Im_Ln_eq_0 powr_def norm_complex_def)

lemma norm_powr_real_powr': $w \in \mathbb{R} \implies \text{norm } (z \text{ powr } w) = \text{norm } z \text{ powr } \text{Re } w$
by (auto simp: powr_def Reals_def)

lemma powr_complexpow [simp]:

fixes $x::\text{complex}$ shows $x \neq 0 \implies x \text{ powr } (\text{of_nat } n) = x^n$
by (simp add: powr_nat')

lemma powr_complexnumeral [simp]:

fixes $x::\text{complex}$ shows $x \text{ powr } (\text{numeral } n) = x^{(\text{numeral } n)}$
by (metis of_nat_numeral power_zero_numeral powr_nat)

lemma cnj_powr:

assumes $\text{Im } a = 0 \implies \text{Re } a \geq 0$

shows $\text{cnj } (a \text{ powr } b) = \text{cnj } a \text{ powr } \text{cnj } b$

proof (cases $a = 0$)

case False

with assms have $a \notin \mathbb{R}_{\leq 0}$ by (auto simp: complex_eq_iff complex_nonpos_Reals_iff)

with False show ?thesis by (simp add: powr_def exp_cnj cnj_Ln)

qed simp

lemma powr_real_real:

assumes $w \in \mathbb{R} \ z \in \mathbb{R} \ 0 < \text{Re } w$

shows $w \text{ powr } z = \text{exp}(\text{Re } z * \ln(\text{Re } w))$

proof -

have $w \neq 0$

using assms by auto

with assms show ?thesis

by (simp add: powr_def Ln_Reals_eq of_real_exp)

qed

lemma powr_of_real:

fixes $x::\text{real}$ and $y::\text{real}$

shows $0 \leq x \implies \text{of_real } x \text{ powr } (\text{of_real } y::\text{complex}) = \text{of_real } (x \text{ powr } y)$

by (simp_all add: powr_def exp_eq_polar)

lemma *powr_of_int*:

fixes $z::\text{complex}$ **and** $n::\text{int}$

assumes $z \neq (0::\text{complex})$

shows $z \text{ powr_of_int } n = (\text{if } n \geq 0 \text{ then } z^{\text{nat } n} \text{ else inverse } (z^{\text{nat } (-n)}))$

by (*metis* *assms* *not_le_of_int_of_nat* *powr_complexpow* *powr_minus*)

lemma *complex_powr_of_int*: $z \neq 0 \vee n \neq 0 \implies z \text{ powr_of_int } n = (z :: \text{complex})^{\text{powi } n}$

by (*cases* $z = 0 \vee n = 0$)

(*auto simp*: *power_int_def* *powr_minus* *powr_nat* *powr_of_int* *power_0_left* *power_inverse*)

lemma *powr_Reals_eq*: $\llbracket x \in \mathbb{R}; y \in \mathbb{R}; \text{Re } x \geq 0 \rrbracket \implies x \text{ powr } y = \text{of_real } (\text{Re } x \text{ powr } \text{Re } y)$

by (*metis* *of_real_Re* *powr_of_real*)

lemma *norm_powr_real_mono*:

$\llbracket w \in \mathbb{R}; 1 < \text{Re } w \rrbracket \implies \text{cmod}(w \text{ powr } z1) \leq \text{cmod}(w \text{ powr } z2) \iff \text{Re } z1 \leq \text{Re } z2$

by (*auto simp*: *powr_def* *algebra_simps* *Reals_def* *Ln_of_real*)

lemma *powr_times_real*:

$\llbracket x \in \mathbb{R}; y \in \mathbb{R}; 0 \leq \text{Re } x; 0 \leq \text{Re } y \rrbracket \implies (x * y) \text{ powr } z = x \text{ powr } z * y \text{ powr } z$

by (*auto simp*: *Reals_def* *powr_def* *Ln_times* *exp_add* *algebra_simps* *less_eq_real_def* *Ln_of_real*)

lemma *Re_powr_le*: $r \in \mathbb{R}_{\geq 0} \implies \text{Re } (r \text{ powr } z) \leq \text{Re } r \text{ powr } \text{Re } z$

by (*auto simp*: *powr_def* *nonneg_Reals_def* *order_trans* [*OF* *complex_Re_le_cmod*])

lemma

fixes $w::\text{complex}$

assumes $w \in \mathbb{R}_{>0}$ $z \in \mathbb{R}$

shows *Reals_powr* [*simp*]: $w \text{ powr } z \in \mathbb{R}$ **and** *nonneg_Reals_powr* [*simp*]: $w \text{ powr } z \in \mathbb{R}_{>0}$

using *assms* **by** (*auto simp*: *nonneg_Reals_def* *Reals_def* *powr_of_real*)

lemma *exp_powr_complex*:

fixes $x::\text{complex}$

assumes $-\pi < \text{Im}(x)$ $\text{Im}(x) \leq \pi$

shows $\exp x \text{ powr } y = \exp (x * y)$

using *assms* **by** (*simp add*: *powr_def* *mult commute*)

lemma *powr_neg_real_complex*:

fixes $w::\text{complex}$

shows $(-\text{of_real } x) \text{ powr } w = (-1) \text{ powr } (\text{of_real } (\text{sgn } x) * w) * \text{of_real } x \text{ powr } w$

proof (*cases* $x = 0$)

assume $x: x \neq 0$

```

hence  $(-x)$  powr  $w = \exp (w * \ln (-\text{of\_real } x))$  by (simp add: powr_def)
also from  $x$  have  $\ln (-\text{of\_real } x) = \text{Ln } (\text{of\_real } x) + \text{of\_real } (\text{sgn } x) * \pi * i$ 
  by (simp add: Ln_minus Ln_of_real)
also from  $x$  have  $\exp (w * \dots) = \text{cis } \pi \text{ powr } (\text{of\_real } (\text{sgn } x) * w) * \text{of\_real } x$ 
powr  $w$ 
  by (simp add: powr_def exp_add algebra_simps Ln_of_real cis_conv_exp)
also note cis_pi
finally show ?thesis by simp
qed simp_all

```

lemma *has_field_derivative_powr*:

```

fixes  $z :: \text{complex}$ 
assumes  $z \notin \mathbb{R}_{\leq 0}$ 
shows  $((\lambda z. z \text{ powr } s) \text{ has\_field\_derivative } (s * z \text{ powr } (s - 1)))$  (at  $z$ )
proof (cases z=0)
  case False
    then have  $\S$ :  $\exp (s * \text{Ln } z) * \text{inverse } z = \exp ((s - 1) * \text{Ln } z)$ 
      by (simp add: divide_complex_def exp_diff left_diff_distrib')
    show ?thesis
      unfolding powr_def
    proof (rule has_field_derivative_transform_within)
      show  $((\lambda z. \exp (s * \text{Ln } z)) \text{ has\_field\_derivative } s * (\text{if } z = 0 \text{ then } 0 \text{ else } \exp$ 
 $((s - 1) * \text{Ln } z)))$ 
        (at  $z$ )
      by (intro derivative_eq_intros | simp add: assms False \S)+
    qed (use False in auto)
qed (use assms in auto)

```

declare *has_field_derivative_powr*[*THEN DERIV_chain2, derivative_intros*]

lemma *has_field_derivative_powr_of_int*:

```

fixes  $z :: \text{complex}$ 
assumes gderiv:  $(g \text{ has\_field\_derivative } gd)$  (at  $z$  within  $S$ ) and  $g z \neq 0$ 
shows  $((\lambda z. g z \text{ powr } \text{of\_int } n) \text{ has\_field\_derivative } (n * g z \text{ powr } (\text{of\_int } n -$ 
 $1) * gd))$  (at  $z$  within  $S$ )
proof -
  obtain  $e$  where  $e > 0$  and  $e\_dist$ :  $\forall y \in S. \text{dist } z y < e \longrightarrow g y \neq 0$ 
    using DERIV_continuous assms continuous_within_avoid gderiv by blast
  define  $D$  where  $D = \text{of\_int } n * g z \text{ powr } (\text{of\_int } (n - 1)) * gd$ 
  define  $E$  where  $E = \text{of\_int } n * g z \text{ powr } (n - 1) * gd$ 
  have  $((\lambda z. g z \text{ powr } \text{of\_int } n) \text{ has\_field\_derivative } D)$  (at  $z$  within  $S$ )
     $\longleftrightarrow ((\lambda z. g z \text{ powr } \text{of\_int } n) \text{ has\_field\_derivative } E)$  (at  $z$  within  $S$ )
    using assms complex_powr_of_int D_def E_def by presburger
  also have  $\dots \longleftrightarrow ((\lambda z. g z \text{ powr } n) \text{ has\_field\_derivative } E)$  (at  $z$  within  $S$ )
  proof (rule has_field_derivative_cong_eventually)
    show  $\forall_F x$  in at  $z$  within  $S$ .  $g x \text{ powr } \text{of\_int } n = g x \text{ powr } n$ 
    unfolding eventually_at by (metis <0 < e> complex_powr_of_int dist_commute
 $e\_dist$ )

```

```

qed (simp add: assms complex_powr_of_int)
also have (( $\lambda z. g z \text{ powr } n$ ) has_field_derivative E) (at z within S)
  unfolding E_def using gderiv assms by (auto intro!: derivative_eq_intros)
finally show ?thesis
  by (simp add: D_def)
qed

```

```

lemma field_differentiable_powr_of_int:
  fixes z :: complex
  assumes g field_differentiable (at z within S) and g z  $\neq 0$ 
  shows ( $\lambda z. g z \text{ powr of\_int } n$ ) field_differentiable (at z within S)
  using has_field_derivative_powr_of_int assms field_differentiable_def by blast

```

```

lemma holomorphic_on_powr_of_int [holomorphic_intros]:
  assumes f holomorphic_on S and  $\bigwedge z. z \in S \implies f z \neq 0$ 
  shows ( $\lambda z. (f z) \text{ powr of\_int } n$ ) holomorphic_on S
  using assms field_differentiable_powr_of_int holomorphic_on_def by auto

```

```

lemma has_field_derivative_powr_right [derivative_intros]:
   $w \neq 0 \implies ((\lambda z. w \text{ powr } z) \text{ has\_field\_derivative } Ln w * w \text{ powr } z)$  (at z)
  unfolding powr_def by (intro derivative_eq_intros | simp)+

```

```

lemma field_differentiable_powr_right [derivative_intros]:
  fixes w :: complex
  shows  $w \neq 0 \implies (\lambda z. w \text{ powr } z)$  field_differentiable (at z)
using field_differentiable_def has_field_derivative_powr_right by blast

```

```

lemma holomorphic_on_powr_right [holomorphic_intros]:
  assumes f holomorphic_on S
  shows ( $\lambda z. w \text{ powr } (f z)$ ) holomorphic_on S
proof (cases w = 0)
  case False
  with assms show ?thesis
  unfolding holomorphic_on_def field_differentiable_def
  by (metis (full_types) DERIV_chain' has_field_derivative_powr_right)
qed simp

```

```

lemma holomorphic_on_divide_gen [holomorphic_intros]:
  assumes f holomorphic_on S g holomorphic_on S and  $\bigwedge z z'. \llbracket z \in S; z' \in S \rrbracket$ 
 $\implies g z = 0 \iff g z' = 0$ 
  shows ( $\lambda z. f z / g z$ ) holomorphic_on S
  by (metis (no_types, lifting) assms division_ring_divide_zero holomorphic_on_divide
holomorphic_transform)

```

```

lemma norm_powr_real_powr:
   $w \in \mathbb{R} \implies 0 \leq \text{Re } w \implies \text{cmod } (w \text{ powr } z) = \text{Re } w \text{ powr } \text{Re } z$ 
  by (metis dual_order.order_iff_strict norm_powr_real norm_zero of_real_0
of_real_Re powr_def)

```


lemma *tendsto_powr_complex*:
fixes $f\ g :: _ \Rightarrow \text{complex}$
assumes $a: a \notin \mathbb{R}_{\leq 0}$
assumes $f: (f \longrightarrow a) F$ **and** $g: (g \longrightarrow b) F$
shows $((\lambda z. f\ z\ \text{powr}\ g\ z) \longrightarrow a\ \text{powr}\ b) F$
proof –
from a **have** $[simp]: a \neq 0$ **by** *auto*
from $f\ g\ a$ **have** $((\lambda z. \text{exp}\ (g\ z * \ln\ (f\ z))) \longrightarrow a\ \text{powr}\ b) F$ (**is** $?P$)
by (*auto intro!: tendsto_intros simp: powr_def*)
also {
have *eventually* $(\lambda z. z \neq 0)$ (*nhds* a)
by (*intro tI_space_nhds simp_all*)
with f **have** *eventually* $(\lambda z. f\ z \neq 0) F$ **using** *filterlim_iff* **by** *blast*
}
hence $?P \longleftrightarrow ((\lambda z. f\ z\ \text{powr}\ g\ z) \longrightarrow a\ \text{powr}\ b) F$
by (*intro tendsto_cong refl*) (*simp_all add: powr_def mult_ac*)
finally show $?thesis$.
qed

lemma *tendsto_powr_complex_0*:
fixes $f\ g :: 'a \Rightarrow \text{complex}$
assumes $f: (f \longrightarrow 0) F$ **and** $g: (g \longrightarrow b) F$ **and** $b: \text{Re}\ b > 0$
shows $((\lambda z. f\ z\ \text{powr}\ g\ z) \longrightarrow 0) F$
proof (*rule tendsto_norm_zero_cancel*)
define h **where**
 $h = (\lambda z. \text{if}\ f\ z = 0\ \text{then}\ 0\ \text{else}\ \text{exp}\ (\text{Re}\ (g\ z) * \ln\ (\text{cmod}\ (f\ z)) + \text{abs}\ (\text{Im}\ (g\ z))$
 $* \text{pi}))$
{
fix $z :: 'a$ **assume** $z: f\ z \neq 0$
define c **where** $c = \text{abs}\ (\text{Im}\ (g\ z)) * \text{pi}$
from *mpi_less_Im_Ln[OF z] Im_Ln_le_pi[OF z]*
have $\text{abs}\ (\text{Im}\ (\ln\ (f\ z))) \leq \text{pi}$ **by** *simp*
from *mult_left_mono[OF this, of abs (Im (g z))]*
have $\text{abs}\ (\text{Im}\ (g\ z) * \text{Im}\ (\ln\ (f\ z))) \leq c$ **by** (*simp add: abs_mult c_def*)
hence $-\text{Im}\ (g\ z) * \text{Im}\ (\ln\ (f\ z)) \leq c$ **by** *simp*
hence $\text{norm}\ (f\ z\ \text{powr}\ g\ z) \leq h\ z$ **by** (*simp add: powr_def field_simps h_def*
 c_def)
}
hence $le: \text{norm}\ (f\ z\ \text{powr}\ g\ z) \leq h\ z$ **for** z
by (*simp add: h_def*)

have $g': (g \longrightarrow b) (\text{inf}\ F\ (\text{principal}\ \{z. f\ z \neq 0\}))$
by (*rule tendsto_mono[OF _ g] simp_all*)
have $((\lambda x. \text{norm}\ (f\ x)) \longrightarrow 0) (\text{inf}\ F\ (\text{principal}\ \{z. f\ z \neq 0\}))$
by (*subst tendsto_norm_zero_iff, rule tendsto_mono[OF _ f] simp_all*)
moreover {
have *filterlim* $(\lambda x. \text{norm}\ (f\ x))$ (*principal* $\{0<..\}$) (*principal* $\{z. f\ z \neq 0\}$)
by (*auto simp: filterlim_def*)
hence *filterlim* $(\lambda x. \text{norm}\ (f\ x))$ (*principal* $\{0<..\}$) (*inf* F (*principal* $\{z. f\ z \neq 0\}$))

```

0}))
  by (rule filterlim_mono) simp_all
}
ultimately have norm: filterlim ( $\lambda x. \text{norm } (f x)$ ) (at_right 0) (inf F (principal
{z. f z  $\neq$  0}))
  by (simp add: filterlim_inf at_within_def)

have A: LIM x inf F (principal {z. f z  $\neq$  0}). Re (g x) * -ln (cmod (f x)) :>
at_top
  by (rule filterlim_tendsto_pos_mult_at_top tendsto_intros g' b
      filterlim_compose[OF filterlim_uminus_at_top_at_bot] filterlim_compose[OF
ln_at_0] norm)+
have B: LIM x inf F (principal {z. f z  $\neq$  0}).
  -|Im (g x)| * pi + -(Re (g x) * ln (cmod (f x))) :> at_top
  by (rule filterlim_tendsto_add_at_top tendsto_intros g') + (insert A, simp_all)
have C: (h  $\longrightarrow$  0) F unfolding h_def
  by (intro filterlim_If tendsto_const filterlim_compose[OF exp_at_bot])
  (insert B, auto simp: filterlim_uminus_at_bot algebra_simps)
show (( $\lambda x. \text{norm } (f x \text{ powr } g x)$ )  $\longrightarrow$  0) F
  by (rule Lim_null_comparison[OF always_eventually C]) (insert le, auto)
qed

```

lemma *tendsto_powr_complex'* [tendsto_intros]:
fixes $f g :: _ \Rightarrow \text{complex}$
assumes $a \notin \mathbb{R}_{\leq 0} \vee (a = 0 \wedge \text{Re } b > 0)$ **and** ($f \longrightarrow a$) F ($g \longrightarrow b$) F
shows (($\lambda z. f z \text{ powr } g z$) \longrightarrow a powr b) F
using *assms tendsto_powr_complex tendsto_powr_complex_0* **by** *fastforce*

lemma *tendsto_neg_powr_complex_of_real*:
assumes *filterlim f at_top F* **and** $\text{Re } s < 0$
shows (($\lambda x. \text{complex_of_real } (f x) \text{ powr } s$) \longrightarrow 0) F
proof -
have (($\lambda x. \text{norm } (\text{complex_of_real } (f x) \text{ powr } s)$) \longrightarrow 0) F
proof (rule *Lim_transform_eventually*)
from *assms(1)* **have** *eventually* ($\lambda x. f x \geq 0$) F
by (*auto simp: filterlim_at_top*)
thus *eventually* ($\lambda x. f x \text{ powr } \text{Re } s = \text{norm } (\text{of_real } (f x) \text{ powr } s)$) F
by *eventually_elim (simp add: norm_powr_real_powr)*
from *assms* **show** (($\lambda x. f x \text{ powr } \text{Re } s$) \longrightarrow 0) F
by (*intro tendsto_neg_powr*)
qed
thus *?thesis* **by** (*simp add: tendsto_norm_zero_iff*)
qed

lemma *tendsto_neg_powr_complex_of_nat*:
assumes *filterlim f at_top F* **and** $\text{Re } s < 0$
shows (($\lambda x. \text{of_nat } (f x) \text{ powr } s$) \longrightarrow 0) F
using *tendsto_neg_powr_complex_of_real [of real o f F s]*
proof -

```

have (( $\lambda x.$  of_real (real (f x)) powr s)  $\longrightarrow$  0) F using assms(2)
by (intro filterlim_compose[OF tendsto_neg_powr_complex_of_real]
      filterlim_compose[OF assms(1)] filterlim_real_sequentially filter-
lim_ident) auto
thus ?thesis by simp
qed

```

```

lemma continuous_powr_complex:
  assumes f (netlimit F)  $\notin$   $\mathbb{R}_{\leq 0}$  continuous F f continuous F g
  shows continuous F ( $\lambda z.$  f z powr g z :: complex)
  using assms unfolding continuous_def by (intro tendsto_powr_complex) simp_all

```

```

lemma isCont_powr_complex [continuous_intros]:
  assumes f z  $\notin$   $\mathbb{R}_{\leq 0}$  isCont f z isCont g z
  shows isCont ( $\lambda z.$  f z powr g z :: complex) z
  using assms unfolding isCont_def by (intro tendsto_powr_complex) simp_all

```

```

lemma continuous_on_powr_complex [continuous_intros]:
  assumes  $A \subseteq \{z. \text{Re } (f z) \geq 0 \vee \text{Im } (f z) \neq 0\}$ 
  assumes  $\bigwedge z. z \in A \implies f z = 0 \implies \text{Re } (g z) > 0$ 
  assumes continuous_on A f continuous_on A g
  shows continuous_on A ( $\lambda z.$  f z powr g z)
  unfolding continuous_on_def

```

```

proof
  fix z assume z: z  $\in$  A
  show (( $\lambda z.$  f z powr g z)  $\longrightarrow$  f z powr g z) (at z within A)
  proof (cases f z = 0)
    case False
      from assms(1,2) z have  $\text{Re } (f z) \geq 0 \vee \text{Im } (f z) \neq 0$  f z = 0  $\longrightarrow$   $\text{Re } (g z) >$ 
0 by auto
      with assms(3,4) z show ?thesis
      by (intro tendsto_powr_complex')
      (auto elim!: nonpos_Reals_cases simp: complex_eq_iff continuous_on_def)
    next
      case True
      with assms z show ?thesis
      by (auto intro!: tendsto_powr_complex_0 simp: continuous_on_def)
  qed
qed

```

6.20.20 Some Limits involving Logarithms

```

lemma lim_Ln_over_power:
  fixes s::complex
  assumes 0 < Re s
  shows ( $\lambda n.$  Ln (of_nat n) / of_nat n powr s)  $\longrightarrow$  0
proof (simp add: lim_sequentially_dist_norm, clarify)
  fix e::real
  assume e: 0 < e

```

2000

```

have  $\exists x_0 > 0. \forall x \geq x_0. 0 < e * 2 + (e * Re\ s * 2 - 2) * x + e * (Re\ s)^2 * x^2$ 
proof (rule_tac x=2/(e * (Re s)^2) in exI, safe)
  show  $0 < 2 / (e * (Re\ s)^2)$ 
  using e assms by (simp add: field_simps)
next
fix x::real
assume x:  $2 / (e * (Re\ s)^2) \leq x$ 
have  $2 / (e * (Re\ s)^2) > 0$ 
  using e assms by simp
with x have  $x > 0$ 
  by linarith
then have  $x * 2 \leq e * (x^2 * (Re\ s)^2)$ 
  using e assms x by (auto simp: power2_eq_square field_simps)
also have  $\dots < e * (2 + (x * (Re\ s * 2) + x^2 * (Re\ s)^2))$ 
  using e assms  $\langle x > 0 \rangle$ 
  by (auto simp: power2_eq_square field_simps add_pos_pos)
finally show  $0 < e * 2 + (e * Re\ s * 2 - 2) * x + e * (Re\ s)^2 * x^2$ 
  by (auto simp: algebra_simps)
qed
then have  $\exists x_0 > 0. \forall x \geq x_0. x / e < 1 + (Re\ s * x) + (1/2) * (Re\ s * x)^2$ 
  using e by (simp add: field_simps)
then have  $\exists x_0 > 0. \forall x \geq x_0. x / e < \exp (Re\ s * x)$ 
  using assms
  by (force intro: less_le_trans [OF exp_lower_Taylor_quadratic])
then obtain x_0 where  $x_0 > 0$  and  $x_0: \bigwedge x. x \geq x_0 \implies x < e * \exp (Re\ s * x)$ 
  using e by (auto simp: field_simps)
have norm (Ln (of_nat n) / of_nat n powr s) < e if  $n \geq \text{nat } [\exp\ x_0]$  for n
proof -
  have  $\ln (\text{real } n) \geq x_0$ 
  using that exp_gt_zero ln_ge_iff [of n] nat_ceiling_le_eq by fastforce
  then show ?thesis
  using e x_0 [of ln n] by (auto simp: norm_divide norm_powr_real field_split_simps)
qed
then show  $\exists n_0. \forall n \geq n_0. \text{norm } (\text{Ln } (\text{of\_nat } n) / \text{of\_nat } n \text{ powr } s) < e$ 
  by blast
qed

lemma lim_Ln_over_n:  $((\lambda n. \text{Ln}(\text{of\_nat } n) / \text{of\_nat } n) \longrightarrow 0)$  sequentially
  using lim_Ln_over_power [of 1] by simp

lemma lim_ln_over_power:
  fixes s :: real
  assumes  $0 < s$ 
  shows  $(\lambda n. \ln (\text{real } n) / \text{real } n \text{ powr } s) \longrightarrow 0$ 
proof -
  have  $(\lambda n. \ln (\text{Suc } n) / (\text{Suc } n) \text{ powr } s) \longrightarrow 0$ 
  using lim_Ln_over_power [of of_real s, THEN filterlim_sequentially_Suc
  [THEN iffD2]] assms
  by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
```

```

norm_divide)
  then show ?thesis
    using filterlim_sequentially_Suc[of  $\lambda n::nat. \ln n / n$   $\text{powr } s$ ] by auto
qed

```

```

lemma lim_ln_over_n [tendsto_intros]: (( $\lambda n. \ln(\text{real\_of\_nat } n) / \text{of\_nat } n$ )
 $\longrightarrow 0$ ) sequentially
  using lim_ln_over_power [of 1] by auto

```

```

lemma lim_log_over_n [tendsto_intros]:
  ( $\lambda n. \log k n/n$ )  $\longrightarrow 0$ 
proof -
  have *:  $\log k n/n = (1/\ln k) * (\ln n / n)$  for  $n$ 
    unfolding log_def by auto
  have ( $\lambda n. (1/\ln k) * (\ln n / n)$ )  $\longrightarrow (1/\ln k) * 0$ 
    by (intro tendsto_intros)
  then show ?thesis
    unfolding * by auto
qed

```

```

lemma lim_1_over_complex_power:
  assumes  $0 < \text{Re } s$ 
  shows ( $\lambda n. 1 / \text{of\_nat } n \text{ powr } s$ )  $\longrightarrow 0$ 
proof (rule Lim_null_comparison)
  have  $\forall n > 0. \exists \leq n \longrightarrow 1 \leq \ln(\text{real\_of\_nat } n)$ 
    using ln_272_gt_1
    by (force intro: order_trans [of _  $\ln(272/100)$ ])
  then show  $\forall_F x$  in sequentially.  $\text{cmod}(1 / \text{of\_nat } x \text{ powr } s) \leq \text{cmod}(\text{Ln}(\text{of\_nat } x) / \text{of\_nat } x \text{ powr } s)$ 
    by (auto simp: norm_divide field_split_simps eventually_sequentially)
  show ( $\lambda n. \text{cmod}(\text{Ln}(\text{of\_nat } n) / \text{of\_nat } n \text{ powr } s)$ )  $\longrightarrow 0$ 
    using lim_Ln_over_power [OF assms] by (metis tendsto_norm_zero_iff)
qed

```

```

lemma lim_1_over_real_power:
  fixes  $s :: \text{real}$ 
  assumes  $0 < s$ 
  shows (( $\lambda n. 1 / (\text{of\_nat } n \text{ powr } s)$ )  $\longrightarrow 0$ ) sequentially
  using lim_1_over_complex_power [of  $\text{of\_real } s$ , THEN filterlim_sequentially_Suc
[THEN iffD2]] assms
  apply (subst filterlim_sequentially_Suc [symmetric])
  by (simp add: lim_sequentially_dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)

```

```

lemma lim_1_over_Ln: ( $\lambda n. 1 / \text{Ln}(\text{complex\_of\_nat } n)$ )  $\longrightarrow 0$ 
proof (clarsimp simp add: lim_sequentially_dist_norm norm_divide field_split_simps)
  fix  $r :: \text{real}$ 
  assume  $0 < r$ 
  have  $\text{ir} : \text{inverse}(\text{exp}(\text{inverse } r)) > 0$ 

```

```

  by simp
  obtain n where n: 1 < of_nat n * inverse (exp (inverse r))
    using ex_less_of_nat_mult [of_ 1, OF ir]
  by auto
  then have exp (inverse r) < of_nat n
    by (simp add: field_split_simps)
  then have ln (exp (inverse r)) < ln (of_nat n)
    by (metis exp_gt_zero less_trans ln_exp ln_less_cancel_iff)
  with ⟨0 < r⟩ have 1 < r * ln (real_of_nat n)
    by (simp add: field_simps)
  moreover have n > 0 using n
    using neq0_conv by fastforce
  ultimately show ∃ no. ∀ k. Ln (of_nat k) ≠ 0 ⟶ no ≤ k ⟶ 1 < r * cmod
    (Ln (of_nat k))
    using n ⟨0 < r⟩
  by (rule_tac x=n in exI) (force simp: field_split_simps intro: less_le_trans)
qed

```

```

lemma lim_1_over_ln: (λn. 1 / ln (real n)) ⟶ 0
  using lim_1_over_Ln [THEN filterlim_sequentially_Suc [THEN iffD2]]
  apply (subst filterlim_sequentially_Suc [symmetric])
  by (simp add: lim_sequentially_dist_norm Ln_Reals_eq norm_powr_real_powr
    norm_divide)

```

```

lemma lim_ln1_over_ln: (λn. ln(Suc n) / ln n) ⟶ 1
proof (rule Lim_transform_eventually)
  have (λn. ln(1 + 1/n) / ln n) ⟶ 0
  proof (rule Lim_transform_bound)
    show (inverse o real) ⟶ 0
      by (metis comp_def lim_inverse_n lim_explicit)
    show ∀F n in sequentially. norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse o
    real) n)

```

proof

fix n::nat

assume n: 3 ≤ n

then have ln 3 ≤ ln n **and** ln0: 0 ≤ ln n

by auto

with ln3_gt_1 **have** 1 / ln n ≤ 1

by (simp add: field_split_simps)

moreover have ln (1 + 1 / real n) ≤ 1/n

by (simp add: ln_add_one_self_le_self)

ultimately have ln (1 + 1 / real n) * (1 / ln n) ≤ (1/n) * 1

by (intro mult_mono) (use n **in** auto)

then show norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse o real) n)

by (simp add: field_simps ln0)

qed

qed

then show (λn. 1 + ln(1 + 1/n) / ln n) ⟶ 1

by (metis (full_types) add.right_neutral tendsto_add_const_iff)

```

show  $\forall_F k$  in sequentially.  $1 + \ln(1 + 1/k) / \ln k = \ln(\text{Suc } k) / \ln k$ 
  by (simp add: field_split_simps ln_div eventually_sequentiallyI [of 2])
qed

```

```

lemma lim_ln_over_ln1:  $(\lambda n. \ln n / \ln(\text{Suc } n)) \longrightarrow 1$ 
  using tendsto_inverse [OF lim_ln1_over_ln] by force

```

6.20.21 Relation between Square Root and exp/ln, hence its derivative

```

lemma csqrt_exp_Ln:
  assumes  $z \neq 0$ 
  shows  $\text{csqrt } z = \exp(\text{Ln } z / 2)$ 
proof -
  have  $(\exp(\text{Ln } z / 2))^2 = (\exp(\text{Ln } z))$ 
    by (metis exp_double nonzero_mult_div_cancel_left times_divide_eq_right
      zero_neq_numeral)
  also have  $\dots = z$ 
    using assms exp_Ln by blast
  finally have  $\text{csqrt } z = \text{csqrt}((\exp(\text{Ln } z / 2))^2)$ 
    by simp
  also have  $\dots = \exp(\text{Ln } z / 2)$ 
    apply (rule csqrt_square)
    using cos_gt_zero_pi [of  $(\text{Im } (\text{Ln } z) / 2)$ ] Im_Ln_le_pi mpi_less_Im_Ln
  assms
    by (fastforce simp: Re_exp Im_exp)
  finally show ?thesis using assms csqrt_square
    by simp
qed

```

```

lemma csqrt_conv_powr:  $\text{csqrt } z = z \text{ powr } (1/2)$ 
  by (auto simp: csqrt_exp_Ln powr_def)

```

```

lemma csqrt_mult:
  assumes  $\text{Arg } z + \text{Arg } w \in \{-\pi < .. \pi\}$ 
  shows  $\text{csqrt}(z * w) = \text{csqrt } z * \text{csqrt } w$ 
proof (cases  $z = 0 \vee w = 0$ )
  case False
  have  $\text{csqrt}(z * w) = \exp((\ln(z * w)) / 2)$ 
    using False by (intro csqrt_exp_Ln) auto
  also have  $\dots = \exp((\text{Ln } z + \text{Ln } w) / 2)$ 
    using False assms by (subst Ln_times_simple) (auto simp: Arg_eq_Im_Ln)
  also have  $(\text{Ln } z + \text{Ln } w) / 2 = \text{Ln } z / 2 + \text{Ln } w / 2$ 
    by (simp add: add_divide_distrib)
  also have  $\exp \dots = \text{csqrt } z * \text{csqrt } w$ 
    using False by (simp add: exp_add csqrt_exp_Ln)
  finally show ?thesis .
qed auto

```

lemma *Arg_csqrt [simp]*: $\text{Arg} (\text{csqrt } z) = \text{Arg } z / 2$
proof (*cases* $z = 0$)
 case *False*
 have $\text{Im} (\text{Ln } z) \in \{-\pi < .. \pi\}$
 by (*simp add: False Im_Ln_le_pi mpi_less_Im_Ln*)
 also have $\dots \subseteq \{-2 * \pi < .. 2 * \pi\}$
 by *auto*
 finally show *?thesis*
 using *False* **by** (*auto simp: csqrt_exp_Ln Arg_exp Arg_eq_Im_Ln*)
qed (*auto simp: Arg_zero*)

lemma *csqrt_inverse*:
 $z \notin \mathbb{R}_{\leq 0} \implies \text{csqrt} (\text{inverse } z) = \text{inverse} (\text{csqrt } z)$
by (*metis Ln_inverse csqrt_eq_0 csqrt_exp_Ln divide_minus_left exp_minus inverse_nonzero_iff_nonzero*)

lemma *cnj_csqrt*: $z \notin \mathbb{R}_{\leq 0} \implies \text{cnj}(\text{csqrt } z) = \text{csqrt}(\text{cnj } z)$
by (*metis cnj_Ln complex_cnj_divide complex_cnj_numeral complex_cnj_zero_iff csqrt_eq_0 csqrt_exp_Ln exp_cnj*)

lemma *has_field_derivative_csqrt*:
assumes $z \notin \mathbb{R}_{\leq 0}$
 shows $(\text{csqrt } \text{has_field_derivative } \text{inverse}(2 * \text{csqrt } z)) \text{ (at } z)$
proof –
 have $z: z \neq 0$
 using *assms* **by** *auto*
 then have $*$: $\text{inverse } z = \text{inverse} (2 * z) * 2$
 by (*simp add: field_split_simps*)
 have [*simp*]: $\text{exp} (\text{Ln } z / 2) * \text{inverse } z = \text{inverse} (\text{csqrt } z)$
 by (*simp add: z field_simps csqrt_exp_Ln [symmetric]*) (*metis power2_csqrt power2_eq_square*)
 have $\text{Im } z = 0 \implies 0 < \text{Re } z$
 using *assms* *complex_nonpos_Reals_iff_not_less* **by** *blast*
 with z **have** $((\lambda z. \text{exp} (\text{Ln } z / 2)) \text{ has_field_derivative } \text{inverse} (2 * \text{csqrt } z))$
 (*at* z)
 by (*force intro: derivative_eq_intros * simp add: assms*)
 then show *?thesis*
 proof (*rule has_field_derivative_transform_within*)
 show $\bigwedge x. \text{dist } x z < \text{cmod } z \implies \text{exp} (\text{Ln } x / 2) = \text{csqrt } x$
 by (*metis csqrt_exp_Ln dist_0_norm less_irrefl*)
 qed (*use* z **in** *auto*)
qed

lemma *field_differentiable_at_csqrt*:
 $z \notin \mathbb{R}_{\leq 0} \implies \text{csqrt } \text{field_differentiable } \text{at } z$
using *field_differentiable_def has_field_derivative_csqrt* **by** *blast*

lemma *field_differentiable_within_csqrt*:
 $z \notin \mathbb{R}_{\leq 0} \implies \text{csqrt } \text{field_differentiable} \text{ (at } z \text{ within } s)$

using field_differentiable_at_csqrt field_differentiable_within_subset by blast

lemma continuous_at_csqrt:

$z \notin \mathbb{R}_{\leq 0} \implies \text{continuous (at } z) \text{ csqrt}$

by (simp add: field_differentiable_within_csqrt field_differentiable_imp_continuous_at)

corollary isCont_csqrt' [simp]:

$\llbracket \text{isCont } f \ z; f \ z \notin \mathbb{R}_{\leq 0} \rrbracket \implies \text{isCont } (\lambda x. \text{csqrt } (f \ x)) \ z$

by (blast intro: isCont_o2 [OF _ continuous_at_csqrt])

lemma continuous_within_csqrt:

$z \notin \mathbb{R}_{\leq 0} \implies \text{continuous (at } z \text{ within } s) \text{ csqrt}$

by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_csqrt)

lemma continuous_on_csqrt [continuous_intros]:

$\text{continuous_on } (-\mathbb{R}_{\leq 0}) \text{ csqrt}$

by (simp add: continuous_at_imp_continuous_on continuous_within_csqrt)

lemma holomorphic_on_csqrt [holomorphic_intros]: csqrt holomorphic_on $-\mathbb{R}_{\leq 0}$

by (simp add: field_differentiable_within_csqrt holomorphic_on_def)

lemma holomorphic_on_csqrt' [holomorphic_intros]:

$f \text{ holomorphic_on } A \implies (\bigwedge z. z \in A \implies f \ z \notin \mathbb{R}_{\leq 0}) \implies (\lambda z. \text{csqrt } (f \ z)) \text{ holomorphic_on } A$

using holomorphic_on_compose_gen[OF _ holomorphic_on_csqrt, of f A] by (auto simp: o_def)

lemma analytic_on_csqrt [analytic_intros]: csqrt analytic_on $-\mathbb{R}_{\leq 0}$

using holomorphic_on_csqrt by (subst analytic_on_open) auto

lemma analytic_on_csqrt' [analytic_intros]:

$f \text{ analytic_on } A \implies (\bigwedge z. z \in A \implies f \ z \notin \mathbb{R}_{\leq 0}) \implies (\lambda z. \text{csqrt } (f \ z)) \text{ analytic_on } A$

using analytic_on_compose_gen[OF _ analytic_on_csqrt, of f A] by (auto simp: o_def)

lemma continuous_within_closed_nontrivial:

$\text{closed } s \implies a \notin s \implies \text{continuous (at } a \text{ within } s) \text{ f}$

using Compl_iff continuous_within_topological_open_Cmpl by fastforce

lemma continuous_within_csqrt_posreal:

$\text{continuous (at } z \text{ within } (\mathbb{R} \cap \{w. 0 \leq \text{Re}(w)\})) \text{ csqrt}$

proof (cases $z \in \mathbb{R}_{\leq 0}$)

case True

then have [simp]: $\text{Im } z = 0$ and $0: \text{Re } z < 0 \vee z = 0$

using complex_nonpos_Reals_iff_complex_eq_iff by force+

show ?thesis

using 0

proof

```

    assume  $Re\ z < 0$ 
    then show ?thesis
    by (auto simp: continuous_within_closed_nontrivial [OF closed_Real_halfspace_Re_ge])
next
assume  $z = 0$ 
moreover
have  $\bigwedge e. 0 < e$ 
   $\implies \forall x' \in \mathbb{R} \cap \{w. 0 \leq Re\ w\}. cmod\ x' < e^2 \implies cmod\ (csqrt\ x') < e$ 
  by (auto simp: Reals_def real_less_sqrt)
ultimately show ?thesis
  using zero_less_power by (fastforce simp: continuous_within_eps_delta)
qed
qed (blast intro: continuous_within_sqrt)

```

6.20.22 Complex arctangent

The branch cut gives standard bounds in the real case.

definition *Arctan* :: *complex* \Rightarrow *complex* **where**
 $Arctan \equiv \lambda z. (i/2) * Ln((1 - i*z) / (1 + i*z))$

lemma *Arctan_def_moebius*: $Arctan\ z = i/2 * Ln\ (moebius\ (-i)\ 1\ i\ 1\ z)$
 by (simp add: Arctan_def moebius_def add_ac)

lemma *Ln_conv_Arctan*:

assumes $z \neq -1$

shows $Ln\ z = -2*i * Arctan\ (moebius\ 1\ (-1)\ (-i)\ (-i)\ z)$

proof –

have $Arctan\ (moebius\ 1\ (-1)\ (-i)\ (-i)\ z) =$
 $i/2 * Ln\ (moebius\ (-i)\ 1\ i\ 1\ (moebius\ 1\ (-1)\ (-i)\ (-i)\ z))$

by (simp add: Arctan_def moebius)

also from *assms* have $i * z \neq i * (-1)$ by (subst mult_left_cancel) simp

hence $i * z - -i \neq 0$ by (simp add: eq_neg_iff_add_eq_0)

from *moebius_inverse'*[OF *this*, of 1 1]

have $moebius\ (-i)\ 1\ i\ 1\ (moebius\ 1\ (-1)\ (-i)\ (-i)\ z) = z$ by simp

finally show ?thesis by (simp add: field_simps)

qed

lemma *Arctan_0* [simp]: $Arctan\ 0 = 0$

by (simp add: Arctan_def)

lemma *Im_complex_div_lemma*: $Im((1 - i*z) / (1 + i*z)) = 0 \iff Re\ z = 0$

by (auto simp: Im_complex_div_eq_0 algebra_simps)

lemma *Re_complex_div_lemma*: $0 < Re((1 - i*z) / (1 + i*z)) \iff norm\ z < 1$

by (simp add: Re_complex_div_gt_0 algebra_simps cmod_def power2_eq_square)

lemma *tan_Arctan*:

assumes $z^2 \neq -1$

```

shows [simp]:  $\tan(\text{Arctan } z) = z$ 
proof -
  obtain  $1 + i*z \neq 0$   $1 - i*z \neq 0$ 
    by (metis add_diff_cancel_left' assms diff_0 i_times_eq_iff mult_cancel_left2
power2_i power2_minus right_minus_eq)
  then show ?thesis
    by (simp add: Arctan_def tan_def sin_exp_eq cos_exp_eq exp_minus divide_simps
flip: csqrt_exp_Ln power2_eq_square)
qed

lemma Arctan_tan [simp]:
  assumes  $|\text{Re } z| < \pi/2$ 
  shows  $\text{Arctan}(\tan z) = z$ 
proof -
  have  $\text{Ln}((1 - i * \tan z) / (1 + i * \tan z)) = 2 * z / i$ 
  proof (rule Ln_unique)
    have  $ge\_pi2: \bigwedge n::\text{int}. |\text{of\_int } (2*n + 1) * \pi/2| \geq \pi/2$ 
      by (case_tac n rule: int_cases) (auto simp: abs_mult)
    have  $\exp(i*z)*\exp(i*z) = -1 \iff \exp(2*i*z) = -1$ 
      by (metis distrib_right exp_add mult_2)
    also have  $\dots \iff \exp(2*i*z) = \exp(i*\pi)$ 
      using cis_conv_exp cis_pi by auto
    also have  $\dots \iff \exp(2*i*z - i*\pi) = 1$ 
      by (metis (no_types) diff_add_cancel diff_minus_eq_add exp_add exp_minus_inverse
mult commute)
    also have  $\dots \iff \text{Re}(i*2*z - i*\pi) = 0 \wedge (\exists n::\text{int}. \text{Im}(i*2*z - i*\pi) =$ 
 $\text{of\_int } (2 * n) * \pi)$ 
      by (simp add: exp_eq_1)
    also have  $\dots \iff \text{Im } z = 0 \wedge (\exists n::\text{int}. 2 * \text{Re } z = \text{of\_int } (2*n + 1) * \pi)$ 
      by (simp add: algebra_simps)
    also have  $\dots \iff \text{False}$ 
      using assms ge_pi2
    by (metis eq_divide_eq linorder_not_less mult commute zero_neq numeral)
  finally have  $\exp(i*z)*\exp(i*z) + 1 \neq 0$ 
    by (auto simp: add commute minus_unique)
  then show  $\exp(2 * z / i) = (1 - i * \tan z) / (1 + i * \tan z)$ 
    apply (simp add: tan_def sin_exp_eq cos_exp_eq exp_minus divide_simps)
    by (simp add: algebra_simps flip: power2_eq_square exp_double)
qed (use assms in auto)
then show ?thesis
  by (auto simp: Arctan_def)
qed

```

```

lemma
  assumes  $\text{Re } z = 0 \implies |\text{Im } z| < 1$ 
  shows  $\text{Re\_Arctan\_bounds}: |\text{Re}(\text{Arctan } z)| < \pi/2$ 
  and  $\text{has\_field\_derivative\_Arctan}: (\text{Arctan has\_field\_derivative inverse}(1 + z^2))$  (at  $z$ )

```

```

proof –
  have nz0:  $1 + i*z \neq 0$ 
    using assms
    by (metis abs_one add_diff_cancel_left' complex_i_mult_minus diff_0 i_squared
imaginary_unit.simps
      less_asym neg_equal_iff_equal)
  have z  $\neq -i$  using assms
    by auto
  then have zz:  $1 + z * z \neq 0$ 
    by (metis abs_one assms i_squared imaginary_unit.simps less_irrefl mi-
nus_unique square_eq_iff)
  have nz1:  $1 - i*z \neq 0$ 
    using assms by (force simp add: i_times_eq_iff)
  have nz2:  $\text{inverse}(1 + i*z) \neq 0$ 
    using assms
    by (metis Im_complex_div_lemma Re_complex_div_lemma cmod_eq_Im di-
vide_complex_def
      less_irrefl mult_zero_right zero_complex.simps(1) zero_complex.simps(2))
  have nzi:  $((1 - i*z) * \text{inverse}(1 + i*z)) \neq 0$ 
    using nz1 nz2 by auto
  have  $\text{Im}((1 - i*z) / (1 + i*z)) = 0 \implies 0 < \text{Re}((1 - i*z) / (1 + i*z))$ 
    by (simp add: Im_complex_div_lemma Re_complex_div_lemma assms cmod_eq_Im)
  then have *:  $((1 - i*z) / (1 + i*z)) \notin \mathbb{R}_{\leq 0}$ 
    by (auto simp add: complex_nonpos_Reals_iff)
  show  $|\text{Re}(\text{Arctan } z)| < \pi/2$ 
    unfolding Arctan_def divide_complex_def
    using mpi_less_Im_Ln [OF nzi]
    by (auto simp: abs_if intro!: Im_Ln_less_pi * [unfolded divide_complex_def])
  show (Arctan has_field_derivative inverse(1 + z^2)) (at z)
    unfolding Arctan_def scaleR_conv_of_real
    apply (intro derivative_eq_intros | simp add: nz0 *)
    using nz1 zz
    apply (simp add: field_split_simps power2_eq_square)
    apply algebra
  done

```

qed

lemma *field_differentiable_at_Arctan*: $(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{Arctan}$
field_differentiable at z

```

using has_field_derivative_Arctan
by (auto simp: field_differentiable_def)

```

lemma *field_differentiable_within_Arctan*:

```

 $(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{Arctan field\_differentiable (at } z \text{ within } s)$ 
using field_differentiable_at_Arctan field_differentiable_at_within by blast

```

declare *has_field_derivative_Arctan [derivative_intros]*

declare *has_field_derivative_Arctan [THEN DERIV_chain2, derivative_intros]*

lemma *continuous_at_Arctan*:

$(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{continuous (at } z) \text{ Arctan}$

by (*simp add: field_differentiable_imp_continuous_at field_differentiable_within_Arctan*)

lemma *continuous_within_Arctan*:

$(\text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{continuous (at } z \text{ within } s) \text{ Arctan}$

using *continuous_at_Arctan continuous_at_imp_continuous_within* **by** *blast*

lemma *continuous_on_Arctan* [*continuous_intros*]:

$(\bigwedge z. z \in s \implies \text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{continuous_on } s \text{ Arctan}$

by (*auto simp: continuous_at_imp_continuous_on continuous_within_Arctan*)

lemma *holomorphic_on_Arctan*:

$(\bigwedge z. z \in s \implies \text{Re } z = 0 \implies |\text{Im } z| < 1) \implies \text{Arctan holomorphic_on } s$

by (*simp add: field_differentiable_within_Arctan holomorphic_on_def*)

theorem *Arctan_series*:

assumes *z: norm (z :: complex) < 1*

defines *g* $\equiv \lambda n. \text{if odd } n \text{ then } -i * i^n / n \text{ else } 0$

defines *h* $\equiv \lambda z n. (-1)^n / \text{of_nat } (2*n+1) * (z::\text{complex})^{(2*n+1)}$

shows $(\lambda n. g \ n * z^n)$ *sums* *Arctan z*

and *h z sums Arctan z*

proof –

define *G* **where** [*abs_def*]: $G \ z = (\sum n. g \ n * z^n)$ **for** *z*

have *summable: summable* $(\lambda n. g \ n * u^n)$ **if** *norm u < 1* **for** *u*

proof (*cases u = 0*)

case *False*

have $(\lambda n. \text{ereal (norm (h u n) / norm (h u (Suc n)))}) = (\lambda n. \text{ereal (inverse (norm u)^2) *}$

$\text{ereal ((2 + inverse (real (Suc n))) / (2 - inverse (real (Suc n))))})$

proof

fix *n*

have $\text{ereal (norm (h u n) / norm (h u (Suc n)))} =$

$\text{ereal (inverse (norm u)^2) * eral (((2*Suc n+1) / (Suc n)) /$

$((2*Suc n-1) / (Suc n)))$

by (*simp add: h_def norm_mult norm_power norm_divide field_split_simps power2_eq_square eval_nat_numeral del: of_nat_add of_nat_Suc*)

also have $\text{of_nat } (2*Suc n+1) / \text{of_nat } (Suc n) = (2::\text{real}) + \text{inverse (real (Suc n))}$

by (*auto simp: field_split_simps simp del: of_nat_Suc simp_all?*)

also have $\text{of_nat } (2*Suc n-1) / \text{of_nat } (Suc n) = (2::\text{real}) - \text{inverse (real (Suc n))}$

by (*auto simp: field_split_simps simp del: of_nat_Suc simp_all?*)

finally show $\text{ereal (norm (h u n) / norm (h u (Suc n)))} = \text{ereal (inverse (norm u)^2) *}$

$\text{ereal ((2 + inverse (real (Suc n))) / (2 - inverse (real (Suc n))))}$.

qed

also have ... $\longrightarrow \text{ereal (inverse (norm u)^2) * eral ((2 + 0) / (2 - 0))}$

by (*intro tendsto_intros LIMSEQ_inverse_real_of_nat simp_all*)

2010

```

finally have  $\liminf (\lambda n. \text{ereal } (\text{cmod } (h \ u \ n) / \text{cmod } (h \ u \ (\text{Suc } n)))) = \text{inverse } (\text{norm } u)^2$ 
by (intro lim_imp_Liminf) simp_all
moreover from power_strict_mono[OF that, of 2] False have  $\text{inverse } (\text{norm } u)^2 > 1$ 
by (simp add: field_split_simps)
ultimately have  $A: \liminf (\lambda n. \text{ereal } (\text{cmod } (h \ u \ n) / \text{cmod } (h \ u \ (\text{Suc } n)))) > 1$  by simp
from False have summable ( $h \ u$ )
by (intro summable_norm_cancel[OF ratio_test_convergence[OF A]])
  (auto simp: h_def norm_divide norm_mult norm_power simp del: of_nat_Suc intro!: mult_pos_pos divide_pos_pos always_eventually)
thus summable ( $\lambda n. g \ n * u^n$ )
by (subst summable_mono_reindex[of  $\lambda n. 2*n+1$ , symmetric])
  (auto simp: power_mult strict_mono_def g_def h_def elim!: oddE)
qed (simp add: h_def)

have  $\exists c. \forall u \in \text{ball } 0 \ 1. \text{Arctan } u - G \ u = c$ 
proof (rule has_field_derivative_zero_constant)
  fix  $u :: \text{complex}$  assume  $u \in \text{ball } 0 \ 1$ 
  hence  $u: \text{norm } u < 1$  by (simp)
  define  $K$  where  $K = (\text{norm } u + 1) / 2$ 
  from  $u$  and abs_Im_le_cmod[of  $u$ ] have  $\text{Im } u: |\text{Im } u| < 1$  by linarith
  from  $u$  have  $K: 0 \leq K \ \text{norm } u < K \ K < 1$  by (simp_all add: K_def)
  hence ( $G$  has_field_derivative ( $\sum n. \text{diffs } g \ n * u^n$ )) (at  $u$ ) unfolding
G_def
    by (intro termdiffs_strong[of of_real  $K$ ] summable) simp_all
  also have ( $\lambda n. \text{diffs } g \ n * u^n$ ) = ( $\lambda n. \text{if even } n \text{ then } (i*u)^n \text{ else } 0$ )
  by (intro ext) (simp_all del: of_nat_Suc add: g_def diffs_def power_mult_distrib)
  also have  $\text{suminf } \dots = (\sum n. -(u^2)^n)$ 
    by (subst suminf_mono_reindex[of  $\lambda n. 2*n$ , symmetric])
    (auto elim!: evenE simp: strict_mono_def power_mult power_mult_distrib)
  also from  $u$  have  $\text{norm } u^2 < 1^2$  by (intro power_strict_mono) simp_all
  hence ( $\sum n. -(u^2)^n$ ) =  $\text{inverse } (1 + u^2)$ 
    by (subst suminf_geometric) (simp_all add: norm_power inverse_eq_divide)
  finally have ( $G$  has_field_derivative  $\text{inverse } (1 + u^2)$ ) (at  $u$ ) .
  from DERIV_diff[OF has_field_derivative_Arctan this] Im_u u
    show ( $\lambda u. \text{Arctan } u - G \ u$ ) has_field_derivative 0 (at  $u$  within ball 0 1)
    by (simp_all add: at_within_open[OF open_ball])
qed simp_all
then obtain  $c$  where  $c: \bigwedge u. \text{norm } u < 1 \implies \text{Arctan } u - G \ u = c$  by auto
from this[of 0] have  $c = 0$  by (simp add: G_def g_def)
with  $c \ z$  have  $\text{Arctan } z = G \ z$  by simp
with summable[OF  $z$ ] show ( $\lambda n. g \ n * z^n$ ) sums  $\text{Arctan } z$  unfolding G_def
by (simp add: sums_iff)
  thus  $h \ z$  sums  $\text{Arctan } z$  by (subst (asm) sums_mono_reindex[of  $\lambda n. 2*n+1$ , symmetric])
  (auto elim!: oddE simp: strict_mono_def power_mult g_def h_def)

```

qed

A quickly-converging series for the logarithm, based on the arctangent.

theorem *ln_series_quadratic*:

assumes $x: x > (0::real)$

shows $(\lambda n. (2*((x - 1) / (x + 1)) ^{(2*n+1)} / of_nat (2*n+1))) \text{ sums } \ln x$

proof –

define $y :: complex$ **where** $y = of_real ((x-1)/(x+1))$

from x **have** $x': complex_of_real\ x \neq of_real\ (-1)$ **by** $(subst\ of_real_eq_iff)$

auto

from x **have** $|x - 1| < |x + 1|$ **by** *linarith*

hence $norm\ (complex_of_real\ (x - 1) / complex_of_real\ (x + 1)) < 1$

by $(simp\ add:\ norm_divide\ del:\ of_real_add\ of_real_diff)$

hence $norm\ (i * y) < 1$ **unfolding** y_def **by** $(subst\ norm_mult)\ simp$

hence $(\lambda n. (-2*i) * ((-1)^n / of_nat (2*n+1) * (i*y)^{(2*n+1)})) \text{ sums } ((-2*i) * Arctan\ (i*y))$

by $(intro\ Arctan_series\ sums_mult)\ simp_all$

also **have** $(\lambda n. (-2*i) * ((-1)^n / of_nat (2*n+1) * (i*y)^{(2*n+1)})) = (\lambda n. (-2*i) * ((-1)^n * (i*y*(-y^2))^n / of_nat (2*n+1)))$

by $(intro\ ext)\ (simp_all\ add:\ power_mult\ power_mult_distrib)$

also **have** $\dots = (\lambda n. 2*y * ((-1) * (-y^2))^n / of_nat (2*n+1))$

by $(intro\ ext,\ subst\ power_mult_distrib)\ (simp\ add:\ algebra_simps\ power_mult)$

also **have** $\dots = (\lambda n. 2*y^{(2*n+1)} / of_nat (2*n+1))$

by $(subst\ power_add,\ subst\ power_mult)\ (simp\ add:\ mult_ac)$

also **have** $\dots = (\lambda n. of_real\ (2*((x-1)/(x+1))^{(2*n+1)} / of_nat (2*n+1)))$

by $(intro\ ext)\ (simp\ add:\ y_def)$

also **have** $i * y = (of_real\ x - 1) / (-i * (of_real\ x + 1))$

by $(subst\ divide_divide_eq_left\ [symmetric])\ (simp\ add:\ y_def)$

also **have** $\dots = moebius\ 1\ (-1)\ (-i)\ (-i)\ (of_real\ x)$ **by** $(simp\ add:\ moebius_def\ algebra_simps)$

also **from** x **have** $-2*i*Arctan\ \dots = Ln\ (of_real\ x)$ **by** $(intro\ Ln_conv_Arctan\ [symmetric])\ simp_all$

also **from** x **have** $\dots = \ln\ x$ **by** $(rule\ Ln_of_real)$

finally **show** *?thesis* **by** $(subst\ (asm)\ sums_of_real_iff)$

qed

6.20.23 Real arctangent

lemma *Im_Arctan_of_real [simp]*: $Im\ (Arctan\ (of_real\ x)) = 0$

proof –

have $ne: 1 + x^2 \neq 0$

by $(metis\ power_one\ sum_power2_eq_zero_iff\ zero_neq_one)$

have $ne1: 1 + i * complex_of_real\ x \neq 0$

using *Complex_eq_complex_eq_cancel_iff2* **by** *fastforce*

have $Re\ (Ln\ ((1 - i * x) * inverse\ (1 + i * x))) = 0$

apply $(rule\ norm_exp_imaginary)$

using ne

apply $(simp\ add:\ ne1\ cmod_def)$

apply $(auto\ simp:\ field_split_simps)$

2012

```
    apply algebra
  done
  then show ?thesis
    unfolding Arctan_def divide_complex_def by (simp add: complex_eq_iff)
  qed

lemma arctan_eq_Re_Arctan: arctan x = Re (Arctan (of_real x))
proof (rule arctan_unique)
  have  $(1 - i * x) / (1 + i * x) \notin \mathbb{R}_{\leq 0}$ 
    by (auto simp: Im_complex_div_lemma complex_nonpos_Reals_iff)
  then show  $-(\pi / 2) < \text{Re} (\text{Arctan} (\text{complex\_of\_real } x))$ 
    by (simp add: Arctan_def Im_Ln_less_pi)
next
  have *:  $(1 - i*x) / (1 + i*x) \neq 0$ 
    by (simp add: field_split_simps) (simp add: complex_eq_iff)
  show  $\text{Re} (\text{Arctan} (\text{complex\_of\_real } x)) < \pi / 2$ 
    using mpi_less_Im_Ln [OF *]
    by (simp add: Arctan_def)
next
  have  $\tan (\text{Re} (\text{Arctan} (\text{of\_real } x))) = \text{Re} (\tan (\text{Arctan} (\text{of\_real } x)))$ 
    by (metis Im_Arctan_of_real Re_complex_of_real complex_is_Real_iff of_real_Re
tan_of_real)
  also have  $\dots = x$ 
  proof -
    have  $(\text{complex\_of\_real } x)^2 \neq -1$ 
      by (smt (verit, best) Im_complex_of_real imaginary_unit.sel(2) of_real_minus
power2_eq_iff power2_i)
    then show ?thesis
      by simp
  qed
  finally show  $\tan (\text{Re} (\text{Arctan} (\text{complex\_of\_real } x))) = x$  .
qed
```

```
lemma Arctan_of_real: Arctan (of_real x) = of_real (arctan x)
  unfolding arctan_eq_Re_Arctan divide_complex_def
  by (simp add: complex_eq_iff)
```

```
lemma Arctan_in_Reals [simp]:  $z \in \mathbb{R} \implies \text{Arctan } z \in \mathbb{R}$ 
  by (metis Reals_cases Reals_of_real Arctan_of_real)
```

```
declare arctan_one [simp]
```

```
lemma arctan_less_pi4_pos:  $x < 1 \implies \text{arctan } x < \pi/4$ 
  by (metis arctan_less_iff arctan_one)
```

```
lemma arctan_less_pi4_neg:  $-1 < x \implies -(\pi/4) < \text{arctan } x$ 
  by (metis arctan_less_iff arctan_minus arctan_one)
```

```
lemma arctan_less_pi4:  $|x| < 1 \implies |\text{arctan } x| < \pi/4$ 
```


by (metis abs_less_iff arctan_less_pi4_pos arctan_minus)

lemma arctan_le_pi4: $|x| \leq 1 \implies |\arctan x| \leq \pi/4$
 by (metis abs_le_iff arctan_le_iff arctan_minus arctan_one)

lemma abs_arctan: $|\arctan x| = \arctan |x|$
 by (simp add: abs_if arctan_minus)

lemma arctan_add_raw:
 assumes $|\arctan x + \arctan y| < \pi/2$
 shows $\arctan x + \arctan y = \arctan((x + y) / (1 - x * y))$
proof (rule arctan_unique [symmetric])
 show 12: $-(\pi / 2) < \arctan x + \arctan y$ $\arctan x + \arctan y < \pi / 2$
 using assms by linarith+
 show $\tan(\arctan x + \arctan y) = (x + y) / (1 - x * y)$
 using cos_gt_zero_pi [OF 12] by (simp add: arctan_tan_add)
qed

lemma arctan_inverse:
 $0 < x \implies \arctan(\text{inverse } x) = \pi/2 - \arctan x$
 by (smt (verit, del_insts) arctan arctan_unique tan_cot zero_less_arctan_iff)

lemma arctan_add_small:
 assumes $|x * y| < 1$
 shows $(\arctan x + \arctan y = \arctan((x + y) / (1 - x * y)))$
proof (cases $x = 0 \vee y = 0$)
 case False
 with assms have $|x| < \text{inverse } |y|$
 by (simp add: field_split_simps abs_mult)
 with False have $|\arctan x| < \pi / 2 - |\arctan y|$ using assms
 by (auto simp add: abs_arctan arctan_inverse [symmetric] arctan_less_iff)
 then show ?thesis
 by (intro arctan_add_raw) linarith
qed auto

lemma abs_arctan_le:
 fixes $x::\text{real}$ shows $|\arctan x| \leq |x|$
proof -
 have 1: $\bigwedge x. x \in \mathbb{R} \implies \text{cmod}(\text{inverse}(1 + x^2)) \leq 1$
 by (simp add: norm_divide divide_simps in_Reals_norm complex_is_Real_iff power2_eq_square)
 have $\text{cmod}(\text{Arctan } w - \text{Arctan } z) \leq 1 * \text{cmod}(w - z)$ if $w \in \mathbb{R} z \in \mathbb{R}$ for $w z$
 apply (rule field_differentiable_bound [OF convex_Reals, of Arctan _ 1])
 apply (rule has_field_derivative_at_within [OF has_field_derivative_Arctan])
 using 1 that by (auto simp: Reals_def)
 then have $\text{cmod}(\text{Arctan}(\text{of_real } x) - \text{Arctan } 0) \leq 1 * \text{cmod}(\text{of_real } x - 0)$
 using Reals_0 Reals_of_real by blast
 then show ?thesis
 by (simp add: Arctan_of_real)

2014

qed

lemma *arctan_le_self*: $0 \leq x \implies \arctan x \leq x$
by (*metis abs_arctan_le abs_of_nonneg zero_le_arctan_iff*)

lemma *abs_tan_ge*: $|x| < \pi/2 \implies |x| \leq |\tan x|$
by (*metis abs_arctan_le abs_less_iff arctan_tan_minus_less_iff*)

lemma *arctan_bounds*:
assumes $0 \leq x < 1$
shows *arctan_lower_bound*:
 $(\sum_{k < 2 * n} (-1)^k * (1 / \text{real } (k * 2 + 1)) * x^{(k * 2 + 1)}) \leq \arctan x$
(is $(\sum_{k < _} _ * ?a k) \leq _$
and *arctan_upper_bound*:
 $\arctan x \leq (\sum_{k < 2 * n + 1} (-1)^k * (1 / \text{real } (k * 2 + 1)) * x^{(k * 2 + 1)})$
proof –
have *tendsto_zero*: $?a \longrightarrow 0$
proof (*rule tendsto_eq_rhs*)
show $(\lambda k. 1 / \text{real } (k * 2 + 1)) * x^{(k * 2 + 1)} \longrightarrow 0 * 0$
using *assms*
by (*intro tendsto_mult real_tendsto_divide_at_top*)
(auto simp: filterlim_sequentially_iff_filterlim_real
intro!: real_tendsto_divide_at_top tendsto_power_zero filterlim_real_sequentially
tendsto_eq_intros filterlim_at_top_mult tendsto_pos filterlim_tendsto_add_at_top)
qed *simp*
have *nonneg*: $0 \leq ?a n$ **for** n
by (*force intro!: divide_nonneg_nonneg mult_nonneg_nonneg zero_le_power*
assms)
have *le*: $?a (Suc n) \leq ?a n$ **for** n
by (*rule mult_mono[OF _ power_decreasing]*) (*auto simp: field_split_simps*
assms less_imp_le)
from *summable_Leibniz'(4)*[*of ?a, OF tendsto_zero nonneg le, of n*]
summable_Leibniz'(2)[*of ?a, OF tendsto_zero nonneg le, of n*]
assms
show $(\sum_{k < 2 * n} (-1)^k * ?a k) \leq \arctan x$ $\arctan x \leq (\sum_{k < 2 * n + 1} (-1)^k * ?a k)$
by (*auto simp: arctan_series*)
qed

6.20.24 Bounds on pi using real arctangent

lemma *pi_machin*: $\pi = 16 * \arctan (1 / 5) - 4 * \arctan (1 / 239)$
using *machin* **by** *simp*

lemma *pi_approx*: $3.141592653588 \leq \pi \leq 3.1415926535899$
unfolding *pi_machin*
using *arctan_bounds*[*of 1/5 4*]
arctan_bounds[*of 1/239 4*]

by (simp_all add: eval_nat_numeral)

lemma pi_gt3: $\pi > 3$
 using pi_approx by simp

6.20.25 Inverse Sine

definition *Arcsin* :: complex \Rightarrow complex **where**

$Arcsin \equiv \lambda z. -i * Ln(i * z + csqrt(1 - z^2))$

lemma *Arcsin_body_lemma*: $i * z + csqrt(1 - z^2) \neq 0$

using power2_sqrt [of $1 - z^2$]

by (metis add.inverse_unique diff_0 diff_add_cancel mult.left_commute mult_minus1_right power2_i power2_minus power_mult_distrib zero_neq_one)

lemma *Arcsin_range_lemma*: $|Re\ z| < 1 \implies 0 < Re(i * z + csqrt(1 - z^2))$

using Complex.cmod_power2 [of z , symmetric]

by (simp add: real_less_sqrt algebra_simps Re_power2 cmod_square_less_1_plus)

lemma *Re_Arcsin*: $Re(Arcsin\ z) = Im(Ln(i * z + csqrt(1 - z^2)))$

by (simp add: *Arcsin_def*)

lemma *Im_Arcsin*: $Im(Arcsin\ z) = -ln(cmod(i * z + csqrt(1 - z^2)))$

by (simp add: *Arcsin_def Arcsin_body_lemma*)

lemma *one_minus_z2_notin_nonpos_Reals*:

assumes $Im\ z = 0 \implies |Re\ z| < 1$

shows $1 - z^2 \notin \mathbf{R}_{\leq 0}$

proof (cases $Im\ z = 0$)

case True

with *assms* show ?thesis

by (simp add: complex_nonpos_Reals_iff flip: abs_square_less_1)

next

case False

have $\neg (Im\ z)^2 \leq -1$

using False_power2_less_eq_zero_iff by fastforce

with False show ?thesis

by (auto simp add: complex_nonpos_Reals_iff Re_power2 Im_power2)

qed

lemma *isCont_Arcsin_lemma*:

assumes $le0: Re(i * z + csqrt(1 - z^2)) \leq 0$ and $(Im\ z = 0 \implies |Re\ z| < 1)$

shows False

proof (cases $Im\ z = 0$)

case True

then show ?thesis

using *assms* by (fastforce simp: cmod_def abs_square_less_1 [symmetric])

next

case False

2016

```

have leim: (cmod (1 - z2) + (1 - Re (z2))) / 2 ≤ (Im z)2
  using le0 sqrt_le_D by fastforce
have neq: (cmod z)2 ≠ 1 + cmod (1 - z2)
proof (clarsimp simp add: cmod_def)
  assume (Re z)2 + (Im z)2 = 1 + sqrt ((1 - Re (z2))2 + (Im (z2))2)
  then have ((Re z)2 + (Im z)2 - 1)2 = ((1 - Re (z2))2 + (Im (z2))2)
    by simp
  then show False using False
    by (simp add: power2_eq_square algebra_simps)
qed
moreover have 2: (Im z)2 = (1 + ((Im z)2 + cmod (1 - z2)) - (Re z)2) / 2
  using leim cmod_power2 [of z] norm_triangle_ineq2 [of z2 1]
  by (simp add: norm_power Re_power2 norm_minus_commute [of 1])
ultimately show False
  by (simp add: Re_power2 Im_power2 cmod_power2)
qed

```

```

lemma isCont_Arcsin:
  assumes (Im z = 0 ⇒ |Re z| < 1)
  shows isCont Arcsin z
proof -
  have 1: i * z + csqrt (1 - z2) ∉ ℝ≤0
    by (metis isCont_Arcsin_lemma assms complex_nonpos_Reals_iff)
  have 2: 1 - z2 ∉ ℝ≤0
    by (simp add: one_minus_z2_notin_nonpos_Reals assms)
  show ?thesis
    using assms unfolding Arcsin_def by (intro isCont_Ln' isCont_csqrt' continuous_intros 1 2)
qed

```

```

lemma isCont_Arcsin' [simp]:
  shows isCont f z ⇒ (Im (f z) = 0 ⇒ |Re (f z)| < 1) ⇒ isCont (λx. Arcsin
(f x)) z
  by (blast intro: isCont_o2 [OF isCont_Arcsin])

```

```

lemma sin_Arcsin [simp]: sin(Arcsin z) = z
proof -
  have i*z*2 + csqrt (1 - z2)*2 = 0 ↔ (i*z)*2 + csqrt (1 - z2)*2 = 0
    by (simp add: algebra_simps) — Cancelling a factor of 2
  moreover have ... ↔ (i*z) + csqrt (1 - z2) = 0
    by (metis Arcsin_body_lemma distrib_right no_zero_divisors zero_neq_numeral)
  ultimately show ?thesis
    apply (simp add: sin_exp_eq Arcsin_def Arcsin_body_lemma exp_minus divide_simps)
    apply (simp add: algebra_simps)
    apply (simp add: right_diff_distrib flip: power2_eq_square)
    done
qed

```

lemma *Re_eq_pihalf_lemma*:

$|Re\ z| = \pi/2 \implies Im\ z = 0 \implies$
 $Re\ ((exp\ (i*z) + inverse\ (exp\ (i*z))) / 2) = 0 \wedge 0 \leq Im\ ((exp\ (i*z) + inverse\ (exp\ (i*z))) / 2)$
apply (*simp add: cos_i_times [symmetric] Re_cos Im_cos abs_if del: eq_divide_eq_numeral1*)
by (*metis cos_minus cos_pi_half*)

lemma *Re_less_pihalf_lemma*:

assumes $|Re\ z| < \pi / 2$
shows $0 < Re\ ((exp\ (i*z) + inverse\ (exp\ (i*z))) / 2)$
proof –
have $0 < cos\ (Re\ z)$ **using** *assms*
using *cos_gt_zero_pi* **by** *auto*
then show *?thesis*
by (*simp add: cos_i_times [symmetric] Re_cos Im_cos add_pos_pos*)
qed

lemma *Arcsin_sin*:

assumes $|Re\ z| < \pi/2 \vee (|Re\ z| = \pi/2 \wedge Im\ z = 0)$
shows $Arcsin(sin\ z) = z$
proof –
have $Arcsin(sin\ z) = - (i * Ln\ (csqrt\ (1 - (i * (exp\ (i*z) - inverse\ (exp\ (i*z))))^2 / 4) - (inverse\ (exp\ (i*z)) - exp\ (i*z)) / 2))$
by (*simp add: sin_exp_eq Arcsin_def exp_minus_power_divide*)
also have $\dots = - (i * Ln\ (csqrt\ (((exp\ (i*z) + inverse\ (exp\ (i*z))) / 2)^2) - (inverse\ (exp\ (i*z)) - exp\ (i*z)) / 2))$
by (*simp add: field_simps power2_eq_square*)
also have $\dots = - (i * Ln\ (((exp\ (i*z) + inverse\ (exp\ (i*z))) / 2) - (inverse\ (exp\ (i*z)) - exp\ (i*z)) / 2))$
apply (*subst csqrt_square*)
using *assms Re_eq_pihalf_lemma Re_less_pihalf_lemma* **by** *auto*
also have $\dots = - (i * Ln\ (exp\ (i*z)))$
by (*simp add: field_simps power2_eq_square*)
also have $\dots = z$
using *assms* **by** (*auto simp: abs_if simp del: eq_divide_eq_numeral1 split: if_split_asm*)
finally show *?thesis* .
qed

lemma *Arcsin_unique*:

$\llbracket sin\ z = w; |Re\ z| < \pi/2 \vee (|Re\ z| = \pi/2 \wedge Im\ z = 0) \rrbracket \implies Arcsin\ w = z$
by (*metis Arcsin_sin*)

lemma *Arcsin_0 [simp]*: $Arcsin\ 0 = 0$

by (*simp add: Arcsin_unique*)

lemma *Arcsin_1 [simp]*: $Arcsin\ 1 = \pi/2$

using *Arcsin_unique sin_of_real_pi_half* **by** *fastforce*

2018

lemma *Arcsin_minus_1* [*simp*]: $\text{Arcsin}(-1) = -(\pi/2)$
by (*simp add: Arcsin_unique*)

lemma *has_field_derivative_Arcsin*:
assumes $\text{Im } z = 0 \implies |\text{Re } z| < 1$
shows (*Arcsin has_field_derivative inverse(cos(Arcsin z))*) (at z)
proof –
have $(\sin(\text{Arcsin } z))^2 \neq 1$
using *assms one_minus_z2_notin_nonpos_Reals* **by force**
then have $\cos(\text{Arcsin } z) \neq 0$
by (*metis diff_0_right power_zero_numerical sin_squared_eq*)
then show ?thesis
by (*rule has_field_derivative_inverse_basic [OF DERIV_sin _ _ open_ball [of z 1]]*) (*auto intro: isCont_Arcsin assms*)
qed

declare *has_field_derivative_Arcsin* [*derivative_intros*]
declare *has_field_derivative_Arcsin* [*THEN DERIV_chain2, derivative_intros*]

lemma *field_differentiable_at_Arcsin*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arcsin field_differentiable at } z$
using *field_differentiable_def has_field_derivative_Arcsin* **by blast**

lemma *field_differentiable_within_Arcsin*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arcsin field_differentiable (at } z \text{ within } s)$
using *field_differentiable_at_Arcsin field_differentiable_within_subset* **by blast**

lemma *continuous_within_Arcsin*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous (at } z \text{ within } s) \text{ Arcsin}$
using *continuous_at_imp_continuous_within isCont_Arcsin* **by blast**

lemma *continuous_on_Arcsin* [*continuous_intros*]:
 $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous_on } s \text{ Arcsin}$
by (*simp add: continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arcsin*: $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arcsin holomorphic_on } s$
by (*simp add: field_differentiable_within_Arcsin holomorphic_on_def*)

6.20.26 Inverse Cosine

definition *Arccos* :: *complex* \Rightarrow *complex* **where**
 $\text{Arccos} \equiv \lambda z. -i * \text{Ln}(z + i * \text{csqrt}(1 - z^2))$

lemma *Arccos_range_lemma*: $|\text{Re } z| < 1 \implies 0 < \text{Im}(z + i * \text{csqrt}(1 - z^2))$
using *Arcsin_range_lemma* [*of -z*] **by simp**

lemma *Arccos_body_lemma*: $z + i * \text{csqrt}(1 - z^2) \neq 0$
by (*metis Arcsin_body_lemma complex_i_mult_minus diff_0 diff_eq_eq power2_minus*)

lemma *Re_Arccos*: $\text{Re}(\text{Arccos } z) = \text{Im}(\text{Ln}(z + i * \text{csqrt}(1 - z^2)))$
by (*simp add: Arccos_def*)

lemma *Im_Arccos*: $\text{Im}(\text{Arccos } z) = -\text{ln}(\text{cmod}(z + i * \text{csqrt}(1 - z^2)))$
by (*simp add: Arccos_def Arccos_body_lemma*)

A very tricky argument to find!

lemma *isCont_Arccos_lemma*:
assumes *eq0*: $\text{Im}(z + i * \text{csqrt}(1 - z^2)) = 0$ **and** $\text{Im } z = 0 \implies |\text{Re } z| < 1$
shows *False*
proof (*cases Im z = 0*)
case True
then show *?thesis*
using *assms* **by** (*fastforce simp add: cmod_def abs_square_less_1 [symmetric]*)
next
case False
have *Imz*: $\text{Im } z = -\text{sqrt}((1 + ((\text{Im } z)^2 + \text{cmod}(1 - z^2)) - (\text{Re } z)^2) / 2)$
using *eq0 abs_Re_le_cmod [of 1-z^2]*
by (*simp add: Re_power2 algebra_simps*)
have $(\text{cmod } z)^2 - 1 \neq \text{cmod}(1 - z^2)$
proof (*clarsimp simp add: cmod_def*)
assume $(\text{Re } z)^2 + (\text{Im } z)^2 - 1 = \text{sqrt}((1 - \text{Re}(z^2))^2 + (\text{Im}(z^2))^2)$
then have $((\text{Re } z)^2 + (\text{Im } z)^2 - 1)^2 = ((1 - \text{Re}(z^2))^2 + (\text{Im}(z^2))^2)$
by *simp*
then show *False* **using** *False*
by (*simp add: power2_eq_square algebra_simps*)
qed
moreover have $(\text{Im } z)^2 = (1 + ((\text{Im } z)^2 + \text{cmod}(1 - z^2)) - (\text{Re } z)^2) / 2$
using *abs_Re_le_cmod [of 1-z^2]* **by** (*subst Imz*) (*simp add: Re_power2*)
ultimately show *False*
by (*simp add: cmod_power2*)
qed

lemma *isCont_Arccos*:
assumes $(\text{Im } z = 0 \implies |\text{Re } z| < 1)$
shows *isCont Arccos z*
proof -
have $z + i * \text{csqrt}(1 - z^2) \notin \mathbf{R}_{\leq 0}$
by (*metis complex_nonpos_Reals_iff isCont_Arccos_lemma assms*)
with *assms* **show** *?thesis*
unfolding *Arccos_def*
by (*simp_all add: one_minus_z2_notin_nonpos_Reals assms*)
qed

lemma *isCont_Arccos' [simp]*:
 $\text{isCont } f z \implies (\text{Im}(f z) = 0 \implies |\text{Re}(f z)| < 1) \implies \text{isCont } (\lambda x. \text{Arccos}(f x)) z$
by (*blast intro: isCont_o2 [OF _ isCont_Arccos]*)

2020

lemma *cos_Arccos* [*simp*]: $\cos(\text{Arccos } z) = z$
proof –
have $z^2 + i * (2 * \text{csqrt } (1 - z^2)) = 0 \iff z^2 + i * \text{csqrt } (1 - z^2)^2 = 0$
by (*simp add: algebra_simps*) — Cancelling a factor of 2
moreover have $\dots \iff z + i * \text{csqrt } (1 - z^2) = 0$
by (*metis distrib_right mult_eq_0_iff zero_neq_numeral*)
ultimately show ?thesis
by (*simp add: cos_exp_eq Arccos_def Arccos_body_lemma exp_minus_field_simps*
flip: power2_eq_square)
qed

lemma *Arccos_cos*:
assumes $0 < \text{Re } z \wedge \text{Re } z < \pi \vee$
 $\text{Re } z = 0 \wedge 0 \leq \text{Im } z \vee$
 $\text{Re } z = \pi \wedge \text{Im } z \leq 0$
shows $\text{Arccos}(\cos z) = z$
proof –
have $*$: $((i - (\exp(i * z))^2 * i) / (2 * \exp(i * z))) = \sin z$
by (*simp add: sin_exp_eq exp_minus_field_simps power2_eq_square*)
have $1 - (\exp(i * z) + \text{inverse}(\exp(i * z)))^2 / 4 = ((i - (\exp(i * z))^2 * i) /$
 $(2 * \exp(i * z)))^2$
by (*simp add: field_simps power2_eq_square*)
then have $\text{Arccos}(\cos z) = - (i * \text{Ln}((\exp(i * z) + \text{inverse}(\exp(i * z))) / 2$
 $+$
 $i * \text{csqrt}(((i - (\exp(i * z))^2 * i) / (2 * \exp(i * z)))^2)))$
by (*simp add: cos_exp_eq Arccos_def exp_minus_power_divide*)
also have $\dots = - (i * \text{Ln}((\exp(i * z) + \text{inverse}(\exp(i * z))) / 2 +$
 $i * ((i - (\exp(i * z))^2 * i) / (2 * \exp(i * z))))$
apply (*subst csqrt_square*)
using *assms Re_sin_pos* [*of z*] *Im_sin_nonneg* [*of z*] *Im_sin_nonneg2* [*of z*]
by (*auto simp: * Re_sin Im_sin*)
also have $\dots = - (i * \text{Ln}(\exp(i * z)))$
by (*simp add: field_simps power2_eq_square*)
also have $\dots = z$
using *assms*
by (*subst Complex_Transcendental.Ln_exp, auto*)
finally show ?thesis .
qed

lemma *Arccos_unique*:
 $\llbracket \cos z = w;$
 $0 < \text{Re } z \wedge \text{Re } z < \pi \vee$
 $\text{Re } z = 0 \wedge 0 \leq \text{Im } z \vee$
 $\text{Re } z = \pi \wedge \text{Im } z \leq 0 \rrbracket \implies \text{Arccos } w = z$
using *Arccos_cos* **by** *blast*

lemma *Arccos_0* [*simp*]: $\text{Arccos } 0 = \pi/2$
by (*rule Arccos_unique*) *auto*

lemma *Arccos_1* [*simp*]: $\text{Arccos } 1 = 0$
by (rule *Arccos_unique*) *auto*

lemma *Arccos_minus1*: $\text{Arccos}(-1) = \pi$
by (rule *Arccos_unique*) *auto*

lemma *has_field_derivative_Arccos*:
assumes ($\text{Im } z = 0 \implies |\text{Re } z| < 1$)
shows ($\text{Arccos has_field_derivative } - \text{inverse}(\sin(\text{Arccos } z))$) (at z)
proof –
have $x^2 \neq -1$ **for** $x::\text{real}$
by (sos (($R < 1 + ((\sim 1) * A = 0) + (R < 1 * (R < 1 * [x]^{2}))$))))
with *assms* **have** $(\cos(\text{Arccos } z))^2 \neq 1$
by (auto *simp*: *complex_eq_iff_Re_power2_Im_power2_abs_square_eq_1*)
then **have** $-\sin(\text{Arccos } z) \neq 0$
by (*metis* *cos_squared_eq_diff_0_right_mult_zero_left_neg_0_equal_iff_equal_power2_eq_square*)
then **have** ($\text{Arccos has_field_derivative } \text{inverse}(-\sin(\text{Arccos } z))$) (at z)
by (rule *has_field_derivative_inverse_basic* [*OF* *DERIV_cos* *open_ball* [*of z 1*]])
(auto intro: isCont_Arccos assms)
then **show** *?thesis*
by *simp*
qed

declare *has_field_derivative_Arcsin* [*derivative_intros*]
declare *has_field_derivative_Arcsin* [*THEN* *DERIV_chain2*, *derivative_intros*]

lemma *field_differentiable_at_Arccos*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arccos field_differentiable at } z$
using *field_differentiable_def* *has_field_derivative_Arccos* **by** *blast*

lemma *field_differentiable_within_Arccos*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arccos field_differentiable (at } z \text{ within } s)$
using *field_differentiable_at_Arccos* *field_differentiable_within_subset* **by** *blast*

lemma *continuous_within_Arccos*:
 $(\text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous (at } z \text{ within } s) \text{ Arccos}$
using *continuous_at_imp_continuous_within* *isCont_Arccos* **by** *blast*

lemma *continuous_on_Arccos* [*continuous_intros*]:
 $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{continuous_on } s \text{ Arccos}$
by (*simp* *add*: *continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arccos*: $(\bigwedge z. z \in s \implies \text{Im } z = 0 \implies |\text{Re } z| < 1) \implies \text{Arccos holomorphic_on } s$
by (*simp* *add*: *field_differentiable_within_Arccos holomorphic_on_def*)

2022

6.20.27 Upper and Lower Bounds for Inverse Sine and Cosine

lemma *Arccsin_bounds*: $|Re\ z| < 1 \implies |Re(Arccsin\ z)| < \pi/2$
unfolding *Re_Arccsin*
by (*blast intro: Re_Ln_pos_lt_imp Arccsin_range_lemma*)

lemma *Arccos_bounds*: $|Re\ z| < 1 \implies 0 < Re(Arccos\ z) \wedge Re(Arccos\ z) < \pi$
unfolding *Re_Arccos*
by (*blast intro!: Im_Ln_pos_lt_imp Arccos_range_lemma*)

lemma *Re_Arccos_bounds*: $-\pi < Re(Arccos\ z) \wedge Re(Arccos\ z) \leq \pi$
unfolding *Re_Arccos*
by (*blast intro!: mpi_less_Im_Ln Im_Ln_le_pi Arccos_body_lemma*)

lemma *Re_Arccos_bound*: $|Re(Arccos\ z)| \leq \pi$
by (*meson Re_Arccos_bounds abs_le_iff less_eq_real_def minus_less_iff*)

lemma *Im_Arccos_bound*: $|Im(Arccos\ w)| \leq cmod\ w$

proof –

have $(Im(Arccos\ w))^2 \leq (cmod(\cos(Arccos\ w)))^2 - (\cos(Re(Arccos\ w)))^2$
using *norm_cos_squared [of Arccos w] real_le_abs_sinh [of Im(Arccos w)]*

by (*simp only: abs_le_square_iff*) (*simp add: field_split_simps*)

also have $\dots \leq (cmod\ w)^2$

by (*auto simp: cmod_power2*)

finally show *?thesis*

using *abs_le_square_iff* **by force**

qed

lemma *Re_Arccsin_bounds*: $-\pi < Re(Arccsin\ z) \wedge Re(Arccsin\ z) \leq \pi$
unfolding *Re_Arccsin*
by (*blast intro!: mpi_less_Im_Ln Im_Ln_le_pi Arccsin_body_lemma*)

lemma *Re_Arccsin_bound*: $|Re(Arccsin\ z)| \leq \pi$
by (*meson Re_Arccsin_bounds abs_le_iff less_eq_real_def minus_less_iff*)

lemma *norm_Arccos_bounded*:

fixes $w :: \text{complex}$

shows $norm(Arccos\ w) \leq \pi + norm\ w$

proof –

have $Re(Arccos\ w)^2 \leq \pi^2 - (Im(Arccos\ w))^2 \leq (cmod\ w)^2$

using *Re_Arccos_bound [of w] Im_Arccos_bound [of w] abs_le_square_iff* **by force+**

have $Arccos\ w \cdot Arccos\ w \leq \pi^2 + (cmod\ w)^2$

using *Re* **by** (*simp add: dot_square_norm cmod_power2 [of Arccos w]*)

then have $cmod(Arccos\ w) \leq \pi + cmod(\cos(Arccos\ w))$

by (*smt (verit) Im_Arccos_bound Re_Arccos_bound cmod_le_cos_Arccos*)

then show $cmod(Arccos\ w) \leq \pi + cmod\ w$

by auto

qed

6.20.28 Interrelations between Arcsin and Arccos

lemma *cos_Arcsin_nonzero*: $z^2 \neq 1 \implies \cos(\text{Arcsin } z) \neq 0$
by (*metis diff_0_right power_zero_numeral sin_Arcsin sin_squared_eq*)

lemma *sin_Arccos_nonzero*: $z^2 \neq 1 \implies \sin(\text{Arccos } z) \neq 0$
by (*metis add.right_neutral cos_Arccos power2_eq_square power_zero_numeral sin_cos_squared_add3*)

lemma *cos_sin_csqrt*:
assumes $0 < \cos(\text{Re } z) \vee \cos(\text{Re } z) = 0 \wedge \text{Im } z * \sin(\text{Re } z) \leq 0$
shows $\cos z = \text{csqrt}(1 - (\sin z)^2)$
proof (*rule csqrt_unique [THEN sym]*)
show $(\cos z)^2 = 1 - (\sin z)^2$
by (*simp add: cos_squared_eq*)
qed (*use assms in <auto simp: Re_cos Im_cos add_pos_pos mult_le_0_iff zero_le_mult_iff>*)

lemma *sin_cos_csqrt*:
assumes $0 < \sin(\text{Re } z) \vee \sin(\text{Re } z) = 0 \wedge 0 \leq \text{Im } z * \cos(\text{Re } z)$
shows $\sin z = \text{csqrt}(1 - (\cos z)^2)$
proof (*rule csqrt_unique [THEN sym]*)
show $(\sin z)^2 = 1 - (\cos z)^2$
by (*simp add: sin_squared_eq*)
qed (*use assms in <auto simp: Re_sin Im_sin add_pos_pos mult_le_0_iff zero_le_mult_iff>*)

lemma *Arcsin_Arccos_csqrt_pos*:
 $(0 < \text{Re } z \vee \text{Re } z = 0 \wedge 0 \leq \text{Im } z) \implies \text{Arcsin } z = \text{Arccos}(\text{csqrt}(1 - z^2))$
by (*simp add: Arcsin_def Arccos_def Complex.csqrt_square add commute*)

lemma *Arccos_Arcsin_csqrt_pos*:
 $(0 < \text{Re } z \vee \text{Re } z = 0 \wedge 0 \leq \text{Im } z) \implies \text{Arccos } z = \text{Arcsin}(\text{csqrt}(1 - z^2))$
by (*simp add: Arcsin_def Arccos_def Complex.csqrt_square add commute*)

lemma *sin_Arccos*:
 $0 < \text{Re } z \vee \text{Re } z = 0 \wedge 0 \leq \text{Im } z \implies \sin(\text{Arccos } z) = \text{csqrt}(1 - z^2)$
by (*simp add: Arccos_Arcsin_csqrt_pos*)

lemma *cos_Arcsin*:
 $0 < \text{Re } z \vee \text{Re } z = 0 \wedge 0 \leq \text{Im } z \implies \cos(\text{Arcsin } z) = \text{csqrt}(1 - z^2)$
by (*simp add: Arcsin_Arccos_csqrt_pos*)

6.20.29 Relationship with Arcsin on the Real Numbers

lemma *of_real_arcsin*: $|x| \leq 1 \implies \text{of_real}(\arcsin x) = \text{Arcsin}(\text{of_real } x)$
by (*smt (verit, best) Arcsin_sin Im_complex_of_real Re_complex_of_real arcsin_sin_of_real*)

lemma *Im_Arcsin_of_real*: $|x| \leq 1 \implies \text{Im}(\text{Arcsin}(\text{of_real } x)) = 0$
by (*metis Im_complex_of_real_of_real_arcsin*)

2024

corollary *Arcsin_in_Reals* [simp]: $z \in \mathbb{R} \implies |\operatorname{Re} z| \leq 1 \implies \operatorname{Arcsin} z \in \mathbb{R}$
by (metis *Im_Arcsin_of_real Re_complex_of_real Reals_cases complex_is_Real_iff*)

lemma *arcsin_eq_Re_Arcsin*: $|x| \leq 1 \implies \operatorname{arcsin} x = \operatorname{Re} (\operatorname{Arcsin} (\operatorname{of_real} x))$
by (metis *Re_complex_of_real of_real_arcsin*)

6.20.30 Relationship with Arccos on the Real Numbers

lemma *of_real_arccos*: $|x| \leq 1 \implies \operatorname{of_real}(\operatorname{arccos} x) = \operatorname{Arccos}(\operatorname{of_real} x)$
by (smt (verit, del_insts) *Arccos_unique Im_complex_of_real Re_complex_of_real arccos_lbound arccos_ubound cos_arccos_abs cos_of_real*)

lemma *Im_Arccos_of_real*: $|x| \leq 1 \implies \operatorname{Im} (\operatorname{Arccos} (\operatorname{of_real} x)) = 0$
by (metis *Im_complex_of_real of_real_arccos*)

corollary *Arccos_in_Reals* [simp]: $z \in \mathbb{R} \implies |\operatorname{Re} z| \leq 1 \implies \operatorname{Arccos} z \in \mathbb{R}$
by (metis *Im_Arccos_of_real complex_is_Real_iff of_real_Re*)

lemma *arccos_eq_Re_Arccos*: $|x| \leq 1 \implies \operatorname{arccos} x = \operatorname{Re} (\operatorname{Arccos} (\operatorname{of_real} x))$
by (metis *Re_complex_of_real of_real_arccos*)

6.20.31 Continuity results for arcsin and arccos

lemma *continuous_on_Arcsin_real* [continuous_intros]:
continuous_on $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$ *Arcsin*

proof –

have *continuous_on* $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$ $(\lambda x. \operatorname{complex_of_real} (\operatorname{arcsin} (\operatorname{Re} x))) =$

continuous_on $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$ $(\lambda x. \operatorname{complex_of_real} (\operatorname{Re} (\operatorname{Arcsin} (\operatorname{of_real} (\operatorname{Re} x))))))$

by (rule *continuous_on_cong* [OF refl]) (simp add: *arcsin_eq_Re_Arcsin*)

also have $\dots = ?thesis$

by (rule *continuous_on_cong* [OF refl]) simp

finally show $?thesis$

using *continuous_on_arcsin* [OF *continuous_on_Re* [OF *continuous_on_id*],
of $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$

continuous_on_of_real

by *fastforce*

qed

lemma *continuous_within_Arcsin_real*:

continuous (at z within $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$) *Arcsin*

using *closed_real_abs_le* *continuous_on_Arcsin_real* *continuous_on_eq* *continuous_within*

continuous_within_closed_nontrivial by *blast*

lemma *continuous_on_Arccos_real*:

continuous_on $\{w \in \mathbb{R}. |\operatorname{Re} w| \leq 1\}$ *Arccos*

proof –

```

have continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (arccos (Re
x))) =
  continuous_on {w ∈ ℝ. |Re w| ≤ 1} (λx. complex_of_real (Re (Arccos
(of_real (Re x)))))
  by (rule continuous_on_cong [OF refl]) (simp add: arccos_eq_Re_Arccos)
  also have ... = ?thesis
  by (rule continuous_on_cong [OF refl]) simp
  finally show ?thesis
  using continuous_on_arccos [OF continuous_on_Re [OF continuous_on_id],
of {w ∈ ℝ. |Re w| ≤ 1}]
  continuous_on_of_real
  by fastforce
qed

```

```

lemma continuous_within_Arccos_real:
  continuous (at z within {w ∈ ℝ. |Re w| ≤ 1}) Arccos
  using closed_real_abs_le continuous_on_Arccos_real continuous_on_eq_continuous_within
  continuous_within_closed_nontrivial by blast

```

```

lemma sinh_ln_complex: x ≠ 0 ⇒ sinh (ln x :: complex) = (x - inverse x) / 2
  by (simp add: sinh_def exp_minus scaleR_conv_of_real exp_of_real)

```

```

lemma cosh_ln_complex: x ≠ 0 ⇒ cosh (ln x :: complex) = (x + inverse x) / 2
  by (simp add: cosh_def exp_minus scaleR_conv_of_real)

```

```

lemma tanh_ln_complex: x ≠ 0 ⇒ tanh (ln x :: complex) = (x ^ 2 - 1) / (x
^ 2 + 1)
  by (simp add: tanh_def sinh_ln_complex cosh_ln_complex divide_simps power2_eq_square)

```

6.20.32 Roots of unity

```

theorem complex_root_unity:
  fixes j::nat
  assumes n ≠ 0
  shows exp(2 * of_real pi * i * of_nat j / of_nat n) ^ n = 1
  by (metis assms bot_nat_0.not_eq_extremum exp_divide_power_eq exp_of_nat2_mult
exp_two_pi_i power_one)

```

```

lemma complex_root_unity_eq:
  fixes j::nat and k::nat
  assumes 1 ≤ n
  shows (exp(2 * of_real pi * i * of_nat j / of_nat n) = exp(2 * of_real pi * i
* of_nat k / of_nat n)
  ↔ j mod n = k mod n)

```

proof –

```

  have (∃ z::int. i * (of_nat j * (of_real pi * 2)) =
    i * (of_nat k * (of_real pi * 2)) + i * (of_int z * (of_nat n * (of_real
pi * 2)))) ↔

```

```

      (∃ z::int. of_nat j * (i * (of_real pi * 2)) =
        (of_nat k + of_nat n * of_int z) * (i * (of_real pi * 2)))
    by (simp add: algebra_simps)
    also have ... ↔ (∃ z::int. of_nat j = of_nat k + of_nat n * (of_int z ::
complex))
    by simp
    also have ... ↔ (∃ z::int. of_nat j = of_nat k + of_nat n * z)
    by (metis (mono_tags, opaque_lifting) of_int_add of_int_eq_iff of_int_mult
of_int_of_nat_eq)
    also have ... ↔ int j mod int n = int k mod int n
    by (auto simp: mod_eq_dvd_iff dvd_def algebra_simps)
    also have ... ↔ j mod n = k mod n
    by (metis of_nat_eq_iff zmod_int)
    finally have (∃ z. i * (of_nat j * (of_real pi * 2)) =
      i * (of_nat k * (of_real pi * 2)) + i * (of_int z * (of_nat n * (of_real
pi * 2)))) ↔ j mod n = k mod n .
    note * = this
    show ?thesis
    using assms by (simp add: exp_eq_field_split_simps *)
qed

```

corollary *bij_betw_roots_unity:*

```

  bij_betw (λj. exp(2 * of_real pi * i * of_nat j / of_nat n))
    {..<n} {exp(2 * of_real pi * i * of_nat j / of_nat n) | j. j < n}
  by (auto simp: bij_betw_def inj_on_def complex_root_unity_eq)

```

lemma *complex_root_unity_eq_1:*

```

  fixes j::nat and k::nat
  assumes 1 ≤ n
  shows exp(2 * of_real pi * i * of_nat j / of_nat n) = 1 ↔ n dvd j
proof -
  have 1 = exp(2 * of_real pi * i * (of_nat n / of_nat n))
  using assms by simp
  then have exp(2 * of_real pi * i * (of_nat j / of_nat n)) = 1 ↔ j mod n =
n mod n
  using complex_root_unity_eq [of n j n] assms
  by simp
  then show ?thesis
  by auto
qed

```

lemma *finite_complex_roots_unity_explicit:*

```

  finite {exp(2 * of_real pi * i * of_nat j / of_nat n) | j::nat. j < n}
  by simp

```

lemma *card_complex_roots_unity_explicit:*

```

  card {exp(2 * of_real pi * i * of_nat j / of_nat n) | j::nat. j < n} = n
  by (simp add: Finite_Set.bij_betw_same_card [OF bij_betw_roots_unity, sym-
metric])

```

```

lemma complex_roots_unity:
  assumes  $1 \leq n$ 
    shows  $\{z::\text{complex}. z^n = 1\} = \{\exp(2 * \text{of\_real } \pi * i * \text{of\_nat } j / \text{of\_nat } n) \mid j. j < n\}$ 
    apply (rule Finite_Set.card_seteq [symmetric])
    using assms
    apply (auto simp: card_complex_roots_unity_explicit finite_roots_unity complex_root_unity card_roots_unity)
    done

lemma card_complex_roots_unity:  $1 \leq n \implies \text{card } \{z::\text{complex}. z^n = 1\} = n$ 
  by (simp add: card_complex_roots_unity_explicit complex_roots_unity)

lemma complex_not_root_unity:
   $1 \leq n \implies \exists u::\text{complex}. \text{norm } u = 1 \wedge u^n \neq 1$ 
  apply (rule_tac x=exp (of_real pi * i * of_real (1 / n)) in exI)
  apply (auto simp: Re_complex_div_eq_0 exp_of_nat_mult [symmetric] mult_ac exp_Euler)
  done

end

```


Chapter 7

Measure and Integration Theory

```
theory Sigma_Algebra
imports
  Complex_Main
  HOL-Library.Countable_Set
  HOL-Library.FuncSet
  HOL-Library.Indicator_Function
  HOL-Library.Extended_Nonnegative_Real
  HOL-Library.Disjoint_Sets
begin
```

7.1 Sigma Algebra

Sigma algebras are an elementary concept in measure theory. To measure — that is to integrate — functions, we first have to measure sets. Unfortunately, when dealing with a large universe, it is often not possible to consistently assign a measure to every subset. Therefore it is necessary to define the set of measurable subsets of the universe. A sigma algebra is such a set that has three very natural and desirable properties.

7.1.1 Families of sets

```
locale subset_class =
  fixes  $\Omega :: 'a \text{ set}$  and  $M :: 'a \text{ set set}$ 
  assumes space_closed:  $M \subseteq \text{Pow } \Omega$ 
```

```
lemma (in subset_class) sets_into_space:  $x \in M \implies x \subseteq \Omega$ 
by (metis PowD contra_subsetD space_closed)
```

Semiring of sets

```
locale semiring_of_sets = subset_class +
```

assumes *empty_sets*[*iff*]: $\{\} \in M$
assumes *Int*[*intro*]: $\bigwedge a b. a \in M \implies b \in M \implies a \cap b \in M$
assumes *Diff_cover*:
 $\bigwedge a b. a \in M \implies b \in M \implies \exists C \subseteq M. \text{finite } C \wedge \text{disjoint } C \wedge a - b = \bigcup C$

lemma (**in** *semiring_of_sets*) *finite_INT*[*intro*]:
assumes *finite I I* $\neq \{\}$ $\bigwedge i. i \in I \implies A i \in M$
shows $(\bigcap i \in I. A i) \in M$
using *assms* **by** (*induct rule: finite_ne_induct*) *auto*

lemma (**in** *semiring_of_sets*) *Int_space_eq1* [*simp*]: $x \in M \implies \Omega \cap x = x$
by (*metis Int_absorb1 sets_into_space*)

lemma (**in** *semiring_of_sets*) *Int_space_eq2* [*simp*]: $x \in M \implies x \cap \Omega = x$
by (*metis Int_absorb2 sets_into_space*)

lemma (**in** *semiring_of_sets*) *sets_Collect_conj*:
assumes $\{x \in \Omega. P x\} \in M$ $\{x \in \Omega. Q x\} \in M$
shows $\{x \in \Omega. Q x \wedge P x\} \in M$

proof –
have $\{x \in \Omega. Q x \wedge P x\} = \{x \in \Omega. Q x\} \cap \{x \in \Omega. P x\}$
by *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *semiring_of_sets*) *sets_Collect_finite_All'*:
assumes $\bigwedge i. i \in S \implies \{x \in \Omega. P i x\} \in M$ *finite S S* $\neq \{\}$
shows $\{x \in \Omega. \forall i \in S. P i x\} \in M$

proof –
have $\{x \in \Omega. \forall i \in S. P i x\} = (\bigcap i \in S. \{x \in \Omega. P i x\})$
using $\langle S \neq \{\} \rangle$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

Ring of sets

locale *ring_of_sets* = *semiring_of_sets* +
assumes *Un* [*intro*]: $\bigwedge a b. a \in M \implies b \in M \implies a \cup b \in M$

lemma (**in** *ring_of_sets*) *finite_Union* [*intro*]:
finite X $\implies X \subseteq M \implies \bigcup X \in M$
by (*induct set: finite*) (*auto simp add: Un*)

lemma (**in** *ring_of_sets*) *finite_UN*[*intro*]:
assumes *finite I* **and** $\bigwedge i. i \in I \implies A i \in M$
shows $(\bigcup i \in I. A i) \in M$
using *assms* **by** *induct auto*

lemma (**in** *ring_of_sets*) *Diff* [*intro*]:

assumes $a \in M$ $b \in M$ shows $a - b \in M$
 using *Diff_cover*[*OF assms*] by *auto*

lemma *ring_of_setsI*:

assumes *space_closed*: $M \subseteq \text{Pow } \Omega$
 assumes *empty_sets_iff*: $\{\} \in M$
 assumes *Un*[*intro*]: $\bigwedge a b. a \in M \implies b \in M \implies a \cup b \in M$
 assumes *Diff*[*intro*]: $\bigwedge a b. a \in M \implies b \in M \implies a - b \in M$
 shows *ring_of_sets* Ω M

proof

fix a b assume *ab*: $a \in M$ $b \in M$
 from *ab* show $\exists C \subseteq M. \text{finite } C \wedge \text{disjoint } C \wedge a - b = \bigcup C$
 by (*intro exI*[*of_* $\{a - b\}$]) (*auto simp: disjoint_def*)
 have $a \cap b = a - (a - b)$ by *auto*
 also have $\dots \in M$ using *ab* by *auto*
 finally show $a \cap b \in M$.

qed *fact+*

lemma *ring_of_sets_iff*: $\text{ring_of_sets } \Omega$ $M \iff M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge$
 $(\forall a \in M. \forall b \in M. a \cup b \in M) \wedge (\forall a \in M. \forall b \in M. a - b \in M)$

proof

assume *ring_of_sets* Ω M
 then interpret *ring_of_sets* Ω M .
 show $M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge (\forall a \in M. \forall b \in M. a \cup b \in M) \wedge (\forall a \in M. \forall b \in M. a - b \in M)$
 using *space_closed* by *auto*
 qed (*auto intro!*: *ring_of_setsI*)

lemma (in *ring_of_sets*) *insert_in_sets*:

assumes $\{x\} \in M$ $A \in M$ shows $\text{insert } x$ $A \in M$

proof -

have $\{x\} \cup A \in M$ using *assms* by (*rule Un*)
 thus *?thesis* by *auto*

qed

lemma (in *ring_of_sets*) *sets_Collect_disj*:

assumes $\{x \in \Omega. P x\} \in M$ $\{x \in \Omega. Q x\} \in M$
 shows $\{x \in \Omega. Q x \vee P x\} \in M$

proof -

have $\{x \in \Omega. Q x \vee P x\} = \{x \in \Omega. Q x\} \cup \{x \in \Omega. P x\}$
 by *auto*
 with *assms* show *?thesis* by *auto*

qed

lemma (in *ring_of_sets*) *sets_Collect_finite_Ex*:

assumes $\bigwedge i. i \in S \implies \{x \in \Omega. P i x\} \in M$ *finite S*
 shows $\{x \in \Omega. \exists i \in S. P i x\} \in M$

proof -

have $\{x \in \Omega. \exists i \in S. P i x\} = (\bigcup i \in S. \{x \in \Omega. P i x\})$

2032

by *auto*
with *assms* show *?thesis* by *auto*
qed

Algebra of sets

locale *algebra* = *ring_of_sets* +
assumes *top* [*iff*]: $\Omega \in M$

lemma (in *algebra*) *compl_sets* [*intro*]:
 $a \in M \implies \Omega - a \in M$
by *auto*

proposition *algebra_iff_Un*:

$algebra\ \Omega\ M \longleftrightarrow$
 $M \subseteq Pow\ \Omega \wedge$
 $\{\} \in M \wedge$
 $(\forall a \in M. \Omega - a \in M) \wedge$
 $(\forall a \in M. \forall b \in M. a \cup b \in M)$ (is $_ \longleftrightarrow ?Un$)

proof

assume *algebra* $\Omega\ M$
then interpret *algebra* $\Omega\ M$.
show *?Un* using *sets_into_space* by *auto*

next

assume *?Un*
then have $\Omega \in M$ by *auto*
interpret *ring_of_sets* $\Omega\ M$
proof (rule *ring_of_setsI*)
show $\Omega: M \subseteq Pow\ \Omega\ \{\} \in M$
using $\langle ?Un \rangle$ by *auto*
fix *a b* assume $a: a \in M$ and $b: b \in M$
then show $a \cup b \in M$ using $\langle ?Un \rangle$ by *auto*
have $a - b = \Omega - ((\Omega - a) \cup b)$
using $\Omega\ a\ b$ by *auto*
then show $a - b \in M$
using $a\ b\ \langle ?Un \rangle$ by *auto*

qed

show *algebra* $\Omega\ M$ proof qed fact

qed

proposition *algebra_iff_Int*:

$algebra\ \Omega\ M \longleftrightarrow$
 $M \subseteq Pow\ \Omega \ \& \ \{\} \in M \ \&$
 $(\forall a \in M. \Omega - a \in M) \ \&$
 $(\forall a \in M. \forall b \in M. a \cap b \in M)$ (is $_ \longleftrightarrow ?Int$)

proof

assume *algebra* $\Omega\ M$
then interpret *algebra* $\Omega\ M$.
show *?Int* using *sets_into_space* by *auto*

```

next
  assume ?Int
  show algebra  $\Omega$   $M$ 
  proof (unfold algebra_iff_Un, intro conjI ballI)
    show  $\Omega: M \subseteq \text{Pow } \Omega \ \{\} \in M$ 
      using <?Int> by auto
    from <?Int> show  $\bigwedge a. a \in M \implies \Omega - a \in M$  by auto
    fix  $a$   $b$  assume  $M: a \in M \ b \in M$ 
    hence  $a \cup b = \Omega - ((\Omega - a) \cap (\Omega - b))$ 
      using  $\Omega$  by blast
    also have  $\dots \in M$ 
      using  $M$  <?Int> by auto
    finally show  $a \cup b \in M$  .
  qed
qed

```

```

lemma (in algebra) sets_Collect_neg:
  assumes  $\{x \in \Omega. P \ x\} \in M$ 
  shows  $\{x \in \Omega. \neg P \ x\} \in M$ 
  proof -
    have  $\{x \in \Omega. \neg P \ x\} = \Omega - \{x \in \Omega. P \ x\}$  by auto
    with assms show ?thesis by auto
  qed

```

```

lemma (in algebra) sets_Collect_imp:
   $\{x \in \Omega. P \ x\} \in M \implies \{x \in \Omega. Q \ x\} \in M \implies \{x \in \Omega. Q \ x \longrightarrow P \ x\} \in M$ 
  unfolding imp_conv_disj by (intro sets_Collect_disj sets_Collect_neg)

```

```

lemma (in algebra) sets_Collect_const:
   $\{x \in \Omega. P\} \in M$ 
  by (cases P) auto

```

```

lemma algebra_single_set:
   $X \subseteq S \implies \text{algebra } S \ \{\ \{\}, X, S - X, S \}$ 
  by (auto simp: algebra_iff_Int)

```

Restricted algebras

```

abbreviation (in algebra)
  restricted_space  $A \equiv ((\cap) \ A) \ ' \ M$ 

```

```

lemma (in algebra) restricted_algebra:
  assumes  $A \in M$  shows algebra  $A$  (restricted_space  $A$ )
  using assms by (auto simp: algebra_iff_Int)

```

Sigma Algebras

```

locale sigma_algebra = algebra +
  assumes countable_nat_UN [intro]:  $\bigwedge A. \text{range } A \subseteq M \implies (\bigcup i::\text{nat}. A \ i) \in M$ 

```

lemma (in algebra) *is_sigma_algebra*:

assumes *finite M*

shows *sigma_algebra* Ω *M*

proof

fix *A* :: *nat* \Rightarrow 'a set **assume** *range A* \subseteq *M*

then have $(\bigcup i. A\ i) = (\bigcup s \in M. \text{range } A.\ s)$

by *auto*

also have $(\bigcup s \in M. \text{range } A.\ s) \in M$

using \langle *finite M* \rangle by *auto*

finally show $(\bigcup i. A\ i) \in M$.

qed

lemma *countable_UN_eq*:

fixes *A* :: 'i::countable \Rightarrow 'a set

shows $(\text{range } A \subseteq M \longrightarrow (\bigcup i. A\ i) \in M) \longleftrightarrow$

$(\text{range } (A \circ \text{from_nat}) \subseteq M \longrightarrow (\bigcup i. (A \circ \text{from_nat})\ i) \in M)$

proof –

let $?A' = A \circ \text{from_nat}$

have *: $(\bigcup i. ?A'\ i) = (\bigcup i. A\ i)$ (is ?l = ?r)

proof *safe*

fix *x* *i* **assume** $x \in A\ i$ **thus** $x \in ?l$

by (*auto intro!*: *exI[of _ to_nat i]*)

next

fix *x* *i* **assume** $x \in ?A'\ i$ **thus** $x \in ?r$

by (*auto intro!*: *exI[of _ from_nat i]*)

qed

have $A \text{ 'range from_nat} = \text{range } A$

using *surj_from_nat* by *simp*

then have **: $\text{range } ?A' = \text{range } A$

by (*simp only: image_comp [symmetric]*)

show *?thesis* **unfolding** * ** ..

qed

lemma (in sigma_algebra) *countable_Union [intro]*:

assumes *countable X* $X \subseteq M$ **shows** $\bigcup X \in M$

proof *cases*

assume $X \neq \{\}$

hence $\bigcup X = (\bigcup n. \text{from_nat_into } X\ n)$

using *assms* by (*auto cong del: SUP_cong*)

also have ... $\in M$ **using** *assms*

by (*auto intro!*: *countable_nat_UN*) (*metis* $\langle X \neq \{\} \rangle$ *from_nat_into_subsetD*)

finally show *?thesis* .

qed *simp*

lemma (in sigma_algebra) *countable_UN [intro]*:

fixes *A* :: 'i::countable \Rightarrow 'a set

assumes $A\ X \subseteq M$

shows $(\bigcup x \in X. A\ x) \in M$

proof –

```

let ?A =  $\lambda i.$  if  $i \in X$  then  $A\ i$  else {}
from assms have  $\text{range } ?A \subseteq M$  by auto
with countable_nat_UN [of ?A  $\circ$  from_nat] countable_UN_eq [of ?A  $M$ ]
have  $(\bigcup x. ?A\ x) \in M$  by auto
moreover have  $(\bigcup x. ?A\ x) = (\bigcup x \in X. A\ x)$  by (auto split: if_split_asm)
ultimately show ?thesis by simp
qed

```

```

lemma (in sigma_algebra) countable_UN':
  fixes  $A :: 'i \Rightarrow 'a\ \text{set}$ 
  assumes  $X:$  countable  $X$ 
  assumes  $A:$   $A'X \subseteq M$ 
  shows  $(\bigcup x \in X. A\ x) \in M$ 
proof -
  have  $(\bigcup x \in X. A\ x) = (\bigcup i \in \text{to\_nat\_on } X'X. A\ (\text{from\_nat\_into } X\ i))$ 
    using  $X$  by auto
  also have  $\dots \in M$ 
    using  $A\ X$ 
    by (intro countable_UN) auto
  finally show ?thesis .
qed

```

```

lemma (in sigma_algebra) countable_UN'':
   $\llbracket \text{countable } X; \bigwedge x\ y. x \in X \implies A\ x \in M \rrbracket \implies (\bigcup x \in X. A\ x) \in M$ 
by (erule countable_UN') (auto)

```

```

lemma (in sigma_algebra) countable_INT [intro]:
  fixes  $A :: 'i::\text{countable} \Rightarrow 'a\ \text{set}$ 
  assumes  $A:$   $A'X \subseteq M$   $X \neq \{\}$ 
  shows  $(\bigcap i \in X. A\ i) \in M$ 
proof -
  from  $A$  have  $\forall i \in X. A\ i \in M$  by fast
  hence  $\Omega - (\bigcup i \in X. \Omega - A\ i) \in M$  by blast
  moreover
  have  $(\bigcap i \in X. A\ i) = \Omega - (\bigcup i \in X. \Omega - A\ i)$  using space_closed  $A$ 
    by blast
  ultimately show ?thesis by metis
qed

```

```

lemma (in sigma_algebra) countable_INT':
  fixes  $A :: 'i \Rightarrow 'a\ \text{set}$ 
  assumes  $X:$  countable  $X$   $X \neq \{\}$ 
  assumes  $A:$   $A'X \subseteq M$ 
  shows  $(\bigcap x \in X. A\ x) \in M$ 
proof -
  have  $(\bigcap x \in X. A\ x) = (\bigcap i \in \text{to\_nat\_on } X'X. A\ (\text{from\_nat\_into } X\ i))$ 
    using  $X$  by auto
  also have  $\dots \in M$ 
    using  $A\ X$ 

```

by (intro countable_INT) auto
 finally show ?thesis .
 qed

lemma (in sigma_algebra) countable_INT'':
 $UNIV \in M \implies \text{countable } I \implies (\bigwedge i. i \in I \implies F i \in M) \implies (\bigcap i \in I. F i) \in M$
 by (cases I = {}) (auto intro: countable_INT')

lemma (in sigma_algebra) countable:
 assumes $\bigwedge a. a \in A \implies \{a\} \in M$ countable A
 shows $A \in M$
 proof -
 have $(\bigcup a \in A. \{a\}) \in M$
 using assms by (intro countable_UN') auto
 also have $(\bigcup a \in A. \{a\}) = A$ by auto
 finally show ?thesis by auto
 qed

lemma ring_of_sets_Pow: ring_of_sets sp (Pow sp)
 by (auto simp: ring_of_sets_iff)

lemma algebra_Pow: algebra sp (Pow sp)
 by (auto simp: algebra_iff_Un)

lemma sigma_algebra_iff:
 $\text{sigma_algebra } \Omega M \iff$
 $\text{algebra } \Omega M \wedge (\forall A. \text{range } A \subseteq M \longrightarrow (\bigcup i :: \text{nat}. A i) \in M)$
 by (simp add: sigma_algebra_def sigma_algebra_axioms_def)

lemma sigma_algebra_Pow: sigma_algebra sp (Pow sp)
 by (auto simp: sigma_algebra_iff algebra_iff_Int)

lemma (in sigma_algebra) sets_Collect_countable_All:
 assumes $\bigwedge i. \{x \in \Omega. P i x\} \in M$
 shows $\{x \in \Omega. \forall i :: 'i :: \text{countable}. P i x\} \in M$
 proof -
 have $\{x \in \Omega. \forall i :: 'i :: \text{countable}. P i x\} = (\bigcap i. \{x \in \Omega. P i x\})$ by auto
 with assms show ?thesis by auto
 qed

lemma (in sigma_algebra) sets_Collect_countable_Ex:
 assumes $\bigwedge i. \{x \in \Omega. P i x\} \in M$
 shows $\{x \in \Omega. \exists i :: 'i :: \text{countable}. P i x\} \in M$
 proof -
 have $\{x \in \Omega. \exists i :: 'i :: \text{countable}. P i x\} = (\bigcup i. \{x \in \Omega. P i x\})$ by auto
 with assms show ?thesis by auto
 qed

lemma (in sigma_algebra) sets_Collect_countable_Ex':


```

  assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
  assumes countable I
  shows  $\{x \in \Omega. \exists i \in I. P\ i\ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \exists i \in I. P\ i\ x\} = (\bigcup i \in I. \{x \in \Omega. P\ i\ x\})$  by auto
  with assms show ?thesis
    by (auto intro!: countable_UN')
qed

```

```

lemma (in sigma_algebra) sets_Collect_countable_All':
  assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
  assumes countable I
  shows  $\{x \in \Omega. \forall i \in I. P\ i\ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \forall i \in I. P\ i\ x\} = (\bigcap i \in I. \{x \in \Omega. P\ i\ x\}) \cap \Omega$  by auto
  with assms show ?thesis
    by (cases I = {}) (auto intro!: countable_INT')
qed

```

```

lemma (in sigma_algebra) sets_Collect_countable_Ex1':
  assumes  $\bigwedge i. i \in I \implies \{x \in \Omega. P\ i\ x\} \in M$ 
  assumes countable I
  shows  $\{x \in \Omega. \exists ! i \in I. P\ i\ x\} \in M$ 
proof -
  have  $\{x \in \Omega. \exists ! i \in I. P\ i\ x\} = \{x \in \Omega. \exists i \in I. P\ i\ x \wedge (\forall j \in I. P\ j\ x \longrightarrow i = j)\}$ 
    by auto
  with assms show ?thesis
    by (auto intro!: sets_Collect_countable_All' sets_Collect_countable_Ex' sets_Collect_conj
        sets_Collect_imp sets_Collect_const)
qed

```

```

lemmas (in sigma_algebra) sets_Collect =
  sets_Collect_imp sets_Collect_disj sets_Collect_conj sets_Collect_neg sets_Collect_const
  sets_Collect_countable_All sets_Collect_countable_Ex sets_Collect_countable_All

```

```

lemma (in sigma_algebra) sets_Collect_countable_Ball:
  assumes  $\bigwedge i. \{x \in \Omega. P\ i\ x\} \in M$ 
  shows  $\{x \in \Omega. \forall i :: 'i :: countable \in X. P\ i\ x\} \in M$ 
  unfolding Ball_def by (intro sets_Collect assms)

```

```

lemma (in sigma_algebra) sets_Collect_countable_Bex:
  assumes  $\bigwedge i. \{x \in \Omega. P\ i\ x\} \in M$ 
  shows  $\{x \in \Omega. \exists i :: 'i :: countable \in X. P\ i\ x\} \in M$ 
  unfolding Bex_def by (intro sets_Collect assms)

```

```

lemma sigma_algebra_single_set:
  assumes  $X \subseteq S$ 
  shows sigma_algebra S { {}, X, S - X, S }
  using algebra.is_sigma_algebra[OF algebra_single_set[OF  $\langle X \subseteq S \rangle$ ]] by simp

```

Binary Unions

definition *binary* :: 'a ⇒ 'a ⇒ nat ⇒ 'a
 where *binary* a b = (λx. b)(0 := a)

lemma *range_binary_eq*: $\text{range}(\text{binary } a \ b) = \{a, b\}$
 by (*auto simp add: binary_def*)

lemma *Un_range_binary*: $a \cup b = (\bigcup i::\text{nat}. \text{binary } a \ b \ i)$
 by (*simp add: range_binary_eq cong del: SUP_cong_simp*)

lemma *Int_range_binary*: $a \cap b = (\bigcap i::\text{nat}. \text{binary } a \ b \ i)$
 by (*simp add: range_binary_eq cong del: INF_cong_simp*)

lemma *sigma_algebra_iff2*:
 $\text{sigma_algebra } \Omega \ M \longleftrightarrow$
 $M \subseteq \text{Pow } \Omega \wedge \{\} \in M \wedge (\forall s \in M. \Omega - s \in M)$
 $\wedge (\forall A. \text{range } A \subseteq M \longrightarrow (\bigcup i::\text{nat}. A \ i) \in M)$ (**is** ?P \longleftrightarrow ?R ∧ ?S ∧ ?V ∧ ?W)

proof

assume ?P

then interpret *sigma_algebra* Ω M .

from *space_closed* show ?R ∧ ?S ∧ ?V ∧ ?W

by *auto*

next

assume ?R ∧ ?S ∧ ?V ∧ ?W

then have ?R ?S ?V ?W

by *simp_all*

show ?P

proof (*rule sigma_algebra.intro*)

show *sigma_algebra_axioms* M

by *standard* (*use* ⟨?W⟩ **in** *simp*)

from ⟨?W⟩ **have** *: $\text{range}(\text{binary } a \ b) \subseteq M \implies \bigcup (\text{range}(\text{binary } a \ b)) \in M$

for a b

by *auto*

show *algebra* Ω M

unfolding *algebra_iff_Un* **using** ⟨?R⟩ ⟨?S⟩ ⟨?V⟩ *

by (*auto simp add: range_binary_eq*)

qed

qed

Initial Sigma Algebra

Sigma algebras can naturally be created as the closure of any set of M with regard to the properties just postulated.

inductive_set *sigma_sets* :: 'a set ⇒ 'a set set ⇒ 'a set set
for *sp* :: 'a set **and** A :: 'a set set
where
Basic[*intro, simp*]: $a \in A \implies a \in \text{sigma_sets } sp \ A$

| *Empty*: $\{\} \in \text{sigma_sets } sp \ A$
 | *Compl*: $a \in \text{sigma_sets } sp \ A \implies sp - a \in \text{sigma_sets } sp \ A$
 | *Union*: $(\bigwedge i::nat. a \ i \in \text{sigma_sets } sp \ A) \implies (\bigcup i. a \ i) \in \text{sigma_sets } sp \ A$

lemma (in *sigma_algebra*) *sigma_sets_subset*:

assumes *a*: $a \subseteq M$

shows *sigma_sets* $\Omega \ a \subseteq M$

proof

fix *x*

assume $x \in \text{sigma_sets } \Omega \ a$

from *this* **show** $x \in M$

by (*induct* rule: *sigma_sets.induct*, *auto*) (*metis* *a_subsetD*)

qed

lemma *sigma_sets_into_sp*: $A \subseteq \text{Pow } sp \implies x \in \text{sigma_sets } sp \ A \implies x \subseteq sp$

by (*erule* *sigma_sets.induct*, *auto*)

lemma *sigma_sets_finite*: $\llbracket x \in \text{sigma_sets } \Omega \ (\text{Pow } \Omega); \text{finite } \Omega \rrbracket \implies \text{finite } x$

by (*meson* *finite_subset* *order.refl* *sigma_sets_into_sp*)

lemma *sigma_algebra_sigma_sets*:

$a \subseteq \text{Pow } \Omega \implies \text{sigma_algebra } \Omega \ (\text{sigma_sets } \Omega \ a)$

by (*auto* *simp* *add*: *sigma_algebra_iff2* *dest*: *sigma_sets_into_sp*

intro!: *sigma_sets.Union* *sigma_sets.Empty* *sigma_sets.Compl*)

lemma *sigma_sets_least_sigma_algebra*:

assumes $A \subseteq \text{Pow } S$

shows *sigma_sets* $S \ A = \bigcap \{B. A \subseteq B \wedge \text{sigma_algebra } S \ B\}$

proof *safe*

fix *B X* **assume** $A \subseteq B$ **and** *sa*: *sigma_algebra* $S \ B$

and *X*: $X \in \text{sigma_sets } S \ A$

from *sigma_algebra.sigma_sets_subset*[*OF sa*, *simplified*, *OF* $\langle A \subseteq B \rangle$] *X*

show $X \in B$ **by** *auto*

next

fix *X* **assume** $X \in \bigcap \{B. A \subseteq B \wedge \text{sigma_algebra } S \ B\}$

then **have** [*intro!*]: $\bigwedge B. A \subseteq B \implies \text{sigma_algebra } S \ B \implies X \in B$

by *simp*

have $A \subseteq \text{sigma_sets } S \ A$ **using** *assms* **by** *auto*

moreover **have** *sigma_algebra* $S \ (\text{sigma_sets } S \ A)$

using *assms* **by** (*intro* *sigma_algebra_sigma_sets*[*of A*]) *auto*

ultimately **show** $X \in \text{sigma_sets } S \ A$ **by** *auto*

qed

lemma *sigma_sets_top*: $sp \in \text{sigma_sets } sp \ A$

by (*metis* *Diff_empty* *sigma_sets.Compl* *sigma_sets.Empty*)

lemma *binary_in_sigma_sets*:

binary *a b i* $\in \text{sigma_sets } sp \ A$ **if** $a \in \text{sigma_sets } sp \ A$ **and** $b \in \text{sigma_sets } sp \ A$

using *that* **by** (*simp* *add*: *binary_def*)

lemma *sigma_sets_Un*:

$a \cup b \in \text{sigma_sets } sp \ A$ **if** $a \in \text{sigma_sets } sp \ A$ **and** $b \in \text{sigma_sets } sp \ A$
using that by (*simp add: Un_range_binary binary_in_sigma_sets Union*)

lemma *sigma_sets_Inter*:

assumes *Asb*: $A \subseteq \text{Pow } sp$

shows $(\bigwedge i::\text{nat. } a \ i \in \text{sigma_sets } sp \ A) \implies (\bigcap i. \ a \ i) \in \text{sigma_sets } sp \ A$

proof –

assume *ai*: $\bigwedge i::\text{nat. } a \ i \in \text{sigma_sets } sp \ A$

hence $\bigwedge i::\text{nat. } sp\text{-}(a \ i) \in \text{sigma_sets } sp \ A$

by (*rule sigma_sets.Compl*)

hence $(\bigcup i. \ sp\text{-}(a \ i)) \in \text{sigma_sets } sp \ A$

by (*rule sigma_sets.Union*)

hence $sp\text{-}(\bigcup i. \ sp\text{-}(a \ i)) \in \text{sigma_sets } sp \ A$

by (*rule sigma_sets.Compl*)

also have $sp\text{-}(\bigcup i. \ sp\text{-}(a \ i)) = sp \ Int \ (\bigcap i. \ a \ i)$

by *auto*

also have $\dots = (\bigcap i. \ a \ i)$ **using** *ai*

by (*blast dest: sigma_sets_into_sp [OF Asb]*)

finally show *?thesis* .

qed

lemma *sigma_sets_INTER*:

assumes *Asb*: $A \subseteq \text{Pow } sp$

and *ai*: $\bigwedge i::\text{nat. } i \in S \implies a \ i \in \text{sigma_sets } sp \ A$ **and** *non*: $S \neq \{\}$

shows $(\bigcap i \in S. \ a \ i) \in \text{sigma_sets } sp \ A$

proof –

from *ai* **have** $\bigwedge i. \ (\text{if } i \in S \ \text{then } a \ i \ \text{else } sp) \in \text{sigma_sets } sp \ A$

by (*simp add: sigma_sets.intros(2-) sigma_sets_top*)

hence $(\bigcap i. \ (\text{if } i \in S \ \text{then } a \ i \ \text{else } sp)) \in \text{sigma_sets } sp \ A$

by (*rule sigma_sets_Inter [OF Asb]*)

also have $(\bigcap i. \ (\text{if } i \in S \ \text{then } a \ i \ \text{else } sp)) = (\bigcap i \in S. \ a \ i)$

by *auto* (*metis ai non sigma_sets_into_sp subset_empty subset_iff Asb*)+

finally show *?thesis* .

qed

lemma *sigma_sets_UNION*:

countable B $\implies (\bigwedge b. \ b \in B \implies b \in \text{sigma_sets } X \ A) \implies \bigcup B \in \text{sigma_sets } X \ A$

using *from_nat_into* [*of B*] *range_from_nat_into* [*of B*] *sigma_sets.Union* [*of from_nat_into B X A*]

by (*cases B = \{\}*) (*simp_all add: sigma_sets.Empty cong del: SUP_cong*)

lemma (**in** *sigma_algebra*) *sigma_sets_eq*:

$\text{sigma_sets } \Omega \ M = M$

proof

show $M \subseteq \text{sigma_sets } \Omega \ M$

by (*metis Set.subsetI sigma_sets.Basic*)

```

next
show sigma_sets  $\Omega$   $M \subseteq M$ 
  by (metis sigma_sets_subset subset_refl)
qed

```

```

lemma sigma_sets_eqI:
  assumes A:  $\bigwedge a. a \in A \implies a \in \text{sigma\_sets } M B$ 
  assumes B:  $\bigwedge b. b \in B \implies b \in \text{sigma\_sets } M A$ 
  shows sigma_sets  $M A = \text{sigma\_sets } M B$ 
proof (intro set_eqI iffI)
  fix a assume a  $\in \text{sigma\_sets } M A$ 
  from this A show a  $\in \text{sigma\_sets } M B$ 
    by induct (auto intro!: sigma_sets.intros(2-) del: sigma_sets.Basic)
next
  fix b assume b  $\in \text{sigma\_sets } M B$ 
  from this B show b  $\in \text{sigma\_sets } M A$ 
    by induct (auto intro!: sigma_sets.intros(2-) del: sigma_sets.Basic)
qed

```

```

lemma sigma_sets_subseteq: assumes  $A \subseteq B$  shows sigma_sets  $X A \subseteq \text{sigma\_sets } X B$ 
proof
  fix x assume x  $\in \text{sigma\_sets } X A$  then show x  $\in \text{sigma\_sets } X B$ 
    by induct (insert  $\langle A \subseteq B \rangle$ , auto intro: sigma_sets.intros(2-))
qed

```

```

lemma sigma_sets_mono: assumes  $A \subseteq \text{sigma\_sets } X B$  shows sigma_sets  $X A \subseteq \text{sigma\_sets } X B$ 
proof
  fix x assume x  $\in \text{sigma\_sets } X A$  then show x  $\in \text{sigma\_sets } X B$ 
    by induct (insert  $\langle A \subseteq \text{sigma\_sets } X B \rangle$ , auto intro: sigma_sets.intros(2-))
qed

```

```

lemma sigma_sets_mono': assumes  $A \subseteq B$  shows sigma_sets  $X A \subseteq \text{sigma\_sets } X B$ 
proof
  fix x assume x  $\in \text{sigma\_sets } X A$  then show x  $\in \text{sigma\_sets } X B$ 
    by induct (insert  $\langle A \subseteq B \rangle$ , auto intro: sigma_sets.intros(2-))
qed

```

```

lemma sigma_sets_superset_generator:  $A \subseteq \text{sigma\_sets } X A$ 
  by (auto intro: sigma_sets.Basic)

```

```

lemma (in sigma_algebra) restriction_in_sets:
  fixes A :: nat  $\Rightarrow$  'a set
  assumes S  $\in M$ 
  and *: range A  $\subseteq (\lambda A. S \cap A) \text{ ' } M$  (is  $\_ \subseteq ?r$ )
  shows range A  $\subseteq M$  ( $\bigcup i. A i$ )  $\in (\lambda A. S \cap A) \text{ ' } M$ 
proof -

```

```

{ fix i have A i ∈ ?r using * by auto
  hence ∃ B. A i = B ∩ S ∧ B ∈ M by auto
  hence A i ⊆ S A i ∈ M using ⟨S ∈ M⟩ by auto }
thus range A ⊆ M (⋃ i. A i) ∈ (λA. S ∩ A) ‘ M
  by (auto intro!: image_eqI[of _ _ (⋃ i. A i)])
qed

```

```

lemma (in sigma_algebra) restricted_sigma_algebra:
  assumes S ∈ M
  shows sigma_algebra S (restricted_space S)
  unfolding sigma_algebra_def sigma_algebra_axioms_def
proof safe
  show algebra S (restricted_space S) using restricted_algebra[OF assms] .
next
  fix A :: nat ⇒ 'a set assume range A ⊆ restricted_space S
  from restriction_in_sets[OF assms this[simplified]]
  show (⋃ i. A i) ∈ restricted_space S by simp
qed

```

```

lemma sigma_sets_Int:
  assumes A ∈ sigma_sets sp st A ⊆ sp
  shows (⋂) A ‘ sigma_sets sp st = sigma_sets A ((⋂) A ‘ st)
proof (intro equalityI subsetI)
  fix x assume x ∈ (⋂) A ‘ sigma_sets sp st
  then obtain y where y ∈ sigma_sets sp st x = y ∩ A by auto
  then have x ∈ sigma_sets (A ∩ sp) ((⋂) A ‘ st)
  proof (induct arbitrary: x)
    case (Compl a)
    then show ?case
      by (force intro!: sigma_sets.Compl simp: Diff_Int_distrib ac_simps)
  next
    case (Union a)
    then show ?case
      by (auto intro!: sigma_sets.Union
          simp add: UN_extend_simps simp del: UN_simps)
  qed (auto intro!: sigma_sets.intros(2-))
  then show x ∈ sigma_sets A ((⋂) A ‘ st)
  using ⟨A ⊆ sp⟩ by (simp add: Int_absorb2)
next
  fix x assume x ∈ sigma_sets A ((⋂) A ‘ st)
  then show x ∈ (⋂) A ‘ sigma_sets sp st
  proof induct
    case (Compl a)
    then obtain x where a = A ∩ x x ∈ sigma_sets sp st by auto
    then show ?case using ⟨A ⊆ sp⟩
      by (force simp add: image_iff intro!: beI[of _ sp - x] sigma_sets.Compl)
  next
    case (Union a)
    then have ∀ i. ∃ x. x ∈ sigma_sets sp st ∧ a i = A ∩ x

```

```

    by (auto simp: image_iff Bex_def)
  then obtain f where  $\forall x. f x \in \text{sigma\_sets } sp \ st \wedge a x = A \cap f x$ 
    by metis
  then show ?case
    by (auto intro!: bexI[of _ ( $\bigcup x. f x$ )] sigma_sets.Union
        simp add: image_iff)
qed (auto intro!: sigma_sets.intros(2-))
qed

```

```

lemma sigma_sets_empty_eq: sigma_sets A {} = {{}}, A}
proof (intro set_eqI iffI)
  fix a assume a  $\in$  sigma_sets A {} then show a  $\in$  {{}}, A}
    by induct blast+
qed (auto intro: sigma_sets.Empty sigma_sets_top)

```

```

lemma sigma_sets_single[simp]: sigma_sets A {A} = {{}}, A}
proof (intro set_eqI iffI)
  fix x assume x  $\in$  sigma_sets A {A}
  then show x  $\in$  {{}}, A}
    by induct blast+
next
  fix x assume x  $\in$  {{}}, A}
  then show x  $\in$  sigma_sets A {A}
    by (auto intro: sigma_sets.Empty sigma_sets_top)
qed

```

```

lemma sigma_sets_sigma_sets_eq:
   $M \subseteq Pow S \implies \text{sigma\_sets } S (\text{sigma\_sets } S M) = \text{sigma\_sets } S M$ 
  by (rule sigma_algebra.sigma_sets_eq[OF sigma_algebra_sigma_sets, of M S])
auto

```

```

lemma sigma_sets_singleton:
  assumes  $X \subseteq S$ 
  shows sigma_sets S { X } = { {}, X, S - X, S }
proof -
  interpret sigma_algebra S { {}, X, S - X, S }
  by (rule sigma_algebra_single_set) fact
  have sigma_sets S { X }  $\subseteq$  sigma_sets S { {}, X, S - X, S }
  by (rule sigma_sets_subseteq) simp
  moreover have ... = { {}, X, S - X, S }
  using sigma_sets_eq by simp
  moreover
  { fix A assume A  $\in$  { {}, X, S - X, S }
    then have A  $\in$  sigma_sets S { X }
      by (auto intro: sigma_sets.intros(2-) sigma_sets_top) }
  ultimately have sigma_sets S { X } = sigma_sets S { {}, X, S - X, S }
  by (intro antisym) auto
  with sigma_sets_eq show ?thesis by simp
qed

```

lemma *restricted_sigma*:

assumes $S: S \in \text{sigma_sets } \Omega \ M$ **and** $M: M \subseteq \text{Pow } \Omega$

shows $\text{algebra.restricted_space } (\text{sigma_sets } \Omega \ M) \ S =$
 $\text{sigma_sets } S \ (\text{algebra.restricted_space } M \ S)$

proof –

from $S \ \text{sigma_sets_into_sp}[\text{OF } M]$

have $S \in \text{sigma_sets } \Omega \ M \ S \subseteq \Omega$ **by** *auto*

from $\text{sigma_sets_Int}[\text{OF } \text{this}]$

show *?thesis* **by** *simp*

qed

lemma *sigma_sets_vimage_commute*:

assumes $X: X \in \Omega \rightarrow \Omega'$

shows $\{X -' A \cap \Omega \mid A. A \in \text{sigma_sets } \Omega' \ M'\}$
 $= \text{sigma_sets } \Omega \ \{X -' A \cap \Omega \mid A. A \in M'\}$ (**is** $?L = ?R$)

proof

show $?L \subseteq ?R$

proof *clarify*

fix A **assume** $A \in \text{sigma_sets } \Omega' \ M'$

then show $X -' A \cap \Omega \in ?R$

proof *induct*

case *Empty* **then show** *?case*

by (*auto intro!*: sigma_sets.Empty)

next

case (*Compl B*)

have [*simp*]: $X -' (\Omega' - B) \cap \Omega = \Omega - (X -' B \cap \Omega)$

by (*auto simp add: funcset_mem* [*OF X*])

with *Compl* **show** *?case*

by (*auto intro!*: sigma_sets.Compl)

next

case (*Union F*)

then show *?case*

by (*auto simp add: vimage_UN UN_extend_simps(4) simp del: UN_simps*
intro!: sigma_sets.Union)

qed *auto*

qed

show $?R \subseteq ?L$

proof *clarify*

fix A **assume** $A \in ?R$

then show $\exists B. A = X -' B \cap \Omega \wedge B \in \text{sigma_sets } \Omega' \ M'$

proof *induct*

case (*Basic B*) **then show** *?case* **by** *auto*

next

case *Empty* **then show** *?case*

by (*auto intro!*: $\text{sigma_sets.Empty exI}[\text{of } _ \{\}]$)

next

case (*Compl B*)

then obtain A **where** $A: B = X -' A \cap \Omega \ A \in \text{sigma_sets } \Omega' \ M'$ **by** *auto*


```

then have [simp]:  $\Omega - B = X - '(\Omega' - A) \cap \Omega$ 
  by (auto simp add: funcset_mem [OF X])
with A(2) show ?case
  by (auto intro: sigma_sets.Compl)
next
case (Union F)
then have  $\forall i. \exists B. F i = X - 'B \cap \Omega \wedge B \in \text{sigma\_sets } \Omega' M'$  by auto
then obtain A where  $\forall x. F x = X - 'A x \cap \Omega \wedge A x \in \text{sigma\_sets } \Omega' M'$ 
  by metis
then show ?case
  by (auto simp: vimage_UN[symmetric] intro: sigma_sets.Union)
qed
qed
qed

```

```

lemma (in ring_of_sets) UNION_in_sets:
  fixes A:: nat  $\Rightarrow$  'a set
  assumes A: range A  $\subseteq$  M
  shows  $(\bigcup_{i \in \{0..<n\}}. A i) \in M$ 
proof (induct n)
  case 0 show ?case by simp
next
  case (Suc n)
  thus ?case
  by (simp add: atLeastLessThanSuc) (metis A Un UNIV_I image_subset_iff)
qed

```

```

lemma (in ring_of_sets) range_disjointed_sets:
  assumes A: range A  $\subseteq$  M
  shows range (disjointed A)  $\subseteq$  M
proof (auto simp add: disjointed_def)
  fix n
  show A n -  $(\bigcup_{i \in \{0..<n\}}. A i) \in M$  using UNION_in_sets
  by (metis A Diff UNIV_I image_subset_iff)
qed

```

```

lemma (in algebra) range_disjointed_sets':
  range A  $\subseteq$  M  $\implies$  range (disjointed A)  $\subseteq$  M
  using range_disjointed_sets .

```

```

lemma sigma_algebra_disjoint_iff:
  sigma_algebra  $\Omega$  M  $\iff$  algebra  $\Omega$  M  $\wedge$ 
  ( $\forall A. \text{range } A \subseteq M \implies \text{disjoint\_family } A \implies (\bigcup_{i::\text{nat}}. A i) \in M$ )
proof (auto simp add: sigma_algebra_iff)
  fix A :: nat  $\Rightarrow$  'a set
  assume M: algebra  $\Omega$  M
  and A: range A  $\subseteq$  M
  and UnA:  $\forall A. \text{range } A \subseteq M \implies \text{disjoint\_family } A \implies (\bigcup_{i::\text{nat}}. A i) \in M$ 
  hence range (disjointed A)  $\subseteq$  M  $\implies$ 

```

$disjoint_family (disjointed A) \longrightarrow$
 $(\bigcup i. disjointed A i) \in M$ **by** *blast*
hence $(\bigcup i. disjointed A i) \in M$
by (*simp add: algebra.range_disjointed_sets'[of Ω] M A disjoint_family_disjointed*)
thus $(\bigcup i::nat. A i) \in M$ **by** (*simp add: UN_disjointed_eq*)
qed

Ring generated by a semiring

definition (*in semiring_of_sets*) *generated_ring* :: 'a set set **where**
 $generated_ring = \{ \bigcup C \mid C. C \subseteq M \wedge finite\ C \wedge disjoint\ C \}$

lemma (*in semiring_of_sets*) *generated_ringE[elim?]*:
assumes $a \in generated_ring$
obtains C **where** *finite C disjoint C* $C \subseteq M$ $a = \bigcup C$
using *assms unfolding generated_ring_def* **by** *auto*

lemma (*in semiring_of_sets*) *generated_ringI[intro?]*:
assumes *finite C disjoint C* $C \subseteq M$ $a = \bigcup C$
shows $a \in generated_ring$
using *assms unfolding generated_ring_def* **by** *auto*

lemma (*in semiring_of_sets*) *generated_ringI_Basic*:
 $A \in M \implies A \in generated_ring$
by (*rule generated_ringI[of {A}] (auto simp: disjoint_def)*)

lemma (*in semiring_of_sets*) *generated_ring_disjoint_Un[intro?]*:
assumes $a: a \in generated_ring$ **and** $b: b \in generated_ring$
and $a \cap b = \{\}$
shows $a \cup b \in generated_ring$

proof –

from $a\ b$ **obtain** $C_a\ C_b$
where $C_a: finite\ C_a\ disjoint\ C_a\ C_a \subseteq M\ a = \bigcup C_a$
and $C_b: finite\ C_b\ disjoint\ C_b\ C_b \subseteq M\ b = \bigcup C_b$
using *generated_ringE* **by** *metis*

show *?thesis*

proof

from $\langle a \cap b = \{\} \rangle\ C_a\ C_b$ **show** *disjoint (C_a \cup C_b)*
by (*auto intro!: disjoint_union*)

qed (*use C_a C_b in auto*)

qed

lemma (*in semiring_of_sets*) *generated_ring_empty*: $\{\} \in generated_ring$
by (*auto simp: generated_ring_def disjoint_def*)

lemma (*in semiring_of_sets*) *generated_ring_disjoint_Union*:
assumes *finite A* **shows** $A \subseteq generated_ring \implies disjoint\ A \implies \bigcup A \in generated_ring$
using *assms* **by** (*induct A (auto simp: disjoint_def intro!: generated_ring_disjoint_Un*)

generated_ring_empty)

lemma (in *semiring_of_sets*) *generated_ring_disjoint_UNION*:
finite I \implies *disjoint* ($A \text{ ' } I$) \implies ($\bigwedge i. i \in I \implies A \ i \in \text{generated_ring}$) $\implies \bigcup (A \text{ ' } I) \in \text{generated_ring}$
 by (*intro generated_ring_disjoint_Union*) *auto*

lemma (in *semiring_of_sets*) *generated_ring_Int*:
 assumes *a*: $a \in \text{generated_ring}$ and *b*: $b \in \text{generated_ring}$
 shows $a \cap b \in \text{generated_ring}$

proof –

from *a b* obtain *Ca Cb*

where *Ca*: *finite Ca disjoint Ca Ca* $Ca \subseteq M$ $a = \bigcup Ca$

and *Cb*: *finite Cb disjoint Cb Cb* $Cb \subseteq M$ $b = \bigcup Cb$

using *generated_ringE* by *metis*

define *C* where $C = (\lambda(a,b). a \cap b) \text{ ' } (Ca \times Cb)$

show *?thesis*

proof

show *disjoint C*

proof (*simp add: disjoint_def C_def, intro ballI impI*)

fix *a1 b1 a2 b2* assume *sets*: $a1 \in Ca$ $b1 \in Cb$ $a2 \in Ca$ $b2 \in Cb$

assume $a1 \cap b1 \neq a2 \cap b2$

then have $a1 \neq a2 \vee b1 \neq b2$ by *auto*

then show $(a1 \cap b1) \cap (a2 \cap b2) = \{\}$

proof

assume $a1 \neq a2$

with *sets Ca* have $a1 \cap a2 = \{\}$

by (*auto simp: disjoint_def*)

then show *?thesis* by *auto*

next

assume $b1 \neq b2$

with *sets Cb* have $b1 \cap b2 = \{\}$

by (*auto simp: disjoint_def*)

then show *?thesis* by *auto*

qed

qed

qed (*use Ca Cb in <auto simp: C_def>*)

qed

lemma (in *semiring_of_sets*) *generated_ring_Inter*:
 assumes *finite A A* $A \neq \{\}$ shows $A \subseteq \text{generated_ring} \implies \bigcap A \in \text{generated_ring}$
 using *assms* by (*induct A rule: finite_ne_induct*) (*auto intro: generated_ring_Int*)

lemma (in *semiring_of_sets*) *generated_ring_INTER*:
finite I $\implies I \neq \{\}$ \implies ($\bigwedge i. i \in I \implies A \ i \in \text{generated_ring}$) $\implies \bigcap (A \text{ ' } I) \in \text{generated_ring}$
 by (*intro generated_ring_Inter*) *auto*

lemma (in *semiring_of_sets*) *generating_ring*:

```

ring_of_sets  $\Omega$  generated_ring
proof (rule ring_of_setsI)
let ?R = generated_ring
show ?R  $\subseteq$  Pow  $\Omega$ 
using sets_into_space by (auto simp: generated_ring_def generated_ring_empty)
show {}  $\in$  ?R by (rule generated_ring_empty)

```

```

{
fix a b assume a  $\in$  ?R b  $\in$  ?R
then obtain Ca Cb
where Ca: finite Ca disjoint Ca Ca  $\subseteq$  M a =  $\bigcup$  Ca
and Cb: finite Cb disjoint Cb Cb  $\subseteq$  M b =  $\bigcup$  Cb
using generated_ringE by metis
show a - b  $\in$  ?R
proof cases
assume Cb = {} with Cb <a  $\in$  ?R> show ?thesis
by simp
next
assume Cb  $\neq$  {}
with Ca Cb have a - b = ( $\bigcup$  a'  $\in$  Ca.  $\bigcap$  b'  $\in$  Cb. a' - b') by auto
also have ...  $\in$  ?R
proof (intro generated_ring_INTER generated_ring_disjoint_UNION)
fix a b assume a  $\in$  Ca b  $\in$  Cb
with Ca Cb Diff_cover[of a b] show a - b  $\in$  ?R
by (auto simp add: generated_ring_def)
(metis DiffI Diff_eq_empty_iff empty_iff)
next
show disjoint (( $\lambda$ a'.  $\bigcap$  b'  $\in$  Cb. a' - b') 'Ca)
using Ca by (auto simp add: disjoint_def <Cb  $\neq$  {}>)
next
show finite Ca finite Cb Cb  $\neq$  {} by fact+
qed
finally show a - b  $\in$  ?R .
qed
}
note Diff = this

```

```

fix a b assume sets: a  $\in$  ?R b  $\in$  ?R
have a  $\cup$  b = (a - b)  $\cup$  (a  $\cap$  b)  $\cup$  (b - a) by auto
also have ...  $\in$  ?R
by (intro sets generated_ring_disjoint_Un generated_ring_Int Diff) auto
finally show a  $\cup$  b  $\in$  ?R .
qed

```

lemma (in semiring_of_sets) sigma_sets_generated_ring_eq: sigma_sets Ω generated_ring = sigma_sets Ω M

```

proof
interpret M: sigma_algebra  $\Omega$  sigma_sets  $\Omega$  M
using space_closed by (rule sigma_algebra_sigma_sets)

```

```

show sigma_sets  $\Omega$  generated_ring  $\subseteq$  sigma_sets  $\Omega$  M
  by (blast intro!: sigma_sets_mono elim: generated_ringE)
qed (auto intro!: generated_ringI_Basic sigma_sets_mono)

```

A Two-Element Series

```

definition binaryset :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  nat  $\Rightarrow$  'a set
  where binaryset A B = ( $\lambda x.$  {x})(0 := A, Suc 0 := B)

```

```

lemma range_binaryset_eq: range(binaryset A B) = {A,B,{}}
  apply (simp add: binaryset_def)
  apply (rule set_eqI)
  apply (auto simp add: image_iff)
done

```

```

lemma UN_binaryset_eq: ( $\bigcup i.$  binaryset A B i) = A  $\cup$  B
  by (simp add: range_binaryset_eq cong del: SUP_cong_simp)

```

Closed CDI

```

definition closed_cdi :: 'a set  $\Rightarrow$  'a set set  $\Rightarrow$  bool where
  closed_cdi  $\Omega$  M  $\longleftrightarrow$ 
  M  $\subseteq$  Pow  $\Omega$  &
  ( $\forall s \in M.$   $\Omega - s \in M$ ) &
  ( $\forall A.$  (range A  $\subseteq$  M) & (A 0 = {}) & ( $\forall n.$  A n  $\subseteq$  A (Suc n))  $\longrightarrow$ 
    ( $\bigcup i.$  A i)  $\in$  M) &
  ( $\forall A.$  (range A  $\subseteq$  M) & disjoint_family A  $\longrightarrow$  ( $\bigcup i::nat.$  A i)  $\in$  M)

```

inductive_set

```

smallest_ccdi_sets :: 'a set  $\Rightarrow$  'a set set  $\Rightarrow$  'a set set
for  $\Omega$  M
where
  Basic [intro]:
    a  $\in$  M  $\implies$  a  $\in$  smallest_ccdi_sets  $\Omega$  M
  | Compl [intro]:
    a  $\in$  smallest_ccdi_sets  $\Omega$  M  $\implies$   $\Omega - a \in$  smallest_ccdi_sets  $\Omega$  M
  | Inc:
    range A  $\in$  Pow(smallest_ccdi_sets  $\Omega$  M)  $\implies$  A 0 = {}  $\implies$  ( $\bigwedge n.$  A n  $\subseteq$  A
(Suc n))
     $\implies$  ( $\bigcup i.$  A i)  $\in$  smallest_ccdi_sets  $\Omega$  M
  | Disj:
    range A  $\in$  Pow(smallest_ccdi_sets  $\Omega$  M)  $\implies$  disjoint_family A
     $\implies$  ( $\bigcup i::nat.$  A i)  $\in$  smallest_ccdi_sets  $\Omega$  M

```

```

lemma (in subset_class) smallest_closed_cdi1: M  $\subseteq$  smallest_ccdi_sets  $\Omega$  M
  by auto

```

```

lemma (in subset_class) smallest_ccdi_sets: smallest_ccdi_sets  $\Omega$  M  $\subseteq$  Pow  $\Omega$ 
  apply (rule subsetI)
  apply (erule smallest_ccdi_sets.induct)

```

2050

apply (*auto intro: range_subsetD dest: sets_into_space*)
done

lemma (*in subset_class*) *smallest_closed_cdi2: closed_cdi* Ω (*smallest_ccdi_sets* Ω M)

apply (*auto simp add: closed_cdi_def smallest_ccdi_sets*)
apply (*blast intro: smallest_ccdi_sets.Inc smallest_ccdi_sets.Disj*) +
done

lemma *closed_cdi_subset: closed_cdi* Ω $M \implies M \subseteq Pow$ Ω

by (*simp add: closed_cdi_def*)

lemma *closed_cdi_Compl: closed_cdi* Ω $M \implies s \in M \implies \Omega - s \in M$

by (*simp add: closed_cdi_def*)

lemma *closed_cdi_Inc:*

closed_cdi Ω $M \implies range$ $A \subseteq M \implies A \ 0 = \{\}$ $\implies (!n. A \ n \subseteq A \ (Suc \ n))$
 $\implies (\bigcup i. A \ i) \in M$

by (*simp add: closed_cdi_def*)

lemma *closed_cdi_Disj:*

closed_cdi Ω $M \implies range$ $A \subseteq M \implies disjoint_family$ $A \implies (\bigcup i::nat. A \ i) \in M$

by (*simp add: closed_cdi_def*)

lemma *closed_cdi_Un:*

assumes *cdi: closed_cdi* Ω M **and** *empty:* $\{\} \in M$

and $A: A \in M$ **and** $B: B \in M$

and *disj:* $A \cap B = \{\}$

shows $A \cup B \in M$

proof –

have *ra:* $range$ (*binaryset* $A \ B$) $\subseteq M$

by (*simp add: range_binaryset_eq empty A B*)

have *di:* *disjoint_family* (*binaryset* $A \ B$) **using** *disj*

by (*simp add: disjoint_family_on_def binaryset_def Int_commute*)

from *closed_cdi_Disj* [*OF cdi ra di*]

show *?thesis*

by (*simp add: UN_binaryset_eq*)

qed

lemma (*in algebra*) *smallest_ccdi_sets_Un:*

assumes $A: A \in smallest_ccdi_sets \ \Omega \ M$ **and** $B: B \in smallest_ccdi_sets \ \Omega \ M$

and *disj:* $A \cap B = \{\}$

shows $A \cup B \in smallest_ccdi_sets \ \Omega \ M$

proof –

have *ra:* $range$ (*binaryset* $A \ B$) $\in Pow$ (*smallest_ccdi_sets* $\Omega \ M$)

by (*simp add: range_binaryset_eq A B smallest_ccdi_sets.Basic*)

have *di:* *disjoint_family* (*binaryset* $A \ B$) **using** *disj*

by (*simp add: disjoint_family_on_def binaryset_def Int_commute*)

```

  from Disj [OF ra di]
  show ?thesis
    by (simp add: UN_binaryset_eq)
qed

lemma (in algebra) smallest_ccdi_sets_Int1:
  assumes a:  $a \in M$ 
  shows  $b \in \text{smallest\_ccdi\_sets } \Omega M \implies a \cap b \in \text{smallest\_ccdi\_sets } \Omega M$ 
proof (induct rule: smallest_ccdi_sets.induct)
  case (Basic x)
  thus ?case
    by (metis a Int smallest_ccdi_sets.Basic)
next
  case (Compl x)
  have  $a \cap (\Omega - x) = \Omega - ((\Omega - a) \cup (a \cap x))$ 
    by blast
  also have ...  $\in \text{smallest\_ccdi\_sets } \Omega M$ 
    by (metis smallest_ccdi_sets.Compl a Compl(2) Diff_Int2 Diff_Int_distrib2
      Diff_disjoint Int_Diff Int_empty_right smallest_ccdi_sets_Un
      smallest_ccdi_sets.Basic smallest_ccdi_sets.Compl)
  finally show ?case .
next
  case (Inc A)
  have 1:  $(\bigcup i. (\lambda i. a \cap A i) i) = a \cap (\bigcup i. A i)$ 
    by blast
  have range  $(\lambda i. a \cap A i) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega M)$  using Inc
    by blast
  moreover have  $(\lambda i. a \cap A i) 0 = \{\}$ 
    by (simp add: Inc)
  moreover have  $!!n. (\lambda i. a \cap A i) n \subseteq (\lambda i. a \cap A i) (\text{Suc } n)$  using Inc
    by blast
  ultimately have 2:  $(\bigcup i. (\lambda i. a \cap A i) i) \in \text{smallest\_ccdi\_sets } \Omega M$ 
    by (rule smallest_ccdi_sets.Inc)
  show ?case
    by (metis 1 2)
next
  case (Disj A)
  have 1:  $(\bigcup i. (\lambda i. a \cap A i) i) = a \cap (\bigcup i. A i)$ 
    by blast
  have range  $(\lambda i. a \cap A i) \in \text{Pow}(\text{smallest\_ccdi\_sets } \Omega M)$  using Disj
    by blast
  moreover have disjoint_family  $(\lambda i. a \cap A i)$  using Disj
    by (auto simp add: disjoint_family_on_def)
  ultimately have 2:  $(\bigcup i. (\lambda i. a \cap A i) i) \in \text{smallest\_ccdi\_sets } \Omega M$ 
    by (rule smallest_ccdi_sets.Disj)
  show ?case
    by (metis 1 2)
qed

```

lemma (in algebra) *smallest_ccdi_sets_Int*:
assumes $b: b \in \text{smallest_ccdi_sets } \Omega M$
shows $a \in \text{smallest_ccdi_sets } \Omega M \implies a \cap b \in \text{smallest_ccdi_sets } \Omega M$
proof (induct rule: *smallest_ccdi_sets.induct*)
case (*Basic x*)
thus ?case
by (*metis b smallest_ccdi_sets_Int1*)
next
case (*Compl x*)
have $(\Omega - x) \cap b = \Omega - (x \cap b \cup (\Omega - b))$
by *blast*
also have $\dots \in \text{smallest_ccdi_sets } \Omega M$
by (*metis Compl(2) Diff_disjoint Int_Diff Int_commute Int_empty_right b smallest_ccdi_sets.Compl smallest_ccdi_sets_Un*)
finally show ?case .
next
case (*Inc A*)
have $1: (\bigcup i. (\lambda i. A i \cap b) i) = (\bigcup i. A i) \cap b$
by *blast*
have range $(\lambda i. A i \cap b) \in \text{Pow}(\text{smallest_ccdi_sets } \Omega M)$ **using** *Inc*
by *blast*
moreover have $(\lambda i. A i \cap b) 0 = \{\}$
by (*simp add: Inc*)
moreover have $!!n. (\lambda i. A i \cap b) n \subseteq (\lambda i. A i \cap b) (\text{Suc } n)$ **using** *Inc*
by *blast*
ultimately have $2: (\bigcup i. (\lambda i. A i \cap b) i) \in \text{smallest_ccdi_sets } \Omega M$
by (*rule smallest_ccdi_sets.Inc*)
show ?case
by (*metis 1 2*)
next
case (*Disj A*)
have $1: (\bigcup i. (\lambda i. A i \cap b) i) = (\bigcup i. A i) \cap b$
by *blast*
have range $(\lambda i. A i \cap b) \in \text{Pow}(\text{smallest_ccdi_sets } \Omega M)$ **using** *Disj*
by *blast*
moreover have *disjoint_family* $(\lambda i. A i \cap b)$ **using** *Disj*
by (*auto simp add: disjoint_family_on_def*)
ultimately have $2: (\bigcup i. (\lambda i. A i \cap b) i) \in \text{smallest_ccdi_sets } \Omega M$
by (*rule smallest_ccdi_sets.Disj*)
show ?case
by (*metis 1 2*)
qed

lemma (in algebra) *sigma_property_disjoint_lemma*:
assumes $sbC: M \subseteq C$
and *ccdi*: *closed_cdi* ΩC
shows *sigma_sets* $\Omega M \subseteq C$
proof –


```

have smallest_ccdi_sets  $\Omega$   $M \in \{B . M \subseteq B \wedge \text{sigma\_algebra } \Omega B\}$ 
  apply (auto simp add: sigma_algebra_disjoint_iff algebra_iff_Int
    smallest_ccdi_sets_Int)
  apply (metis Union_Pow_eq Union_upper subsetD smallest_ccdi_sets)
  apply (blast intro: smallest_ccdi_sets.Disj)
done
hence sigma_sets  $(\Omega)$   $(M) \subseteq \text{smallest\_ccdi\_sets } \Omega M$ 
  by clarsimp
  (drule sigma_algebra.sigma_sets_subset [where a=M], auto)
also have ...  $\subseteq C$ 
proof
  fix x
  assume x:  $x \in \text{smallest\_ccdi\_sets } \Omega M$ 
  thus  $x \in C$ 
  proof (induct rule: smallest_ccdi_sets.induct)
    case (Basic x)
    thus ?case
      by (metis Basic subsetD sbC)
  next
    case (Compl x)
    thus ?case
      by (blast intro: closed_ccdi_Compl [OF ccdi, simplified])
  next
    case (Inc A)
    thus ?case
      by (auto intro: closed_ccdi_Inc [OF ccdi, simplified])
  next
    case (Disj A)
    thus ?case
      by (auto intro: closed_ccdi_Disj [OF ccdi, simplified])
  qed
qed
finally show ?thesis .
qed

```

```

lemma (in algebra) sigma_property_disjoint:
  assumes sbC:  $M \subseteq C$ 
    and compl:  $\forall s. s \in C \cap \text{sigma\_sets } (\Omega) (M) \implies \Omega - s \in C$ 
    and inc:  $\forall A. \text{range } A \subseteq C \cap \text{sigma\_sets } (\Omega) (M) \implies A \ 0 = \{\} \implies (\forall n. A \ n \subseteq A \ (\text{Suc } n)) \implies (\bigcup i. A \ i) \in C$ 
    and disj:  $\forall A. \text{range } A \subseteq C \cap \text{sigma\_sets } (\Omega) (M) \implies \text{disjoint\_family } A \implies (\bigcup i::\text{nat}. A \ i) \in C$ 
  shows sigma_sets  $(\Omega)$   $(M) \subseteq C$ 
proof -
  have sigma_sets  $(\Omega)$   $(M) \subseteq C \cap \text{sigma\_sets } (\Omega) (M)$ 
  proof (rule sigma_property_disjoint_lemma)
    show  $M \subseteq C \cap \text{sigma\_sets } (\Omega) (M)$ 
    by (metis Int_greatest Set.subsetI sbC sigma_sets.Basic)
  qed

```

```

next
  show closed_cdi  $\Omega$  ( $C \cap \text{sigma\_sets } (\Omega) (M)$ )
  by (simp add: closed_cdi_def compl inc disj)
    (metis PowI Set.subsetI le_infI2 sigma_sets_into_sp space_closed
      IntE sigma_sets.Compl range_subsetD sigma_sets.Union)
qed
thus ?thesis
  by blast
qed

```

Dynkin systems

```

locale Dynkin_system = subset_class +
  assumes space:  $\Omega \in M$ 
  and compl[intro!]:  $\bigwedge A. A \in M \implies \Omega - A \in M$ 
  and UN[intro!]:  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq M$ 
     $\implies (\bigcup i::\text{nat. } A \ i) \in M$ 

```

```

lemma (in Dynkin_system) empty[intro, simp]:  $\{\} \in M$ 
  using space compl[of  $\Omega$ ] by simp

```

```

lemma (in Dynkin_system) diff:
  assumes sets:  $D \in M \ E \in M$  and  $D \subseteq E$ 
  shows  $E - D \in M$ 

```

proof –

```

let ?f =  $\lambda x. \text{if } x = 0 \text{ then } D \text{ else if } x = \text{Suc } 0 \text{ then } \Omega - E \text{ else } \{\}$ 

```

```

have range ?f =  $\{D, \Omega - E, \{\}$ 

```

```

  by (auto simp: image_iff)

```

```

moreover have  $D \cup (\Omega - E) = (\bigcup i. ?f \ i)$ 

```

```

  by (auto simp: image_iff split: if_split_asm)

```

moreover

```

have disjoint_family ?f unfolding disjoint_family_on_def

```

```

  using  $\langle D \in M \rangle [THEN \text{sets\_into\_space}] \langle D \subseteq E \rangle$  by auto

```

```

ultimately have  $\Omega - (D \cup (\Omega - E)) \in M$ 

```

```

  using sets UN by auto fastforce

```

```

also have  $\Omega - (D \cup (\Omega - E)) = E - D$ 

```

```

  using assms sets_into_space by auto

```

```

finally show ?thesis .

```

qed

```

lemma Dynkin_systemI:

```

```

  assumes  $\bigwedge A. A \in M \implies A \subseteq \Omega$   $\Omega \in M$ 

```

```

  assumes  $\bigwedge A. A \in M \implies \Omega - A \in M$ 

```

```

  assumes  $\bigwedge A. \text{disjoint\_family } A \implies \text{range } A \subseteq M$ 

```

```

     $\implies (\bigcup i::\text{nat. } A \ i) \in M$ 

```

```

  shows Dynkin_system  $\Omega \ M$ 

```

```

  using assms by (auto simp: Dynkin_system_def Dynkin_system_axioms_def
    subset_class_def)

```

lemma *Dynkin_systemI'*:
assumes 1: $\bigwedge A. A \in M \implies A \subseteq \Omega$
assumes *empty*: $\{\} \in M$
assumes *Diff*: $\bigwedge A. A \in M \implies \Omega - A \in M$
assumes 2: $\bigwedge A. \text{disjoint_family } A \implies \text{range } A \subseteq M$
 $\implies (\bigcup i::\text{nat. } A \ i) \in M$
shows *Dynkin_system* Ω M
proof –
from *Diff* [*OF empty*] **have** $\Omega \in M$ **by** *auto*
from 1 *this Diff* 2 **show** ?*thesis*
by (*intro Dynkin_systemI*) *auto*
qed

lemma *Dynkin_system_trivial*:
shows *Dynkin_system* A (*Pow A*)
by (*rule Dynkin_systemI*) *auto*

lemma *sigma_algebra_imp_Dynkin_system*:
assumes *sigma_algebra* Ω M **shows** *Dynkin_system* Ω M
proof –
interpret *sigma_algebra* Ω M **by** *fact*
show ?*thesis* **using** *sets_into_space* **by** (*fastforce intro!*: *Dynkin_systemI*)
qed

Intersection sets systems

definition *Int_stable* :: 'a set set \implies bool **where**
Int_stable $M \iff (\forall a \in M. \forall b \in M. a \cap b \in M)$

lemma (*in algebra*) *Int_stable*: *Int_stable* M
unfolding *Int_stable_def* **by** *auto*

lemma *Int_stableI_image*:
 $(\bigwedge i \ j. i \in I \implies j \in I \implies \exists k \in I. A \ i \cap A \ j = A \ k) \implies \text{Int_stable } (A \ ' I)$
by (*auto simp: Int_stable_def image_def*)

lemma *Int_stableI*:
 $(\bigwedge a \ b. a \in A \implies b \in A \implies a \cap b \in A) \implies \text{Int_stable } A$
unfolding *Int_stable_def* **by** *auto*

lemma *Int_stableD*:
 $\text{Int_stable } M \implies a \in M \implies b \in M \implies a \cap b \in M$
unfolding *Int_stable_def* **by** *auto*

lemma (*in Dynkin_system*) *sigma_algebra_eq_Int_stable*:
 $\text{sigma_algebra } \Omega \ M \iff \text{Int_stable } M$
proof
assume *sigma_algebra* Ω M **then show** *Int_stable* M
unfolding *sigma_algebra_def* **using** *algebra.Int_stable* **by** *auto*

```

next
  assume Int_stable M
  show sigma_algebra Ω M
    unfolding sigma_algebra_disjoint_iff algebra_iff_Un
  proof (intro conjI ballI allI impI)
    show M ⊆ Pow (Ω) using sets_into_space by auto
  next
    fix A B assume A ∈ M B ∈ M
    then have A ∪ B = Ω - ((Ω - A) ∩ (Ω - B))
      Ω - A ∈ M Ω - B ∈ M
      using sets_into_space by auto
    then show A ∪ B ∈ M
      using ⟨Int_stable M⟩ unfolding Int_stable_def by auto
  qed auto
qed

```

Smallest Dynkin systems

definition *Dynkin* :: 'a set ⇒ 'a set set ⇒ 'a set set **where**
Dynkin Ω M = (∩ {D. *Dynkin_system* Ω D ∧ M ⊆ D})

```

lemma Dynkin_system_Dynkin:
  assumes M ⊆ Pow (Ω)
  shows Dynkin_system Ω (Dynkin Ω M)
proof (rule Dynkin_systemI)
  fix A assume A ∈ Dynkin Ω M
  moreover
  { fix D assume A ∈ D and d: Dynkin_system Ω D
    then have A ⊆ Ω by (auto simp: Dynkin_system_def subset_class_def) }
  moreover have {D. Dynkin_system Ω D ∧ M ⊆ D} ≠ {}
    using assms Dynkin_system_trivial by fastforce
  ultimately show A ⊆ Ω
    unfolding Dynkin_def using assms
    by auto
  next
  show Ω ∈ Dynkin Ω M
    unfolding Dynkin_def using Dynkin_system.space by fastforce
  next
  fix A assume A ∈ Dynkin Ω M
  then show Ω - A ∈ Dynkin Ω M
    unfolding Dynkin_def using Dynkin_system.compl by force
  next
  fix A :: nat ⇒ 'a set
  assume A: disjoint_family A range A ⊆ Dynkin Ω M
  show (∪ i. A i) ∈ Dynkin Ω M unfolding Dynkin_def
  proof (simp, safe)
    fix D assume Dynkin_system Ω D M ⊆ D
    with A have (∪ i. A i) ∈ D
      by (intro Dynkin_system.UN) (auto simp: Dynkin_def)
  
```

```

    then show  $(\bigcup i. A i) \in D$  by auto
  qed
qed

```

```

lemma Dynkin_Basic[intro]:  $A \in M \implies A \in \text{Dynkin } \Omega M$ 
  unfolding Dynkin_def by auto

```

```

lemma (in Dynkin_system) restricted_Dynkin_system:

```

```

  assumes  $D \in M$ 
  shows Dynkin_system  $\Omega \{Q. Q \subseteq \Omega \wedge Q \cap D \in M\}$ 
proof (rule Dynkin_systemI, simp_all)
  have  $\Omega \cap D = D$ 
    using  $\langle D \in M \rangle$  sets_into_space by auto
  then show  $\Omega \cap D \in M$ 
    using  $\langle D \in M \rangle$  by auto

```

```

next

```

```

  fix A assume  $A \subseteq \Omega \wedge A \cap D \in M$ 
  moreover have  $(\Omega - A) \cap D = (\Omega - (A \cap D)) - (\Omega - D)$ 
    by auto
  ultimately show  $(\Omega - A) \cap D \in M$ 
    using  $\langle D \in M \rangle$  by (auto intro: diff)

```

```

next

```

```

  fix A :: nat  $\Rightarrow$  'a set
  assume disjoint_family A range  $A \subseteq \{Q. Q \subseteq \Omega \wedge Q \cap D \in M\}$ 
  then have  $\bigwedge i. A i \subseteq \Omega$  disjoint_family  $(\lambda i. A i \cap D)$ 
    range  $(\lambda i. A i \cap D) \subseteq M$   $(\bigcup x. A x) \cap D = (\bigcup x. A x \cap D)$ 
    by ((fastforce simp: disjoint_family_on_def)+)
  then show  $(\bigcup x. A x) \subseteq \Omega \wedge (\bigcup x. A x) \cap D \in M$ 
    by (auto simp del: UN_simps)

```

```

qed

```

```

lemma (in Dynkin_system) Dynkin_subset:

```

```

  assumes  $N \subseteq M$ 
  shows Dynkin  $\Omega N \subseteq M$ 
proof -
  have Dynkin_system  $\Omega M$  ..
  then have Dynkin_system  $\Omega M$ 
    using assms unfolding Dynkin_system_def Dynkin_system_axioms_def sub-
    set_class_def by simp
  with  $\langle N \subseteq M \rangle$  show ?thesis by (auto simp add: Dynkin_def)

```

```

qed

```

```

lemma sigma_eq_Dynkin:

```

```

  assumes sets:  $M \subseteq \text{Pow } \Omega$ 
  assumes Int_stable M
  shows sigma_sets  $\Omega M = \text{Dynkin } \Omega M$ 
proof -
  have Dynkin  $\Omega M \subseteq \text{sigma\_sets } (\Omega) (M)$ 
    using sigma_algebra_imp_Dynkin_system

```

```

    unfolding Dynkin_def sigma_sets_least_sigma_algebra[OF sets] by auto
  moreover
  interpret Dynkin_system  $\Omega$  Dynkin  $\Omega$  M
    using Dynkin_system_Dynkin[OF sets] .
  have sigma_algebra  $\Omega$  (Dynkin  $\Omega$  M)
    unfolding sigma_algebra_eq_Int_stable_Int_stable_def
  proof (intro ballI)
    fix A B assume A  $\in$  Dynkin  $\Omega$  M B  $\in$  Dynkin  $\Omega$  M
    let ?D =  $\lambda E. \{Q. Q \subseteq \Omega \wedge Q \cap E \in \text{Dynkin } \Omega \text{ } M\}$ 
    have M  $\subseteq$  ?D B
    proof
      fix E assume E  $\in$  M
      then have M  $\subseteq$  ?D E E  $\in$  Dynkin  $\Omega$  M
        using sets_into_space  $\langle$ Int_stable M $\rangle$  by (auto simp: Int_stable_def)
      then have Dynkin  $\Omega$  M  $\subseteq$  ?D E
        using restricted_Dynkin_system  $\langle$ E  $\in$  Dynkin  $\Omega$  M $\rangle$ 
        by (intro Dynkin_system.Dynkin_subset) simp_all
      then have B  $\in$  ?D E
        using  $\langle$ B  $\in$  Dynkin  $\Omega$  M $\rangle$  by auto
      then have E  $\cap$  B  $\in$  Dynkin  $\Omega$  M
        by (subst Int_commute) simp
      then show E  $\in$  ?D B
        using sets  $\langle$ E  $\in$  M $\rangle$  by auto
    qed
    then have Dynkin  $\Omega$  M  $\subseteq$  ?D B
      using restricted_Dynkin_system  $\langle$ B  $\in$  Dynkin  $\Omega$  M $\rangle$ 
      by (intro Dynkin_system.Dynkin_subset) simp_all
    then show A  $\cap$  B  $\in$  Dynkin  $\Omega$  M
      using  $\langle$ A  $\in$  Dynkin  $\Omega$  M $\rangle$  sets_into_space by auto
  qed
  from sigma_algebra.sigma_sets_subset[OF this, of M]
  have sigma_sets ( $\Omega$ ) (M)  $\subseteq$  Dynkin  $\Omega$  M by auto
  ultimately have sigma_sets ( $\Omega$ ) (M) = Dynkin  $\Omega$  M by auto
  then show ?thesis
    by (auto simp: Dynkin_def)
qed

lemma (in Dynkin_system) Dynkin_idem:
  Dynkin  $\Omega$  M = M
proof -
  have Dynkin  $\Omega$  M = M
  proof
    show M  $\subseteq$  Dynkin  $\Omega$  M
      using Dynkin_Basic by auto
    show Dynkin  $\Omega$  M  $\subseteq$  M
      by (intro Dynkin_subset) auto
  qed
  then show ?thesis
    by (auto simp: Dynkin_def)

```

qed

lemma (in *Dynkin_system*) *Dynkin_lemma*:
assumes *Int_stable E*
and *E*: $E \subseteq M$ $M \subseteq \text{sigma_sets } \Omega E$
shows $\text{sigma_sets } \Omega E = M$
proof –
have $E \subseteq \text{Pow } \Omega$
using *E sets_into_space* **by** *force*
then have $*$: $\text{sigma_sets } \Omega E = \text{Dynkin } \Omega E$
using $\langle \text{Int_stable } E \rangle$ **by** (rule *sigma_eq_Dynkin*)
then have $\text{Dynkin } \Omega E = M$
using *assms Dynkin_subset[OF E(1)]* **by** *simp*
with $*$ **show** *?thesis*
using *assms* **by** (auto *simp: Dynkin_def*)

qed

Induction rule for intersection-stable generators

The reason to introduce Dynkin-systems is the following induction rules for σ -algebras generated by a generator closed under intersection.

proposition *sigma_sets_induct_disjoint*[*consumes 3, case_names basic empty compl union*]:

assumes *Int_stable G*
and *closed*: $G \subseteq \text{Pow } \Omega$
and *A*: $A \in \text{sigma_sets } \Omega G$
assumes *basic*: $\bigwedge A. A \in G \implies P A$
and *empty*: $P \{\}$
and *compl*: $\bigwedge A. A \in \text{sigma_sets } \Omega G \implies P A \implies P (\Omega - A)$
and *union*: $\bigwedge A. \text{disjoint_family } A \implies \text{range } A \subseteq \text{sigma_sets } \Omega G \implies (\bigwedge i. P (A i)) \implies P (\bigcup i::\text{nat}. A i)$
shows $P A$

proof –

let $?D = \{ A \in \text{sigma_sets } \Omega G. P A \}$
interpret *sigma_algebra* $\Omega \text{sigma_sets } \Omega G$
using *closed* **by** (rule *sigma_algebra_sigma_sets*)
from *compl[OF _ empty]* *closed* **have** *space*: $P \Omega$ **by** *simp*
interpret *Dynkin_system* $\Omega ?D$
by *standard* (auto *dest: sets_into_space intro!: space compl union*)
have $\text{sigma_sets } \Omega G = ?D$
by (rule *Dynkin_lemma*) (auto *simp: basic* $\langle \text{Int_stable } G \rangle$)
with *A* **show** *?thesis* **by** *auto*

qed

7.1.2 Measure type

definition *positive* :: $'a \text{ set } \text{set} \Rightarrow ('a \text{ set} \Rightarrow \text{ennreal}) \Rightarrow \text{bool}$ **where**
positive $M \mu \longleftrightarrow \mu \{\} = 0$

definition *countably_additive* :: 'a set set \Rightarrow ('a set \Rightarrow ennreal) \Rightarrow bool **where**
countably_additive $M f \longleftrightarrow$
 $(\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint_family } A \longrightarrow (\bigcup i. A i) \in M \longrightarrow$
 $(\sum i. f (A i)) = f (\bigcup i. A i))$

definition *measure_space* :: 'a set \Rightarrow 'a set set \Rightarrow ('a set \Rightarrow ennreal) \Rightarrow bool
where
measure_space $\Omega A \mu \longleftrightarrow$
 $\text{sigma_algebra } \Omega A \wedge \text{positive } A \mu \wedge \text{countably_additive } A \mu$

typedef 'a measure =
 $\{(\Omega :: 'a \text{ set}, A, \mu). (\forall a \in -A. \mu a = 0) \wedge \text{measure_space } \Omega A \mu \}$

proof

have *sigma_algebra* UNIV $\{\{\}, \text{UNIV}\}$
by (*auto simp: sigma_algebra_iff2*)
then show (UNIV, $\{\{\}, \text{UNIV}\}$, $\lambda A. 0$) $\in \{(\Omega, A, \mu). (\forall a \in -A. \mu a = 0) \wedge$
measure_space $\Omega A \mu\}$
by (*auto simp: measure_space_def positive_def countably_additive_def*)
qed

definition *space* :: 'a measure \Rightarrow 'a set **where**
space $M = \text{fst } (\text{Rep_measure } M)$

definition *sets* :: 'a measure \Rightarrow 'a set set **where**
sets $M = \text{fst } (\text{snd } (\text{Rep_measure } M))$

definition *emeasure* :: 'a measure \Rightarrow 'a set \Rightarrow ennreal **where**
emeasure $M = \text{snd } (\text{snd } (\text{Rep_measure } M))$

definition *measure* :: 'a measure \Rightarrow 'a set \Rightarrow real **where**
measure $M A = \text{enn2real } (\text{emeasure } M A)$

declare $[[\text{coercion } \text{sets}]]$

declare $[[\text{coercion } \text{measure}]]$

declare $[[\text{coercion } \text{emeasure}]]$

lemma *measure_space*: *measure_space* (*space* M) (*sets* M) (*emeasure* M)
by (*cases* M) (*auto simp: space_def sets_def emeasure_def Abs_measure_inverse*)

interpretation *sets*: *sigma_algebra* *space* M *sets* M **for** $M :: 'a \text{ measure}$
using *measure_space*[of M] **by** (*auto simp: measure_space_def*)

definition *measure_of* :: 'a set \Rightarrow 'a set set \Rightarrow ('a set \Rightarrow ennreal) \Rightarrow 'a measure
where
measure_of $\Omega A \mu =$
 $\text{Abs_measure } (\Omega, \text{if } A \subseteq \text{Pow } \Omega \text{ then } \text{sigma_sets } \Omega A \text{ else } \{\{\}, \Omega\},$
 $\lambda a. \text{if } a \in \text{sigma_sets } \Omega A \wedge \text{measure_space } \Omega (\text{sigma_sets } \Omega A) \mu \text{ then } \mu a$

else 0)

abbreviation $\text{sigma } \Omega A \equiv \text{measure_of } \Omega A (\lambda x. 0)$

lemma $\text{measure_space_0}: A \subseteq \text{Pow } \Omega \implies \text{measure_space } \Omega (\text{sigma_sets } \Omega A)$
 $(\lambda x. 0)$

unfolding measure_space_def

by (auto intro!: $\text{sigma_algebra_sigma_sets simp: positive_def countably_additive_def}$)

lemma $\text{sigma_algebra_trivial}: \text{sigma_algebra } \Omega \{\{\}, \Omega\}$

by $\text{unfold_locales}(\text{fastforce intro: } \text{exI}[\mathbf{where } x=\{\{\}\}] \text{ exI}[\mathbf{where } x=\{\Omega\}])+$

lemma $\text{measure_space_0'}: \text{measure_space } \Omega \{\{\}, \Omega\} (\lambda x. 0)$

by ($\text{simp add: measure_space_def positive_def countably_additive_def sigma_algebra_trivial}$)

lemma $\text{measure_space_closed}$:

assumes $\text{measure_space } \Omega M \mu$

shows $M \subseteq \text{Pow } \Omega$

proof –

interpret $\text{sigma_algebra } \Omega M$ **using** assms **by** ($\text{simp add: measure_space_def}$)

show $?thesis$ **by** (rule space_closed)

qed

lemma (in ring_of_sets) positive_cong_eq :

$(\bigwedge a. a \in M \implies \mu' a = \mu a) \implies \text{positive } M \mu' = \text{positive } M \mu$

by ($\text{auto simp add: positive_def}$)

lemma (in sigma_algebra) $\text{countably_additive_eq}$:

$(\bigwedge a. a \in M \implies \mu' a = \mu a) \implies \text{countably_additive } M \mu' = \text{countably_additive } M \mu$

unfolding $\text{countably_additive_def}$

by ($\text{intro arg_cong}[\mathbf{where } f=\text{All}] \text{ ext}$) ($\text{auto simp add: countably_additive_def subset_eq}$)

lemma measure_space_eq :

assumes $\text{closed}: A \subseteq \text{Pow } \Omega$ **and** $\text{eq}: \bigwedge a. a \in \text{sigma_sets } \Omega A \implies \mu a = \mu' a$

shows $\text{measure_space } \Omega (\text{sigma_sets } \Omega A) \mu = \text{measure_space } \Omega (\text{sigma_sets } \Omega A) \mu'$

proof –

interpret $\text{sigma_algebra } \Omega \text{sigma_sets } \Omega A$ **using** closed **by** ($\text{rule sigma_algebra_sigma_sets}$)

from $\text{positive_cong_eq}[\text{OF eq, of } \lambda i. i]$ $\text{countably_additive_eq}[\text{OF eq, of } \lambda i. i]$

show $?thesis$

by ($\text{auto simp: measure_space_def}$)

qed

lemma measure_of_eq :

assumes $\text{closed}: A \subseteq \text{Pow } \Omega$ **and** $\text{eq}: \bigwedge a. a \in \text{sigma_sets } \Omega A \implies \mu a = \mu' a$

shows $\text{measure_of } \Omega A \mu = \text{measure_of } \Omega A \mu'$

proof –

2062

```
  have measure_space  $\Omega$  (sigma_sets  $\Omega$  A)  $\mu$  = measure_space  $\Omega$  (sigma_sets  $\Omega$ 
A)  $\mu'$ 
  using assms by (rule measure_space_eq)
  with eq show ?thesis
  by (auto simp add: measure_of_def intro!: arg_cong[where f=Abs_measure])
qed
```

```
lemma measure_space_Pow_eq:
  assumes  $\bigwedge X. X \in Pow \Omega \implies \mu X = \mu' X$ 
  shows measure_space  $\Omega$  (Pow  $\Omega$ )  $\mu$  = measure_space  $\Omega$  (Pow  $\Omega$ )  $\mu'$ 
  by (smt (verit, best) assms measure_space_eq sigma_algebra.sigma_sets_eq
sigma_algebra_Pow subset_eq)
```

```
lemma
  shows space_measure_of_conv: space (measure_of  $\Omega$  A  $\mu$ ) =  $\Omega$  (is ?space)
  and sets_measure_of_conv:
    sets (measure_of  $\Omega$  A  $\mu$ ) = (if  $A \subseteq Pow \Omega$  then sigma_sets  $\Omega$  A else  $\{\{\}, \Omega\}$ )
(is ?sets)
  and emeasure_measure_of_conv:
    emeasure (measure_of  $\Omega$  A  $\mu$ ) =
    ( $\lambda B. if B \in sigma\_sets \Omega A \wedge measure\_space \Omega (sigma\_sets \Omega A) \mu$  then  $\mu B$ 
else 0) (is ?emeasure)
proof -
  have ?space  $\wedge$  ?sets  $\wedge$  ?emeasure
  proof (cases measure_space  $\Omega$  (sigma_sets  $\Omega$  A)  $\mu$ )
  case True
    from measure_space_closed[OF this] sigma_sets_superset_generator[of A  $\Omega$ ]
    have  $A \subseteq Pow \Omega$  by simp
    hence measure_space  $\Omega$  (sigma_sets  $\Omega$  A)  $\mu$  = measure_space  $\Omega$  (sigma_sets
 $\Omega$  A)
      ( $\lambda a. if a \in sigma\_sets \Omega A$  then  $\mu a$  else 0)
    by (rule measure_space_eq) auto
  with True  $\langle A \subseteq Pow \Omega \rangle$  show ?thesis
  by (simp add: measure_of_def space_def sets_def emeasure_def Abs_measure_inverse)
  next
  case False thus ?thesis
    by (cases  $A \subseteq Pow \Omega$ ) (simp_all add: Abs_measure_inverse measure_of_def
sets_def space_def emeasure_def measure_space_0 measure_space_0')
  qed
  thus ?space ?sets ?emeasure by simp_all
qed
```

```
lemma [simp]:
  assumes A:  $A \subseteq Pow \Omega$ 
  shows sets_measure_of: sets (measure_of  $\Omega$  A  $\mu$ ) = sigma_sets  $\Omega$  A
  and space_measure_of: space (measure_of  $\Omega$  A  $\mu$ ) =  $\Omega$ 
using assms
by (simp_all add: sets_measure_of_conv space_measure_of_conv)
```

lemma *space_in_measure_of*[simp]: $\Omega \in \text{sets } (\text{measure_of } \Omega \ M \ \mu)$
by (*subst sets_measure_of_conv*) (*auto simp: sigma_sets_top*)

lemma (**in** *sigma_algebra*) *sets_measure_of_eq*[simp]: $\text{sets } (\text{measure_of } \Omega \ M \ \mu)$
 $= M$
using *space_closed* **by** (*auto intro!: sigma_sets_eq*)

lemma (**in** *sigma_algebra*) *space_measure_of_eq*[simp]: $\text{space } (\text{measure_of } \Omega \ M \ \mu) = \Omega$
by (*rule space_measure_of_conv*)

lemma *measure_of_subset*: $M \subseteq \text{Pow } \Omega \implies M' \subseteq M \implies \text{sets } (\text{measure_of } \Omega \ M' \ \mu) \subseteq \text{sets } (\text{measure_of } \Omega \ M \ \mu')$
by (*auto intro!: sigma_sets_subseteq*)

lemma *emeasure_sigma*: $\text{emeasure } (\text{sigma } \Omega \ A) = (\lambda x. 0)$
unfolding *measure_of_def emeasure_def*
by (*subst Abs_measure_inverse*)
(auto simp: measure_space_def positive_def countably_additive_def intro!: sigma_algebra_sigma_sets sigma_algebra_trivial)

lemma *sigma_sets_mono''*:
assumes $A \in \text{sigma_sets } C \ D$
assumes $B \subseteq D$
assumes $D \subseteq \text{Pow } C$
shows $\text{sigma_sets } A \ B \subseteq \text{sigma_sets } C \ D$
proof
fix x **assume** $x \in \text{sigma_sets } A \ B$
thus $x \in \text{sigma_sets } C \ D$
proof *induct*
case (*Basic a*) **with** *assms* **have** $a \in D$ **by** *auto*
thus *?case ..*
next
case *Empty* **show** *?case* **by** (*rule sigma_sets.Empty*)
next
from *assms* **have** $A \in \text{sets } (\text{sigma } C \ D)$ **by** (*subst sets_measure_of[OF ‹D ⊆ Pow C›]*)
moreover **case** (*Compl a*) **hence** $a \in \text{sets } (\text{sigma } C \ D)$ **by** (*subst sets_measure_of[OF ‹D ⊆ Pow C›]*)
ultimately **have** $A - a \in \text{sets } (\text{sigma } C \ D)$ **..**
thus *?case* **by** (*subst (asm) sets_measure_of[OF ‹D ⊆ Pow C›]*)
next
case (*Union a*)
thus *?case* **by** (*intro sigma_sets.Union*)
qed
qed

lemma *in_measure_of*[*intro, simp*]: $M \subseteq \text{Pow } \Omega \implies A \in M \implies A \in \text{sets } (\text{measure_of } \Omega \ M \ \mu)$

2064

by *auto*

lemma *space_empty_iff*: $\text{space } N = \{\} \longleftrightarrow \text{sets } N = \{\{\}\}$
by (*metis Pow_empty Sup_bot_conv(1) cSup_singleton empty_iff*
sets.sigma_sets_eq sets.space_closed sigma_sets_top subset_singletonD)

Constructing simple 'a measure

proposition *emeasure_measure_of*:

assumes *M*: $M = \text{measure_of } \Omega \ A \ \mu$

assumes *ms*: $A \subseteq \text{Pow } \Omega$ *positive* (*sets M*) μ *countably_additive* (*sets M*) μ

assumes *X*: $X \in \text{sets } M$

shows *emeasure M X* = $\mu \ X$

proof –

interpret *sigma_algebra* Ω *sigma_sets* $\Omega \ A$ **by** (*rule sigma_algebra_sigma_sets*)
fact

have *measure_space* Ω (*sigma_sets* $\Omega \ A$) μ

using *ms M* **by** (*simp add: measure_space_def sigma_algebra_sigma_sets*)

thus *?thesis using X ms*

by(*simp add: M emeasure_measure_of_conv sets_measure_of_conv*)

qed

lemma *emeasure_measure_of_sigma*:

assumes *ms*: *sigma_algebra* $\Omega \ M$ *positive* $M \ \mu$ *countably_additive* $M \ \mu$

assumes *A*: $A \in M$

shows *emeasure* (*measure_of* $\Omega \ M \ \mu$) *A* = $\mu \ A$

proof –

interpret *sigma_algebra* $\Omega \ M$ **by** *fact*

have *measure_space* Ω (*sigma_sets* $\Omega \ M$) μ

using *ms sigma_sets_eq* **by** (*simp add: measure_space_def*)

thus *?thesis* **by**(*simp add: emeasure_measure_of_conv A*)

qed

lemma *measure_cases*[*cases type: measure*]:

obtains (*measure*) $\Omega \ A \ \mu$ **where** $x = \text{Abs_measure } (\Omega, A, \mu) \ \forall a \in -A. \ \mu \ a = 0$
measure_space $\Omega \ A \ \mu$

by *atomize_elim* (*cases x, auto*)

lemma *sets_le_imp_space_le*: $\text{sets } A \subseteq \text{sets } B \implies \text{space } A \subseteq \text{space } B$

by (*auto dest: sets_sets_into_space*)

lemma *sets_eq_imp_space_eq*: $\text{sets } M = \text{sets } M' \implies \text{space } M = \text{space } M'$

by (*auto intro!: antisym sets_le_imp_space_le*)

lemma *emeasure_notin_sets*: $A \notin \text{sets } M \implies \text{emeasure } M \ A = 0$

by (*cases M*) (*auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def*)

lemma *emeasure_neq_0_sets*: $\text{emeasure } M \ A \neq 0 \implies A \in \text{sets } M$

using *emeasure_notin_sets*[of A M] by *blast*

lemma *measure_notin_sets*: $A \notin \text{sets } M \implies \text{measure } M A = 0$
 by (*simp add: measure_def emeasure_notin_sets zero_ennreal.rep_eq*)

lemma *measure_eqI*:

fixes $M N :: 'a \text{ measure}$

assumes $\text{sets } M = \text{sets } N$ **and** $\text{eq}: \bigwedge A. A \in \text{sets } M \implies \text{emeasure } M A = \text{emeasure } N A$

shows $M = N$

proof (*cases M N rule: measure_cases[case_product measure_cases]*)

case (*measure_measure* $\Omega A \mu \Omega' A' \mu'$)

interpret M : *sigma_algebra* ΩA **using** *measure_measure* **by** (*auto simp: measure_space_def*)

interpret N : *sigma_algebra* $\Omega' A'$ **using** *measure_measure* **by** (*auto simp: measure_space_def*)

have $A = \text{sets } M A' = \text{sets } N$

using *measure_measure* **by** (*simp_all add: sets_def Abs_measure_inverse*)

with $\langle \text{sets } M = \text{sets } N \rangle$ **have** $AA': A = A'$ **by** *simp*

moreover from $M.\text{top } N.\text{top } M.\text{space_closed } N.\text{space_closed } AA'$ **have** $\Omega = \Omega'$

by *auto*

moreover { **fix** B **have** $\mu B = \mu' B$

proof *cases*

assume $B \in A$

with $\text{eq } \langle A = \text{sets } M \rangle$ **have** $\text{emeasure } M B = \text{emeasure } N B$ **by** *simp*

with *measure_measure* **show** $\mu B = \mu' B$

by (*simp add: emeasure_def Abs_measure_inverse*)

next

assume $B \notin A$

with $\langle A = \text{sets } M \rangle \langle A' = \text{sets } N \rangle \langle A = A' \rangle$ **have** $B \notin \text{sets } M B \notin \text{sets } N$

by *auto*

then have $\text{emeasure } M B = 0 \text{ emeasure } N B = 0$

by (*simp_all add: emeasure_notin_sets*)

with *measure_measure* **show** $\mu B = \mu' B$

by (*simp add: emeasure_def Abs_measure_inverse*)

qed }

then have $\mu = \mu'$ **by** *auto*

ultimately show $M = N$

by (*simp add: measure_measure*)

qed

lemma *sigma_eqI*:

assumes [*simp*]: $M \subseteq \text{Pow } \Omega N \subseteq \text{Pow } \Omega$ *sigma_sets* $\Omega M = \text{sigma_sets } \Omega N$

shows *sigma* $\Omega M = \text{sigma } \Omega N$

by (*rule measure_eqI*) (*simp_all add: emeasure_sigma*)

Measurable functions

definition *measurable* :: $'a \text{ measure} \Rightarrow 'b \text{ measure} \Rightarrow ('a \Rightarrow 'b) \text{ set}$

(**infixr** \rightarrow_M 60) **where**
measurable $A\ B = \{f \in \text{space } A \rightarrow \text{space } B. \forall y \in \text{sets } B. f\ -' y \cap \text{space } A \in \text{sets } A\}$

lemma *measurableI*:

$(\bigwedge x. x \in \text{space } M \implies f\ x \in \text{space } N) \implies (\bigwedge A. A \in \text{sets } N \implies f\ -' A \cap \text{space } M \in \text{sets } M) \implies$
 $f \in \text{measurable } M\ N$
by (*auto simp: measurable_def*)

lemma *measurable_space*:

$f \in \text{measurable } M\ A \implies x \in \text{space } M \implies f\ x \in \text{space } A$
unfolding *measurable_def* **by** *auto*

lemma *measurable_sets*:

$f \in \text{measurable } M\ A \implies S \in \text{sets } A \implies f\ -' S \cap \text{space } M \in \text{sets } M$
unfolding *measurable_def* **by** *auto*

lemma *measurable_sets_Collect*:

assumes $f: f \in \text{measurable } M\ N$ **and** $P: \{x \in \text{space } N. P\ x\} \in \text{sets } N$ **shows**
 $\{x \in \text{space } M. P\ (f\ x)\} \in \text{sets } M$

proof -

have $f\ -' \{x \in \text{space } N. P\ x\} \cap \text{space } M = \{x \in \text{space } M. P\ (f\ x)\}$

using *measurable_space[OF f]* **by** *auto*

with *measurable_sets[OF f P]* **show** *?thesis*

by *simp*

qed

lemma *measurable_sigma_sets*:

assumes $B: \text{sets } N = \text{sigma_sets } \Omega\ A\ A \subseteq \text{Pow } \Omega$

and $f: f \in \text{space } M \rightarrow \Omega$

and $ba: \bigwedge y. y \in A \implies (f\ -' y) \cap \text{space } M \in \text{sets } M$

shows $f \in \text{measurable } M\ N$

proof -

interpret $A: \text{sigma_algebra } \Omega\ \text{sigma_sets } \Omega\ A$ **using** $B(2)$ **by** (*rule sigma_algebra_sigma_sets*)

from $B.\text{sets.top[of } N\]\ A.\text{top sets.space_closed[of } N\]\ A.\text{space_closed}$ **have** $\Omega: \Omega$

$= \text{space } N$ **by** *force*

{ fix X **assume** $X \in \text{sigma_sets } \Omega\ A$

then have $f\ -' X \cap \text{space } M \in \text{sets } M \wedge X \subseteq \Omega$

proof *induct*

case (*Basic a*) **then show** *?case*

by (*auto simp add: ba*) (*metis B(2) subsetD PowD*)

next

case (*Compl a*)

have [*simp*]: $f\ -' \Omega \cap \text{space } M = \text{space } M$

by (*auto simp add: funcset_mem [OF f]*)

then show *?case*

by (*auto simp add: vimage_Diff Diff_Int_distrib2 sets.compl_sets Compl*)

```

next
  case (Union a)
  then show ?case
    by (simp add: vimage_UN, simp only: UN_extend_simps(4)) blast
  qed auto }
with f show ?thesis
  by (auto simp add: measurable_def B Ω)
qed

```

```

lemma measurable_measure_of:
  assumes  $B: N \subseteq \text{Pow } \Omega$ 
  and  $f: f \in \text{space } M \rightarrow \Omega$ 
  and  $ba: \bigwedge y. y \in N \implies (f -' y) \cap \text{space } M \in \text{sets } M$ 
  shows  $f \in \text{measurable } M (\text{measure\_of } \Omega N \mu)$ 
proof -
  have  $\text{sets } (\text{measure\_of } \Omega N \mu) = \text{sigma\_sets } \Omega N$ 
  using  $B$  by (rule sets_measure_of)
  from this assms show ?thesis by (rule measurable_sigma_sets)
qed

```

```

lemma measurable_iff_measure_of:
  assumes  $N \subseteq \text{Pow } \Omega$   $f \in \text{space } M \rightarrow \Omega$ 
  shows  $f \in \text{measurable } M (\text{measure\_of } \Omega N \mu) \longleftrightarrow (\forall A \in N. f -' A \cap \text{space } M \in \text{sets } M)$ 
  by (metis assms in_measure_of measurable_measure_of assms measurable_sets)

```

```

lemma measurable_cong_sets:
  assumes  $\text{sets: sets } M = \text{sets } M' \text{ sets } N = \text{sets } N'$ 
  shows  $\text{measurable } M N = \text{measurable } M' N'$ 
  using  $\text{sets}[THEN \text{sets\_eq\_imp\_space\_eq}] \text{sets}$  by (simp add: measurable_def)

```

```

lemma measurable_cong:
  assumes  $\bigwedge w. w \in \text{space } M \implies f w = g w$ 
  shows  $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } M M'$ 
  unfolding measurable_def using assms
  by (simp cong: vimage_inter_cong Pi_cong)

```

```

lemma measurable_cong':
  assumes  $\bigwedge w. w \in \text{space } M = \text{simp} \implies f w = g w$ 
  shows  $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } M M'$ 
  unfolding measurable_def using assms
  by (simp cong: vimage_inter_cong Pi_cong add: simp_implies_def)

```

```

lemma measurable_cong_simp:
   $M = N \implies M' = N' \implies (\bigwedge w. w \in \text{space } M \implies f w = g w) \implies$ 
   $f \in \text{measurable } M M' \longleftrightarrow g \in \text{measurable } N N'$ 
  by (metis measurable_cong)

```

```

lemma measurable_compose:

```

assumes $f: f \in \text{measurable } M N$ **and** $g: g \in \text{measurable } N L$
shows $(\lambda x. g (f x)) \in \text{measurable } M L$
proof –
have $\bigwedge A. (\lambda x. g (f x)) -' A \cap \text{space } M = f -' (g -' A \cap \text{space } N) \cap \text{space } M$
using `measurable_space[OF f]` **by** `auto`
with `measurable_space[OF f]` `measurable_space[OF g]` **show** `?thesis`
by (`auto intro: measurable_sets[OF f] measurable_sets[OF g]`
`simp del: vimage_Int simp add: measurable_def`)
qed

lemma `measurable_comp`:
 $f \in \text{measurable } M N \implies g \in \text{measurable } N L \implies g \circ f \in \text{measurable } M L$
using `measurable_compose[of f M N g L]` **by** (`simp add: comp_def`)

lemma `measurable_const`:
 $c \in \text{space } M' \implies (\lambda x. c) \in \text{measurable } M M'$
by (`auto simp add: measurable_def`)

lemma `measurable_ident`: $\text{id} \in \text{measurable } M M$
by (`auto simp add: measurable_def`)

lemma `measurable_id`: $(\lambda x. x) \in \text{measurable } M M$
by (`simp add: measurable_def`)

lemma `measurable_ident_sets`:
assumes `eq: sets M = sets M'` **shows** $(\lambda x. x) \in \text{measurable } M M'$
using `measurable_ident[of M]`
unfolding `id_def measurable_def eq_sets_eq_imp_space_eq[OF eq]` .

lemma `sets_Least`:
assumes `meas: $\bigwedge i::\text{nat}. \{x \in \text{space } M. P i x\} \in M$`
shows $(\lambda x. \text{LEAST } j. P j x) -' A \cap \text{space } M \in \text{sets } M$
proof –
{ fix i **have** $(\lambda x. \text{LEAST } j. P j x) -' \{i\} \cap \text{space } M \in \text{sets } M$
proof `cases`
assume $i: (\text{LEAST } j. \text{False}) = i$
have $(\lambda x. \text{LEAST } j. P j x) -' \{i\} \cap \text{space } M =$
 $\{x \in \text{space } M. P i x\} \cap (\text{space } M - (\bigcup j < i. \{x \in \text{space } M. P j x\})) \cup (\text{space } M - (\bigcup i. \{x \in \text{space } M. P i x\}))$
by (`simp add: set_eq_iff, safe`)
 $(\text{insert } i, \text{auto dest: Least_le intro: LeastI intro!: Least_equality})$
with `meas` **show** `?thesis`
by (`auto intro!: sets.Int`)
next
assume $i: (\text{LEAST } j. \text{False}) \neq i$
then **have** $(\lambda x. \text{LEAST } j. P j x) -' \{i\} \cap \text{space } M =$
 $\{x \in \text{space } M. P i x\} \cap (\text{space } M - (\bigcup j < i. \{x \in \text{space } M. P j x\}))$
proof (`simp add: set_eq_iff, safe`)
fix x **assume** `neq: $(\text{LEAST } j. \text{False}) \neq (\text{LEAST } j. P j x)$`


```

    have  $\exists j. P j x$ 
      by (rule ccontr) (insert neg, auto)
    then show  $P (LEAST j. P j x) x$  by (rule LeastI_ex)
  qed (auto dest: Least_le intro!: Least_equality)
  with meas show ?thesis
    by auto
  qed }
  then have  $(\bigcup_{i \in A}. (\lambda x. LEAST j. P j x) - \{i\} \cap \text{space } M) \in \text{sets } M$ 
    by (intro sets.countable_UN) auto
  moreover have  $(\bigcup_{i \in A}. (\lambda x. LEAST j. P j x) - \{i\} \cap \text{space } M) =$ 
     $(\lambda x. LEAST j. P j x) - A \cap \text{space } M$  by auto
  ultimately show ?thesis by auto
qed

```

lemma *measurable_mono1*:

```

 $M' \subseteq \text{Pow } \Omega \implies M \subseteq M' \implies$ 
  measurable (measure_of  $\Omega$   $M$   $\mu$ )  $N \subseteq$  measurable (measure_of  $\Omega$   $M'$   $\mu'$ )  $N$ 
  using measure_of_subset[of  $M'$   $\Omega$   $M$ ] by (auto simp add: measurable_def)

```

Counting space

definition *count_space* :: 'a set \Rightarrow 'a measure **where**

```

count_space  $\Omega = \text{measure\_of } \Omega (\text{Pow } \Omega) (\lambda A. \text{if finite } A \text{ then of\_nat (card } A) \text{ else } \infty)$ 

```

lemma

```

shows space_count_space[simp]:  $\text{space (count\_space } \Omega) = \Omega$ 
  and sets_count_space[simp]:  $\text{sets (count\_space } \Omega) = \text{Pow } \Omega$ 
  using sigma_sets_into_sp[of  $\text{Pow } \Omega$   $\Omega$ ]
  by (auto simp: count_space_def)

```

lemma *measurable_count_space_eq1*[simp]:

```

 $f \in \text{measurable (count\_space } A) M \longleftrightarrow f \in A \rightarrow \text{space } M$ 
  unfolding measurable_def by simp

```

lemma *finite_count_space*: $\text{finite } \Omega \implies \text{count_space } \Omega = \text{measure_of } \Omega (\text{Pow } \Omega) \text{ card}$

unfolding *count_space_def*

```

  by (smt (verit, best) PowD Pow_top count_space_def finite_subset measure_of_eq
    sets_count_space sets_measure_of)

```

lemma *measurable_compose_countable'*:

```

assumes  $f: \bigwedge i. i \in I \implies (\lambda x. f i x) \in \text{measurable } M N$ 
  and  $g: g \in \text{measurable } M (\text{count\_space } I)$  and  $I: \text{countable } I$ 
shows  $(\lambda x. f (g x) x) \in \text{measurable } M N$ 
  unfolding measurable_def

```

proof *safe*

```

  fix  $x$  assume  $x \in \text{space } M$  then show  $f (g x) x \in \text{space } N$ 
    using measurable_space[OF  $f$ ]  $g$ [THEN measurable_space] by auto

```

2070

next

fix A **assume** $A: A \in \text{sets } N$
have $(\lambda x. f (g x) x) -' A \cap \text{space } M = (\bigcup_{i \in I. (g -' \{i\} \cap \text{space } M) \cap (f i -' A \cap \text{space } M))$
using $\text{measurable_space}[OF g]$ **by** auto
also have $\dots \in \text{sets } M$
using $f[THEN \text{measurable_sets}, OF _ A] g[THEN \text{measurable_sets}]$
by $(\text{auto intro!}: \text{sets.countable_UN}' I \text{ intro}: \text{sets.Int}[OF \text{measurable_sets measurable_sets}])$
finally show $(\lambda x. f (g x) x) -' A \cap \text{space } M \in \text{sets } M .$
qed

lemma $\text{measurable_count_space_eq_countable}$:

assumes $\text{countable } A$
shows $f \in \text{measurable } M (\text{count_space } A) \longleftrightarrow (f \in \text{space } M \rightarrow A \wedge (\forall a \in A. f -' \{a\} \cap \text{space } M \in \text{sets } M))$
proof –
{ **fix** X **assume** $X \subseteq A$ $f \in \text{space } M \rightarrow A$
with $\langle \text{countable } A \rangle$ **have** $f -' X \cap \text{space } M = (\bigcup_{a \in X. f -' \{a\} \cap \text{space } M)$
 $\text{countable } X$
by $(\text{auto dest}: \text{countable_subset})$
moreover assume $\forall a \in A. f -' \{a\} \cap \text{space } M \in \text{sets } M$
ultimately have $f -' X \cap \text{space } M \in \text{sets } M$
using $\langle X \subseteq A \rangle$ **by** $(\text{auto intro!}: \text{sets.countable_UN}' \text{ simp del}: \text{UN_simps})$ }
then show $?thesis$
unfolding measurable_def **by** auto
qed

lemma $\text{measurable_count_space_eq2}$:

$\text{finite } A \implies f \in \text{measurable } M (\text{count_space } A) \longleftrightarrow (f \in \text{space } M \rightarrow A \wedge (\forall a \in A. f -' \{a\} \cap \text{space } M \in \text{sets } M))$
by $(\text{intro measurable_count_space_eq_countable countable_finite})$

lemma $\text{measurable_count_space_eq2_countable}$:

fixes $f :: 'a \Rightarrow 'c::\text{countable}$
shows $f \in \text{measurable } M (\text{count_space } A) \longleftrightarrow (f \in \text{space } M \rightarrow A \wedge (\forall a \in A. f -' \{a\} \cap \text{space } M \in \text{sets } M))$
by $(\text{intro measurable_count_space_eq_countable countableI_type})$

lemma $\text{measurable_compose_countable}$:

assumes $f: \bigwedge i::'i::\text{countable}. (\lambda x. f i x) \in \text{measurable } M N$ **and** $g: g \in \text{measurable } M (\text{count_space } UNIV)$
shows $(\lambda x. f (g x) x) \in \text{measurable } M N$
by $(\text{rule measurable_compose_countable}'[OF \text{assms}]) \text{ auto}$

lemma $\text{measurable_count_space_const}$:

$(\lambda x. c) \in \text{measurable } M (\text{count_space } UNIV)$
by $(\text{simp add}: \text{measurable_const})$

lemma *measurable_count_space*:
 $f \in \text{measurable } (\text{count_space } A) (\text{count_space } \text{UNIV})$
by *simp*

lemma *measurable_compose_rev*:
assumes $f: f \in \text{measurable } L \ N$ **and** $g: g \in \text{measurable } M \ L$
shows $(\lambda x. f (g x)) \in \text{measurable } M \ N$
using *measurable_compose[OF g f]* .

lemma *measurable_empty_iff*:
 $\text{space } N = \{\} \implies f \in \text{measurable } M \ N \longleftrightarrow \text{space } M = \{\}$
by (*auto simp add: measurable_def Pi_iff*)

Extend measure

definition *extend_measure* :: $'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow ('b \Rightarrow 'a \text{ set}) \Rightarrow ('b \Rightarrow \text{ennreal}) \Rightarrow 'a \text{ measure}$

where
 $\text{extend_measure } \Omega \ I \ G \ \mu =$
 (if $(\exists \mu'. (\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure_space } \Omega (\text{sigma_sets } \Omega (G'I)) \ \mu')$
 $\wedge \neg (\forall i \in I. \mu i = 0)$
 then $\text{measure_of } \Omega (G'I) (\text{SOME } \mu'. (\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure_space } \Omega (\text{sigma_sets } \Omega (G'I)) \ \mu')$
 else $\text{measure_of } \Omega (G'I) (\lambda_. 0)$)

lemma *space_extend_measure*: $G \text{ ' } I \subseteq \text{Pow } \Omega \implies \text{space } (\text{extend_measure } \Omega \ I \ G \ \mu) = \Omega$
unfolding *extend_measure_def* **by** *simp*

lemma *sets_extend_measure*: $G \text{ ' } I \subseteq \text{Pow } \Omega \implies \text{sets } (\text{extend_measure } \Omega \ I \ G \ \mu) = \text{sigma_sets } \Omega (G'I)$
unfolding *extend_measure_def* **by** *simp*

lemma *emeasure_extend_measure*:
assumes $M: M = \text{extend_measure } \Omega \ I \ G \ \mu$
and $eq: \bigwedge i. i \in I \implies \mu' (G i) = \mu i$
and $ms: G \text{ ' } I \subseteq \text{Pow } \Omega \text{ positive } (\text{sets } M) \ \mu' \text{ countably_additive } (\text{sets } M) \ \mu'$
and $i \in I$
shows $\text{emeasure } M (G i) = \mu i$

proof *cases*

assume $*$: $(\forall i \in I. \mu i = 0)$
with M **have** $M_eq: M = \text{measure_of } \Omega (G'I) (\lambda_. 0)$
by (*simp add: extend_measure_def*)
from *measure_space_0[OF ms(1)] ms <i∈I>*
have $\text{emeasure } M (G i) = 0$
by (*intro emeasure_measure_of[OF M_eq]*) (*auto simp add: M measure_space_def sets_extend_measure*)
with $\langle i \in I \rangle *$ **show** *?thesis*
by *simp*

```

next
define P where P  $\mu'$   $\longleftrightarrow$   $(\forall i \in I. \mu' (G i) = \mu i) \wedge \text{measure\_space } \Omega (\text{sigma\_sets } \Omega (G'I)) \mu'$  for  $\mu'$ 
assume  $\neg (\forall i \in I. \mu i = 0)$ 
moreover
have measure_space (space M) (sets M)  $\mu'$ 
using ms unfolding measure_space_def by auto standard
with ms eq have  $\exists \mu'. P \mu'$ 
unfolding P_def
by (intro exI[of _  $\mu'$ ]) (auto simp add: M space_extend_measure sets_extend_measure)
ultimately have M_eq:  $M = \text{measure\_of } \Omega (G'I) (Eps P)$ 
by (simp add: M extend_measure_def P_def[symmetric])

from  $\langle \exists \mu'. P \mu' \rangle$  have P:  $P (Eps P)$  by (rule someI_ex)
show emeasure M  $(G i) = \mu i$ 
proof (subst emeasure_measure_of[OF M_eq])
have sets_M: sets M = sigma_sets  $\Omega (G'I)$ 
using M_eq ms by (auto simp: sets_extend_measure)
then show  $G i \in \text{sets } M$  using  $\langle i \in I \rangle$  by auto
show positive (sets M) (Eps P) countably_additive (sets M) (Eps P) Eps P
 $(G i) = \mu i$ 
using P  $\langle i \in I \rangle$  by (auto simp add: sets_M measure_space_def P_def)
qed fact
qed

```

```

lemma emeasure_extend_measure_Pair:
assumes M:  $M = \text{extend\_measure } \Omega \{(i, j). I i j\} (\lambda(i, j). G i j) (\lambda(i, j). \mu i j)$ 
and eq:  $\bigwedge i j. I i j \implies \mu' (G i j) = \mu i j$ 
and ms:  $\bigwedge i j. I i j \implies G i j \in \text{Pow } \Omega \text{ positive (sets M) } \mu' \text{ countably\_additive (sets M) } \mu'$ 
and I i j
shows emeasure M  $(G i j) = \mu i j$ 
using emeasure_extend_measure[OF M __ ms(2,3), of (i,j)] eq ms(1)  $\langle I i j \rangle$ 
by (auto simp: subset_eq)

```

7.1.3 The smallest σ -algebra regarding a function

definition *vimage_algebra* :: $'a \text{ set} \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \text{ measure} \Rightarrow 'a \text{ measure}$
where

vimage_algebra $X f M = \text{sigma } X \{f -' A \cap X \mid A. A \in \text{sets } M\}$

lemma *space_vimage_algebra*[simp]: $\text{space (vimage_algebra } X f M) = X$
unfolding *vimage_algebra_def* by (rule space_measure_of) auto

lemma *sets_vimage_algebra*: $\text{sets (vimage_algebra } X f M) = \text{sigma_sets } X \{f -' A \cap X \mid A. A \in \text{sets } M\}$
unfolding *vimage_algebra_def* by (rule sets_measure_of) auto

lemma *sets_vimage_algebra2*:

$f \in X \rightarrow \text{space } M \implies \text{sets } (\text{vimage_algebra } X f M) = \{f^{-1} A \cap X \mid A. A \in \text{sets } M\}$

using *sigma_sets_vimage_commute*[of $f X \text{space } M \text{sets } M$]
unfolding *sets_vimage_algebra* *sets.sigma_sets_eq* **by** *simp*

lemma *sets_vimage_algebra_cong*: $\text{sets } M = \text{sets } N \implies \text{sets } (\text{vimage_algebra } X f M) = \text{sets } (\text{vimage_algebra } X f N)$

by (*simp add: sets_vimage_algebra*)

lemma *vimage_algebra_cong*:

assumes $X = Y$

assumes $\bigwedge x. x \in Y \implies f x = g x$

assumes $\text{sets } M = \text{sets } N$

shows $\text{vimage_algebra } X f M = \text{vimage_algebra } Y g N$

by (*auto simp: vimage_algebra_def assms intro!: arg_cong2*[**where** $f = \text{sigma}$])

lemma *in_vimage_algebra*: $A \in \text{sets } M \implies f^{-1} A \cap X \in \text{sets } (\text{vimage_algebra } X f M)$

by (*auto simp: vimage_algebra_def*)

lemma *sets_image_in_sets*:

assumes $N: \text{space } N = X$

assumes $f: f \in \text{measurable } N M$

shows $\text{sets } (\text{vimage_algebra } X f M) \subseteq \text{sets } N$

unfolding *sets_vimage_algebra* N [*symmetric*]

by (*rule sets.sigma_sets_subset*) (*auto intro!: measurable_sets f*)

lemma *measurable_vimage_algebra1*: $f \in X \rightarrow \text{space } M \implies f \in \text{measurable } (\text{vimage_algebra } X f M) M$

unfolding *measurable_def* **by** (*auto intro: in_vimage_algebra*)

lemma *measurable_vimage_algebra2*:

assumes $g: g \in \text{space } N \rightarrow X$ **and** $f: (\lambda x. f (g x)) \in \text{measurable } N M$

shows $g \in \text{measurable } N (\text{vimage_algebra } X f M)$

unfolding *vimage_algebra_def*

proof (*rule measurable_measure_of*)

fix A **assume** $A \in \{f^{-1} A \cap X \mid A. A \in \text{sets } M\}$

then obtain Y **where** $Y: Y \in \text{sets } M$ **and** $A = f^{-1} Y \cap X$

by *auto*

then have $g^{-1} A \cap \text{space } N = (\lambda x. f (g x))^{-1} Y \cap \text{space } N$

using g **by** *auto*

also have $\dots \in \text{sets } N$

using $f Y$ **by** (*rule measurable_sets*)

finally show $g^{-1} A \cap \text{space } N \in \text{sets } N$.

qed (*insert g, auto*)

lemma *vimage_algebra_sigma*:

assumes $X: X \subseteq \text{Pow } \Omega'$ **and** $f: f \in \Omega \rightarrow \Omega'$

shows $\text{vimage_algebra } \Omega f (\text{sigma } \Omega' X) = \text{sigma } \Omega \{f^{-1} A \cap \Omega \mid A. A \in X\}$

(is ?V = ?S)
proof (rule measure_eqI)
 have $\Omega: \{f -' A \cap \Omega \mid A. A \in X\} \subseteq Pow \Omega$ **by** auto
 show sets ?V = sets ?S
 using sigma_sets_vimage_commute[OF f, of X]
 by (simp add: space_measure_of_conv f sets_vimage_algebra2 Ω X)
qed (simp add: vimage_algebra_def emeasure_sigma)

lemma vimage_algebra_vimage_algebra_eq:
 assumes *: $f \in X \rightarrow Y$ $g \in Y \rightarrow space M$
 shows $vimage_algebra X f (vimage_algebra Y g M) = vimage_algebra X (\lambda x. g (f x)) M$
 (is ?VV = ?V)
proof (rule measure_eqI)
 have $(\lambda x. g (f x)) \in X \rightarrow space M \wedge A. A \cap f -' Y \cap X = A \cap X$
 using * **by** auto
 with * **show** sets ?VV = sets ?V
 by (simp add: sets_vimage_algebra2 vimage_comp comp_def flip: ex_simps)
qed (simp add: vimage_algebra_def emeasure_sigma)

Restricted Space Sigma Algebra

definition restrict_space :: 'a measure \Rightarrow 'a set \Rightarrow 'a measure **where**
 restrict_space M $\Omega = measure_of (\Omega \cap space M) ((\cap) \Omega) 'sets M$ (emeasure M)

lemma space_restrict_space: $space (restrict_space M \Omega) = \Omega \cap space M$
 using sets.sets_into_space **unfolding** restrict_space_def **by** (subst space_measure_of) auto

lemma space_restrict_space2 [simp]: $\Omega \in sets M \Longrightarrow space (restrict_space M \Omega) = \Omega$
 by (simp add: space_restrict_space sets.sets_into_space)

lemma sets_restrict_space: $sets (restrict_space M \Omega) = ((\cap) \Omega) 'sets M$
unfolding restrict_space_def

proof (subst sets_measure_of)
 show $(\cap) \Omega 'sets M \subseteq Pow (\Omega \cap space M)$
 by (auto dest: sets.sets_into_space)
 have $sigma_sets (\Omega \cap space M) \{((\lambda x. x) -' X) \cap (\Omega \cap space M) \mid X. X \in sets M\} =$
 $(\lambda X. X \cap (\Omega \cap space M)) 'sets M$
 by (subst sigma_sets_vimage_commute[symmetric, **where** $\Omega' = space M$])
 (auto simp add: sets.sigma_sets_eq)
moreover have $\{((\lambda x. x) -' X) \cap (\Omega \cap space M) \mid X. X \in sets M\} = (\lambda X. X \cap (\Omega \cap space M)) 'sets M$
 by auto
moreover have $(\lambda X. X \cap (\Omega \cap space M)) 'sets M = ((\cap) \Omega) 'sets M$
 by (intro image_cong) (auto dest: sets.sets_into_space)

ultimately show $\text{sigma_sets } (\Omega \cap \text{space } M) ((\cap) \Omega \text{ ' sets } M) = (\cap) \Omega \text{ ' sets } M$
by simp
qed

lemma *restrict_space_sets_cong*:

$A = B \implies \text{sets } M = \text{sets } N \implies \text{sets } (\text{restrict_space } M A) = \text{sets } (\text{restrict_space } N B)$
by (*auto simp: sets_restrict_space*)

lemma *sets_restrict_space_count_space* :

$\text{sets } (\text{restrict_space } (\text{count_space } A) B) = \text{sets } (\text{count_space } (A \cap B))$
by(*auto simp add: sets_restrict_space*)

lemma *sets_restrict_UNIV[simp]*: $\text{sets } (\text{restrict_space } M \text{ UNIV}) = \text{sets } M$

by (*auto simp add: sets_restrict_space*)

lemma *sets_restrict_restrict_space*:

$\text{sets } (\text{restrict_space } (\text{restrict_space } M A) B) = \text{sets } (\text{restrict_space } M (A \cap B))$
unfolding *sets_restrict_space image_comp* **by** (*intro image_cong auto*)

lemma *sets_restrict_space_iff*:

$\Omega \cap \text{space } M \in \text{sets } M \implies A \in \text{sets } (\text{restrict_space } M \Omega) \iff (A \subseteq \Omega \wedge A \in \text{sets } M)$

proof (*subst sets_restrict_space, safe*)

fix A **assume** $\Omega \cap \text{space } M \in \text{sets } M$ **and** $A: A \in \text{sets } M$

then have $(\Omega \cap \text{space } M) \cap A \in \text{sets } M$

by rule

also have $(\Omega \cap \text{space } M) \cap A = \Omega \cap A$

using *sets.sets_into_space[OF A]* **by auto**

finally show $\Omega \cap A \in \text{sets } M$

by auto

qed *auto*

lemma *sets_restrict_space_cong*: $\text{sets } M = \text{sets } N \implies \text{sets } (\text{restrict_space } M \Omega) = \text{sets } (\text{restrict_space } N \Omega)$

by (*simp add: sets_restrict_space*)

lemma *restrict_space_eq_vimage_algebra*:

$\Omega \subseteq \text{space } M \implies \text{sets } (\text{restrict_space } M \Omega) = \text{sets } (\text{vimage_algebra } \Omega (\lambda x. x) M)$

unfolding *restrict_space_def*

apply (*subst sets_measure_of*)

apply (*auto simp add: image_subset_iff dest: sets.sets_into_space*) []

apply (*auto simp add: sets_vimage_algebra intro!: arg_cong2[where f=sigma_sets]*)
done

lemma *sets_Collect_restrict_space_iff*:

assumes $S \in \text{sets } M$

shows $\{x \in \text{space } (\text{restrict_space } M S). P x\} \in \text{sets } (\text{restrict_space } M S) \iff$

$\{x \in \text{space } M. x \in S \wedge P x\} \in \text{sets } M$
proof –
have $\{x \in S. P x\} = \{x \in \text{space } M. x \in S \wedge P x\}$
using *sets.sets_into_space*[*OF assms*] **by** *auto*
then show *?thesis*
by (*subst sets_restrict_space_iff*) (*auto simp add: space_restrict_space assms*)
qed

lemma *measurable_restrict_space1*:
assumes $f: f \in \text{measurable } M N$
shows $f \in \text{measurable } (\text{restrict_space } M \Omega) N$
unfolding *measurable_def*
proof (*intro CollectI conjI ballI*)
show $sp: f \in \text{space } (\text{restrict_space } M \Omega) \rightarrow \text{space } N$
using *measurable_space*[*OF f*] **by** (*auto simp: space_restrict_space*)

fix A **assume** $A \in \text{sets } N$
have $f - ' A \cap \text{space } (\text{restrict_space } M \Omega) = (f - ' A \cap \text{space } M) \cap (\Omega \cap \text{space } M)$
by (*auto simp: space_restrict_space*)
also have $\dots \in \text{sets } (\text{restrict_space } M \Omega)$
unfolding *sets_restrict_space*
using *measurable_sets*[*OF f* $\langle A \in \text{sets } N \rangle$] **by** *blast*
finally show $f - ' A \cap \text{space } (\text{restrict_space } M \Omega) \in \text{sets } (\text{restrict_space } M \Omega)$
qed

lemma *measurable_restrict_space2_iff*:
 $f \in \text{measurable } M (\text{restrict_space } N \Omega) \iff (f \in \text{measurable } M N \wedge f \in \text{space } M \rightarrow \Omega)$
proof –
have $\bigwedge A. f \in \text{space } M \rightarrow \Omega \implies f - ' \Omega \cap f - ' A \cap \text{space } M = f - ' A \cap \text{space } M$
by *auto*
then show *?thesis*
by (*auto simp: measurable_def space_restrict_space Pi_Int[symmetric] sets_restrict_space*)
qed

lemma *measurable_restrict_space2*:
 $f \in \text{space } M \rightarrow \Omega \implies f \in \text{measurable } M N \implies f \in \text{measurable } M (\text{restrict_space } N \Omega)$
by (*simp add: measurable_restrict_space2_iff*)

lemma *measurable_piecewise_restrict*:
assumes $I: \text{countable } C$
and $X: \bigwedge \Omega. \Omega \in C \implies \Omega \cap \text{space } M \in \text{sets } M \text{ space } M \subseteq \bigcup C$
and $f: \bigwedge \Omega. \Omega \in C \implies f \in \text{measurable } (\text{restrict_space } M \Omega) N$
shows $f \in \text{measurable } M N$
proof (*rule measurableI*)


```

fix  $x$  assume  $x \in \text{space } M$ 
with  $X$  obtain  $\Omega$  where  $\Omega \in C$   $x \in \Omega$   $x \in \text{space } M$  by auto
then show  $f x \in \text{space } N$ 
  by (auto simp: space_restrict_space intro: f measurable_space)
next
fix  $A$  assume  $A: A \in \text{sets } N$ 
have  $f -' A \cap \text{space } M = (\bigcup \Omega \in C. (f -' A \cap (\Omega \cap \text{space } M)))$ 
  using  $X$  by (auto simp: subset_eq)
also have  $\dots \in \text{sets } M$ 
  using measurable_sets[OF f A] X I
  by (intro sets.countable_UN' (auto simp: sets_restrict_space_iff space_restrict_space))
finally show  $f -' A \cap \text{space } M \in \text{sets } M$  .
qed

```

```

lemma measurable_piecewise_restrict_iff:
  countable C  $\implies (\bigwedge \Omega. \Omega \in C \implies \Omega \cap \text{space } M \in \text{sets } M) \implies \text{space } M \subseteq (\bigcup C)$ 
 $\implies$ 
   $f \in \text{measurable } M N \iff (\forall \Omega \in C. f \in \text{measurable } (\text{restrict\_space } M \Omega) N)$ 
  by (auto intro: measurable_piecewise_restrict measurable_restrict_space1)

```

```

lemma measurable_If_restrict_space_iff:
   $\{x \in \text{space } M. P x\} \in \text{sets } M \implies$ 
   $(\lambda x. \text{if } P x \text{ then } f x \text{ else } g x) \in \text{measurable } M N \iff$ 
   $(f \in \text{measurable } (\text{restrict\_space } M \{x. P x\}) N \wedge g \in \text{measurable } (\text{restrict\_space } M \{x. \neg P x\}) N)$ 
  by (subst measurable_piecewise_restrict_iff[where C={{x. P x}, {x. ¬ P x}}])
  (auto simp: Int_def sets.sets_Collect_neg space_restrict_space conj_commute[of _ x \in space M for x])
  (cong: measurable_cong')

```

```

lemma measurable_If:
   $f \in \text{measurable } M M' \implies g \in \text{measurable } M M' \implies \{x \in \text{space } M. P x\} \in \text{sets } M \implies$ 
   $(\lambda x. \text{if } P x \text{ then } f x \text{ else } g x) \in \text{measurable } M M'$ 
  unfolding measurable_If_restrict_space_iff by (auto intro: measurable_restrict_space1)

```

```

lemma measurable_If_set:
  assumes measure: f \in measurable M M' g \in measurable M M'
  assumes P: A \cap space M \in sets M
  shows  $(\lambda x. \text{if } x \in A \text{ then } f x \text{ else } g x) \in \text{measurable } M M'$ 
proof (rule measurable_If[OF measure])
  have  $\{x \in \text{space } M. x \in A\} = A \cap \text{space } M$  by auto
  thus  $\{x \in \text{space } M. x \in A\} \in \text{sets } M$  using  $\langle A \cap \text{space } M \in \text{sets } M \rangle$  by auto
qed

```

```

lemma measurable_restrict_space_iff:
   $\Omega \cap \text{space } M \in \text{sets } M \implies c \in \text{space } N \implies$ 
   $f \in \text{measurable } (\text{restrict\_space } M \Omega) N \iff (\lambda x. \text{if } x \in \Omega \text{ then } f x \text{ else } c) \in \text{measurable } M N$ 

```

by (subst measurable>If_restrict_space_iff)
 (simp_all add: Int_def conj_commute measurable_const)

lemma restrict_space_singleton: $\{x\} \in \text{sets } M \implies \text{sets } (\text{restrict_space } M \{x\})$
 $= \text{sets } (\text{count_space } \{x\})$
 using sets_restrict_space_iff[of $\{x\}$ M]
 by (auto simp add: sets_restrict_space_iff dest!: subset_singletonD)

lemma measurable_restrict_countable:
 assumes $X[\text{intro}]$: countable X
 assumes sets[simp]: $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$
 assumes space[simp]: $\bigwedge x. x \in X \implies f x \in \text{space } N$
 assumes f : $f \in \text{measurable } (\text{restrict_space } M (- X)) N$
 shows $f \in \text{measurable } M N$
 using f sets.countable[OF sets X]
 by (intro measurable_piecewise_restrict[where $M=M$ and $C=\{- X\} \cup ((\lambda x. \{x\}) ' X)$])
 (auto simp: Diff_Int_distrib2 Compl_eq_Diff_UNIV Int_insert_left sets.Diff
 restrict_space_singleton
 simp del: sets_count_space cong: measurable_cong_sets)

lemma measurable_discrete_difference:
 assumes f : $f \in \text{measurable } M N$
 assumes X : countable X $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ $\bigwedge x. x \in X \implies g x \in \text{space } N$
 assumes eq: $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$
 shows $g \in \text{measurable } M N$
 by (rule measurable_restrict_countable[OF X])
 (auto simp: eq[symmetric] space_restrict_space cong: measurable_cong' intro:
 f measurable_restrict_space1)

lemma measurable_count_space_extend: $A \subseteq B \implies f \in \text{space } M \rightarrow A \implies f \in M \rightarrow_M \text{count_space } B \implies f \in M \rightarrow_M \text{count_space } A$
 by (auto simp: measurable_def)

end

7.2 Measurability Prover

theory Measurable
 imports
 Sigma_Algebra
 HOL-Library.Order_Continuity
begin

lemma (in algebra) sets_Collect_finite_All:
 assumes $\bigwedge i. i \in S \implies \{x \in \Omega. P i x\} \in M$ finite S
 shows $\{x \in \Omega. \forall i \in S. P i x\} \in M$

proof –

have $\{x \in \Omega. \forall i \in S. P\ i\ x\} = (\text{if } S = \{\} \text{ then } \Omega \text{ else } \bigcap i \in S. \{x \in \Omega. P\ i\ x\})$

by *auto*

with *assms* **show** *?thesis* **by** (*auto intro!*: *sets_Collect_finite_All'*)

qed

abbreviation $\text{pred } M\ P \equiv P \in \text{measurable } M (\text{count_space } (UNIV::\text{bool set}))$

lemma *pred_def*: $\text{pred } M\ P \longleftrightarrow \{x \in \text{space } M. P\ x\} \in \text{sets } M$

proof

assume *pred M P*

then have $P - \{True\} \cap \text{space } M \in \text{sets } M$

by (*auto simp: measurable_count_space_eq2*)

also have $P - \{True\} \cap \text{space } M = \{x \in \text{space } M. P\ x\}$ **by** *auto*

finally show $\{x \in \text{space } M. P\ x\} \in \text{sets } M$.

next

assume $P: \{x \in \text{space } M. P\ x\} \in \text{sets } M$

moreover

{ fix *X*

have $X \in \text{Pow } (UNIV :: \text{bool set})$ **by** *simp*

then have $P - X \cap \text{space } M = \{x \in \text{space } M. ((X = \{True\} \longrightarrow P\ x) \wedge (X = \{False\} \longrightarrow \neg P\ x) \wedge X \neq \{\})\}$

unfolding *UNIV_bool Pow_insert Pow_empty* **by** *auto*

then have $P - X \cap \text{space } M \in \text{sets } M$

by (*auto intro!*: *sets.sets_Collect_neg sets.sets_Collect_imp sets.sets_Collect_conj sets.sets_Collect_const P*) }

then show *pred M P*

by (*auto simp: measurable_def*)

qed

lemma *pred_sets1*: $\{x \in \text{space } M. P\ x\} \in \text{sets } M \implies f \in \text{measurable } N\ M \implies \text{pred } N (\lambda x. P (f\ x))$

by (*rule measurable_compose*[**where** $f=f$ **and** $N=M$]) (*auto simp: pred_def*)

lemma *pred_sets2*: $A \in \text{sets } N \implies f \in \text{measurable } M\ N \implies \text{pred } M (\lambda x. f\ x \in A)$

by (*rule measurable_compose*[**where** $f=f$ **and** $N=N$]) (*auto simp: pred_def Int_def[symmetric]*)

ML_file $\langle \text{measurable.ML} \rangle$

attribute_setup *measurable* = \langle

Scan.lift (\langle

(Args.add >> K true || Args.del >> K false || Scan.succeed true) --

Scan.optional (Args.parens (\langle

Scan.optional (Args.\$\$\$ raw >> K true) false --

Scan.optional (Args.\$\$\$ generic >> K Measurable.Generic) Measurable.Concrete))

(false, Measurable.Concrete) >>

Measurable.measurable_thm_attr)

\rangle)

› *declaration of measurability theorems*

attribute_setup *measurable_dest* = *Measurable.dest_thm_attr*
add dest rule to measurability prover

attribute_setup *measurable_cong* = *Measurable.cong_thm_attr*
add congruence rules to measurability prover

method_setup *measurable* = ‹ *Scan.lift (Scan.succeed (METHOD o Measurable.measurable_tac))* ›
measurability prover

simproc_setup *measurable* (*A* ∈ *sets M* | *f* ∈ *measurable M N*) =
 ‹*K Measurable.proc*›

setup ‹
Global_Theory.add_thms_dynamic (binding ‹measurable›, Measurable.get_all)
 ›

declare
pred_sets1[*measurable_dest*]
pred_sets2[*measurable_dest*]
sets.sets_into_space[*measurable_dest*]

declare
sets.top[*measurable*]
sets.empty_sets[*measurable (raw)*]
sets.Un[*measurable (raw)*]
sets.Diff[*measurable (raw)*]

declare
measurable_count_space[*measurable (raw)*]
measurable_ident[*measurable (raw)*]
measurable_id[*measurable (raw)*]
measurable_const[*measurable (raw)*]
measurable_If[*measurable (raw)*]
measurable_comp[*measurable (raw)*]
measurable_sets[*measurable (raw)*]

declare *measurable_cong_sets*[*measurable_cong*]
declare *sets_restrict_space_cong*[*measurable_cong*]
declare *sets_restrict_UNIV*[*measurable_cong*]

lemma *predE*[*measurable (raw)*]:
pred M P ⇒ {*x* ∈ *space M*. *P x*} ∈ *sets M*
unfolding *pred_def* .

lemma *pred_intros_imp'*[*measurable (raw)*]:
 (*K* ⇒ *pred M (λx. P x)*) ⇒ *pred M (λx. K → P x)*

by (cases K) auto

lemma *pred_intros_conj1* [measurable (raw)]:

$(K \implies \text{pred } M (\lambda x. P x)) \implies \text{pred } M (\lambda x. K \wedge P x)$

by (cases K) auto

lemma *pred_intros_conj2* [measurable (raw)]:

$(K \implies \text{pred } M (\lambda x. P x)) \implies \text{pred } M (\lambda x. P x \wedge K)$

by (cases K) auto

lemma *pred_intros_disj1* [measurable (raw)]:

$(\neg K \implies \text{pred } M (\lambda x. P x)) \implies \text{pred } M (\lambda x. K \vee P x)$

by (cases K) auto

lemma *pred_intros_disj2* [measurable (raw)]:

$(\neg K \implies \text{pred } M (\lambda x. P x)) \implies \text{pred } M (\lambda x. P x \vee K)$

by (cases K) auto

lemma *pred_intros_logic* [measurable (raw)]:

$\text{pred } M (\lambda x. x \in \text{space } M)$

$\text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. \neg P x)$

$\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \wedge P x)$

$\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \longrightarrow P x)$

$\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x \vee P x)$

$\text{pred } M (\lambda x. Q x) \implies \text{pred } M (\lambda x. P x) \implies \text{pred } M (\lambda x. Q x = P x)$

$\text{pred } M (\lambda x. f x \in \text{UNIV})$

$\text{pred } M (\lambda x. f x \in \{\})$

$\text{pred } M (\lambda x. P' (f x) x) \implies \text{pred } M (\lambda x. f x \in \{y. P' y x\})$

$\text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in - (B x))$

$\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) - (B x))$

$\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) \cap (B x))$

$\text{pred } M (\lambda x. f x \in (A x)) \implies \text{pred } M (\lambda x. f x \in (B x)) \implies \text{pred } M (\lambda x. f x \in (A x) \cup (B x))$

$\text{pred } M (\lambda x. g x (f x) \in (X x)) \implies \text{pred } M (\lambda x. f x \in (g x) - ' (X x))$

by (auto simp: iff_conv_conj_imp pred_def)

lemma *pred_intros_countable* [measurable (raw)]:

fixes $P :: 'a \Rightarrow 'i :: \text{countable} \Rightarrow \text{bool}$

shows

$(\bigwedge i. \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i. P x i)$

$(\bigwedge i. \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i. P x i)$

by (auto intro!: sets.sets_Collect_countable_All sets.sets_Collect_countable_Ex simp: pred_def)

lemma *pred_intros_countable_bounded* [measurable (raw)]:

fixes $X :: 'i :: \text{countable set}$

shows

$(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcap_{i \in X}. N x i))$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcup_{i \in X}. N x i))$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i \in X. P x i)$
 $(\bigwedge i. i \in X \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i \in X. P x i)$
by *simp_all (auto simp: Bex_def Ball_def)*

lemma *pred_intros_finite*[*measurable (raw)*]:

$\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcap_{i \in I}. N x i))$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. x \in N x i)) \implies \text{pred } M (\lambda x. x \in (\bigcup_{i \in I}. N x i))$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \forall i \in I. P x i)$
 $\text{finite } I \implies (\bigwedge i. i \in I \implies \text{pred } M (\lambda x. P x i)) \implies \text{pred } M (\lambda x. \exists i \in I. P x i)$
by (*auto intro!: sets.sets_Collect_finite_Ex sets.sets_Collect_finite_All simp: iff_conv_conj_imp pred_def*)

lemma *countable_Un_Int*[*measurable (raw)*]:

$(\bigwedge i :: 'i :: \text{countable}. i \in I \implies N i \in \text{sets } M) \implies (\bigcup_{i \in I}. N i) \in \text{sets } M$
 $I \neq \{\} \implies (\bigwedge i :: 'i :: \text{countable}. i \in I \implies N i \in \text{sets } M) \implies (\bigcap_{i \in I}. N i) \in \text{sets } M$
by *auto*

declare

finite_UN[*measurable (raw)*]
finite_INT[*measurable (raw)*]

lemma *sets_Int_pred*[*measurable (raw)*]:

assumes *space*: $A \cap B \subseteq \text{space } M$ **and** [*measurable*]: $\text{pred } M (\lambda x. x \in A)$ $\text{pred } M (\lambda x. x \in B)$
shows $A \cap B \in \text{sets } M$
proof –
have $\{x \in \text{space } M. x \in A \cap B\} \in \text{sets } M$ **by** *auto*
also have $\{x \in \text{space } M. x \in A \cap B\} = A \cap B$
using *space* **by** *auto*
finally show *?thesis* .
qed

lemma [*measurable (raw generic)*]:

assumes *f*: $f \in \text{measurable } M N$ **and** *c*: $c \in \text{space } N \implies \{c\} \in \text{sets } N$
shows *pred_eq_const1*: $\text{pred } M (\lambda x. f x = c)$
and *pred_eq_const2*: $\text{pred } M (\lambda x. c = f x)$
proof –
show $\text{pred } M (\lambda x. f x = c)$
proof *cases*
assume $c \in \text{space } N$
with *measurable_sets*[*OF f c*] **show** *?thesis*
by (*auto simp: Int_def conj_commute pred_def*)
next
assume $c \notin \text{space } N$

```

  with f[THEN measurable_space] have  $\{x \in \text{space } M. f x = c\} = \{\}$  by auto
  then show ?thesis by (auto simp: pred_def cong: conj_cong)
qed
then show pred M ( $\lambda x. c = f x$ )
  by (simp add: eq_commute)
qed

```

```

lemma pred_count_space_const1[measurable (raw)]:
   $f \in \text{measurable } M (\text{count\_space } UNIV) \implies \text{Measurable.pred } M (\lambda x. f x = c)$ 
  by (intro pred_eq_const1[where N=count_space UNIV]) (auto)

```

```

lemma pred_count_space_const2[measurable (raw)]:
   $f \in \text{measurable } M (\text{count\_space } UNIV) \implies \text{Measurable.pred } M (\lambda x. c = f x)$ 
  by (intro pred_eq_const2[where N=count_space UNIV]) (auto)

```

```

lemma pred_le_const[measurable (raw generic)]:
  assumes  $f: f \in \text{measurable } M N$  and  $c: \{.. c\} \in \text{sets } N$  shows pred M ( $\lambda x. f x \leq c$ )
  using measurable_sets[OF f c]
  by (auto simp: Int_def conj_commute eq_commute pred_def)

```

```

lemma pred_const_le[measurable (raw generic)]:
  assumes  $f: f \in \text{measurable } M N$  and  $c: \{c ..\} \in \text{sets } N$  shows pred M ( $\lambda x. c \leq f x$ )
  using measurable_sets[OF f c]
  by (auto simp: Int_def conj_commute eq_commute pred_def)

```

```

lemma pred_less_const[measurable (raw generic)]:
  assumes  $f: f \in \text{measurable } M N$  and  $c: \{.. < c\} \in \text{sets } N$  shows pred M ( $\lambda x. f x < c$ )
  using measurable_sets[OF f c]
  by (auto simp: Int_def conj_commute eq_commute pred_def)

```

```

lemma pred_const_less[measurable (raw generic)]:
  assumes  $f: f \in \text{measurable } M N$  and  $c: \{c < ..\} \in \text{sets } N$  shows pred M ( $\lambda x. c < f x$ )
  using measurable_sets[OF f c]
  by (auto simp: Int_def conj_commute eq_commute pred_def)

```

```

declare
  sets.Int[measurable (raw)]

```

```

lemma pred_in_If[measurable (raw)]:
   $(P \implies \text{pred } M (\lambda x. x \in A x)) \implies (\neg P \implies \text{pred } M (\lambda x. x \in B x)) \implies$ 
   $\text{pred } M (\lambda x. x \in (\text{if } P \text{ then } A x \text{ else } B x))$ 
  by auto

```

```

lemma sets_range[measurable_dest]:
   $A ' I \subseteq \text{sets } M \implies i \in I \implies A i \in \text{sets } M$ 

```

2084

by *auto*

lemma *pred_sets_range*[*measurable_dest*]:

$A \text{ ' } I \subseteq \text{sets } N \implies i \in I \implies f \in \text{measurable } M \ N \implies \text{pred } M \ (\lambda x. f \ x \in A \ i)$
using *pred_sets2*[*OF sets_range*] **by** *auto*

lemma *sets_All*[*measurable_dest*]:

$\forall i. A \ i \in \text{sets } (M \ i) \implies A \ i \in \text{sets } (M \ i)$
by *auto*

lemma *pred_sets_All*[*measurable_dest*]:

$\forall i. A \ i \in \text{sets } (N \ i) \implies f \in \text{measurable } M \ (N \ i) \implies \text{pred } M \ (\lambda x. f \ x \in A \ i)$
using *pred_sets2*[*OF sets_All, of A N f*] **by** *auto*

lemma *sets_Ball*[*measurable_dest*]:

$\forall i \in I. A \ i \in \text{sets } (M \ i) \implies i \in I \implies A \ i \in \text{sets } (M \ i)$
by *auto*

lemma *pred_sets_Ball*[*measurable_dest*]:

$\forall i \in I. A \ i \in \text{sets } (N \ i) \implies i \in I \implies f \in \text{measurable } M \ (N \ i) \implies \text{pred } M \ (\lambda x. f \ x \in A \ i)$
using *pred_sets2*[*OF sets_Ball, of _ _ _ f*] **by** *auto*

lemma *measurable_finite*[*measurable (raw)*]:

fixes $S :: 'a \Rightarrow \text{nat set}$
assumes [*measurable*]: $\bigwedge i. \{x \in \text{space } M. i \in S \ x\} \in \text{sets } M$
shows $\text{pred } M \ (\lambda x. \text{finite } (S \ x))$
unfolding *finite_nat_set_iff_bounded* **by** (*simp add: Ball_def*)

lemma *measurable_Least*[*measurable*]:

assumes [*measurable*]: $(\bigwedge i :: \text{nat}. (\lambda x. P \ i \ x) \in \text{measurable } M \ (\text{count_space } UNIV))$
shows $(\lambda x. \text{LEAST } i. P \ i \ x) \in \text{measurable } M \ (\text{count_space } UNIV)$
unfolding *measurable_def* **by** (*safe intro!: sets_Least*) *simp_all*

lemma *measurable_Max_nat*[*measurable (raw)*]:

fixes $P :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$
assumes [*measurable*]: $\bigwedge i. \text{Measurable.pred } M \ (P \ i)$
shows $(\lambda x. \text{Max } \{i. P \ i \ x\}) \in \text{measurable } M \ (\text{count_space } UNIV)$
unfolding *measurable_count_space_eq2_countable*
proof *safe*
fix n

{ **fix** x **assume** $\forall i. \exists n \geq i. P \ n \ x$
then have *infinite* $\{i. P \ i \ x\}$
unfolding *infinite_nat_iff_unbounded_le* **by** *auto*
then have $\text{Max } \{i. P \ i \ x\} = \text{the None}$
by (*rule Max.infinite*) }
note $1 = \text{this}$


```

{ fix x i j assume P i x  $\forall n \geq j. \neg P n x$ 
  then have finite {i. P i x}
    by (auto simp: subset_eq not_le[symmetric] finite_nat_iff_bounded)
  with <P i x> have P (Max {i. P i x}) x i  $\leq$  Max {i. P i x} finite {i. P i x}
    using Max_in[of {i. P i x}] by auto }
note 2 = this

have ( $\lambda x. \text{Max } \{i. P i x\}$ ) -' {n}  $\cap$  space M = {x  $\in$  space M. Max {i. P i x} =
n}
  by auto
also have ... =
  {x  $\in$  space M. if ( $\forall i. \exists n \geq i. P n x$ ) then the None = n else
  if ( $\exists i. P i x$ ) then P n x  $\wedge$  ( $\forall i > n. \neg P i x$ )
  else Max {} = n}
  by (intro arg_cong[where f=Collect] ext conj_cong)
  (auto simp add: 1 2 not_le[symmetric] intro!: Max_eqI)
also have ...  $\in$  sets M
  by measurable
finally show ( $\lambda x. \text{Max } \{i. P i x\}$ ) -' {n}  $\cap$  space M  $\in$  sets M .
qed simp

```

```

lemma measurable_Min_nat[measurable (raw)]:
  fixes P :: nat  $\Rightarrow$  'a  $\Rightarrow$  bool
  assumes [measurable]:  $\bigwedge i. \text{Measurable.pred } M (P i)$ 
  shows ( $\lambda x. \text{Min } \{i. P i x\}$ )  $\in$  measurable M (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
  proof safe
    fix n

```

```

{ fix x assume  $\forall i. \exists n \geq i. P n x$ 
  then have infinite {i. P i x}
    unfolding infinite_nat_iff_unbounded_le by auto
  then have Min {i. P i x} = the None
    by (rule Min.infinite) }
note 1 = this

```

```

{ fix x i j assume P i x  $\forall n \geq j. \neg P n x$ 
  then have finite {i. P i x}
    by (auto simp: subset_eq not_le[symmetric] finite_nat_iff_bounded)
  with <P i x> have P (Min {i. P i x}) x Min {i. P i x}  $\leq$  i finite {i. P i x}
    using Min_in[of {i. P i x}] by auto }
note 2 = this

```

```

have ( $\lambda x. \text{Min } \{i. P i x\}$ ) -' {n}  $\cap$  space M = {x  $\in$  space M. Min {i. P i x} =
n}
  by auto
also have ... =
  {x  $\in$  space M. if ( $\forall i. \exists n \geq i. P n x$ ) then the None = n else

```

if $(\exists i. P i x)$ then $P n x \wedge (\forall i < n. \neg P i x)$
 else $Min \{ \} = n$
 by (intro arg_cong[where f=Collect] ext conj_cong)
 (auto simp add: 1 2 not_le[symmetric] intro!: Min_eqI)
 also have ... \in sets M
 by measurable
 finally show $(\lambda x. Min \{ i. P i x \}) - \{ n \} \cap space M \in sets M$.
 qed simp

lemma measurable_count_space_insert[measurable (raw)]:
 $s \in S \implies A \in sets (count_space S) \implies insert s A \in sets (count_space S)$
 by simp

lemma sets_UNIV [measurable (raw)]: $A \in sets (count_space UNIV)$
 by simp

lemma measurable_card[measurable]:
 fixes $S :: 'a \Rightarrow nat set$
 assumes [measurable]: $\bigwedge i. \{ x \in space M. i \in S x \} \in sets M$
 shows $(\lambda x. card (S x)) \in measurable M (count_space UNIV)$
 unfolding measurable_count_space_eq2_countable
 proof safe
 fix n show $(\lambda x. card (S x)) - \{ n \} \cap space M \in sets M$
 proof (cases n)
 case 0
 then have $(\lambda x. card (S x)) - \{ n \} \cap space M = \{ x \in space M. infinite (S x) \vee$
 $(\forall i. i \notin S x) \}$
 by auto
 also have ... \in sets M
 by measurable
 finally show ?thesis .
 next
 case (Suc i)
 then have $(\lambda x. card (S x)) - \{ n \} \cap space M =$
 $(\bigcup F \in \{ A \in \{ A. finite A \}. card A = n \}. \{ x \in space M. (\forall i. i \in S x \longleftrightarrow i \in F) \})$
 unfolding set_eq_iff[symmetric] Collect_bex_eq[symmetric] by (auto intro:
 card_ge_0_finite)
 also have ... \in sets M
 by (intro sets.countable_UN' countable_Collect countable_Collect_finite)
 auto
 finally show ?thesis .
 qed
 qed rule

lemma measurable_pred_countable[measurable (raw)]:
 assumes countable X
 shows
 $(\bigwedge i. i \in X \implies Measurable.pred M (\lambda x. P x i)) \implies Measurable.pred M (\lambda x.$

```

 $\forall i \in X. P x i$ 
  ( $\bigwedge i. i \in X \implies \text{Measurable.pred } M (\lambda x. P x i) \implies \text{Measurable.pred } M (\lambda x.$ 
 $\exists i \in X. P x i)$ 
  unfolding pred_def
  by (auto intro!: sets.sets_Collect_countable_All' sets.sets_Collect_countable_Ex'
  assms)

```

7.2.1 Measurability for (co)inductive predicates

```

lemma measurable_bot[measurable]: bot  $\in$  measurable  $M$  (count_space UNIV)
  by (simp add: bot_fun_def)

```

```

lemma measurable_top[measurable]: top  $\in$  measurable  $M$  (count_space UNIV)
  by (simp add: top_fun_def)

```

```

lemma measurable_SUP[measurable]:
  fixes  $F :: 'i \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\}$ 
  assumes [simp]: countable  $I$ 
  assumes [measurable]:  $\bigwedge i. i \in I \implies F i \in \text{measurable } M$  (count_space UNIV)
  shows ( $\lambda x. \text{SUP } i \in I. F i x$ )  $\in$  measurable  $M$  (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
proof (safe intro!: UNIV_I)
  fix  $a$ 
  have ( $\lambda x. \text{SUP } i \in I. F i x$ ) - '  $\{a\} \cap \text{space } M =$ 
    { $x \in \text{space } M. (\forall i \in I. F i x \leq a) \wedge (\forall b. (\forall i \in I. F i x \leq b) \longrightarrow a \leq b)$ }
  unfolding SUP_le_iff[symmetric] by auto
  also have ...  $\in$  sets  $M$ 
  by measurable
  finally show ( $\lambda x. \text{SUP } i \in I. F i x$ ) - '  $\{a\} \cap \text{space } M \in \text{sets } M$  .
qed

```

```

lemma measurable_INF[measurable]:
  fixes  $F :: 'i \Rightarrow 'a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\}$ 
  assumes [simp]: countable  $I$ 
  assumes [measurable]:  $\bigwedge i. i \in I \implies F i \in \text{measurable } M$  (count_space UNIV)
  shows ( $\lambda x. \text{INF } i \in I. F i x$ )  $\in$  measurable  $M$  (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
proof (safe intro!: UNIV_I)
  fix  $a$ 
  have ( $\lambda x. \text{INF } i \in I. F i x$ ) - '  $\{a\} \cap \text{space } M =$ 
    { $x \in \text{space } M. (\forall i \in I. a \leq F i x) \wedge (\forall b. (\forall i \in I. b \leq F i x) \longrightarrow b \leq a)$ }
  unfolding le_INF_iff[symmetric] by auto
  also have ...  $\in$  sets  $M$ 
  by measurable
  finally show ( $\lambda x. \text{INF } i \in I. F i x$ ) - '  $\{a\} \cap \text{space } M \in \text{sets } M$  .
qed

```

```

lemma measurable_lfp_coinduct[consumes 1, case_names continuity step]:
  fixes  $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b :: \{\text{complete\_lattice, countable}\})$ 

```

```

assumes P M
assumes F: sup_continuous F
assumes *:  $\bigwedge M A. P M \implies (\bigwedge N. P N \implies A \in \text{measurable } N \text{ (count\_space UNIV)}) \implies F A \in \text{measurable } M \text{ (count\_space UNIV)}$ 
shows lfp F  $\in$  measurable M (count_space UNIV)
proof -
  { fix i from  $\langle P M \rangle$  have ((F  $\sim$  i) bot)  $\in$  measurable M (count_space UNIV)
    by (induct i arbitrary: M) (auto intro!: *) }
  then have ( $\lambda x. \text{SUP } i. (F \sim i) \text{ bot } x$ )  $\in$  measurable M (count_space UNIV)
    by measurable
  also have ( $\lambda x. \text{SUP } i. (F \sim i) \text{ bot } x$ ) = lfp F
    by (subst sup_continuous_lfp) (auto intro: F simp: image_comp)
  finally show ?thesis .
qed

```

```

lemma measurable_lfp:
  fixes F :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  ('a  $\Rightarrow$  'b::{complete_lattice, countable})
  assumes F: sup_continuous F
  assumes *:  $\bigwedge A. A \in \text{measurable } M \text{ (count\_space UNIV)} \implies F A \in \text{measurable } M \text{ (count\_space UNIV)}$ 
  shows lfp F  $\in$  measurable M (count_space UNIV)
  by (coinduction rule: measurable_lfp_coinduct[OF _ F]) (blast intro: *)

```

```

lemma measurable_gfp_coinduct[consumes 1, case_names continuity step]:
  fixes F :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  ('a  $\Rightarrow$  'b::{complete_lattice, countable})
  assumes P M
  assumes F: inf_continuous F
  assumes *:  $\bigwedge M A. P M \implies (\bigwedge N. P N \implies A \in \text{measurable } N \text{ (count\_space UNIV)}) \implies F A \in \text{measurable } M \text{ (count\_space UNIV)}$ 
  shows gfp F  $\in$  measurable M (count_space UNIV)
proof -
  { fix i from  $\langle P M \rangle$  have ((F  $\sim$  i) top)  $\in$  measurable M (count_space UNIV)
    by (induct i arbitrary: M) (auto intro!: *) }
  then have ( $\lambda x. \text{INF } i. (F \sim i) \text{ top } x$ )  $\in$  measurable M (count_space UNIV)
    by measurable
  also have ( $\lambda x. \text{INF } i. (F \sim i) \text{ top } x$ ) = gfp F
    by (subst inf_continuous_gfp) (auto intro: F simp: image_comp)
  finally show ?thesis .
qed

```

```

lemma measurable_gfp:
  fixes F :: ('a  $\Rightarrow$  'b)  $\Rightarrow$  ('a  $\Rightarrow$  'b::{complete_lattice, countable})
  assumes F: inf_continuous F
  assumes *:  $\bigwedge A. A \in \text{measurable } M \text{ (count\_space UNIV)} \implies F A \in \text{measurable } M \text{ (count\_space UNIV)}$ 
  shows gfp F  $\in$  measurable M (count_space UNIV)
  by (coinduction rule: measurable_gfp_coinduct[OF _ F]) (blast intro: *)

```

```

lemma measurable_lfp2_coinduct[consumes 1, case_names continuity step]:

```

```

fixes  $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b) :: \{complete\_lattice, countable\}$ 
assumes  $P M s$ 
assumes  $F: sup\_continuous F$ 
assumes  $*$ :  $\bigwedge M A s. P M s \Longrightarrow (\bigwedge N t. P N t \Longrightarrow A t \in measurable N$ 
 $(count\_space UNIV)) \Longrightarrow F A s \in measurable M (count\_space UNIV)$ 
shows  $lfp F s \in measurable M (count\_space UNIV)$ 
proof -
  { fix  $i$  from  $\langle P M s \rangle$  have  $(\lambda x. (F \rightsquigarrow i) bot s x) \in measurable M (count\_space UNIV)$ 
    by  $(induct i arbitrary: M s) (auto intro!: *)$  }
  then have  $(\lambda x. SUP i. (F \rightsquigarrow i) bot s x) \in measurable M (count\_space UNIV)$ 
    by measurable
  also have  $(\lambda x. SUP i. (F \rightsquigarrow i) bot s x) = lfp F s$ 
    by  $(subst sup\_continuous\_lfp) (auto simp: F simp: image\_comp)$ 
  finally show  $?thesis$  .
qed

```

```

lemma measurable_gfp2_coinduct $[consumes 1, case\_names continuity step]$ :
fixes  $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b) :: \{complete\_lattice, countable\}$ 
assumes  $P M s$ 
assumes  $F: inf\_continuous F$ 
assumes  $*$ :  $\bigwedge M A s. P M s \Longrightarrow (\bigwedge N t. P N t \Longrightarrow A t \in measurable N$ 
 $(count\_space UNIV)) \Longrightarrow F A s \in measurable M (count\_space UNIV)$ 
shows  $gfp F s \in measurable M (count\_space UNIV)$ 
proof -
  { fix  $i$  from  $\langle P M s \rangle$  have  $(\lambda x. (F \rightsquigarrow i) top s x) \in measurable M (count\_space UNIV)$ 
    by  $(induct i arbitrary: M s) (auto intro!: *)$  }
  then have  $(\lambda x. INF i. (F \rightsquigarrow i) top s x) \in measurable M (count\_space UNIV)$ 
    by measurable
  also have  $(\lambda x. INF i. (F \rightsquigarrow i) top s x) = gfp F s$ 
    by  $(subst inf\_continuous\_gfp) (auto simp: F simp: image\_comp)$ 
  finally show  $?thesis$  .
qed

```

```

lemma measurable_enat_coinduct:
fixes  $f :: 'a \Rightarrow enat$ 
assumes  $R f$ 
assumes  $*$ :  $\bigwedge f. R f \Longrightarrow \exists g h i P. R g \wedge f = (\lambda x. if P x then h x else eSuc (g$ 
 $(i x))) \wedge$ 
 $Measurable.pred M P \wedge$ 
 $i \in measurable M M \wedge$ 
 $h \in measurable M (count\_space UNIV)$ 
shows  $f \in measurable M (count\_space UNIV)$ 
proof  $(simp add: measurable\_count\_space\_eq2\_countable, rule)$ 
fix  $a :: enat$ 
have  $f - \{a\} \cap space M = \{x \in space M. f x = a\}$ 
by auto
  { fix  $i :: nat$ 

```

```

from ⟨R f⟩ have Measurable.pred M (λx. f x = enat i)
proof (induction i arbitrary: f)
  case 0
  from *[OF this] obtain g h i P
    where f: f = (λx. if P x then h x else eSuc (g (i x))) and
      [measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable M
(count_space UNIV)
  by auto
  have Measurable.pred M (λx. P x ∧ h x = 0)
  by measurable
  also have (λx. P x ∧ h x = 0) = (λx. f x = enat 0)
  by (auto simp: f zero_enat_def[symmetric])
  finally show ?case .
next
  case (Suc n)
  from *[OF Suc.prem] obtain g h i P
    where f: f = (λx. if P x then h x else eSuc (g (i x))) and R g and
      M [measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable
M (count_space UNIV)
  by auto
  have (λx. f x = enat (Suc n)) =
    (λx. (P x → h x = enat (Suc n)) ∧ (¬ P x → g (i x) = enat n))
  by (auto simp: f zero_enat_def[symmetric] eSuc_enat[symmetric])
  also have Measurable.pred M ...
    by (intro pred_intros_logic measurable_compose[OF M(2)] Suc ⟨R g⟩)
measurable
  finally show ?case .
  qed
  then have f - ' {enat i} ∩ space M ∈ sets M
  by (simp add: pred_def Int_def conj_commute) }
note fin = this
show f - ' {a} ∩ space M ∈ sets M
proof (cases a)
  case infinity
  then have f - ' {a} ∩ space M = space M - (∪ n. f - ' {enat n} ∩ space M)
  by auto
  also have ... ∈ sets M
  by (intro sets.Diff sets.top sets.Un sets.countable_UN) (auto intro!: fin)
  finally show ?thesis .
qed (simp add: fin)
qed

```

lemma measurable_THE:

```

fixes P :: 'a ⇒ 'b ⇒ bool
assumes [measurable]: Λi. Measurable.pred M (P i)
assumes I[simp]: countable I ∧ i x. x ∈ space M ⇒ P i x ⇒ i ∈ I
assumes unique: Λx i j. x ∈ space M ⇒ P i x ⇒ P j x ⇒ i = j
shows (λx. THE i. P i x) ∈ measurable M (count_space UNIV)
unfolding measurable_def

```

```

proof safe
  fix X
  define f where f x = (THE i. P i x) for x
  define undef where undef = (THE i::'a. False)
  { fix i x assume x ∈ space M P i x then have f x = i
    unfolding f_def using unique by auto }
  note f_eq = this
  { fix x assume x ∈ space M  $\forall i \in I. \neg P i x$ 
    then have  $\bigwedge i. \neg P i x$ 
    using I(2)[of x] by auto
    then have f x = undef
    by (auto simp: undef_def f_def) }
  then have f - ' X ∩ space M = ( $\bigcup i \in I \cap X. \{x \in \text{space } M. P i x\}$ ) ∪
    (if undef ∈ X then space M - ( $\bigcup i \in I. \{x \in \text{space } M. P i x\}$ ) else {})
    by (auto dest: f_eq)
  also have ... ∈ sets M
    by (auto intro!: sets.Diff sets.countable_UN')
  finally show f - ' X ∩ space M ∈ sets M .
qed simp

```

```

lemma measurable_Ex1[measurable (raw)]:
  assumes [simp]: countable I and [measurable]:  $\bigwedge i. i \in I \implies \text{Measurable.pred } M$ 
  (P i)
  shows Measurable.pred M (λx.  $\exists ! i \in I. P i x$ )
  unfolding bex1_def by measurable

```

```

lemma measurable_Sup_nat[measurable (raw)]:
  fixes F :: 'a ⇒ nat set
  assumes [measurable]:  $\bigwedge i. \text{Measurable.pred } M (\lambda x. i \in F x)$ 
  shows (λx. Sup (F x)) ∈ M →M count_space UNIV
proof (clarsimp simp add: measurable_count_space_eq2_countable)
  fix a
  have F_empty_iff: F x = {} ↔ (∀ i. i ∉ F x) for x
    by auto
  have Measurable.pred M (λx. if finite (F x) then if F x = {} then a = 0
    else a ∈ F x ∧ (∀ j. j ∈ F x → j ≤ a) else a = the None)
    unfolding finite_nat_set_iff_bounded Ball_def F_empty_iff by measurable
  moreover have (λx. Sup (F x)) - ' {a} ∩ space M =
    {x ∈ space M. if finite (F x) then if F x = {} then a = 0
    else a ∈ F x ∧ (∀ j. j ∈ F x → j ≤ a) else a = the None}
    by (intro set_eqI)
    (auto simp: Sup_nat_def Max.infinite intro!: Max_in Max_eqI)
  ultimately show (λx. Sup (F x)) - ' {a} ∩ space M ∈ sets M
    by auto
qed

```

```

lemma measurable_if_split[measurable (raw)]:
  (c ⇒ Measurable.pred M f) ⇒ (¬ c ⇒ Measurable.pred M g) ⇒
  Measurable.pred M (if c then f else g)

```

2092

by *simp*

lemma *pred_restrict_space*:

assumes $S \in \text{sets } M$

shows $\text{Measurable.pred } (\text{restrict_space } M S) P \longleftrightarrow \text{Measurable.pred } M (\lambda x. x \in S \wedge P x)$

unfolding *pred_def sets_Collect_restrict_space_iff*[*OF assms*] ..

lemma *measurable_predpow*[*measurable*]:

assumes $\text{Measurable.pred } M T$

assumes $\bigwedge Q. \text{Measurable.pred } M Q \implies \text{Measurable.pred } M (R Q)$

shows $\text{Measurable.pred } M ((R \overset{\sim}{\sim} n) T)$

by (*induct n*) (*auto intro: assms*)

lemma *measurable_compose_countable_restrict*:

assumes $P: \text{countable } \{i. P i\}$

and $f: f \in M \rightarrow_M \text{count_space UNIV}$

and $Q: \bigwedge i. P i \implies \text{pred } M (Q i)$

shows $\text{pred } M (\lambda x. P (f x) \wedge Q (f x) x)$

proof –

have $P_f: \{x \in \text{space } M. P (f x)\} \in \text{sets } M$

unfolding *pred_def[symmetric]* **by** (*rule measurable_compose*[*OF f*]) *simp*

have $\text{pred } (\text{restrict_space } M \{x \in \text{space } M. P (f x)\}) (\lambda x. Q (f x) x)$

proof (*rule measurable_compose_countable*[**where** $g=f, OF _ _ P$])

show $f \in \text{restrict_space } M \{x \in \text{space } M. P (f x)\} \rightarrow_M \text{count_space } \{i. P i\}$

by (*rule measurable_count_space_extend*[*OF subset_UNIV*])

(*auto simp: space_restrict_space intro!: measurable_restrict_space1 f*)

qed (*auto intro!: measurable_restrict_space1 Q*)

then show *?thesis*

unfolding *pred_restrict_space*[*OF P_f*] **by** (*simp cong: measurable_cong*)

qed

lemma *measurable_limsup* [*measurable (raw)*]:

assumes [*measurable*]: $\bigwedge n. A n \in \text{sets } M$

shows $\text{limsup } A \in \text{sets } M$

by (*subst limsup_INF_SUP, auto*)

lemma *measurable_liminf* [*measurable (raw)*]:

assumes [*measurable*]: $\bigwedge n. A n \in \text{sets } M$

shows $\text{liminf } A \in \text{sets } M$

by (*subst liminf_SUP_INF, auto*)

lemma *measurable_case_enat*[*measurable (raw)*]:

assumes $f: f \in M \rightarrow_M \text{count_space UNIV}$ **and** $g: \bigwedge i. g i \in M \rightarrow_M N$ **and** $h: h \in M \rightarrow_M N$

shows $(\lambda x. \text{case } f x \text{ of } \text{enat } i \Rightarrow g i x \mid \infty \Rightarrow h x) \in M \rightarrow_M N$

apply (*rule measurable_compose_countable*[*OF _ f*])

subgoal for i

by (*cases i*) (*auto intro: g h*)


```

done

hide_const (open) pred

end

```

7.3 Measure Spaces

```

theory Measure_Space
imports
  Measurable HOL-Library.Extended_Nonnegative_Real
begin

```

7.3.1 Relate extended reals and the indicator function

```

lemma suminf_cmult_indicator:
  fixes f :: nat ⇒ ennreal
  assumes disjoint_family A x ∈ A i
  shows (∑ n. f n * indicator (A n) x) = f i
proof -
  have **: ∧ n. f n * indicator (A n) x = (if n = i then f n else 0 :: ennreal)
    using ⟨x ∈ A i⟩ assms unfolding disjoint_family_on_def indicator_def by
  auto
  then have ∧ n. (∑ j < n. f j * indicator (A j) x) = (if i < n then f i else 0 ::
ennreal)
    by (auto simp: sum.If_cases)
  moreover have (SUP n. if i < n then f i else 0) = (f i :: ennreal)
  proof (rule SUP_eqI)
    fix y :: ennreal assume ∧ n. n ∈ UNIV ⇒ (if i < n then f i else 0) ≤ y
    from this[of Suc i] show f i ≤ y by auto
  qed (use assms in simp)
  ultimately show ?thesis using assms
    by (simp add: suminf_eq_SUP)
qed

```

```

lemma suminf_indicator:
  assumes disjoint_family A
  shows (∑ n. indicator (A n) x :: ennreal) = indicator (⋃ i. A i) x
proof cases
  assume *: x ∈ (⋃ i. A i)
  then obtain i where x ∈ A i by auto
  from suminf_cmult_indicator[OF assms(1), OF ⟨x ∈ A i⟩, of λk. 1]
  show ?thesis using * by simp
qed simp

```

```

lemma sum_indicator_disjoint_family:
  fixes f :: 'd ⇒ 'e::semiring_1
  assumes d: disjoint_family_on A P and x ∈ A j and finite P and j ∈ P
  shows (∑ i ∈ P. f i * indicator (A i) x) = f j

```

```

proof –
  have  $P \cap \{i. x \in A\} = \{j\}$ 
    using  $d \langle x \in A \rangle \langle j \in P \rangle$  unfolding disjoint_family_on_def
    by auto
  with  $\langle \text{finite } P \rangle$  show ?thesis
    by (simp add: indicator_def)
qed

```

The type for emeasure spaces is already defined in *HOL-Analysis.Sigma_Algebra*, as it is also used to represent sigma algebras (with an arbitrary emeasure).

7.3.2 Extend binary sets

```

lemma LIMSEQ_binaryset:
  assumes  $f: f \{\} = 0$ 
  shows  $(\lambda n. \sum i < n. f (\text{binaryset } A \ B \ i)) \longrightarrow f \ A + f \ B$ 
proof –
  have  $(\lambda n. \sum i < \text{Suc } (\text{Suc } n). f (\text{binaryset } A \ B \ i)) = (\lambda n. f \ A + f \ B)$ 
  proof
    fix  $n$ 
    show  $(\sum i < \text{Suc } (\text{Suc } n). f (\text{binaryset } A \ B \ i)) = f \ A + f \ B$ 
    by (induct n) (auto simp add: binaryset_def)
  qed
  moreover
  have  $\dots \longrightarrow f \ A + f \ B$  by (rule tendsto_const)
  ultimately have  $(\lambda n. \sum i < n+2. f (\text{binaryset } A \ B \ i)) \longrightarrow f \ A + f \ B$ 
    by simp
  thus ?thesis by (rule LIMSEQ_offset [where k=2])
qed

```

```

lemma binaryset_sums:
  assumes  $f: f \{\} = 0$ 
  shows  $(\lambda n. f (\text{binaryset } A \ B \ n)) \text{ sums } (f \ A + f \ B)$ 
  using LIMSEQ_binaryset f_sums_def by blast

```

```

lemma suminf_binaryset_eq:
  fixes  $f :: 'a \ \text{set} \Rightarrow 'b::\{\text{comm\_monoid\_add}, \text{t2\_space}\}$ 
  shows  $f \{\} = 0 \implies (\sum n. f (\text{binaryset } A \ B \ n)) = f \ A + f \ B$ 
  by (metis binaryset_sums sums_unique)

```

7.3.3 Properties of a premeasure μ

The definitions for *positive* and *countably_additive* should be here, by they are necessary to define 'a measure in *HOL-Analysis.Sigma_Algebra*.

```

definition subadditive where
  subadditive  $M \ f \longleftrightarrow (\forall x \in M. \forall y \in M. x \cap y = \{\} \longrightarrow f (x \cup y) \leq f \ x + f \ y)$ 

```

```

lemma subadditiveD: subadditive  $M \ f \implies x \cap y = \{\} \implies x \in M \implies y \in M \implies f (x \cup y) \leq f \ x + f \ y$ 

```

by (auto simp add: subadditive_def)

definition *countably_subadditive* **where**

countably_subadditive $M f \longleftrightarrow$
 $(\forall A. \text{range } A \subseteq M \longrightarrow \text{disjoint_family } A \longrightarrow (\bigcup i. A \ i) \in M \longrightarrow (f (\bigcup i. A \ i) \leq (\sum i. f (A \ i))))$

lemma (in *ring_of_sets*) *countably_subadditive_subadditive*:

fixes $f :: 'a \text{ set} \Rightarrow \text{ennreal}$

assumes f : *positive* $M f$ **and** cs : *countably_subadditive* $M f$

shows *subadditive* $M f$

proof (auto simp add: subadditive_def)

fix $x \ y$

assume $x: x \in M$ **and** $y: y \in M$ **and** $x \cap y = \{\}$

hence *disjoint_family* (binaryset $x \ y$)

by (auto simp add: disjoint_family_on_def binaryset_def)

hence $\text{range (binaryset } x \ y) \subseteq M \longrightarrow$

$(\bigcup i. \text{binaryset } x \ y \ i) \in M \longrightarrow$

$f (\bigcup i. \text{binaryset } x \ y \ i) \leq (\sum n. f (\text{binaryset } x \ y \ n))$

using cs **by** (auto simp add: countably_subadditive_def)

hence $\{x, y, \{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow$

$f (x \cup y) \leq (\sum n. f (\text{binaryset } x \ y \ n))$

by (simp add: range_binaryset_eq UN_binaryset_eq)

thus $f (x \cup y) \leq f x + f y$ **using** $f \ x \ y$

by (auto simp add: Un_o_def suminf_binaryset_eq positive_def)

qed

definition *additive* **where**

additive $M \ \mu \longleftrightarrow (\forall x \in M. \forall y \in M. x \cap y = \{\} \longrightarrow \mu (x \cup y) = \mu x + \mu y)$

definition *increasing* **where**

increasing $M \ \mu \longleftrightarrow (\forall x \in M. \forall y \in M. x \subseteq y \longrightarrow \mu x \leq \mu y)$

lemma *positiveD1*: *positive* $M f \Longrightarrow f \ \{\} = 0$ **by** (auto simp: positive_def)

lemma *positiveD_empty*:

positive $M f \Longrightarrow f \ \{\} = 0$

by (auto simp add: positive_def)

lemma *additiveD*:

additive $M f \Longrightarrow x \cap y = \{\} \Longrightarrow x \in M \Longrightarrow y \in M \Longrightarrow f (x \cup y) = f x + f y$

by (auto simp add: additive_def)

lemma *increasingD*:

increasing $M f \Longrightarrow x \subseteq y \Longrightarrow x \in M \Longrightarrow y \in M \Longrightarrow f x \leq f y$

by (auto simp add: increasing_def)

lemma *countably_additiveI*[*case_names* *countably*]:

$(\bigwedge A. [\text{range } A \subseteq M; \text{disjoint_family } A; (\bigcup i. A \ i) \in M] \Longrightarrow (\sum i. f (A \ i)) =$

$f(\bigcup i. A i)$
 \implies *countably_additive* $M f$
 by (*simp add: countably_additive_def*)

lemma (*in ring_of_sets*) *disjointed_additive*:
 assumes f : *positive* $M f$ *additive* $M f$ **and** A : *range* $A \subseteq M$ *incseq* A
 shows $(\sum i \leq n. f (\text{disjointed } A i)) = f (A n)$
proof (*induct n*)
 case (*Suc n*)
 then have $(\sum i \leq \text{Suc } n. f (\text{disjointed } A i)) = f (A n) + f (\text{disjointed } A (\text{Suc } n))$
 by *simp*
 also have $\dots = f (A n \cup \text{disjointed } A (\text{Suc } n))$
 using A by (*subst* $f(2)$ [*THEN additiveD*]) (*auto simp: disjointed_mono*)
 also have $A n \cup \text{disjointed } A (\text{Suc } n) = A (\text{Suc } n)$
 using $\langle \text{incseq } A \rangle$ by (*auto dest: incseq_SucD simp: disjointed_mono*)
 finally show ?*case* .
qed *simp*

lemma (*in ring_of_sets*) *additive_sum*:
 fixes A :: 'i \Rightarrow 'a *set*
 assumes f : *positive* $M f$ **and** ad : *additive* $M f$ **and** *finite* S
 and A : $A S \subseteq M$
 and $disj$: *disjoint_family_on* $A S$
 shows $(\sum i \in S. f (A i)) = f (\bigcup i \in S. A i)$
 using $\langle \text{finite } S \rangle$ $disj A$
proof *induct*
 case *empty* show ?*case* using f by (*simp add: positive_def*)
next
 case (*insert s S*)
 then have $A s \cap (\bigcup i \in S. A i) = \{\}$
 by (*auto simp add: disjoint_family_on_def neq_iff*)
moreover
 have $A s \in M$ using *insert* by *blast*
moreover have $(\bigcup i \in S. A i) \in M$
 using *insert* $\langle \text{finite } S \rangle$ by *auto*
ultimately have $f (A s \cup (\bigcup i \in S. A i)) = f (A s) + f (\bigcup i \in S. A i)$
 using ad *UNION_in_sets* A by (*auto simp add: additive_def*)
with *insert* show ?*case* using ad *disjoint_family_on_mono*[*of S insert s S A*]
 by (*auto simp add: additive_def subset_insertI*)
qed

lemma (*in ring_of_sets*) *additive_increasing*:
 fixes f :: 'a *set* \Rightarrow *ennreal*
 assumes $posf$: *positive* $M f$ **and** $addf$: *additive* $M f$
 shows *increasing* $M f$
proof (*auto simp add: increasing_def*)
 fix $x y$
 assume xy : $x \in M y \in M x \subseteq y$
 then have $y - x \in M$ by *auto*

```

then have  $f x + 0 \leq f x + f (y-x)$  by (intro add_left_mono zero_le)
also have  $\dots = f (x \cup (y-x))$ 
  by (metis addf Diff_disjoint ‹ $y - x \in M$ › additiveD xy(1))
also have  $\dots = f y$ 
  by (metis Un_Diff_cancel Un_absorb1 xy(3))
finally show  $f x \leq f y$  by simp
qed

```

lemma (in ring_of_sets) subadditive:

```

fixes  $f :: 'a \text{ set} \Rightarrow \text{ennreal}$ 
assumes  $f$ : positive  $M f$  additive  $M f$  and  $A$ :  $A'S \subseteq M$  and  $S$ : finite  $S$ 
shows  $f (\bigcup i \in S. A i) \leq (\sum i \in S. f (A i))$ 
using  $S A$ 
proof (induct  $S$ )
  case empty thus ?case using  $f$  by (auto simp: positive_def)
next
  case (insert  $x F$ )
  hence  $\text{in\_}M$ :  $A x \in M (\bigcup i \in F. A i) \in M (\bigcup i \in F. A i) - A x \in M$  using  $A$ 
by force+
  have subs:  $(\bigcup i \in F. A i) - A x \subseteq (\bigcup i \in F. A i)$  by auto
  have  $(\bigcup i \in (\text{insert } x F). A i) = A x \cup ((\bigcup i \in F. A i) - A x)$  by auto
  hence  $f (\bigcup i \in (\text{insert } x F). A i) = f (A x \cup ((\bigcup i \in F. A i) - A x))$ 
    by simp
  also have  $\dots = f (A x) + f ((\bigcup i \in F. A i) - A x)$ 
    using  $f(2)$  by (rule additiveD) (insert in_M, auto)
  also have  $\dots \leq f (A x) + f (\bigcup i \in F. A i)$ 
    using additive_increasing[OF  $f$ ] in_M subs
    by (simp add: increasingD)
  also have  $\dots \leq f (A x) + (\sum i \in F. f (A i))$ 
    using insert by (auto intro: add_left_mono)
  finally show  $f (\bigcup i \in (\text{insert } x F). A i) \leq (\sum i \in (\text{insert } x F). f (A i))$ 
    by (simp add: insert)
qed

```

lemma (in ring_of_sets) countably_additive_additive:

```

fixes  $f :: 'a \text{ set} \Rightarrow \text{ennreal}$ 
assumes posf: positive  $M f$  and ca: countably_additive  $M f$ 
shows additive  $M f$ 
proof (auto simp add: additive_def)
  fix  $x y$ 
  assume  $x$ :  $x \in M$  and  $y$ :  $y \in M$  and  $x \cap y = \{\}$ 
  hence disjoint_family (binaryset  $x y$ )
    by (auto simp add: disjoint_family_on_def binaryset_def)
  hence range (binaryset  $x y$ )  $\subseteq M \longrightarrow$ 
     $(\bigcup i. \text{binaryset } x y i) \in M \longrightarrow$ 
     $f (\bigcup i. \text{binaryset } x y i) = (\sum n. f (\text{binaryset } x y n))$ 
    using ca by (simp add: countably_additive_def)
  hence  $\{x, y, \{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow f (x \cup y) = (\sum n. f (\text{binaryset } x y n))$ 
    by (simp add: range_binaryset_eq UN_binaryset_eq)

```

2098

thus $f (x \cup y) = f x + f y$ **using** *posf x y*
by (*auto simp add: Un suminf_binaryset_eq positive_def*)
qed

lemma (*in algebra*) *increasing_additive_bound*:
fixes $A :: \text{nat} \Rightarrow 'a \text{ set}$ **and** $f :: 'a \text{ set} \Rightarrow \text{ennreal}$
assumes f : *positive M f* **and** ad : *additive M f*
and inc : *increasing M f*
and A : *range A \subseteq M*
and $disj$: *disjoint_family A*
shows $(\sum i. f (A i)) \leq f \Omega$
proof (*safe intro!: suminf_le_const*)
fix N
note $disj_N = disjoint_family_on_mono[OF _ disj, of \{..<N\}]$
have $(\sum i < N. f (A i)) = f (\bigcup i \in \{..<N\}. A i)$
using A **by** (*intro additive_sum [OF f ad]*) (*auto simp: disj_N*)
also have $\dots \leq f \Omega$ **using** *space_closed A*
by (*intro increasingD [OF inc] finite_UN*) *auto*
finally show $(\sum i < N. f (A i)) \leq f \Omega$ **by** *simp*
qed (*insert f A, auto simp: positive_def*)

lemma (*in ring_of_sets*) *countably_additiveI_finite*:
fixes $\mu :: 'a \text{ set} \Rightarrow \text{ennreal}$
assumes $finite \Omega$ *positive M μ additive M μ*
shows *countably_additive M μ*
proof (*rule countably_additiveI*)
fix $F :: \text{nat} \Rightarrow 'a \text{ set}$ **assume** F : *range F \subseteq M* $(\bigcup i. F i) \in M$ **and** $disj$:
disjoint_family F

have $\forall i. F i \neq \{\}$ $\longrightarrow (\exists x. x \in F i)$ **by** *auto*
then obtain f **where** $f: \bigwedge i. F i \neq \{\} \implies f i \in F i$ **by** *metis*

have $finU$: *finite* $(\bigcup i. F i)$
by (*metis F(2) assms(1) infinite_super_sets_into_space*)

have F_subset : $\{i. \mu (F i) \neq 0\} \subseteq \{i. F i \neq \{\}$
by (*auto simp: positiveD_empty [OF \langle positive M μ \rangle]*)
moreover have fin_not_empty : *finite* $\{i. F i \neq \{\}$

proof (*rule finite_imageD*)
from f **have** $f\{i. F i \neq \{\}$ $\subseteq (\bigcup i. F i)$ **by** *auto*
then show *finite* $(f\{i. F i \neq \{\}$)
by (*simp add: finU finite_subset*)
show inj_f : *inj_on* $f \{i. F i \neq \{\}$
using f *disj*
by (*simp add: inj_on_def disjoint_family_on_def disjoint_iff*) *metis*

qed
ultimately have fin_not_0 : *finite* $\{i. \mu (F i) \neq 0\}$
by (*rule finite_subset*)

```

have disj_not_empty: disjoint_family_on F {i. F i ≠ {}}
  using disj by (auto simp: disjoint_family_on_def)

from fin_not_0 have (∑ i. μ (F i)) = (∑ i | μ (F i) ≠ 0. μ (F i))
  by (rule suminf_finite) auto
also have ... = (∑ i | F i ≠ {}. μ (F i))
  using fin_not_empty F_subset by (rule sum.mono_neutral_left) auto
also have ... = μ (⋃ i∈{i. F i ≠ {}}. F i)
  using ‹positive M μ› ‹additive M μ› fin_not_empty disj_not_empty F by
(intro additive_sum) auto
also have ... = μ (⋃ i. F i)
  by (rule arg_cong[where f=μ]) auto
finally show (∑ i. μ (F i)) = μ (⋃ i. F i) .
qed

```

lemma (in ring_of_sets) countably_additive_iff_continuous_from_below:

```

fixes f :: 'a set ⇒ ennreal
assumes f: positive M f additive M f
shows countably_additive M f ⟷
  (∀ A. range A ⊆ M ⟶ incseq A ⟶ (⋃ i. A i) ∈ M ⟶ (λi. f (A i)) ⟶
f (⋃ i. A i))
  unfolding countably_additive_def
proof safe
  assume count_sum: ∀ A. range A ⊆ M ⟶ disjoint_family A ⟶ ⋃ (A ' UNIV)
  ∈ M ⟶ (∑ i. f (A i)) = f (⋃ (A ' UNIV))
  fix A :: nat ⇒ 'a set assume A: range A ⊆ M incseq A (⋃ i. A i) ∈ M
  then have dA: range (disjointed A) ⊆ M by (auto simp: range_disjointed_sets)
  with count_sum[THEN spec, of disjointed A] A(3)
  have f_UN: (∑ i. f (disjointed A i)) = f (⋃ i. A i)
    by (auto simp: UN_disjointed_eq disjoint_family_disjointed)
  moreover have (λn. (∑ i<n. f (disjointed A i))) ⟶ (∑ i. f (disjointed A
i))
    by (simp add: summable_LIMSEQ)
  from LIMSEQ_Suc[OF this]
  have (λn. (∑ i≤n. f (disjointed A i))) ⟶ (∑ i. f (disjointed A i))
    unfolding lessThan_Suc_atMost .
  moreover have ∧n. (∑ i≤n. f (disjointed A i)) = f (A n)
    using disjointed_additive[OF f A(1,2)] .
  ultimately show (λi. f (A i)) ⟶ f (⋃ i. A i) by simp
next
  assume cont[rule_format]: ∀ A. range A ⊆ M ⟶ incseq A ⟶ (⋃ i. A i) ∈ M
  ⟶ (λi. f (A i)) ⟶ f (⋃ i. A i)
  fix A :: nat ⇒ 'a set assume A: range A ⊆ M disjoint_family A (⋃ i. A i) ∈ M
  have *: (⋃ n. (⋃ i<n. A i)) = (⋃ i. A i) by auto
  have range (λi. ⋃ i<i. A i) ⊆ M (⋃ i. ⋃ i<i. A i) ∈ M
    using A * by auto
  then have (λn. f (⋃ i<n. A i)) ⟶ f (⋃ i. A i)
    unfolding *[symmetric] by (force intro!: cont incseq_SucI)+
  moreover have ∧n. f (⋃ i<n. A i) = (∑ i<n. f (A i))

```

```

    using A
    by (intro additive_sum[OF f, symmetric]) (auto intro: disjoint_family_on_mono)
  ultimately
  have  $(\lambda i. f (A i)) \text{ sums } f (\bigcup i. A i)$ 
    unfolding sums_def by simp
  then show  $(\sum i. f (A i)) = f (\bigcup i. A i)$ 
    by (metis sums_unique)
qed

```

lemma (in ring_of_sets) continuous_from_above_iff_empty_continuous:

```

  fixes f :: 'a set  $\Rightarrow$  ennreal
  assumes f: positive M f additive M f
  shows  $(\forall A. \text{range } A \subseteq M \longrightarrow \text{decseq } A \longrightarrow (\bigcap i. A i) \in M \longrightarrow (\forall i. f (A i) \neq \infty) \longrightarrow (\lambda i. f (A i)) \longrightarrow f (\bigcap i. A i))$ 
     $\longleftrightarrow (\forall A. \text{range } A \subseteq M \longrightarrow \text{decseq } A \longrightarrow (\bigcap i. A i) = \{\} \longrightarrow (\forall i. f (A i) \neq \infty) \longrightarrow (\lambda i. f (A i)) \longrightarrow 0)$ 

```

proof safe

```

  assume cont[rule_format]:  $(\forall A. \text{range } A \subseteq M \longrightarrow \text{decseq } A \longrightarrow (\bigcap i. A i) \in M \longrightarrow (\forall i. f (A i) \neq \infty) \longrightarrow (\lambda i. f (A i)) \longrightarrow f (\bigcap i. A i))$ 

```

```

  fix A :: nat  $\Rightarrow$  'a set

```

```

  assume A:  $\text{range } A \subseteq M \text{ decseq } A (\bigcap i. A i) = \{\} \forall i. f (A i) \neq \infty$ 

```

```

  with cont[of A] show  $(\lambda i. f (A i)) \longrightarrow 0$ 

```

```

    using <positive M f>[unfolded positive_def] by auto

```

next

```

  assume cont[rule_format]:  $\forall A. \text{range } A \subseteq M \longrightarrow \text{decseq } A \longrightarrow (\bigcap i. A i) = \{\} \longrightarrow (\forall i. f (A i) \neq \infty) \longrightarrow (\lambda i. f (A i)) \longrightarrow 0$ 

```

```

  fix A :: nat  $\Rightarrow$  'a set

```

```

  assume A:  $\text{range } A \subseteq M \text{ decseq } A (\bigcap i. A i) \in M \forall i. f (A i) \neq \infty$ 

```

```

  have f_mono:  $\bigwedge a b. a \in M \Longrightarrow b \in M \Longrightarrow a \subseteq b \Longrightarrow f a \leq f b$ 

```

```

    using additive_increasing[OF f] unfolding increasing_def by simp

```

```

  have decseq_fA:  $\text{decseq } (\lambda i. f (A i))$ 

```

```

    using A by (auto simp: decseq_def intro!: f_mono)

```

```

  have decseq:  $\text{decseq } (\lambda i. A i - (\bigcap i. A i))$ 

```

```

    using A by (auto simp: decseq_def)

```

```

  then have decseq_f:  $\text{decseq } (\lambda i. f (A i - (\bigcap i. A i)))$ 

```

```

    using A unfolding decseq_def by (auto intro!: f_mono Diff)

```

```

  have f  $(\bigcap x. A x) \leq f (A 0)$ 

```

```

    using A by (auto intro!: f_mono)

```

```

  then have f_Int_fin:  $f (\bigcap x. A x) \neq \infty$ 

```

```

    using A by (auto simp: top_unique)

```

```

  have f_fin:  $f (A i - (\bigcap i. A i)) \neq \infty$  for i

```

```

  using A by (metis Diff Diff_subset f_mono infinity_ennreal_def range_subsetD top_unique)

```

```

  have  $(\lambda i. f (A i - (\bigcap i. A i))) \longrightarrow 0$ 

```

```

  proof (intro cont[ OF _ decseq _ f_fin])

```

```

    show  $\text{range } (\lambda i. A i - (\bigcap i. A i)) \subseteq M (\bigcap i. A i - (\bigcap i. A i)) = \{\}$ 

```

```

      using A by auto

```



```

qed
with INF_Lim_decseq_f have (INF n. f (A n - ( $\bigcap$  i. A i))) = 0 by metis
moreover have (INF n. f ( $\bigcap$  i. A i)) = f ( $\bigcap$  i. A i)
  by auto
ultimately have (INF n. f (A n - ( $\bigcap$  i. A i)) + f ( $\bigcap$  i. A i)) = 0 + f ( $\bigcap$  i. A
i)
  using A(4) f_fin f_Int_fin
  using INF_ennreal_add_const by presburger
moreover {
  fix n
  have f (A n - ( $\bigcap$  i. A i)) + f ( $\bigcap$  i. A i) = f ((A n - ( $\bigcap$  i. A i))  $\cup$  ( $\bigcap$  i. A i))
    using A by (subst f(2)[THEN additiveD]) auto
  also have (A n - ( $\bigcap$  i. A i))  $\cup$  ( $\bigcap$  i. A i) = A n
    by auto
  finally have f (A n - ( $\bigcap$  i. A i)) + f ( $\bigcap$  i. A i) = f (A n) . }
ultimately have (INF n. f (A n)) = f ( $\bigcap$  i. A i)
  by simp
with LIMSEQ_INF[OF decseq_fA]
show ( $\lambda$ i. f (A i))  $\longrightarrow$  f ( $\bigcap$  i. A i) by simp
qed

```

```

lemma (in ring_of_sets) empty_continuous_imp_continuous_from_below:
  fixes f :: 'a set  $\Rightarrow$  ennreal
  assumes f: positive M f additive M f  $\forall$  A  $\in$  M. f A  $\neq$   $\infty$ 
  assumes cont:  $\forall$  A. range A  $\subseteq$  M  $\longrightarrow$  decseq A  $\longrightarrow$  ( $\bigcap$  i. A i) = {}  $\longrightarrow$  ( $\lambda$ i. f
(A i))  $\longrightarrow$  0
  assumes A: range A  $\subseteq$  M incseq A ( $\bigcup$  i. A i)  $\in$  M
  shows ( $\lambda$ i. f (A i))  $\longrightarrow$  f ( $\bigcup$  i. A i)
proof -
  from A have ( $\lambda$ i. f (( $\bigcup$  i. A i) - A i))  $\longrightarrow$  0
  by (intro cont[rule_format]) (auto simp: decseq_def incseq_def)
moreover
{ fix i
  have f (( $\bigcup$  i. A i) - A i  $\cup$  A i) = f (( $\bigcup$  i. A i) - A i) + f (A i)
    using A by (intro f(2)[THEN additiveD]) auto
  also have (( $\bigcup$  i. A i) - A i)  $\cup$  A i = ( $\bigcup$  i. A i)
    by auto
  finally have f (( $\bigcup$  i. A i) - A i) = f ( $\bigcup$  i. A i) - f (A i)
    using assms f by fastforce
}
}
moreover have  $\forall$  F i in sequentially. f (A i)  $\leq$  f ( $\bigcup$  i. A i)
  using increasingD[OF additive_increasing[OF f(1, 2)], of A  $\_$   $\bigcup$  i. A i] A
  by (auto intro!: always_eventually_simp: subset_eq)
ultimately show ( $\lambda$ i. f (A i))  $\longrightarrow$  f ( $\bigcup$  i. A i)
  by (auto intro: ennreal_tendsto_const_minus)
qed

```

```

lemma (in ring_of_sets) empty_continuous_imp_countably_additive:
  fixes f :: 'a set  $\Rightarrow$  ennreal

```

assumes f : positive M f additive M f **and** fin : $\forall A \in M. f A \neq \infty$
assumes $cont$: $\bigwedge A. range A \subseteq M \implies decseq A \implies (\bigcap i. A i) = \{\} \implies (\lambda i. f (A i)) \longrightarrow 0$
shows countably_additive M f
using countably_additive_iff_continuous_from_below[OF f]
using empty_continuous_imp_continuous_from_below[OF f fin] $cont$
by blast

7.3.4 Properties of *emeasure*

lemma *emeasure_positive*: positive (sets M) (emeasure M)
by (cases M) (auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def)

lemma *emeasure_empty*[simp, intro]: emeasure M $\{\} = 0$
using *emeasure_positive*[of M] **by** (simp add: positive_def)

lemma *emeasure_single_in_space*: emeasure M $\{x\} \neq 0 \implies x \in space M$
using *emeasure_notin_sets*[of $\{x\}$ M] **by** (auto dest: sets.sets_into_space zero_less_iff_neq_zero[*TH. iffD2*])

lemma *emeasure_countably_additive*: countably_additive (sets M) (emeasure M)
by (cases M) (auto simp: sets_def emeasure_def Abs_measure_inverse measure_space_def)

lemma *suminf_emeasure*:
 $range A \subseteq sets M \implies disjoint_family A \implies (\sum i. emeasure M (A i)) = emeasure M (\bigcup i. A i)$
using sets.countable_UN[of A $UNIV$ M] *emeasure_countably_additive*[of M]
by (simp add: countably_additive_def)

lemma *sums_emeasure*:
 $disjoint_family F \implies (\bigwedge i. F i \in sets M) \implies (\lambda i. emeasure M (F i))$ sums
 $emeasure M (\bigcup i. F i)$
unfolding *sums_iff* **by** (intro conjI *suminf_emeasure*) auto

lemma *emeasure_additive*: additive (sets M) (emeasure M)
by (metis sets.countably_additive_additive *emeasure_positive* *emeasure_countably_additive*)

lemma *plus_emeasure*:
 $a \in sets M \implies b \in sets M \implies a \cap b = \{\} \implies emeasure M a + emeasure M b$
 $= emeasure M (a \cup b)$
using *additiveD*[OF *emeasure_additive*] ..

lemma *emeasure_Un*:
 $A \in sets M \implies B \in sets M \implies emeasure M (A \cup B) = emeasure M A +$
 $emeasure M (B - A)$
using *plus_emeasure*[of A M $B - A$] **by** auto

lemma *emeasure_Un_Int*:

assumes $A \in \text{sets } M$ $B \in \text{sets } M$

shows $\text{emeasure } M A + \text{emeasure } M B = \text{emeasure } M (A \cup B) + \text{emeasure } M (A \cap B)$

proof –

have $A = (A - B) \cup (A \cap B)$ **by** *auto*

then have $\text{emeasure } M A = \text{emeasure } M (A - B) + \text{emeasure } M (A \cap B)$

by (*metis Diff_Diff_Int Diff_disjoint assms plus_emeasure sets.Diff*)

moreover have $A \cup B = (A - B) \cup B$ **by** *auto*

then have $\text{emeasure } M (A \cup B) = \text{emeasure } M (A - B) + \text{emeasure } M B$

by (*metis Diff_disjoint Int_commute assms plus_emeasure sets.Diff*)

ultimately show *?thesis* **by** (*metis add.assoc add.commute*)

qed

lemma *sum_emeasure*:

$F \subseteq \text{sets } M \implies \text{disjoint_family_on } F \ I \implies \text{finite } I \implies$

$(\sum_{i \in I} \text{emeasure } M (F \ i)) = \text{emeasure } M (\bigcup_{i \in I} F \ i)$

by (*metis sets.additive_sum emeasure_positive emeasure_additive*)

lemma *emeasure_mono*:

$a \subseteq b \implies b \in \text{sets } M \implies \text{emeasure } M a \leq \text{emeasure } M b$

by (*metis zero_le sets.additive_increasing emeasure_additive emeasure_notin_sets emeasure_positive increasingD*)

lemma *emeasure_space*:

$\text{emeasure } M A \leq \text{emeasure } M (\text{space } M)$

by (*metis emeasure_mono emeasure_notin_sets sets.sets_into_space sets.top zero_le*)

lemma *emeasure_Diff*:

assumes $\text{emeasure } M B \neq \infty$

and $A \in \text{sets } M$ $B \in \text{sets } M$ **and** $B \subseteq A$

shows $\text{emeasure } M (A - B) = \text{emeasure } M A - \text{emeasure } M B$

by (*smt (verit, best) add_diff_self ennreal assms emeasure_Un emeasure_mono*

ennreal_add_left_cancel le_iff_sup)

lemma *emeasure_compl*:

$s \in \text{sets } M \implies \text{emeasure } M s \neq \infty \implies \text{emeasure } M (\text{space } M - s) = \text{emeasure } M (\text{space } M) - \text{emeasure } M s$

by (*rule emeasure_Diff*) (*auto dest: sets.sets_into_space*)

lemma *Lim_emeasure_incseq*:

$\text{range } A \subseteq \text{sets } M \implies \text{incseq } A \implies (\lambda i. (\text{emeasure } M (A \ i))) \longrightarrow \text{emeasure } M (\bigcup i. A \ i)$

using *emeasure_countably_additive*

by (*auto simp add: sets.countably_additive_iff_continuous_from_below emeasure_positive*

emeasure_additive)

lemma *incseq_emeasure*:

assumes *range* $B \subseteq \text{sets } M \text{ incseq } B$
shows $\text{incseq } (\lambda i. \text{emeasure } M (B i))$
using *assms* **by** (*auto simp: incseq_def intro!: emeasure_mono*)

lemma *SUP_emeasure_incseq*:

assumes $A: \text{range } A \subseteq \text{sets } M \text{ incseq } A$
shows $(\text{SUP } n. \text{emeasure } M (A n)) = \text{emeasure } M (\bigcup i. A i)$
using *LIMSEQ_SUP[OF incseq_emeasure, OF A] Lim_emeasure_incseq[OF A]*
by (*simp add: LIMSEQ_unique*)

lemma *decseq_emeasure*:

assumes *range* $B \subseteq \text{sets } M \text{ decseq } B$
shows $\text{decseq } (\lambda i. \text{emeasure } M (B i))$
using *assms* **by** (*auto simp: decseq_def intro!: emeasure_mono*)

lemma *INF_emeasure_decseq*:

assumes $A: \text{range } A \subseteq \text{sets } M$ **and** *decseq* A
and *finite*: $\bigwedge i. \text{emeasure } M (A i) \neq \infty$
shows $(\text{INF } n. \text{emeasure } M (A n)) = \text{emeasure } M (\bigcap i. A i)$

proof –

have *le_MI*: $\text{emeasure } M (\bigcap i. A i) \leq \text{emeasure } M (A 0)$
using A **by** (*auto intro!: emeasure_mono*)
hence $*$: $\text{emeasure } M (\bigcap i. A i) \neq \infty$ **using** *finite[of 0]* **by** (*auto simp: top_unique*)

have $\text{emeasure } M (A 0) - (\text{INF } n. \text{emeasure } M (A n)) = (\text{SUP } n. \text{emeasure } M (A 0) - \text{emeasure } M (A n))$

by (*simp add: ennreal_INF_const_minus*)

also **have** $\dots = (\text{SUP } n. \text{emeasure } M (A 0 - A n))$

using A *finite* $\langle \text{decseq } A \rangle$ [*unfolded decseq_def*] **by** (*subst emeasure_Diff*) *auto*

also **have** $\dots = \text{emeasure } M (\bigcup i. A 0 - A i)$

proof (*rule SUP_emeasure_incseq*)

show *range* $(\lambda n. A 0 - A n) \subseteq \text{sets } M$

using A **by** *auto*

show *incseq* $(\lambda n. A 0 - A n)$

using $\langle \text{decseq } A \rangle$ **by** (*auto simp add: incseq_def decseq_def*)

qed

also **have** $\dots = \text{emeasure } M (A 0) - \text{emeasure } M (\bigcap i. A i)$

using A *finite* $*$ **by** (*simp, subst emeasure_Diff*) *auto*

finally **show** *?thesis*

by (*smt (verit, best) Inf_lower diff_diff_ennreal le_MI finite range_eqI*)

qed

lemma *INF_emeasure_decseq'*:

assumes $A: \bigwedge i. A i \in \text{sets } M$ **and** *decseq* A

and *finite*: $\exists i. \text{emeasure } M (A i) \neq \infty$

shows $(\text{INF } n. \text{emeasure } M (A n)) = \text{emeasure } M (\bigcap i. A i)$

proof –

```

from finite obtain i where i: emeasure M (A i) < ∞
by (auto simp: less_top)
have fin: i ≤ j ⇒ emeasure M (A j) < ∞ for j
by (rule le_less_trans[OF emeasure_mono i]) (auto intro!: decseqD[OF <decseq
A>] A)

```

```

have (INF n. emeasure M (A n)) = (INF n. emeasure M (A (n + i)))
proof (rule INF_eq)
  show ∃j∈UNIV. emeasure M (A (j + i)) ≤ emeasure M (A i') for i'
  by (meson A <decseq A> decseq_def emeasure_mono iso_tuple_UNIV_I
nat_le_iff_add)
  qed auto
  also have ... = emeasure M (INF n. (A (n + i)))
  using A <decseq A> fin by (intro INF_emeasure_decseq) (auto simp: decseq_def
less_top)
  also have (INF n. (A (n + i))) = (INF n. A n)
  by (meson INF_eq UNIV_I assms(2) decseqD le_add1)
  finally show ?thesis .
qed

```

lemma *emeasure_INT_decseq_subset*:

```

fixes F :: nat ⇒ 'a set
assumes I: I ≠ {} and F: ∧i j. i ∈ I ⇒ j ∈ I ⇒ i ≤ j ⇒ F j ⊆ F i
assumes F_sets[measurable]: ∧i. i ∈ I ⇒ F i ∈ sets M
  and fin: ∧i. i ∈ I ⇒ emeasure M (F i) ≠ ∞
shows emeasure M (∩i∈I. F i) = (INF i∈I. emeasure M (F i))
proof cases
  assume finite I
  have (∩i∈I. F i) = F (Max I)
  using I <finite I> by (intro antisym INF_lower INF_greatest F) auto
  moreover have (INF i∈I. emeasure M (F i)) = emeasure M (F (Max I))
  using I <finite I> by (intro antisym INF_lower INF_greatest F emeasure_mono)
  auto
  ultimately show ?thesis
  by simp

```

next

```

assume infinite I
define L where L n = (LEAST i. i ∈ I ∧ i ≥ n) for n
have L: L n ∈ I ∧ n ≤ L n for n
  unfolding L_def
proof (rule LeastI_ex)
  show ∃x. x ∈ I ∧ n ≤ x
  using <infinite I> finite_subset[of I {..n}]
  by (rule_tac ccontr) (auto simp: not_le)
qed
have L_eq[simp]: i ∈ I ⇒ L i = i for i
  unfolding L_def by (intro Least_equality) auto
have L_mono: i ≤ j ⇒ L i ≤ L j for i j
  using L[of j] unfolding L_def by (intro Least_le) (auto simp: L_def)

```

```

have emeasure  $M$   $(\bigcap i. F (L i)) = (INF i. \textit{emeasure } M (F (L i)))$ 
proof (intro INF_emeasure_decseq[symmetric])
  show decseq  $(\lambda i. F (L i))$ 
    using  $L$  by (intro antimonoI F L_mono) auto
qed (insert  $L$  fin, auto)
also have  $\dots = (INF i \in I. \textit{emeasure } M (F i))$ 
proof (intro antisym INF_greatest)
  show  $i \in I \implies (INF i. \textit{emeasure } M (F (L i))) \leq \textit{emeasure } M (F i)$  for  $i$ 
    by (intro INF_lower2[of i]) auto
qed (insert  $L$ , auto intro: INF_lower)
also have  $(\bigcap i. F (L i)) = (\bigcap i \in I. F i)$ 
proof (intro antisym INF_greatest)
  show  $i \in I \implies (\bigcap i. F (L i)) \subseteq F i$  for  $i$ 
    by (intro INF_lower2[of i]) auto
qed (insert  $L$ , auto)
finally show ?thesis .
qed

```

```

lemma Lim_emeasure_decseq:
  assumes  $A$ : range  $A \subseteq \textit{sets } M$  decseq  $A$  and fin:  $\bigwedge i. \textit{emeasure } M (A i) \neq \infty$ 
  shows  $(\lambda i. \textit{emeasure } M (A i)) \longrightarrow \textit{emeasure } M (\bigcap i. A i)$ 
  using LIMSEQ_INF[OF decseq_emeasure, OF A]
  using INF_emeasure_decseq[OF A fin] by simp

```

```

lemma emeasure_lfp[consumes 1, case_names cont measurable]:
  assumes  $P M$ 
  assumes cont: sup_continuous F
  assumes  $*$ :  $\bigwedge M A. P M \implies (\bigwedge N. P N \implies \textit{Measurable.pred } N A) \implies \textit{Measurable.pred } M (F A)$ 
  shows  $\textit{emeasure } M \{x \in \textit{space } M. \textit{lfp } F x\} = (SUP i. \textit{emeasure } M \{x \in \textit{space } M. (F \hat{\sim} i) (\lambda x. \textit{False}) x\})$ 
proof -
  have  $\textit{emeasure } M \{x \in \textit{space } M. \textit{lfp } F x\} = \textit{emeasure } M (\bigcup i. \{x \in \textit{space } M. (F \hat{\sim} i) (\lambda x. \textit{False}) x\})$ 
    using sup_continuous_lfp[OF cont] by (auto simp add: bot_fun_def intro!: arg_cong2[where f=emeasure])
  moreover  $\{ \textit{fix } i \textit{ from } \langle P M \rangle \textit{ have } \{x \in \textit{space } M. (F \hat{\sim} i) (\lambda x. \textit{False}) x\} \in \textit{sets } M$ 
    by (induct i arbitrary: M) (auto simp add: pred_def[symmetric] intro: *) }
  moreover have incseq  $(\lambda i. \{x \in \textit{space } M. (F \hat{\sim} i) (\lambda x. \textit{False}) x\})$ 
proof (rule incseq_SucI)
  fix  $i$ 
  have  $(F \hat{\sim} i) (\lambda x. \textit{False}) \leq (F \hat{\sim} (\textit{Suc } i)) (\lambda x. \textit{False})$ 
proof (induct i)
  case  $0$  show ?case by (simp add: le_fun_def)
next
  case Suc thus ?case using monoD[OF sup_continuous_mono[OF cont] Suc]
by auto

```

```

qed
then show  $\{x \in \text{space } M. (F \rightsquigarrow i) (\lambda x. \text{False}) x\} \subseteq \{x \in \text{space } M. (F \rightsquigarrow \text{Suc } i) (\lambda x. \text{False}) x\}$ 
by auto
qed
ultimately show ?thesis
by (subst SUP_emeasure_incseq) auto
qed

```

lemma *emeasure_lfp*:

```

assumes [simp]:  $\bigwedge s. \text{sets } (M s) = \text{sets } N$ 
assumes cont: sup_continuous F sup_continuous f
assumes meas:  $\bigwedge P. \text{Measurable.pred } N P \implies \text{Measurable.pred } N (F P)$ 
assumes iter:  $\bigwedge P s. \text{Measurable.pred } N P \implies P \leq \text{lfp } F \implies \text{emeasure } (M s) \{x \in \text{space } N. F P x\} = f (\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. P x\}) s$ 
shows  $\text{emeasure } (M s) \{x \in \text{space } N. \text{lfp } F x\} = \text{lfp } f s$ 
proof (subst lfp_transfer_bounded[where  $\alpha = \lambda F s. \text{emeasure } (M s) \{x \in \text{space } N. F x\}$  and  $f = F$ , symmetric])
fix C assume incseq C  $\bigwedge i. \text{Measurable.pred } N (C i)$ 
then show  $(\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. (\text{SUP } i. C i) x\}) = (\text{SUP } i. (\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. C i x\}))$ 
unfolding SUP_apply
by (subst SUP_emeasure_incseq) (auto simp: mono_def fun_eq_iff intro!: arg_cong2[where  $f = \text{emeasure}$ ])
qed (auto simp add: iter le_fun_def SUP_apply intro!: meas cont)

```

lemma *emeasure_subadditive_finite*:

```

finite I  $\implies A \text{ ' } I \subseteq \text{sets } M \implies \text{emeasure } M (\bigcup i \in I. A i) \leq (\sum i \in I. \text{emeasure } M (A i))$ 
by (rule sets.subadditive[OF emeasure_positive emeasure_additive]) auto

```

lemma *emeasure_subadditive*:

```

A  $\in \text{sets } M \implies B \in \text{sets } M \implies \text{emeasure } M (A \cup B) \leq \text{emeasure } M A + \text{emeasure } M B$ 
using emeasure_subadditive_finite[of {True, False}  $\lambda \text{True} \Rightarrow A \mid \text{False} \Rightarrow B M$ ]
by simp

```

lemma *emeasure_subadditive_countably*:

```

assumes range f  $\subseteq \text{sets } M$ 
shows  $\text{emeasure } M (\bigcup i. f i) \leq (\sum i. \text{emeasure } M (f i))$ 
proof -
have  $\text{emeasure } M (\bigcup i. f i) = \text{emeasure } M (\bigcup i. \text{disjointed } f i)$ 
unfolding UN_disjointed_eq ..
also have  $\dots = (\sum i. \text{emeasure } M (\text{disjointed } f i))$ 
using sets.range_disjointed_sets[OF assms] suminf_emeasure[of disjointed f]
by (simp add: disjoint_family_disjointed_comp_def)
also have  $\dots \leq (\sum i. \text{emeasure } M (f i))$ 
using sets.range_disjointed_sets[OF assms] assms
by (auto intro!: suminf_le emeasure_mono disjointed_subset)

```

finally show ?thesis .
qed

lemma *emeasure_insert*:

assumes sets: $\{x\} \in \text{sets } M$ $A \in \text{sets } M$ and $x \notin A$

shows $\text{emeasure } M (\text{insert } x A) = \text{emeasure } M \{x\} + \text{emeasure } M A$

proof –

have $\{x\} \cap A = \{\}$ using $\langle x \notin A \rangle$ by auto

from *plus_emeasure[OF sets this]* show ?thesis by simp

qed

lemma *emeasure_insert_ne*:

$A \neq \{\} \implies \{x\} \in \text{sets } M \implies A \in \text{sets } M \implies x \notin A \implies \text{emeasure } M (\text{insert } x A) = \text{emeasure } M \{x\} + \text{emeasure } M A$

by (rule *emeasure_insert*)

lemma *emeasure_eq_sum_singleton*:

assumes finite $S \wedge x. x \in S \implies \{x\} \in \text{sets } M$

shows $\text{emeasure } M S = (\sum_{x \in S}. \text{emeasure } M \{x\})$

using *sum_emeasure[of $\lambda x. \{x\} S M$]* *assms*

by (auto simp: *disjoint_family_on_def subset_eq*)

lemma *sum_emeasure_cover*:

assumes finite S and $A \in \text{sets } M$ and *br_in_M*: $B \text{ ' } S \subseteq \text{sets } M$

assumes *A*: $A \subseteq (\bigcup_{i \in S}. B i)$

assumes *disj*: *disjoint_family_on* $B S$

shows $\text{emeasure } M A = (\sum_{i \in S}. \text{emeasure } M (A \cap (B i)))$

proof –

have $(\sum_{i \in S}. \text{emeasure } M (A \cap (B i))) = \text{emeasure } M (\bigcup_{i \in S}. A \cap (B i))$

proof (rule *sum_emeasure*)

show *disjoint_family_on* $(\lambda i. A \cap B i) S$

using $\langle \text{disjoint_family_on } B S \rangle$

unfolding *disjoint_family_on_def* by auto

qed (insert *assms*, auto)

also have $(\bigcup_{i \in S}. A \cap (B i)) = A$

using *A* by auto

finally show ?thesis by simp

qed

lemma *emeasure_eq_0*:

$N \in \text{sets } M \implies \text{emeasure } M N = 0 \implies K \subseteq N \implies \text{emeasure } M K = 0$

by (*metis emeasure_mono order_eq_iff zero_le*)

lemma *emeasure_UN_eq_0*:

assumes $\bigwedge i::\text{nat}. \text{emeasure } M (N i) = 0$ and *range N* $\subseteq \text{sets } M$

shows $\text{emeasure } M (\bigcup i. N i) = 0$

proof –

have $\text{emeasure } M (\bigcup i. N i) \leq 0$

using *emeasure_subadditive_countably[OF assms(2)]* *assms(1)* by simp


```

then show ?thesis
  by (auto intro: antisym zero_le)
qed

```

```

lemma measure_eqI_finite:
  assumes [simp]: sets M = Pow A sets N = Pow A and finite A
  assumes eq:  $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$ 
  shows M = N
proof (rule measure_eqI)
  fix X assume X  $\in$  sets M
  then have X: X  $\subseteq$  A by auto
  then have emeasure M X =  $(\sum a \in X. \text{emeasure } M \{a\})$ 
    using  $\langle$ finite A $\rangle$  by (subst emeasure_eq_sum_singleton) (auto dest: finite_subset)
  also have ... =  $(\sum a \in X. \text{emeasure } N \{a\})$ 
    using X eq by (auto intro!: sum.cong)
  also have ... = emeasure N X
    using X  $\langle$ finite A $\rangle$  by (subst emeasure_eq_sum_singleton) (auto dest: finite_subset)
  finally show emeasure M X = emeasure N X .
qed simp

```

```

lemma measure_eqI_generator_eq:
  fixes M N :: 'a measure and E :: 'a set set and A :: nat  $\implies$  'a set
  assumes Int_stable E E  $\subseteq$  Pow  $\Omega$ 
  and eq:  $\bigwedge X. X \in E \implies \text{emeasure } M X = \text{emeasure } N X$ 
  and M: sets M = sigma_sets  $\Omega$  E
  and N: sets N = sigma_sets  $\Omega$  E
  and A: range A  $\subseteq$  E  $(\bigcup i. A i) = \Omega \bigwedge i. \text{emeasure } M (A i) \neq \infty$ 
  shows M = N
proof -
  let ? $\mu$  = emeasure M and ? $\nu$  = emeasure N
  interpret S: sigma_algebra  $\Omega$  sigma_sets  $\Omega$  E by (rule sigma_algebra_sigma_sets)
  fact
  have space M =  $\Omega$ 
    using sets.top[of M] sets.space_closed[of M] S.top S.space_closed  $\langle$ sets M = sigma_sets  $\Omega$  E $\rangle$ 
    by blast

  { fix F D assume F  $\in$  E and ? $\mu$  F  $\neq$   $\infty$ 
    then have [intro]: F  $\in$  sigma_sets  $\Omega$  E by auto
    have ? $\nu$  F  $\neq$   $\infty$  using  $\langle$ ? $\mu$  F  $\neq$   $\infty$  $\rangle$   $\langle$ F  $\in$  E $\rangle$  eq by simp
    assume D  $\in$  sets M
    with  $\langle$ Int_stable E $\rangle$   $\langle$ E  $\subseteq$  Pow  $\Omega$  $\rangle$  have emeasure M (F  $\cap$  D) = emeasure N (F  $\cap$  D)
    unfolding M
    proof (induct rule: sigma_sets_induct_disjoint)
    case (basic A)
    then have F  $\cap$  A  $\in$  E using  $\langle$ Int_stable E $\rangle$   $\langle$ F  $\in$  E $\rangle$  by (auto simp: Int_stable_def)

```

```

    then show ?case using eq by auto
  next
  case empty then show ?case by simp
next
case (compl A)
then have **:  $F \cap (\Omega - A) = F - (F \cap A)$ 
  and [intro]:  $F \cap A \in \text{sigma\_sets } \Omega E$ 
  using  $\langle F \in E \rangle S.\text{sets\_into\_space}$  by (auto simp: M)
  have  $?v (F \cap A) \leq ?v F$  by (auto intro!: emeasure_mono simp: M N)
  then have  $?v (F \cap A) \neq \infty$  using  $\langle ?v F \neq \infty \rangle$  by (auto simp: top_unique)
  have  $?mu (F \cap A) \leq ?mu F$  by (auto intro!: emeasure_mono simp: M N)
  then have  $?mu (F \cap A) \neq \infty$  using  $\langle ?mu F \neq \infty \rangle$  by (auto simp: top_unique)
  then have  $?mu (F \cap (\Omega - A)) = ?mu F - ?mu (F \cap A)$  unfolding **
  using  $\langle F \cap A \in \text{sigma\_sets } \Omega E \rangle$  by (auto intro!: emeasure_Diff simp: M
N)
  also have  $\dots = ?v F - ?v (F \cap A)$  using eq  $\langle F \in E \rangle$  compl by simp
  also have  $\dots = ?v (F \cap (\Omega - A))$  unfolding **
  using  $\langle F \cap A \in \text{sigma\_sets } \Omega E \rangle \langle ?v (F \cap A) \neq \infty \rangle$ 
  by (auto intro!: emeasure_Diff[symmetric] simp: M N)
  finally show ?case
  using  $\langle \text{space } M = \Omega \rangle$  by auto
next
case (union A)
then have  $?mu (\bigcup x. F \cap A x) = ?v (\bigcup x. F \cap A x)$ 
  by (subst (1 2) suminf_emeasure[symmetric]) (auto simp: disjoint_family_on_def
subset_eq M N)
  with A show ?case
  by auto
qed }
note * = this
show  $M = N$ 
proof (rule measure_eqI)
  show sets  $M =$  sets  $N$ 
  using  $M N$  by simp
  have [simp, intro]:  $\bigwedge i. A i \in \text{sets } M$ 
  using  $A(1)$  by (auto simp: subset_eq M)
  fix  $F$  assume  $F \in \text{sets } M$ 
  let  $?D = \text{disjointed } (\lambda i. F \cap A i)$ 
  from  $\langle \text{space } M = \Omega \rangle$  have  $F\_eq: F = (\bigcup i. ?D i)$ 
  using  $\langle F \in \text{sets } M \rangle [\text{THEN } \text{sets.sets\_into\_space}] A(2)[\text{symmetric}]$  by (auto
simp: UN_disjointed_eq)
  have [simp, intro]:  $\bigwedge i. ?D i \in \text{sets } M$ 
  using sets.range_disjointed_sets[ $\text{of } \lambda i. F \cap A i M$ ]  $\langle F \in \text{sets } M \rangle$ 
  by (auto simp: subset_eq)
  have disjoint_family ?D
  by (auto simp: disjoint_family_disjointed)
moreover
have  $(\sum i. \text{emeasure } M (?D i)) = (\sum i. \text{emeasure } N (?D i))$ 
proof (intro arg_cong[where  $f = \text{suminf}$ ] ext)

```

```

fix i
have  $A \cap ?D \ i = ?D \ i$ 
  by (auto simp: disjointed_def)
then show  $\text{emeasure } M \ ( ?D \ i) = \text{emeasure } N \ ( ?D \ i)$ 
  using  $*[of \ A \ i \ ?D \ i, \ OF \ \_ \ A(\beta)] \ A(1)$  by auto
qed
ultimately show  $\text{emeasure } M \ F = \text{emeasure } N \ F$ 
by (simp add: image_subset_iff <sets M = sets N>[symmetric] F_eq[symmetric]
suminf_emeasure)
qed
qed

```

```

lemma space_empty:  $\text{space } M = \{\} \implies M = \text{count\_space } \{\}$ 
  by (rule measure_eqI) (simp_all add: space_empty_iff)

```

```

lemma measure_eqI_generator_eq_countable:

```

```

  fixes  $M \ N :: 'a \ \text{measure}$  and  $E :: 'a \ \text{set set}$  and  $A :: 'a \ \text{set set}$ 
  assumes  $E: \text{Int\_stable } E \ E \subseteq \text{Pow } \Omega \ \wedge \ X. X \in E \implies \text{emeasure } M \ X = \text{emeasure } N \ X$ 
  and sets:  $\text{sets } M = \text{sigma\_sets } \Omega \ E$   $\text{sets } N = \text{sigma\_sets } \Omega \ E$ 
  and  $A: A \subseteq E \ (\bigcup A) = \Omega \ \text{countable } A \ \wedge \ a. a \in A \implies \text{emeasure } M \ a \neq \infty$ 
  shows  $M = N$ 

```

```

proof cases

```

```

  assume  $\Omega = \{\}$ 
  have  $*$ :  $\text{sigma\_sets } \Omega \ E = \text{sets } (\text{sigma } \Omega \ E)$ 
    using  $E(2)$  by simp
  have  $\text{space } M = \Omega \ \text{space } N = \Omega$ 
    using sets  $E(2)$  unfolding  $*$  by (auto dest: sets_eq_imp_space_eq simp del: sets_measure_of)
  then show  $M = N$ 
    unfolding  $\langle \Omega = \{\} \rangle$  by (auto dest: space_empty)

```

```

next

```

```

  assume  $\Omega \neq \{\}$  with  $\langle \bigcup A = \Omega \rangle$  have  $A \neq \{\}$  by auto
  from this  $\langle \text{countable } A \rangle$  have  $\text{rng: range } (\text{from\_nat\_into } A) = A$ 
    by (rule range_from_nat_into)
  show  $M = N$ 
  proof (rule measure_eqI_generator_eq[OF E sets])
    show  $\text{range } (\text{from\_nat\_into } A) \subseteq E$ 
      unfolding rng using  $\langle A \subseteq E \rangle$  .
    show  $(\bigcup i. \text{from\_nat\_into } A \ i) = \Omega$ 
      unfolding rng using  $\langle \bigcup A = \Omega \rangle$  .
    show  $\text{emeasure } M \ (\text{from\_nat\_into } A \ i) \neq \infty$  for  $i$ 
      using rng by (intro A) auto

```

```

qed

```

```

qed

```

```

lemma measure_of_of_measure:  $\text{measure\_of } (\text{space } M) \ (\text{sets } M) \ (\text{emeasure } M) = M$ 

```

```

proof (intro measure_eqI emeasure_measure_of_sigma)

```

```

show sigma_algebra (space M) (sets M) ..
show positive (sets M) (emeasure M)
  by (simp add: positive_def)
show countably_additive (sets M) (emeasure M)
  by (simp add: emeasure_countably_additive)
qed simp_all

```

7.3.5 μ -null sets

definition *null_sets* :: 'a measure \Rightarrow 'a set set **where**
null_sets M = {N \in sets M. emeasure M N = 0}

lemma *null_setsD1*[dest]: A \in *null_sets* M \implies emeasure M A = 0
 by (simp add: *null_sets_def*)

lemma *null_setsD2*[dest]: A \in *null_sets* M \implies A \in sets M
 unfolding *null_sets_def* by simp

lemma *null_setsI*[intro]: emeasure M A = 0 \implies A \in sets M \implies A \in *null_sets* M
 unfolding *null_sets_def* by simp

interpretation *null_sets*: ring_of_sets space M *null_sets* M **for** M

proof (rule ring_of_setsI)

show *null_sets* M \subseteq Pow (space M)

using sets.sets_into_space by auto

show {} \in *null_sets* M

by auto

fix A B **assume** *null_sets*: A \in *null_sets* M B \in *null_sets* M

then have sets: A \in sets M B \in sets M

by auto

then have *: emeasure M (A \cup B) \leq emeasure M A + emeasure M B

emeasure M (A - B) \leq emeasure M A

by (auto intro!: emeasure_subadditive emeasure_mono)

then have emeasure M B = 0 emeasure M A = 0

using *null_sets* by auto

with sets * **show** A - B \in *null_sets* M A \cup B \in *null_sets* M

by (auto intro!: antisym zero_le)

qed

lemma *UN_from_nat_into*:

assumes I: countable I I \neq {}

shows ($\bigcup_{i \in I} N i$) = ($\bigcup_{i \in I} N (from_nat_into I i)$)

proof -

have ($\bigcup_{i \in I} N i$) = $\bigcup (N \text{ ` } range (from_nat_into I))$

using I by simp

also have ... = ($\bigcup_{i \in I} (N \circ from_nat_into I) i$)

by simp

finally show ?thesis by simp

qed

lemma null_sets_UN':

assumes countable I

assumes $\bigwedge i. i \in I \implies N i \in \text{null_sets } M$

shows $(\bigcup_{i \in I}. N i) \in \text{null_sets } M$

proof cases

assume $I = \{\}$ then show ?thesis by simp

next

assume $I \neq \{\}$

show ?thesis

proof (intro conjI CollectI null_setsI)

show $(\bigcup_{i \in I}. N i) \in \text{sets } M$

using assms by (intro sets.countable_UN') auto

have $\text{emeasure } M (\bigcup_{i \in I}. N i) \leq (\sum n. \text{emeasure } M (N (\text{from_nat_into } I n)))$

unfolding UN_from_nat_into[OF <countable I> <I ≠ {}>]

using assms <I ≠ {}> by (intro emeasure_subadditive_countably) (auto intro: from_nat_into)

also have $(\lambda n. \text{emeasure } M (N (\text{from_nat_into } I n))) = (\lambda_. 0)$

using assms <I ≠ {}> by (auto intro: from_nat_into)

finally show $\text{emeasure } M (\bigcup_{i \in I}. N i) = 0$

by (intro antisym zero_le) simp

qed

qed

lemma null_sets_UN[intro]:

$(\bigwedge i::i::\text{countable}. N i \in \text{null_sets } M) \implies (\bigcup i. N i) \in \text{null_sets } M$

by (rule null_sets_UN') auto

lemma null_set_Int1:

assumes $B \in \text{null_sets } M$ $A \in \text{sets } M$ shows $A \cap B \in \text{null_sets } M$

proof (intro CollectI conjI null_setsI)

show $\text{emeasure } M (A \cap B) = 0$ using assms

by (intro emeasure_eq_0[of B _ A ∩ B]) auto

qed (insert assms, auto)

lemma null_set_Int2:

assumes $B \in \text{null_sets } M$ $A \in \text{sets } M$ shows $B \cap A \in \text{null_sets } M$

using assms by (subst Int_commute) (rule null_set_Int1)

lemma emeasure_Diff_null_set:

assumes $B \in \text{null_sets } M$ $A \in \text{sets } M$

shows $\text{emeasure } M (A - B) = \text{emeasure } M A$

proof -

have *: $A - B = (A - (A \cap B))$ by auto

have $A \cap B \in \text{null_sets } M$ using assms by (rule null_set_Int1)

then show ?thesis

unfolding * using assms

by (subst emeasure_Diff) auto

2114

qed

lemma *null_set_Diff*:

assumes $B \in \text{null_sets } M$ $A \in \text{sets } M$ **shows** $B - A \in \text{null_sets } M$

proof (intro CollectI conjI null_setsI)

show $\text{emeasure } M (B - A) = 0$ **using** *assms* **by** (intro *emeasure_eq_0*[of $B - A$]) *auto*

qed (*insert assms, auto*)

lemma *emeasure_Un_null_set*:

assumes $A \in \text{sets } M$ $B \in \text{null_sets } M$

shows $\text{emeasure } M (A \cup B) = \text{emeasure } M A$

proof –

have $*$: $A \cup B = A \cup (B - A)$ **by** *auto*

have $B - A \in \text{null_sets } M$ **using** *assms*(2,1) **by** (rule *null_set_Diff*)

then show *?thesis*

unfolding $*$ **using** *assms*

by (*subst plus_emeasure[symmetric]*) *auto*

qed

lemma *emeasure_Un'*:

assumes $A \in \text{sets } M$ $B \in \text{sets } M$ $A \cap B \in \text{null_sets } M$

shows $\text{emeasure } M (A \cup B) = \text{emeasure } M A + \text{emeasure } M B$

proof –

have $A \cup B = A \cup (B - A \cap B)$ **by** *blast*

also have $\text{emeasure } M \dots = \text{emeasure } M A + \text{emeasure } M (B - A \cap B)$

using *assms* **by** (*subst plus_emeasure*) *auto*

also have $\text{emeasure } M (B - A \cap B) = \text{emeasure } M B$

using *assms* **by** (intro *emeasure_Diff_null_set*) *auto*

finally show *?thesis* .

qed

7.3.6 The almost everywhere filter (i.e. quantifier)

definition *ae_filter* :: 'a measure \Rightarrow 'a filter **where**

$\text{ae_filter } M = (\text{INF } N \in \text{null_sets } M. \text{principal } (\text{space } M - N))$

abbreviation *almost_everywhere* :: 'a measure \Rightarrow ('a \Rightarrow bool) \Rightarrow bool **where**

$\text{almost_everywhere } M P \equiv \text{eventually } P (\text{ae_filter } M)$

syntax

$_ \text{almost_everywhere} :: \text{pttrn} \Rightarrow 'a \Rightarrow \text{bool} \Rightarrow \text{bool} (\text{AE } _ \text{ in } _ _ [0,0,10] 10)$

translations

$\text{AE } x \text{ in } M. P \equiv \text{CONST } \text{almost_everywhere } M (\lambda x. P)$

abbreviation

$\text{set_almost_everywhere } A M P \equiv \text{AE } x \text{ in } M. x \in A \longrightarrow P x$

syntax

`_set_almost_everywhere` :: *pttrn* \Rightarrow '*a* *set* \Rightarrow '*a* \Rightarrow *bool* \Rightarrow *bool*
 $(AE _ \in _ \text{ in } _ / _ [0,0,0,10] 10)$

translations

$AE \ x \in A \ \text{in } M. \ P \Leftrightarrow \text{CONST } \text{set_almost_everywhere } A \ M \ (\lambda x. \ P)$

lemma *eventually_ae_filter*: $\text{eventually } P \ (\text{ae_filter } M) \longleftrightarrow (\exists N \in \text{null_sets } M. \{x \in \text{space } M. \neg P \ x\} \subseteq N)$

unfolding *ae_filter_def* **by** (*subst eventually_INF_base*) (*auto simp: eventually_principal subset_eq*)

lemma *AE_I'*:

$N \in \text{null_sets } M \Longrightarrow \{x \in \text{space } M. \neg P \ x\} \subseteq N \Longrightarrow (AE \ x \ \text{in } M. \ P \ x)$

unfolding *eventually_ae_filter* **by** *auto*

lemma *AE_iff_null*:

assumes $\{x \in \text{space } M. \neg P \ x\} \in \text{sets } M$ (**is** $?P \in \text{sets } M$)

shows $(AE \ x \ \text{in } M. \ P \ x) \longleftrightarrow \{x \in \text{space } M. \neg P \ x\} \in \text{null_sets } M$

proof

assume $AE \ x \ \text{in } M. \ P \ x$ **then obtain** *N* **where** $N: N \in \text{sets } M \ ?P \subseteq N \ \text{emeasure } M \ N = 0$

unfolding *eventually_ae_filter* **by** *auto*

have $\text{emeasure } M \ ?P \leq \text{emeasure } M \ N$

using *assms N(1,2)* **by** (*auto intro: emeasure_mono*)

then have $\text{emeasure } M \ ?P = 0$

unfolding $\langle \text{emeasure } M \ N = 0 \rangle$ **by** *auto*

then show $?P \in \text{null_sets } M$ **using** *assms* **by** *auto*

next

assume $?P \in \text{null_sets } M$ **with** *assms* **show** $AE \ x \ \text{in } M. \ P \ x$ **by** (*auto intro: AE_I'*)

qed

lemma *AE_iff_null_sets*:

$N \in \text{sets } M \Longrightarrow N \in \text{null_sets } M \longleftrightarrow (AE \ x \ \text{in } M. \ x \notin N)$

using *Int_absorb1[OF sets.sets_into_space, of N M]*

by (*subst AE_iff_null*) (*auto simp: Int_def[symmetric]*)

lemma *ae_filter_eq_bot_iff*: $\text{ae_filter } M = \text{bot} \longleftrightarrow \text{emeasure } M \ (\text{space } M) = 0$

proof –

have $\text{ae_filter } M = \text{bot} \longleftrightarrow (AE \ x \ \text{in } M. \ \text{False})$

using *trivial_limit_def* **by** *blast*

also have $\dots \longleftrightarrow \text{space } M \in \text{null_sets } M$

by (*simp add: AE_iff_null_sets eventually_ae_filter*)

also have $\dots \longleftrightarrow \text{emeasure } M \ (\text{space } M) = 0$

by *auto*

finally show *?thesis* .

qed

lemma *AE_not_in*:

$N \in \text{null_sets } M \implies AE\ x\ \text{in } M. x \notin N$
by (*metis AE_iff_null_sets null_setsD2*)

lemma *AE_iff_measurable*:

$N \in \text{sets } M \implies \{x \in \text{space } M. \neg P\ x\} = N \implies (AE\ x\ \text{in } M. P\ x) \longleftrightarrow \text{emeasure } M\ N = 0$
using *AE_iff_null[of _ P]* **by** *auto*

lemma *AE_E[consumes 1]*:

assumes $AE\ x\ \text{in } M. P\ x$
obtains N **where** $\{x \in \text{space } M. \neg P\ x\} \subseteq N$ $\text{emeasure } M\ N = 0$ $N \in \text{sets } M$
using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_E2*:

assumes $AE\ x\ \text{in } M. P\ x$
shows $\text{emeasure } M\ \{x \in \text{space } M. \neg P\ x\} = 0$
by (*metis (mono_tags, lifting) AE_iff_null assms emeasure_notin_sets null_setsD1*)

lemma *AE_E3*:

assumes $AE\ x\ \text{in } M. P\ x$
obtains N **where** $\bigwedge x. x \in \text{space } M - N \implies P\ x$ $N \in \text{null_sets } M$
using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_I*:

assumes $\{x \in \text{space } M. \neg P\ x\} \subseteq N$ $\text{emeasure } M\ N = 0$ $N \in \text{sets } M$
shows $AE\ x\ \text{in } M. P\ x$
using *assms unfolding eventually_ae_filter* **by** *auto*

lemma *AE_mp[elim!]*:

assumes $AE_P: AE\ x\ \text{in } M. P\ x$ **and** $AE_imp: AE\ x\ \text{in } M. P\ x \longrightarrow Q\ x$
shows $AE\ x\ \text{in } M. Q\ x$
using *assms* **by** (*fact eventually_rev_mp*)

The next lemma is convenient to combine with a lemma whose conclusion is of the form $AE\ x\ \text{in } M. P\ x = Q\ x$: for such a lemma, there is no *[symmetric]* variant, but using *AE_symmetric[OF...]* will replace it.

lemma

shows $AE_iffI: AE\ x\ \text{in } M. P\ x \implies AE\ x\ \text{in } M. P\ x \longleftrightarrow Q\ x \implies AE\ x\ \text{in } M. Q\ x$
and $AE_disjI1: AE\ x\ \text{in } M. P\ x \implies AE\ x\ \text{in } M. P\ x \vee Q\ x$
and $AE_disjI2: AE\ x\ \text{in } M. Q\ x \implies AE\ x\ \text{in } M. P\ x \vee Q\ x$
and $AE_conjI: AE\ x\ \text{in } M. P\ x \implies AE\ x\ \text{in } M. Q\ x \implies AE\ x\ \text{in } M. P\ x \wedge Q\ x$
and $AE_conj_iff[simp]: (AE\ x\ \text{in } M. P\ x \wedge Q\ x) \longleftrightarrow (AE\ x\ \text{in } M. P\ x) \wedge (AE\ x\ \text{in } M. Q\ x)$
by *auto*

lemma *AE_symmetric*:

assumes $AE\ x\ in\ M.\ P\ x = Q\ x$
shows $AE\ x\ in\ M.\ Q\ x = P\ x$
using *assms* **by** *auto*

lemma *AE_impI*:
 $(P \implies AE\ x\ in\ M.\ Q\ x) \implies AE\ x\ in\ M.\ P \longrightarrow Q\ x$
by *fastforce*

lemma *AE_measure*:
assumes AE : $AE\ x\ in\ M.\ P\ x$ **and** *sets*: $\{x \in space\ M.\ P\ x\} \in sets\ M$ (**is** $?P \in sets\ M$)
shows $emeasure\ M\ \{x \in space\ M.\ P\ x\} = emeasure\ M\ (space\ M)$
proof –
from *AE_E[OF AE]* **obtain** N
where N : $\{x \in space\ M.\ \neg P\ x\} \subseteq N$ $emeasure\ M\ N = 0$ $N \in sets\ M$
by *auto*
with *sets* **have** $emeasure\ M\ (space\ M) \leq emeasure\ M\ (?P \cup N)$
by (*intro emeasure_mono*) *auto*
also **have** $\dots \leq emeasure\ M\ ?P + emeasure\ M\ N$
using *sets N* **by** (*intro emeasure_subadditive*) *auto*
also **have** $\dots = emeasure\ M\ ?P$ **using** N **by** *simp*
finally **show** $emeasure\ M\ ?P = emeasure\ M\ (space\ M)$
using *emeasure_space[of M ?P]* **by** *auto*
qed

lemma *AE_space*: $AE\ x\ in\ M.\ x \in space\ M$
by (*rule AE_I[where N={}]*) *auto*

lemma *AE_I2[simp, intro]*:
 $(\bigwedge x.\ x \in space\ M \implies P\ x) \implies AE\ x\ in\ M.\ P\ x$
using *AE_space* **by** *force*

lemma *AE_Ball_mp*:
 $\forall x \in space\ M.\ P\ x \implies AE\ x\ in\ M.\ P\ x \longrightarrow Q\ x \implies AE\ x\ in\ M.\ Q\ x$
by *auto*

lemma *AE_cong[cong]*:
 $(\bigwedge x.\ x \in space\ M \implies P\ x \longleftrightarrow Q\ x) \implies (AE\ x\ in\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ M.\ Q\ x)$
by *auto*

lemma *AE_cong_simp*: $M = N \implies (\bigwedge x.\ x \in space\ N = simp \implies P\ x = Q\ x) \implies (AE\ x\ in\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ N.\ Q\ x)$
by (*auto simp: simp_implies_def*)

lemma *AE_all_countable*:
 $(AE\ x\ in\ M.\ \forall i.\ P\ i\ x) \longleftrightarrow (\forall i::'i::countable.\ AE\ x\ in\ M.\ P\ i\ x)$
proof
assume $\forall i.\ AE\ x\ in\ M.\ P\ i\ x$

from *this*[*unfolded eventually_ae_filter Bex_def, THEN choice*]
obtain N **where** $N: \bigwedge i. N i \in \text{null_sets } M \bigwedge i. \{x \in \text{space } M. \neg P i x\} \subseteq N i$
by *auto*
have $\{x \in \text{space } M. \neg (\forall i. P i x)\} \subseteq (\bigcup i. \{x \in \text{space } M. \neg P i x\})$ **by** *auto*
also have $\dots \subseteq (\bigcup i. N i)$ **using** N **by** *auto*
finally have $\{x \in \text{space } M. \neg (\forall i. P i x)\} \subseteq (\bigcup i. N i)$.
moreover from N **have** $(\bigcup i. N i) \in \text{null_sets } M$
by (*intro null_sets_UN*) *auto*
ultimately show $AE x \text{ in } M. \forall i. P i x$
unfolding *eventually_ae_filter* **by** *auto*
qed *auto*

lemma *AE_ball_countable*:

assumes [*intro*]: *countable* X

shows $(AE x \text{ in } M. \forall y \in X. P x y) \longleftrightarrow (\forall y \in X. AE x \text{ in } M. P x y)$

proof

assume $\forall y \in X. AE x \text{ in } M. P x y$

from *this*[*unfolded eventually_ae_filter Bex_def, THEN bchoice*]

obtain N **where** $N: \bigwedge y. y \in X \implies N y \in \text{null_sets } M \bigwedge y. y \in X \implies \{x \in \text{space } M. \neg P x y\} \subseteq N y$

by *auto*

have $\{x \in \text{space } M. \neg (\forall y \in X. P x y)\} \subseteq (\bigcup y \in X. \{x \in \text{space } M. \neg P x y\})$

by *auto*

also have $\dots \subseteq (\bigcup y \in X. N y)$

using N **by** *auto*

finally have $\{x \in \text{space } M. \neg (\forall y \in X. P x y)\} \subseteq (\bigcup y \in X. N y)$.

moreover from N **have** $(\bigcup y \in X. N y) \in \text{null_sets } M$

by (*intro null_sets_UN'*) *auto*

ultimately show $AE x \text{ in } M. \forall y \in X. P x y$

unfolding *eventually_ae_filter* **by** *auto*

qed *auto*

lemma *AE_ball_countable'*:

$(\bigwedge N. N \in I \implies AE x \text{ in } M. P N x) \implies \text{countable } I \implies AE x \text{ in } M. \forall N \in I. P N x$

unfolding *AE_ball_countable* **by** *simp*

lemma *AE_pairwise*: *countable* $F \implies \text{pairwise } (\lambda A B. AE x \text{ in } M. R x A B) F$
 $\longleftrightarrow (AE x \text{ in } M. \text{pairwise } (R x) F)$

unfolding *pairwise_alt* **by** (*simp add: AE_ball_countable*)

lemma *AE_discrete_difference*:

assumes X : *countable* X

assumes *null*: $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$

assumes *sets*: $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$

shows $AE x \text{ in } M. x \notin X$

proof –

have $(\bigcup x \in X. \{x\}) \in \text{null_sets } M$

using *assms* **by** (*intro null_sets_UN'*) *auto*

from AE_not_in [OF *this*] **show** $AE\ x\ in\ M.\ x \notin X$
by *auto*
qed

lemma AE_finite_all :
assumes f : *finite* S **shows** $(AE\ x\ in\ M.\ \forall i \in S.\ P\ i\ x) \longleftrightarrow (\forall i \in S.\ AE\ x\ in\ M.\ P\ i\ x)$
using f **by** *induct auto*

lemma AE_finite_allI :
assumes *finite* S
shows $(\bigwedge s.\ s \in S \implies AE\ x\ in\ M.\ Q\ s\ x) \implies AE\ x\ in\ M.\ \forall s \in S.\ Q\ s\ x$
using AE_finite_all [OF $\langle finite\ S \rangle$] **by** *auto*

lemma $emeasure_mono_AE$:
assumes imp : $AE\ x\ in\ M.\ x \in A \longrightarrow x \in B$
and B : $B \in sets\ M$
shows $emeasure\ M\ A \leq emeasure\ M\ B$
proof *cases*
assume A : $A \in sets\ M$
from imp **obtain** N **where** N : $\{x \in space\ M.\ \neg (x \in A \longrightarrow x \in B)\} \subseteq N$ $N \in null_sets\ M$
by (*auto simp: eventually_ae_filter*)
have $emeasure\ M\ A = emeasure\ M\ (A - N)$
using $N\ A$ **by** (*subst emeasure_Diff_null_set auto*)
also **have** $emeasure\ M\ (A - N) \leq emeasure\ M\ (B - N)$
using $N\ A\ B\ sets.sets_into_space$ **by** (*auto intro!: emeasure_mono*)
also **have** $emeasure\ M\ (B - N) = emeasure\ M\ B$
using $N\ B$ **by** (*subst emeasure_Diff_null_set auto*)
finally **show** *?thesis* .
qed (*simp add: emeasure_notin_sets*)

lemma $emeasure_eq_AE$:
assumes iff : $AE\ x\ in\ M.\ x \in A \longleftrightarrow x \in B$
assumes A : $A \in sets\ M$ **and** B : $B \in sets\ M$
shows $emeasure\ M\ A = emeasure\ M\ B$
using $assms$ **by** (*safe intro!: antisym emeasure_mono_AE auto*)

lemma $emeasure_Collect_eq_AE$:
 $AE\ x\ in\ M.\ P\ x \longleftrightarrow Q\ x \implies Measurable.pred\ M\ Q \implies Measurable.pred\ M\ P$
 \implies
 $emeasure\ M\ \{x \in space\ M.\ P\ x\} = emeasure\ M\ \{x \in space\ M.\ Q\ x\}$
by (*intro emeasure_eq_AE auto*)

lemma $emeasure_eq_0_AE$: $AE\ x\ in\ M.\ \neg P\ x \implies emeasure\ M\ \{x \in space\ M.\ P\ x\} = 0$
using $AE_iff_measurable$ [OF $_ refl$, of $M\ \lambda x.\ \neg P\ x$]
by (*cases* $\{x \in space\ M.\ P\ x\} \in sets\ M$) (*simp_all add: emeasure_notin_sets*)

lemma *emeasure_0_AE*:
assumes *emeasure M (space M) = 0*
shows *AE x in M. P x*
using *eventually_ae_filter assms by blast*

lemma *emeasure_add_AE*:
assumes [*measurable*]: *A ∈ sets M B ∈ sets M C ∈ sets M*
assumes 1: *AE x in M. x ∈ C ↔ x ∈ A ∨ x ∈ B*
assumes 2: *AE x in M. ¬ (x ∈ A ∧ x ∈ B)*
shows *emeasure M C = emeasure M A + emeasure M B*
proof –
have *emeasure M C = emeasure M (A ∪ B)*
by (*rule emeasure_eq_AE*) (*insert 1, auto*)
also have *... = emeasure M A + emeasure M (B - A)*
by (*subst plus_emeasure*) *auto*
also have *emeasure M (B - A) = emeasure M B*
by (*rule emeasure_eq_AE*) (*insert 2, auto*)
finally show *?thesis .*
qed

7.3.7 σ -finite Measures

locale *sigma_finite_measure* =
fixes *M :: 'a measure*
assumes *sigma_finite_countable*:
 $\exists A :: 'a \text{ set set. countable } A \wedge A \subseteq \text{sets } M \wedge (\bigcup A) = \text{space } M \wedge (\forall a \in A. \text{emeasure } M a \neq \infty)$

lemma (*in sigma_finite_measure*) *sigma_finite*:
obtains *A :: nat ⇒ 'a set*
where *range A ⊆ sets M (∪ i. A i) = space M ∧ i. emeasure M (A i) ≠ ∞*
proof –
obtain *A :: 'a set set where*
 $[simp]: \text{countable } A$ **and**
 $A: A \subseteq \text{sets } M (\bigcup A) = \text{space } M \wedge a. a \in A \implies \text{emeasure } M a \neq \infty$
using *sigma_finite_countable by metis*
show *thesis*
proof *cases*
assume *A = {} with ⟨(∪ A) = space M⟩ show* *thesis*
by (*intro that[of λ_. {}]*) *auto*
next
assume *A ≠ {}*
show *thesis*
proof
show *range (from_nat_into A) ⊆ sets M*
using $\langle A \neq \{\} \rangle A$ **by** *auto*
have $(\bigcup i. \text{from_nat_into } A \ i) = \bigcup A$
using *range_from_nat_into[OF ⟨A ≠ {}⟩ ⟨countable A⟩]* **by** *auto*
with *A show* $(\bigcup i. \text{from_nat_into } A \ i) = \text{space } M$

```

    by auto
  qed (intro A from_nat_into ⟨A ≠ {}⟩)
qed
qed

```

lemma (in *sigma_finite_measure*) *sigma_finite_disjoint*:

```

obtains A :: nat ⇒ 'a set
where range A ⊆ sets M (⋃ i. A i) = space M ∧ i. emeasure M (A i) ≠ ∞
disjoint_family A
proof -
obtain A :: nat ⇒ 'a set where
  range: range A ⊆ sets M and
  space: (⋃ i. A i) = space M and
  measure: ∧ i. emeasure M (A i) ≠ ∞
using sigma_finite by blast
show thesis
proof (rule that[of disjointed A])
show range (disjointed A) ⊆ sets M
  by (rule sets.range_disjointed_sets[OF range])
show (⋃ i. disjointed A i) = space M
  and disjoint_family (disjointed A)
  using disjoint_family_disjointed UN_disjointed_eq[of A] space range
  by auto
show emeasure M (disjointed A i) ≠ ∞ for i
proof -
  have emeasure M (disjointed A i) ≤ emeasure M (A i)
    using range_disjointed_subset[of A i] by (auto intro!: emeasure_mono)
  then show ?thesis using measure[of i] by (auto simp: top_unique)
qed
qed
qed

```

lemma (in *sigma_finite_measure*) *sigma_finite_incseq*:

```

obtains A :: nat ⇒ 'a set
where range A ⊆ sets M (⋃ i. A i) = space M ∧ i. emeasure M (A i) ≠ ∞
incseq A
proof -
obtain F :: nat ⇒ 'a set where
  F: range F ⊆ sets M (⋃ i. F i) = space M ∧ i. emeasure M (F i) ≠ ∞
using sigma_finite by blast
show thesis
proof (rule that[of λn. ⋃ i≤n. F i])
show range (λn. ⋃ i≤n. F i) ⊆ sets M
  using F by (force simp: incseq_def)
show (⋃ n. ⋃ i≤n. F i) = space M
proof -
from F have ∧ x. x ∈ space M ⇒ ∃ i. x ∈ F i by auto
with F show ?thesis by fastforce
qed

```

```

show emeasure M ( $\bigcup_{i \leq n}. F i$ )  $\neq \infty$  for n
proof -
  have emeasure M ( $\bigcup_{i \leq n}. F i$ )  $\leq (\sum_{i \leq n}. \text{emeasure } M (F i))$ 
    using F by (auto intro!: emeasure_subadditive_finite)
  also have ...  $< \infty$ 
    using F by (auto simp: sum_Pinfty less_top)
  finally show ?thesis by simp
qed
show incseq ( $\lambda n. \bigcup_{i \leq n}. F i$ )
  by (force simp: incseq_def)
qed
qed

lemma (in sigma_finite_measure) approx_PInf_emeasure_with_finite:
  fixes C::real
  assumes W_meas: W  $\in$  sets M
    and W_inf: emeasure M W =  $\infty$ 
  obtains Z where Z  $\in$  sets M Z  $\subseteq$  W emeasure M Z  $< \infty$  emeasure M Z  $> C$ 
proof -
  obtain A :: nat  $\Rightarrow$  'a set
    where A: range A  $\subseteq$  sets M ( $\bigcup i. A i$ ) = space M  $\wedge i. \text{emeasure } M (A i) \neq \infty$ 
    incseq A
    using sigma_finite_incseq by blast
  define B where B = ( $\lambda i. W \cap A i$ )
  have B_meas:  $\wedge i. B i \in$  sets M using W_meas  $\langle$ range A  $\subseteq$  sets M $\rangle$  B_def by
  blast
  have b:  $\wedge i. B i \subseteq$  W using B_def by blast

  { fix i
    have emeasure M (B i)  $\leq$  emeasure M (A i)
      using A by (intro emeasure_mono) (auto simp: B_def)
    also have emeasure M (A i)  $< \infty$ 
      using  $\langle \wedge i. \text{emeasure } M (A i) \neq \infty \rangle$  by (simp add: less_top)
    finally have emeasure M (B i)  $< \infty$  . }
  note c = this

  have W = ( $\bigcup i. B i$ ) using B_def  $\langle$  $\bigcup i. A i$  $\rangle$  W_meas by auto
  moreover have incseq B using B_def  $\langle$ incseq A $\rangle$  by (simp add: incseq_def
  subset_eq)
  ultimately have ( $\lambda i. \text{emeasure } M (B i)$ )  $\longrightarrow$  emeasure M W using W_meas
  B_meas
    by (simp add: B_meas Lim_emeasure_incseq_image_subset_iff)
  then have ( $\lambda i. \text{emeasure } M (B i)$ )  $\longrightarrow \infty$  using W_inf by simp
  from order_tendstoD(1)[OF this, of C]
  obtain i where d: emeasure M (B i)  $> C$ 
    by (auto simp: eventually_sequentially)
  have B i  $\in$  sets M B i  $\subseteq$  W emeasure M (B i)  $< \infty$  emeasure M (B i)  $> C$ 
    using B_meas b c d by auto
  then show ?thesis using that by blast

```

qed

7.3.8 Measure space induced by distribution of (\rightarrow_M) -functions

definition *distr* :: 'a measure \Rightarrow 'b measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b measure **where**
distr M N f =
 measure_of (space N) (sets N) ($\lambda A. \text{emeasure } M (f - ' A \cap \text{space } M)$)

lemma

shows *sets_distr*[simp, measurable_cong]: sets (distr M N f) = sets N
and *space_distr*[simp]: space (distr M N f) = space N
by (auto simp: distr_def)

lemma

shows *measurable_distr_eq1*[simp]: measurable (distr Mf Nf f) Mf' = measurable Nf Mf'
and *measurable_distr_eq2*[simp]: measurable Mg' (distr Mg Ng g) = measurable Mg' Ng
by (auto simp: measurable_def)

lemma *distr_cong*:

$M = K \Longrightarrow \text{sets } N = \text{sets } L \Longrightarrow (\bigwedge x. x \in \text{space } M \Longrightarrow f x = g x) \Longrightarrow \text{distr } M N f = \text{distr } K L g$
using *sets_eq_imp_space_eq*[of N L] **by** (simp add: distr_def Int_def cong: rev_conj_cong)

lemma *emeasure_distr*:

fixes f :: 'a \Rightarrow 'b
assumes f: f \in measurable M N **and** A: A \in sets N
shows *emeasure* (distr M N f) A = *emeasure* M (f - ' A \cap space M) (**is** _ = ? μ A)

unfolding *distr_def*

proof (rule *emeasure_measure_of_sigma*)

show *positive* (sets N) ? μ
by (auto simp: positive_def)

show *countably_additive* (sets N) ? μ

proof (intro *countably_additiveI*)

fix A :: nat \Rightarrow 'b **set** **assume** range A \subseteq sets N *disjoint_family* A

then have A: $\bigwedge i. A i \in \text{sets } N (\bigcup i. A i) \in \text{sets } N$ **by** auto

then have *: range ($\lambda i. f - ' (A i) \cap \text{space } M$) \subseteq sets M

using f **by** (auto simp: measurable_def)

moreover have ($\bigcup i. f - ' A i \cap \text{space } M$) \in sets M

using * **by** blast

moreover have **: *disjoint_family* ($\lambda i. f - ' A i \cap \text{space } M$)

using <*disjoint_family* A> **by** (auto simp: *disjoint_family_on_def*)

ultimately show ($\sum i. ?\mu (A i)$) = ? μ ($\bigcup i. A i$)

using *suminf_emeasure[OF _ **]* A f

by (auto simp: *comp_def vimage_UN*)

qed
show *sigma_algebra* (*space N*) (*sets N*) ..
qed fact

lemma *emeasure_Collect_distr*:
assumes *X*[*measurable*]: $X \in \text{measurable } M \ N \ \text{Measurable.pred } N \ P$
shows $\text{emeasure } (\text{distr } M \ N \ X) \ \{x \in \text{space } N. \ P \ x\} = \text{emeasure } M \ \{x \in \text{space } M. \ P \ (X \ x)\}$
by (*subst* *emeasure_distr*)
(auto intro!: arg_cong2[where f=emeasure] X(1)[THEN measurable_space])

lemma *emeasure_lfp2*[*consumes 1, case_names cont f measurable*]:
assumes *P M*
assumes *cont*: *sup_continuous F*
assumes *f*: $\bigwedge M. \ P \ M \ \Longrightarrow \ f \in \text{measurable } M' \ M$
assumes ***: $\bigwedge M \ A. \ P \ M \ \Longrightarrow \ (\bigwedge N. \ P \ N \ \Longrightarrow \ \text{Measurable.pred } N \ A) \ \Longrightarrow \ \text{Measurable.pred } M \ (F \ A)$
shows $\text{emeasure } M' \ \{x \in \text{space } M'. \ \text{lfp } F \ (f \ x)\} = (\text{SUP } i. \ \text{emeasure } M' \ \{x \in \text{space } M'. \ (F \ \sim i) \ (\lambda x. \ \text{False}) \ (f \ x)\})$
proof (*subst* (1 2) *emeasure_Collect_distr*[*symmetric, where X=f*])
show $f \in \text{measurable } M' \ M \ f \in \text{measurable } M' \ M$
using *f[OF <P M>]* **by** *auto*
{ fix i show *Measurable.pred M ((F ~ i) (λx. False))*
using *<P M>* **by** (*induction i arbitrary: M*) (*auto intro!: **) }
show *Measurable.pred M (lfp F)*
using *<P M>* *cont ** **by** (*rule measurable_lfp_coinduct[of P]*)

have $\text{emeasure } (\text{distr } M' \ M \ f) \ \{x \in \text{space } (\text{distr } M' \ M \ f). \ \text{lfp } F \ x\} =$
 $(\text{SUP } i. \ \text{emeasure } (\text{distr } M' \ M \ f) \ \{x \in \text{space } (\text{distr } M' \ M \ f). \ (F \ \sim i) \ (\lambda x. \ \text{False}) \ x\})$
using *<P M>*
proof (*coinduction arbitrary: M rule: emeasure_lfp'*)
case (*measurable A N*) **then have** $\bigwedge N. \ P \ N \ \Longrightarrow \ \text{Measurable.pred } (\text{distr } M' \ N \ f) \ A$
by *metis*
then have $\bigwedge N. \ P \ N \ \Longrightarrow \ \text{Measurable.pred } N \ A$
by *simp*
with *<P N>*[*THEN **] **show** *?case*
by *auto*
qed fact
then show $\text{emeasure } (\text{distr } M' \ M \ f) \ \{x \in \text{space } M. \ \text{lfp } F \ x\} =$
 $(\text{SUP } i. \ \text{emeasure } (\text{distr } M' \ M \ f) \ \{x \in \text{space } M. \ (F \ \sim i) \ (\lambda x. \ \text{False}) \ x\})$
by *simp*
qed

lemma *distr_id*[*simp*]: $\text{distr } N \ N \ (\lambda x. \ x) = N$
by (*rule measure_eqI*) (*auto simp: emeasure_distr*)

lemma *distr_id2*: $\text{sets } M = \text{sets } N \ \Longrightarrow \ \text{distr } N \ M \ (\lambda x. \ x) = N$

by (rule measure_eqI) (auto simp: emeasure_distr)

lemma *measure_distr*:

$f \in \text{measurable } M \ N \implies S \in \text{sets } N \implies \text{measure } (\text{distr } M \ N \ f) \ S = \text{measure } M$
 $(f \text{ - ' } S \cap \text{space } M)$

by (simp add: emeasure_distr measure_def)

lemma *distr_cong_AE*:

assumes $1: M = K$ *sets* $N = \text{sets } L$ **and**

$2: (AE \ x \ \text{in } M. f \ x = g \ x)$ **and** $f \in \text{measurable } M \ N$ **and** $g \in \text{measurable } K \ L$

shows $\text{distr } M \ N \ f = \text{distr } K \ L \ g$

proof (rule measure_eqI)

fix A **assume** $A \in \text{sets } (\text{distr } M \ N \ f)$

with *assms* **show** $\text{emeasure } (\text{distr } M \ N \ f) \ A = \text{emeasure } (\text{distr } K \ L \ g) \ A$

by (auto simp add: emeasure_distr intro!: emeasure_eq_AE measurable_sets)

qed (insert 1, simp)

lemma *AE_distrD*:

assumes $f: f \in \text{measurable } M \ M'$

and $AE: AE \ x \ \text{in } \text{distr } M \ M' \ f. P \ x$

shows $AE \ x \ \text{in } M. P \ (f \ x)$

proof –

from $AE[THEN \ AE_E]$ **obtain** N

where $\{x \in \text{space } (\text{distr } M \ M' \ f). \neg P \ x\} \subseteq N$

$\text{emeasure } (\text{distr } M \ M' \ f) \ N = 0$

$N \in \text{sets } (\text{distr } M \ M' \ f)$

by *auto*

with f **show** *?thesis*

by (simp add: eventually_ae_filter, intro bexI[of _ $f \text{ - ' } N \cap \text{space } M$])

(auto simp: emeasure_distr measurable_def)

qed

lemma *AE_distr_iff*:

assumes $f[\text{measurable}]: f \in \text{measurable } M \ N$ **and** $P[\text{measurable}]: \{x \in \text{space } N. P \ x\} \in \text{sets } N$

shows $(AE \ x \ \text{in } \text{distr } M \ N \ f. P \ x) \longleftrightarrow (AE \ x \ \text{in } M. P \ (f \ x))$

proof (subst (1 2) *AE_iff_measurable[OF _ refl]*)

have $f \text{ - ' } \{x \in \text{space } N. \neg P \ x\} \cap \text{space } M = \{x \in \text{space } M. \neg P \ (f \ x)\}$

using $f[THEN \ \text{measurable_space}]$ **by** *auto*

then **show** $(\text{emeasure } (\text{distr } M \ N \ f) \ \{x \in \text{space } (\text{distr } M \ N \ f). \neg P \ x\} = 0) =$
 $(\text{emeasure } M \ \{x \in \text{space } M. \neg P \ (f \ x)\} = 0)$

by (simp add: emeasure_distr)

qed *auto*

lemma *null_sets_distr_iff*:

$f \in \text{measurable } M \ N \implies A \in \text{null_sets } (\text{distr } M \ N \ f) \longleftrightarrow f \text{ - ' } A \cap \text{space } M \in$
 $\text{null_sets } M \wedge A \in \text{sets } N$

by (auto simp add: null_sets_def emeasure_distr)

proposition *distr_distr*:

$g \in \text{measurable } N L \implies f \in \text{measurable } M N \implies \text{distr } (\text{distr } M N f) L g = \text{distr } M L (g \circ f)$

by (*auto simp add: emeasure_distr measurable_space*
intro!: arg_cong[where f=emeasure M] measure_eqI)

7.3.9 Real measure values

lemma *ring_of_finite_sets*: $\text{ring_of_sets } (\text{space } M) \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\}$

proof (*rule ring_of_setsI*)

show $a \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\} \implies b \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\} \implies$

$a \cup b \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\}$ **for** $a b$

using *emeasure_subadditive[of a M b]* **by** (*auto simp: top_unique*)

show $a \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\} \implies b \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\} \implies$

$a - b \in \{A \in \text{sets } M. \text{emeasure } M A \neq \text{top}\}$ **for** $a b$

using *emeasure_mono[of a - b a M]* **by** (*auto simp: top_unique*)

qed (*auto dest: sets.sets_into_space*)

lemma *measure_nonneg[simp]*: $0 \leq \text{measure } M A$

unfolding *measure_def* **by** *auto*

lemma *measure_nonneg'* [*simp*]: $\neg \text{measure } M A < 0$

using *measure_nonneg not_le* **by** *blast*

lemma *zero_less_measure_iff*: $0 < \text{measure } M A \longleftrightarrow \text{measure } M A \neq 0$

using *measure_nonneg[of M A]* **by** (*auto simp add: le_less*)

lemma *measure_le_0_iff*: $\text{measure } M X \leq 0 \longleftrightarrow \text{measure } M X = 0$

using *measure_nonneg[of M X]* **by** *linarith*

lemma *measure_empty[simp]*: $\text{measure } M \{\} = 0$

unfolding *measure_def* **by** (*simp add: zero_ennreal.rep_eq*)

lemma *emeasure_eq_ennreal_measure*:

$\text{emeasure } M A \neq \text{top} \implies \text{emeasure } M A = \text{ennreal } (\text{measure } M A)$

by (*cases emeasure M A rule: ennreal_cases*) (*auto simp: measure_def*)

lemma *measure_zero_top*: $\text{emeasure } M A = \text{top} \implies \text{measure } M A = 0$

by (*simp add: measure_def*)

lemma *measure_eq_emeasure_eq_ennreal*: $0 \leq x \implies \text{emeasure } M A = \text{ennreal } x \implies \text{measure } M A = x$

using *emeasure_eq_ennreal_measure[of M A]*

by (*cases A \in M*) (*auto simp: measure_notin_sets emeasure_notin_sets*)

lemma *enn2real_plus*: $a < \text{top} \implies b < \text{top} \implies \text{enn2real } (a + b) = \text{enn2real } a + \text{enn2real } b$

by (*simp add*: *enn2real_def plus_ennreal.rep_eq real_of_ereal_add less_top del*: *real_of_ereal_enn2ereal*)

lemma *enn2real_sum*: $(\bigwedge i. i \in I \implies f\ i < \text{top}) \implies \text{enn2real } (\text{sum } f\ I) = \text{sum } (\text{enn2real } \circ f)\ I$

by (*induction I rule*: *infinite_finite_induct*) (*auto simp*: *enn2real_plus*)

lemma *measure_eq_AE*:

assumes *iff*: $A \in \text{sets } M. x \in A \longleftrightarrow x \in B$

assumes *A*: $A \in \text{sets } M$ **and** *B*: $B \in \text{sets } M$

shows $\text{measure } M\ A = \text{measure } M\ B$

using *assms emeasure_eq_AE[OF assms]* **by** (*simp add*: *measure_def*)

lemma *measure_Union*:

$\text{emeasure } M\ A \neq \infty \implies \text{emeasure } M\ B \neq \infty \implies A \in \text{sets } M \implies B \in \text{sets } M \implies A \cap B = \{\}$

$\text{measure } M\ (A \cup B) = \text{measure } M\ A + \text{measure } M\ B$

by (*simp add*: *measure_def plus_emeasure[symmetric] enn2real_plus less_top*)

lemma *measure_finite_Union*:

$\text{finite } S \implies A\ S \subseteq \text{sets } M \implies \text{disjoint_family_on } A\ S \implies (\bigwedge i. i \in S \implies \text{emeasure } M\ (A\ i) \neq \infty) \implies$

$\text{measure } M\ (\bigcup i \in S. A\ i) = (\sum i \in S. \text{measure } M\ (A\ i))$

by (*induction S rule*: *finite_induct*)

(*auto simp*: *disjoint_family_on_insert measure_Union sum_emeasure[symmetric] sets.countable_UN'[OF countable_finite]*)

lemma *measure_Diff*:

assumes *finite*: $\text{emeasure } M\ A \neq \infty$

and *measurable*: $A \in \text{sets } M\ B \in \text{sets } M\ B \subseteq A$

shows $\text{measure } M\ (A - B) = \text{measure } M\ A - \text{measure } M\ B$

proof –

have $\text{emeasure } M\ (A - B) \leq \text{emeasure } M\ A$ $\text{emeasure } M\ B \leq \text{emeasure } M\ A$

using *measurable* **by** (*auto intro!*: *emeasure_mono*)

hence $\text{measure } M\ ((A - B) \cup B) = \text{measure } M\ (A - B) + \text{measure } M\ B$

using *measurable finite* **by** (*rule_tac measure_Union*) (*auto simp*: *top_unique*)

thus *thesis* **using** $\langle B \subseteq A \rangle$ **by** (*auto simp*: *Un_absorb2*)

qed

lemma *measure_UNION*:

assumes *measurable*: $\text{range } A \subseteq \text{sets } M\ \text{disjoint_family } A$

assumes *finite*: $\text{emeasure } M\ (\bigcup i. A\ i) \neq \infty$

shows $(\lambda i. \text{measure } M\ (A\ i)) \text{ sums } (\text{measure } M\ (\bigcup i. A\ i))$

proof –

have $(\lambda i. \text{emeasure } M\ (A\ i)) \text{ sums } (\text{emeasure } M\ (\bigcup i. A\ i))$

unfolding *suminf_emeasure*[*OF measurable, symmetric*] **by** (*simp add*: *summable_sums*)

moreover

```

{ fix i
  have emeasure M (A i) ≤ emeasure M (⋃ i. A i)
    using measurable by (auto intro!: emeasure_mono)
  then have emeasure M (A i) = ennreal ((measure M (A i)))
    using finite by (intro emeasure_eq_ennreal_measure) (auto simp: top_unique)
}
ultimately show ?thesis using finite
  by (subst (asm) (2) emeasure_eq_ennreal_measure) simp_all
qed

```

lemma *measure_subadditive*:

```

assumes measurable: A ∈ sets M B ∈ sets M
and fin: emeasure M A ≠ ∞ emeasure M B ≠ ∞
shows measure M (A ∪ B) ≤ measure M A + measure M B
proof -
  have emeasure M (A ∪ B) ≠ ∞
    using emeasure_subadditive[OF measurable] fin by (auto simp: top_unique)
  then show (measure M (A ∪ B)) ≤ (measure M A) + (measure M B)
    unfolding measure_def
    by (metis emeasure_subadditive[OF measurable] fin enn2real_mono enn2real_plus

        ennreal_add_less_top infinity_ennreal_def less_top)
qed

```

lemma *measure_subadditive_finite*:

```

assumes A: finite I A'I ⊆ sets M and fin: ⋀i. i ∈ I ⇒ emeasure M (A i) ≠
∞
shows measure M (⋃ i ∈ I. A i) ≤ (∑ i ∈ I. measure M (A i))
proof -
  { have emeasure M (⋃ i ∈ I. A i) ≤ (∑ i ∈ I. emeasure M (A i))
    using emeasure_subadditive_finite[OF A] .
    also have ... < ∞
      using fin by (simp add: less_top A)
    finally have emeasure M (⋃ i ∈ I. A i) ≠ top by simp }
note * = this
show ?thesis
  using emeasure_subadditive_finite[OF A] fin
  unfolding emeasure_eq_ennreal_measure[OF *]
  by (simp_all add: sum_nonneg emeasure_eq_ennreal_measure)
qed

```

lemma *measure_subadditive_countably*:

```

assumes A: range A ⊆ sets M and fin: (∑ i. emeasure M (A i)) ≠ ∞
shows measure M (⋃ i. A i) ≤ (∑ i. measure M (A i))
proof -
  have **: ⋀i. emeasure M (A i) ≠ top
    using fin ennreal_suminf_lessD[of λi. emeasure M (A i)] by (simp add:
less_top)
  have ge0: (∑ i. Sigma_Algebra.measure M (A i)) ≥ 0

```

```

using fin_emeasure_eq_ennreal_measure[OF **]
by (metis infinity_ennreal_def measure_nonneg suminf_cong suminf_nonneg
summable_suminf_not_top)
have emeasure M ( $\bigcup i. A i$ )  $\neq$  top
by (metis A_emeasure_subadditive_countably_fin infinity_ennreal_def neg_top_trans)
then have ennreal (measure M ( $\bigcup i. A i$ )) = emeasure M ( $\bigcup i. A i$ )
by (rule emeasure_eq_ennreal_measure[symmetric])
also have ...  $\leq$  ( $\sum i. emeasure M (A i)$ )
using emeasure_subadditive_countably[OF A] .
also have ... = ennreal ( $\sum i. measure M (A i)$ )
using fin_unfolding_emeasure_eq_ennreal_measure[OF **]
by (subst suminf_ennreal) (auto simp: **)
finally show ?thesis
using ge0_ennreal_le_iff by blast
qed

```

lemma *measure_Un_null_set*: $A \in \text{sets } M \implies B \in \text{null_sets } M \implies \text{measure } M (A \cup B) = \text{measure } M A$
by (simp add: measure_def_emeasure_Un_null_set)

lemma *measure_Diff_null_set*: $A \in \text{sets } M \implies B \in \text{null_sets } M \implies \text{measure } M (A - B) = \text{measure } M A$
by (simp add: measure_def_emeasure_Diff_null_set)

lemma *measure_eq_sum_singleton*:
 $\text{finite } S \implies (\bigwedge x. x \in S \implies \{x\} \in \text{sets } M) \implies (\bigwedge x. x \in S \implies \text{emeasure } M \{x\} \neq \infty) \implies$
 $\text{measure } M S = (\sum_{x \in S}. \text{measure } M \{x\})$
using emeasure_eq_sum_singleton[of S M]
by (intro measure_eq_emeasure_eq_ennreal) (auto simp: sum_nonneg_emeasure_eq_ennreal_measure)

lemma *Lim_measure_incseq*:
assumes A: $\text{range } A \subseteq \text{sets } M$ *incseq* A **and** fin: $\text{emeasure } M (\bigcup i. A i) \neq \infty$
shows $(\lambda i. \text{measure } M (A i)) \longrightarrow \text{measure } M (\bigcup i. A i)$
proof (rule tendsto_ennrealD)
have ennreal (measure M ($\bigcup i. A i$)) = emeasure M ($\bigcup i. A i$)
using fin **by** (auto simp: emeasure_eq_ennreal_measure)
moreover have ennreal (measure M (A i)) = emeasure M (A i) **for** i
using assms emeasure_mono[of A $\bigcup i. A i$ M]
by (intro emeasure_eq_ennreal_measure[symmetric]) (auto simp: less_top_UN_upper intro: le_less_trans)
ultimately show $(\lambda x. \text{ennreal } (\text{measure } M (A x))) \longrightarrow \text{ennreal } (\text{measure } M (\bigcup i. A i))$
using A **by** (auto intro!: Lim_emeasure_incseq)
qed auto

lemma *Lim_measure_decseq*:
assumes A: $\text{range } A \subseteq \text{sets } M$ *decseq* A **and** fin: $\bigwedge i. \text{emeasure } M (A i) \neq \infty$

2130

shows $(\lambda n. \text{measure } M (A n)) \longrightarrow \text{measure } M (\bigcap i. A i)$
proof (rule tendsto_ennrealD)
have $\text{ennreal } (\text{measure } M (\bigcap i. A i)) = \text{emeasure } M (\bigcap i. A i)$
using $\text{fin}[of\ 0] A \text{ emeasure_mono}[of\ \bigcap i. A i\ A\ 0\ M]$
by (auto intro!: $\text{emeasure_eq_ennreal_measure}[\text{symmetric}] \text{ simp: INT_lower_less_top}$ intro: le_less_trans)
moreover have $\text{ennreal } (\text{measure } M (A i)) = \text{emeasure } M (A i)$ **for** i
using $A \text{ fin}[of\ i]$ **by** (intro $\text{emeasure_eq_ennreal_measure}[\text{symmetric}]$) auto
ultimately show $(\lambda x. \text{ennreal } (\text{measure } M (A x))) \longrightarrow \text{ennreal } (\text{measure } M (\bigcap i. A i))$
using $\text{fin } A$ **by** (auto intro!: $\text{Lim_emeasure_decseq}$)
qed auto

7.3.10 Set of measurable sets with finite measure

definition $\text{fmeasurable} :: 'a \text{ measure} \Rightarrow 'a \text{ set set}$ **where**
 $\text{fmeasurable } M = \{A \in \text{sets } M. \text{emeasure } M A < \infty\}$

lemma $\text{fmeasurableD}[dest, \text{measurable_dest}]: A \in \text{fmeasurable } M \Longrightarrow A \in \text{sets } M$
by (auto simp: fmeasurable_def)

lemma $\text{fmeasurableD2}: A \in \text{fmeasurable } M \Longrightarrow \text{emeasure } M A \neq \text{top}$
by (auto simp: fmeasurable_def)

lemma $\text{fmeasurableI}: A \in \text{sets } M \Longrightarrow \text{emeasure } M A < \infty \Longrightarrow A \in \text{fmeasurable } M$
by (auto simp: fmeasurable_def)

lemma $\text{fmeasurableI_null_sets}: A \in \text{null_sets } M \Longrightarrow A \in \text{fmeasurable } M$
by (auto simp: fmeasurable_def)

lemma $\text{fmeasurableI2}: A \in \text{fmeasurable } M \Longrightarrow B \subseteq A \Longrightarrow B \in \text{sets } M \Longrightarrow B \in \text{fmeasurable } M$
using $\text{emeasure_mono}[of\ B\ A\ M]$ **by** (auto simp: fmeasurable_def)

lemma $\text{measure_mono_fmeasurable}$:
 $A \subseteq B \Longrightarrow A \in \text{sets } M \Longrightarrow B \in \text{fmeasurable } M \Longrightarrow \text{measure } M A \leq \text{measure } M B$
by (auto simp: $\text{measure_def fmeasurable_def}$ intro!: $\text{emeasure_mono enn2real_mono}$)

lemma $\text{emeasure_eq_measure2}: A \in \text{fmeasurable } M \Longrightarrow \text{emeasure } M A = \text{measure } M A$
by (simp add: $\text{emeasure_eq_ennreal_measure fmeasurable_def less_top}$)

interpretation fmeasurable : $\text{ring_of_sets space } M \text{ fmeasurable } M$

proof (rule ring_of_setsI)

show $\text{fmeasurable } M \subseteq \text{Pow } (\text{space } M) \{\}$ $\in \text{fmeasurable } M$

by (auto simp: $\text{fmeasurable_def dest: sets.sets_into_space}$)

fix $a\ b$ **assume** $*$: $a \in \text{fmeasurable } M\ b \in \text{fmeasurable } M$

```

then have  $\text{emeasure } M (a \cup b) \leq \text{emeasure } M a + \text{emeasure } M b$ 
  by (intro emeasure_subadditive) auto
also have  $\dots < \text{top}$ 
  using * by (auto simp: fmeasurable_def)
finally show  $a \cup b \in \text{fmeasurable } M$ 
  using * by (auto intro: fmeasurableI)
show  $a - b \in \text{fmeasurable } M$ 
  using emeasure_mono[of  $a - b$   $a$   $M$ ] * by (auto simp: fmeasurable_def)
qed

```

7.3.11 Measurable sets formed by unions and intersections

```

lemma fmeasurable_Diff:  $A \in \text{fmeasurable } M \implies B \in \text{sets } M \implies A - B \in \text{fmeasurable } M$ 
  using fmeasurableI2[of  $A$   $M$   $A - B$ ] by auto

```

```

lemma fmeasurable_Int_fmeasurable:
   $\llbracket S \in \text{fmeasurable } M; T \in \text{sets } M \rrbracket \implies (S \cap T) \in \text{fmeasurable } M$ 
  by (meson fmeasurableD fmeasurableI2 inf_le1 sets.Int)

```

```

lemma fmeasurable_UN:
  assumes countable  $I \wedge i. i \in I \implies F i \subseteq A \wedge i. i \in I \implies F i \in \text{sets } M$   $A \in \text{fmeasurable } M$ 
  shows  $(\bigcup i \in I. F i) \in \text{fmeasurable } M$ 
proof (rule fmeasurableI2)
  show  $A \in \text{fmeasurable } M (\bigcup i \in I. F i) \subseteq A$  using assms by auto
  show  $(\bigcup i \in I. F i) \in \text{sets } M$ 
  using assms by (intro sets.countable_UN') auto
qed

```

```

lemma fmeasurable_INT:
  assumes countable  $I i \in I \wedge i. i \in I \implies F i \in \text{sets } M$   $F i \in \text{fmeasurable } M$ 
  shows  $(\bigcap i \in I. F i) \in \text{fmeasurable } M$ 
proof (rule fmeasurableI2)
  show  $F i \in \text{fmeasurable } M (\bigcap i \in I. F i) \subseteq F i$ 
  using assms by auto
  show  $(\bigcap i \in I. F i) \in \text{sets } M$ 
  using assms by (intro sets.countable_INT') auto
qed

```

```

lemma measurable_measure_Diff:
  assumes  $A \in \text{fmeasurable } M$   $B \in \text{sets } M$   $B \subseteq A$ 
  shows  $\text{measure } M (A - B) = \text{measure } M A - \text{measure } M B$ 
  by (simp add: assms fmeasurableD fmeasurableD2 measure_Diff)

```

```

lemma measurable_Un_null_set:
  assumes  $B \in \text{null\_sets } M$ 
  shows  $(A \cup B \in \text{fmeasurable } M \wedge A \in \text{sets } M) \longleftrightarrow A \in \text{fmeasurable } M$ 
  using assms by (fastforce simp add: fmeasurable.Un fmeasurableI_null_sets)

```

2132

intro: fmeasurableI2)

lemma *measurable_Diff_null_set:*

assumes $B \in \text{null_sets } M$

shows $(A - B) \in \text{fmeasurable } M \wedge A \in \text{sets } M \iff A \in \text{fmeasurable } M$

using *assms*

by (*metis Un_Diff_cancel2 fmeasurable.Diff fmeasurableD fmeasurableI_null_sets measurable_Un_null_set*)

lemma *fmeasurable_Diff_D:*

assumes $m: T - S \in \text{fmeasurable } M$ $S \in \text{fmeasurable } M$ **and** *sub: $S \subseteq T$*

shows $T \in \text{fmeasurable } M$

proof –

have $T = S \cup (T - S)$

using *assms* **by** *blast*

then show *?thesis*

by (*metis m fmeasurable.Un*)

qed

lemma *measure_Un2:*

$A \in \text{fmeasurable } M \implies B \in \text{fmeasurable } M \implies \text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M (B - A)$

using *measure_Union[of M A B - A]* **by** (*auto simp: fmeasurableD2 fmeasurable.Diff*)

lemma *measure_Un3:*

assumes $A \in \text{fmeasurable } M$ $B \in \text{fmeasurable } M$

shows $\text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M B - \text{measure } M (A \cap B)$

proof –

have $\text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M (B - A)$

using *assms* **by** (*rule measure_Un2*)

also have $B - A = B - (A \cap B)$

by *auto*

also have $\text{measure } M (B - (A \cap B)) = \text{measure } M B - \text{measure } M (A \cap B)$

using *assms* **by** (*intro measure_Diff*) (*auto simp: fmeasurable_def*)

finally show *?thesis*

by *simp*

qed

lemma *measure_Un_AE:*

$\text{AE } x \text{ in } M. x \notin A \vee x \notin B \implies A \in \text{fmeasurable } M \implies B \in \text{fmeasurable } M \implies \text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M B$

by (*subst measure_Un2*) (*auto intro!: measure_eq_AE*)

lemma *measure_UNION_AE:*

assumes *I: finite I*

shows $(\bigwedge i. i \in I \implies F i \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda i j. \text{AE } x \text{ in } M. x \notin F i \vee x \notin F j) I \implies$


```

    measure M ( $\bigcup i \in I. F i$ ) = ( $\sum i \in I. \text{measure } M (F i)$ )
  unfolding AE_pairwise[OF countable_finite, OF I]
  using I
proof (induction I rule: finite_induct)
  case (insert x I)
  have measure M ( $F x \cup \bigcup (F ` I)$ ) = measure M (F x) + measure M ( $\bigcup (F ` I)$ )
  by (rule measure_Un_AE) (use insert in <auto simp: pairwise_insert>)
  with insert show ?case
  by (simp add: pairwise_insert)
qed simp

```

lemma *measure_UNION'*:

```

finite I  $\implies (\bigwedge i. i \in I \implies F i \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda i j. \text{disjnt } (F i) (F j)) I \implies$ 
  measure M ( $\bigcup i \in I. F i$ ) = ( $\sum i \in I. \text{measure } M (F i)$ )
  by (intro measure_UNION_AE) (auto simp: disjnt_def elim!: pairwise_mono intro!: always_eventually)

```

lemma *measure_Union_AE*:

```

finite F  $\implies (\bigwedge S. S \in F \implies S \in \text{fmeasurable } M) \implies \text{pairwise } (\lambda S T. \text{AE } x \text{ in } M. x \notin S \vee x \notin T) F \implies$ 
  measure M ( $\bigcup F$ ) = ( $\sum S \in F. \text{measure } M S$ )
  using measure_UNION_AE[of F  $\lambda x. x M$ ] by simp

```

lemma *measure_Union'*:

```

finite F  $\implies (\bigwedge S. S \in F \implies S \in \text{fmeasurable } M) \implies \text{pairwise disjnt } F \implies$ 
  measure M ( $\bigcup F$ ) = ( $\sum S \in F. \text{measure } M S$ )
  using measure_UNION'[of F  $\lambda x. x M$ ] by simp

```

lemma *measure_Un_le*:

```

assumes A ∈ sets M B ∈ sets M shows measure M (A ∪ B) ≤ measure M A + measure M B
proof cases
  assume A ∈ fmeasurable M ∧ B ∈ fmeasurable M
  with measure_subadditive[of A M B] assms show ?thesis
  by (auto simp: fmeasurableD2)
next
  assume ¬ (A ∈ fmeasurable M ∧ B ∈ fmeasurable M)
  then have A ∪ B ∉ fmeasurable M
  using fmeasurableI2[of A ∪ B M A] fmeasurableI2[of A ∪ B M B] assms by auto
  with assms show ?thesis
  by (auto simp: fmeasurable_def measure_def less_top[symmetric])
qed

```

lemma *measure_UNION_le*:

```

finite I  $\implies (\bigwedge i. i \in I \implies F i \in \text{sets } M) \implies \text{measure } M (\bigcup i \in I. F i) \leq (\sum i \in I. \text{measure } M (F i))$ 

```

proof (*induction I rule: finite_induct*)
case (*insert i I*)
then have $\text{measure } M (\bigcup_{i \in \text{insert } i I} F i) = \text{measure } M (F i \cup \bigcup (F ' I))$
by *simp*
also from insert have $\text{measure } M (F i \cup \bigcup (F ' I)) \leq \text{measure } M (F i) +$
 $\text{measure } M (\bigcup (F ' I))$
by (*intro measure_Un_le sets.finite_Union*) *auto*
also have $\text{measure } M (\bigcup_{i \in I} F i) \leq (\sum_{i \in I} \text{measure } M (F i))$
using *insert by auto*
finally show *?case*
using *insert by simp*
qed *simp*

lemma *measure_Union_le*:
 $\text{finite } F \implies (\bigwedge S. S \in F \implies S \in \text{sets } M) \implies \text{measure } M (\bigcup F) \leq (\sum_{S \in F} \text{measure } M S)$
using *measure_UNION_le[of F λx. x M]* **by** *simp*

Version for indexed union over a countable set

lemma

assumes *countable I and I: $\bigwedge i. i \in I \implies A i \in \text{fmeasurable } M$*
and bound: $\bigwedge I'. I' \subseteq I \implies \text{finite } I' \implies \text{measure } M (\bigcup_{i \in I'} A i) \leq B$
shows *fmeasurable_UN_bound: $(\bigcup_{i \in I} A i) \in \text{fmeasurable } M$ (is ?fm)*
and *measure_UN_bound: $\text{measure } M (\bigcup_{i \in I} A i) \leq B$ (is ?m)*

proof –

have $B \geq 0$
using *bound by force*
have *?fm \wedge ?m*

proof *cases*

assume $I = \{\}$
with $\langle B \geq 0 \rangle$ **show** *?thesis*
by *simp*

next

assume $I \neq \{\}$
have $(\bigcup_{i \in I} A i) = (\bigcup i. (\bigcup_{n \leq i} A (\text{from_nat_into } I n)))$
by (*subst range_from_nat_into[symmetric, OF $\langle I \neq \{\} \rangle$ $\langle \text{countable } I \rangle$]*) *auto*
then have $\text{emeasure } M (\bigcup_{i \in I} A i) = \text{emeasure } M (\bigcup i. (\bigcup_{n \leq i} A (\text{from_nat_into } I n)))$
by *simp*

also have $\dots = (\text{SUP } i. \text{emeasure } M (\bigcup_{n \leq i} A (\text{from_nat_into } I n)))$
using $\langle I \neq \{\} \rangle$ [*THEN from_nat_into*] **by** (*intro SUP_emeasure_incseq[symmetric]*)
(fastforce simp: incseq_Suc_iff)+

also have $\dots \leq B$

proof (*intro SUP_least*)

fix $i :: \text{nat}$

have $\text{emeasure } M (\bigcup_{n \leq i} A (\text{from_nat_into } I n)) = \text{measure } M (\bigcup_{n \leq i} A (\text{from_nat_into } I n))$

using $\langle I \neq \{\} \rangle$ [*THEN from_nat_into*] **by** (*intro emeasure_eq_measure2 fmeasurable.finite_UN*) *auto*

also have $\dots = \text{measure } M (\bigcup_{n \in \text{from_nat_into } I ' \{..i\}} A n)$

```

    by simp
  also have ... ≤ B
    by (intro ennreal_leI bound) (auto intro: from_nat_into[OF ‹I ≠ {}›])
  finally show emeasure M (⋃ n≤i. A (from_nat_into I n)) ≤ ennreal B .
qed
finally have *: emeasure M (⋃ i∈I. A i) ≤ B .
then have ?fm
  using I ‹countable I› by (intro fmeasurableI conjI) (auto simp: less_top[symmetric]
top_unique)
  with * ‹0 ≤ B› show ?thesis
  by (simp add: emeasure_eq_measure2)
qed
then show ?fm ?m by auto
qed

```

Version for big union of a countable set

lemma

```

  assumes countable D
  and meas:  $\bigwedge D. D \in \mathcal{D} \implies D \in \text{fmeasurable } M$ 
  and bound:  $\bigwedge \mathcal{E}. [\mathcal{E} \subseteq \mathcal{D}; \text{finite } \mathcal{E}] \implies \text{measure } M (\bigcup \mathcal{E}) \leq B$ 
  shows fmeasurable_Union_bound:  $\bigcup \mathcal{D} \in \text{fmeasurable } M$  (is ?fm)
  and measure_Union_bound:  $\text{measure } M (\bigcup \mathcal{D}) \leq B$  (is ?m)
proof -
  have B ≥ 0
  using bound by force
  have ?fm ∧ ?m
  proof (cases D = {})
    case True
    with ‹B ≥ 0› show ?thesis
    by auto
  next
    case False
  then obtain D :: nat ⇒ 'a set where D: D = range D
  using ‹countable D› uncountable_def by force
  have 1:  $\bigwedge i. D i \in \text{fmeasurable } M$ 
  by (simp add: D meas)
  have 2:  $\bigwedge I'. \text{finite } I' \implies \text{measure } M (\bigcup x \in I'. D x) \leq B$ 
  by (simp add: D bound image_subset_iff)
  show ?thesis
  unfolding D
  by (intro conjI fmeasurable_UN_bound [OF _ 1 2] measure_UN_bound
[OF _ 1 2]) auto
  qed
  then show ?fm ?m by auto
qed

```

Version for indexed union over the type of naturals

lemma

```

  fixes S :: nat ⇒ 'a set

```

assumes $S: \bigwedge i. S\ i \in \text{fmeasurable } M$ **and** $B: \bigwedge n. \text{measure } M (\bigcup_{i \leq n} S\ i) \leq B$
shows $\text{fmeasurable_countable_Union}: (\bigcup i. S\ i) \in \text{fmeasurable } M$
and $\text{measure_countable_Union_le}: \text{measure } M (\bigcup i. S\ i) \leq B$
proof –
have $mB: \text{measure } M (\bigcup_{i \in I} S\ i) \leq B$ **if finite** I **for** I
proof –
have $(\bigcup_{i \in I} S\ i) \subseteq (\bigcup_{i \leq \text{Max } I} S\ i)$
using Max_ge **that by force**
then have $\text{measure } M (\bigcup_{i \in I} S\ i) \leq \text{measure } M (\bigcup_{i \leq \text{Max } I} S\ i)$
by $(\text{rule } \text{measure_mono_fmeasurable})$ $(\text{use } S \text{ in } \langle \text{blast} \rangle)$
then show $?thesis$
using B order_trans **by blast**
qed
show $(\bigcup i. S\ i) \in \text{fmeasurable } M$
by $(\text{auto intro: fmeasurable_UN_bound } [OF_ \ S\ mB])$
show $\text{measure } M (\bigcup n. S\ n) \leq B$
by $(\text{auto intro: measure_UN_bound } [OF_ \ S\ mB])$
qed

lemma $\text{measure_diff_le_measure_setdiff}$:
assumes $S \in \text{fmeasurable } M$ $T \in \text{fmeasurable } M$
shows $\text{measure } M\ S - \text{measure } M\ T \leq \text{measure } M (S - T)$
proof –
have $\text{measure } M\ S \leq \text{measure } M ((S - T) \cup T)$
by $(\text{simp add: asms fmeasurable.Un fmeasurableD measure_mono_fmeasurable})$
also have $\dots \leq \text{measure } M (S - T) + \text{measure } M\ T$
using asms **by** $(\text{blast intro: measure_Un_le})$
finally show $?thesis$
by $(\text{simp add: algebra_simps})$
qed

lemma $\text{suminf_exist_split2}$:
fixes $f :: \text{nat} \Rightarrow 'a::\text{real_normed_vector}$
assumes $\text{summable } f$
shows $(\lambda n. (\sum k. f(k+n))) \longrightarrow 0$
by $(\text{subst } \text{lim_sequentially, auto simp add: dist_norm suminf_exist_split} [OF_ \ \text{asms}])$

lemma $\text{emeasure_union_summable}$:
assumes $[\text{measurable}]: \bigwedge n. A\ n \in \text{sets } M$
and $\bigwedge n. \text{emeasure } M (A\ n) < \infty$ $\text{summable } (\lambda n. \text{measure } M (A\ n))$
shows $\text{emeasure } M (\bigcup n. A\ n) < \infty$ $\text{emeasure } M (\bigcup n. A\ n) \leq (\sum n. \text{measure } M (A\ n))$
proof –
define B **where** $B = (\lambda N. (\bigcup_{n \in \{..<N\}} A\ n))$
have $[\text{measurable}]: B\ N \in \text{sets } M$ **for** N **unfolding** B_def **by** auto
have $(\lambda N. \text{emeasure } M (B\ N)) \longrightarrow \text{emeasure } M (\bigcup N. B\ N)$
apply $(\text{rule } \text{Lim_emeasure_incseq})$ **unfolding** B_def **by** $(\text{auto simp add: SUP_subset_mono incseq_def})$

```

moreover have  $\text{emeasure } M (B N) \leq \text{ennreal } (\sum n. \text{measure } M (A n))$  for  $N$ 
proof –
  have  $*$ :  $(\sum n < N. \text{measure } M (A n)) \leq (\sum n. \text{measure } M (A n))$ 
    using  $\text{assms}(3)$   $\text{measure\_nonneg\_sum\_le\_suminf}$  by  $\text{blast}$ 

  have  $\text{emeasure } M (B N) \leq (\sum n < N. \text{emeasure } M (A n))$ 
    unfolding  $B\_def$  by  $(\text{rule } \text{emeasure\_subadditive\_finite}, \text{auto})$ 
  also have  $\dots = (\sum n < N. \text{ennreal}(\text{measure } M (A n)))$ 
    using  $\text{assms}(2)$  by  $(\text{simp add: } \text{emeasure\_eq\_ennreal\_measure\_less\_top})$ 
  also have  $\dots = \text{ennreal } (\sum n < N. \text{measure } M (A n))$ 
    by  $\text{auto}$ 
  also have  $\dots \leq \text{ennreal } (\sum n. \text{measure } M (A n))$ 
    using  $*$  by  $(\text{auto simp: } \text{ennreal\_leI})$ 
  finally show  $?thesis$  by  $\text{simp}$ 
qed

ultimately have  $\text{emeasure } M (\bigcup N. B N) \leq \text{ennreal } (\sum n. \text{measure } M (A n))$ 
  by  $(\text{simp add: } \text{Lim\_bounded})$ 
then show  $\text{emeasure } M (\bigcup n. A n) \leq (\sum n. \text{measure } M (A n))$ 
  unfolding  $B\_def$  by  $(\text{metis } \text{UN\_UN\_flatten } \text{UN\_lessThan\_UNIV})$ 
then show  $\text{emeasure } M (\bigcup n. A n) < \infty$ 
  by  $(\text{auto simp: } \text{less\_top[symmetric] } \text{top\_unique})$ 
qed

lemma  $\text{borel\_cantelli\_limsup1}$ :
  assumes  $[\text{measurable}]$ :  $\bigwedge n. A n \in \text{sets } M$ 
    and  $\bigwedge n. \text{emeasure } M (A n) < \infty$   $\text{summable } (\lambda n. \text{measure } M (A n))$ 
  shows  $\text{limsup } A \in \text{null\_sets } M$ 
proof –
  have  $\text{emeasure } M (\text{limsup } A) \leq 0$ 
  proof  $(\text{rule } \text{LIMSEQ\_le\_const})$ 
    have  $(\lambda n. (\sum k. \text{measure } M (A (k+n)))) \longrightarrow 0$  by  $(\text{rule } \text{suminf\_exist\_split2}[OF \text{assms}(3)])$ 
    then show  $(\lambda n. \text{ennreal } (\sum k. \text{measure } M (A (k+n)))) \longrightarrow 0$ 
      unfolding  $\text{ennreal\_0[symmetric]}$  by  $(\text{intro } \text{tendsto\_ennrealI})$ 
    have  $\text{emeasure } M (\text{limsup } A) \leq (\sum k. \text{measure } M (A (k+n)))$  for  $n$ 
    proof –
      have  $I: (\bigcup k \in \{n..\}. A k) = (\bigcup k. A (k+n))$  by  $(\text{auto}, \text{metis } \text{le\_add\_diff\_inverse2}, \text{fastforce})$ 
      have  $\text{emeasure } M (\text{limsup } A) \leq \text{emeasure } M (\bigcup k \in \{n..\}. A k)$ 
        by  $(\text{rule } \text{emeasure\_mono}, \text{auto simp add: } \text{limsup\_INF\_SUP})$ 
      also have  $\dots = \text{emeasure } M (\bigcup k. A (k+n))$ 
        using  $I$  by  $\text{auto}$ 
      also have  $\dots \leq (\sum k. \text{measure } M (A (k+n)))$ 
        apply  $(\text{rule } \text{emeasure\_union\_summable})$ 
        using  $\text{assms summable\_ignore\_initial\_segment}[OF \text{assms}(3), \text{of } n]$  by  $\text{auto}$ 
      finally show  $?thesis$  by  $\text{simp}$ 
    qed
  then show  $\exists N. \forall n \geq N. \text{emeasure } M (\text{limsup } A) \leq (\sum k. \text{measure } M (A (k+n)))$ 

```

by auto
 qed
 then show ?thesis using assms(1) measurable_limsup by auto
 qed

lemma borel_cantelli_AE1:
 assumes [measurable]: $\bigwedge n. A\ n \in \text{sets } M$
 and $\bigwedge n. \text{emeasure } M (A\ n) < \infty$ summable $(\lambda n. \text{measure } M (A\ n))$
 shows AE x in M . eventually $(\lambda n. x \in \text{space } M - A\ n)$ sequentially
 proof –
 have AE x in M . $x \notin \text{limsup } A$
 using borel_cantelli_limsup1[OF assms] unfolding eventually_ae_filter by
 auto
 moreover have $\forall_F n$ in sequentially. $x \notin A\ n$ if $x \notin \text{limsup } A$ for x
 using that by (auto simp: limsup_INF_SUP eventually_sequentially)
 ultimately show ?thesis by auto
 qed

7.3.12 Measure spaces with $\text{emeasure } M (\text{space } M) < \infty$

locale finite_measure = sigma_finite_measure M for M +
 assumes finite_emeasure_space: $\text{emeasure } M (\text{space } M) \neq \text{top}$

lemma finite_measureI[Pure.intro]:
 $\text{emeasure } M (\text{space } M) \neq \infty \implies \text{finite_measure } M$
 proof qed (auto intro!: exI[of _ {space M}])

lemma (in finite_measure) emeasure_finite[simp, intro]: $\text{emeasure } M A \neq \text{top}$
 using finite_emeasure_space emeasure_space[of M A] by (auto simp: top_unique)

lemma (in finite_measure) fmeasurable_eq_sets: $\text{fmeasurable } M = \text{sets } M$
 by (auto simp: fmeasurable_def less_top[symmetric])

lemma (in finite_measure) emeasure_eq_ennreal: $\text{emeasure } M A = \text{ennreal } (\text{measure } M A)$
 by (intro emeasure_eq_ennreal_measure) simp

lemma (in finite_measure) emeasure_real: $\exists r. 0 \leq r \wedge \text{emeasure } M A = \text{ennreal } r$
 using emeasure_finite[of A] by (cases emeasure M A rule: ennreal_cases) auto

lemma (in finite_measure) bounded_measure: $\text{measure } M A \leq \text{measure } M (\text{space } M)$
 using emeasure_space[of M A] emeasure_real[of A] emeasure_real[of space M]
 by (auto simp: measure_def)

lemma (in finite_measure) finite_measure_Diff:
 assumes sets: $A \in \text{sets } M$ $B \in \text{sets } M$ and $B \subseteq A$
 shows $\text{measure } M (A - B) = \text{measure } M A - \text{measure } M B$

using *measure_Diff*[*OF _ assms*] **by simp**

lemma (in *finite_measure*) *finite_measure_Union*:
assumes *sets*: $A \in \text{sets } M \ B \in \text{sets } M$ **and** $A \cap B = \{\}$
shows $\text{measure } M (A \cup B) = \text{measure } M A + \text{measure } M B$
using *measure_Union*[*OF _ _ assms*] **by simp**

lemma (in *finite_measure*) *finite_measure_finite_Union*:
assumes *measurable*: $\text{finite } S \ A \cdot S \subseteq \text{sets } M$ *disjoint_family_on* $A \ S$
shows $\text{measure } M (\bigcup_{i \in S}. A \ i) = (\sum_{i \in S}. \text{measure } M (A \ i))$
using *measure_finite_Union*[*OF assms*] **by simp**

lemma (in *finite_measure*) *finite_measure_UNION*:
assumes A : $\text{range } A \subseteq \text{sets } M$ *disjoint_family* A
shows $(\lambda i. \text{measure } M (A \ i)) \text{ sums } (\text{measure } M (\bigcup i. A \ i))$
using *measure_UNION*[*OF A*] **by simp**

lemma (in *finite_measure*) *finite_measure_mono*:
assumes $A \subseteq B \ B \in \text{sets } M$ **shows** $\text{measure } M A \leq \text{measure } M B$
using *emeasure_mono*[*OF assms*] *emeasure_real*[*of A*] *emeasure_real*[*of B*] **by**
(*auto simp: measure_def*)

lemma (in *finite_measure*) *finite_measure_subadditive*:
assumes m : $A \in \text{sets } M \ B \in \text{sets } M$
shows $\text{measure } M (A \cup B) \leq \text{measure } M A + \text{measure } M B$
using *measure_subadditive*[*OF m*] **by simp**

lemma (in *finite_measure*) *finite_measure_subadditive_finite*:
assumes *finite* $I \ A \cdot I \subseteq \text{sets } M$ **shows** $\text{measure } M (\bigcup_{i \in I}. A \ i) \leq (\sum_{i \in I}. \text{measure } M (A \ i))$
using *measure_subadditive_finite*[*OF assms*] **by simp**

lemma (in *finite_measure*) *finite_measure_subadditive_countably*:
 $\text{range } A \subseteq \text{sets } M \implies \text{summable } (\lambda i. \text{measure } M (A \ i)) \implies \text{measure } M (\bigcup i. A \ i) \leq (\sum i. \text{measure } M (A \ i))$
by (*rule measure_subadditive_countably*)
(*simp_all add: ennreal_suminf_neq_top emeasure_eq_measure*)

lemma (in *finite_measure*) *finite_measure_eq_sum_singleton*:
assumes *finite* S **and** $*$: $\bigwedge x. x \in S \implies \{x\} \in \text{sets } M$
shows $\text{measure } M S = (\sum_{x \in S}. \text{measure } M \{x\})$
using *measure_eq_sum_singleton*[*OF assms*] **by simp**

lemma (in *finite_measure*) *finite_Lim_measure_incseq*:
assumes A : $\text{range } A \subseteq \text{sets } M$ *incseq* A
shows $(\lambda i. \text{measure } M (A \ i)) \longrightarrow \text{measure } M (\bigcup i. A \ i)$
using *Lim_measure_incseq*[*OF A*] **by simp**

lemma (in *finite_measure*) *finite_Lim_measure_decseq*:

assumes A : $\text{range } A \subseteq \text{sets } M \text{ decseq } A$
shows $(\lambda n. \text{measure } M (A n)) \longrightarrow \text{measure } M (\bigcap i. A i)$
using $\text{Lim_measure_decseq}[OF A]$ **by** simp

lemma (**in** finite_measure) $\text{finite_measure_compl}$:
assumes S : $S \in \text{sets } M$
shows $\text{measure } M (\text{space } M - S) = \text{measure } M (\text{space } M) - \text{measure } M S$
using $\text{measure_Diff}[OF _ \text{sets.top } S \text{ sets.sets_into_space}] S$ **by** simp

lemma (**in** finite_measure) $\text{finite_measure_mono_AE}$:
assumes imp : $\text{AE } x \text{ in } M. x \in A \longrightarrow x \in B$ **and** B : $B \in \text{sets } M$
shows $\text{measure } M A \leq \text{measure } M B$
using $\text{assms } \text{emeasure_mono_AE}[OF \text{imp } B]$
by ($\text{simp add: } \text{emeasure_eq_measure}$)

lemma (**in** finite_measure) $\text{finite_measure_eq_AE}$:
assumes iff : $\text{AE } x \text{ in } M. x \in A \longleftrightarrow x \in B$
assumes A : $A \in \text{sets } M$ **and** B : $B \in \text{sets } M$
shows $\text{measure } M A = \text{measure } M B$
using $\text{assms } \text{emeasure_eq_AE}[OF \text{assms}]$ **by** ($\text{simp add: } \text{emeasure_eq_measure}$)

lemma (**in** finite_measure) $\text{measure_increasing}$: $\text{increasing } M (\text{measure } M)$
by ($\text{auto intro!: } \text{finite_measure_mono } \text{simp: } \text{increasing_def}$)

lemma (**in** finite_measure) $\text{measure_zero_union}$:
assumes $s \in \text{sets } M$ $t \in \text{sets } M$ $\text{measure } M t = 0$
shows $\text{measure } M (s \cup t) = \text{measure } M s$
using assms
proof –
have $\text{measure } M (s \cup t) \leq \text{measure } M s$
using $\text{finite_measure_subadditive}[of s t] \text{assms}$ **by** auto
moreover **have** $\text{measure } M (s \cup t) \geq \text{measure } M s$
using assms **by** ($\text{blast intro: } \text{finite_measure_mono}$)
ultimately show $?thesis$ **by** simp
qed

lemma (**in** finite_measure) measure_eq_compl :
assumes $s \in \text{sets } M$ $t \in \text{sets } M$
assumes $\text{measure } M (\text{space } M - s) = \text{measure } M (\text{space } M - t)$
shows $\text{measure } M s = \text{measure } M t$
using $\text{assms } \text{finite_measure_compl}$ **by** auto

lemma (**in** finite_measure) $\text{measure_eq_bigunion_image}$:
assumes $\text{range } f \subseteq \text{sets } M$ $\text{range } g \subseteq \text{sets } M$
assumes $\text{disjoint_family } f$ $\text{disjoint_family } g$
assumes $\bigwedge n :: \text{nat. } \text{measure } M (f n) = \text{measure } M (g n)$
shows $\text{measure } M (\bigcup i. f i) = \text{measure } M (\bigcup i. g i)$
using assms
proof –


```

have a: ( $\lambda$  i. measure M (f i)) sums (measure M ( $\bigcup$  i. f i))
  by (rule finite_measure_UNION[OF assms(1,3)])
have b: ( $\lambda$  i. measure M (g i)) sums (measure M ( $\bigcup$  i. g i))
  by (rule finite_measure_UNION[OF assms(2,4)])
show ?thesis using sums_unique[OF b] sums_unique[OF a] assms by simp
qed

```

```

lemma (in finite_measure) measure_countably_zero:
  assumes range c  $\subseteq$  sets M
  assumes  $\bigwedge$  i. measure M (c i) = 0
  shows measure M ( $\bigcup$  i :: nat. c i) = 0
proof (rule antisym)
  show measure M ( $\bigcup$  i :: nat. c i)  $\leq$  0
    using finite_measure_subadditive_countably[OF assms(1)] by (simp add: assms(2))
qed simp

```

```

lemma (in finite_measure) measure_space_inter:
  assumes events: s  $\in$  sets M t  $\in$  sets M
  assumes measure M t = measure M (space M)
  shows measure M (s  $\cap$  t) = measure M s
proof -
  have measure M ((space M - s)  $\cup$  (space M - t)) = measure M (space M - s)
    using events assms finite_measure_compl[of t] by (auto intro!: measure_zero_union)
  also have (space M - s)  $\cup$  (space M - t) = space M - (s  $\cap$  t)
    by blast
  finally show measure M (s  $\cap$  t) = measure M s
    using events by (auto intro!: measure_eq_compl[of s  $\cap$  t s])
qed

```

```

lemma (in finite_measure) measure_equiprobable_finite_unions:
  assumes s: finite s  $\wedge$  x. x  $\in$  s  $\implies$  {x}  $\in$  sets M
  assumes  $\bigwedge$  x y.  $\llbracket$ x  $\in$  s; y  $\in$  s $\rrbracket \implies$  measure M {x} = measure M {y}
  shows measure M s = real (card s) * measure M {SOME x. x  $\in$  s}
proof cases
  assume s  $\neq$  {}
  then have  $\exists$  x. x  $\in$  s by blast
  from someI_ex[OF this] assms
  have prob_some:  $\bigwedge$  x. x  $\in$  s  $\implies$  measure M {x} = measure M {SOME y. y  $\in$  s}
    by blast
  have measure M s = ( $\sum$  x  $\in$  s. measure M {x})
    using finite_measure_eq_sum_singleton[OF s] by simp
  also have ... = ( $\sum$  x  $\in$  s. measure M {SOME y. y  $\in$  s}) using prob_some by
auto
  also have ... = real (card s) * measure M {(SOME x. x  $\in$  s)}
    using sum_constant assms by simp
  finally show ?thesis by simp
qed simp

```

```

lemma (in finite_measure) measure_real_sum_image_fn:

```

assumes $e \in \text{sets } M$
assumes $\bigwedge x. x \in s \implies e \cap f x \in \text{sets } M$
assumes *finite s*
assumes *disjoint*: $\bigwedge x y. \llbracket x \in s ; y \in s ; x \neq y \rrbracket \implies f x \cap f y = \{\}$
assumes *upper*: $\text{space } M \subseteq (\bigcup i \in s. f i)$
shows $\text{measure } M e = (\sum x \in s. \text{measure } M (e \cap f x))$
proof –
have $e \subseteq (\bigcup i \in s. f i)$
using $\langle e \in \text{sets } M \rangle$ *sets.sets_into_space upper* **by** *blast*
then have $e = (\bigcup i \in s. e \cap f i)$
by *auto*
hence $\text{measure } M e = \text{measure } M (\bigcup i \in s. e \cap f i)$ **by** *simp*
also have $\dots = (\sum x \in s. \text{measure } M (e \cap f x))$
proof (*rule finite_measure_finite_Union*)
show *finite s* **by** *fact*
show $(\lambda i. e \cap f i)'s \subseteq \text{sets } M$ **using** *assms(2)* **by** *auto*
show *disjoint_family_on* $(\lambda i. e \cap f i) s$
using *disjoint* **by** (*auto simp: disjoint_family_on_def*)
qed
finally show *?thesis* .
qed

lemma (*in finite_measure*) *measure_exclude*:
assumes $A \in \text{sets } M B \in \text{sets } M$
assumes $\text{measure } M A = \text{measure } M (\text{space } M) A \cap B = \{\}$
shows $\text{measure } M B = 0$
using *measure_space_inter[of B A]* *assms* **by** (*auto simp: ac_simps*)
lemma (*in finite_measure*) *finite_measure_distr*:
assumes $f: f \in \text{measurable } M M'$
shows *finite_measure* $(\text{distr } M M' f)$
proof (*rule finite_measureI*)
have $f -' \text{space } M' \cap \text{space } M = \text{space } M$ **using** f **by** (*auto dest: measurable_space*)
with f **show** $\text{emeasure } (\text{distr } M M' f) (\text{space } (\text{distr } M M' f)) \neq \infty$ **by** (*auto simp: emeasure_distr*)
qed

lemma *emeasure_gfp[consumes 1, case_names cont measurable]*:
assumes *sets[simp]*: $\bigwedge s. \text{sets } (M s) = \text{sets } N$
assumes $\bigwedge s. \text{finite_measure } (M s)$
assumes *cont*: $\text{inf_continuous } F \text{ inf_continuous } f$
assumes *meas*: $\bigwedge P. \text{Measurable.pred } N P \implies \text{Measurable.pred } N (F P)$
assumes *iter*: $\bigwedge P s. \text{Measurable.pred } N P \implies \text{emeasure } (M s) \{x \in \text{space } N. F P x\} = f (\lambda s. \text{emeasure } (M s) \{x \in \text{space } N. P x\}) s$
assumes *bound*: $\bigwedge P. f P \leq f (\lambda s. \text{emeasure } (M s) (\text{space } (M s)))$
shows $\text{emeasure } (M s) \{x \in \text{space } N. \text{gfp } F x\} = \text{gfp } f s$
proof (*subst gfp_transfer_bounded[where $\alpha = \lambda F s. \text{emeasure } (M s) \{x \in \text{space } N. F x\}$ and $P = \text{Measurable.pred } N, \text{symmetric}$]*)
interpret *finite_measure* $M s$ **for** s **by** *fact*

```

fix C assume decseq C  $\wedge$  i. Measurable.pred N (C i)
then show ( $\lambda$ s. emeasure (M s) {x  $\in$  space N. (INF i. C i) x}) = (INF i. ( $\lambda$ s.
emeasure (M s) {x  $\in$  space N. C i x}))
  unfolding INF_apply
  by (subst INF_emeasure_decseq) (auto simp: antimono_def fun_eq_iff intro!:
arg_cong2[where f=emeasure])
next
  show f x  $\leq$  ( $\lambda$ s. emeasure (M s) {x  $\in$  space N. F top x}) for x
  using bound[of x] sets_eq_imp_space_eq[OF sets] by (simp add: iter)
qed (auto simp add: iter le_fun_def INF_apply[abs_def] intro!: meas cont)

```

7.3.13 Counting space

lemma strict_monoI_Suc:

```

assumes ( $\wedge$ n. f n < f (Suc n)) shows strict_mono f
by (simp add: assms strict_mono_Suc_iff)

```

lemma emeasure_count_space:

```

assumes X  $\subseteq$  A shows emeasure (count_space A) X = (if finite X then of_nat
(card X) else  $\infty$ )

```

```

(is _ = ?M X)

```

```

unfolding count_space_def

```

proof (rule emeasure_measure_of_sigma)

```

show X  $\in$  Pow A using  $\langle$ X  $\subseteq$  A $\rangle$  by auto

```

```

show sigma_algebra A (Pow A) by (rule sigma_algebra_Pow)

```

```

show positive: positive (Pow A) ?M

```

```

by (auto simp: positive_def)

```

```

have additive: additive (Pow A) ?M

```

```

by (auto simp: card_Un_disjoint additive_def)

```

interpret ring_of_sets A Pow A

```

by (rule ring_of_setsI) auto

```

```

show countably_additive (Pow A) ?M

```

```

unfolding countably_additive_iff_continuous_from_below[OF positive addi-
tive]

```

proof safe

```

fix F :: nat  $\Rightarrow$  'a set assume incseq F

```

```

show ( $\lambda$ i. ?M (F i))  $\longrightarrow$  ?M ( $\bigcup$  i. F i)

```

proof cases

```

assume  $\exists$  i.  $\forall$  j  $\geq$  i. F i = F j

```

```

then obtain i where i:  $\forall$  j  $\geq$  i. F i = F j ..

```

```

with  $\langle$ incseq F $\rangle$  have F j  $\subseteq$  F i for j

```

```

by (cases i  $\leq$  j) (auto simp: incseq_def)

```

```

then have eq: ( $\bigcup$  i. F i) = F i

```

```

by auto

```

```

with i show ?thesis

```

```

by (auto intro!: Lim_transform_eventually[OF tendsto_const] eventu-
ally_sequentiallyI[where c=i])

```

next

```

assume  $\neg (\exists i. \forall j \geq i. F i = F j)$ 
then obtain  $f$  where  $f: \bigwedge i. i \leq f i \wedge i. F i \neq F (f i)$  by metis
then have  $\bigwedge i. F i \subseteq F (f i)$  using  $\langle incseq F \rangle$  by (auto simp: incseq_def)
with  $f$  have  $*$ :  $\bigwedge i. F i \subset F (f i)$  by auto

have incseq  $(\lambda i. ?M (F i))$ 
  using  $\langle incseq F \rangle$  unfolding incseq_def by (auto simp: card_mono dest: finite_subset)
then have  $(\lambda i. ?M (F i)) \longrightarrow (SUP n. ?M (F n))$ 
  by (rule LIMSEQ_SUP)

moreover have  $(SUP n. ?M (F n)) = top$ 
proof (rule ennreal_SUP_eq_top)
  fix  $n :: nat$  show  $\exists k :: nat \in UNIV. of\_nat n \leq ?M (F k)$ 
  proof (induct n)
    case (Suc n)
    then obtain  $k$  where  $of\_nat n \leq ?M (F k)$  ..
    moreover have  $finite (F k) \implies finite (F (f k)) \implies card (F k) < card (F (f k))$ 
      using  $\langle F k \subset F (f k) \rangle$  by (simp add: psubset_card_mono)
      moreover have  $finite (F (f k)) \implies finite (F k)$ 
      using  $\langle k \leq f k \rangle \langle incseq F \rangle$  by (auto simp: incseq_def dest: finite_subset)
      ultimately show ?case
      by (auto intro!: exI[of _ f k] simp del: of_nat_Suc)
    qed auto
  qed

moreover
have inj  $(\lambda n. F ((f \smallfrown n) 0))$ 
  by (intro strict_mono_imp_inj_on strict_monoI_Suc) (simp add: *)
then have  $1: infinite (range (\lambda i. F ((f \smallfrown i) 0)))$ 
  by (rule range_inj_infinite)
have infinite  $(Pow (\bigcup i. F i))$ 
  by (rule infinite_super[OF _ 1]) auto
then have infinite  $(\bigcup i. F i)$ 
  by auto
ultimately show ?thesis by (simp only:) simp

qed
qed
qed

lemma distr_bij_count_space:
  assumes  $f: bij\_betw f A B$ 
  shows distr  $(count\_space A) (count\_space B) f = count\_space B$ 
proof (rule measure_eqI)
  have  $f': f \in measurable (count\_space A) (count\_space B)$ 
  using  $f$  unfolding Pi_def bij_betw_def by auto
  fix  $X$  assume  $X \in sets (distr (count\_space A) (count\_space B) f)$ 

```

then have $X: X \in \text{sets } (\text{count_space } B)$ **by** *auto*
moreover from X **have** $f^{-1} X \cap A = \text{the_inv_into } A f^{-1} X$
using f **by** (*auto simp: bij_betw_def subset_image_iff image_iff the_inv_into_f_f*
intro: the_inv_into_f_f[symmetric])
moreover have $\text{inj_on } (\text{the_inv_into } A f) B$
using $X f$ **by** (*auto simp: bij_betw_def inj_on_the_inv_into*)
with X **have** $\text{inj_on } (\text{the_inv_into } A f) X$
by (*auto intro: subset_inj_on*)
ultimately show $\text{emeasure } (\text{distr } (\text{count_space } A) (\text{count_space } B) f) X =$
 $\text{emeasure } (\text{count_space } B) X$
using f **unfolding** $\text{emeasure_distr}[OF f' X]$
by (*subst (1 2) emeasure_count_space*) (*auto simp: card_image dest: fi-*
nite_imageD)
qed *simp*

lemma $\text{emeasure_count_space_finite}[simp]:$
 $X \subseteq A \implies \text{finite } X \implies \text{emeasure } (\text{count_space } A) X = \text{of_nat } (\text{card } X)$
using $\text{emeasure_count_space}[of X A]$ **by** *simp*

lemma $\text{emeasure_count_space_infinite}[simp]:$
 $X \subseteq A \implies \text{infinite } X \implies \text{emeasure } (\text{count_space } A) X = \infty$
using $\text{emeasure_count_space}[of X A]$ **by** *simp*

lemma $\text{measure_count_space}: \text{measure } (\text{count_space } A) X = (\text{if } X \subseteq A \text{ then}$
 $\text{of_nat } (\text{card } X) \text{ else } 0)$
by (*cases finite X*) (*auto simp: measure_notin_sets ennreal_of_nat_eq_real_of_nat*
 $\text{measure_zero_top measure_eq_emeasure_eq_ennreal}$)

lemma $\text{emeasure_count_space_eq_0}:$
 $\text{emeasure } (\text{count_space } A) X = 0 \iff (X \subseteq A \implies X = \{\})$

proof *cases*
assume $X: X \subseteq A$
then show *?thesis*
proof (*intro iffI impI*)
assume $\text{emeasure } (\text{count_space } A) X = 0$
with X **show** $X = \{\}$
by (*subst (asm) emeasure_count_space*) (*auto split: if_split_asm*)
qed *simp*
qed (*simp add: emeasure_notin_sets*)

lemma $\text{null_sets_count_space}: \text{null_sets } (\text{count_space } A) = \{ \{\} \}$
unfolding null_sets_def **by** (*auto simp add: emeasure_count_space_eq_0*)

lemma $\text{AE_count_space}: (\text{AE } x \text{ in } \text{count_space } A. P x) \iff (\forall x \in A. P x)$
unfolding $\text{eventually_ae_filter}$ **by** (*auto simp add: null_sets_count_space*)

lemma $\text{sigma_finite_measure_count_space_countable}:$
assumes $A: \text{countable } A$
shows $\text{sigma_finite_measure } (\text{count_space } A)$

proof qed (*insert A, auto intro!*: exI [of _ ($\lambda a. \{a\}$) ' A])

lemma *sigma_finite_measure_count_space*:

fixes $A :: 'a::countable\ set$ **shows** *sigma_finite_measure* (*count_space* A)
by (*rule sigma_finite_measure_count_space_countable*) *auto*

lemma *finite_measure_count_space*:

assumes [*simp*]: *finite* A
shows *finite_measure* (*count_space* A)
by *rule simp*

lemma *sigma_finite_measure_count_space_finite*:

assumes A: *finite* A **shows** *sigma_finite_measure* (*count_space* A)

proof –

interpret *finite_measure* *count_space* A **using** A **by** (*rule finite_measure_count_space*)
show *sigma_finite_measure* (*count_space* A) ..

qed

7.3.14 Measure restricted to space

lemma *emeasure_restrict_space*:

assumes $\Omega \cap space\ M \in sets\ M\ A \subseteq \Omega$
shows *emeasure* (*restrict_space* M Ω) A = *emeasure* M A

proof (*cases* A $\in sets\ M$)

case *True*

show ?*thesis*

proof (*rule emeasure_measure_of*[*OF restrict_space_def*])

show $(\cap) \Omega \text{ ' } sets\ M \subseteq Pow\ (\Omega \cap space\ M)\ A \in sets\ (restrict_space\ M\ \Omega)$

using $\langle A \subseteq \Omega \rangle \langle A \in sets\ M \rangle sets.space_closed$ **by** (*auto simp: sets_restrict_space*)

show *positive* (*sets* (*restrict_space* M Ω)) (*emeasure* M)

by (*auto simp: positive_def*)

show *countably_additive* (*sets* (*restrict_space* M Ω)) (*emeasure* M)

proof (*rule countably_additiveI*)

fix A :: *nat* \Rightarrow _ **assume** $range\ A \subseteq sets\ (restrict_space\ M\ \Omega)\ disjoint_family$

A

with *assms* **have** $\bigwedge i. A\ i \in sets\ M \wedge i. A\ i \subseteq space\ M\ disjoint_family\ A$

by (*fastforce simp: sets_restrict_space_iff*[*OF assms(1)*] *image_subset_iff*
dest: sets.sets_into_space)+

then **show** $(\sum\ i. emeasure\ M\ (A\ i)) = emeasure\ M\ (\bigcup\ i. A\ i)$

by (*subst suminf_emeasure*) (*auto simp: disjoint_family_subset*)

qed

qed

next

case *False*

with *assms* **have** $A \notin sets\ (restrict_space\ M\ \Omega)$

by (*simp add: sets_restrict_space_iff*)

with *False* **show** ?*thesis*

by (*simp add: emeasure_notin_sets*)

qed

lemma *measure_restrict_space*:

assumes $\Omega \cap \text{space } M \in \text{sets } M$ $A \subseteq \Omega$

shows $\text{measure } (\text{restrict_space } M \ \Omega) \ A = \text{measure } M \ A$

using *emeasure_restrict_space[OF assms]* **by** (*simp add: measure_def*)

lemma *AE_restrict_space_iff*:

assumes $\Omega \cap \text{space } M \in \text{sets } M$

shows $(AE \ x \ \text{in } \text{restrict_space } M \ \Omega. \ P \ x) \longleftrightarrow (AE \ x \ \text{in } M. \ x \in \Omega \longrightarrow P \ x)$

proof –

have *ex_cong*: $\bigwedge P \ Q \ f. (\bigwedge x. P \ x \implies Q \ x) \implies (\bigwedge x. Q \ x \implies P \ (f \ x)) \implies (\exists x. P \ x) \longleftrightarrow (\exists x. Q \ x)$

by *auto*

{ fix } X **assume $X \in \text{sets } M$ $\text{emeasure } M \ X = 0$**

then have $\text{emeasure } M \ (\Omega \cap \text{space } M \cap X) \leq \text{emeasure } M \ X$

by (*intro emeasure_mono*) *auto*

then have $\text{emeasure } M \ (\Omega \cap \text{space } M \cap X) = 0$

using } X **by (*auto intro!: antisym*) }**

with *assms* **show** *?thesis*

unfolding *eventually_ae_filter*

by (*auto simp add: space_restrict_space null_sets_def sets_restrict_space_iff*

emeasure_restrict_space cong: conj_cong

intro!: ex_cong[**where** $f = \lambda X. (\Omega \cap \text{space } M) \cap X$])

qed

lemma *restrict_restrict_space*:

assumes $A \cap \text{space } M \in \text{sets } M$ $B \cap \text{space } M \in \text{sets } M$

shows $\text{restrict_space } (\text{restrict_space } M \ A) \ B = \text{restrict_space } M \ (A \cap B)$ (**is ?l = ?r**)

proof (*rule measure_eqI[symmetric]*)

show *sets ?r = sets ?l*

unfolding *sets_restrict_space image_comp* **by** (*intro image_cong*) *auto*

next

fix } X **assume $X \in \text{sets } (\text{restrict_space } M \ (A \cap B))$**

then obtain } Y **where $Y \in \text{sets } M \ X = Y \cap A \cap B$**

by (*auto simp: sets_restrict_space*)

with *assms* *sets.Int*[*OF assms*] **show** $\text{emeasure } ?r \ X = \text{emeasure } ?l \ X$

by (*subst (1 2) emeasure_restrict_space*)

(*auto simp: space_restrict_space sets_restrict_space_iff emeasure_restrict_space ac_simps*)

qed

lemma *restrict_count_space*: $\text{restrict_space } (\text{count_space } B) \ A = \text{count_space } (A \cap B)$

proof (*rule measure_eqI*)

show *sets* $(\text{restrict_space } (\text{count_space } B) \ A) = \text{sets } (\text{count_space } (A \cap B))$

by (*subst sets_restrict_space*) *auto*

moreover fix } X **assume $X \in \text{sets } (\text{restrict_space } (\text{count_space } B) \ A)$**

ultimately have $X \subseteq A \cap B$ **by** *auto*

then show $\text{emeasure } (\text{restrict_space } (\text{count_space } B) A) X = \text{emeasure } (\text{count_space } (A \cap B)) X$
by (*cases finite X*) (*auto simp add: emeasure_restrict_space*)
qed

lemma *sigma_finite_measure_restrict_space*:

assumes *sigma_finite_measure M*

and *A: A ∈ sets M*

shows $\text{sigma_finite_measure } (\text{restrict_space } M A)$

proof –

interpret *sigma_finite_measure M by fact*

from *sigma_finite_countable* **obtain** *C*

where *C: countable C C ⊆ sets M (⋃ C) = space M ∀ a ∈ C. emeasure M a ≠*

∞

by *blast*

let $?C = (\cap) A \text{ ' } C$

from *C* **have** *countable ?C ?C ⊆ sets (restrict_space M A) (⋃ ?C) = space (restrict_space M A)*

by (*auto simp add: sets_restrict_space space_restrict_space*)

moreover {

fix *a*

assume $a \in ?C$

then obtain *a'* **where** $a = A \cap a' \ a' \in C \ ..$

then have $\text{emeasure } (\text{restrict_space } M A) a \leq \text{emeasure } M a'$

using *A C* **by** (*auto simp add: emeasure_restrict_space intro: emeasure_mono*)

also have $\dots < \infty$ **using** *C(4)* [*rule_format, of a'*] $\langle a' \in C \rangle$ **by** (*simp add: less_top*)

finally have $\text{emeasure } (\text{restrict_space } M A) a \neq \infty$ **by** *simp* }

ultimately show *?thesis*

by (*unfold_locales (rule exI conjI|assumption|blast)*)+

qed

lemma *finite_measure_restrict_space*:

assumes *finite_measure M*

and *A: A ∈ sets M*

shows $\text{finite_measure } (\text{restrict_space } M A)$

proof –

interpret *finite_measure M by fact*

show *?thesis*

by (*rule finite_measureI*) (*simp add: emeasure_restrict_space A space_restrict_space*)

qed

lemma *restrict_distr*:

assumes [*measurable*]: $f \in \text{measurable } M N$

assumes [*simp*]: $\Omega \cap \text{space } N \in \text{sets } N$ **and** *restrict*: $f \in \text{space } M \rightarrow \Omega$

shows $\text{restrict_space } (\text{distr } M N f) \Omega = \text{distr } M (\text{restrict_space } N \Omega) f$

(**is** $?l = ?r$)

proof (*rule measure_eqI*)

fix *A* **assume** $A \in \text{sets } (\text{restrict_space } (\text{distr } M N f) \Omega)$


```

with restrict show emeasure ?l A = emeasure ?r A
by (subst emeasure_distr)
    (auto simp: sets_restrict_space_iff emeasure_restrict_space emeasure_distr
      intro!: measurable_restrict_space2)
qed (simp add: sets_restrict_space)

lemma measure_eqI_restrict_generator:
assumes E: Int_stable E E ⊆ Pow Ω ∧ X. X ∈ E ⇒ emeasure M X = emeasure
N X
assumes sets_eq: sets M = sets N and Ω: Ω ∈ sets M
assumes sets (restrict_space M Ω) = sigma_sets Ω E
assumes sets (restrict_space N Ω) = sigma_sets Ω E
assumes ae: AE x in M. x ∈ Ω AE x in N. x ∈ Ω
assumes A: countable A A ≠ {} A ⊆ E ∪ A = Ω ∧ a. a ∈ A ⇒ emeasure M
a ≠ ∞
shows M = N
proof (rule measure_eqI)
fix X assume X: X ∈ sets M
then have emeasure M X = emeasure (restrict_space M Ω) (X ∩ Ω)
using ae Ω by (auto simp add: emeasure_restrict_space intro!: emeasure_eq_AE)
also have restrict_space M Ω = restrict_space N Ω
proof (rule measure_eqI_generator_eq)
fix X assume X ∈ E
then show emeasure (restrict_space M Ω) X = emeasure (restrict_space N
Ω) X
using E Ω by (subst (1 2) emeasure_restrict_space) (auto simp: sets_eq
sets_eq[THEN sets_eq_imp_space_eq])
next
show range (from_nat_into A) ⊆ E (∪ i. from_nat_into A i) = Ω
using A by (auto cong del: SUP_cong_simp)
next
fix i
have emeasure (restrict_space M Ω) (from_nat_into A i) = emeasure M
(from_nat_into A i)
using A Ω by (subst emeasure_restrict_space)
    (auto simp: sets_eq sets_eq[THEN sets_eq_imp_space_eq] intro:
from_nat_into)
with A show emeasure (restrict_space M Ω) (from_nat_into A i) ≠ ∞
by (auto intro: from_nat_into)
qed fact+
also have emeasure (restrict_space N Ω) (X ∩ Ω) = emeasure N X
using X ae Ω by (auto simp add: emeasure_restrict_space sets_eq intro!:
emeasure_eq_AE)
finally show emeasure M X = emeasure N X .
qed fact

```

7.3.15 Null measure

definition null_measure :: 'a measure ⇒ 'a measure **where**

2150

$\text{null_measure } M = \text{sigma } (\text{space } M) (\text{sets } M)$

lemma $\text{space_null_measure}$ [simp]: $\text{space } (\text{null_measure } M) = \text{space } M$
by (simp add: null_measure_def)

lemma sets_null_measure [simp, measurable_cong]: $\text{sets } (\text{null_measure } M) = \text{sets } M$
by (simp add: null_measure_def)

lemma $\text{emeasure_null_measure}$ [simp]: $\text{emeasure } (\text{null_measure } M) X = 0$
by (cases $X \in \text{sets } M$, rule $\text{emeasure_measure_of}$)
(auto simp: positive_def countably_additive_def $\text{emeasure_notin_sets } \text{null_measure_def}$
dest: sets.sets_into_space)

lemma $\text{measure_null_measure}$ [simp]: $\text{measure } (\text{null_measure } M) X = 0$
by (intro $\text{measure_eq_emeasure_eq_ennreal}$) auto

lemma null_measure_idem [simp]: $\text{null_measure } (\text{null_measure } M) = \text{null_measure } M$
by(rule measure_eqI) simp_all

7.3.16 Scaling a measure

definition $\text{scale_measure} :: \text{ennreal} \Rightarrow 'a \text{ measure} \Rightarrow 'a \text{ measure}$ **where**
 $\text{scale_measure } r M = \text{measure_of } (\text{space } M) (\text{sets } M) (\lambda A. r * \text{emeasure } M A)$

lemma $\text{space_scale_measure}$: $\text{space } (\text{scale_measure } r M) = \text{space } M$
by (simp add: scale_measure_def)

lemma $\text{sets_scale_measure}$ [simp, measurable_cong]: $\text{sets } (\text{scale_measure } r M) = \text{sets } M$
by (simp add: scale_measure_def)

lemma $\text{emeasure_scale_measure}$ [simp]:
 $\text{emeasure } (\text{scale_measure } r M) A = r * \text{emeasure } M A$
(is $_ = ?\mu A$)

proof(cases $A \in \text{sets } M$)

case True

show *?thesis unfolding scale_measure_def*

proof(rule $\text{emeasure_measure_of_sigma}$)

show $\text{sigma_algebra } (\text{space } M) (\text{sets } M) ..$

show $\text{positive } (\text{sets } M) ?\mu$ by (simp add: positive_def)

show $\text{countably_additive } (\text{sets } M) ?\mu$

proof (rule $\text{countably_additiveI}$)

fix $A :: \text{nat} \Rightarrow _$ **assume** *: $\text{range } A \subseteq \text{sets } M$ $\text{disjoint_family } A$

have $(\sum i. ?\mu (A i)) = r * (\sum i. \text{emeasure } M (A i))$

by simp

also have $\dots = ?\mu (\bigcup i. A i)$ **using** * **by**(simp add: suminf_emeasure)

finally show $(\sum i. ?\mu (A i)) = ?\mu (\bigcup i. A i)$.

qed
qed(fact True)
qed(simp add: emeasure_notin_sets)

lemma *scale_measure_1* [simp]: $scale_measure\ 1\ M = M$
by(rule *measure_eqI*) *simp_all*

lemma *scale_measure_0*[simp]: $scale_measure\ 0\ M = null_measure\ M$
by(rule *measure_eqI*) *simp_all*

lemma *measure_scale_measure* [simp]: $0 \leq r \implies measure\ (scale_measure\ r\ M)$
 $A = r * measure\ M\ A$
using *emeasure_scale_measure*[of $r\ M\ A$]
emeasure_eq_ennreal_measure[of $M\ A$]
measure_eq_emeasure_eq_ennreal[of $scale_measure\ r\ M\ A$]
by (*cases* *emeasure* ($scale_measure\ r\ M$) $A = top$)
(*auto simp del: emeasure_scale_measure*
simp: ennreal_top_eq_mult_iff ennreal_mult_eq_top_iff measure_zero_top
ennreal_mult[symmetric])

lemma *scale_scale_measure* [simp]:
 $scale_measure\ r\ (scale_measure\ r'\ M) = scale_measure\ (r * r')\ M$
by (rule *measure_eqI*) (*simp_all add: max_def mult.assoc*)

lemma *scale_null_measure* [simp]: $scale_measure\ r\ (null_measure\ M) = null_measure\ M$
by (rule *measure_eqI*) *simp_all*

7.3.17 Complete lattice structure on measures

lemma (in *finite_measure*) *finite_measure_Diff'*:
 $A \in sets\ M \implies B \in sets\ M \implies measure\ M\ (A - B) = measure\ M\ A - measure\ M\ (A \cap B)$
using *finite_measure_Diff*[of $A\ A \cap B$] **by** (*auto simp: Diff_Int*)

lemma (in *finite_measure*) *finite_measure_Union'*:
 $A \in sets\ M \implies B \in sets\ M \implies measure\ M\ (A \cup B) = measure\ M\ A + measure\ M\ (B - A)$
using *finite_measure_Union*[of $A\ B - A$] **by** *auto*

lemma *finite_unsigned_Hahn_decomposition*:
assumes *finite_measure* M *finite_measure* N **and** [simp]: $sets\ N = sets\ M$
shows $\exists Y \in sets\ M. (\forall X \in sets\ M. X \subseteq Y \implies N\ X \leq M\ X) \wedge (\forall X \in sets\ M. X \cap Y = \{\} \implies M\ X \leq N\ X)$

proof –
interpret M : *finite_measure* M **by** *fact*
interpret N : *finite_measure* N **by** *fact*

define d **where** $d\ X = measure\ M\ X - measure\ N\ X$ **for** X

```

have [intro]: bdd_above (d'sets M)
  using sets.sets_into_space[of _ M]
  by (intro bdd_aboveI[where M=measure M (space M)])
  (auto simp: d_def field_simps subset_eq intro!: add_increasing M.finite_measure_mono)

define  $\gamma$  where  $\gamma = (\text{SUP } X \in \text{sets } M. d X)$ 
have le_ $\gamma$ [intro]:  $X \in \text{sets } M \implies d X \leq \gamma$  for  $X$ 
  by (auto simp:  $\gamma$ _def intro!: cSUP_upper)

have  $\exists f. \forall n. f n \in \text{sets } M \wedge d (f n) > \gamma - 1 / 2^n$ 
proof (intro choice_iff[THEN iffD1] allI)
  fix  $n$ 
  have  $\exists X \in \text{sets } M. \gamma - 1 / 2^n < d X$ 
  unfolding  $\gamma$ _def by (intro less_cSUP_iff[THEN iffD1]) auto
  then show  $\exists y. y \in \text{sets } M \wedge \gamma - 1 / 2^n < d y$ 
  by auto
qed
then obtain  $E$  where [measurable]:  $E n \in \text{sets } M$  and  $E$ :  $d (E n) > \gamma - 1 / 2^n$  for  $n$ 
  by auto

define  $F$  where  $F m n = (\text{if } m \leq n \text{ then } \bigcap_{i \in \{m..n\}}. E i \text{ else space } M)$  for  $m$ 
 $n$ 

have [measurable]:  $m \leq n \implies F m n \in \text{sets } M$  for  $m n$ 
  by (auto simp: F_def)

have 1:  $\gamma - 2 / 2^m + 1 / 2^n \leq d (F m n)$  if  $m \leq n$  for  $m n$ 
  using that
proof (induct rule: dec_induct)
  case base with  $E$ [of  $m$ ] show ?case
  by (simp add: F_def field_simps)
next
  case (step  $i$ )
  have  $F\_Suc$ :  $F m (Suc i) = F m i \cap E (Suc i)$ 
  using  $\langle m \leq i \rangle$  by (auto simp: F_def le_Suc_eq)

  have  $\gamma + (\gamma - 2 / 2^m + 1 / 2^{Suc i}) \leq (\gamma - 1 / 2^{Suc i}) + (\gamma - 2 / 2^m + 1 / 2^i)$ 
  by (simp add: field_simps)
  also have  $\dots \leq d (E (Suc i)) + d (F m i)$ 
  using  $E$ [of  $Suc i$ ] by (intro add_mono step) auto
  also have  $\dots = d (E (Suc i)) + d (F m i - E (Suc i)) + d (F m (Suc i))$ 
  using  $\langle m \leq i \rangle$  by (simp add: d_def field_simps F_Suc M.finite_measure_Diff'
    N.finite_measure_Diff')
  also have  $\dots = d (E (Suc i) \cup F m i) + d (F m (Suc i))$ 
  using  $\langle m \leq i \rangle$  by (simp add: d_def field_simps M.finite_measure_Union'
    N.finite_measure_Union')

```

```

    also have ... ≤ γ + d (F m (Suc i))
      using ⟨m ≤ i⟩ by auto
    finally show ?case
      by auto
  qed

define F' where F' m = (⋂ i∈{m..}. E i) for m
have F'_eq: F' m = (⋂ i. F m (i + m)) for m
  by (fastforce simp: le_iff_add[of m] F'_def F_def)

have [measurable]: F' m ∈ sets M for m
  by (auto simp: F'_def)

have γ_le: γ - 0 ≤ d (⋃ m. F' m)
proof (rule LIMSEQ_le)
  show (λn. γ - 2 / 2 ^ n) → γ - 0
    by (intro tendsto_intros LIMSEQ_divide_realpow_zero) auto
  have incseq F'
    by (auto simp: incseq_def F'_def)
  then show (λm. d (F' m)) → d (⋃ m. F' m)
    unfolding d_def
    by (intro tendsto_diff M.finite_Lim_measure_incseq N.finite_Lim_measure_incseq)
  auto

  have γ - 2 / 2 ^ m + 0 ≤ d (F' m) for m
  proof (rule LIMSEQ_le)
    have *: decseq (λn. F m (n + m))
      by (auto simp: decseq_def F_def)
    show (λn. d (F m n)) → d (F' m)
      unfolding d_def F'_eq
      by (rule LIMSEQ_offset[where k=m])
      (auto intro!: tendsto_diff M.finite_Lim_measure_decseq N.finite_Lim_measure_decseq
    *)
    show (λn. γ - 2 / 2 ^ m + 1 / 2 ^ n) → γ - 2 / 2 ^ m + 0
      by (intro tendsto_add LIMSEQ_divide_realpow_zero tendsto_const) auto
    show ∃ N. ∀ n ≥ N. γ - 2 / 2 ^ m + 1 / 2 ^ n ≤ d (F m n)
      using 1[of m] by (intro exI[of _ m]) auto
  qed
  then show ∃ N. ∀ n ≥ N. γ - 2 / 2 ^ n ≤ d (F' n)
    by auto
  qed

show ?thesis
proof (safe intro!: bexI[of _ ⋃ m. F' m])
  fix X assume [measurable]: X ∈ sets M and X: X ⊆ (⋃ m. F' m)
  have d (⋃ m. F' m) - d X = d ((⋃ m. F' m) - X)
  using X by (auto simp: d_def M.finite_measure_Diff N.finite_measure_Diff)
  also have ... ≤ γ
    by auto

```

```

finally have  $0 \leq d X$ 
  using  $\gamma\_le$  by auto
then show  $emeasure N X \leq emeasure M X$ 
  by (auto simp: d_def M.emeasure_eq_measure N.emeasure_eq_measure)
next
fix  $X$  assume [measurable]:  $X \in sets M$  and  $X: X \cap (\bigcup m. F' m) = \{\}$ 
then have  $d (\bigcup m. F' m) + d X = d (X \cup (\bigcup m. F' m))$ 
  by (auto simp: d_def M.finite_measure_Union N.finite_measure_Union)
also have  $\dots \leq \gamma$ 
  by auto
finally have  $d X \leq 0$ 
  using  $\gamma\_le$  by auto
then show  $emeasure M X \leq emeasure N X$ 
  by (auto simp: d_def M.emeasure_eq_measure N.emeasure_eq_measure)
qed auto
qed

```

proposition *unsigned_Hahn_decomposition*:

```

assumes [simp]:  $sets N = sets M$  and [measurable]:  $A \in sets M$ 
  and [simp]:  $emeasure M A \neq top$   $emeasure N A \neq top$ 
shows  $\exists Y \in sets M. Y \subseteq A \wedge (\forall X \in sets M. X \subseteq Y \longrightarrow NX \leq MX) \wedge (\forall X \in sets M. X \subseteq A \longrightarrow X \cap Y = \{\} \longrightarrow MX \leq NX)$ 
proof -
  have  $\exists Y \in sets (restrict\_space M A).$ 
     $(\forall X \in sets (restrict\_space M A). X \subseteq Y \longrightarrow (restrict\_space N A) X \leq (restrict\_space M A) X) \wedge$ 
     $(\forall X \in sets (restrict\_space M A). X \cap Y = \{\} \longrightarrow (restrict\_space M A) X \leq (restrict\_space N A) X)$ 
  proof (rule finite_unsigned_Hahn_decomposition)
    show  $finite\_measure (restrict\_space M A) finite\_measure (restrict\_space N A)$ 
    by (auto simp: space_restrict_space emeasure_restrict_space less_top intro!: finite_measureI)
  qed (simp add: sets_restrict_space)
  with assms show ?thesis
  by (metis Int_subset_iff emeasure_restrict_space sets.Int_space_eq2 sets_restrict_space_iff space_restrict_space)
qed

```

Define a lexicographical order on *measure*, in the order space, sets and measure. The parts of the lexicographical order are point-wise ordered.

```

instantiation measure :: (type) order_bot
begin

```

```

inductive less_eq_measure :: 'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  bool where
   $space M \subset space N \Longrightarrow less\_eq\_measure M N$ 
|  $space M = space N \Longrightarrow sets M \subset sets N \Longrightarrow less\_eq\_measure M N$ 
|  $space M = space N \Longrightarrow sets M = sets N \Longrightarrow emeasure M \leq emeasure N \Longrightarrow less\_eq\_measure M N$ 

```

lemma *le_measure_iff*:

$M \leq N \longleftrightarrow$ (if space $M =$ space N then
if sets $M =$ sets N then $\text{emeasure } M \leq \text{emeasure } N$ else sets $M \subseteq$ sets N else
space $M \subseteq$ space N)
by (auto elim: less_eq_measure.cases intro: less_eq_measure.intros)

definition *less_measure* :: 'a measure \Rightarrow 'a measure \Rightarrow bool **where**

$\text{less_measure } M N \longleftrightarrow (M \leq N \wedge \neg N \leq M)$

definition *bot_measure* :: 'a measure **where**

$\text{bot_measure} = \text{sigma } \{\} \{\}$

lemma

shows *space_bot[simp]*: space bot = $\{\}$
and *sets_bot[simp]*: sets bot = $\{\{\}\}$
and *emeasure_bot[simp]*: $\text{emeasure bot } X = 0$
by (auto simp: bot_measure_def sigma_sets_empty_eq emeasure_sigma)

instance

proof standard

show bot \leq a **for** a :: 'a measure

by (simp add: le_measure_iff bot_measure_def sigma_sets_empty_eq emeasure_sigma le_fun_def)

qed (auto simp: le_measure_iff less_measure_def split: if_split_asm intro: measure_eqI)

end

proposition *le_measure*: sets $M =$ sets $N \implies M \leq N \longleftrightarrow (\forall A \in \text{sets } M. \text{emeasure } M A \leq \text{emeasure } N A)$

by (metis emeasure_neq_0_sets le_fun_def le_measure_iff order_class.order_eq_iff sets_eq_imp_space_eq)

definition *sup_measure'* :: 'a measure \Rightarrow 'a measure \Rightarrow 'a measure **where**

$\text{sup_measure}' A B =$
 $\text{measure_of } (\text{space } A) (\text{sets } A)$
 $(\lambda X. \text{SUP } Y \in \text{sets } A. \text{emeasure } A (X \cap Y) + \text{emeasure } B (X \cap - Y))$

lemma *assumes [simp]*: sets $B =$ sets A

shows *space_sup_measure'[simp]*: space (sup_measure' $A B$) = space A

and *sets_sup_measure'[simp]*: sets (sup_measure' $A B$) = sets A

using sets_eq_imp_space_eq[OF assms] **by** (simp_all add: sup_measure'_def)

lemma *emeasure_sup_measure'*:

assumes sets_eq[simp]: sets $B =$ sets A **and** [simp, intro]: $X \in \text{sets } A$

shows $\text{emeasure } (\text{sup_measure}' A B) X = (\text{SUP } Y \in \text{sets } A. \text{emeasure } A (X \cap Y) + \text{emeasure } B (X \cap - Y))$

(is $_ = ?S X$)

proof -

```

note sets_eq_imp_space_eq[OF sets_eq, simp]
show ?thesis
  using sup_measure'_def
proof (rule emeasure_measure_of)
  let ?d =  $\lambda X Y. \text{emeasure } A (X \cap Y) + \text{emeasure } B (X \cap - Y)$ 
  show countably_additive (sets (sup_measure' A B)) ( $\lambda X. \text{SUP } Y \in \text{sets } A. \text{emeasure } A (X \cap Y) + \text{emeasure } B (X \cap - Y)$ )
proof (rule countably_additiveI, goal_cases)
  case (1 X)
  then have [measurable]:  $\bigwedge i. X i \in \text{sets } A$  and disjoint_family X
  by auto
  have disjoint: disjoint_family ( $\lambda i. X i \cap Y$ ) disjoint_family ( $\lambda i. X i - Y$ )
for Y
  using 1(2) disjoint_family_subset by fastforce+
  have ( $\sum i. ?S (X i) = (\text{SUP } Y \in \text{sets } A. \sum i. ?d (X i) Y)$ )
proof (rule ennreal_suminf_SUP_eq_directed)
  fix J :: nat set and a b assume finite J and [measurable]:  $a \in \text{sets } A$   $b \in \text{sets } A$ 
  have  $\exists c \in \text{sets } A. c \subseteq X i \wedge (\forall a \in \text{sets } A. ?d (X i) a \leq ?d (X i) c)$  for i
  proof cases
  assume emeasure A (X i) = top  $\vee$  emeasure B (X i) = top
  then show ?thesis
  by force
  next
  assume finite:  $\neg (\text{emeasure } A (X i) = \text{top} \vee \text{emeasure } B (X i) = \text{top})$ 
  then have  $\exists Y \in \text{sets } A. Y \subseteq X i \wedge (\forall C \in \text{sets } A. C \subseteq Y \longrightarrow B C \leq A C) \wedge (\forall C \in \text{sets } A. C \subseteq X i \longrightarrow C \cap Y = \{\} \longrightarrow A C \leq B C)$ 
  using unsigned_Hahn_decomposition[of B A X i] by simp
  then obtain Y where [measurable]:  $Y \in \text{sets } A$  and [simp]:  $Y \subseteq X i$ 
  and B_le_A:  $\bigwedge C. C \in \text{sets } A \Longrightarrow C \subseteq Y \Longrightarrow B C \leq A C$ 
  and A_le_B:  $\bigwedge C. C \in \text{sets } A \Longrightarrow C \subseteq X i \Longrightarrow C \cap Y = \{\} \Longrightarrow A C \leq B C$ 
  by auto

  show ?thesis
proof (intro bexI ballI conjI)
  fix a assume [measurable]:  $a \in \text{sets } A$ 
  have *:  $(X i \cap a \cap Y \cup (X i \cap a - Y)) = X i \cap a (X i - a) \cap Y \cup (X i - a - Y) = X i \cap - a$ 
  for a Y by auto
  then have ?d (X i) a =
     $(A (X i \cap a \cap Y) + A (X i \cap a \cap - Y)) + (B (X i \cap - a \cap Y) + B (X i \cap - a \cap - Y))$ 
  by (subst (1 2) plus_emeasure) (auto simp: Diff_eq[symmetric])
  also have ...  $\leq (A (X i \cap a \cap Y) + B (X i \cap a \cap - Y)) + (A (X i \cap - a \cap Y) + B (X i \cap - a \cap - Y))$ 
  by (intro add_mono order_refl B_le_A A_le_B) (auto simp: Diff_eq[symmetric])
  also have ...  $\leq (A (X i \cap Y \cap a) + A (X i \cap Y \cap - a)) + (B (X i \cap$ 

```



```

- Y ∩ a) + B (X i ∩ - Y ∩ - a))
  by (simp add: ac_simps)
  also have ... ≤ A (X i ∩ Y) + B (X i ∩ - Y)
    by (subst (1 2) plus_emeasure) (auto simp: Diff_eq[symmetric] *)
  finally show ?d (X i) a ≤ ?d (X i) Y .
qed auto
qed
then obtain C where [measurable]: C i ∈ sets A and C i ⊆ X i
  and C: ⋀a. a ∈ sets A ⇒ ?d (X i) a ≤ ?d (X i) (C i) for i
  by metis
have *: X i ∩ (⋃i. C i) = X i ∩ C i for i
  using ‹disjoint_family X› ‹⋀i. C i ⊆ X i›
  by (simp add: disjoint_family_on_def disjoint_iff_not_equal set_eq_iff)
(metis subsetD)
then have **: X i ∩ - (⋃i. C i) = X i ∩ - C i for i by blast
moreover have (⋃i. C i) ∈ sets A
  by fastforce
ultimately show ∃c∈sets A. ∀i∈J. ?d (X i) a ≤ ?d (X i) c ∧ ?d (X i) b
≤ ?d (X i) c
  by (metis * C ‹a ∈ sets A› ‹b ∈ sets A›)
qed
also have ... = ?S (⋃i. X i)
proof -
  have ⋀Y. Y ∈ sets A ⇒ (∑i. emeasure A (X i ∩ Y) + emeasure B (X
i ∩ - Y))
    = emeasure A (⋃i. X i ∩ Y) + emeasure B (⋃i. X i ∩
- Y)
  using disjoint
  by (auto simp flip: suminf_add Diff_eq simp add: image_subset_iff
suminf_emeasure)
  then show ?thesis by force
qed
finally show (∑i. ?S (X i)) = ?S (⋃i. X i) .
qed
qed (auto dest: sets.sets_into_space simp: positive_def intro!: SUP_const)
qed

```

lemma *le_emeasure_sup_measure'1:*

```

  assumes sets B = sets A X ∈ sets A shows emeasure A X ≤ emeasure
(sup_measure' A B) X
  by (subst emeasure_sup_measure'[OF assms]) (auto intro!: SUP_upper2[of X]
assms)

```

lemma *le_emeasure_sup_measure'2:*

```

  assumes sets B = sets A X ∈ sets A shows emeasure B X ≤ emeasure (sup_measure'
A B) X
  by (subst emeasure_sup_measure'[OF assms]) (auto intro!: SUP_upper2[of {}]
assms)

```

lemma *emeasure_sup_measure'_le2*:
assumes [*simp*]: *sets B = sets C sets A = sets C* **and** [*measurable*]: $X \in \text{sets } C$
assumes $A: \bigwedge Y. Y \subseteq X \implies Y \in \text{sets } A \implies \text{emeasure } A \ Y \leq \text{emeasure } C \ Y$
assumes $B: \bigwedge Y. Y \subseteq X \implies Y \in \text{sets } A \implies \text{emeasure } B \ Y \leq \text{emeasure } C \ Y$
shows $\text{emeasure } (\text{sup_measure}' \ A \ B) \ X \leq \text{emeasure } C \ X$
proof (*subst emeasure_sup_measure'*)
show $(\text{SUP } Y \in \text{sets } A. \text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - \ Y)) \leq \text{emeasure } C \ X$
unfolding $\langle \text{sets } A = \text{sets } C \rangle$
proof (*intro SUP_least*)
fix Y **assume** [*measurable*]: $Y \in \text{sets } C$
have [*simp*]: $X \cap Y \cup (X - Y) = X$
by *auto*
have $\text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - \ Y) \leq \text{emeasure } C \ (X \cap Y) + \text{emeasure } C \ (X \cap - \ Y)$
by (*intro add_mono A B*) (*auto simp: Diff_eq[symmetric]*)
also have $\dots = \text{emeasure } C \ X$
by (*subst plus_emeasure*) (*auto simp: Diff_eq[symmetric]*)
finally show $\text{emeasure } A \ (X \cap Y) + \text{emeasure } B \ (X \cap - \ Y) \leq \text{emeasure } C \ X$.
qed
qed *simp_all*

definition *sup_lexord* :: $'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b::\text{order}) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 $\text{sup_lexord } A \ B \ k \ s \ c =$
(if $k \ A = k \ B$ *then* c *else*
if $\neg k \ A \leq k \ B \wedge \neg k \ B \leq k \ A$ *then* s *else*
if $k \ B \leq k \ A$ *then* A *else* B)

lemma *sup_lexord*:
 $(k \ A < k \ B \implies P \ B) \implies (k \ B < k \ A \implies P \ A) \implies (k \ A = k \ B \implies P \ c) \implies$
 $(\neg k \ B \leq k \ A \implies \neg k \ A \leq k \ B \implies P \ s) \implies P \ (\text{sup_lexord } A \ B \ k \ s \ c)$
by (*auto simp: sup_lexord_def*)

lemmas *le_sup_lexord* = *sup_lexord*[**where** $P = \lambda a. c \leq a$ **for** c]

lemma *sup_lexord1*: $k \ A = k \ B \implies \text{sup_lexord } A \ B \ k \ s \ c = c$
by (*simp add: sup_lexord_def*)

lemma *sup_lexord_commute*: $\text{sup_lexord } A \ B \ k \ s \ c = \text{sup_lexord } B \ A \ k \ s \ c$
by (*auto simp: sup_lexord_def*)

lemma *sigma_sets_le_sets_iff*: $(\text{sigma_sets } (\text{space } x) \ \mathcal{A} \subseteq \text{sets } x) = (\mathcal{A} \subseteq \text{sets } x)$
using *sets.sigma_sets_subset*[*of* $\mathcal{A} \ x$] **by** *auto*

lemma *sigma_le_iff*: $\mathcal{A} \subseteq \text{Pow } \Omega \implies \text{sigma } \Omega \ \mathcal{A} \leq x \longleftrightarrow (\Omega \subseteq \text{space } x \wedge (\text{space } x = \Omega \longrightarrow \mathcal{A} \subseteq \text{sets } x))$
by (*cases* $\Omega = \text{space } x$)

```

      (simp_all add: eq_commute[of _ sets x] le_measure_iff emeasure_sigma
le_fun_def
      sigma_sets_superset_generator sigma_sets_le_sets_iff)

```

```

instantiation measure :: (type) semilattice_sup
begin

```

```

definition sup_measure :: 'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  'a measure where
  sup_measure A B =
    sup_lexord A B space (sigma (space A  $\cup$  space B) {})
    (sup_lexord A B sets (sigma (space A) (sets A  $\cup$  sets B)) (sup_measure' A
B))

```

```

instance

```

```

proof

```

```

  fix x y z :: 'a measure

```

```

  show x  $\leq$  sup x y

```

```

    unfolding sup_measure_def

```

```

  proof (intro le_sup_lexord)

```

```

    assume space x = space y

```

```

    then have *: sets x  $\cup$  sets y  $\subseteq$  Pow (space x)

```

```

      using sets.space_closed by auto

```

```

    assume  $\neg$  sets y  $\subseteq$  sets x  $\neg$  sets x  $\subseteq$  sets y

```

```

    then have sets x  $\subset$  sets x  $\cup$  sets y

```

```

      by auto

```

```

    also have ...  $\leq$  sigma (space x) (sets x  $\cup$  sets y)

```

```

      by (subst sets_measure_of[OF *]) (rule sigma_sets_superset_generator)

```

```

    finally show x  $\leq$  sigma (space x) (sets x  $\cup$  sets y)

```

```

      by (simp add: space_measure_of[OF *] less_eq_measure.intros(2))

```

```

  next

```

```

    assume  $\neg$  space y  $\subseteq$  space x  $\neg$  space x  $\subseteq$  space y

```

```

    then show x  $\leq$  sigma (space x  $\cup$  space y) {}

```

```

      by (intro less_eq_measure.intros) auto

```

```

  next

```

```

    assume sets x = sets y then show x  $\leq$  sup_measure' x y

```

```

      by (simp add: le_measure le_emeasure_sup_measure'1)

```

```

  qed (auto intro: less_eq_measure.intros)

```

```

show y  $\leq$  sup x y

```

```

  unfolding sup_measure_def

```

```

proof (intro le_sup_lexord)

```

```

  assume **: space x = space y

```

```

  then have *: sets x  $\cup$  sets y  $\subseteq$  Pow (space y)

```

```

    using sets.space_closed by auto

```

```

  assume  $\neg$  sets y  $\subseteq$  sets x  $\neg$  sets x  $\subseteq$  sets y

```

```

  then have sets y  $\subset$  sets x  $\cup$  sets y

```

```

    by auto

```

```

  also have ...  $\leq$  sigma (space y) (sets x  $\cup$  sets y)

```

```

    by (subst sets_measure_of[OF *]) (rule sigma_sets_superset_generator)

```

```

  finally show y  $\leq$  sigma (space x) (sets x  $\cup$  sets y)

```

```

    by (simp add: ** space_measure_of[OF *] less_eq_measure.intros(2))
  next
    assume  $\neg$  space  $y \subseteq$  space  $x \neg$  space  $x \subseteq$  space  $y$ 
    then show  $y \leq$  sigma (space  $x \cup$  space  $y$ ) {}
      by (intro less_eq_measure.intros) auto
  next
    assume sets  $x =$  sets  $y$  then show  $y \leq$  sup_measure'  $x y$ 
      by (simp add: le_measure le_emeasure_sup_measure'2)
  qed (auto intro: less_eq_measure.intros)
  show  $x \leq y \implies z \leq y \implies$  sup  $x z \leq y$ 
    unfolding sup_measure_def
  proof (intro sup_lexord[where  $P = \lambda x. x \leq y$ ])
    assume  $x \leq y z \leq y$  and [simp]: space  $x =$  space  $z$  sets  $x =$  sets  $z$ 
    from  $\langle x \leq y \rangle$  show sup_measure'  $x z \leq y$ 
  proof cases
    case 1 then show ?thesis
      by (intro less_eq_measure.intros(1)) simp
    next
    case 2 then show ?thesis
      by (intro less_eq_measure.intros(2)) simp_all
    next
    case 3 with  $\langle z \leq y \rangle \langle x \leq y \rangle$  show ?thesis
      by (auto simp add: le_measure intro!: emeasure_sup_measure'_le2)
  qed
  next
    assume **:  $x \leq y z \leq y$  space  $x =$  space  $z \neg$  sets  $z \subseteq$  sets  $x \neg$  sets  $x \subseteq$  sets  $z$ 
    then have *: sets  $x \cup$  sets  $z \subseteq$  Pow (space  $x$ )
      using sets.space_closed by auto
    show sigma (space  $x$ ) (sets  $x \cup$  sets  $z$ )  $\leq y$ 
      unfolding sigma_le_iff[OF *] using ** by (auto simp: le_measure_iff split:
if_split_asm)
  next
    assume  $x \leq y z \leq y \neg$  space  $z \subseteq$  space  $x \neg$  space  $x \subseteq$  space  $z$ 
    then have space  $x \subseteq$  space  $y$  space  $z \subseteq$  space  $y$ 
      by (auto simp: le_measure_iff split: if_split_asm)
    then show sigma (space  $x \cup$  space  $z$ ) {}  $\leq y$ 
      by (simp add: sigma_le_iff)
  qed
  qed
end

lemma space_empty_eq_bot: space  $a =$  {}  $\longleftrightarrow a =$  bot
  using space_empty[of  $a$ ] by (auto intro!: measure_eqI)

lemma sets_eq_iff_bounded:  $A \leq B \implies B \leq C \implies$  sets  $A =$  sets  $C \implies$  sets  $B =$  sets  $A$ 
  by (auto dest: sets_eq_imp_space_eq simp add: le_measure_iff split: if_split_asm)

```

lemma *sets_sup*: $\text{sets } A = \text{sets } M \implies \text{sets } B = \text{sets } M \implies \text{sets } (\text{sup } A \ B) = \text{sets } M$

by (*auto simp add: sup_measure_def sup_lexord_def dest: sets_eq_imp_space_eq*)

lemma *le_measureD1*: $A \leq B \implies \text{space } A \leq \text{space } B$

by (*auto simp: le_measure_iff split: if_split_asm*)

lemma *le_measureD2*: $A \leq B \implies \text{space } A = \text{space } B \implies \text{sets } A \leq \text{sets } B$

by (*auto simp: le_measure_iff split: if_split_asm*)

lemma *le_measureD3*: $A \leq B \implies \text{sets } A = \text{sets } B \implies \text{emeasure } A \ X \leq \text{emeasure } B \ X$

by (*auto simp: le_measure_iff le_fun_def dest: sets_eq_imp_space_eq split: if_split_asm*)

lemma *UN_space_closed*: $\bigcup (\text{sets } 'S) \subseteq \text{Pow } (\bigcup (\text{space } 'S))$

using *sets.space_closed* **by** *auto*

definition

Sup_lexord :: $('a \Rightarrow 'b :: \text{complete_lattice}) \Rightarrow ('a \ \text{set} \Rightarrow 'a) \Rightarrow ('a \ \text{set} \Rightarrow 'a) \Rightarrow 'a \ \text{set} \Rightarrow 'a$

where

Sup_lexord *k c s* *A* =

(*let* *U* = ($\text{SUP } a \in A. k \ a$))

in if $\exists a \in A. k \ a = U$ *then* $c \ \{a \in A. k \ a = U\}$ *else* *s* *A*)

lemma *Sup_lexord*:

$(\bigwedge a \ S. a \in A \implies k \ a = (\text{SUP } a \in A. k \ a) \implies S = \{a' \in A. k \ a' = k \ a\} \implies P \ (c \ S)) \implies ((\bigwedge a. a \in A \implies k \ a \neq (\text{SUP } a \in A. k \ a)) \implies P \ (s \ A)) \implies$

$P \ (\text{Sup_lexord } k \ c \ s \ A)$

by (*auto simp: Sup_lexord_def Let_def*)

lemma *Sup_lexord1*:

assumes *A*: $A \neq \{\}$ $(\bigwedge a. a \in A \implies k \ a = (\bigcup a \in A. k \ a)) \ P \ (c \ A)$

shows $P \ (\text{Sup_lexord } k \ c \ s \ A)$

unfolding *Sup_lexord_def* *Let_def*

proof (*clarsimp, safe*)

show $\forall a \in A. k \ a \neq (\bigcup x \in A. k \ x) \implies P \ (s \ A)$

by (*metis assms(1,2) ex_in_conv*)

next

fix *a* **assume** $a \in A \ k \ a = (\bigcup x \in A. k \ x)$

then have $\{a \in A. k \ a = (\bigcup x \in A. k \ x)\} = \{a \in A. k \ a = k \ a\}$

by (*metis A(2)[symmetric]*)

then show $P \ (c \ \{a \in A. k \ a = (\bigcup x \in A. k \ x)\})$

by (*simp add: A(3)*)

qed

instantiation *measure* :: $(\text{type}) \ \text{complete_lattice}$

begin

interpretation *sup_measure*: *comm_monoid_set sup bot* :: 'a measure
 by *standard* (*auto intro!*: *antisym*)

lemma *sup_measure_F_mono'*:

finite J \implies *finite I* \implies *sup_measure.F id I* \leq *sup_measure.F id (I \cup J)*

proof (*induction J rule: finite_induct*)

case empty then show ?*case*

by *simp*

next

case (*insert i J*)

show ?*case*

proof *cases*

assume *i* \in *I* **with** *insert* **show** ?*thesis*

by (*auto simp: insert_absorb*)

next

assume *i* \notin *I*

have *sup_measure.F id I* \leq *sup_measure.F id (I \cup J)*

by (*intro insert*)

also have $\dots \leq$ *sup_measure.F id (insert i (I \cup J))*

using *insert* $\langle i \notin I \rangle$ **by** (*subst sup_measure.insert*) *auto*

finally show ?*thesis*

by *auto*

qed

qed

lemma *sup_measure_F_mono*: *finite I* \implies *J* \subseteq *I* \implies *sup_measure.F id J* \leq
sup_measure.F id I

using *sup_measure_F_mono'*[*of I J*] **by** (*auto simp: finite_subset Un_absorb1*)

lemma *sets_sup_measure_F*:

finite I \implies *I* \neq $\{\}$ \implies ($\bigwedge i. i \in I \implies$ *sets i* = *sets M*) \implies *sets (sup_measure.F id I)* = *sets M*

by (*induction I rule: finite_ne_induct*) (*simp_all add: sets_sup*)

definition *Sup_measure'* :: 'a measure set \Rightarrow 'a measure **where**

Sup_measure' M =

measure_of ($\bigcup a \in M. \text{space } a$) ($\bigcup a \in M. \text{sets } a$)

($\lambda X. (\text{SUP } P \in \{P. \text{finite } P \wedge P \subseteq M\}. \text{sup_measure.F id } P X)$)

lemma *space_Sup_measure'2*: *space (Sup_measure' M)* = ($\bigcup m \in M. \text{space } m$)

unfolding *Sup_measure'_def* **by** (*intro space_measure_of[OF UN_space_closed]*)

lemma *sets_Sup_measure'2*: *sets (Sup_measure' M)* = *sigma_sets* ($\bigcup m \in M. \text{space } m$) ($\bigcup m \in M. \text{sets } m$)

unfolding *Sup_measure'_def* **by** (*intro sets_measure_of[OF UN_space_closed]*)

lemma *sets_Sup_measure'*:

assumes *sets_eq[simp]*: $\bigwedge m. m \in M \implies \text{sets } m = \text{sets } A$ **and** *M* \neq $\{\}$

shows $sets (Sup_measure' M) = sets A$
using $sets_eq[THEN sets_eq_imp_space_eq, simp] \langle M \neq \{\} \rangle$ **by** ($simp add: Sup_measure'_def$)

lemma $space_Sup_measure'$:

assumes $sets_eq[simp]: \bigwedge m. m \in M \implies sets m = sets A$ **and** $M \neq \{\}$
shows $space (Sup_measure' M) = space A$
using $sets_eq[THEN sets_eq_imp_space_eq, simp] \langle M \neq \{\} \rangle$
by ($simp add: Sup_measure'_def$)

lemma $emeasure_Sup_measure'$:

assumes $sets_eq[simp]: \bigwedge m. m \in M \implies sets m = sets A$ **and** $X \in sets A$ $M \neq \{\}$

shows $emeasure (Sup_measure' M) X = (SUP P \in \{P. finite P \wedge P \subseteq M\}. sup_measure.F id P X)$

($is _ = ?S X$)

using $Sup_measure'_def$

proof ($rule emeasure_measure_of$)

note $sets_eq[THEN sets_eq_imp_space_eq, simp]$

have $*$: $sets (Sup_measure' M) = sets A$ $space (Sup_measure' M) = space A$

using $\langle M \neq \{\} \rangle$ **by** ($simp_all add: Sup_measure'_def$)

let $?\mu = sup_measure.F id$

show $countably_additive (sets (Sup_measure' M)) ?S$

proof ($rule countably_additiveI, goal_cases$)

case ($1 F$)

then have $**$: $range F \subseteq sets A$

by ($auto simp: *$)

show $(\sum i. ?S (F i)) = ?S (\bigcup i. F i)$

proof ($subst ennreal_suminf_SUP_eq_directed$)

fix $i j$ **and** $N :: nat$ **set assume** $ij: i \in \{P. finite P \wedge P \subseteq M\}$ $j \in \{P. finite P \wedge P \subseteq M\}$

have $(i \neq \{\}) \longrightarrow sets (? \mu i) = sets A$ $\wedge (j \neq \{\}) \longrightarrow sets (? \mu j) = sets A$ \wedge
 $(i \neq \{\}) \vee j \neq \{\} \longrightarrow sets (? \mu (i \cup j)) = sets A$

using ij **by** ($intro impI sets_sup_measure_F conjI$) $auto$

then have $? \mu j (F n) \leq ? \mu (i \cup j) (F n) \wedge ? \mu i (F n) \leq ? \mu (i \cup j) (F n)$

for n

using ij

by ($cases i = \{\}; cases j = \{\}$)

($auto intro!: le_measureD3 sup_measure_F_mono simp: sets_sup_measure_F simp del: id_apply$)

with ij **show** $\exists k \in \{P. finite P \wedge P \subseteq M\}. \forall n \in N. ? \mu i (F n) \leq ? \mu k (F n) \wedge ? \mu j (F n) \leq ? \mu k (F n)$

by ($safe intro!: bexI[of _ i \cup j]$) $auto$

next

show $(SUP P \in \{P. finite P \wedge P \subseteq M\}. \sum n. ? \mu P (F n)) = (SUP P \in \{P. finite P \wedge P \subseteq M\}. ? \mu P (\bigcup (F ' UNIV)))$

proof ($intro arg_cong [of _ _ Sup] image_cong refl$)

fix i **assume** $i: i \in \{P. finite P \wedge P \subseteq M\}$

show $(\sum n. ? \mu i (F n)) = ? \mu i (\bigcup (F ' UNIV))$

```

proof cases
  assume  $i \neq \{\}$  with  $i$  ** show  $?thesis$ 
  by (smt (verit, best) 1(2) Measure_Space.sets_sup_measure_F assms(1))
mem_Collect_eq subset_eq suminf_cong suminf_emeasure)
  qed simp
qed
qed
show positive (sets (Sup_measure'  $M$ ))  $?S$ 
  by (auto simp: positive_def bot_enereal[symmetric])
show  $X \in \text{sets } (Sup\_measure' M)$ 
  using assms * by auto
qed (rule UN_space_closed)

```

```

definition Sup_measure :: 'a measure set  $\Rightarrow$  'a measure where
Sup_measure =
  Sup_lexord space
  (Sup_lexord sets Sup_measure'
  ( $\lambda U. \text{sigma } (\bigcup u \in U. \text{space } u) (\bigcup u \in U. \text{sets } u)$ ))
  ( $\lambda U. \text{sigma } (\bigcup u \in U. \text{space } u) \{\}$ )

```

```

definition Inf_measure :: 'a measure set  $\Rightarrow$  'a measure where
Inf_measure  $A = \text{Sup } \{x. \forall a \in A. x \leq a\}$ 

```

```

definition inf_measure :: 'a measure  $\Rightarrow$  'a measure where
inf_measure  $a b = \text{Inf } \{a, b\}$ 

```

```

definition top_measure :: 'a measure where
top_measure = Inf  $\{\}$ 

```

instance

proof

```

note UN_space_closed [simp]
show upper:  $x \leq \text{Sup } A$  if  $x \in A$  for  $x ::$  'a measure and  $A$ 
  unfolding Sup_measure_def
proof (intro Sup_lexord[where  $P = \lambda y. x \leq y$ ])
  assume  $\bigwedge a. a \in A \implies \text{space } a \neq (\bigcup a \in A. \text{space } a)$ 
  from this[OF  $\langle x \in A \rangle \langle x \in A \rangle$ ] show  $x \leq \text{sigma } (\bigcup a \in A. \text{space } a) \{\}$ 
  by (intro less_eq_measure.intros) auto
next
  fix  $a S$  assume  $a \in A$  and  $a: \text{space } a = (\bigcup a \in A. \text{space } a)$  and  $S: S = \{a' \in$ 
 $A. \text{space } a' = \text{space } a\}$ 
  and neg:  $\bigwedge aa. aa \in S \implies \text{sets } aa \neq (\bigcup a \in S. \text{sets } a)$ 
  have sp_a:  $\text{space } a = (\bigcup (\text{space } ' S))$ 
  using  $\langle a \in A \rangle$  by (auto simp: S)
  show  $x \leq \text{sigma } (\bigcup (\text{space } ' S)) (\bigcup (\text{sets } ' S))$ 
proof cases
  assume [simp]:  $\text{space } x = \text{space } a$ 
  have  $\text{sets } x \subset (\bigcup a \in S. \text{sets } a)$ 

```



```

    using ⟨x∈A⟩ neg[of x] by (auto simp: S)
  also have ... ⊆ sigma_sets (⋃ x∈S. space x) (⋃ x∈S. sets x)
    by (rule sigma_sets_superset_generator)
  finally show ?thesis
    by (intro less_eq_measure.intros(2)) (simp_all add: sp_a)
next
  assume space x ≠ space a
  moreover have space x ≤ space a
    unfolding a using ⟨x∈A⟩ by auto
  ultimately show ?thesis
    by (intro less_eq_measure.intros) (simp add: less_le sp_a)
qed
next
  fix a b S S' assume a ∈ A and a: space a = (⋃ a∈A. space a) and S: S =
  {a' ∈ A. space a' = space a}
    and b ∈ S and b: sets b = (⋃ a∈S. sets a) and S': S' = {a' ∈ S. sets a' =
  sets b}
  then have S' ≠ {} space b = space a
    by auto
  have sets_eq: ⋀x. x ∈ S' ⇒ sets x = sets b
    by (auto simp: S')
  note sets_eq[THEN sets_eq_imp_space_eq, simp]
  have *: sets (Sup_measure' S') = sets b space (Sup_measure' S') = space b
    using ⟨S' ≠ {}⟩ by (simp_all add: Sup_measure'_def sets_eq)
  show x ≤ Sup_measure' S'
  proof cases
    assume x ∈ S
    with ⟨b ∈ S⟩ have space x = space b
      by (simp add: S)
    show ?thesis
  proof cases
    assume x ∈ S'
    show x ≤ Sup_measure' S'
    proof (intro le_measure[THEN iffD2] ballI)
      show sets x = sets (Sup_measure' S')
        using ⟨x∈S'⟩ * by (simp add: S')
      fix X assume X ∈ sets x
      show emeasure x X ≤ emeasure (Sup_measure' S') X
      proof (subst emeasure_Sup_measure'[OF _ ⟨X ∈ sets x⟩])
        show emeasure x X ≤ (SUP P ∈ {P. finite P ∧ P ⊆ S'}). emeasure
        (sup_measure.F id P) X
          using ⟨x∈S'⟩ by (intro SUP_upper2[where i={x}]) auto
      qed (insert ⟨x∈S'⟩ S', auto)
    qed
  next
    assume x ∉ S'
    then have sets x ≠ sets b
      using ⟨x∈S⟩ by (auto simp: S')
    moreover have sets x ≤ sets b

```

```

    using ⟨x∈S⟩ unfolding b by auto
  ultimately show ?thesis
    using * ⟨x ∈ S⟩
    by (intro less_eq_measure.intros(2))
      (simp_all add: * ⟨space x = space b⟩ less_le)
  qed
next
  assume x ∉ S
  with ⟨x∈A⟩ ⟨x ∉ S⟩ ⟨space b = space a⟩ show ?thesis
    by (intro less_eq_measure.intros)
      (simp_all add: * less_le a SUP_upper S)
  qed
qed
show least: Sup A ≤ x if x:  $\bigwedge z. z \in A \implies z \leq x$  for x :: 'a measure and A
  unfolding Sup_measure_def
  proof (intro Sup_lexord[where P= $\lambda y. y \leq x$ ])
    assume  $\bigwedge a. a \in A \implies \text{space } a \neq (\bigcup a \in A. \text{space } a)$ 
    show sigma ( $\bigcup (\text{space } 'A)$ ) {} ≤ x
      using x[THEN le_measureD1] by (subst sigma_le_iff) auto
  next
    fix a S assume a ∈ A space a = ( $\bigcup a \in A. \text{space } a$ ) and S: S = {a' ∈ A. space
a' = space a}
       $\bigwedge a. a \in S \implies \text{sets } a \neq (\bigcup a \in S. \text{sets } a)$ 
    have  $\bigcup (\text{space } 'S) \subseteq \text{space } x$ 
      using S le_measureD1[OF x] by auto
    moreover
    have  $\bigcup (\text{space } 'S) = \text{space } a$ 
      using ⟨a∈A⟩ S by auto
    then have space x =  $\bigcup (\text{space } 'S) \implies \bigcup (\text{sets } 'S) \subseteq \text{sets } x$ 
      using ⟨a ∈ A⟩ le_measureD2[OF x] by (auto simp: S)
    ultimately show sigma ( $\bigcup (\text{space } 'S)$ ) ( $\bigcup (\text{sets } 'S)$ ) ≤ x
      by (subst sigma_le_iff) simp_all
  next
    fix a b S S' assume a ∈ A and a: space a = ( $\bigcup a \in A. \text{space } a$ ) and S: S =
{a' ∈ A. space a' = space a}
      and b ∈ S and b: sets b = ( $\bigcup a \in S. \text{sets } a$ ) and S': S' = {a' ∈ S. sets a' =
sets b}
    then have S' ≠ {} space b = space a
      by auto
    have sets_eq:  $\bigwedge x. x \in S' \implies \text{sets } x = \text{sets } b$ 
      by (auto simp: S')
    note sets_eq[THEN sets_eq_imp_space_eq, simp]
    have *: sets (Sup_measure' S') = sets b space (Sup_measure' S') = space b
      using ⟨S' ≠ {}⟩ by (simp_all add: Sup_measure'_def sets_eq)
    show Sup_measure' S' ≤ x
  proof cases
    assume space x = space a
    show ?thesis
  proof cases

```

```

assume **: sets x = sets b
show ?thesis
proof (intro le_measure[THEN iffD2] ballI)
  show **: sets (Sup_measure' S') = sets x
    by (simp add: * **)
  fix X assume X ∈ sets (Sup_measure' S')
  show emeasure (Sup_measure' S') X ≤ emeasure x X
    unfolding ***
  proof (subst emeasure_Sup_measure'[OF _ ⟨X ∈ sets (Sup_measure'
S')⟩])
    show (SUP P ∈ {P. finite P ∧ P ⊆ S'}. emeasure (sup_measure.F id
P) X) ≤ emeasure x X
    proof (safe intro!: SUP_least)
      fix P assume P: finite P P ⊆ S'
      show emeasure (sup_measure.F id P) X ≤ emeasure x X
      proof cases
        assume P = {} then show ?thesis
          by auto
        next
          assume P ≠ {}
          from P have finite P P ⊆ A
            unfolding S' S by (simp_all add: subset_eq)
          then have sup_measure.F id P ≤ x
            by (induction P) (auto simp: x)
          moreover have sets (sup_measure.F id P) = sets x
            using ⟨finite P⟩ ⟨P ≠ {}⟩ ⟨P ⊆ S'⟩ ⟨sets x = sets b⟩
            by (intro sets_sup_measure_F) (auto simp: S')
          ultimately show emeasure (sup_measure.F id P) X ≤ emeasure x X
            by (rule le_measureD3)
        qed
      qed
    show m ∈ S' ⇒ sets m = sets (Sup_measure' S') for m
      unfolding * by (simp add: S')
    qed fact
  qed
next
  assume sets x ≠ sets b
  moreover have sets b ≤ sets x
    unfolding b S using x[THEN le_measureD2] ⟨space x = space a⟩ by auto
  ultimately show Sup_measure' S' ≤ x
    using ⟨space x = space a⟩ ⟨b ∈ S⟩
    by (intro less_eq_measure.intros(2)) (simp_all add: * S)
  qed
next
  assume space x ≠ space a
  then have space a < space x
    using le_measureD1[OF x[OF ⟨a∈A⟩]] by auto
  then show Sup_measure' S' ≤ x
    by (intro less_eq_measure.intros) (simp add: * ⟨space b = space a⟩)

```

```

    qed
  qed
  show  $Sup \{ \} = (bot::'a \text{ measure}) Inf \{ \} = (top::'a \text{ measure})$ 
    by (auto intro!: antisym least simp: top_measure_def)
  show lower:  $x \in A \implies Inf A \leq x$  for  $x :: 'a \text{ measure}$  and  $A$ 
    unfolding Inf_measure_def by (intro least) auto
  show greatest:  $(\bigwedge z. z \in A \implies x \leq z) \implies x \leq Inf A$  for  $x :: 'a \text{ measure}$  and  $A$ 
    unfolding Inf_measure_def by (intro upper) auto
  show  $inf\ x\ y \leq x$   $inf\ x\ y \leq y$   $x \leq y \implies x \leq z \implies x \leq inf\ y\ z$  for  $x\ y\ z :: 'a$ 
    measure
    by (auto simp: inf_measure_def intro!: lower greatest)
  qed

```

end

lemma sets_SUP:

```

  assumes  $\bigwedge x. x \in I \implies sets\ (M\ x) = sets\ N$ 
  shows  $I \neq \{ \} \implies sets\ (SUP\ i \in I. M\ i) = sets\ N$ 
  unfolding Sup_measure_def
  using assms assms[THEN sets_eq_imp_space_eq]
  sets_Sup_measure'[where  $A=N$  and  $M=M'I$ ]
  by (intro Sup_lexord1[where  $P=\lambda x. sets\ x = sets\ N$ ]) auto

```

lemma emeasure_SUP:

```

  assumes sets:  $\bigwedge i. i \in I \implies sets\ (M\ i) = sets\ N$   $X \in sets\ N$   $I \neq \{ \}$ 
  shows emeasure  $(SUP\ i \in I. M\ i)\ X = (SUP\ J \in \{ J. J \neq \{ \} \wedge finite\ J \wedge J \subseteq I \}.$ 
    emeasure  $(SUP\ i \in J. M\ i)\ X)$ 
  proof -
    interpret sup_measure: comm_monoid_set sup bot :: 'b measure
    by standard (auto intro!: antisym)
    have eq:  $finite\ J \implies sup\_measure.F\ id\ J = (SUP\ i \in J. i)$  for  $J :: 'b \text{ measure}$ 
    set
    by (induction J rule: finite_induct) auto
    have 1:  $J \neq \{ \} \implies J \subseteq I \implies sets\ (SUP\ x \in J. M\ x) = sets\ N$  for  $J$ 
    by (intro sets_SUP sets) (auto)
    from  $\langle I \neq \{ \} \rangle$  obtain  $i$  where  $i \in I$  by auto
    have Sup_measure'(M'I)  $X = (SUP\ P \in \{ P. finite\ P \wedge P \subseteq M'I \}.$  sup_measure.F
    id P X)
    using sets by (intro emeasure_Sup_measure') auto
    also have Sup_measure'(M'I) =  $(SUP\ i \in I. M\ i)$ 
    unfolding Sup_measure_def using  $\langle I \neq \{ \} \rangle$  sets sets(1)[THEN sets_eq_imp_space_eq]
    by (intro Sup_lexord1[where  $P=\lambda x. \_ = x$ ]) auto
    also have  $(SUP\ P \in \{ P. finite\ P \wedge P \subseteq M'I \}.$  sup_measure.F id P X) =
     $(SUP\ J \in \{ J. J \neq \{ \} \wedge finite\ J \wedge J \subseteq I \}.$   $(SUP\ i \in J. M\ i)\ X)$ 
    proof (intro SUP_eq)
      fix J assume  $J \in \{ P. finite\ P \wedge P \subseteq M'I \}$ 
      then obtain J' where  $J': J' \subseteq I$   $finite\ J'$  and  $J: J = M'I\ J'$  and  $finite\ J$ 
      using finite_subset_image[of J M I] by auto
      show  $\exists j \in \{ J. J \neq \{ \} \wedge finite\ J \wedge J \subseteq I \}.$  sup_measure.F id J X  $\leq (SUP\ i \in j.$ 

```

```

M i) X
  proof cases
    assume J' = {} with ⟨i ∈ I⟩ show ?thesis
      by (auto simp add: J)
    next
      assume J' ≠ {} with J J' show ?thesis
        by (intro bexI[of _ J']) (auto simp add: eq simp del: id_apply)
      qed
    next
      fix J assume J: J ∈ {P. P ≠ {} ∧ finite P ∧ P ⊆ I}
        show ∃ J' ∈ {J. finite J ∧ J ⊆ M' I}. (SUP i ∈ J. M i) X ≤ sup_measure.F id
        J' X
          using J by (intro bexI[of _ M' J]) (auto simp add: eq simp del: id_apply)
        qed
      finally show ?thesis .
    qed
  qed

```

lemma *emeasure_SUP_chain*:

```

  assumes sets:  $\bigwedge i. i \in A \implies \text{sets } (M i) = \text{sets } N X \in \text{sets } N$ 
  assumes ch: Complete_Partial_Order.chain ( $\leq$ ) (M ' A) and A ≠ {}
  shows emeasure (SUP i ∈ A. M i) X = (SUP i ∈ A. emeasure (M i) X)
  proof (subst emeasure_SUP[OF sets ⟨A ≠ {}⟩])
    show (SUP J ∈ {J. J ≠ {} ∧ finite J ∧ J ⊆ A}. emeasure (Sup (M ' J)) X) =
    (SUP i ∈ A. emeasure (M i) X)
    proof (rule SUP_eq)
      fix J assume J ∈ {J. J ≠ {} ∧ finite J ∧ J ⊆ A}
      then have J: Complete_Partial_Order.chain ( $\leq$ ) (M ' J) finite J J ≠ {} and
      J ⊆ A
        using ch[THEN chain_subset, of M' J] by auto
      with in_chain_finite[OF J(1)] obtain j where j ∈ J (SUP j ∈ J. M j) = M j
        by auto
      with ⟨J ⊆ A⟩ show ∃ j ∈ A. emeasure (Sup (M ' J)) X ≤ emeasure (M j) X
        by auto
    next
      fix j assume j ∈ A then show ∃ i ∈ {J. J ≠ {} ∧ finite J ∧ J ⊆ A}. emeasure
      (M j) X ≤ emeasure (Sup (M ' i)) X
        by (intro bexI[of _ {j}]) auto
    qed
  qed
  qed

```

Supremum of a set of σ -algebras

lemma *space_Sup_eq_UN*: $\text{space } (\text{Sup } M) = (\bigcup x \in M. \text{space } x)$ (is ?L=?R)

proof

```

  show ?L ⊆ ?R
    using Sup_lexord[where P=λx. space x = _]
    apply (clarsimp simp: Sup_measure_def)
    by (smt (verit) Sup_lexord_def UN_E mem_Collect_eq space_Sup_measure'2
    space_measure_of_conv)

```

qed (use *Sup_upper le_measureD1* in *fastforce*)

lemma *sets_Sup_eq*:

assumes *: $\bigwedge m. m \in M \implies \text{space } m = X$ **and** $M \neq \{\}$

shows $\text{sets } (\text{Sup } M) = \text{sigma_sets } X (\bigcup_{x \in M}. \text{sets } x)$

unfolding *Sup_measure_def*

proof (rule *Sup_lexord1* [*OF* $\langle M \neq \{\} \rangle$])

show $\text{sets } (\text{Sup_lexord } \text{sets } \text{Sup_measure}' (\lambda U. \text{sigma } (\bigcup (\text{space } 'U)) (\bigcup (\text{sets } 'U)))) M$

$= \text{sigma_sets } X (\bigcup (\text{sets } 'M))$

apply (rule *Sup_lexord*)

apply (*metis* (*mono_tags*, *lifting*) * *empty_iff mem_Collect_eq sets.sigma_sets_eq sets_Sup_measure'*)

by (*metis* * *SUP_eq_const UN_space_closed assms(2) sets_measure_of*)

qed (use * in *blast*)

lemma *in_sets_Sup*: $(\bigwedge m. m \in M \implies \text{space } m = X) \implies m \in M \implies A \in \text{sets } m \implies A \in \text{sets } (\text{Sup } M)$

by (*subst sets_Sup_eq*[**where** $X=X$]) *auto*

lemma *Sup_lexord_rel*:

assumes $\bigwedge i. i \in I \implies k (A \ i) = k (B \ i)$

$R (c (A \ \{a \in I. k (B \ a) = (\text{SUP } x \in I. k (B \ x))\})) (c (B \ \{a \in I. k (B \ a) = (\text{SUP } x \in I. k (B \ x))\}))$

$R (s (A \ I)) (s (B \ I))$

shows $R (\text{Sup_lexord } k \ c \ s (A \ I)) (\text{Sup_lexord } k \ c \ s (B \ I))$

proof –

have $A \ \{a \in I. k (B \ a) = (\text{SUP } x \in I. k (B \ x))\} = \{a \in A \ \{I. k \ a = (\text{SUP } x \in I. k (B \ x))\}\}$

using *assms(1)* **by** *auto*

moreover have $B \ \{a \in I. k (B \ a) = (\text{SUP } x \in I. k (B \ x))\} = \{a \in B \ \{I. k \ a = (\text{SUP } x \in I. k (B \ x))\}\}$

by *auto*

ultimately show *?thesis*

using *assms* **by** (*auto simp: Sup_lexord_def Let_def image_comp*)

qed

lemma *sets_SUP_cong*:

assumes *eq*: $\bigwedge i. i \in I \implies \text{sets } (M \ i) = \text{sets } (N \ i)$

shows $\text{sets } (\text{SUP } i \in I. M \ i) = \text{sets } (\text{SUP } i \in I. N \ i)$

unfolding *Sup_measure_def*

using *eq* [*THEN sets_eq_imp_space_eq*]

by (*intro Sup_lexord_rel*[**where** $R=\lambda x \ y. \text{sets } x = \text{sets } y$], *simp_all add: sets_Sup_measure'2*)

lemma *sets_Sup_in_sets*:

assumes $M \neq \{\}$

assumes $\bigwedge m. m \in M \implies \text{space } m = \text{space } N$

```

  assumes  $\bigwedge m. m \in M \implies \text{sets } m \subseteq \text{sets } N$ 
  shows  $\text{sets } (\text{Sup } M) \subseteq \text{sets } N$ 
  proof -
    have *:  $\bigcup (\text{space } \langle M \rangle) = \text{space } N$ 
      using assms by auto
    show ?thesis
      unfolding * using assms by (subst sets_Sup_eq[of M space N]) (auto intro!:
sets.sigma_sets_subset)
  qed

```

```

lemma measurable_Sup1:
  assumes m:  $m \in M$  and f:  $f \in \text{measurable } m N$ 
  and const_space:  $\bigwedge m n. m \in M \implies n \in M \implies \text{space } m = \text{space } n$ 
  shows  $f \in \text{measurable } (\text{Sup } M) N$ 
  proof -
    have  $\text{space } (\text{Sup } M) = \text{space } m$ 
      using m by (auto simp add: space_Sup_eq_UN dest: const_space)
    then show ?thesis
      using m f unfolding measurable_def by (auto intro: in_sets_Sup[OF const_space])
  qed

```

```

lemma measurable_Sup2:
  assumes M:  $M \neq \{\}$ 
  assumes f:  $\bigwedge m. m \in M \implies f \in \text{measurable } N m$ 
  and const_space:  $\bigwedge m n. m \in M \implies n \in M \implies \text{space } m = \text{space } n$ 
  shows  $f \in \text{measurable } N (\text{Sup } M)$ 
  proof -
    from M obtain m where  $m \in M$  by auto
    have space_eq:  $\bigwedge n. n \in M \implies \text{space } n = \text{space } m$ 
      by (intro const_space  $\langle m \in M \rangle$ )
    have eq:  $\text{sets } (\text{sigma } (\bigcup (\text{space } \langle M \rangle)) (\bigcup (\text{sets } \langle M \rangle))) = \text{sets } (\text{Sup } M)$ 
      by (metis M SUP_eq_const UN_space_closed sets_Sup_eq sets_measure_of
space_eq)
    have  $f \in \text{measurable } N (\text{sigma } (\bigcup_{m \in M. \text{space } m}) (\bigcup_{m \in M. \text{sets } m}))$ 
      proof (rule measurable_measure_of)
        show  $f \in \text{space } N \rightarrow \bigcup (\text{space } \langle M \rangle)$ 
          using measurable_space[OF f] M by auto
        qed (auto intro: measurable_sets f dest: sets_sets_into_space)
    also have  $\text{measurable } N (\text{sigma } (\bigcup_{m \in M. \text{space } m}) (\bigcup_{m \in M. \text{sets } m})) = \text{measurable } N (\text{Sup } M)$ 
      using eq measurable_cong_sets by blast
    finally show ?thesis .
  qed

```

```

lemma measurable_SUP2:
   $I \neq \{\} \implies (\bigwedge i. i \in I \implies f \in \text{measurable } N (M i)) \implies$ 
   $(\bigwedge i j. i \in I \implies j \in I \implies \text{space } (M i) = \text{space } (M j)) \implies f \in \text{measurable } N$ 
   $(\text{SUP } i \in I. M i)$ 
  by (auto intro!: measurable_Sup2)

```

lemma *sets_Sup_sigma*:
assumes [*simp*]: $M \neq \{\}$ **and** $M: \bigwedge m. m \in M \implies m \subseteq \text{Pow } \Omega$
shows $\text{sets } (\text{SUP } m \in M. \text{sigma } \Omega m) = \text{sets } (\text{sigma } \Omega (\bigcup M))$
proof –
{ **fix** $a m$ **assume** $a \in \text{sigma_sets } \Omega m m \in M$
then have $a \in \text{sigma_sets } \Omega (\bigcup M)$
by *induction* (*auto intro: sigma_sets.intros(2-)*) }
then have $\text{sigma_sets } \Omega (\bigcup (\text{sigma_sets } \Omega ' M)) = \text{sigma_sets } \Omega (\bigcup M)$
by (*smt (verit, best) UN_iff Union_iff sigma_sets.Basic sigma_sets_eqI*)
then show $\text{sets } (\text{SUP } m \in M. \text{sigma } \Omega m) = \text{sets } (\text{sigma } \Omega (\bigcup M))$
by (*subst sets_Sup_eq*) (*fastforce simp add: M Union_least*)+
qed

lemma *Sup_sigma*:
assumes [*simp*]: $M \neq \{\}$ **and** $M: \bigwedge m. m \in M \implies m \subseteq \text{Pow } \Omega$
shows $(\text{SUP } m \in M. \text{sigma } \Omega m) = (\text{sigma } \Omega (\bigcup M))$
proof (*intro antisym SUP_least*)
have $*$: $\bigcup M \subseteq \text{Pow } \Omega$
using M **by** *auto*
show $\text{sigma } \Omega (\bigcup M) \leq (\text{SUP } m \in M. \text{sigma } \Omega m)$
proof (*intro less_eq_measure.intros(3)*)
show $\text{space } (\text{sigma } \Omega (\bigcup M)) = \text{space } (\text{SUP } m \in M. \text{sigma } \Omega m)$
 $\text{sets } (\text{sigma } \Omega (\bigcup M)) = \text{sets } (\text{SUP } m \in M. \text{sigma } \Omega m)$
by (*auto simp add: M sets_Sup_sigma sets_eq_imp_space_eq space_measure_of_conv*)
qed (*simp add: emeasure_sigma le_fun_def*)
fix m **assume** $m \in M$ **then show** $\text{sigma } \Omega m \leq \text{sigma } \Omega (\bigcup M)$
by (*subst sigma_le_iff*) (*auto simp add: M **)
qed

lemma *SUP_sigma_sigma*:
 $M \neq \{\} \implies (\bigwedge m. m \in M \implies f m \subseteq \text{Pow } \Omega) \implies (\text{SUP } m \in M. \text{sigma } \Omega (f m))$
 $= \text{sigma } \Omega (\bigcup m \in M. f m)$
using *Sup_sigma[of f'M Ω]* **by** (*auto simp: image_comp*)

lemma *sets_vimage_Sup_eq*:
assumes $*$: $M \neq \{\}$ $f \in X \rightarrow Y \bigwedge m. m \in M \implies \text{space } m = Y$
shows $\text{sets } (\text{vimage_algebra } X f (\text{Sup } M)) = \text{sets } (\text{SUP } m \in M. \text{vimage_algebra } X f m)$
(is ?L = ?R)
proof
have $\bigwedge m. m \in M \implies f \in \text{Sup } (\text{vimage_algebra } X f ' M) \rightarrow_M m$
using *assms*
by (*smt (verit, del_insts) Pi_iff imageE image_eqI measurable_Sup1 measurable_vimage_algebra1 space_vimage_algebra*)
then show ?L \subseteq ?R
by (*intro sets_image_in_sets measurable_Sup2*) (*simp_all add: space_Sup_eq_UN*
 $*$)
show ?R \subseteq ?L


```

  apply (intro sets_Sup_in_sets)
  apply (force simp add: * space_Sup_eq_UN sets_vimage_algebra2 intro:
in_sets_Sup)+
  done
qed

```

lemma *restrict_space_eq_vimage_algebra'*:

sets (restrict_space M Ω) = sets (vimage_algebra ($\Omega \cap \text{space } M$) ($\lambda x. x$) M)

proof –

have *: $\{A \cap (\Omega \cap \text{space } M) \mid A. A \in \text{sets } M\} = \{A \cap \Omega \mid A. A \in \text{sets } M\}$

using *sets.sets_into_space[of _ M]* **by** *blast*

show *?thesis*

unfolding *restrict_space_def*

by (*subst sets_measure_of*)

(*auto simp add: image_subset_iff sets_vimage_algebra * dest: sets.sets_into_space*
intro!: arg_cong2[where f=sigma_sets])

qed

lemma *sigma_le_sets*:

assumes [*simp*]: $A \subseteq \text{Pow } X$ **shows** $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N \longleftrightarrow X \in \text{sets } N \wedge A \subseteq \text{sets } N$

proof

have $X \in \text{sigma_sets } X A \wedge A \subseteq \text{sigma_sets } X A$

by (*auto intro: sigma_sets_top*)

moreover assume $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N$

ultimately show $X \in \text{sets } N \wedge A \subseteq \text{sets } N$

by *auto*

next

assume *: $X \in \text{sets } N \wedge A \subseteq \text{sets } N$

{ **fix** Y **assume** $Y \in \text{sigma_sets } X A$ **from this** * **have** $Y \in \text{sets } N$

by *induction auto* }

then show $\text{sets } (\text{sigma } X A) \subseteq \text{sets } N$

by *auto*

qed

lemma *measurable_iff_sets*:

$f \in \text{measurable } M N \longleftrightarrow (f \in \text{space } M \rightarrow \text{space } N \wedge \text{sets } (\text{vimage_algebra } (\text{space } M) f N) \subseteq \text{sets } M)$

unfolding *measurable_def*

by (*smt (verit, ccfv_threshold) mem_Collect_eq sets_vimage_algebra sigma_sets_le_sets_iff subset_eq*)

lemma *sets_vimage_algebra_space*: $X \in \text{sets } (\text{vimage_algebra } X f M)$

using *sets.top[of vimage_algebra X f M]* **by** *simp*

lemma *measurable_mono*:

assumes $N: \text{sets } N' \leq \text{sets } N \text{ space } N = \text{space } N'$

assumes $M: \text{sets } M \leq \text{sets } M' \text{ space } M = \text{space } M'$

shows $\text{measurable } M \ N \subseteq \text{measurable } M' \ N'$
unfolding measurable_def
proof *safe*
fix $f \ A$ **assume** $f \in \text{space } M \rightarrow \text{space } N \ A \in \text{sets } N'$
moreover **assume** $\forall y \in \text{sets } N. f -' y \cap \text{space } M \in \text{sets } M$ **note** *this[THEN*
bspec, of A]
ultimately **show** $f -' A \cap \text{space } M' \in \text{sets } M'$
using *assms by auto*
qed (*use N M in auto*)

lemma $\text{measurable_Sup_measurable}$:
assumes $f: f \in \text{space } N \rightarrow A$
shows $f \in \text{measurable } N \ (\text{Sup } \{M. \text{space } M = A \wedge f \in \text{measurable } N \ M\})$
proof (*rule measurable_Sup2*)
show $\{M. \text{space } M = A \wedge f \in \text{measurable } N \ M\} \neq \{\}$
using f **unfolding** $\text{ex_in_conv[symmetric]}$
by (*intro exI[of _ sigma A {}]*) (*auto intro!: measurable_measure_of*)
qed *auto*

lemma (*in sigma_algebra*) $\text{sigma_sets_subset'}$:
assumes $a: a \subseteq M \ \Omega' \in M$
shows $\text{sigma_sets } \Omega' \ a \subseteq M$
proof
show $x \in M$ **if** $x: x \in \text{sigma_sets } \Omega' \ a$ **for** x
using x **by** (*induct rule: sigma_sets.induct*) (*use a in auto*)
qed

lemma in_sets_SUP : $i \in I \implies (\bigwedge i. i \in I \implies \text{space } (M \ i) = Y) \implies X \in \text{sets}$
 $(M \ i) \implies X \in \text{sets } (\text{SUP } i \in I. M \ i)$
by (*intro in_sets_Sup[where X=Y]*) *auto*

lemma measurable_SUP1 :
 $i \in I \implies f \in \text{measurable } (M \ i) \ N \implies (\bigwedge m \ n. m \in I \implies n \in I \implies \text{space } (M$
 $m) = \text{space } (M \ n)) \implies$
 $f \in \text{measurable } (\text{SUP } i \in I. M \ i) \ N$
by (*auto intro: measurable_Sup1*)

lemma $\text{sets_image_in_sets'}$:
assumes $X: X \in \text{sets } N$
assumes $f: \bigwedge A. A \in \text{sets } M \implies f -' A \cap X \in \text{sets } N$
shows $\text{sets_vimage_algebra } X \ f \ M \subseteq \text{sets } N$
unfolding $\text{sets_vimage_algebra}$
by (*rule sets.sigma_sets_subset'*) (*auto intro!: measurable_sets X f*)

lemma $\text{mono_vimage_algebra}$:
 $\text{sets } M \leq \text{sets } N \implies \text{sets } (\text{vimage_algebra } X \ f \ M) \subseteq \text{sets } (\text{vimage_algebra } X \ f$
 $N)$
using $\text{sets.top[of sigma } X \ \{f -' A \cap X \mid A. A \in \text{sets } N\}]$
unfolding $\text{vimage_algebra_def}$

by (smt (verit, del_insts) space_measure_of_sigma_le_sets Pow_iff_inf_le2 mem_Collect_eq subset_eq)

lemma *mono_restrict_space*: $sets\ M \leq sets\ N \implies sets\ (restrict_space\ M\ X) \subseteq sets\ (restrict_space\ N\ X)$

unfolding *sets_restrict_space* by (rule *image_mono*)

lemma *sets_eq_bot*: $sets\ M = \{\{\}\} \longleftrightarrow M = bot$

by (metis *measure_eqI* *emeasure_empty* *sets_bot* *singletonD*)

lemma *sets_eq_bot2*: $\{\{\}\} = sets\ M \longleftrightarrow M = bot$

using *sets_eq_bot*[of *M*] by blast

lemma (in *finite_measure*) *countable_support*:

countable $\{x.\ measure\ M\ \{x\} \neq 0\}$

proof *cases*

assume *measure* *M* (*space* *M*) = 0

then show *?thesis*

by (metis (mono_tags, lifting) *bounded_measure* *measure_le_0_iff* *Collect_empty_eq* *countable_empty*)

next

let *?M* = *measure* *M* (*space* *M*) and *?m* = $\lambda x.\ measure\ M\ \{x\}$

assume *?M* $\neq 0$

then have *: $\{x.\ ?m\ x \neq 0\} = (\bigcup n.\ \{x.\ ?M / Suc\ n < ?m\ x\})$

using *reals_Archimedean*[of *?m* *x* / *?M* for *x*]

by (auto simp: *field_simps* *not_le*[*symmetric*] *divide_le_0_iff* *measure_le_0_iff*)

have **: $\bigwedge n.\ finite\ \{x.\ ?M / Suc\ n < ?m\ x\}$

proof (rule *ccontr*)

fix *n* assume *infinite* $\{x.\ ?M / Suc\ n < ?m\ x\}$ (is *infinite* *?X*)

then obtain *X* where *finite* *X* *card* *X* = *Suc* (*Suc* *n*) $X \subseteq ?X$

by (metis *infinite_arbitrarily_large*)

then have *: $\bigwedge x.\ x \in X \implies ?M / Suc\ n \leq ?m\ x$

by *auto*

{ fix *x* assume *x* $\in X$

from $\langle ?M \neq 0 \rangle$ **[OF this]* have *?m* *x* $\neq 0$ by (auto simp: *field_simps* *measure_le_0_iff*)

then have $\{x\} \in sets\ M$ by (auto dest: *measure_notin_sets*) }

note *singleton_sets* = *this*

have *?M* $< (\sum_{x \in X} ?M / Suc\ n)$

using $\langle ?M \neq 0 \rangle$

by (simp add: $\langle card\ X = Suc\ (Suc\ n) \rangle$ *field_simps* *less_le*)

also have $\dots \leq (\sum_{x \in X} ?m\ x)$

by (rule *sum_mono*) *fact*

also have $\dots = measure\ M\ (\bigcup_{x \in X} \{x\})$

using *singleton_sets* $\langle finite\ X \rangle$

by (intro *finite_measure_finite_Union*[*symmetric*]) (auto simp: *disjoint_family_on_def*)

finally have *?M* $< measure\ M\ (\bigcup_{x \in X} \{x\})$.

moreover have *measure* *M* $(\bigcup_{x \in X} \{x\}) \leq ?M$

```

    using singleton_sets[THEN sets.sets_into_space] by (intro finite_measure_mono)
  auto
  ultimately show False by simp
  qed
  show ?thesis
  unfolding * by (intro countable_UN countableI_type countable_finite[OF **])
  qed
end

```

7.4 Borel Space

theory *Borel_Space*

imports

Measurable Derivative Ordered_Euclidean_Space Extended_Real_Limits

begin

lemma *is_interval_real_ereal_oo: is_interval (real_of_ereal ‘{N<..)
by (*auto simp: real_atLeastGreaterThan_eq*)*

lemma *sets_Collect_eventually_sequentially[measurable]:*

$(\bigwedge i. \{x \in \text{space } M. P\ x\ i\} \in \text{sets } M) \implies \{x \in \text{space } M. \text{eventually } (P\ x)\ \text{sequentially}\} \in \text{sets } M$

unfolding *eventually_sequentially* **by** *simp*

lemma *topological_basis_trivial: topological_basis {A. open A}*

by (*auto simp: topological_basis_def*)

proposition *open_prod_generated: open = generate_topology {A × B | A B. open A ∧ open B}*

proof –

have $\{A \times B :: ('a \times 'b)\ \text{set} \mid A\ B.\ \text{open } A \wedge \text{open } B\} = ((\lambda(a, b). a \times b)\ ‘\{\text{A. open } A\} \times \{\text{A. open } A\})$

by *auto*

then show *?thesis*

by (*auto intro: topological_basis_prod topological_basis_trivial topological_basis_imp_subbasis*)
qed

proposition *mono_on_imp_deriv_nonneg:*

assumes *mono: mono_on A f* **and** *deriv: (f has_real_derivative D) (at x)*

assumes $x \in \text{interior } A$

shows $D \geq 0$

proof (*rule tendsto_lowerbound*)

let $?A' = (\lambda y. y - x)\ ‘\text{interior } A$

from *deriv* **show** $((\lambda h. (f(x + h) - f(x)) / h) \longrightarrow D)\ \text{(at } 0)$

by (*simp add: field_has_derivative_at has_field_derivative_def*)

from *mono* **have** *mono': mono_on (interior A) f* **by** (*rule mono_on_subset*)
(rule interior_subset)

```

show eventually ( $\lambda h. (f (x + h) - f x) / h \geq 0$ ) (at 0)
proof (subst eventually_at_topological, intro exI conjI ballI impI)
  have open (interior A) by simp
  hence open ((+) (-x) 'interior A) by (rule open_translation)
  also have ((+) (-x) 'interior A) = ?A' by auto
  finally show open ?A'.
next
  from  $\langle x \in \text{interior } A \rangle$  show  $0 \in ?A'$  by auto
next
  fix h assume  $h \in ?A'$ 
  hence  $x + h \in \text{interior } A$  by auto
  with mono' and  $\langle x \in \text{interior } A \rangle$  show  $(f (x + h) - f x) / h \geq 0$ 
  by (cases h rule: linorder_cases[of_ 0])
  (simp_all add: divide_nonpos_neg divide_nonneg_pos mono_onD field_simps)
qed
qed simp

proposition mono_on_ctble_discont:
  fixes f :: real  $\Rightarrow$  real
  fixes A :: real set
  assumes mono_on A f
  shows countable  $\{a \in A. \neg \text{continuous (at } a \text{ within } A) f\}$ 
proof -
  have mono:  $\bigwedge x y. x \in A \implies y \in A \implies x \leq y \implies f x \leq f y$ 
  using  $\langle \text{mono\_on } A f \rangle$  by (simp add: mono_on_def)
  have  $\forall a \in \{a \in A. \neg \text{continuous (at } a \text{ within } A) f\}. \exists q :: \text{nat} \times \text{rat.}$ 
    ( $\text{fst } q = 0 \wedge \text{of\_rat (snd } q) < f a \wedge (\forall x \in A. x < a \implies f x < \text{of\_rat (snd } q))$ )  $\vee$ 
    ( $\text{fst } q = 1 \wedge \text{of\_rat (snd } q) > f a \wedge (\forall x \in A. x > a \implies f x > \text{of\_rat (snd } q))$ )
  proof (clarsimp simp del: One_nat_def)
    fix a assume  $a \in A$  assume  $\neg \text{continuous (at } a \text{ within } A) f$ 
    thus  $\exists q1 q2.$ 
       $q1 = 0 \wedge \text{real\_of\_rat } q2 < f a \wedge (\forall x \in A. x < a \implies f x < \text{real\_of\_rat } q2)$   $\vee$ 
       $q1 = 1 \wedge f a < \text{real\_of\_rat } q2 \wedge (\forall x \in A. a < x \implies \text{real\_of\_rat } q2 < f x)$ 
  proof (auto simp add: continuous_within order_tendsto_iff eventually_at)
    fix l assume  $l < f a$ 
    then obtain q2 where  $q2: l < \text{of\_rat } q2 \text{ of\_rat } q2 < f a$ 
    using of_rat_dense by blast
    assume * [rule_format]:  $\forall d > 0. \exists x \in A. x \neq a \wedge \text{dist } x a < d \wedge \neg l < f x$ 
    from q2 have  $\text{real\_of\_rat } q2 < f a \wedge (\forall x \in A. x < a \implies f x < \text{real\_of\_rat } q2)$ 
  using q2 *[of a - _] dist_real_def mono by fastforce
  thus ?thesis by auto
next
  fix u assume  $u > f a$ 
  then obtain q2 where  $q2: f a < \text{of\_rat } q2 \text{ of\_rat } q2 < u$ 

```

```

    using of_rat_dense by blast
    assume *[rule_format]:  $\forall d > 0. \exists x \in A. x \neq a \wedge \text{dist } x \ a < d \wedge \neg u > f \ x$ 
    from q2 have real_of_rat q2 > f a  $\wedge (\forall x \in A. x > a \longrightarrow f \ x > \text{real\_of\_rat } q2)$ 
    using q2 *[of_ - a] dist_real_def mono by fastforce
    thus ?thesis by auto
  qed
  then obtain g :: real  $\Rightarrow$  nat  $\times$  rat where  $\forall a \in \{a \in A. \neg \text{continuous (at a within A) } f\}$ .
    (fst (g a) = 0  $\wedge$  of_rat (snd (g a)) < f a  $\wedge (\forall x \in A. x < a \longrightarrow f \ x < \text{of\_rat (snd (g a))})$ ) |
    (fst (g a) = 1  $\wedge$  of_rat (snd (g a)) > f a  $\wedge (\forall x \in A. x > a \longrightarrow f \ x > \text{of\_rat (snd (g a))})$ )
  by (rule bchoice [THEN exE]) blast
  hence g:  $\bigwedge a \ x. a \in A \Longrightarrow \neg \text{continuous (at a within A) } f \Longrightarrow x \in A \Longrightarrow$ 
    (fst (g a) = 0  $\wedge$  of_rat (snd (g a)) < f a  $\wedge (x < a \longrightarrow f \ x < \text{of\_rat (snd (g a))})$ ) |
    (fst (g a) = 1  $\wedge$  of_rat (snd (g a)) > f a  $\wedge (x > a \longrightarrow f \ x > \text{of\_rat (snd (g a))})$ )
  by auto
  have inj_on g {a  $\in$  A.  $\neg$  continuous (at a within A) f}
  proof (auto simp add: inj_on_def)
    fix w z
    assume 1: w  $\in$  A and 2:  $\neg$  continuous (at w within A) f and
      3: z  $\in$  A and 4:  $\neg$  continuous (at z within A) f and
      5: g w = g z
    from g [OF 1 2 3] g [OF 3 4 1] 5
    show w = z by auto
  qed
  thus ?thesis
  by (rule countableI')
  qed

```

lemma mono_on_ctble_discont_open:

```

  fixes f :: real  $\Rightarrow$  real
  fixes A :: real set
  assumes open A mono_on A f
  shows countable {a  $\in$  A.  $\neg$  isCont f a}
  using continuous_within_open [OF _ <open A>] <mono_on A f>
  by (smt (verit, ccfv_threshold) Collect_cong mono_on_ctble_discont)

```

lemma mono_ctble_discont:

```

  fixes f :: real  $\Rightarrow$  real
  assumes mono f
  shows countable {a.  $\neg$  isCont f a}
  using assms mono_on_ctble_discont [of UNIV f] unfolding mono_on_def
  mono_def by auto

```

lemma *has_real_derivative_imp_continuous_on*:

assumes $\bigwedge x. x \in A \implies (f \text{ has_real_derivative } f' x) \text{ (at } x)$

shows *continuous_on* A f

by (*meson* *DERIV_isCont* *assms* *continuous_at_imp_continuous_on*)

lemma *continuous_interval_vimage_Int*:

assumes *continuous_on* $\{a::\text{real}..b\}$ g **and** *mono*: $\bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies g x \leq g y$

assumes $a \leq b$ $(c::\text{real}) \leq d$ $\{c..d\} \subseteq \{g a..g b\}$

obtains $c' d'$ **where** $\{a..b\} \cap g^{-1} \{c..d\} = \{c'..d'\}$ $c' \leq d'$ $g c' = c$ $g d' = d$

proof–

let $?A = \{a..b\} \cap g^{-1} \{c..d\}$

from *IVT'*[*of* g a c b , *OF* g $\langle a \leq b \rangle$ *assms*(1)] *assms*(4,5)

obtain c'' **where** c'' : $c'' \in ?A$ $g c'' = c$ **by** *auto*

from *IVT'*[*of* g a d b , *OF* g $\langle a \leq b \rangle$ *assms*(1)] *assms*(4,5)

obtain d'' **where** d'' : $d'' \in ?A$ $g d'' = d$ **by** *auto*

hence [*simp*]: $?A \neq \{\}$ **by** *blast*

define c' **where** $c' = \text{Inf } ?A$

define d' **where** $d' = \text{Sup } ?A$

have $?A \subseteq \{c'..d'\}$ **unfolding** *c'_def* *d'_def*

by (*intro* *subsetI*) (*auto* *intro*: *cInf_lower* *cSup_upper*)

moreover from *assms* **have** *closed* $?A$

using *continuous_on_closed_vimage*[*of* $\{a..b\}$ g] **by** (*subst* *Int_commute*) *simp*

hence $c' d'$ *in_set*: $c' \in ?A$ $d' \in ?A$ **unfolding** *c'_def* *d'_def*

by ((*intro* *closed_contains_Inf* *closed_contains_Sup*, *simp_all*[])**+**)

hence $\{c'..d'\} \subseteq ?A$ **using** *assms*

by (*intro* *subsetI*)

(*auto* *intro!*: *order_trans*[*of* c g c' g x **for** x] *order_trans*[*of* g x g d' d **for** x]
intro!: *mono*)

moreover have $c' \leq d'$ **using** $c' d'$ *in_set*(2) **unfolding** *c'_def* **by** (*intro* *cInf_lower*) *auto*

moreover have $g c' \leq c$ $g d' \geq d$

using $c'' d''$ *calculation* **by** (*metis* *IntE* *atLeastAtMost_iff* *mono* *order_class.order_eq_iff*)**+**

with $c' d'$ *in_set* **have** $g c' = c$ $g d' = d$

by *auto*

ultimately show *?thesis*

using *that* **by** *blast*

qed

7.4.1 Generic Borel spaces

definition (*in* *topological_space*) *borel* :: 'a *measure* **where**

borel = *sigma* *UNIV* $\{S. \text{open } S\}$

abbreviation *borel_measurable* $M \equiv \text{measurable } M \text{ borel}$

lemma *in_borel_measurable*:

$f \in \text{borel_measurable } M \longleftrightarrow$

$(\forall S \in \text{sigma_sets UNIV } \{S. \text{open } S\}. f - ' S \cap \text{space } M \in \text{sets } M)$
by (*auto simp add: measurable_def borel_def*)

lemma *in_borel_measurable_borel*:

$f \in \text{borel_measurable } M \longleftrightarrow (\forall S \in \text{sets borel}. f - ' S \cap \text{space } M \in \text{sets } M)$
by (*auto simp add: measurable_def borel_def*)

lemma *space_borel[simp]*: *space borel = UNIV*

unfolding *borel_def* **by** *auto*

lemma *space_in_borel[measurable]*: *UNIV ∈ sets borel*

unfolding *borel_def* **by** *auto*

lemma *sets_borel*: *sets borel = sigma_sets UNIV {S. open S}*

unfolding *borel_def* **by** (*rule sets_measure_of*) *simp*

lemma *measurable_sets_borel*:

$\llbracket f \in \text{measurable borel } M; A \in \text{sets } M \rrbracket \Longrightarrow f - ' A \in \text{sets borel}$
by (*drule (1) measurable_sets*) *simp*

lemma *pred_Collect_borel[measurable (raw)]*: *Measurable.pred borel P \Longrightarrow {x. P x} ∈ sets borel*

unfolding *borel_def pred_def* **by** *auto*

lemma *borel_open[measurable (raw generic)]*:

assumes *open A* **shows** *A ∈ sets borel*
by (*simp add: assms sets_borel*)

lemma *borel_closed[measurable (raw generic)]*:

assumes *closed A* **shows** *A ∈ sets borel*

proof –

have *space borel – (– A) ∈ sets borel*

using *assms* **unfolding** *closed_def* **by** (*blast intro: borel_open*)

thus *?thesis* **by** *simp*

qed

lemma *borel_singleton[measurable]*:

$A \in \text{sets borel} \Longrightarrow \text{insert } x A \in \text{sets (borel :: 'a::t1_space measure)}$

unfolding *insert_def* **by** (*rule sets.Un*) *auto*

lemma *sets_borel_eq_count_space*: *sets (borel :: 'a::{countable, t2_space} measure) = count_space UNIV*

by (*simp add: set_eq_iff sets.countable*)

lemma *borel_comp[measurable]*: *A ∈ sets borel \Longrightarrow – A ∈ sets borel*

unfolding *Compl_eq_Diff_UNIV* **by** *simp*

lemma *borel_measurable_vimage*:

fixes $f :: 'a \Rightarrow 'x::t2_space$

assumes $\text{borel}[\text{measurable}]$: $f \in \text{borel_measurable } M$
shows $f - \{x\} \cap \text{space } M \in \text{sets } M$
by *simp*

lemma *borel_measurableI*:
fixes $f :: 'a \Rightarrow 'x :: \text{topological_space}$
assumes $\bigwedge S. \text{open } S \implies f - \{S \cap \text{space } M\} \in \text{sets } M$
shows $f \in \text{borel_measurable } M$
unfolding *borel_def*
proof (*rule measurable_measure_of, simp_all*)
fix $S :: 'x \text{ set}$ **assume** $\text{open } S$ **thus** $f - \{S \cap \text{space } M\} \in \text{sets } M$
using *assms[of S]* **by** *simp*
qed

lemma *borel_measurable_const*:
 $(\lambda x. c) \in \text{borel_measurable } M$
by *auto*

lemma *borel_measurable_indicator*:
assumes $A: A \in \text{sets } M$
shows $\text{indicator } A \in \text{borel_measurable } M$
unfolding *indicator_def [abs_def]* **using** A
by (*auto intro!: measurable>If_set*)

lemma *borel_measurable_count_space*[*measurable (raw)*]:
 $f \in \text{borel_measurable } (\text{count_space } S)$
unfolding *measurable_def* **by** *auto*

lemma *borel_measurable_indicator'*[*measurable (raw)*]:
assumes [*measurable*]: $\{x \in \text{space } M. f x \in A\} \in \text{sets } M$
shows $(\lambda x. \text{indicator } (A) x (f x)) \in \text{borel_measurable } M$
unfolding *indicator_def [abs_def]*
by (*auto intro!: measurable>If*)

lemma *borel_measurable_indicator_iff*:
 $(\text{indicator } A :: 'a \Rightarrow 'x :: \{\text{t1_space, zero_neq_one}\}) \in \text{borel_measurable } M \iff$
 $A \cap \text{space } M \in \text{sets } M$
(is $?I \in \text{borel_measurable } M \iff _)$

proof
assume $?I \in \text{borel_measurable } M$
then have $?I - \{1\} \cap \text{space } M \in \text{sets } M$
unfolding *measurable_def* **by** *auto*
also have $?I - \{1\} \cap \text{space } M = A \cap \text{space } M$
unfolding *indicator_def [abs_def]* **by** *auto*
finally show $A \cap \text{space } M \in \text{sets } M$.
next
assume $A \cap \text{space } M \in \text{sets } M$
moreover have $?I \in \text{borel_measurable } M \iff$
 $(\text{indicator } (A \cap \text{space } M) :: 'a \Rightarrow 'x) \in \text{borel_measurable } M$

by (intro measurable_cong) (auto simp: indicator_def)
ultimately show $?I \in \text{borel_measurable } M$ by auto
qed

lemma borel_measurable_subalgebra:
assumes $\text{sets } N \subseteq \text{sets } M$ $\text{space } N = \text{space } M$ $f \in \text{borel_measurable } N$
shows $f \in \text{borel_measurable } M$
using *assms* **unfolding** *measurable_def* **by** auto

lemma borel_measurable_restrict_space_iff_ereal:
fixes $f :: 'a \Rightarrow \text{ereal}$
assumes $\Omega[\text{measurable, simp}]: \Omega \cap \text{space } M \in \text{sets } M$
shows $f \in \text{borel_measurable } (\text{restrict_space } M \ \Omega) \iff$
 $(\lambda x. f \ x * \text{indicator } \Omega \ x) \in \text{borel_measurable } M$
by (subst *measurable_restrict_space_iff*)
(auto simp: *indicator_def of_bool_def if_distrib* [where $f = \lambda x. a * x$ for a])
cong: *if_cong*)

lemma borel_measurable_restrict_space_iff_ennreal:
fixes $f :: 'a \Rightarrow \text{ennreal}$
assumes $\Omega[\text{measurable, simp}]: \Omega \cap \text{space } M \in \text{sets } M$
shows $f \in \text{borel_measurable } (\text{restrict_space } M \ \Omega) \iff$
 $(\lambda x. f \ x * \text{indicator } \Omega \ x) \in \text{borel_measurable } M$
by (subst *measurable_restrict_space_iff*)
(auto simp: *indicator_def of_bool_def if_distrib* [where $f = \lambda x. a * x$ for a])
cong: *if_cong*)

lemma borel_measurable_restrict_space_iff:
fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$
assumes $\Omega[\text{measurable, simp}]: \Omega \cap \text{space } M \in \text{sets } M$
shows $f \in \text{borel_measurable } (\text{restrict_space } M \ \Omega) \iff$
 $(\lambda x. \text{indicator } \Omega \ x *_R f \ x) \in \text{borel_measurable } M$
by (subst *measurable_restrict_space_iff*)
(auto simp: *indicator_def of_bool_def if_distrib* [where $f = \lambda x. x *_R a$ for a])
ac_simps
cong: *if_cong*)

lemma cbox_borel[measurable]: $\text{cbox } a \ b \in \text{sets borel}$
by (auto intro: *borel_closed*)

lemma box_borel[measurable]: $\text{box } a \ b \in \text{sets borel}$
by (auto intro: *borel_open*)

lemma borel_compact: $\text{compact } (A::'a::t2_space \ \text{set}) \implies A \in \text{sets borel}$
by (*simp add: borel_closed compact_imp_closed*)

lemma borel_sigma_sets_subset:
 $A \subseteq \text{sets borel} \implies \text{sigma_sets UNIV } A \subseteq \text{sets borel}$
using *sets.sigma_sets_subset* [of A *borel*] **by** *simp*

lemma borel_eq_sigmaI1:

fixes $F :: 'i \Rightarrow 'a::\text{topological_space set}$ **and** $X :: 'a::\text{topological_space set set}$
assumes $\text{borel_eq: borel} = \text{sigma UNIV } X$
assumes $X: \bigwedge x. x \in X \implies x \in \text{sets} (\text{sigma UNIV } (F \text{ ' } A))$
assumes $F: \bigwedge i. i \in A \implies F \ i \in \text{sets borel}$
shows $\text{borel} = \text{sigma UNIV } (F \text{ ' } A)$
unfolding borel_def
proof (*intro sigma_eqI antisym*)
have $\text{borel_rev_eq: sigma_sets UNIV } \{S::'a \text{ set. open } S\} = \text{sets borel}$
unfolding borel_def **by** *simp*
also have $\dots = \text{sigma_sets UNIV } X$
unfolding borel_eq **by** *simp*
also have $\dots \subseteq \text{sigma_sets UNIV } (F \text{ ' } A)$
using X **by** (*intro sigma_algebra.sigma_sets_subset[OF sigma_algebra_sigma_sets]*)
auto
finally show $\text{sigma_sets UNIV } \{S. \text{open } S\} \subseteq \text{sigma_sets UNIV } (F \text{ ' } A)$.
show $\text{sigma_sets UNIV } (F \text{ ' } A) \subseteq \text{sigma_sets UNIV } \{S. \text{open } S\}$
by (*metis F image_subset_iff sets_borel sigma_sets_mono*)
qed *auto*

lemma borel_eq_sigmaI2:

fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a::\text{topological_space set}$
and $G :: 'l \Rightarrow 'k \Rightarrow 'a::\text{topological_space set}$
assumes $\text{borel_eq: borel} = \text{sigma UNIV } ((\lambda(i, j). G \ i \ j) \text{ ' } B)$
assumes $X: \bigwedge i \ j. (i, j) \in B \implies G \ i \ j \in \text{sets} (\text{sigma UNIV } ((\lambda(i, j). F \ i \ j) \text{ ' } A))$
assumes $F: \bigwedge i \ j. (i, j) \in A \implies F \ i \ j \in \text{sets borel}$
shows $\text{borel} = \text{sigma UNIV } ((\lambda(i, j). F \ i \ j) \text{ ' } A)$
using *assms*
by (*smt (verit, del_insts) borel_eq_sigmaI1 image_iff prod.collapse split_beta*)

lemma borel_eq_sigmaI3:

fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a::\text{topological_space set}$ **and** $X :: 'a::\text{topological_space set set}$
assumes $\text{borel_eq: borel} = \text{sigma UNIV } X$
assumes $X: \bigwedge x. x \in X \implies x \in \text{sets} (\text{sigma UNIV } ((\lambda(i, j). F \ i \ j) \text{ ' } A))$
assumes $F: \bigwedge i \ j. (i, j) \in A \implies F \ i \ j \in \text{sets borel}$
shows $\text{borel} = \text{sigma UNIV } ((\lambda(i, j). F \ i \ j) \text{ ' } A)$
using *assms* **by** (*intro borel_eq_sigmaI1[where X=X and F=(\lambda(i, j). F i j)]*)
auto

lemma borel_eq_sigmaI4:

fixes $F :: 'i \Rightarrow 'a::\text{topological_space set}$
and $G :: 'l \Rightarrow 'k \Rightarrow 'a::\text{topological_space set}$
assumes $\text{borel_eq: borel} = \text{sigma UNIV } ((\lambda(i, j). G \ i \ j) \text{ ' } A)$
assumes $X: \bigwedge i \ j. (i, j) \in A \implies G \ i \ j \in \text{sets} (\text{sigma UNIV } (\text{range } F))$
assumes $F: \bigwedge i. F \ i \in \text{sets borel}$
shows $\text{borel} = \text{sigma UNIV } (\text{range } F)$
using *assms* **by** (*intro borel_eq_sigmaI1[where X=(\lambda(i, j). G i j) ' A and*

2184

$F=F]$) *auto*

lemma *borel_eq_sigmaI5*:

fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a::\text{topological_space set}$ **and** $G :: 'l \Rightarrow 'a::\text{topological_space set}$

assumes *borel_eq*: $\text{borel} = \text{sigma UNIV (range G)}$

assumes $X: \bigwedge i. G i \in \text{sets (sigma UNIV (range (\lambda(i, j). F i j)))}$

assumes $F: \bigwedge i j. F i j \in \text{sets borel}$

shows $\text{borel} = \text{sigma UNIV (range (\lambda(i, j). F i j))}$

using *assms* **by** (*intro borel_eq_sigmaI1*[**where** $X=\text{range G}$ **and** $F=(\lambda(i, j). F i j)$]) *auto*

theorem *second_countable_borel_measurable*:

fixes $X :: 'a::\text{second_countable_topology set set}$

assumes *eq*: $\text{open} = \text{generate_topology X}$

shows $\text{borel} = \text{sigma UNIV X}$

unfolding *borel_def*

proof (*intro sigma_eqI sigma_sets_eqI*)

interpret $X: \text{sigma_algebra UNIV sigma_sets UNIV X}$

by (*rule sigma_algebra_sigma_sets simp*)

fix $S :: 'a \text{ set}$ **assume** $S \in \text{Collect open}$

then have $\text{generate_topology X S}$

by (*auto simp: eq*)

then show $S \in \text{sigma_sets UNIV X}$

proof *induction*

case ($UN K$)

then have $K: \bigwedge k. k \in K \implies \text{open } k$

unfolding *eq* **by** *auto*

from *ex_countable_basis* **obtain** $B :: 'a \text{ set set}$ **where**

$B: \bigwedge b. b \in B \implies \text{open } b \bigwedge X. \text{open } X \implies \exists b \subseteq B. (\bigcup b) = X$ **and** *countable*

B

by (*auto simp: topological_basis_def*)

from $B(2)[OF K]$ **obtain** m **where** $m: \bigwedge k. k \in K \implies m k \subseteq B \bigwedge k. k \in K \implies \bigcup (m k) = k$

by *metis*

define U **where** $U = (\bigcup k \in K. m k)$

with m **have** *countable U*

by (*intro countable_subset[OF _ <countable B>]*) *auto*

have $\bigcup U = (\bigcup A \in U. A)$ **by** *simp*

also have $\dots = \bigcup K$

unfolding *U_def UN_simps* **by** (*simp add: m*)

finally have $\bigcup U = \bigcup K$.

have $\forall b \in U. \exists k \in K. b \subseteq k$

using m **by** (*auto simp: U_def*)

then obtain u **where** $u: \bigwedge b. b \in U \implies u b \in K$ **and** $\bigwedge b. b \in U \implies b \subseteq u b$

by *metis*

then have $(\bigcup b \in U. u b) \subseteq \bigcup K \cup U \subseteq (\bigcup b \in U. u b)$

```

    by auto
  then have  $\bigcup K = (\bigcup b \in U. u\ b)$ 
    unfolding  $\langle \bigcup U = \bigcup K \rangle$  by auto
  also have  $\dots \in \text{sigma\_sets UNIV } X$ 
    using  $u\ UN$  by (intro  $X.\text{countable\_UN}' \langle \text{countable } U \rangle$ ) auto
  finally show  $\bigcup K \in \text{sigma\_sets UNIV } X$  .
qed auto
qed (auto simp: eq intro: generate_topology.Basis)

lemma borel_eq_closed: borel = sigma UNIV (Collect closed)
proof -
  have  $x \in \text{sigma\_sets UNIV (Collect closed)}$ 
    if open  $x$  for  $x :: 'a\ \text{set}$ 
  by (metis that Compl_eq_Diff_UNIV closed_Compl double_complement mem_Collect_eq

      sigma_sets.Basic sigma_sets.Compl)
  then show ?thesis
    unfolding borel_def
    by (metis Pow_UNIV borel_closed mem_Collect_eq sets_borel sigma_eqI
sigma_sets_eqI top_greatest)
qed

proposition borel_eq_countable_basis:
  fixes  $B :: 'a :: \text{topological\_space set set}$ 
  assumes countable  $B$ 
  assumes topological_basis  $B$ 
  shows borel = sigma UNIV  $B$ 
  unfolding borel_def
proof (intro sigma_eqI sigma_sets_eqI, safe)
  interpret countable_basis open  $B$  using assms by (rule countable_basis_openI)
  fix  $X :: 'a\ \text{set}$  assume open  $X$ 
  from open_countable_basisE[OF this] obtain  $B'$  where  $B' : B' \subseteq B\ X = \bigcup B'$ 
  .
  then show  $X \in \text{sigma\_sets UNIV } B$ 
    by (blast intro: sigma_sets_UNION  $\langle \text{countable } B \rangle$  countable_subset)
next
  fix  $b$  assume  $b \in B$ 
  hence open  $b$  by (rule topological_basis_open[OF assms(2)])
  thus  $b \in \text{sigma\_sets UNIV (Collect open)}$  by auto
qed simp_all

lemma borel_measurable_continuous_on_restrict:
  fixes  $f :: 'a :: \text{topological\_space} \Rightarrow 'b :: \text{topological\_space}$ 
  assumes  $f : \text{continuous\_on } A\ f$ 
  shows  $f \in \text{borel\_measurable (restrict\_space borel } A)$ 
proof (rule borel_measurableI)
  fix  $S :: 'b\ \text{set}$  assume open  $S$ 
  with  $f$  obtain  $T$  where  $f - ' S \cap A = T \cap A$  open  $T$ 
    by (metis continuous_on_open_invariant)

```

then show $f - 'S \cap \text{space } (\text{restrict_space } \text{borel } A) \in \text{sets } (\text{restrict_space } \text{borel } A)$
by (*force simp add: sets_restrict_space space_restrict_space*)
qed

lemma *borel_measurable_continuous_onI*: $\text{continuous_on } UNIV f \implies f \in \text{borel_measurable } \text{borel}$
by (*drule borel_measurable_continuous_on_restrict*) *simp*

lemma *borel_measurable_continuous_on_if*:
 $A \in \text{sets } \text{borel} \implies \text{continuous_on } A f \implies \text{continuous_on } (- A) g \implies$
 $(\lambda x. \text{if } x \in A \text{ then } f x \text{ else } g x) \in \text{borel_measurable } \text{borel}$
by (*auto simp add: measurable_If_restrict_space_iff Collect_neg_eq intro!: borel_measurable_continuous_on_restrict*)

lemma *borel_measurable_continuous_countable_exceptions*:
fixes $f :: 'a::t1_space \Rightarrow 'b::\text{topological_space}$
assumes $X: \text{countable } X$
assumes $\text{continuous_on } (- X) f$
shows $f \in \text{borel_measurable } \text{borel}$
proof (*rule measurable_discrete_difference[OF _ X]*)
have $X \in \text{sets } \text{borel}$
by (*rule sets.countable[OF _ X]*) *auto*
then show $(\lambda x. \text{if } x \in X \text{ then undefined else } f x) \in \text{borel_measurable } \text{borel}$
by (*intro borel_measurable_continuous_on_if assms continuous_intros*)
qed *auto*

lemma *borel_measurable_continuous_on*:
assumes $f: \text{continuous_on } UNIV f$ **and** $g: g \in \text{borel_measurable } M$
shows $(\lambda x. f (g x)) \in \text{borel_measurable } M$
using *measurable_comp[OF g borel_measurable_continuous_onI[OF f]]* **by** (*simp add: comp_def*)

lemma *borel_measurable_continuous_on_indicator*:
fixes $f g :: 'a::\text{topological_space} \Rightarrow 'b::\text{real_normed_vector}$
shows $A \in \text{sets } \text{borel} \implies \text{continuous_on } A f \implies (\lambda x. \text{indicator } A x *_R f x) \in \text{borel_measurable } \text{borel}$
using *borel_measurable_continuous_on_restrict borel_measurable_restrict_space_iff inf_top.right_neutral* **by** *blast*

lemma *borel_measurable_Pair*[*measurable (raw)*]:
fixes $f :: 'a \Rightarrow 'b::\text{second_countable_topology}$ **and** $g :: 'a \Rightarrow 'c::\text{second_countable_topology}$
assumes $f[\text{measurable}]: f \in \text{borel_measurable } M$
assumes $g[\text{measurable}]: g \in \text{borel_measurable } M$
shows $(\lambda x. (f x, g x)) \in \text{borel_measurable } M$
proof (*subst borel_eq_countable_basis*)
let $?B = \text{SOME } B::'b \text{ set set. countable } B \wedge \text{topological_basis } B$
let $?C = \text{SOME } B::'c \text{ set set. countable } B \wedge \text{topological_basis } B$
let $?P = (\lambda(b, c). b \times c) - ' (?B \times ?C)$

```

show countable ?P topological_basis ?P
  by (auto intro!: countable_basis_topological_basis_prod_is_basis)

show ( $\lambda x. (f x, g x) \in \text{measurable } M \text{ (sigma UNIV ?P)}$ )
proof (rule measurable_measure_of)
  fix S assume S  $\in$  ?P
  then obtain b c where b  $\in$  ?B c  $\in$  ?C and S: S = b  $\times$  c by auto
  then have borel: open b open c
    by (auto intro: is_basis_topological_basis_open)
  have ( $\lambda x. (f x, g x) - ' S \cap \text{space } M = (f - ' b \cap \text{space } M) \cap (g - ' c \cap \text{space } M)$ )
proof
  unfolding S by auto
  also have ...  $\in$  sets M
  using borel by simp
  finally show ( $\lambda x. (f x, g x) - ' S \cap \text{space } M \in \text{sets } M$  .
qed auto
qed

```

```

lemma borel_measurable_continuous_Pair:
fixes f :: 'a  $\Rightarrow$  'b::second_countable_topology and g :: 'a  $\Rightarrow$  'c::second_countable_topology
assumes [measurable]: f  $\in$  borel_measurable M
assumes [measurable]: g  $\in$  borel_measurable M
assumes H: continuous_on UNIV ( $\lambda x. H (fst x) (snd x)$ )
shows ( $\lambda x. H (f x) (g x) \in \text{borel\_measurable } M$ )
proof -
  have eq: ( $\lambda x. H (f x) (g x) = (\lambda x. (\lambda x. H (fst x) (snd x)) (f x, g x)$ ) by auto
  show ?thesis
  unfolding eq by (rule borel_measurable_continuous_on[OF H]) auto
qed

```

7.4.2 Borel spaces on order topologies

```

lemma [measurable]:
fixes a b :: 'a::linorder_topology
shows lessThan_borel: {.. $a$ }  $\in$  sets borel
  and greaterThan_borel: { $a$  <..}  $\in$  sets borel
  and greaterThanLessThan_borel: { $a$  <.. $b$ }  $\in$  sets borel
  and atMost_borel: {.. $a$ }  $\in$  sets borel
  and atLeast_borel: { $a$ ..}  $\in$  sets borel
  and atLeastAtMost_borel: { $a$ .. $b$ }  $\in$  sets borel
  and greaterThanAtMost_borel: { $a$  <.. $b$ }  $\in$  sets borel
  and atLeastLessThan_borel: { $a$ .. $b$ }  $\in$  sets borel
unfolding greaterThanAtMost_def atLeastLessThan_def
by (blast intro: borel_open borel_closed open_lessThan open_greaterThan open_greaterThanLessThan
  closed_atMost closed_atLeast closed_atLeastAtMost)+

```

```

lemma borel_Iio:
  borel = sigma UNIV (range lessThan :: 'a::{\linorder_topology, second_countable_topology}
  set set)

```

```

unfolding second_countable_borel_measurable[OF open_generated_order]
proof (intro sigma_eqI sigma_sets_eqI)
  obtain D :: 'a set where D: countable D  $\wedge$  X. open X  $\implies$  X  $\neq$  {}  $\implies$   $\exists d \in D$ .
  d  $\in$  X
  by (rule countable_dense_setE) blast

```

```

interpret L: sigma_algebra UNIV sigma_sets UNIV (range lessThan)
  by (rule sigma_algebra_sigma_sets) simp

```

```

fix A :: 'a set assume A  $\in$  range lessThan  $\cup$  range greaterThan
then obtain y where A = {y <..<}  $\vee$  A = {..< y}
  by blast

```

```

then show A  $\in$  sigma_sets UNIV (range lessThan)

```

```

proof

```

```

  assume A: A = {y <..<}

```

```

  show ?thesis

```

```

  proof cases

```

```

    assume  $\forall x > y$ .  $\exists d$ . y < d  $\wedge$  d < x

```

```

    with D(2)[of {y <..< x} for x] have  $\forall x > y$ .  $\exists d \in D$ . y < d  $\wedge$  d < x

```

```

    by (auto simp: set_eq_iff)

```

```

    then have A = UNIV - ( $\bigcap$  d  $\in$  {d  $\in$  D. y < d}. {..< d})

```

```

    by (auto simp: A) (metis less_asym)

```

```

    also have ...  $\in$  sigma_sets UNIV (range lessThan)

```

```

    using D(1) by (intro L.Diff L.top L.countable_INT'') auto

```

```

    finally show ?thesis .

```

```

  next

```

```

    assume  $\neg$  ( $\forall x > y$ .  $\exists d$ . y < d  $\wedge$  d < x)

```

```

    then obtain x where y < x  $\wedge$  d. y < d  $\implies$   $\neg$  d < x

```

```

    by auto

```

```

    then have A = UNIV - {..< x}

```

```

    unfolding A by (auto simp: not_less[symmetric])

```

```

    also have ...  $\in$  sigma_sets UNIV (range lessThan)

```

```

    by auto

```

```

    finally show ?thesis .

```

```

  qed

```

```

qed auto

```

```

qed auto

```

```

lemma borel_Ioi:

```

```

  borel = sigma UNIV (range greaterThan :: 'a::{\linorder_topology, second_countable_topology}
  set set)

```

```

  unfolding second_countable_borel_measurable[OF open_generated_order]

```

```

proof (intro sigma_eqI sigma_sets_eqI)

```

```

  obtain D :: 'a set where D: countable D  $\wedge$  X. open X  $\implies$  X  $\neq$  {}  $\implies$   $\exists d \in D$ .
  d  $\in$  X

```

```

  by (rule countable_dense_setE) blast

```

```

interpret L: sigma_algebra UNIV sigma_sets UNIV (range greaterThan)

```

```

  by (rule sigma_algebra_sigma_sets) simp

```



```

fix A :: 'a set assume A ∈ range lessThan ∪ range greaterThan
then obtain y where A = {y <..} ∨ A = {..< y}
  by blast
then show A ∈ sigma_sets UNIV (range greaterThan)
proof
  assume A: A = {..< y}
  show ?thesis
  proof cases
    assume ∀x<y. ∃d. x < d ∧ d < y
    with D(2)[of {x <..< y} for x] have ∀x<y. ∃d∈D. x < d ∧ d < y
      by (auto simp: set_eq_iff)
    then have A = UNIV - (∩ d∈{d∈D. d < y}. {d <..})
      by (auto simp: A) (metis less_asym)
    also have ... ∈ sigma_sets UNIV (range greaterThan)
      using D(1) by (intro L.Diff L.top L.countable_INT'') auto
    finally show ?thesis .
  next
    assume ¬ (∀x<y. ∃d. x < d ∧ d < y)
    then obtain x where x < y ∧ d. y > d ⇒ x ≥ d
      by (auto simp: not_less[symmetric])
    then have A = UNIV - {x <..}
      unfolding A Compl_eq_Diff_UNIV[symmetric] by auto
    also have ... ∈ sigma_sets UNIV (range greaterThan)
      by auto
    finally show ?thesis .
  qed
qed auto
qed auto

lemma borel_measurableI_less:
  fixes f :: 'a ⇒ 'b::{linorder_topology, second_countable_topology}
  shows (∧y. {x∈space M. f x < y} ∈ sets M) ⇒ f ∈ borel_measurable M
  unfolding borel_Ioi
  by (rule measurable_measure_of) (auto simp: Int_def conj_commute)

lemma borel_measurableI_greater:
  fixes f :: 'a ⇒ 'b::{linorder_topology, second_countable_topology}
  shows (∧y. {x∈space M. y < f x} ∈ sets M) ⇒ f ∈ borel_measurable M
  unfolding borel_Ioi
  by (rule measurable_measure_of) (auto simp: Int_def conj_commute)

lemma borel_measurableI_le:
  fixes f :: 'a ⇒ 'b::{linorder_topology, second_countable_topology}
  shows (∧y. {x∈space M. f x ≤ y} ∈ sets M) ⇒ f ∈ borel_measurable M
  by (rule borel_measurableI_greater) (auto simp: not_le[symmetric])

lemma borel_measurableI_ge:
  fixes f :: 'a ⇒ 'b::{linorder_topology, second_countable_topology}

```

shows $(\bigwedge y. \{x \in \text{space } M. y \leq f x\} \in \text{sets } M) \implies f \in \text{borel_measurable } M$
by (rule borel_measurableI_less) (auto simp: not_le[symmetric])

lemma borel_measurable_less[measurable]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, linorder_topology}\}$

assumes $f \in \text{borel_measurable } M$

assumes $g \in \text{borel_measurable } M$

shows $\{w \in \text{space } M. f w < g w\} \in \text{sets } M$

proof –

have $\{w \in \text{space } M. f w < g w\} = (\lambda x. (f x, g x)) - ' \{x. \text{fst } x < \text{snd } x\} \cap \text{space } M$

by auto

also have $\dots \in \text{sets } M$

by (intro measurable_sets[OF borel_measurable_Pair borel_open, OF assms open_Collect_less]

continuous_intros)

finally show ?thesis .

qed

lemma

fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, linorder_topology}\}$

assumes $f[\text{measurable}]: f \in \text{borel_measurable } M$

assumes $g[\text{measurable}]: g \in \text{borel_measurable } M$

shows borel_measurable_le[measurable]: $\{w \in \text{space } M. f w \leq g w\} \in \text{sets } M$

and borel_measurable_eq[measurable]: $\{w \in \text{space } M. f w = g w\} \in \text{sets } M$

and borel_measurable_neq: $\{w \in \text{space } M. f w \neq g w\} \in \text{sets } M$

unfolding eq_iff_not_less[symmetric]

by measurable

lemma borel_measurable_SUP[measurable (raw)]:

fixes $F :: _ \Rightarrow _ \Rightarrow _ :: \{\text{complete_linorder, linorder_topology, second_countable_topology}\}$

assumes [simp]: countable I

assumes [measurable]: $\bigwedge i. i \in I \implies F i \in \text{borel_measurable } M$

shows $(\lambda x. \text{SUP } i \in I. F i x) \in \text{borel_measurable } M$

by (rule borel_measurableI_greater) (simp add: less_SUP_iff)

lemma borel_measurable_INF[measurable (raw)]:

fixes $F :: _ \Rightarrow _ \Rightarrow _ :: \{\text{complete_linorder, linorder_topology, second_countable_topology}\}$

assumes [simp]: countable I

assumes [measurable]: $\bigwedge i. i \in I \implies F i \in \text{borel_measurable } M$

shows $(\lambda x. \text{INF } i \in I. F i x) \in \text{borel_measurable } M$

by (rule borel_measurableI_less) (simp add: INF_less_iff)

lemma borel_measurable_cSUP[measurable (raw)]:

fixes $F :: _ \Rightarrow _ \Rightarrow 'a :: \{\text{conditionally_complete_linorder, linorder_topology, second_countable_topology}\}$

assumes [simp]: countable I

assumes [measurable]: $\bigwedge i. i \in I \implies F i \in \text{borel_measurable } M$

assumes bdd: $\bigwedge x. x \in \text{space } M \implies \text{bdd_above } ((\lambda i. F i x) ' I)$

```

  shows  $(\lambda x. \text{SUP } i \in I. F \ i \ x) \in \text{borel\_measurable } M$ 
proof cases
  assume  $I = \{\}$  then show ?thesis
    by (simp add: borel_measurable_const)
next
  assume  $I \neq \{\}$ 
  show ?thesis
  proof (rule borel_measurableI_le)
    fix  $y$ 
    have  $\{x \in \text{space } M. \forall i \in I. F \ i \ x \leq y\} \in \text{sets } M$ 
      by measurable
    also have  $\{x \in \text{space } M. \forall i \in I. F \ i \ x \leq y\} = \{x \in \text{space } M. (\text{SUP } i \in I. F \ i \ x) \leq y\}$ 
      by (simp add: cSUP_le_iff  $\langle I \neq \{\} \rangle$  bdd cong: conj_cong)
    finally show  $\{x \in \text{space } M. (\text{SUP } i \in I. F \ i \ x) \leq y\} \in \text{sets } M$  .
  qed
qed

```

```

lemma borel_measurable_cINF[measurable (raw)]:
  fixes  $F :: \_ \Rightarrow \_ \Rightarrow 'a::\{\text{conditionally\_complete\_linorder, linorder\_topology, second\_countable\_topology}\}$ 
  assumes [simp]: countable I
  assumes [measurable]:  $\bigwedge i. i \in I \implies F \ i \in \text{borel\_measurable } M$ 
  assumes bdd:  $\bigwedge x. x \in \text{space } M \implies \text{bdd\_below } ((\lambda i. F \ i \ x) \ ` I)$ 
  shows  $(\lambda x. \text{INF } i \in I. F \ i \ x) \in \text{borel\_measurable } M$ 
proof cases
  assume  $I = \{\}$  then show ?thesis
    by (simp add: borel_measurable_const)
next
  assume  $I \neq \{\}$ 
  show ?thesis
  proof (rule borel_measurableI_ge)
    fix  $y$ 
    have  $\{x \in \text{space } M. \forall i \in I. y \leq F \ i \ x\} \in \text{sets } M$ 
      by measurable
    also have  $\{x \in \text{space } M. \forall i \in I. y \leq F \ i \ x\} = \{x \in \text{space } M. y \leq (\text{INF } i \in I. F \ i \ x)\}$ 
      by (simp add: le_cINF_iff  $\langle I \neq \{\} \rangle$  bdd cong: conj_cong)
    finally show  $\{x \in \text{space } M. y \leq (\text{INF } i \in I. F \ i \ x)\} \in \text{sets } M$  .
  qed
qed

```

```

lemma borel_measurable_lfp[consumes 1, case_names continuity step]:
  fixes  $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)::\{\text{complete\_linorder, linorder\_topology, second\_countable\_topology}\}$ 
  assumes sup_continuous F
  assumes *:  $\bigwedge f. f \in \text{borel\_measurable } M \implies F \ f \in \text{borel\_measurable } M$ 
  shows  $\text{lfp } F \in \text{borel\_measurable } M$ 
proof –

```

```

{ fix i have ((F ^^ i) bot) ∈ borel_measurable M
  by (induct i) (auto intro!: *) }
then have (λx. SUP i. (F ^^ i) bot x) ∈ borel_measurable M
  by measurable
also have (λx. SUP i. (F ^^ i) bot x) = (SUP i. (F ^^ i) bot)
  by (auto simp add: image_comp)
also have (SUP i. (F ^^ i) bot) = lfp F
  by (rule sup_continuous_lfp[symmetric]) fact
finally show ?thesis .
qed

```

lemma *borel_measurable_gfp*[consumes 1, case_names continuity step]:
 fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{\text{complete_linorder, linorder_topology, second_countable_topology}\})$
 assumes *inf_continuous* F
 assumes *: $\bigwedge f. f \in \text{borel_measurable } M \implies F f \in \text{borel_measurable } M$
 shows $\text{gfp } F \in \text{borel_measurable } M$
proof –
 { fix i have ((F ^^ i) top) ∈ borel_measurable M
 by (induct i) (auto intro!: * simp: bot_fun_def) }
 then have (λx. INF i. (F ^^ i) top x) ∈ borel_measurable M
 by measurable
 also have (λx. INF i. (F ^^ i) top x) = (INF i. (F ^^ i) top)
 by (auto simp add: image_comp)
 also have ... = gfp F
 by (rule inf_continuous_gfp[symmetric]) fact
 finally show ?thesis .
qed

lemma *borel_measurable_max*[measurable (raw)]:
 $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies (\lambda x. \max (g x) (f x))$
 $:: 'b::\{\text{second_countable_topology, linorder_topology}\} \in \text{borel_measurable } M$
 by (rule borel_measurableI_less) simp

lemma *borel_measurable_min*[measurable (raw)]:
 $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies (\lambda x. \min (g x) (f x))$
 $:: 'b::\{\text{second_countable_topology, linorder_topology}\} \in \text{borel_measurable } M$
 by (rule borel_measurableI_greater) simp

lemma *borel_measurable_Min*[measurable (raw)]:
 $\text{finite } I \implies (\bigwedge i. i \in I \implies f i \in \text{borel_measurable } M) \implies (\lambda x. \text{Min } ((\lambda i. f i x) 'I))$
 $:: 'b::\{\text{second_countable_topology, linorder_topology}\} \in \text{borel_measurable } M$
proof (induct I rule: finite_induct)
 case (insert i I) then show ?case
 by (cases I = {}) auto
qed auto

lemma *borel_measurable_Max*[measurable (raw)]:
 $\text{finite } I \implies (\bigwedge i. i \in I \implies f i \in \text{borel_measurable } M) \implies (\lambda x. \text{Max } ((\lambda i. f$

$i\ x) \cdot I) :: 'b :: \{second_countable_topology, linorder_topology\} \in borel_measurable\ M$

proof (induct I rule: finite_induct)
 case (insert i I) then show ?case
 by (cases I = {}) auto
qed auto

lemma borel_measurable_sup[measurable (raw)]:

$f \in borel_measurable\ M \implies g \in borel_measurable\ M \implies (\lambda x. sup\ (g\ x)\ (f\ x)) :: 'b :: \{lattice, second_countable_topology, linorder_topology\} \in borel_measurable\ M$

unfolding sup_max **by** measurable

lemma borel_measurable_inf[measurable (raw)]:

$f \in borel_measurable\ M \implies g \in borel_measurable\ M \implies (\lambda x. inf\ (g\ x)\ (f\ x)) :: 'b :: \{lattice, second_countable_topology, linorder_topology\} \in borel_measurable\ M$

unfolding inf_min **by** measurable

lemma [measurable (raw)]:

fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: \{complete_linorder, second_countable_topology, linorder_topology\}$

assumes $\bigwedge i. f\ i \in borel_measurable\ M$

shows borel_measurable_liminf: $(\lambda x. liminf\ (\lambda i. f\ i\ x)) \in borel_measurable\ M$

and borel_measurable_limsup: $(\lambda x. limsup\ (\lambda i. f\ i\ x)) \in borel_measurable\ M$

unfolding liminf_SUP_INF limsup_INF_SUP **using** assms **by** auto

lemma measurable_convergent[measurable (raw)]:

fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: \{complete_linorder, second_countable_topology, linorder_topology\}$

assumes [measurable]: $\bigwedge i. f\ i \in borel_measurable\ M$

shows Measurable.pred M $(\lambda x. convergent\ (\lambda i. f\ i\ x))$

unfolding convergent_ereal **by** measurable

lemma sets_Collect_convergent[measurable]:

fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: \{complete_linorder, second_countable_topology, linorder_topology\}$

assumes $f[measurable]: \bigwedge i. f\ i \in borel_measurable\ M$

shows $\{x \in space\ M. convergent\ (\lambda i. f\ i\ x)\} \in sets\ M$

by measurable

lemma borel_measurable_lim[measurable (raw)]:

fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: \{complete_linorder, second_countable_topology, linorder_topology\}$

assumes [measurable]: $\bigwedge i. f\ i \in borel_measurable\ M$

shows $(\lambda x. lim\ (\lambda i. f\ i\ x)) \in borel_measurable\ M$

proof –

have $\bigwedge x. lim\ (\lambda i. f\ i\ x) = (if\ convergent\ (\lambda i. f\ i\ x)\ then\ limsup\ (\lambda i. f\ i\ x)\ else\ (THE\ i. False))$

by (simp add: lim_def convergent_def convergent_limsup_cl)

then show ?thesis

by simp

qed

lemma *borel_measurable_LIMSEQ_order*:
fixes $u :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{complete_linorder, second_countable_topology, linorder_topology}\}$
assumes $u': \bigwedge x. x \in \text{space } M \implies (\lambda i. u \ i \ x) \longrightarrow u' \ x$
and $u: \bigwedge i. u \ i \in \text{borel_measurable } M$
shows $u' \in \text{borel_measurable } M$
proof –
have $\bigwedge x. x \in \text{space } M \implies u' \ x = \text{liminf } (\lambda n. u \ n \ x)$
using u' **by** (*simp add: lim_imp_Liminf[symmetric]*)
with u **show** *?thesis* **by** (*simp cong: measurable_cong*)
qed

7.4.3 Borel spaces on topological monoids

lemma *borel_measurable_add*[*measurable (raw)*]:
fixes $f \ g :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, topological_monoid_add}\}$
assumes $f: f \in \text{borel_measurable } M$
assumes $g: g \in \text{borel_measurable } M$
shows $(\lambda x. f \ x + g \ x) \in \text{borel_measurable } M$
using $f \ g$ **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_sum*[*measurable (raw)*]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b :: \{\text{second_countable_topology, topological_comm_monoid_add}\}$
assumes $\bigwedge i. i \in S \implies f \ i \in \text{borel_measurable } M$
shows $(\lambda x. \sum_{i \in S}. f \ i \ x) \in \text{borel_measurable } M$
proof *cases*
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel_measurable_suminf_order*[*measurable (raw)*]:
fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{complete_linorder, second_countable_topology, linorder_topology, topological_comm_monoid_add}\}$
assumes $f[\text{measurable}]: \bigwedge i. f \ i \in \text{borel_measurable } M$
shows $(\lambda x. \text{suminf } (\lambda i. f \ i \ x)) \in \text{borel_measurable } M$
unfolding *suminf_def sums_def*[*abs_def*] *lim_def*[*symmetric*] **by** *simp*

7.4.4 Borel spaces on Euclidean spaces

lemma *borel_measurable_inner*[*measurable (raw)*]:
fixes $f \ g :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, real_inner}\}$
assumes $f \in \text{borel_measurable } M$
assumes $g \in \text{borel_measurable } M$
shows $(\lambda x. f \ x \cdot g \ x) \in \text{borel_measurable } M$
using *assms*
by (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

notation

eucl_less (**infix** $<e$ 50)

lemma *box_oc*: $\{x. a <e \ x \wedge x \leq b\} = \{x. a <e \ x\} \cap \{..b\}$

and *box_co*: $\{x. a \leq x \wedge x < e\ b\} = \{a..\} \cap \{x. x < e\ b\}$
by *auto*

lemma *eucl_ivals*[*measurable*]:
fixes *a b* :: 'a::ordered_euclidean_space
shows $\{x. x < e\ a\} \in \text{sets borel}$
and $\{x. a < e\ x\} \in \text{sets borel}$
and $\{..a\} \in \text{sets borel}$
and $\{a..\} \in \text{sets borel}$
and $\{a..b\} \in \text{sets borel}$
and $\{x. a < e\ x \wedge x \leq b\} \in \text{sets borel}$
and $\{x. a \leq x \wedge x < e\ b\} \in \text{sets borel}$
unfolding *box_oc box_co*
by (*auto intro: borel_open borel_closed*)

lemma
fixes *i* :: 'a::{second_countable_topology, real_inner}
shows *halfspace_less_borel*: $\{x. a < x \cdot i\} \in \text{sets borel}$
and *halfspace_greater_borel*: $\{x. x \cdot i < a\} \in \text{sets borel}$
and *halfspace_less_eq_borel*: $\{x. a \leq x \cdot i\} \in \text{sets borel}$
and *halfspace_greater_eq_borel*: $\{x. x \cdot i \leq a\} \in \text{sets borel}$
by *simp_all*

lemma *borel_eq_box*:
borel = sigma UNIV (range ($\lambda (a, b). \text{box } a\ b :: 'a :: \text{euclidean_space set}$))
(is _ = ?SIGMA)
proof (*rule borel_eq_sigmaI1[OF borel_def]*)
fix *M* :: 'a set **assume** *M* $\in \{S. \text{open } S\}$
then have *open M* **by** *simp*
show *M* $\in ?SIGMA$
apply (*subst open_UNION_box[OF <open M>*)
apply (*safe intro!: sets.countable_UN' countable_PiE countable_Collect*)
apply (*auto intro: countable_rat*)
done
qed (*auto simp: box_def*)

lemma *halfspace_gt_in_halfspace*:
assumes *i*: $i \in A$
shows $\{x::'a. a < x \cdot i\} \in$
*sigma_sets UNIV (($\lambda (a, i). \{x::'a::\text{euclidean_space}. x \cdot i < a\}$) ' (*UNIV* \times
A))*
(is ?set $\in ?SIGMA$)
proof –
interpret *sigma_algebra UNIV ?SIGMA*
by (*intro sigma_algebra_sigma_sets*) *simp_all*
have *: *?set* = $(\bigcup n. \text{UNIV} - \{x::'a. x \cdot i < a + 1 / \text{real } (\text{Suc } n)\})$
proof (*safe, simp_all add: not_less del: of_nat_Suc*)
fix *x* :: 'a **assume** $a < x \cdot i$
with *reals_Archimedean*[*of* $x \cdot i - a$]

```

    obtain  $n$  where  $a + 1 / \text{real } (\text{Suc } n) < x \cdot i$ 
      by (auto simp: field_simps)
    then show  $\exists n. a + 1 / \text{real } (\text{Suc } n) \leq x \cdot i$ 
      by (blast intro: less_imp_le)
  next
    fix  $x$   $n$ 
    have  $a < a + 1 / \text{real } (\text{Suc } n)$  by auto
    also assume  $\dots \leq x$ 
    finally show  $a < x$  .
  qed
  show  $?set \in ?SIGMA$  unfolding *
    by (auto intro!: Diff sigma_sets_Inter i)
qed

lemma borel_eq_halfspace_less:
  borel = sigma UNIV (( $\lambda(a, i). \{x::'a::\text{euclidean\_space}. x \cdot i < a\}$ ) ' ( $UNIV \times$ 
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_box])
  fix  $a$   $b :: 'a$ 
  have  $\text{box } a \ b = \{x \in \text{space } ?SIGMA. \forall i \in \text{Basis}. a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i\}$ 
    by (auto simp: box_def)
  also have  $\dots \in \text{sets } ?SIGMA$ 
    by (intro sets.sets_Collect_conj sets.sets_Collect_finite_All sets.sets_Collect_const)
      (auto intro!: halfspace_gt_in_halfspace countable_PiE countable_rat)
  finally show  $\text{box } a \ b \in \text{sets } ?SIGMA$  .
qed auto

lemma borel_eq_halfspace_le:
  borel = sigma UNIV (( $\lambda(a, i). \{x::'a::\text{euclidean\_space}. x \cdot i \leq a\}$ ) ' ( $UNIV \times$ 
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix  $a :: \text{real}$  and  $i :: 'a$  assume  $(a, i) \in UNIV \times \text{Basis}$ 
  then have  $i \in \text{Basis}$  by auto
  have  $*: \{x::'a. x \cdot i < a\} = (\bigcup n. \{x. x \cdot i \leq a - 1 / \text{real } (\text{Suc } n)\})$ 
  proof (safe, simp_all del: of_nat_Suc)
    fix  $x::'a$  assume  $*: x \cdot i < a$ 
    with reals_Archimedean[of  $a - x \cdot i$ ]
    obtain  $n$  where  $x \cdot i < a - 1 / \text{real } (\text{Suc } n)$ 
      by (auto simp: field_simps)
    then show  $\exists n. x \cdot i \leq a - 1 / \text{real } (\text{Suc } n)$ 
      by (blast intro: less_imp_le)
  next
    fix  $x::'a$  and  $n$ 
    assume  $x \cdot i \leq a - 1 / \text{real } (\text{Suc } n)$ 
    also have  $\dots < a$  by auto
    finally show  $x \cdot i < a$  .
  qed

```



```

show  $\{x. x \cdot i < a\} \in ?SIGMA \text{ unfolding } *$ 
by (intro sets.countable_UN) (auto intro: i)
qed auto

```

```

lemma borel_eq_halfspace_ge:
  borel = sigma UNIV (( $\lambda (a, i). \{x::'a::euclidean\_space. a \leq x \cdot i\}$ ) ' (UNIV  $\times$ 
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix a :: real and i :: 'a assume i: (a, i)  $\in$  UNIV  $\times$  Basis
  have *:  $\{x::'a. x \cdot i < a\} = \text{space } ?SIGMA - \{x::'a. a \leq x \cdot i\}$  by auto
  show  $\{x. x \cdot i < a\} \in ?SIGMA \text{ unfolding } *$ 
  using i by (intro sets.compl_sets) auto
qed auto

```

```

lemma borel_eq_halfspace_greater:
  borel = sigma UNIV (( $\lambda (a, i). \{x::'a::euclidean\_space. a < x \cdot i\}$ ) ' (UNIV  $\times$ 
  Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i)  $\in$  (UNIV  $\times$  Basis)
  then have i: i  $\in$  Basis by auto
  have *:  $\{x::'a. x \cdot i \leq a\} = \text{space } ?SIGMA - \{x::'a. a < x \cdot i\}$  by auto
  show  $\{x. x \cdot i \leq a\} \in ?SIGMA \text{ unfolding } *$ 
  by (intro sets.compl_sets) (auto intro: i)
qed auto

```

```

lemma borel_eq_atMost:
  borel = sigma UNIV (range ( $\lambda a. \{..a::'a::ordered\_euclidean\_space\}$ ))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i)  $\in$  UNIV  $\times$  Basis
  then have i  $\in$  Basis by auto
  then have *:  $\{x::'a. x \cdot i \leq a\} = (\bigcup k::nat. \{.. (\sum n \in \text{Basis}. (\text{if } n = i \text{ then } a \text{ else }
  \text{real } k) *_{\mathbb{R}} n)\})$ 
  proof (safe, simp_all add: eucl_le[where 'a='a] split: if_split_asm)
  fix x :: 'a
  obtain k where Max (( $\cdot$ ) x ' Basis)  $\leq$  real k
  using real_arch_simple by blast
  then have  $\bigwedge i. i \in \text{Basis} \implies x \cdot i \leq \text{real } k$ 
  by (subst (asm) Max_le_iff) auto
  then show  $\exists k::nat. \forall ia \in \text{Basis}. ia \neq i \implies x \cdot ia \leq \text{real } k$ 
  by (auto intro!: exI[of _ k])
  qed
  show  $\{x. x \cdot i \leq a\} \in ?SIGMA \text{ unfolding } *$ 
  by (intro sets.countable_UN) auto
qed auto

```

```

lemma borel_eq_greaterThan:

```

```

    borel = sigma UNIV (range (λa::'a::ordered_euclidean_space. {x. a < e x}))
    (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have **: ∃ y. ∀ j ∈ Basis. j ≠ i → - real y < x · j if a < x · i for x
  proof -
    obtain k where k: Max ((·) (- x) 'Basis) < real k
    using reals_Archimedean2 by blast
    { fix i :: 'a assume i ∈ Basis
      then have -x·i < real k
        using k by (subst (asm) Max_less_iff) auto
      then have - real k < x·i by simp }
    then show ?thesis
      by (auto intro!: exI[of _ k])
  qed
  have {x::'a. x·i ≤ a} = UNIV - {x::'a. a < x·i} by auto
  also have *: {x::'a. a < x·i} = (∪ k::nat. {x. (∑ n ∈ Basis. (if n = i then a else
  -k) *R n) < e x})
    using i ** by (force simp add: eucl_less_def split: if_split_asm)
  finally have eq: {x. x · i ≤ a} = UNIV - (∪ x. {x a. (∑ n ∈ Basis. (if n = i
  then a else - real x) *R n) < e x a}) .
  show {x. x·i ≤ a} ∈ ?SIGMA
    unfolding eq by (fastforce intro!: sigma_sets_top sets.Diff)
qed auto

lemma borel_eq_lessThan:
  borel = sigma UNIV (range (λa::'a::ordered_euclidean_space. {x. x < e a}))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_ge])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have **: ∃ y. ∀ j ∈ Basis. j ≠ i → real y > x · j if a > x · i for x
  proof -
    obtain k where k: Max ((·) x 'Basis) < real k
    using reals_Archimedean2 by blast
    { fix i :: 'a assume i ∈ Basis
      then have x·i < real k
        using k by (subst (asm) Max_less_iff) auto
      then have x·i < real k by simp }
    then show ?thesis
      by (auto intro!: exI[of _ k])
  qed
  have {x::'a. a ≤ x·i} = UNIV - {x::'a. x·i < a} by auto
  also have *: {x::'a. x·i < a} = (∪ k::nat. {x. x < e (∑ n ∈ Basis. (if n = i then
  a else real k) *R n)}) using ⟨i ∈ Basis⟩
    using i ** by (force simp add: eucl_less_def split: if_split_asm)
  finally
  have eq: {x. a ≤ x · i} =

```

$UNIV - (\bigcup k. \{x. x < e (\sum n \in Basis. (if\ n = i\ then\ a\ else\ real\ k) *R\ n)\}) .$

show $\{x. a \leq x \cdot i\} \in ?SIGMA$
unfolding *eq* **by** (*fastforce intro!*: *sigma_sets_top sets.Diff*)
qed *auto*

lemma *borel_eq_atLeastAtMost*:

borel = sigma UNIV (range ($\lambda(a,b). \{a..b\} :: 'a::ordered_euclidean_space\ set$))
(is _ = ?SIGMA)

proof (*rule borel_eq_sigmaI5[OF borel_eq_atMost]*)

fix *a* :: 'a

have $*$: $\{..a\} = (\bigcup n::nat. \{-\ real\ n *R\ One .. a\})$

proof (*safe, simp_all add: eucl_le[where 'a='a]*)

fix *x* :: 'a

obtain *k* **where** $k: Max\ ((\cdot)\ (-\ x)\ 'Basis) \leq real\ k$

using *real_arch_simple* **by** *blast*

{ fix *i* :: 'a **assume** *i* $\in Basis$

with *k* **have** $-x \cdot i \leq real\ k$

by (*subst (asm) Max_le_iff*) (*auto simp: field_simps*)

then **have** $-real\ k \leq x \cdot i$ **by** *simp* }

then **show** $\exists n::nat. \forall i \in Basis. -real\ n \leq x \cdot i$

by (*auto intro!: exI[of _ k]*)

qed

show $\{..a\} \in ?SIGMA$ **unfolding** $*$

by (*intro sets.countable_UN*)

(*auto intro!: sigma_sets_top*)

qed *auto*

lemma *borel_set_induct[consumes 1, case_names empty interval compl union]*:

assumes *A* $\in sets\ borel$

assumes *empty*: $P\ \{\}$ **and** *int*: $\bigwedge a\ b. a \leq b \implies P\ \{a..b\}$ **and** *compl*: $\bigwedge A. A \in sets\ borel \implies P\ A \implies P\ (-A)$ **and**

un: $\bigwedge f. disjoint_family\ f \implies (\bigwedge i. f\ i \in sets\ borel) \implies (\bigwedge i. P\ (f\ i)) \implies P\ (\bigcup i::nat. f\ i)$

shows $P\ (A::real\ set)$

proof $-$

let $?G = range\ (\lambda(a,b). \{a..b::real\})$

have *Int_stable* $?G\ ?G \subseteq Pow\ UNIV\ A \in sigma_sets\ UNIV\ ?G$

using *assms(1)* **by** (*auto simp add: borel_eq_atLeastAtMost Int_stable_def*)

thus *?thesis*

proof (*induction rule: sigma_sets_induct_disjoint*)

case (*union f*)

from *union.hyps(2)* **have** $\bigwedge i. f\ i \in sets\ borel$ **by** (*auto simp: borel_eq_atLeastAtMost*)

with *union* **show** *?case* **by** (*auto intro: un*)

next

case (*basic A*)

then **obtain** *a b* **where** $A = \{a .. b\}$ **by** *auto*

then **show** *?case*

2200

by (cases $a \leq b$) (auto intro: int empty)
qed (auto intro: empty compl simp: Compl_eq_Diff_UNIV[symmetric] borel_eq_atLeastAtMost)
qed

lemma borel_sigma_sets_Ioc: borel = sigma UNIV (range ($\lambda(a, b). \{a <.. b::real\}$))

proof (rule borel_eq_sigmaI5[OF borel_eq_atMost])

fix $i :: real$

have $\{..i\} = (\bigcup j::nat. \{-j <.. i\})$

by (auto simp: minus_less_iff reals_Archimedean2)

also have $\dots \in sets (sigma UNIV (range (\lambda(i, j). \{i <..j\})))$

by (intro sets.countable_nat_UN) auto

finally show $\{..i\} \in sets (sigma UNIV (range (\lambda(i, j). \{i <..j\})))$.

qed simp

lemma eucl_lessThan: $\{x::real. x < e a\} = lessThan a$

by (simp add: eucl_less_def lessThan_def)

lemma borel_eq_atLeastLessThan:

borel = sigma UNIV (range ($\lambda(a, b). \{a ..< b :: real\}$)) (is _ = ?SIGMA)

proof (rule borel_eq_sigmaI5[OF borel_eq_lessThan])

have move_uminus: $\bigwedge x y::real. -x \leq y \longleftrightarrow -y \leq x$ by auto

fix $x :: real$

have $\{..<x\} = (\bigcup i::nat. \{-real i ..< x\})$

by (auto simp: move_uminus real_arch_simple)

then show $\{y. y < e x\} \in ?SIGMA$

by (auto intro: sigma_sets.intros(2-)) simp: eucl_lessThan

qed auto

lemma borel_measurable_halfspacesI:

fixes $f :: 'a \Rightarrow 'c::euclidean_space$

assumes F : borel = sigma UNIV ($F \text{ ' } (UNIV \times Basis)$)

and S_eq : $\bigwedge a i. S a i = f \text{ - ' } F (a, i) \cap space M$

shows $f \in borel_measurable M = (\forall i \in Basis. \forall a::real. S a i \in sets M)$

proof safe

fix $a :: real$ and $i :: 'b$ assume $i: i \in Basis$ and $f: f \in borel_measurable M$

then show $S a i \in sets M$ unfolding assms

by (auto intro!: measurable_sets simp: assms(1))

next

assume $a: \forall i \in Basis. \forall a. S a i \in sets M$

then show $f \in borel_measurable M$

by (auto intro!: measurable_measure_of simp: S_eq F)

qed

lemma borel_measurable_iff_halfspace_le:

fixes $f :: 'a \Rightarrow 'c::euclidean_space$

shows $f \in borel_measurable M = (\forall i \in Basis. \forall a. \{w \in space M. f w \cdot i \leq a\} \in sets M)$

by (rule borel_measurable_halfspacesI[OF borel_eq_halfspace_le]) auto

lemma *borel_measurable_iff_halfspace_less*:
fixes $f :: 'a \Rightarrow 'c::\text{euclidean_space}$
shows $f \in \text{borel_measurable } M \longleftrightarrow (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. f w \cdot i < a\} \in \text{sets } M)$
by (rule *borel_measurable_halfspacesI*[OF *borel_eq_halfspace_less*]) *auto*

lemma *borel_measurable_iff_halfspace_ge*:
fixes $f :: 'a \Rightarrow 'c::\text{euclidean_space}$
shows $f \in \text{borel_measurable } M = (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. a \leq f w \cdot i\} \in \text{sets } M)$
by (rule *borel_measurable_halfspacesI*[OF *borel_eq_halfspace_ge*]) *auto*

lemma *borel_measurable_iff_halfspace_greater*:
fixes $f :: 'a \Rightarrow 'c::\text{euclidean_space}$
shows $f \in \text{borel_measurable } M \longleftrightarrow (\forall i \in \text{Basis}. \forall a. \{w \in \text{space } M. a < f w \cdot i\} \in \text{sets } M)$
by (rule *borel_measurable_halfspacesI*[OF *borel_eq_halfspace_greater*]) *auto*

lemma *borel_measurable_iff_le*:
 $(f :: 'a \Rightarrow \text{real}) \in \text{borel_measurable } M = (\forall a. \{w \in \text{space } M. f w \leq a\} \in \text{sets } M)$
using *borel_measurable_iff_halfspace_le*[**where** $'c = \text{real}$] **by** *simp*

lemma *borel_measurable_iff_less*:
 $(f :: 'a \Rightarrow \text{real}) \in \text{borel_measurable } M = (\forall a. \{w \in \text{space } M. f w < a\} \in \text{sets } M)$
using *borel_measurable_iff_halfspace_less*[**where** $'c = \text{real}$] **by** *simp*

lemma *borel_measurable_iff_ge*:
 $(f :: 'a \Rightarrow \text{real}) \in \text{borel_measurable } M = (\forall a. \{w \in \text{space } M. a \leq f w\} \in \text{sets } M)$
using *borel_measurable_iff_halfspace_ge*[**where** $'c = \text{real}$] **by** *simp*

lemma *borel_measurable_iff_greater*:
 $(f :: 'a \Rightarrow \text{real}) \in \text{borel_measurable } M = (\forall a. \{w \in \text{space } M. a < f w\} \in \text{sets } M)$
using *borel_measurable_iff_halfspace_greater*[**where** $'c = \text{real}$] **by** *simp*

lemma *borel_measurable_euclidean_space*:
fixes $f :: 'a \Rightarrow 'c::\text{euclidean_space}$
shows $f \in \text{borel_measurable } M \longleftrightarrow (\forall i \in \text{Basis}. (\lambda x. f x \cdot i) \in \text{borel_measurable } M)$
proof *safe*
assume $f: \forall i \in \text{Basis}. (\lambda x. f x \cdot i) \in \text{borel_measurable } M$
then show $f \in \text{borel_measurable } M$
by (*subst borel_measurable_iff_halfspace_le*) *auto*
qed *auto*

7.4.5 Borel measurable operators

lemma *borel_measurable_norm*[*measurable*]: $\text{norm} \in \text{borel_measurable borel}$
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sgn* [*measurable*]: (*sgn*::'*a*::*real_normed_vector* \Rightarrow '*a*) \in *borel_measurable borel*
by (*rule borel_measurable_continuous_countable_exceptions*[**where** *X*={0}])
(auto intro!: *continuous_on_sgn continuous_on_id*)

lemma *borel_measurable_uminus*[*measurable (raw)*]:
fixes *g* :: '*a* \Rightarrow '*b*::{*second_countable_topology, real_normed_vector*}
assumes *g*: *g* \in *borel_measurable M*
shows ($\lambda x. - g x$) \in *borel_measurable M*
by (*rule borel_measurable_continuous_on*[*OF _ g*]) (*intro continuous_intros*)

lemma *borel_measurable_diff*[*measurable (raw)*]:
fixes *f* :: '*a* \Rightarrow '*b*::{*second_countable_topology, real_normed_vector*}
assumes *f*: *f* \in *borel_measurable M*
assumes *g*: *g* \in *borel_measurable M*
shows ($\lambda x. f x - g x$) \in *borel_measurable M*
using *borel_measurable_add* [*of f M - g*] *assms* **by** (*simp add: fun_Compl_def*)

lemma *borel_measurable_times*[*measurable (raw)*]:
fixes *f* :: '*a* \Rightarrow '*b*::{*second_countable_topology, real_normed_algebra*}
assumes *f*: *f* \in *borel_measurable M*
assumes *g*: *g* \in *borel_measurable M*
shows ($\lambda x. f x * g x$) \in *borel_measurable M*
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_prod*[*measurable (raw)*]:
fixes *f* :: '*c* \Rightarrow '*a* \Rightarrow '*b*::{*second_countable_topology, real_normed_field*}
assumes $\bigwedge i. i \in S \Rightarrow f i \in$ *borel_measurable M*
shows ($\lambda x. \prod_{i \in S}. f i x$) \in *borel_measurable M*
proof cases
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed simp

lemma *borel_measurable_dist*[*measurable (raw)*]:
fixes *g f* :: '*a* \Rightarrow '*b*::{*second_countable_topology, metric_space*}
assumes *f*: *f* \in *borel_measurable M*
assumes *g*: *g* \in *borel_measurable M*
shows ($\lambda x. dist (f x) (g x)$) \in *borel_measurable M*
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_scaleR*[*measurable (raw)*]:
fixes *g* :: '*a* \Rightarrow '*b*::{*second_countable_topology, real_normed_vector*}
assumes *f*: *f* \in *borel_measurable M*
assumes *g*: *g* \in *borel_measurable M*
shows ($\lambda x. f x *_{\mathbb{R}} g x$) \in *borel_measurable M*
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_uminus_eq* [*simp*]:
fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, real_normed_vector}\}$
shows $(\lambda x. - f x) \in \text{borel_measurable } M \longleftrightarrow f \in \text{borel_measurable } M$ (**is** $?l = ?r$)
by (*smt* (*verit*, *ccfv_SIG*) *borel_measurable_uminus equation_minus_iff measurable_cong*)

lemma *affine_borel_measurable_vector*:
fixes $f :: 'a \Rightarrow 'x :: \text{real_normed_vector}$
assumes $f \in \text{borel_measurable } M$
shows $(\lambda x. a + b *_R f x) \in \text{borel_measurable } M$
proof (*rule* *borel_measurableI*)
fix $S :: 'x \text{ set}$ **assume** *open S*
show $(\lambda x. a + b *_R f x) - ' S \cap \text{space } M \in \text{sets } M$
proof *cases*
assume $b \neq 0$
with $\langle \text{open } S \rangle$ **have** *open* $((\lambda x. (- a + x) /_R b) - ' S)$ (**is** *open* $?S$)
using *open_affinity* [*of S inverse b - a /_R b*]
by (*auto simp: algebra_simps*)
hence $?S \in \text{sets borel}$ **by** *auto*
moreover
have $\bigwedge x. \llbracket a + b *_R f x \in S \rrbracket \implies f x \in (\lambda x. (x - a) /_R b) - ' S$
using $\langle b \neq 0 \rangle$ *image_iff* **by** *fastforce*
with $\langle b \neq 0 \rangle$ **have** $(\lambda x. a + b *_R f x) - ' S = f - ' ?S$
by *auto*
ultimately show *?thesis using assms unfolding in_borel_measurable_borel*
by *auto*
qed *simp*
qed

lemma *borel_measurable_const_scaleR*[*measurable (raw)*]:
 $f \in \text{borel_measurable } M \implies (\lambda x. b *_R f x :: 'a :: \text{real_normed_vector}) \in \text{borel_measurable } M$
using *affine_borel_measurable_vector*[*of f M 0 b*] **by** *simp*

lemma *borel_measurable_const_add*[*measurable (raw)*]:
 $f \in \text{borel_measurable } M \implies (\lambda x. a + f x :: 'a :: \text{real_normed_vector}) \in \text{borel_measurable } M$
using *affine_borel_measurable_vector*[*of f M a 1*] **by** *simp*

lemma *borel_measurable_inverse*[*measurable (raw)*]:
fixes $f :: 'a \Rightarrow 'b :: \text{real_normed_div_algebra}$
assumes $f \in \text{borel_measurable } M$
shows $(\lambda x. \text{inverse } (f x)) \in \text{borel_measurable } M$
proof $-$
have *countable* $\{0 :: 'b\}$ *continuous_on* $(- \{0 :: 'b\})$ *inverse*
by (*auto intro!*: *continuous_on_inverse continuous_on_id*)
then show *?thesis*
by (*metis* *borel_measurable_continuous_countable_exceptions f measurable_compose*)

qed

lemma *borel_measurable_divide*[*measurable (raw)*]:
 $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies$
 $(\lambda x. f x / g x :: 'b :: \{\text{second_countable_topology, real_normed_div_algebra}\}) \in$
 $\text{borel_measurable } M$
by (*simp add: divide_inverse*)

lemma *borel_measurable_abs*[*measurable (raw)*]:
 $f \in \text{borel_measurable } M \implies (\lambda x. |f x :: \text{real}|) \in \text{borel_measurable } M$
unfolding *abs_real_def* **by** *simp*

lemma *borel_measurable_nth*[*measurable (raw)*]:
 $(\lambda x :: \text{real}^n. x \$ i) \in \text{borel_measurable borel}$
by (*simp add: cart_eq_inner_axis*)

lemma *convex_measurable*:
fixes $A :: 'a :: \text{euclidean_space set}$
shows $X \in \text{borel_measurable } M \implies X \text{ 'space } M \subseteq A \implies \text{open } A \implies \text{convex_on}$
 $A q \implies$
 $(\lambda x. q (X x)) \in \text{borel_measurable } M$
by (*rule measurable_compose*[**where** $f=X$ **and** $N=\text{restrict_space borel } A$])
(auto intro!: borel_measurable_continuous_on_restrict convex_on_continuous
measurable_restrict_space2)

lemma *borel_measurable_ln*[*measurable (raw)*]:
assumes $f: f \in \text{borel_measurable } M$
shows $(\lambda x. \ln (f x :: \text{real})) \in \text{borel_measurable } M$
using *borel_measurable_continuous_countable_exceptions*[*of {0}*] *measurable_compose*[*OF*
 f]
by (*auto intro!: continuous_on_ln continuous_on_id*)

lemma *borel_measurable_log*[*measurable (raw)*]:
 $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies (\lambda x. \log (g x) (f x))$
 $\in \text{borel_measurable } M$
unfolding *log_def* **by** *auto*

lemma *borel_measurable_exp*[*measurable*]:
 $(\text{exp} :: 'a :: \{\text{real_normed_field, banach}\} \Rightarrow 'a) \in \text{borel_measurable borel}$
by (*intro borel_measurable_continuous_onI continuous_at_imp_continuous_on*
ballI isCont_exp)

lemma *measurable_real_floor*[*measurable*]:
 $(\text{floor} :: \text{real} \Rightarrow \text{int}) \in \text{measurable borel (count_space UNIV)}$

proof –

have $\bigwedge a x. \lfloor x \rfloor = a \iff (\text{real_of_int } a \leq x \wedge x < \text{real_of_int } (a + 1))$

by (*auto intro: floor_eq2*)

then show *?thesis*

by (*auto simp: vimage_def measurable_count_space_eq2_countable*)

qed

lemma *measurable_real_ceiling* [measurable]:
 (*ceiling* :: *real* \Rightarrow *int*) \in *measurable borel* (*count_space UNIV*)
unfolding *ceiling_def* [*abs_def*] **by** *simp*

lemma *borel_measurable_real_floor*: ($\lambda x::\text{real}. \text{real_of_int } \lfloor x \rfloor$) \in *borel_measurable borel*
by *simp*

lemma *borel_measurable_root* [measurable]: *root* *n* \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sqrt* [measurable]: *sqrt* \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_power* [measurable (*raw*)]:
fixes *f* :: $_ \Rightarrow 'b::\{\text{power,real_normed_algebra}\}$
assumes *f*: *f* \in *borel_measurable M*
shows ($\lambda x. (f\ x) ^ n$) \in *borel_measurable M*
by (*intro borel_measurable_continuous_on* [*OF* $_$] *continuous_intros*)

lemma *borel_measurable_Re* [measurable]: *Re* \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_Im* [measurable]: *Im* \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_of_real* [measurable]: (*of_real* :: $_ \Rightarrow (_::\text{real_normed_algebra})$)
 \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sin* [measurable]: (*sin* :: $_ \Rightarrow (_::\{\text{real_normed_field,banach}\})$)
 \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_cos* [measurable]: (*cos* :: $_ \Rightarrow (_::\{\text{real_normed_field,banach}\})$)
 \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_arctan* [measurable]: *arctan* \in *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_complex_iff*:
f \in *borel_measurable M* \iff
 ($\lambda x. \text{Re } (f\ x)$) \in *borel_measurable M* \wedge ($\lambda x. \text{Im } (f\ x)$) \in *borel_measurable M*
 (*is ?lhs* \iff *?rhs*)

proof

show *?lhs* \implies *?rhs*

```

using borel_measurable_Im borel_measurable_Re measurable_compose by
blast
assume R: ?rhs
then have ( $\lambda x.$  complex_of_real (Re (f x)) + i * complex_of_real (Im (f x)))
 $\in$  borel_measurable M
by (intro borel_measurable_add) auto
then show ?lhs
using complex_eq by force
qed

```

```

lemma powr_real_measurable [measurable]:
assumes f  $\in$  measurable M borel g  $\in$  measurable M borel
shows ( $\lambda x.$  f x powr g x :: real)  $\in$  measurable M borel
using assms by (simp_all add: powr_def)

```

```

lemma measurable_of_bool [measurable]: of_bool  $\in$  count_space UNIV  $\rightarrow_M$  borel
by simp

```

7.4.6 Borel space on the extended reals

```

lemma borel_measurable_ereal [measurable (raw)]:
assumes f: f  $\in$  borel_measurable M shows ( $\lambda x.$  ereal (f x))  $\in$  borel_measurable
M
using continuous_on_ereal f by (rule borel_measurable_continuous_on) (rule
continuous_on_id)

```

```

lemma borel_measurable_real_of_ereal [measurable (raw)]:
fixes f :: 'a  $\Rightarrow$  ereal
assumes f: f  $\in$  borel_measurable M
shows ( $\lambda x.$  real_of_ereal (f x))  $\in$  borel_measurable M
using measurable_compose[OF f] borel_measurable_continuous_countable_exceptions[of
 $\{\infty, -\infty\}$ ]
by (auto intro: continuous_on_real simp: Compl_eq_Diff_UNIV)

```

```

lemma borel_measurable_ereal_cases:
fixes f :: 'a  $\Rightarrow$  ereal
assumes f: f  $\in$  borel_measurable M
assumes H: ( $\lambda x.$  H (ereal (real_of_ereal (f x))))  $\in$  borel_measurable M
shows ( $\lambda x.$  H (f x))  $\in$  borel_measurable M

```

```

proof -
let ?F =  $\lambda x.$  if f x =  $\infty$  then H  $\infty$  else if f x =  $-\infty$  then H ( $-\infty$ ) else H (ereal
(real_of_ereal (f x)))
{ fix x have H (f x) = ?F x by (cases f x) auto }
with f H show ?thesis by simp
qed

```

```

lemma
fixes f :: 'a  $\Rightarrow$  ereal assumes f [measurable]: f  $\in$  borel_measurable M
shows borel_measurable_ereal_abs [measurable (raw)]: ( $\lambda x.$  |f x|)  $\in$  borel_measurable

```

M

and *borel_measurable_ereal_inverse*[*measurable(raw)*]: $(\lambda x. \text{inverse } (f \ x) :: \text{ereal}) \in \text{borel_measurable } M$
and *borel_measurable_uminus_ereal*[*measurable(raw)*]: $(\lambda x. - \ f \ x :: \text{ereal}) \in \text{borel_measurable } M$
by (*auto simp del: abs_real_of_ereal simp: borel_measurable_ereal_cases[OF f] measurable_Iif*)

lemma *borel_measurable_uminus_eq_ereal*[*simp*]:

$(\lambda x. - \ f \ x :: \text{ereal}) \in \text{borel_measurable } M \longleftrightarrow f \in \text{borel_measurable } M$
by (*smt (verit, ccfv_SIG) borel_measurable_uminus_ereal_ereal_uminus_uminus measurable_cong*)

lemma *set_Collect_ereal2*:

fixes *f g* :: 'a \Rightarrow *ereal*
assumes *f*: $f \in \text{borel_measurable } M$
assumes *g*: $g \in \text{borel_measurable } M$
assumes *H*: $\{x \in \text{space } M. H \ (\text{ereal } (\text{real_of_ereal } (f \ x))) \ (\text{ereal } (\text{real_of_ereal } (g \ x)))\} \in \text{sets } M$
 $\{x \in \text{space } \text{borel}. H \ (-\infty) \ (\text{ereal } x)\} \in \text{sets } \text{borel}$
 $\{x \in \text{space } \text{borel}. H \ (\infty) \ (\text{ereal } x)\} \in \text{sets } \text{borel}$
 $\{x \in \text{space } \text{borel}. H \ (\text{ereal } x) \ (-\infty)\} \in \text{sets } \text{borel}$
 $\{x \in \text{space } \text{borel}. H \ (\text{ereal } x) \ (\infty)\} \in \text{sets } \text{borel}$
shows $\{x \in \text{space } M. H \ (f \ x) \ (g \ x)\} \in \text{sets } M$
proof –
let *?G* = $\lambda y \ x. \text{if } g \ x = \infty \text{ then } H \ y \ \infty \text{ else if } g \ x = -\infty \text{ then } H \ y \ (-\infty) \text{ else } H \ y \ (\text{ereal } (\text{real_of_ereal } (g \ x)))$
let *?F* = $\lambda x. \text{if } f \ x = \infty \text{ then } ?G \ \infty \ x \text{ else if } f \ x = -\infty \text{ then } ?G \ (-\infty) \ x \text{ else } ?G \ (\text{ereal } (\text{real_of_ereal } (f \ x))) \ x$
{ fix } x **have $H \ (f \ x) \ (g \ x) = ?F \ x$ **by** (*cases f x g x rule: ereal2_cases*) **auto** }**
note * = *this*
from *assms* **show** *?thesis*
by (*subst **) (*simp del: space_borel split del: if_split*)
qed

lemma *borel_measurable_ereal_iff*:

shows $(\lambda x. \text{ereal } (f \ x)) \in \text{borel_measurable } M \longleftrightarrow f \in \text{borel_measurable } M$
proof
assume $(\lambda x. \text{ereal } (f \ x)) \in \text{borel_measurable } M$
from *borel_measurable_real_of_ereal*[*OF this*]
show $f \in \text{borel_measurable } M$ **by** *auto*
qed *auto*

lemma *borel_measurable_erealD*[*measurable_dest*]:

$(\lambda x. \text{ereal } (f \ x)) \in \text{borel_measurable } M \Longrightarrow g \in \text{measurable } N \ M \Longrightarrow (\lambda x. f \ (g \ x)) \in \text{borel_measurable } N$
unfolding *borel_measurable_ereal_iff* **by** *simp*

theorem *borel_measurable_ereal_iff_real*:

fixes $f :: 'a \Rightarrow \text{ereal}$
shows $f \in \text{borel_measurable } M \longleftrightarrow$
 $((\lambda x. \text{real_of_ereal } (f x)) \in \text{borel_measurable } M \wedge f -' \{\infty\} \cap \text{space } M \in \text{sets } M \wedge f -' \{-\infty\} \cap \text{space } M \in \text{sets } M)$
proof safe
assume $*$: $(\lambda x. \text{real_of_ereal } (f x)) \in \text{borel_measurable } M \wedge f -' \{\infty\} \cap \text{space } M \in \text{sets } M \wedge f -' \{-\infty\} \cap \text{space } M \in \text{sets } M$
have $f -' \{\infty\} \cap \text{space } M = \{x \in \text{space } M. f x = \infty\} \wedge f -' \{-\infty\} \cap \text{space } M = \{x \in \text{space } M. f x = -\infty\}$ **by auto**
with $*$ **have** $**$: $\{x \in \text{space } M. f x = \infty\} \in \text{sets } M \wedge \{x \in \text{space } M. f x = -\infty\} \in \text{sets } M$ **by simp_all**
let $?f = \lambda x. \text{if } f x = \infty \text{ then } \infty \text{ else if } f x = -\infty \text{ then } -\infty \text{ else } \text{ereal } (\text{real_of_ereal } (f x))$
have $?f \in \text{borel_measurable } M$ **using** $*$ $**$ **by** $(\text{intro measurable_If}) \text{ auto}$
also have $?f = f$ **by** $(\text{auto simp: fun_eq_iff } \text{ereal_real})$
finally show $f \in \text{borel_measurable } M$.
qed simp_all

lemma borel_measurable_ereal_iff_Iio:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel_measurable } M \longleftrightarrow (\forall a. f -' \{.. < a\} \cap \text{space } M \in \text{sets } M)$
by $(\text{auto simp: borel_Iio measurable_iff_measure_of})$

lemma borel_measurable_ereal_iff_Ioi:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel_measurable } M \longleftrightarrow (\forall a. f -' \{a < ..\} \cap \text{space } M \in \text{sets } M)$
by $(\text{auto simp: borel_Ioi measurable_iff_measure_of})$

lemma vimage_sets_compl_iff:
 $f -' A \cap \text{space } M \in \text{sets } M \longleftrightarrow f -' (- A) \cap \text{space } M \in \text{sets } M$
by $(\text{metis Diff_Compl Diff_Diff_Int diff_eq inf_aci(1) sets.Diff sets.top vimage_Compl})$

lemma borel_measurable_iff_Iic_ereal:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel_measurable } M \longleftrightarrow (\forall a. f -' \{.. a\} \cap \text{space } M \in \text{sets } M)$
unfolding $\text{borel_measurable_ereal_iff_Ioi vimage_sets_compl_iff}$ **[where** $A = \{a < ..\}$ **for** $a]$ **by simp**

lemma borel_measurable_iff_Ici_ereal:
 $(f :: 'a \Rightarrow \text{ereal}) \in \text{borel_measurable } M \longleftrightarrow (\forall a. f -' \{a.. \} \cap \text{space } M \in \text{sets } M)$
unfolding $\text{borel_measurable_ereal_iff_Ioi vimage_sets_compl_iff}$ **[where** $A = \{.. < a\}$ **for** $a]$ **by simp**

lemma borel_measurable_ereal2:
fixes $f g :: 'a \Rightarrow \text{ereal}$
assumes f : $f \in \text{borel_measurable } M$
assumes g : $g \in \text{borel_measurable } M$
assumes H : $(\lambda x. H (\text{ereal } (\text{real_of_ereal } (f x))) (\text{ereal } (\text{real_of_ereal } (g x)))) \in \text{borel_measurable } M$

```

    ( $\lambda x. H (-\infty) (ereal (real\_of\_ereal (g x)))$ )  $\in$  borel_measurable M
    ( $\lambda x. H (\infty) (ereal (real\_of\_ereal (g x)))$ )  $\in$  borel_measurable M
    ( $\lambda x. H (ereal (real\_of\_ereal (f x))) (-\infty)$ )  $\in$  borel_measurable M
    ( $\lambda x. H (ereal (real\_of\_ereal (f x))) (\infty)$ )  $\in$  borel_measurable M
  shows ( $\lambda x. H (f x) (g x)$ )  $\in$  borel_measurable M
proof -
  let ?G =  $\lambda y x. \text{if } g x = \infty \text{ then } H y \infty \text{ else if } g x = -\infty \text{ then } H y (-\infty) \text{ else}$ 
 $H y (ereal (real\_of\_ereal (g x)))$ 
  let ?F =  $\lambda x. \text{if } f x = \infty \text{ then } ?G \infty x \text{ else if } f x = -\infty \text{ then } ?G (-\infty) x \text{ else}$ 
 $?G (ereal (real\_of\_ereal (f x))) x$ 
  { fix x have  $H (f x) (g x) = ?F x$  by (cases f x g x rule: ereal2_cases) auto }
  note * = this
  from assms show ?thesis unfolding * by simp
qed

```

```

lemma [measurable(raw)]:
  fixes f :: 'a  $\Rightarrow$  ereal
  assumes [measurable]: f  $\in$  borel_measurable M g  $\in$  borel_measurable M
  shows borel_measurable_ereal_add: ( $\lambda x. f x + g x$ )  $\in$  borel_measurable M
  and borel_measurable_ereal_times: ( $\lambda x. f x * g x$ )  $\in$  borel_measurable M
  by (simp_all add: borel_measurable_ereal2)

```

```

lemma [measurable(raw)]:
  fixes f g :: 'a  $\Rightarrow$  ereal
  assumes f  $\in$  borel_measurable M
  assumes g  $\in$  borel_measurable M
  shows borel_measurable_ereal_diff: ( $\lambda x. f x - g x$ )  $\in$  borel_measurable M
  and borel_measurable_ereal_divide: ( $\lambda x. f x / g x$ )  $\in$  borel_measurable M
  using assms by (simp_all add: minus_ereal_def divide_ereal_def)

```

```

lemma borel_measurable_ereal_sum[measurable (raw)]:
  fixes f :: 'c  $\Rightarrow$  'a  $\Rightarrow$  ereal
  assumes  $\bigwedge i. i \in S \Rightarrow f i \in$  borel_measurable M
  shows ( $\lambda x. \sum_{i \in S}. f i x$ )  $\in$  borel_measurable M
  using assms by (induction S rule: infinite_finite_induct) auto

```

```

lemma borel_measurable_ereal_prod[measurable (raw)]:
  fixes f :: 'c  $\Rightarrow$  'a  $\Rightarrow$  ereal
  assumes  $\bigwedge i. i \in S \Rightarrow f i \in$  borel_measurable M
  shows ( $\lambda x. \prod_{i \in S}. f i x$ )  $\in$  borel_measurable M
  using assms by (induction S rule: infinite_finite_induct) auto

```

```

lemma borel_measurable_extreal_suminf[measurable (raw)]:
  fixes f :: nat  $\Rightarrow$  'a  $\Rightarrow$  ereal
  assumes [measurable]:  $\bigwedge i. f i \in$  borel_measurable M
  shows ( $\lambda x. (\sum i. f i x)$ )  $\in$  borel_measurable M
  unfolding suminf_def sums_def[abs_def] lim_def[symmetric] by simp

```

7.4.7 Borel space on the extended non-negative reals

ennreal is a topological monoid, so no rules for plus are required, also all order statements are usually done on type classes.

lemma *measurable_enn2ereal*[*measurable*]: $enn2ereal \in \text{borel} \rightarrow_M \text{borel}$
by (*intro borel_measurable_continuous_onI continuous_on_enn2ereal*)

lemma *measurable_e2ennreal*[*measurable*]: $e2ennreal \in \text{borel} \rightarrow_M \text{borel}$
by (*intro borel_measurable_continuous_onI continuous_on_e2ennreal*)

lemma *borel_measurable_enn2real*[*measurable (raw)*]:
 $f \in M \rightarrow_M \text{borel} \implies (\lambda x. \text{enn2real } (f x)) \in M \rightarrow_M \text{borel}$
unfolding *enn2real_def*[*abs_def*] **by** *measurable*

definition [*simp*]: $is_borel f M \iff f \in \text{borel_measurable } M$

lemma *is_borel_transfer*[*transfer_rule*]: $rel_fun (rel_fun (=) pcr_ennreal) (=)$
 $is_borel is_borel$

unfolding *is_borel_def*[*abs_def*]

proof (*safe intro!*: *rel_funI ext dest!*: *rel_fun_eq_pcr_ennreal*[*THEN iffD1*])

fix *f* **and** *M* :: 'a *measure*

show $f \in \text{borel_measurable } M$ **if** $f: \text{enn2ereal} \circ f \in \text{borel_measurable } M$

using *measurable_compose*[*OF f measurable_e2ennreal*] **by** *simp*

qed *simp*

context

includes *ennreal.lifting*

begin

lemma *measurable_ennreal*[*measurable*]: $ennreal \in \text{borel} \rightarrow_M \text{borel}$

unfolding *is_borel_def*[*symmetric*]

by *transfer simp*

lemma *borel_measurable_ennreal_iff*[*simp*]:

assumes [*simp*]: $\bigwedge x. x \in \text{space } M \implies 0 \leq f x$

shows $(\lambda x. \text{ennreal } (f x)) \in M \rightarrow_M \text{borel} \iff f \in M \rightarrow_M \text{borel}$

proof *safe*

assume $(\lambda x. \text{ennreal } (f x)) \in M \rightarrow_M \text{borel}$

then have $(\lambda x. \text{enn2real } (\text{ennreal } (f x))) \in M \rightarrow_M \text{borel}$

by *measurable*

then show $f \in M \rightarrow_M \text{borel}$

by (*rule measurable_cong*[*THEN iffD1, rotated*]) *auto*

qed *measurable*

lemma *borel_measurable_times_ennreal*[*measurable (raw)*]:

fixes *f g* :: 'a \Rightarrow *ennreal*

shows $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x * g x) \in M \rightarrow_M \text{borel}$

unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_inverse_ennreal*[*measurable (raw)*]:

fixes $f :: 'a \Rightarrow \text{ennreal}$

shows $f \in M \rightarrow_M \text{borel} \implies (\lambda x. \text{inverse } (f x)) \in M \rightarrow_M \text{borel}$

unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_divide_ennreal*[*measurable (raw)*]:

fixes $f :: 'a \Rightarrow \text{ennreal}$

shows $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x / g x) \in M \rightarrow_M \text{borel}$

unfolding *divide_ennreal_def* **by** *simp*

lemma *borel_measurable_minus_ennreal*[*measurable (raw)*]:

fixes $f :: 'a \Rightarrow \text{ennreal}$

shows $f \in M \rightarrow_M \text{borel} \implies g \in M \rightarrow_M \text{borel} \implies (\lambda x. f x - g x) \in M \rightarrow_M \text{borel}$

unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_power_ennreal* [*measurable (raw)*]:

fixes $f :: _ \Rightarrow \text{ennreal}$

assumes $f: f \in \text{borel_measurable } M$

shows $(\lambda x. (f x) ^ n) \in \text{borel_measurable } M$

by (*induction n*) (*use f in auto*)

lemma *borel_measurable_prod_ennreal*[*measurable (raw)*]:

fixes $f :: 'c \Rightarrow 'a \Rightarrow \text{ennreal}$

assumes $\bigwedge i. i \in S \implies f i \in \text{borel_measurable } M$

shows $(\lambda x. \prod_{i \in S}. f i x) \in \text{borel_measurable } M$

using *assms* **by** (*induction S rule: infinite_finite_induct*) *auto*

end

hide_const (**open**) *is_borel*

7.4.8 LIMSEQ is borel measurable

lemma *borel_measurable_LIMSEQ_real*:

fixes $u :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $u': \bigwedge x. x \in \text{space } M \implies (\lambda i. u i x) \longrightarrow u' x$

and $u: \bigwedge i. u i \in \text{borel_measurable } M$

shows $u' \in \text{borel_measurable } M$

proof –

have $\bigwedge x. x \in \text{space } M \implies \text{liminf } (\lambda n. \text{ereal } (u n x)) = \text{ereal } (u' x)$

using u' **by** (*simp add: lim_imp_Liminf*)

moreover from u **have** $(\lambda x. \text{liminf } (\lambda n. \text{ereal } (u n x))) \in \text{borel_measurable } M$

by *auto*

ultimately show *?thesis* **by** (*simp cong: measurable_cong add: borel_measurable_ereal_iff*)

qed

```

lemma borel_measurable_LIMSEQ_metric:
  fixes f :: nat => 'a => 'b :: metric_space
  assumes [measurable]:  $\bigwedge i. f\ i \in \text{borel\_measurable } M$ 
  assumes lim:  $\bigwedge x. x \in \text{space } M \implies (\lambda i. f\ i\ x) \longrightarrow g\ x$ 
  shows  $g \in \text{borel\_measurable } M$ 
  unfolding borel_eq_closed
proof (safe intro!: measurable_measure_of)
  fix A :: 'b set assume closed A

  have [measurable]:  $(\lambda x. \text{infdist } (g\ x)\ A) \in \text{borel\_measurable } M$ 
proof (rule borel_measurable_LIMSEQ_real)
  show  $\bigwedge x. x \in \text{space } M \implies (\lambda i. \text{infdist } (f\ i\ x)\ A) \longrightarrow \text{infdist } (g\ x)\ A$ 
    by (intro tendsto_infdist lim)
  show  $\bigwedge i. (\lambda x. \text{infdist } (f\ i\ x)\ A) \in \text{borel\_measurable } M$ 
    by (intro borel_measurable_continuous_on [where f= $\lambda x. \text{infdist } x\ A$ ]
        continuous_at_imp_continuous_on ballI continuous_infdist continuous_ident)
auto
qed

  show  $g - 'A \cap \text{space } M \in \text{sets } M$ 
proof cases
  assume  $A \neq \{\}$ 
  then have  $\bigwedge x. \text{infdist } x\ A = 0 \iff x \in A$ 
    using <closed A> by (simp add: in_closed_iff_infdist_zero)
  then have  $g - 'A \cap \text{space } M = \{x \in \text{space } M. \text{infdist } (g\ x)\ A = 0\}$ 
    by auto
  also have  $\dots \in \text{sets } M$ 
    by measurable
  finally show ?thesis .
qed simp
qed auto

lemma sets_Collect_Cauchy[measurable]:
  fixes f :: nat => 'a => 'b::{metric_space, second_countable_topology}
  assumes f[measurable]:  $\bigwedge i. f\ i \in \text{borel\_measurable } M$ 
  shows  $\{x \in \text{space } M. \text{Cauchy } (\lambda i. f\ i\ x)\} \in \text{sets } M$ 
  unfolding metric_Cauchy_iff2 using f by auto

lemma borel_measurable_lim_metric[measurable (raw)]:
  fixes f :: nat => 'a => 'b::{banach, second_countable_topology}
  assumes f[measurable]:  $\bigwedge i. f\ i \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. \text{lim } (\lambda i. f\ i\ x)) \in \text{borel\_measurable } M$ 
proof -
  define u' where  $u'\ x = \text{lim } (\lambda i. \text{if } \text{Cauchy } (\lambda i. f\ i\ x) \text{ then } f\ i\ x \text{ else } 0)$  for x
  then have *:  $\bigwedge x. \text{lim } (\lambda i. f\ i\ x) = (\text{if } \text{Cauchy } (\lambda i. f\ i\ x) \text{ then } u'\ x \text{ else } (\text{THE } x. \text{False}))$ 
    by (auto simp: lim_def convergent_eq_Cauchy[symmetric])
  have  $u' \in \text{borel\_measurable } M$ 
proof (rule borel_measurable_LIMSEQ_metric)

```



```

fix x
have convergent ( $\lambda i. \text{if Cauchy } (\lambda i. f i x) \text{ then } f i x \text{ else } 0$ )
  by (cases Cauchy ( $\lambda i. f i x$ ))
      (auto simp add: convergent_eq_Cauchy[symmetric] convergent_def)
then show ( $\lambda i. \text{if Cauchy } (\lambda i. f i x) \text{ then } f i x \text{ else } 0$ )  $\longrightarrow$   $u' x$ 
  unfolding  $u'_\text{def}$ 
  by (rule convergent_LIMSEQ_iff[THEN iffD1])
qed measurable
then show ?thesis
  unfolding * by measurable
qed

```

```

lemma borel_measurable_suminf[measurable (raw)]:
  fixes  $f :: \text{nat} \Rightarrow 'a \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
  assumes  $f[\text{measurable}]: \bigwedge i. f i \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. \text{suminf } (\lambda i. f i x)) \in \text{borel\_measurable } M$ 
  unfolding suminf_def sums_def[abs_def] lim_def[symmetric] by simp

```

```

lemma Collect_closed_imp_pred_borel:  $\text{closed } \{x. P x\} \Longrightarrow \text{Measurable.pred borel } P$ 
  by (simp add: pred_def)

```

Proof by Jeremy Avigad and Luke Serafin

```

lemma isCont_borel_pred[measurable]:
  fixes  $f :: 'b::\text{metric\_space} \Rightarrow 'a::\text{metric\_space}$ 
  shows  $\text{Measurable.pred borel } (\text{isCont } f)$ 
proof (subst measurable_cong)
  let  $?I = \lambda j. \text{inverse}(\text{real } (\text{Suc } j))$ 
  show  $\text{isCont } f x = (\forall i. \exists j. \forall y z. \text{dist } x y < ?I j \wedge \text{dist } x z < ?I j \longrightarrow \text{dist } (f y) (f z) \leq ?I i)$  for  $x$ 
    unfolding continuous_at_eps_delta
  proof safe
    fix  $i$  assume  $\forall e > 0. \exists d > 0. \forall y. \text{dist } y x < d \longrightarrow \text{dist } (f y) (f x) < e$ 
    moreover have  $0 < ?I i / 2$ 
      by simp
    ultimately obtain  $d$  where  $d: 0 < d \wedge y. \text{dist } x y < d \Longrightarrow \text{dist } (f y) (f x) < ?I i / 2$ 
      by (metis dist_commute)
    then obtain  $j$  where  $j: ?I j < d$ 
      by (metis reals_Archimedean)

  show  $\exists j. \forall y z. \text{dist } x y < ?I j \wedge \text{dist } x z < ?I j \longrightarrow \text{dist } (f y) (f z) \leq ?I i$ 
  proof (safe intro!: exI[where  $x=j$ ])
    fix  $y z$  assume *:  $\text{dist } x y < ?I j \wedge \text{dist } x z < ?I j$ 
    have  $\text{dist } (f y) (f z) \leq \text{dist } (f y) (f x) + \text{dist } (f z) (f x)$ 
      by (rule dist_triangle2)
    also have  $\dots < ?I i / 2 + ?I i / 2$ 
      by (intro add_strict_mono d_less_trans[OF  $j$ ] *)
    also have  $\dots \leq ?I i$ 

```

```

      by (simp add: field_simps)
    finally show  $\text{dist } (f y) (f z) \leq ?I i$ 
      by simp
  qed
next
fix  $e::\text{real}$  assume  $0 < e$ 
then obtain  $n$  where  $n: ?I n < e$ 
  by (metis reals_Archimedean)
assume  $\forall i. \exists j. \forall y z. \text{dist } x y < ?I j \wedge \text{dist } x z < ?I j \longrightarrow \text{dist } (f y) (f z) \leq$ 
 $?I i$ 
from this[THEN spec, of Suc n]
obtain  $j$  where  $j: \bigwedge y z. \text{dist } x y < ?I j \implies \text{dist } x z < ?I j \implies \text{dist } (f y) (f$ 
 $z) \leq ?I (Suc n)$ 
  by auto

show  $\exists d > 0. \forall y. \text{dist } y x < d \longrightarrow \text{dist } (f y) (f x) < e$ 
proof (safe intro!: exI[of_ ?I j])
  fix  $y$  assume  $\text{dist } y x < ?I j$ 
  then have  $\text{dist } (f y) (f x) \leq ?I (Suc n)$ 
    by (intro j) (auto simp: dist_commute)
  also have  $?I (Suc n) < ?I n$ 
    by simp
  also note  $n$ 
  finally show  $\text{dist } (f y) (f x) < e$  .
qed simp
qed (intro pred_intros_countable closed_Collect_all closed_Collect_le open_Collect_less
  Collect_closed_imp_pred_borel closed_Collect_imp_open_Collect_conj
  continuous_intros)

lemma isCont_borel:
  fixes  $f :: 'b::\text{metric\_space} \Rightarrow 'a::\text{metric\_space}$ 
  shows  $\{x. \text{isCont } f x\} \in \text{sets borel}$ 
  by simp

lemma is_real_interval:
  assumes  $S: \text{is\_interval } S$ 
  shows  $\exists a b::\text{real}. S = \{\} \vee S = \text{UNIV} \vee S = \{..<b\} \vee S = \{..b\} \vee S = \{a<..\}$ 
 $\vee S = \{a.. \vee$ 
 $S = \{a<..<b\} \vee S = \{a<..b\} \vee S = \{a..<b\} \vee S = \{a..b\}$ 
  using  $S$  unfolding is_interval_1 by (blast intro: interval_cases)

lemma real_interval_borel_measurable:
  assumes is_interval (S::real set)
  shows  $S \in \text{sets borel}$ 
proof -
  from assms is_real_interval have  $\exists a b::\text{real}. S = \{\} \vee S = \text{UNIV} \vee S = \{..<b\}$ 
 $\vee S = \{..b\} \vee$ 
 $S = \{a<.. \vee S = \{a.. \vee S = \{a<..<b\} \vee S = \{a<..b\} \vee S = \{a..<b\} \vee S$ 

```

```

= {a..b} by auto
  then show ?thesis
    by auto
qed

```

The next lemmas hold in any second countable linorder (including ennreal or ereal for instance), but in the current state they are restricted to reals.

```

lemma borel_measurable_mono_on_fnc:
  fixes f :: real  $\Rightarrow$  real and A :: real set
  assumes mono_on A f
  shows f  $\in$  borel_measurable (restrict_space borel A)
proof -
  have  $\bigwedge x. x \in A \implies \{x\} \in \text{sets } (\text{restrict\_space } \text{borel } A)$ 
    using sets_restrict_space by fastforce
  moreover
  have continuous_on (A  $\cap$  - {a  $\in$  A.  $\neg$  continuous (at a within A) f}) f
    by (force simp: continuous_on_eq_continuous_within intro: continuous_within_subset)
  then have f  $\in$  borel_measurable (restrict_space (restrict_space borel A)
    (- {a  $\in$  A.  $\neg$  continuous (at a within A) f}))
    by (smt (verit, best) borel_measurable_continuous_on_restrict measurable_cong_sets
    sets_restrict_restrict_space)
  ultimately show ?thesis
    using measurable_restrict_countable[OF mono_on_ctble_discont[OF assms]]
    by (smt (verit, del_insts) UNIV_I mem_Collect_eq space_borel)
qed

```

```

lemma borel_measurable_piecewise_mono:
  fixes f::real  $\Rightarrow$  real and C::real set set
  assumes countable C  $\bigwedge c. c \in C \implies c \in \text{sets borel}$   $\bigwedge c. c \in C \implies \text{mono\_on } c$ 
  f ( $\bigcup C$ ) = UNIV
  shows f  $\in$  borel_measurable borel
  by (rule measurable_piecewise_restrict[of C], auto intro: borel_measurable_mono_on_fnc
  simp: assms)

```

```

lemma borel_measurable_mono:
  fixes f :: real  $\Rightarrow$  real
  shows mono f  $\implies$  f  $\in$  borel_measurable borel
  using borel_measurable_mono_on_fnc[of UNIV f] by (simp add: mono_def
  mono_on_def)

```

```

lemma measurable_bdd_below_real[measurable (raw)]:
  fixes F :: 'a  $\Rightarrow$  'i  $\Rightarrow$  real
  assumes [simp]: countable I and [measurable]:  $\bigwedge i. i \in I \implies F i \in M \rightarrow_M \text{borel}$ 
  shows Measurable.pred M ( $\lambda x. \text{bdd\_below } ((\lambda i. F i x)'I)$ )
proof (subst measurable_cong)
  show bdd_below (( $\lambda i. F i x$ )'I)  $\longleftrightarrow$  ( $\exists q \in \mathbb{Z}. \forall i \in I. q \leq F i x$ ) for x
    by (auto simp: bdd_below_def intro!: bexI[of _ of_int (floor _)] intro: order_trans of_int_floor_le)
  show Measurable.pred M ( $\lambda w. \exists q \in \mathbb{Z}. \forall i \in I. q \leq F i w$ )

```

using countable_int by measurable
qed

lemma borel_measurable_cINF_real[measurable (raw)]:

fixes $F :: _ \Rightarrow _ \Rightarrow \text{real}$

assumes [simp]: countable I

assumes $F[\text{measurable}]$: $\bigwedge i. i \in I \implies F i \in \text{borel_measurable } M$

shows $(\lambda x. \text{INF } i \in I. F i x) \in \text{borel_measurable } M$

proof (rule measurable_piecewise_restrict)

let $?\Omega = \{x \in \text{space } M. \text{bdd_below } ((\lambda i. F i x)'I)\}$

show countable $\{?\Omega, - ?\Omega\}$ $\text{space } M \subseteq \bigcup \{?\Omega, - ?\Omega\} \wedge X. X \in \{?\Omega, - ?\Omega\}$
 $\implies X \cap \text{space } M \in \text{sets } M$

by auto

fix X assume $X \in \{?\Omega, - ?\Omega\}$ then show $(\lambda x. \text{INF } i \in I. F i x) \in \text{borel_measurable}$
 $(\text{restrict_space } M X)$

proof safe

show $(\lambda x. \text{INF } i \in I. F i x) \in \text{borel_measurable } (\text{restrict_space } M ?\Omega)$

by (intro borel_measurable_cINF measurable_restrict_space1 F)

(auto simp: space_restrict_space)

show $(\lambda x. \text{INF } i \in I. F i x) \in \text{borel_measurable } (\text{restrict_space } M (-?\Omega))$

proof (subst measurable_cong)

fix x assume $x \in \text{space } (\text{restrict_space } M (-?\Omega))$

then have $\neg (\forall i \in I. - F i x \leq y)$ for y

by (auto simp: space_restrict_space bdd_above_def bdd_above_uminus[symmetric])

then show $(\text{INF } i \in I. F i x) = - (\text{THE } x. \text{False})$

by (auto simp: space_restrict_space Inf_real_def Sup_real_def Least_def

simp del: Set.ball_simps(10))

qed simp

qed

qed

lemma borel_Ici: borel = sigma UNIV (range $(\lambda x :: \text{real}. \{x ..\})$)

proof (safe intro!: borel_eq_sigmaI1[OF borel_Iio])

fix $x :: \text{real}$

have eq: $\{..<x\} = \text{space } (\text{sigma UNIV } (\text{range atLeast})) - \{x ..\}$

by auto

show $\{..<x\} \in \text{sets } (\text{sigma UNIV } (\text{range atLeast}))$

unfolding eq by (intro sets.compl_sets) auto

qed auto

lemma borel_measurable_pred_less[measurable (raw)]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, linorder_topology}\}$

shows $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies \text{Measurable.pred}$
 $M (\lambda w. f w < g w)$

unfolding Measurable.pred_def by (rule borel_measurable_less)

no_notation

eucl_less (infix <math>< e 50)

lemma *borel_measurable_Max2* [*measurable (raw)*]:
fixes $f :: _ \Rightarrow _ \Rightarrow 'a :: \{\text{second_countable_topology, dense_linorder, linorder_topology}\}$
assumes *finite I*
and [*measurable*]: $\bigwedge i. f\ i \in \text{borel_measurable } M$
shows $(\lambda x. \text{Max}\{f\ i\ x \mid i. i \in I\}) \in \text{borel_measurable } M$
by (*simp add: borel_measurable_Max[OF assms(1), where ?f=f and ?M=M]*
Setcompr_eq_image)

lemma *measurable_compose_n* [*measurable (raw)*]:
assumes $T \in \text{measurable } M\ M$
shows $(T^{\wedge} n) \in \text{measurable } M\ M$
by (*induction n, auto simp add: measurable_compose[OF _ assms]*)

lemma *measurable_real_imp_nat*:
fixes $f :: 'a \Rightarrow \text{nat}$
assumes [*measurable*]: $(\lambda x. \text{real}(f\ x)) \in \text{borel_measurable } M$
shows $f \in \text{measurable } M$ (*count_space UNIV*)
proof –
let $?g = (\lambda x. \text{real}(f\ x))$
have $\bigwedge (n :: \text{nat}). ?g^{-1}(\{\text{real } n\}) \cap \text{space } M = f^{-1}\{n\} \cap \text{space } M$ **by** *auto*
moreover **have** $\bigwedge (n :: \text{nat}). ?g^{-1}(\{\text{real } n\}) \cap \text{space } M \in \text{sets } M$ **using** *assms* **by**
measurable
ultimately **have** $\bigwedge (n :: \text{nat}). f^{-1}\{n\} \cap \text{space } M \in \text{sets } M$ **by** *simp*
then show *?thesis* **using** *measurable_count_space_eq2_countable* **by** *blast*
qed

lemma *measurable_equality_set* [*measurable*]:
fixes $f\ g :: _ \Rightarrow 'a :: \{\text{second_countable_topology, t2_space}\}$
assumes [*measurable*]: $f \in \text{borel_measurable } M$ $g \in \text{borel_measurable } M$
shows $\{x \in \text{space } M. f\ x = g\ x\} \in \text{sets } M$
proof –
define A **where** $A = \{x \in \text{space } M. f\ x = g\ x\}$
define B **where** $B = \{y. \exists x :: 'a. y = (x, x)\}$
have $A = (\lambda x. (f\ x, g\ x))^{-1} B \cap \text{space } M$ **unfolding** $A_def\ B_def$ **by** *auto*
moreover **have** $(\lambda x. (f\ x, g\ x)) \in \text{borel_measurable } M$ **by** *simp*
moreover **have** $B \in \text{sets borel}$ **unfolding** B_def **by** (*simp add: closed_diagonal*)
ultimately **have** $A \in \text{sets } M$ **by** *simp*
then show *?thesis* **unfolding** A_def **by** *simp*
qed

Logically equivalent to those with the opposite orientation, still these are needed

lemma *measurable_inequality_set_flipped*:
fixes $f\ g :: _ \Rightarrow 'a :: \{\text{second_countable_topology, linorder_topology}\}$
assumes [*measurable*]: $f \in \text{borel_measurable } M$ $g \in \text{borel_measurable } M$
shows $\{x \in \text{space } M. f\ x \geq g\ x\} \in \text{sets } M$
 $\{x \in \text{space } M. f\ x > g\ x\} \in \text{sets } M$
by *auto*

lemmas *measurable_inequality_set* [*measurable*] =
borel_measurable_le borel_measurable_less measurable_inequality_set_flipped

proposition *measurable_limit* [*measurable*]:

fixes $f::\text{nat} \Rightarrow 'a \Rightarrow 'b::\text{first_countable_topology}$
assumes [*measurable*]: $\bigwedge n::\text{nat}. f\ n \in \text{borel_measurable } M$
shows *Measurable.pred* $M (\lambda x. (\lambda n. f\ n\ x) \longrightarrow c)$

proof –

obtain $A :: \text{nat} \Rightarrow 'b$ **set where** A :

$\bigwedge i. \text{open } (A\ i)$

$\bigwedge i. c \in A\ i$

$\bigwedge S. \text{open } S \implies c \in S \implies \text{eventually } (\lambda i. A\ i \subseteq S)$ *sequentially*

by (*rule countable_basis_at_decseq*) *blast*

have [*measurable*]: $\bigwedge N\ i. (f\ N) \text{--}'(A\ i) \cap \text{space } M \in \text{sets } M$ **using** $A(1)$ **by** *auto*
then have *mes*: $(\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f\ N) \text{--}'(A\ i) \cap \text{space } M) \in \text{sets } M$ **by**
blast

have $(u \longrightarrow c) \iff (\forall i. \text{eventually } (\lambda n. u\ n \in A\ i))$ **for** $u::\text{nat}$
 $\Rightarrow 'b$

proof

assume $u \longrightarrow c$

then have *eventually* $(\lambda n. u\ n \in A\ i)$ *sequentially for i using* $A(1)$ [*of i*]
 $A(2)$ [*of i*]

by (*simp add: topological_tendstoD*)

then show $(\forall i. \text{eventually } (\lambda n. u\ n \in A\ i))$ *sequentially by auto*

next

assume H : $(\forall i. \text{eventually } (\lambda n. u\ n \in A\ i))$ *sequentially*

show $(u \longrightarrow c)$

proof (*rule topological_tendstoI*)

fix S **assume** *open S c ∈ S*

with $A(3)$ [*OF this*] **obtain** i **where** $A\ i \subseteq S$

using *eventually_False_sequentially eventually_mono* **by** *blast*

moreover have *eventually* $(\lambda n. u\ n \in A\ i)$ *sequentially using H* **by** *simp*

ultimately show $\forall_F n$ *in sequentially. u n ∈ S*

by (*simp add: eventually_mono subset_eq*)

qed

qed

then have $\{x. (\lambda n. f\ n\ x) \longrightarrow c\} = (\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f\ N) \text{--}'(A\ i))$

by (*auto simp add: atLeast_def eventually_at_top_linorder*)

then have $\{x \in \text{space } M. (\lambda n. f\ n\ x) \longrightarrow c\} = (\bigcap i. \bigcup n. \bigcap N \in \{n..\}. (f\ N) \text{--}'(A\ i) \cap \text{space } M)$

by *auto*

then have $\{x \in \text{space } M. (\lambda n. f\ n\ x) \longrightarrow c\} \in \text{sets } M$ **using** *mes* **by** *simp*

then show *?thesis* **by** *auto*

qed

lemma *measurable_limit2* [*measurable*]:

fixes $u::\text{nat} \Rightarrow 'a \Rightarrow \text{real}$

```

assumes [measurable]:  $\bigwedge n. u\ n \in \text{borel\_measurable } M\ v \in \text{borel\_measurable } M$ 
shows  $\text{Measurable.pred } M\ (\lambda x. (\lambda n. u\ n\ x) \longrightarrow v\ x)$ 
proof -
  define  $w$  where  $w = (\lambda n\ x. u\ n\ x - v\ x)$ 
  have [measurable]:  $w\ n \in \text{borel\_measurable } M$  for  $n$  unfolding  $w\_def$  by auto
  have  $((\lambda n. u\ n\ x) \longrightarrow v\ x) \longleftrightarrow ((\lambda n. w\ n\ x) \longrightarrow 0)$  for  $x$ 
    unfolding  $w\_def$  using Lim_null by auto
  then show ?thesis using measurable_limit by auto
qed

```

```

lemma measurable_P_restriction [measurable (raw)]:
  assumes [measurable]:  $\text{Measurable.pred } M\ P\ A \in \text{sets } M$ 
  shows  $\{x \in A. P\ x\} \in \text{sets } M$ 
proof -
  have  $A \subseteq \text{space } M$  using sets.sets_into_space[OF assms(2)].
  then have  $\{x \in A. P\ x\} = A \cap \{x \in \text{space } M. P\ x\}$  by blast
  then show ?thesis by auto
qed

```

```

lemma measurable_sum_nat [measurable (raw)]:
  fixes  $f :: 'c \Rightarrow 'a \Rightarrow \text{nat}$ 
  assumes  $\bigwedge i. i \in S \implies f\ i \in \text{measurable } M\ (\text{count\_space } \text{UNIV})$ 
  shows  $(\lambda x. \sum_{i \in S. f\ i\ x}) \in \text{measurable } M\ (\text{count\_space } \text{UNIV})$ 
proof cases
  assume finite S
  then show ?thesis using assms by induct auto
qed simp

```

```

lemma measurable_abs_powr [measurable]:
  fixes  $p :: \text{real}$ 
  assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
  shows  $(\lambda x. |f\ x| \text{ powr } p) \in \text{borel\_measurable } M$ 
  by simp

```

The next one is a variation around *measurable_restrict_space*.

```

lemma measurable_restrict_space3:
  assumes  $f \in \text{measurable } M\ N$  and
     $f \in A \rightarrow B$ 
  shows  $f \in \text{measurable } (\text{restrict\_space } M\ A)\ (\text{restrict\_space } N\ B)$ 
proof -
  have  $f \in \text{measurable } (\text{restrict\_space } M\ A)\ N$  using assms(1) measurable_restrict_space1
by auto
  then show ?thesis by (metis Int_iff funcsetI funcset_mem
    measurable_restrict_space2[of f, of restrict_space M A, of B, of N] assms(2)
    space_restrict_space)
qed

```

```

lemma measurable_restrict_mono:

```

```

assumes  $f: f \in \text{restrict\_space } M A \rightarrow_M N$  and  $B \subseteq A$ 
shows  $f \in \text{restrict\_space } M B \rightarrow_M N$ 
by (rule measurable_compose[OF measurable_restrict_space3 f]) (use  $\langle B \subseteq A \rangle$ )
in auto

```

The next one is a variation around *measurable_piecewise_restrict*.

```

lemma measurable_piecewise_restrict2:
assumes [measurable]:  $\bigwedge n. A n \in \text{sets } M$ 
and  $\text{space } M = (\bigcup (n::\text{nat}). A n)$ 
and  $\bigwedge n. \exists h \in \text{measurable } M N. (\forall x \in A n. f x = h x)$ 
shows  $f \in \text{measurable } M N$ 
proof (rule measurableI)
fix  $B$  assume [measurable]:  $B \in \text{sets } N$ 
{
fix  $n::\text{nat}$ 
obtain  $h$  where [measurable]:  $h \in \text{measurable } M N$  and  $\forall x \in A n. f x = h x$ 
using assms(3) by blast
then have  $*$ :  $f - 'B \cap A n = h - 'B \cap A n$  by auto
have  $h - 'B \cap A n = h - 'B \cap \text{space } M \cap A n$ 
using assms(2) sets.sets_into_space by auto
then have  $f - 'B \cap A n \in \text{sets } M$ 
by (simp add: *)
}
then have  $(\bigcup n. f - 'B \cap A n) \in \text{sets } M$ 
by measurable
moreover have  $f - 'B \cap \text{space } M = (\bigcup n. f - 'B \cap A n)$ 
using assms(2) by blast
ultimately show  $f - 'B \cap \text{space } M \in \text{sets } M$  by simp
next
fix  $x$  assume  $x \in \text{space } M$ 
then obtain  $n$  where  $x \in A n$ 
using assms(2) by blast
obtain  $h$  where [measurable]:  $h \in \text{measurable } M N$  and  $\forall x \in A n. f x = h x$ 
using assms(3) by blast
then show  $f x \in \text{space } N$ 
by (metis  $\langle x \in A n \rangle \langle x \in \text{space } M \rangle$  measurable_space)
qed

end

```

7.5 Lebesgue Integration for Nonnegative Functions

```

theory Nonnegative_Lebesgue_Integration
imports Measure_Space Borel_Space
begin

```


7.5.1 Approximating functions

lemma *AE_upper_bound_inf_ennreal*:

fixes $F G::'a \Rightarrow \text{ennreal}$

assumes $\bigwedge e. (e::\text{real}) > 0 \implies \text{AE } x \text{ in } M. F x \leq G x + e$

shows $\text{AE } x \text{ in } M. F x \leq G x$

proof –

have $\text{AE } x \text{ in } M. \forall n::\text{nat}. F x \leq G x + \text{ennreal } (1 / \text{Suc } n)$

using *assms* **by** (*auto simp: AE_all_countable*)

then show *?thesis*

proof (*eventually_elim*)

fix x **assume** $x: \forall n::\text{nat}. F x \leq G x + \text{ennreal } (1 / \text{Suc } n)$

show $F x \leq G x$

proof (*rule ennreal_le_epsilon*)

fix $e :: \text{real}$ **assume** $0 < e$

then obtain n **where** $n: 1 / \text{Suc } n < e$

by (*blast elim: nat_approx_posE*)

have $F x \leq G x + 1 / \text{Suc } n$

using x **by** *simp*

also have $\dots \leq G x + e$

using n **by** (*intro add_mono ennreal_leI*) *auto*

finally show $F x \leq G x + \text{ennreal } e$.

qed

qed

qed

lemma *AE_upper_bound_inf*:

fixes $F G::'a \Rightarrow \text{real}$

assumes $\bigwedge e. e > 0 \implies \text{AE } x \text{ in } M. F x \leq G x + e$

shows $\text{AE } x \text{ in } M. F x \leq G x$

proof –

have $\text{AE } x \text{ in } M. F x \leq G x + 1/\text{real } (n+1)$ **for** $n::\text{nat}$

by (*rule assms, auto*)

then have $\text{AE } x \text{ in } M. \forall n::\text{nat} \in \text{UNIV}. F x \leq G x + 1/\text{real } (n+1)$

by (*rule AE_ball_countable', auto*)

moreover

{

fix x **assume** $i: \forall n::\text{nat} \in \text{UNIV}. F x \leq G x + 1/\text{real } (n+1)$

have $(\lambda n. G x + 1/\text{real } (n+1)) \longrightarrow G x + 0$

by (*rule tendsto_add, simp, rule LIMSEQ_ignore_initial_segment[OF lim_1_over_n, of 1]*)

then have $F x \leq G x$ **using** i *LIMSEQ_le_const* **by** *fastforce*

}

ultimately show *?thesis* **by** *auto*

qed

lemma *not_AE_zero_ennreal_E*:

fixes $f::'a \Rightarrow \text{ennreal}$

assumes $\neg (\text{AE } x \text{ in } M. f x = 0)$ **and** [*measurable*]: $f \in \text{borel_measurable } M$

shows $\exists A \in \text{sets } M. \exists e::\text{real} > 0. \text{emeasure } M A > 0 \wedge (\forall x \in A. f x \geq e)$

2222

proof –

```
{ assume  $\neg (\exists e::\text{real} > 0. \{x \in \text{space } M. f x \geq e\} \notin \text{null\_sets } M)$ 
  then have  $0 < e \implies \text{AE } x \text{ in } M. f x \leq e$  for  $e :: \text{real}$ 
    by (auto simp: not_le less_imp_le dest!: AE_not_in)
  then have  $\text{AE } x \text{ in } M. f x \leq 0$ 
    by (intro AE_upper_bound_inf_enreal[where G= $\lambda_. 0$ ] simp)
  then have False
    using assms by auto }
then obtain  $e::\text{real}$  where  $e > 0 \{x \in \text{space } M. f x \geq e\} \notin \text{null\_sets } M$  by
auto
define  $A$  where  $A = \{x \in \text{space } M. f x \geq e\}$ 
have 1 [measurable]:  $A \in \text{sets } M$  unfolding  $A\_def$  by auto
have 2:  $\text{emeasure } M A > 0$ 
  using  $e(2)$   $A\_def$   $\langle A \in \text{sets } M \rangle$  by auto
have 3:  $\bigwedge x. x \in A \implies f x \geq e$  unfolding  $A\_def$  by auto
show ?thesis using  $e(1)$  1 2 3 by blast
```

qed

lemma *not_AE_zero_E*:

```
fixes  $f::'a \Rightarrow \text{real}$ 
assumes  $\text{AE } x \text{ in } M. f x \geq 0$ 
   $\neg(\text{AE } x \text{ in } M. f x = 0)$ 
  and [measurable]:  $f \in \text{borel\_measurable } M$ 
shows  $\exists A e. A \in \text{sets } M \wedge e > 0 \wedge \text{emeasure } M A > 0 \wedge (\forall x \in A. f x \geq e)$ 
proof –
  have  $\exists e. e > 0 \wedge \{x \in \text{space } M. f x \geq e\} \notin \text{null\_sets } M$ 
  proof (rule ccontr)
    assume *:  $\neg(\exists e. e > 0 \wedge \{x \in \text{space } M. f x \geq e\} \notin \text{null\_sets } M)$ 
    {
      fix  $e::\text{real}$  assume  $e > 0$ 
      then have  $\{x \in \text{space } M. f x \geq e\} \in \text{null\_sets } M$  using * by blast
      then have  $\text{AE } x \text{ in } M. x \notin \{x \in \text{space } M. f x \geq e\}$  using  $\text{AE\_not\_in}$  by
blast
      then have  $\text{AE } x \text{ in } M. f x \leq e$  by auto
    }
    then have  $\text{AE } x \text{ in } M. f x \leq 0$  by (rule AE_upper_bound_inf, auto)
    then have  $\text{AE } x \text{ in } M. f x = 0$  using  $\text{assms}(1)$  by auto
    then show False using  $\text{assms}(2)$  by auto
```

qed

```
then obtain  $e$  where  $e > 0 \{x \in \text{space } M. f x \geq e\} \notin \text{null\_sets } M$  by auto
define  $A$  where  $A = \{x \in \text{space } M. f x \geq e\}$ 
have 1 [measurable]:  $A \in \text{sets } M$  unfolding  $A\_def$  by auto
have 2:  $\text{emeasure } M A > 0$ 
  using  $e(2)$   $A\_def$   $\langle A \in \text{sets } M \rangle$  by auto
have 3:  $\bigwedge x. x \in A \implies f x \geq e$  unfolding  $A\_def$  by auto
show ?thesis
  using  $e(1)$  1 2 3 by blast
```

qed

7.5.2 Simple function

Our simple functions are not restricted to nonnegative real numbers. Instead they are just functions with a finite range and are measurable when singleton sets are measurable.

definition *simple_function* $M g \longleftrightarrow$
 $finite (g \text{ ' space } M) \wedge$
 $(\forall x \in g \text{ ' space } M. g \text{ - ' } \{x\} \cap \text{space } M \in \text{sets } M)$

lemma *simple_functionD*:
assumes *simple_function* $M g$
shows $finite (g \text{ ' space } M)$ **and** $g \text{ - ' } X \cap \text{space } M \in \text{sets } M$
proof –
show $finite (g \text{ ' space } M)$
using *assms* **unfolding** *simple_function_def* **by** *auto*
have $g \text{ - ' } X \cap \text{space } M = g \text{ - ' } (X \cap g \text{ ' space } M) \cap \text{space } M$ **by** *auto*
also have $\dots = (\bigcup x \in X \cap g \text{ ' space } M. g \text{ - ' } \{x\} \cap \text{space } M)$ **by** *auto*
finally show $g \text{ - ' } X \cap \text{space } M \in \text{sets } M$ **using** *assms*
by (*auto simp del: UN_simps simp: simple_function_def*)
qed

lemma *measurable_simple_function*[*measurable_dest*]:
 $simple_function\ M\ f \implies f \in \text{measurable } M\ (\text{count_space } UNIV)$
unfolding *simple_function_def* *measurable_def*
proof *safe*
fix A **assume** $finite (f \text{ ' space } M) \forall x \in f \text{ ' space } M. f \text{ - ' } \{x\} \cap \text{space } M \in \text{sets } M$
then have $(\bigcup x \in f \text{ ' space } M. \text{if } x \in A \text{ then } f \text{ - ' } \{x\} \cap \text{space } M \text{ else } \{\}) \in \text{sets } M$
 M
by (*intro sets.finite_UN*) *auto*
also have $(\bigcup x \in f \text{ ' space } M. \text{if } x \in A \text{ then } f \text{ - ' } \{x\} \cap \text{space } M \text{ else } \{\}) = f \text{ - ' } A \cap \text{space } M$
by (*auto split: if_split_asm*)
finally show $f \text{ - ' } A \cap \text{space } M \in \text{sets } M$.
qed *simp*

lemma *borel_measurable_simple_function*:
 $simple_function\ M\ f \implies f \in \text{borel_measurable } M$
by (*auto dest!: measurable_simple_function simp: measurable_def*)

lemma *simple_function_measurable2*[*intro*]:
assumes *simple_function* $M f$ *simple_function* $M g$
shows $f \text{ - ' } A \cap g \text{ - ' } B \cap \text{space } M \in \text{sets } M$
proof –
have $f \text{ - ' } A \cap g \text{ - ' } B \cap \text{space } M = (f \text{ - ' } A \cap \text{space } M) \cap (g \text{ - ' } B \cap \text{space } M)$
by *auto*
then show *?thesis* **using** *assms*[*THEN simple_functionD(2)*] **by** *auto*
qed

lemma *simple_function_indicator_representation*:

```

fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
assumes  $f: \text{simple\_function } M f$  and  $x: x \in \text{space } M$ 
shows  $f x = (\sum y \in f \text{ ' space } M. y * \text{indicator } (f \text{ - ' } \{y\} \cap \text{space } M) x)$ 
(is ?l = ?r)
proof -
  have  $?r = (\sum y \in f \text{ ' space } M.$ 
     $(\text{if } y = f x \text{ then } y * \text{indicator } (f \text{ - ' } \{y\} \cap \text{space } M) x \text{ else } 0))$ 
    by  $(\text{auto intro!: sum.cong})$ 
  also have  $\dots = f x * \text{indicator } (f \text{ - ' } \{f x\} \cap \text{space } M) x$ 
    using  $\text{assms by } (\text{auto dest: simple\_functionD})$ 
  also have  $\dots = f x$  using  $x$  by  $(\text{auto simp: indicator\_def})$ 
  finally show  $?thesis$  by  $\text{auto}$ 
qed

```

```

lemma  $\text{simple\_function\_notspace}$ :
   $\text{simple\_function } M (\lambda x. h x * \text{indicator } (- \text{space } M) x :: \text{ennreal})$  (is simple\_function
   $M ?h)$ 
proof -
  have  $?h \text{ ' space } M \subseteq \{0\}$  unfolding  $\text{indicator\_def}$  by  $\text{auto}$ 
  hence  $[\text{simp, intro}]: \text{finite } (?h \text{ ' space } M)$  by  $(\text{auto intro: finite\_subset})$ 
  have  $?h \text{ - ' } \{0\} \cap \text{space } M = \text{space } M$  by  $\text{auto}$ 
  thus  $?thesis$  unfolding  $\text{simple\_function\_def}$  by  $(\text{auto simp add: image\_constant\_conv})$ 
qed

```

```

lemma  $\text{simple\_function\_cong}$ :
  assumes  $\bigwedge t. t \in \text{space } M \implies f t = g t$ 
  shows  $\text{simple\_function } M f \longleftrightarrow \text{simple\_function } M g$ 
proof -
  have  $\bigwedge x. f \text{ - ' } \{x\} \cap \text{space } M = g \text{ - ' } \{x\} \cap \text{space } M$ 
    using  $\text{assms by auto}$ 
  with  $\text{assms show } ?thesis$ 
    by  $(\text{simp add: simple\_function\_def cong: image\_cong})$ 
qed

```

```

lemma  $\text{simple\_function\_cong\_algebra}$ :
  assumes  $\text{sets } N = \text{sets } M$   $\text{space } N = \text{space } M$ 
  shows  $\text{simple\_function } M f \longleftrightarrow \text{simple\_function } N f$ 
  unfolding  $\text{simple\_function\_def}$   $\text{assms ..}$ 

```

```

lemma  $\text{simple\_function\_borel\_measurable}$ :
  fixes  $f :: 'a \Rightarrow 'x :: \{t2\_space\}$ 
  assumes  $f \in \text{borel\_measurable } M$  and  $\text{finite } (f \text{ ' space } M)$ 
  shows  $\text{simple\_function } M f$ 
  using  $\text{assms unfolding simple\_function\_def}$ 
  by  $(\text{auto intro: borel\_measurable\_vimage})$ 

```

```

lemma  $\text{simple\_function\_iff\_borel\_measurable}$ :
  fixes  $f :: 'a \Rightarrow 'x :: \{t2\_space\}$ 
  shows  $\text{simple\_function } M f \longleftrightarrow \text{finite } (f \text{ ' space } M) \wedge f \in \text{borel\_measurable } M$ 

```

by (metis borel_measurable_simple_function simple_functionD(1) simple_function_borel_measurable)

lemma *simple_function_eq_measurable*:

simple_function M $f \longleftrightarrow$ *finite* (f '*space* M) $\wedge f \in$ *measurable* M (*count_space* *UNIV*)

using *measurable_simple_function*[of M f] **by** (*fastforce simp: simple_function_def*)

lemma *simple_function_const*[*intro, simp*]:

simple_function M ($\lambda x. c$)

by (*auto intro: finite_subset simp: simple_function_def*)

lemma *simple_function_compose*[*intro, simp*]:

assumes *simple_function* M f

shows *simple_function* M ($g \circ f$)

unfolding *simple_function_def*

proof *safe*

show *finite* ($(g \circ f)$ '*space* M)

using *assms unfolding simple_function_def image_comp [symmetric]* **by** *auto*

next

fix x **assume** $x \in$ *space* M

let $?G = g$ - ' $\{g(f\ x)\} \cap (f$ '*space* $M)$

have $*$: $(g \circ f)$ - ' $\{(g \circ f)\ x\} \cap$ *space* $M =$

$(\bigcup_{x \in ?G. f$ - ' $\{x\} \cap$ *space* $M)$ **by** *auto*

show $(g \circ f)$ - ' $\{(g \circ f)\ x\} \cap$ *space* $M \in$ *sets* M

using *assms unfolding simple_function_def **

by (*rule_tac sets.finite_UN*) *auto*

qed

lemma *simple_function_indicator*[*intro, simp*]:

assumes $A \in$ *sets* M

shows *simple_function* M (*indicator* A)

proof -

have *indicator* A '*space* $M \subseteq \{0, 1\}$ (**is** $?S \subseteq _$)

by (*auto simp: indicator_def*)

hence *finite* $?S$ **by** (*rule finite_subset*) *simp*

moreover **have** $- A \cap$ *space* $M =$ *space* $M - A$ **by** *auto*

ultimately **show** $?thesis$ **unfolding** *simple_function_def*

using *assms* **by** (*auto simp: indicator_def [abs_def]*)

qed

lemma *simple_function_Pair*[*intro, simp*]:

assumes *simple_function* M f

assumes *simple_function* M g

shows *simple_function* M ($\lambda x. (f\ x, g\ x)$) (**is** *simple_function* M $?p$)

unfolding *simple_function_def*

proof *safe*

show *finite* ($?p$ '*space* M)

using *assms unfolding simple_function_def*

by (*rule_tac finite_subset*[of $_ f$ '*space* $M \times g$ '*space* M]) *auto*

next

```

fix  $x$  assume  $x \in \text{space } M$ 
have  $(\lambda x. (f x, g x)) -' \{(f x, g x)\} \cap \text{space } M =$ 
   $(f -' \{f x\} \cap \text{space } M) \cap (g -' \{g x\} \cap \text{space } M)$ 
by auto
with  $\langle x \in \text{space } M \rangle$  show  $(\lambda x. (f x, g x)) -' \{(f x, g x)\} \cap \text{space } M \in \text{sets } M$ 
using assms unfolding simple_function_def by auto
qed

```

```

lemma simple_function_compose1:
assumes simple_function  $M f$ 
shows simple_function  $M (\lambda x. g (f x))$ 
using simple_function_compose[OF assms, of g]
by (simp add: comp_def)

```

```

lemma simple_function_compose2:
assumes simple_function  $M f$  and simple_function  $M g$ 
shows simple_function  $M (\lambda x. h (f x) (g x))$ 
proof -
have simple_function  $M ((\lambda(x, y). h x y) \circ (\lambda x. (f x, g x)))$ 
using assms by auto
thus ?thesis by (simp_all add: comp_def)
qed

```

```

lemmas simple_function_add[intro, simp] = simple_function_compose2[where
 $h=(+)$ ]
and simple_function_diff[intro, simp] = simple_function_compose2[where  $h=(-)$ ]
and simple_function_uminus[intro, simp] = simple_function_compose[where
 $g=\text{uminus}$ ]
and simple_function_mult[intro, simp] = simple_function_compose2[where
 $h=(*)$ ]
and simple_function_div[intro, simp] = simple_function_compose2[where  $h=(/)$ ]
and simple_function_inverse[intro, simp] = simple_function_compose[where
 $g=\text{inverse}$ ]
and simple_function_max[intro, simp] = simple_function_compose2[where  $h=\text{max}$ ]

```

```

lemma simple_function_sum[intro, simp]:
assumes  $\bigwedge i. i \in P \implies \text{simple\_function } M (f i)$ 
shows simple_function  $M (\lambda x. \sum_{i \in P}. f i x)$ 
proof cases
assume finite P from this assms show ?thesis by induct auto
qed auto

```

```

lemma simple_function_ennreal[intro, simp]:
fixes  $f g :: 'a \Rightarrow \text{real}$  assumes sf: simple_function  $M f$ 
shows simple_function  $M (\lambda x. \text{ennreal } (f x))$ 
by (rule simple_function_compose1[OF sf])

```

```

lemma simple_function_real_of_nat[intro, simp]:
fixes  $f g :: 'a \Rightarrow \text{nat}$  assumes sf: simple_function  $M f$ 

```

shows *simple_function* M $(\lambda x. \text{real } (f x))$
by (*rule simple_function_compose1* [*OF sf*])

lemma *borel_measurable_implies_simple_function_sequence*:

fixes $u :: 'a \Rightarrow \text{ennreal}$

assumes $u[\text{measurable}]$: $u \in \text{borel_measurable } M$

shows $\exists f. \text{incseq } f \wedge (\forall i. (\forall x. f i x < \text{top}) \wedge \text{simple_function } M (f i)) \wedge u =$
 $(\text{SUP } i. f i)$

proof –

define f **where** [*abs_def*]:

$f i x = \text{real_of_int } (\text{floor } (\text{enn2real } (\text{min } i (u x)) * 2^i)) / 2^i$ **for** $i x$

have [*simp*]: $0 \leq f i x$ **for** $i x$

by (*auto simp: f_def intro!: divide_nonneg_nonneg mult_nonneg_nonneg*
enn2real_nonneg)

have *: $2^n * \text{real_of_int } x = \text{real_of_int } (2^n * x)$ **for** $n x$

by *simp*

have $\text{real_of_int } [\text{real } i * 2^i] = \text{real_of_int } [i * 2^i]$ **for** i

by (*intro arg_cong[where f=real_of_int] simp*)

then have [*simp*]: $\text{real_of_int } [\text{real } i * 2^i] = i * 2^i$ **for** i

unfolding *floor_of_nat* **by** *simp*

have *incseq* f

proof (*intro monoI le_funI*)

fix $m n :: \text{nat}$ **and** x **assume** $m \leq n$

moreover

{ **fix** $d :: \text{nat}$

have $[2^d::\text{real}] * [2^m * \text{enn2real } (\text{min } (\text{of_nat } m) (u x))] \leq$
 $[2^d * (2^m * \text{enn2real } (\text{min } (\text{of_nat } m) (u x)))]$

by (*rule le_mult_floor*) (*auto*)

also have ... $\leq [2^d * (2^m * \text{enn2real } (\text{min } (\text{of_nat } d + \text{of_nat } m) (u$
 $x)))]$

by (*intro floor_mono mult_mono enn2real_mono min.mono*)

(*auto simp: min_less_iff_disj of_nat_less_top*)

finally have $f m x \leq f (m + d) x$

unfolding *f_def*

by (*auto simp: field_simps power_add * simp del: of_int_mult*) }

ultimately show $f m x \leq f n x$

by (*auto simp add: le_iff_add*)

qed

then have *inc_f*: *incseq* $(\lambda i. \text{ennreal } (f i x))$ **for** x

by (*auto simp: incseq_def le_fun_def*)

then have *incseq* $(\lambda i x. \text{ennreal } (f i x))$

by (*auto simp: incseq_def le_fun_def*)

moreover

have *simple_function* $M (f i)$ **for** i

proof (*rule simple_function_borel_measurable*)

```

have ⌊enn2real (min (of_nat i) (u x)) * 2 ^ i⌋ ≤ ⌊int i * 2 ^ i⌋ for x
  by (cases u x rule: ennreal_cases)
    (auto split: split_min intro!: floor_mono)
then have f i ' space M ⊆ (λn. real_of_int n / 2 ^ i) ' {0 .. of_nat i * 2 ^ i}
  unfolding floor_of_int by (auto simp: f_def intro!: imageI)
then show finite (f i ' space M)
  by (rule finite_subset) auto
show f i ∈ borel_measurable M
  unfolding f_def enn2real_def by measurable
qed
moreover
{ fix x
  have (SUP i. ennreal (f i x)) = u x
  proof (cases u x rule: ennreal_cases)
    case top then show ?thesis
      by (simp add: f_def inf_min[symmetric] ennreal_of_nat_eq_real_of_nat[symmetric]
        ennreal_SUP_of_nat_eq_top)
  next
    case (real r)
    obtain n where r ≤ of_nat n using real_arch_simple by auto
    then have min_eq_r: ∀_F x in sequentially. min (real x) r = r
      by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)

    have (λi. real_of_int ⌊min (real i) r * 2 ^ i⌋ / 2 ^ i) → r
    proof (rule tendsto_sandwich)
      show (λn. r - (1/2) ^ n) → r
        by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
      show ∀_F n in sequentially. real_of_int ⌊min (real n) r * 2 ^ n⌋ / 2 ^ n ≤
        r
        using min_eq_r by eventually_elim (auto simp: field_simps)
      have *: r * (2 ^ n * 2 ^ n) ≤ 2 ^ n + 2 ^ n * real_of_int ⌊r * 2 ^ n⌋ for n
        using real_of_int_floor_ge_diff_one[of r * 2 ^ n, THEN mult_left_mono,
of 2 ^ n]
        by (auto simp: field_simps)
      show ∀_F n in sequentially. r - (1/2) ^ n ≤ real_of_int ⌊min (real n) r *
2 ^ n⌋ / 2 ^ n
        using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
    qed auto
    then have (λi. ennreal (f i x)) → ennreal r
      by (simp add: real_f_def ennreal_of_nat_eq_real_of_nat min_ennreal)
    from LIMSEQ_unique[OF LIMSEQ_SUP[OF inc_f] this]
    show ?thesis
      by (simp add: real)
  }
ultimately show ?thesis
  by (intro exI [of _ λi x. ennreal (f i x)]) (auto simp add: image_comp)
qed

```

lemma borel_measurable_implies_simple_function_sequence':


```

fixes u :: 'a  $\Rightarrow$  ennreal
assumes u: u  $\in$  borel_measurable M
obtains f where
   $\bigwedge i.$  simple_function M (f i) incseq f  $\bigwedge i x.$  f i x < top  $\bigwedge x.$  (SUP i. f i x) = u x
using borel_measurable_implies_simple_function_sequence [OF u]
by (metis SUP_apply)

lemma simple_function_induct
  [consumes 1, case_names cong set mult add, induct set: simple_function]:
  fixes u :: 'a  $\Rightarrow$  ennreal
  assumes u: simple_function M u
  assumes cong:  $\bigwedge f g.$  simple_function M f  $\Longrightarrow$  simple_function M g  $\Longrightarrow$  (AE x
  in M. f x = g x)  $\Longrightarrow$  P f  $\Longrightarrow$  P g
  assumes set:  $\bigwedge A.$  A  $\in$  sets M  $\Longrightarrow$  P (indicator A)
  assumes mult:  $\bigwedge u c.$  P u  $\Longrightarrow$  P ( $\lambda x.$  c * u x)
  assumes add:  $\bigwedge u v.$  P u  $\Longrightarrow$  P v  $\Longrightarrow$  P ( $\lambda x.$  v x + u x)
  shows P u
proof (rule cong)
  from AE_space show AE x in M. ( $\sum y \in u$  ' space M. y * indicator (u -' {y}
   $\cap$  space M) x) = u x
  proof eventually_elim
    fix x assume x: x  $\in$  space M
    from simple_function_indicator_representation[OF u x]
    show ( $\sum y \in u$  ' space M. y * indicator (u -' {y}  $\cap$  space M) x) = u x ..
  qed
next
  from u have finite (u ' space M)
  unfolding simple_function_def by auto
  then show P ( $\lambda x.$   $\sum y \in u$  ' space M. y * indicator (u -' {y}  $\cap$  space M) x)
  proof induct
    case empty show ?case
    using set[of {}] by (simp add: indicator_def[abs_def])
  qed (auto intro!: add_mult_set simple_functionD u)
next
  show simple_function M ( $\lambda x.$  ( $\sum y \in u$  ' space M. y * indicator (u -' {y}  $\cap$ 
  space M) x))
  apply (subst simple_function_cong)
  apply (rule simple_function_indicator_representation[symmetric])
  apply (auto intro: u)
  done
qed fact

```

```

lemma simple_function_induct_nn[consumes 1, case_names cong set mult add]:
  fixes u :: 'a  $\Rightarrow$  ennreal
  assumes u: simple_function M u
  assumes cong:  $\bigwedge f g.$  simple_function M f  $\Longrightarrow$  simple_function M g  $\Longrightarrow$  ( $\bigwedge x.$  x
   $\in$  space M  $\Longrightarrow$  f x = g x)  $\Longrightarrow$  P f  $\Longrightarrow$  P g
  assumes set:  $\bigwedge A.$  A  $\in$  sets M  $\Longrightarrow$  P (indicator A)
  assumes mult:  $\bigwedge u c.$  simple_function M u  $\Longrightarrow$  P u  $\Longrightarrow$  P ( $\lambda x.$  c * u x)

```

```

assumes add:  $\bigwedge u v. \text{simple\_function } M u \implies P u \implies \text{simple\_function } M v$ 
 $\implies (\bigwedge x. x \in \text{space } M \implies u x = 0 \vee v x = 0) \implies P v \implies P (\lambda x. v x + u x)$ 
shows  $P u$ 
proof -
  show ?thesis
  proof (rule cong)
    fix  $x$  assume  $x: x \in \text{space } M$ 
    from  $\text{simple\_function\_indicator\_representation}[OF u x]$ 
    show  $(\sum_{y \in u} \text{'space } M. y * \text{indicator } (u - \{y\} \cap \text{space } M) x) = u x ..$ 
  next
    show  $\text{simple\_function } M (\lambda x. (\sum_{y \in u} \text{'space } M. y * \text{indicator } (u - \{y\} \cap \text{space } M) x))$ 
    apply (subst simple_function_cong)
    apply (rule simple_function_indicator_representation[symmetric])
    apply (auto intro: u)
    done
  next
    from  $u$  have finite  $(u \text{'space } M)$ 
    unfolding simple_function_def by auto
    then show  $P (\lambda x. \sum_{y \in u} \text{'space } M. y * \text{indicator } (u - \{y\} \cap \text{space } M) x)$ 
    proof induct
      case empty show ?case
        using set[of {}] by (simp add: indicator_def[abs_def])
      next
        case (insert  $x S$ )
        { fix  $z$  have  $(\sum_{y \in S} y * \text{indicator } (u - \{y\} \cap \text{space } M) z) = 0 \vee$ 
           $x * \text{indicator } (u - \{x\} \cap \text{space } M) z = 0$ 
          using insert by (subst sum_eq_0_iff) (auto simp: indicator_def) }
        note disj = this
        from insert show ?case
          by (auto intro!: add_mult_set simple_functionD u simple_function_sum disj)
      qed
    qed fact
  qed

```

lemma borel_measurable_induct

```

[consumes 1, case_names cong set mult add seq, induct set: borel_measurable]:
fixes  $u :: 'a \Rightarrow \text{ennreal}$ 
assumes  $u: u \in \text{borel\_measurable } M$ 
assumes cong:  $\bigwedge f g. f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies$ 
 $(\bigwedge x. x \in \text{space } M \implies f x = g x) \implies P g \implies P f$ 
assumes set:  $\bigwedge A. A \in \text{sets } M \implies P (\text{indicator } A)$ 
assumes mult':  $\bigwedge u c. c < \text{top} \implies u \in \text{borel\_measurable } M \implies (\bigwedge x. x \in \text{space } M \implies u x < \text{top}) \implies P u \implies P (\lambda x. c * u x)$ 
assumes add:  $\bigwedge u v. u \in \text{borel\_measurable } M \implies (\bigwedge x. x \in \text{space } M \implies u x < \text{top}) \implies P u \implies v \in \text{borel\_measurable } M \implies (\bigwedge x. x \in \text{space } M \implies v x < \text{top}) \implies$ 
 $(\bigwedge x. x \in \text{space } M \implies u x = 0 \vee v x = 0) \implies P v \implies P (\lambda x. v x + u x)$ 
assumes seq:  $\bigwedge U. (\bigwedge i. U i \in \text{borel\_measurable } M) \implies (\bigwedge i x. x \in \text{space } M \implies U i x < \text{top}) \implies (\bigwedge i. P (U i)) \implies \text{incseq } U \implies u = (\text{SUP } i. U i) \implies P (\text{SUP } i. U i)$ 

```

```

i. U i)
  shows P u
  using u
proof (induct rule: borel_measurable_implies_simple_function_sequence)
  fix U assume U:  $\bigwedge i. \text{simple\_function } M (U i) \text{ incseq } U \bigwedge i x. U i x < \text{top}$  and
sup:  $\bigwedge x. (\text{SUP } i. U i x) = u x$ 
  have u_eq:  $u = (\text{SUP } i. U i)$ 
  using u by (auto simp add: image_comp sup)

  have not_inf:  $\bigwedge x i. x \in \text{space } M \implies U i x < \text{top}$ 
  using U by (auto simp: image_iff eq_commute)

  from U have  $\bigwedge i. U i \in \text{borel\_measurable } M$ 
  by (simp add: borel_measurable_simple_function)

  show P u
  unfolding u_eq
proof (rule seq)
  fix i show P (U i)
  using  $\langle \text{simple\_function } M (U i) \rangle \text{ not\_inf}[of \_ i]$ 
proof (induct rule: simple_function_induct_nn)
  case (mult u c)
  show ?case
  proof cases
    assume  $c = 0 \vee \text{space } M = \{\}$   $\vee (\forall x \in \text{space } M. u x = 0)$ 
    with mult(1) show ?thesis
    by (intro cong[of  $\lambda x. c * u x \text{ indicator } \{\}$ ] set)
      (auto dest!: borel_measurable_simple_function)
  next
    assume  $\neg (c = 0 \vee \text{space } M = \{\}) \vee (\forall x \in \text{space } M. u x = 0)$ 
    then obtain x where  $\text{space } M \neq \{\}$  and  $x: x \in \text{space } M \ u x \neq 0 \ c \neq 0$ 
    by auto
    with mult(3)[of x] have  $c < \text{top}$ 
    by (auto simp: ennreal_mult_less_top)
    then have u_fin:  $x' \in \text{space } M \implies u x' < \text{top}$  for  $x'$ 
    using mult(3)[of x']  $\langle c \neq 0 \rangle$  by (auto simp: ennreal_mult_less_top)
    then have P u
    by (rule mult)
    with u_fin  $\langle c < \text{top} \rangle$  mult(1) show ?thesis
    by (intro mult') (auto dest!: borel_measurable_simple_function)
  qed
qed (auto intro: cong intro!: set add dest!: borel_measurable_simple_function)
qed fact+
qed

```

lemma simple_function>If_set:

```

  assumes sf:  $\text{simple\_function } M \ f \ \text{simple\_function } M \ g$  and A:  $A \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{simple\_function } M (\lambda x. \text{if } x \in A \text{ then } f x \text{ else } g x)$  (is simple_function

```

```

M ?IF)
proof –
  define F where F x = f -‘ {x} ∩ space M for x
  define G where G x = g -‘ {x} ∩ space M for x
  show ?thesis unfolding simple_function_def
  proof safe
    have ?IF -‘ space M ⊆ f -‘ space M ∪ g -‘ space M by auto
    from finite_subset[OF this] assms
    show finite (?IF -‘ space M) unfolding simple_function_def by auto
  next
    fix x assume x ∈ space M
    then have *: ?IF -‘ {?IF x} ∩ space M = (if x ∈ A
      then ((F (f x) ∩ (A ∩ space M)) ∪ (G (f x) - (G (f x) ∩ (A ∩ space M))))
      else ((F (g x) ∩ (A ∩ space M)) ∪ (G (g x) - (G (g x) ∩ (A ∩ space M))))))
    using sets.sets_into_space[OF A] by (auto split: if_split_asm simp: G_def
F_def)
    have [intro]: ∧x. F x ∈ sets M ∧x. G x ∈ sets M
    unfolding F_def G_def using sf[THEN simple_functionD(2)] by auto
    show ?IF -‘ {?IF x} ∩ space M ∈ sets M unfolding * using A by auto
  qed
qed

```

```

lemma simple_function_If:
  assumes sf: simple_function M f simple_function M g and P: {x ∈ space M. P
x} ∈ sets M
  shows simple_function M (λx. if P x then f x else g x)
proof –
  have {x ∈ space M. P x} = {x. P x} ∩ space M by auto
  with simple_function_If_set[OF sf, of {x. P x}] P show ?thesis by simp
qed

```

```

lemma simple_function_subalgebra:
  assumes simple_function N f
  and N_subalgebra: sets N ⊆ sets M space N = space M
  shows simple_function M f
  using assms unfolding simple_function_def by auto

```

```

lemma simple_function_comp:
  assumes T: T ∈ measurable M M'
  and f: simple_function M' f
  shows simple_function M (λx. f (T x))
proof (intro simple_function_def[THEN iffD2] conjI ballI)
  have (λx. f (T x)) -‘ space M ⊆ f -‘ space M'
  using T unfolding measurable_def by auto
  then show finite ((λx. f (T x)) -‘ space M)
  using f unfolding simple_function_def by (auto intro: finite_subset)
  fix i assume i: i ∈ (λx. f (T x)) -‘ space M
  then have i ∈ f -‘ space M'
  using T unfolding measurable_def by auto

```

```

then have  $f - \{i\} \cap \text{space } M' \in \text{sets } M'$ 
using  $f$  unfolding  $\text{simple\_function\_def}$  by auto
then have  $T - \{f - \{i\} \cap \text{space } M'\} \cap \text{space } M \in \text{sets } M$ 
using  $T$  unfolding  $\text{measurable\_def}$  by auto
also have  $T - \{f - \{i\} \cap \text{space } M'\} \cap \text{space } M = (\lambda x. f (T x)) - \{i\} \cap \text{space } M$ 
using  $T$  unfolding  $\text{measurable\_def}$  by auto
finally show  $(\lambda x. f (T x)) - \{i\} \cap \text{space } M \in \text{sets } M$  .
qed

```

7.5.3 Simple integral

definition $\text{simple_integral} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{ennreal}) \Rightarrow \text{ennreal} (\text{integral}^S)$

where

$$\text{integral}^S M f = (\sum x \in f - \{x\} \cap \text{space } M. x * \text{emeasure } M (f - \{x\} \cap \text{space } M))$$

syntax

$\text{simple_integral} :: \text{pttrn} \Rightarrow \text{ennreal} \Rightarrow 'a \text{ measure} \Rightarrow \text{ennreal} (\int^S _ . _ \partial _ [60,61] 110)$

translations

$$\int^S x. f \partial M == \text{CONST } \text{simple_integral } M (\%x. f)$$

lemma $\text{simple_integral_cong}$:

assumes $\bigwedge t. t \in \text{space } M \implies f t = g t$

shows $\text{integral}^S M f = \text{integral}^S M g$

proof –

have $f - \{x\} \cap \text{space } M = g - \{x\} \cap \text{space } M$

$\bigwedge x. f - \{x\} \cap \text{space } M = g - \{x\} \cap \text{space } M$

using assms **by** $(\text{auto intro!: image_eqI})$

thus thesis **unfolding** $\text{simple_integral_def}$ **by simp**

qed

lemma $\text{simple_integral_const}[\text{simp}]$:

$$(\int^S x. c \partial M) = c * (\text{emeasure } M) (\text{space } M)$$

proof $(\text{cases } \text{space } M = \{\})$

case True **thus** thesis **unfolding** $\text{simple_integral_def}$ **by simp**

next

case False **hence** $(\lambda x. c) - \{c\} \cap \text{space } M = \{c\}$ **by auto**

thus thesis **unfolding** $\text{simple_integral_def}$ **by simp**

qed

lemma $\text{simple_function_partition}$:

assumes f : $\text{simple_function } M f$ **and** g : $\text{simple_function } M g$

assumes sub: $\bigwedge x y. x \in \text{space } M \implies y \in \text{space } M \implies g x = g y \implies f x = f y$

assumes v: $\bigwedge x. x \in \text{space } M \implies f x = v (g x)$

shows $\text{integral}^S M f = (\sum y \in g - \{y\} \cap \text{space } M. v y * \text{emeasure } M \{x \in \text{space } M. g x = y\})$

(is $_ = ?r$ **)**

proof –

from $f g$ **have** $[simp]: finite (f'space M) finite (g'space M)$
by $(auto simp: simple_function_def)$
from $f g$ **have** $[measurable]: f \in measurable M (count_space UNIV) g \in measurable M (count_space UNIV)$
by $(auto intro: measurable_simple_function)$

{ fix y **assume** $y \in space M$
then have $f \text{ ' } space M \cap \{i. \exists x \in space M. i = f x \wedge g y = g x\} = \{v (g y)\}$
by $(auto cong: sub simp: v[symmetric])$ **}**
note $eq = this$

have $integral^S M f =$
 $(\sum y \in f'space M. y * (\sum z \in g'space M.$
 $if \exists x \in space M. y = f x \wedge z = g x \text{ then } emeasure M \{x \in space M. g x = z\}$
 $else 0))$

unfolding $simple_integral_def$

proof $(safe intro!: sum.cong ennreal_mult_left_cong)$

fix y **assume** $y: y \in space M f y \neq 0$

have $[simp]: g \text{ ' } space M \cap \{z. \exists x \in space M. f y = f x \wedge z = g x\} =$
 $\{z. \exists x \in space M. f y = f x \wedge z = g x\}$

by $auto$

have $eq: (\bigcup i \in \{z. \exists x \in space M. f y = f x \wedge z = g x\}. \{x \in space M. g x = i\})$
 $=$

$f \text{ - ' } \{f y\} \cap space M$

by $(auto simp: eq_commute cong: sub rev_conj_cong)$

have $finite (g'space M)$ **by** $simp$

then have $finite \{z. \exists x \in space M. f y = f x \wedge z = g x\}$

by $(rule rev_finite_subset) auto$

then show $emeasure M (f \text{ - ' } \{f y\} \cap space M) =$

$(\sum z \in g \text{ ' } space M. if \exists x \in space M. f y = f x \wedge z = g x \text{ then } emeasure M \{x \in space M. g x = z\} else 0)$

apply $(simp add: sum.If_cases)$

apply $(subst sum_emeasure)$

apply $(auto simp: disjoint_family_on_def eq)$

done

qed

also have $\dots = (\sum y \in f'space M. (\sum z \in g'space M.$

$if \exists x \in space M. y = f x \wedge z = g x \text{ then } y * emeasure M \{x \in space M. g x = z\} else 0))$

by $(auto intro!: sum.cong simp: sum_distrib_left)$

also have $\dots = ?r$

by $(subst sum.swap)$

$(auto intro!: sum.cong simp: sum.If_cases scaleR_sum_right[symmetric] eq)$

finally show $integral^S M f = ?r$.

qed

lemma $simple_integral_add[simp]:$

assumes $f: simple_function M f$ **and** $\bigwedge x. 0 \leq f x$ **and** $g: simple_function M g$

and $\bigwedge x. 0 \leq g\ x$
shows $(\int^S x. f\ x + g\ x\ \partial M) = \text{integral}^S\ M\ f + \text{integral}^S\ M\ g$
proof –
have $(\int^S x. f\ x + g\ x\ \partial M) =$
 $(\sum y \in (\lambda x. (f\ x, g\ x))\ \text{'space } M. (\text{fst } y + \text{snd } y) * \text{emeasure } M\ \{x \in \text{space } M. (f\ x, g\ x) = y\})$
by $(\text{intro simple_function_partition})\ (\text{auto intro: } f\ g)$
also have $\dots = (\sum y \in (\lambda x. (f\ x, g\ x))\ \text{'space } M. \text{fst } y * \text{emeasure } M\ \{x \in \text{space } M. (f\ x, g\ x) = y\}) +$
 $(\sum y \in (\lambda x. (f\ x, g\ x))\ \text{'space } M. \text{snd } y * \text{emeasure } M\ \{x \in \text{space } M. (f\ x, g\ x) = y\})$
using $\text{assms}(2,4)$ **by** $(\text{auto intro!: } \text{sum.cong distrib_right simp: } \text{sum.distrib[symmetric]})$
also have $(\sum y \in (\lambda x. (f\ x, g\ x))\ \text{'space } M. \text{fst } y * \text{emeasure } M\ \{x \in \text{space } M. (f\ x, g\ x) = y\}) = (\int^S x. f\ x\ \partial M)$
by $(\text{intro simple_function_partition[symmetric]})\ (\text{auto intro: } f\ g)$
also have $(\sum y \in (\lambda x. (f\ x, g\ x))\ \text{'space } M. \text{snd } y * \text{emeasure } M\ \{x \in \text{space } M. (f\ x, g\ x) = y\}) = (\int^S x. g\ x\ \partial M)$
by $(\text{intro simple_function_partition[symmetric]})\ (\text{auto intro: } f\ g)$
finally show $?thesis$.
qed

lemma $\text{simple_integral_sum[simp]}$:
assumes $\bigwedge i\ x. i \in P \implies 0 \leq f\ i\ x$
assumes $\bigwedge i. i \in P \implies \text{simple_function } M\ (f\ i)$
shows $(\int^S x. (\sum i \in P. f\ i\ x)\ \partial M) = (\sum i \in P. \text{integral}^S\ M\ (f\ i))$
proof cases
assume $\text{finite } P$
from this assms **show** $?thesis$
by induct (auto)
qed auto

lemma $\text{simple_integral_mult[simp]}$:
assumes $f: \text{simple_function } M\ f$
shows $(\int^S x. c * f\ x\ \partial M) = c * \text{integral}^S\ M\ f$
proof –
have $(\int^S x. c * f\ x\ \partial M) = (\sum y \in f\ \text{'space } M. (c * y) * \text{emeasure } M\ \{x \in \text{space } M. f\ x = y\})$
using f **by** $(\text{intro simple_function_partition})\ \text{auto}$
also have $\dots = c * \text{integral}^S\ M\ f$
using f **unfolding** $\text{simple_integral_def}$
by $(\text{subst sum_distrib_left})\ (\text{auto simp: } \text{mult.assoc Int_def conj_commute})$
finally show $?thesis$.
qed

lemma $\text{simple_integral_mono_AE}$:
assumes $f[\text{measurable}]: \text{simple_function } M\ f$ **and** $g[\text{measurable}]: \text{simple_function } M\ g$
and $\text{mono: } \text{AE } x\ \text{in } M. f\ x \leq g\ x$
shows $\text{integral}^S\ M\ f \leq \text{integral}^S\ M\ g$

proof –
let $\mu = \lambda P. \text{emeasure } M \{x \in \text{space } M. P \ x\}$
have $\text{integral}^S M f = (\sum y \in (\lambda x. (f \ x, g \ x))) \text{'space } M. \text{fst } y * \mu (\lambda x. (f \ x, g \ x) = y)$
using $f \ g$ **by** $(\text{intro simple_function_partition}) \text{ auto}$
also have $\dots \leq (\sum y \in (\lambda x. (f \ x, g \ x))) \text{'space } M. \text{snd } y * \mu (\lambda x. (f \ x, g \ x) = y)$
proof $(\text{clarsimp intro!} : \text{sum_mono})$
fix x **assume** $x \in \text{space } M$
let $?M = \mu (\lambda y. f \ y = f \ x \wedge g \ y = g \ x)$
show $f \ x * ?M \leq g \ x * ?M$
proof cases
assume $?M \neq 0$
then have $0 < ?M$
by $(\text{simp add} : \text{less_le})$
also have $\dots \leq \mu (\lambda y. f \ x \leq g \ x)$
using mono **by** $(\text{intro emeasure_mono_AE}) \text{ auto}$
finally have $\neg \neg f \ x \leq g \ x$
by $(\text{intro notI}) \text{ auto}$
then show $?thesis$
by $(\text{intro mult_right_mono}) \text{ auto}$
qed simp
qed
also have $\dots = \text{integral}^S M g$
using $f \ g$ **by** $(\text{intro simple_function_partition}[\text{symmetric}]) \text{ auto}$
finally show $?thesis .$
qed

lemma $\text{simple_integral_mono}$:
assumes $\text{simple_function } M f$ **and** $\text{simple_function } M g$
and $\text{mono} : \bigwedge x. x \in \text{space } M \implies f \ x \leq g \ x$
shows $\text{integral}^S M f \leq \text{integral}^S M g$
using assms **by** $(\text{intro simple_integral_mono_AE}) \text{ auto}$

lemma $\text{simple_integral_cong_AE}$:
assumes $\text{simple_function } M f$ **and** $\text{simple_function } M g$
and $\text{AE } x \text{ in } M. f \ x = g \ x$
shows $\text{integral}^S M f = \text{integral}^S M g$
using assms **by** $(\text{auto simp} : \text{eq_iff intro!} : \text{simple_integral_mono_AE})$

lemma $\text{simple_integral_cong'}$:
assumes $\text{sf} : \text{simple_function } M f$ $\text{simple_function } M g$
and $\text{mea} : (\text{emeasure } M) \{x \in \text{space } M. f \ x \neq g \ x\} = 0$
shows $\text{integral}^S M f = \text{integral}^S M g$
proof $(\text{intro simple_integral_cong_AE sf AE_I})$
show $(\text{emeasure } M) \{x \in \text{space } M. f \ x \neq g \ x\} = 0$ **by fact**
show $\{x \in \text{space } M. f \ x \neq g \ x\} \in \text{sets } M$
using $\text{sf}[\text{THEN borel_measurable_simple_function}]$ **by auto**
qed simp

lemma *simple_integral_indicator*:

assumes $A: A \in \text{sets } M$

assumes $f: \text{simple_function } M f$

shows $(\int^S x. f x * \text{indicator } A x \partial M) =$

$(\sum x \in f \text{ ' space } M. x * \text{emeasure } M (f \text{ - ' } \{x\} \cap \text{space } M \cap A))$

proof –

have $\text{eq}: (\lambda x. (f x, \text{indicator } A x)) \text{ ' space } M \cap \{x. \text{snd } x = 1\} = (\lambda x. (f x, 1::\text{ennreal})) \text{ ' } A$

using $A[\text{THEN sets.sets_into_space}]$ **by** $(\text{auto simp: indicator_def image_iff split: if_split_asm})$

have $\text{eq2}: \bigwedge x. f x \notin f \text{ ' } A \implies f \text{ - ' } \{f x\} \cap \text{space } M \cap A = \{\}$

by $(\text{auto simp: image_iff})$

have $(\int^S x. f x * \text{indicator } A x \partial M) =$

$(\sum y \in (\lambda x. (f x, \text{indicator } A x)) \text{ ' space } M. (\text{fst } y * \text{snd } y) * \text{emeasure } M \{x \in \text{space } M. (f x, \text{indicator } A x) = y\})$

using assms **by** $(\text{intro simple_function_partition})$ **auto**

also **have** $\dots = (\sum y \in (\lambda x. (f x, \text{indicator } A x::\text{ennreal})) \text{ ' space } M.$

$\text{if } \text{snd } y = 1 \text{ then } \text{fst } y * \text{emeasure } M (f \text{ - ' } \{\text{fst } y\} \cap \text{space } M \cap A) \text{ else } 0)$

by $(\text{auto simp: indicator_def split: if_split_asm intro!: arg_cong2}[\text{where } f=(*)] \text{ arg_cong2}[\text{where } f=\text{emeasure}] \text{ sum.cong})$

also **have** $\dots = (\sum y \in (\lambda x. (f x, 1::\text{ennreal})) \text{ ' } A. \text{fst } y * \text{emeasure } M (f \text{ - ' } \{\text{fst } y\} \cap \text{space } M \cap A))$

using assms **by** $(\text{subst sum.If_cases})$ $(\text{auto intro!: simple_functionD}(1) \text{ simp: eq})$

also **have** $\dots = (\sum y \in \text{fst} \text{ ' } (\lambda x. (f x, 1::\text{ennreal})) \text{ ' } A. y * \text{emeasure } M (f \text{ - ' } \{y\} \cap \text{space } M \cap A))$

by $(\text{subst sum.reindex } [\text{of } \text{fst}])$ $(\text{auto simp: inj_on_def})$

also **have** $\dots = (\sum x \in f \text{ ' space } M. x * \text{emeasure } M (f \text{ - ' } \{x\} \cap \text{space } M \cap A))$

using $A[\text{THEN sets.sets_into_space}]$

by $(\text{intro sum.mono_neutral_cong_left simple_functionD } f) (\text{auto simp: image_comp comp_def eq2})$

finally **show** $?thesis$.

qed

lemma *simple_integral_indicator_only* $[\text{simp}]$:

assumes $A \in \text{sets } M$

shows $\text{integral}^S M (\text{indicator } A) = \text{emeasure } M A$

using *simple_integral_indicator* $[\text{OF } \text{assms}, \text{ of } \lambda x. 1]$ *sets.sets_into_space* $[\text{OF } \text{assms}]$

by $(\text{simp_all add: image_constant_conv Int_absorb1 split: if_split_asm})$

lemma *simple_integral_null_set*:

assumes *simple_function* $M u \bigwedge x. 0 \leq u x$ **and** $N \in \text{null_sets } M$

shows $(\int^S x. u x * \text{indicator } N x \partial M) = 0$

proof –

have $AE x \text{ in } M. \text{indicator } N x = (0 :: \text{ennreal})$

using $\langle N \in \text{null_sets } M \rangle$ **by** $(\text{auto simp: indicator_def intro!: AE_I}[\text{of } _ _ _])$

2238

N)
 then have $(\int^S x. u x * \text{indicator } N x \partial M) = (\int^S x. 0 \partial M)$
 using *assms* **apply** (*intro simple_integral_cong_AE*) **by** *auto*
 then show *?thesis* **by** *simp*
qed

lemma *simple_integral_cong_AE_mult_indicator*:
 assumes *sf*: *simple_function* $M f$ **and** *eq*: *AE* x *in* M . $x \in S$ **and** $S \in \text{sets } M$
 shows $\text{integral}^S M f = (\int^S x. f x * \text{indicator } S x \partial M)$
 using *assms* **by** (*intro simple_integral_cong_AE*) *auto*

lemma *simple_integral_cmult_indicator*:
 assumes $A: A \in \text{sets } M$
 shows $(\int^S x. c * \text{indicator } A x \partial M) = c * \text{emeasure } M A$
 using *simple_integral_mult[OF simple_function_indicator[OF A]]*
 unfolding *simple_integral_indicator_only[OF A]* **by** *simp*

lemma *simple_integral_nonneg*:
 assumes *f*: *simple_function* $M f$ **and** *ae*: *AE* x *in* M . $0 \leq f x$
 shows $0 \leq \text{integral}^S M f$
proof –
 have $\text{integral}^S M (\lambda x. 0) \leq \text{integral}^S M f$
 using *simple_integral_mono_AE[OF _ f ae]* **by** *auto*
 then show *?thesis* **by** *simp*
qed

7.5.4 Integral on nonnegative functions

definition *nn_integral* :: '*a* *measure* \Rightarrow (*a* \Rightarrow *ennreal*) \Rightarrow *ennreal* (*integral*^{*N*})
where

$\text{integral}^N M f = (\text{SUP } g \in \{g. \text{simple_function } M g \wedge g \leq f\}. \text{integral}^S M g)$

syntax

_nn_integral :: *pttrn* \Rightarrow *ennreal* \Rightarrow '*a* *measure* \Rightarrow *ennreal* ($\int^+ ((\lambda _./ _)/ \partial _)$)
[60,61] 110)

translations

$\int^+ x. f \partial M == \text{CONST } \text{nn_integral } M (\lambda x. f)$

lemma *nn_integral_def_finite*:

$\text{integral}^N M f = (\text{SUP } g \in \{g. \text{simple_function } M g \wedge g \leq f \wedge (\forall x. g x < \text{top})\}. \text{integral}^S M g)$

(*is* $_ = \text{Sup } (?A \text{ ' } ?f)$)

unfolding *nn_integral_def*

proof (*safe intro!*: *antisym SUP_least*)

fix g **assume** $g[\text{measurable}]$: *simple_function* $M g g \leq f$

show $\text{integral}^S M g \leq \text{Sup } (?A \text{ ' } ?f)$

proof *cases*

```

assume ae: AE x in M. g x ≠ top
let ?G = {x ∈ space M. g x ≠ top}
have integralS M g = integralS M (λx. g x * indicator ?G x)
proof (rule simple_integral_cong_AE)
  show AE x in M. g x = g x * indicator ?G x
    using ae AE_space by eventually_elim auto
qed (insert g, auto)
also have ... ≤ Sup (?A ' ?f)
  using g by (intro SUP_upper) (auto simp: le_fun_def less_top split: split_indicator)
finally show ?thesis .
next
assume nAE: ¬ (AE x in M. g x ≠ top)
then have emeasure M {x ∈ space M. g x = top} ≠ 0 (is_emeasure M ?G ≠ 0)
  by (subst (asm) AE_iff_measurable[OF _ refl]) auto
then have top = (SUP n. (∫S x. of_nat n * indicator ?G x ∂M))
by (simp add: ennreal_SUP_of_nat_eq_top ennreal_top_eq_mult_iff SUP_mult_right_ennreal[symmetric])
also have ... ≤ Sup (?A ' ?f)
  using g
  by (safe intro!: SUP_least SUP_upper)
  (auto simp: le_fun_def of_nat_less_top top_unique[symmetric] split: split_indicator)
  (intro: order_trans[of _ g x f x for x, OF order_trans[of _ top]])
finally show ?thesis
  by (simp add: top_unique del: SUP_eq_top_iff Sup_eq_top_iff)
qed
qed (auto intro: SUP_upper)

lemma nn_integral_mono_AE:
assumes ae: AE x in M. u x ≤ v x shows integralN M u ≤ integralN M v
unfolding nn_integral_def
proof (safe intro!: SUP_mono)
fix n assume n: simple_function M n n ≤ u
from ae[THEN AE_E] obtain N
  where N: {x ∈ space M. ¬ u x ≤ v x} ⊆ N emeasure M N = 0 N ∈ sets M
  by auto
then have ae_N: AE x in M. x ∉ N by (auto intro: AE_not_in)
let ?n = λx. n x * indicator (space M - N) x
have AE x in M. n x ≤ ?n x simple_function M ?n
  using n N ae_N by auto
moreover
{ fix x have ?n x ≤ v x
  proof cases
    assume x: x ∈ space M - N
    with N have u x ≤ v x by auto
    with n(2)[THEN le_funD, of x] x show ?thesis
      by (auto simp: max_def split: if_split_asm)
    qed simp }
then have ?n ≤ v by (auto simp: le_funI)

```

2240

moreover have $\text{integral}^S M n \leq \text{integral}^S M ?n$
using ae_N N n **by** $(\text{auto intro!}: \text{simple_integral_mono_AE})$
ultimately show $\exists m \in \{g. \text{simple_function } M g \wedge g \leq v\}. \text{integral}^S M n \leq \text{integral}^S M m$
by force
qed

lemma nn_integral_mono :
 $(\bigwedge x. x \in \text{space } M \implies u x \leq v x) \implies \text{integral}^N M u \leq \text{integral}^N M v$
by $(\text{auto intro}: \text{nn_integral_mono_AE})$

lemma mono_nn_integral : $\text{mono } F \implies \text{mono } (\lambda x. \text{integral}^N M (F x))$
by $(\text{auto simp add}: \text{mono_def le_fun_def intro!}: \text{nn_integral_mono})$

lemma $\text{nn_integral_cong_AE}$:
 $\text{AE } x \text{ in } M. u x = v x \implies \text{integral}^N M u = \text{integral}^N M v$
by $(\text{auto simp}: \text{eq_iff intro!}: \text{nn_integral_mono_AE})$

lemma nn_integral_cong :
 $(\bigwedge x. x \in \text{space } M \implies u x = v x) \implies \text{integral}^N M u = \text{integral}^N M v$
by $(\text{auto intro}: \text{nn_integral_cong_AE})$

lemma $\text{nn_integral_cong_simp}$:
 $(\bigwedge x. x \in \text{space } M =\text{simp}\implies u x = v x) \implies \text{integral}^N M u = \text{integral}^N M v$
by $(\text{auto intro}: \text{nn_integral_cong simp}: \text{simp_implies_def})$

lemma $\text{incseq_nn_integral}$:
assumes $\text{incseq } f$ **shows** $\text{incseq } (\lambda i. \text{integral}^N M (f i))$
proof –
have $\bigwedge i x. f i x \leq f (\text{Suc } i) x$
using assms **by** $(\text{auto dest!}: \text{incseq_SucD simp}: \text{le_fun_def})$
then show $?thesis$
by $(\text{auto intro!}: \text{incseq_SucI nn_integral_mono})$
qed

lemma $\text{nn_integral_eq_simple_integral}$:
assumes $f: \text{simple_function } M f$ **shows** $\text{integral}^N M f = \text{integral}^S M f$
proof –
let $?f = \lambda x. f x * \text{indicator } (\text{space } M) x$
have $f': \text{simple_function } M ?f$ **using** f **by** auto
have $\text{integral}^N M ?f \leq \text{integral}^S M ?f$ **using** f'
by $(\text{force intro!}: \text{SUP_least simple_integral_mono simp}: \text{le_fun_def nn_integral_def})$
moreover have $\text{integral}^S M ?f \leq \text{integral}^N M ?f$
unfolding nn_integral_def
using f' **by** $(\text{auto intro!}: \text{SUP_upper})$
ultimately show $?thesis$
by $(\text{simp cong}: \text{nn_integral_cong simple_integral_cong})$
qed

Beppo-Levi monotone convergence theorem

```

lemma nn_integral_monotone_convergence_SUP:
  assumes f: incseq f and [measurable]:  $\bigwedge i. f i \in \text{borel\_measurable } M$ 
  shows  $(\int^+ x. (\text{SUP } i. f i x) \partial M) = (\text{SUP } i. \text{integral}^N M (f i))$ 
proof (rule antisym)
  show  $(\int^+ x. (\text{SUP } i. f i x) \partial M) \leq (\text{SUP } i. (\int^+ x. f i x \partial M))$ 
    unfolding nn_integral_def_finite[of  $\_ \lambda x. \text{SUP } i. f i x$ ]
  proof (safe intro!: SUP_least)
    fix u assume sf_u[simp]: simple_function M u and
      u:  $u \leq (\lambda x. \text{SUP } i. f i x)$  and u_range:  $\forall x. u x < \text{top}$ 
    note sf_u[THEN borel_measurable_simple_function, measurable]
    show  $\text{integral}^S M u \leq (\text{SUP } j. \int^+ x. f j x \partial M)$ 
    proof (rule ennreal_approx_unit)
      fix a :: ennreal assume  $a < 1$ 
      let ?au =  $\lambda x. a * u x$ 

      let ?B =  $\lambda c i. \{x \in \text{space } M. ?au x = c \wedge c \leq f i x\}$ 
      have  $\text{integral}^S M ?au = (\sum c \in ?au \text{'space } M. c * (\text{SUP } i. \text{emeasure } M (?B c i)))$ 
    proof
      unfolding simple_integral_def
    proof (intro sum.cong ennreal_mult_left_cong refl)
      fix c assume  $c \in ?au \text{'space } M$   $c \neq 0$ 
      { fix x' assume  $x' \in \text{space } M$   $?au x' = c$ 
        with  $\langle c \neq 0 \rangle$  u_range have  $?au x' < 1 * u x'$ 
          by (intro ennreal_mult_strict_right_mono  $\langle a < 1 \rangle$ ) (auto simp: less_le)
        also have  $\dots \leq (\text{SUP } i. f i x')$ 
          using u by (auto simp: le_fun_def)
        finally have  $\exists i. ?au x' \leq f i x'$ 
          by (auto simp: less_SUP_iff intro: less_imp_le) }
      then have *:  $?au - \{c\} \cap \text{space } M = (\bigcup i. ?B c i)$ 
        by auto
      show  $\text{emeasure } M (?au - \{c\} \cap \text{space } M) = (\text{SUP } i. \text{emeasure } M (?B c i))$ 
    proof
      unfolding * using f
    by (intro SUP_emeasure_incseq[symmetric])
      (auto simp: incseq_def le_fun_def intro: order_trans)
    qed
    also have  $\dots = (\text{SUP } i. \sum c \in ?au \text{'space } M. c * \text{emeasure } M (?B c i))$ 
    unfolding SUP_mult_left_ennreal using f
    by (intro ennreal_SUP_sum[symmetric])
      (auto intro!: mult_mono_emeasure_mono simp: incseq_def le_fun_def
        intro: order_trans)
    also have  $\dots \leq (\text{SUP } i. \text{integral}^N M (f i))$ 
    proof (intro SUP_subset_mono order_refl)
      fix i
      have  $(\sum c \in ?au \text{'space } M. c * \text{emeasure } M (?B c i)) =$ 
         $(\int^S x. (a * u x) * \text{indicator } \{x \in \text{space } M. a * u x \leq f i x\} x \partial M)$ 
        by (subst simple_integral_indicator)
          (auto intro!: sum.cong ennreal_mult_left_cong arg_cong2[where
            f=emeasure])

```

also have $\dots = (\int^+ x. (a * u x) * \text{indicator } \{x \in \text{space } M. a * u x \leq f i x\} x \partial M)$
by $(\text{rule } nn_integral_eq_simple_integral[\text{symmetric}]) \text{ simp}$
also have $\dots \leq (\int^+ x. f i x \partial M)$
by $(\text{intro } nn_integral_mono) (\text{auto split: split_indicator})$
finally show $(\sum c \in ?au' \text{space } M. c * \text{emeasure } M (?B c i)) \leq (\int^+ x. f i x \partial M)$.
qed
finally show $a * \text{integral}^S M u \leq (\text{SUP } i. \text{integral}^N M (f i))$
by simp
qed
qed
qed $(\text{auto intro!: SUP_least SUP_upper } nn_integral_mono)$

lemma $sup_continuous_nn_integral[\text{order_continuous_intros}]$:
assumes $f: \bigwedge y. sup_continuous (f y)$
assumes $[\text{measurable}]: \bigwedge x. (\lambda y. f y x) \in \text{borel_measurable } M$
shows $sup_continuous (\lambda x. (\int^+ y. f y x \partial M))$
unfolding $sup_continuous_def$
proof safe
fix $C :: \text{nat} \Rightarrow 'b$ **assume** $C: \text{incseq } C$
with $sup_continuous_mono[OF f]$ **show** $(\int^+ y. f y (\text{Sup } (C ' UNIV)) \partial M) = (\text{SUP } i. \int^+ y. f y (C i) \partial M)$
unfolding $sup_continuousD[OF f C]$
by $(\text{subst } nn_integral_monotone_convergence_SUP) (\text{auto simp: mono_def le_fun_def})$
qed

theorem $nn_integral_monotone_convergence_SUP_AE$:
assumes $f: \bigwedge i. AE x \text{ in } M. f i x \leq f (\text{Suc } i) x \wedge i. f i \in \text{borel_measurable } M$
shows $(\int^+ x. (\text{SUP } i. f i x) \partial M) = (\text{SUP } i. \text{integral}^N M (f i))$
proof $-$
from f **have** $AE x \text{ in } M. \forall i. f i x \leq f (\text{Suc } i) x$
by $(\text{simp add: AE_all_countable})$
from $\text{this}[\text{THEN } AE_E]$ **obtain** N
where $N: \{x \in \text{space } M. \neg (\forall i. f i x \leq f (\text{Suc } i) x)\} \subseteq N \text{ emeasure } M N = 0$
 $N \in \text{sets } M$
by auto
let $?f = \lambda i x. \text{if } x \in \text{space } M - N \text{ then } f i x \text{ else } 0$
have $f_eq: AE x \text{ in } M. \forall i. ?f i x = f i x$ **using** N **by** $(\text{auto intro!: AE_I[of } _ N])$
then have $(\int^+ x. (\text{SUP } i. f i x) \partial M) = (\int^+ x. (\text{SUP } i. ?f i x) \partial M)$
by $(\text{auto intro!: } nn_integral_cong_AE)$
also have $\dots = (\text{SUP } i. (\int^+ x. ?f i x \partial M))$
proof $(\text{rule } nn_integral_monotone_convergence_SUP)$
show $\text{incseq } ?f$ **using** $N(1)$ **by** $(\text{force intro!: incseq_SucI le_funI})$
{ fix } i **show** $(\lambda x. \text{if } x \in \text{space } M - N \text{ then } f i x \text{ else } 0) \in \text{borel_measurable } M$
using $f N(3)$ **by** $(\text{intro measurable_If_set}) \text{ auto }$
qed

also have ... = (SUP i. ($\int^+ x. f i x \partial M$))
 using f_eq by (force intro!: arg_cong[where f = $\lambda f. \text{Sup}(\text{range } f)$] nn_integral_cong_AE ext)
 finally show ?thesis .
 qed

lemma nn_integral_monotone_convergence_simple:
 $\text{incseq } f \implies (\bigwedge i. \text{simple_function } M (f i)) \implies (\text{SUP } i. \int^S x. f i x \partial M) = (\int^+ x. (\text{SUP } i. f i x) \partial M)$
 using nn_integral_monotone_convergence_SUP[of f M]
 by (simp add: nn_integral_eq_simple_integral[symmetric] borel_measurable_simple_function)

lemma SUP_simple_integral_sequences:
 assumes f: incseq f $\bigwedge i. \text{simple_function } M (f i)$
 and g: incseq g $\bigwedge i. \text{simple_function } M (g i)$
 and eq: AE x in M. (SUP i. f i x) = (SUP i. g i x)
 shows (SUP i. $\text{integral}^S M (f i)$) = (SUP i. $\text{integral}^S M (g i)$)
 (is $\text{Sup} (?F \text{ ' } _) = \text{Sup} (?G \text{ ' } _)$)

proof –
 have (SUP i. $\text{integral}^S M (f i)$) = ($\int^+ x. (\text{SUP } i. f i x) \partial M$)
 using f by (rule nn_integral_monotone_convergence_simple)
 also have ... = ($\int^+ x. (\text{SUP } i. g i x) \partial M$)
 unfolding eq[THEN nn_integral_cong_AE] ..
 also have ... = (SUP i. $?G i$)
 using g by (rule nn_integral_monotone_convergence_simple[symmetric])
 finally show ?thesis by simp
 qed

lemma nn_integral_const[simp]: ($\int^+ x. c \partial M$) = c * emeasure M (space M)
 by (subst nn_integral_eq_simple_integral) auto

lemma nn_integral_linear:
 assumes f: $f \in \text{borel_measurable } M$ and g: $g \in \text{borel_measurable } M$
 shows ($\int^+ x. a * f x + g x \partial M$) = a * $\text{integral}^N M f$ + $\text{integral}^N M g$
 (is $\text{integral}^N M ?L = _$)

proof –
 obtain u
 where $\bigwedge i. \text{simple_function } M (u i)$ incseq u $\bigwedge i x. u i x < \text{top} \bigwedge x. (\text{SUP } i. u i x) = f x$
 using borel_measurable_implies_simple_function_sequence' f(1)
 by auto
 note u = nn_integral_monotone_convergence_simple[OF this(2,1)] this

obtain v where
 $\bigwedge i. \text{simple_function } M (v i)$ incseq v $\bigwedge i x. v i x < \text{top} \bigwedge x. (\text{SUP } i. v i x) = g x$
 using borel_measurable_implies_simple_function_sequence' g(1)
 by auto
 note v = nn_integral_monotone_convergence_simple[OF this(2,1)] this

```

let ?L' =  $\lambda i x. a * u i x + v i x$ 

have ?L  $\in$  borel_measurable M using assms by auto
from borel_measurable_implies_simple_function_sequence'[OF this]
obtain l where  $\bigwedge i. \text{simple\_function } M (l i) \text{ incseq } l \bigwedge i x. l i x < \text{top} \bigwedge x. (\text{SUP } i. l i x) = a * f x + g x$ 
by auto
note l = nn_integral_monotone_convergence_simple[OF this(2,1)] this

have inc: incseq ( $\lambda i. a * \text{integral}^S M (u i)$ ) incseq ( $\lambda i. \text{integral}^S M (v i)$ )
using u v by (auto simp: incseq_Suc_iff le_fun_def intro!: add_mono mult_left_mono simple_integral_mono)

have l': ( $\text{SUP } i. \text{integral}^S M (l i)$ ) = ( $\text{SUP } i. \text{integral}^S M (?L' i)$ )
proof (rule SUP_simple_integral_sequences[OF l(3,2)])
show incseq ?L'  $\bigwedge i. \text{simple\_function } M (?L' i)$ 
using u v unfolding incseq_Suc_iff le_fun_def
by (auto intro!: add_mono mult_left_mono)
{ fix x
have ( $\text{SUP } i. a * u i x + v i x$ ) = a * ( $\text{SUP } i. u i x$ ) + ( $\text{SUP } i. v i x$ )
using u(3) v(3) u(4)[of _ x] v(4)[of _ x] unfolding SUP_mult_left_enreal
by (auto intro!: enreal_SUP_add simp: incseq_Suc_iff le_fun_def add_mono mult_left_mono) }
then show AE x in M. ( $\text{SUP } i. l i x$ ) = ( $\text{SUP } i. ?L' i x$ )
unfolding l(5) using u(5) v(5) by (intro AE_I2) auto
qed
also have ... = ( $\text{SUP } i. a * \text{integral}^S M (u i) + \text{integral}^S M (v i)$ )
using u(2) v(2) by auto
finally show ?thesis
unfolding l(5)[symmetric] l(1)[symmetric]
by (simp add: enreal_SUP_add[OF inc] v u SUP_mult_left_enreal[symmetric])
qed

lemma nn_integral_cmult:  $f \in \text{borel\_measurable } M \implies (\int^+ x. c * f x \partial M) = c * \text{integral}^N M f$ 
using nn_integral_linear[of f M  $\lambda x. 0 c$ ] by simp

lemma nn_integral_multc:  $f \in \text{borel\_measurable } M \implies (\int^+ x. f x * c \partial M) = \text{integral}^N M f * c$ 
unfolding mult.commute[of _ c] nn_integral_cmult by simp

lemma nn_integral_divide:  $f \in \text{borel\_measurable } M \implies (\int^+ x. f x / c \partial M) = (\int^+ x. f x \partial M) / c$ 
unfolding divide_enreal_def by (rule nn_integral_multc)

lemma nn_integral_indicator[simp]:  $A \in \text{sets } M \implies (\int^+ x. \text{indicator } A x \partial M) = (\text{emeasure } M) A$ 
by (subst nn_integral_eq_simple_integral) (auto simp: simple_integral_indicator)

```


lemma *nn_integral_cmult_indicator*: $A \in \text{sets } M \implies (\int^+ x. c * \text{indicator } A x \partial M) = c * \text{emeasure } M A$
by (*subst nn_integral_eq_simple_integral*) (*auto*)

lemma *nn_integral_indicator'*:
assumes [*measurable*]: $A \cap \text{space } M \in \text{sets } M$
shows $(\int^+ x. \text{indicator } A x \partial M) = \text{emeasure } M (A \cap \text{space } M)$
proof –
have $(\int^+ x. \text{indicator } A x \partial M) = (\int^+ x. \text{indicator } (A \cap \text{space } M) x \partial M)$
by (*intro nn_integral_cong*) (*simp split: split_indicator*)
also have $\dots = \text{emeasure } M (A \cap \text{space } M)$
by *simp*
finally show *?thesis* .
qed

lemma *nn_integral_indicator_singleton[simp]*:
assumes [*measurable*]: $\{y\} \in \text{sets } M$ **shows** $(\int^+ x. f x * \text{indicator } \{y\} x \partial M) = f y * \text{emeasure } M \{y\}$
proof –
have $(\int^+ x. f x * \text{indicator } \{y\} x \partial M) = (\int^+ x. f y * \text{indicator } \{y\} x \partial M)$
by (*auto intro!: nn_integral_cong split: split_indicator*)
then show *?thesis*
by (*simp add: nn_integral_cmult*)
qed

lemma *nn_integral_set_ennreal*:
 $(\int^+ x. \text{ennreal } (f x) * \text{indicator } A x \partial M) = (\int^+ x. \text{ennreal } (f x * \text{indicator } A x) \partial M)$
by (*rule nn_integral_cong*) (*simp split: split_indicator*)

lemma *nn_integral_indicator_singleton'[simp]*:
assumes [*measurable*]: $\{y\} \in \text{sets } M$
shows $(\int^+ x. \text{ennreal } (f x * \text{indicator } \{y\} x) \partial M) = f y * \text{emeasure } M \{y\}$
by (*subst nn_integral_set_ennreal[symmetric]*) (*simp*)

lemma *nn_integral_add*:
 $f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies (\int^+ x. f x + g x \partial M) = \text{integral}^N M f + \text{integral}^N M g$
using *nn_integral_linear[of f M g 1]* **by** *simp*

lemma *nn_integral_sum*:
 $(\bigwedge i. i \in P \implies f i \in \text{borel_measurable } M) \implies (\int^+ x. (\sum i \in P. f i x) \partial M) = (\sum i \in P. \text{integral}^N M (f i))$
by (*induction P rule: infinite_finite_induct*) (*auto simp: nn_integral_add*)

theorem *nn_integral_suminf*:
assumes $f: \bigwedge i. f i \in \text{borel_measurable } M$
shows $(\int^+ x. (\sum i. f i x) \partial M) = (\sum i. \text{integral}^N M (f i))$

proof –

have $all_pos: AE\ x\ in\ M. \forall i. 0 \leq f\ i\ x$
using $assms$ **by** $(auto\ simp: AE_all_countable)$
have $(\sum i. integral^N\ M\ (f\ i)) = (SUP\ n. \sum i < n. integral^N\ M\ (f\ i))$
by $(rule\ suminf_eq_SUP)$
also have $\dots = (SUP\ n. \int^{+x}. (\sum i < n. f\ i\ x)\ \partial M)$
unfolding $nn_integral_sum[OF\ f]$..
also have $\dots = \int^{+x}. (SUP\ n. \sum i < n. f\ i\ x)\ \partial M$ **using** $f\ all_pos$
by $(intro\ nn_integral_monotone_convergence_SUP_AE[symmetric])$
 $(elim\ AE_mp, auto\ simp: sum_nonneg\ simp\ del: sum.lessThan_Suc\ intro!:$
 $AE_I2\ sum_mono2)$
also have $\dots = \int^{+x}. (\sum i. f\ i\ x)\ \partial M$ **using** all_pos
by $(intro\ nn_integral_cong_AE)\ (auto\ simp: suminf_eq_SUP)$
finally show $?thesis$ **by** $simp$
qed

lemma $nn_integral_bound_simple_function:$

assumes $bnd: \bigwedge x. x \in space\ M \implies f\ x < \infty$
assumes $f[measurable]: simple_function\ M\ f$
assumes $supp: emeasure\ M\ \{x \in space\ M. f\ x \neq 0\} < \infty$
shows $nn_integral\ M\ f < \infty$

proof *cases*

assume $space\ M = \{\}$
then have $nn_integral\ M\ f = (\int^{+x}. 0\ \partial M)$
by $(intro\ nn_integral_cong)\ auto$
then show $?thesis$ **by** $simp$

next

assume $space\ M \neq \{\}$
with $simple_functionD(1)[OF\ f]\ bnd$ **have** $bnd: 0 \leq Max\ (f'space\ M) \wedge Max\ (f'space\ M) < \infty$
by $(subst\ Max_less_iff)\ (auto\ simp: Max_ge_iff)$

have $nn_integral\ M\ f \leq (\int^{+x}. Max\ (f'space\ M) * indicator\ \{x \in space\ M. f\ x \neq 0\}\ x\ \partial M)$

proof $(rule\ nn_integral_mono)$

fix x **assume** $x \in space\ M$

with f **show** $f\ x \leq Max\ (f'space\ M) * indicator\ \{x \in space\ M. f\ x \neq 0\}\ x$

by $(auto\ split: split_indicator\ intro!: Max_ge\ simple_functionD)$

qed

also have $\dots < \infty$

using $bnd\ supp$ **by** $(subst\ nn_integral_cmult)\ (auto\ simp: ennreal_mult_less_top)$

finally show $?thesis$.

qed

theorem $nn_integral_Markov_inequality:$

assumes $u: (\lambda x. u\ x * indicator\ A\ x) \in borel_measurable\ M$ **and** $A \in sets\ M$

shows $(emeasure\ M)\ (\{x \in A. 1 \leq c * u\ x\}) \leq c * (\int^{+x}. u\ x * indicator\ A\ x\ \partial M)$

(is $(emeasure\ M)\ ?A \leq _ * ?PI)$

proof –

define u' **where** $u' = (\lambda x. u\ x * \text{indicator}\ A\ x)$
have $[measurable]: u' \in \text{borel_measurable}\ M$
using u **unfolding** u'_def .
have $\{x \in \text{space}\ M. c * u'\ x \geq 1\} \in \text{sets}\ M$
by $measurable$
also have $\{x \in \text{space}\ M. c * u'\ x \geq 1\} = ?A$
using $\text{sets.sets_into_space}[OF\ \langle A \in \text{sets}\ M \rangle]$ **by** $(\text{auto simp: } u'_\text{def}\ \text{indicator_def})$
finally have $(\text{emeasure}\ M)\ ?A = (\int^+ x. \text{indicator}\ ?A\ x\ \partial M)$
using $nn_integral_indicator$ **by** $simp$
also have $\dots \leq (\int^+ x. c * (u\ x * \text{indicator}\ A\ x)\ \partial M)$
using u **by** $(\text{auto intro!: } nn_integral_mono_AE\ \text{simp: indicator_def})$
also have $\dots = c * (\int^+ x. u\ x * \text{indicator}\ A\ x\ \partial M)$
using $assms$ **by** $(\text{auto intro!: } nn_integral_c\text{mult})$
finally show $?thesis$.
qed

lemma $\text{Chernoff_ineq_nn_integral_ge}$:

assumes $s: s > 0$ **and** $[measurable]: A \in \text{sets}\ M$
assumes $[measurable]: (\lambda x. f\ x * \text{indicator}\ A\ x) \in \text{borel_measurable}\ M$
shows $\text{emeasure}\ M\ \{x \in A. f\ x \geq a\} \leq$
 $\text{ennreal}\ (\exp\ (-s * a)) * nn_integral\ M\ (\lambda x. \text{ennreal}\ (\exp\ (s * f\ x)) * \text{indicator}\ A\ x)$

proof –

define f' **where** $f' = (\lambda x. f\ x * \text{indicator}\ A\ x)$
have $[measurable]: f' \in \text{borel_measurable}\ M$
using $assms(3)$ **unfolding** f'_def **by** $assumption$
have $(\lambda x. \text{ennreal}\ (\exp\ (s * f'\ x)) * \text{indicator}\ A\ x) \in \text{borel_measurable}\ M$
by $simp$
also have $(\lambda x. \text{ennreal}\ (\exp\ (s * f'\ x)) * \text{indicator}\ A\ x) =$
 $(\lambda x. \text{ennreal}\ (\exp\ (s * f\ x)) * \text{indicator}\ A\ x)$
by $(\text{auto simp: } f'_\text{def}\ \text{indicator_def}\ \text{fun_eq_iff})$
finally have $\text{meas}: \dots \in \text{borel_measurable}\ M$.

have $\{x \in A. f\ x \geq a\} = \{x \in A. \text{ennreal}\ (\exp\ (-s * a)) * \text{ennreal}\ (\exp\ (s * f\ x)) \geq 1\}$

using s **by** $(\text{auto simp: exp_minus_field_simps}\ \text{simp flip: ennreal_mult})$
also have $\text{emeasure}\ M\ \dots \leq \text{ennreal}\ (\exp\ (-s * a)) *$
 $(\int^+ x. \text{ennreal}\ (\exp\ (s * f\ x)) * \text{indicator}\ A\ x\ \partial M)$
by $(\text{intro order.trans}[OF\ nn_integral_Markov_inequality]\ \text{meas})\ \text{auto}$
finally show $?thesis$.
qed

lemma $\text{Chernoff_ineq_nn_integral_le}$:

assumes $s: s > 0$ **and** $[measurable]: A \in \text{sets}\ M$
assumes $[measurable]: f \in \text{borel_measurable}\ M$
shows $\text{emeasure}\ M\ \{x \in A. f\ x \leq a\} \leq$
 $\text{ennreal}\ (\exp\ (s * a)) * nn_integral\ M\ (\lambda x. \text{ennreal}\ (\exp\ (-s * f\ x)) * \text{indicator}\ A\ x)$

indicator A x)

using *Chernoff_ineq_nn_integral_ge*[of *s A M λx. -f x -a*] *assms by simp*

lemma *nn_integral_noteq_infinite*:

assumes *g*: $g \in \text{borel_measurable } M$ **and** $\text{integral}^N M g \neq \infty$

shows *AE x in M. g x ≠ ∞*

proof (*rule ccontr*)

assume *c*: $\neg (\text{AE } x \text{ in } M. g x \neq \infty)$

have $(\text{emeasure } M) \{x \in \text{space } M. g x = \infty\} \neq 0$

using *c g by (auto simp add: AE_iff_null)*

then have $0 < (\text{emeasure } M) \{x \in \text{space } M. g x = \infty\}$

by (*auto simp: zero_less_iff_neq_zero*)

then have $\infty = \infty * (\text{emeasure } M) \{x \in \text{space } M. g x = \infty\}$

by (*auto simp: ennreal_top_eq_mult_iff*)

also have $\dots \leq (\int^+ x. \infty * \text{indicator } \{x \in \text{space } M. g x = \infty\} x \partial M)$

using *g by (subst nn_integral_cmult_indicator) auto*

also have $\dots \leq \text{integral}^N M g$

using *assms by (auto intro!: nn_integral_mono_AE simp: indicator_def)*

finally show *False*

using $\langle \text{integral}^N M g \neq \infty \rangle$ **by** (*auto simp: top_unique*)

qed

lemma *nn_integral_PInf*:

assumes *f*: $f \in \text{borel_measurable } M$ **and** *not_Inf*: $\text{integral}^N M f \neq \infty$

shows $\text{emeasure } M (f - \{ \infty \} \cap \text{space } M) = 0$

proof -

have $\infty * \text{emeasure } M (f - \{ \infty \} \cap \text{space } M) = (\int^+ x. \infty * \text{indicator } (f - \{ \infty \} \cap \text{space } M) x \partial M)$

using *f by (subst nn_integral_cmult_indicator) (auto simp: measurable_sets)*

also have $\dots \leq \text{integral}^N M f$

by (*auto intro!: nn_integral_mono simp: indicator_def*)

finally have $\infty * (\text{emeasure } M) (f - \{ \infty \} \cap \text{space } M) \leq \text{integral}^N M f$

by *simp*

then show *?thesis*

using *assms by (auto simp: ennreal_top_mult top_unique split: if_split_asm)*

qed

lemma *simple_integral_PInf*:

simple_function M f $\implies \text{integral}^S M f \neq \infty \implies \text{emeasure } M (f - \{ \infty \} \cap \text{space } M) = 0$

by (*rule nn_integral_PInf*) (*auto simp: nn_integral_eq_simple_integral borel_measurable_simple_fun*)

lemma *nn_integral_PInf_AE*:

assumes *f* $\in \text{borel_measurable } M$ $\text{integral}^N M f \neq \infty$ **shows** *AE x in M. f x ≠ ∞*

proof (*rule AE_I*)

show $(\text{emeasure } M) (f - \{ \infty \} \cap \text{space } M) = 0$

by (*rule nn_integral_PInf[OF assms]*)

show $f - \{ \infty \} \cap \text{space } M \in \text{sets } M$

using *assms* **by** (*auto intro: borel_measurable_vimage*)
qed *auto*

lemma *nn_integral_diff*:

assumes *f*: $f \in \text{borel_measurable } M$
and *g*: $g \in \text{borel_measurable } M$
and *fin*: $\text{integral}^N M g \neq \infty$
and *mono*: $AE\ x\ \text{in } M. g\ x \leq f\ x$
shows $(\int^+ x. f\ x - g\ x\ \partial M) = \text{integral}^N M f - \text{integral}^N M g$
proof –
have *diff*: $(\lambda x. f\ x - g\ x) \in \text{borel_measurable } M$
using *assms* **by** *auto*
have $AE\ x\ \text{in } M. f\ x = f\ x - g\ x + g\ x$
using *diff_add_cancel_ennreal mono nn_integral_noteq_infinite[OF g fin]*
assms **by** *auto*
then **have** ****: $\text{integral}^N M f = (\int^+ x. f\ x - g\ x\ \partial M) + \text{integral}^N M g$
unfolding *nn_integral_add[OF diff g, symmetric]*
by (*rule nn_integral_cong_AE*)
show *?thesis* **unfolding** ****
using *fin*
by (*cases rule: ennreal2_cases[of \int^+ x. f x - g x \partial M \text{integral}^N M g]*) *auto*
qed

lemma *nn_integral_mult_bounded_inf*:

assumes *f*: $f \in \text{borel_measurable } M$ $(\int^+ x. f\ x\ \partial M) < \infty$ **and** *c*: $c \neq \infty$ **and**
ae: $AE\ x\ \text{in } M. g\ x \leq c * f\ x$
shows $(\int^+ x. g\ x\ \partial M) < \infty$
proof –
have $(\int^+ x. g\ x\ \partial M) \leq (\int^+ x. c * f\ x\ \partial M)$
by (*intro nn_integral_mono_AE ae*)
also **have** $(\int^+ x. c * f\ x\ \partial M) < \infty$
using *c f* **by** (*subst nn_integral_cmult*) (*auto simp: ennreal_mult_less_top top_unique not_less*)
finally **show** *?thesis* .
qed

Fatou's lemma: convergence theorem on limes inferior

lemma *nn_integral_monotone_convergence_INF_AE'*:

assumes *f*: $\bigwedge i. AE\ x\ \text{in } M. f\ (\text{Suc } i)\ x \leq f\ i\ x$ **and** [*measurable*]: $\bigwedge i. f\ i \in \text{borel_measurable } M$
and ***: $(\int^+ x. f\ 0\ x\ \partial M) < \infty$
shows $(\int^+ x. (\text{INF } i. f\ i\ x)\ \partial M) = (\text{INF } i. \text{integral}^N M (f\ i))$
proof (*rule ennreal_minus_cancel*)
have $\text{integral}^N M (f\ 0) - (\int^+ x. (\text{INF } i. f\ i\ x)\ \partial M) = (\int^+ x. f\ 0\ x - (\text{INF } i. f\ i\ x)\ \partial M)$
proof (*rule nn_integral_diff[symmetric]*)
have $(\int^+ x. (\text{INF } i. f\ i\ x)\ \partial M) \leq (\int^+ x. f\ 0\ x\ \partial M)$
by (*intro nn_integral_mono_INF_lower*) *simp*
with *** **show** $(\int^+ x. (\text{INF } i. f\ i\ x)\ \partial M) \neq \infty$

```

    by simp
  qed (auto intro: INF_lower)
  also have ... = (∫+ x. (SUP i. f 0 x - f i x) ∂M)
    by (simp add: ennreal_INF_const_minus)
  also have ... = (SUP i. (∫+ x. f 0 x - f i x ∂M))
  proof (intro nn_integral_monotone_convergence_SUP_AE)
    show AE x in M. f 0 x - f i x ≤ f 0 x - f (Suc i) x for i
      using f[of i] by eventually_elim (auto simp: ennreal_mono_minus)
  qed simp
  also have ... = (SUP i. nn_integral M (f 0) - (∫+ x. f i x ∂M))
  proof (subst nn_integral_diff[symmetric])
    fix i
    have dec: AE x in M. ∀ i. f (Suc i) x ≤ f i x
      unfolding AE_all_countable using f by auto
    then show AE x in M. f i x ≤ f 0 x
      using dec by eventually_elim (auto intro: lift_Suc_antimono_le[of λi. f i x
0 i for x])
    then have (∫+ x. f i x ∂M) ≤ (∫+ x. f 0 x ∂M)
      by (rule nn_integral_mono_AE)
    with * show (∫+ x. f i x ∂M) ≠ ∞
      by simp
  qed (insert f, auto simp: decseq_def le_fun_def)
  finally show integralN M (f 0) - (∫+ x. (INF i. f i x) ∂M) =
    integralN M (f 0) - (INF i. ∫+ x. f i x ∂M)
    by (simp add: ennreal_INF_const_minus)
  qed (insert *, auto intro!: nn_integral_mono intro: INF_lower)

theorem nn_integral_monotone_convergence_INF_AE:
  fixes f :: nat ⇒ 'a ⇒ ennreal
  assumes f: ∧i. AE x in M. f (Suc i) x ≤ f i x
    and [measurable]: ∧i. f i ∈ borel_measurable M
    and fin: (∫+ x. f i x ∂M) < ∞
  shows (∫+ x. (INF i. f i x) ∂M) = (INF i. integralN M (f i))
  proof -
    { fix f :: nat ⇒ ennreal and j assume decseq f
      then have (INF i. f i) = (INF i. f (i + j))
        apply (intro INF_eq)
        apply (rule_tac x=i in bexI)
        apply (auto simp: decseq_def le_fun_def)
        done }
    note INF_shift = this
  have mono: AE x in M. ∀ i. f (Suc i) x ≤ f i x
    using f by (auto simp: AE_all_countable)
  then have AE x in M. (INF i. f i x) = (INF n. f (n + i) x)
    by eventually_elim (auto intro!: decseq_SucI INF_shift)
  then have (∫+ x. (INF i. f i x) ∂M) = (∫+ x. (INF n. f (n + i) x) ∂M)
    by (rule nn_integral_cong_AE)
  also have ... = (INF n. (∫+ x. f (n + i) x ∂M))
    by (rule nn_integral_monotone_convergence_INF_AE) (insert assms, auto)

```

also have ... = (INF n. ($\int^+ x. f n x \partial M$))
 by (intro INF_shift[symmetric] decseq_SucI nn_integral_mono_AE f)
 finally show ?thesis .
 qed

lemma nn_integral_monotone_convergence_INF_decseq:
 assumes f: decseq f and *: $\bigwedge i. f i \in \text{borel_measurable } M$ ($\int^+ x. f i x \partial M$) <
 ∞
 shows ($\int^+ x. (\text{INF } i. f i x) \partial M$) = (INF i. integral^N M (f i))
 using nn_integral_monotone_convergence_INF_AE[of f M i, OF _ *] f by
 (simp add: decseq_SucD le_funD)

theorem nn_integral_liminf:
 fixes u :: nat \Rightarrow 'a \Rightarrow ennreal
 assumes u: $\bigwedge i. u i \in \text{borel_measurable } M$
 shows ($\int^+ x. \text{liminf } (\lambda n. u n x) \partial M$) \leq liminf ($\lambda n. \text{integral}^N M (u n)$)
 proof -
 have ($\int^+ x. \text{liminf } (\lambda n. u n x) \partial M$) = (SUP n. $\int^+ x. (\text{INF } i \in \{n..\}. u i x) \partial M$)
 unfolding liminf_SUP_INF using u
 by (intro nn_integral_monotone_convergence_SUP_AE)
 (auto intro!: AE_I2 intro: INF_greatest INF_superset_mono)
 also have ... \leq liminf ($\lambda n. \text{integral}^N M (u n)$)
 by (auto simp: liminf_SUP_INF intro!: SUP_mono INF_greatest nn_integral_mono
 INF_lower)
 finally show ?thesis .
 qed

theorem nn_integral_limsup:
 fixes u :: nat \Rightarrow 'a \Rightarrow ennreal
 assumes [measurable]: $\bigwedge i. u i \in \text{borel_measurable } M$ w $\in \text{borel_measurable } M$
 assumes bounds: $\bigwedge i. \text{AE } x \text{ in } M. u i x \leq w x$ and w: ($\int^+ x. w x \partial M$) < ∞
 shows limsup ($\lambda n. \text{integral}^N M (u n)$) \leq ($\int^+ x. \text{limsup } (\lambda n. u n x) \partial M$)
 proof -
 have bnd: AE x in M. $\forall i. u i x \leq w x$
 using bounds by (auto simp: AE_all_countable)
 then have ($\int^+ x. (\text{SUP } n. u n x) \partial M$) \leq ($\int^+ x. w x \partial M$)
 by (auto intro!: nn_integral_mono_AE elim: eventually_mono intro: SUP_least)
 then have ($\int^+ x. \text{limsup } (\lambda n. u n x) \partial M$) = (INF n. $\int^+ x. (\text{SUP } i \in \{n..\}. u i$
 $x) \partial M$)
 unfolding limsup_INF_SUP using bnd w
 by (intro nn_integral_monotone_convergence_INF_AE')
 (auto intro!: AE_I2 intro: SUP_least SUP_subset_mono)
 also have ... \geq limsup ($\lambda n. \text{integral}^N M (u n)$)
 by (auto simp: limsup_INF_SUP intro!: INF_mono SUP_least exI nn_integral_mono
 SUP_upper)
 finally (xtrans) show ?thesis .
 qed

lemma nn_integral_LIMSEQ:

2252

assumes $f: \text{incseq } f \wedge i. f i \in \text{borel_measurable } M$
and $u: \bigwedge x. (\lambda i. f i x) \longrightarrow u x$
shows $(\lambda n. \text{integral}^N M (f n)) \longrightarrow \text{integral}^N M u$
proof –
have $(\lambda n. \text{integral}^N M (f n)) \longrightarrow (\text{SUP } n. \text{integral}^N M (f n))$
using f **by** $(\text{intro LIMSEQ_SUP}[\text{of } \lambda n. \text{integral}^N M (f n)] \text{ incseq_nn_integral})$
also have $(\text{SUP } n. \text{integral}^N M (f n)) = \text{integral}^N M (\lambda x. \text{SUP } n. f n x)$
using f **by** $(\text{intro nn_integral_monotone_convergence_SUP}[\text{symmetric}])$
also have $\text{integral}^N M (\lambda x. \text{SUP } n. f n x) = \text{integral}^N M (\lambda x. u x)$
using f **by** $(\text{subst LIMSEQ_SUP}[\text{THEN LIMSEQ_unique, OF } _ u])$ $(\text{auto simp: incseq_def le_fun_def})$
finally show $?thesis$.
qed

theorem $\text{nn_integral_dominated_convergence}$:

assumes $[\text{measurable}]$:
 $\bigwedge i. u i \in \text{borel_measurable } M \ u' \in \text{borel_measurable } M \ w \in \text{borel_measurable } M$
and $\text{bound: } \bigwedge j. \text{AE } x \text{ in } M. u j x \leq w x$
and $w: (\int^+ x. w x \partial M) < \infty$
and $u': \text{AE } x \text{ in } M. (\lambda i. u i x) \longrightarrow u' x$
shows $(\lambda i. (\int^+ x. u i x \partial M)) \longrightarrow (\int^+ x. u' x \partial M)$
proof –
have $\text{limsup } (\lambda n. \text{integral}^N M (u n)) \leq (\int^+ x. \text{limsup } (\lambda n. u n x) \partial M)$
by $(\text{intro nn_integral_limsup}[\text{OF } _ _ \text{bound } w]) \text{ auto}$
moreover have $(\int^+ x. \text{limsup } (\lambda n. u n x) \partial M) = (\int^+ x. u' x \partial M)$
using u' **by** $(\text{intro nn_integral_cong_AE, eventually_elim})$ $(\text{metis tendsto_iff_Liminf_eq_Limsup sequentially_bot})$
moreover have $(\int^+ x. \text{liminf } (\lambda n. u n x) \partial M) = (\int^+ x. u' x \partial M)$
using u' **by** $(\text{intro nn_integral_cong_AE, eventually_elim})$ $(\text{metis tendsto_iff_Liminf_eq_Limsup sequentially_bot})$
moreover have $(\int^+ x. \text{liminf } (\lambda n. u n x) \partial M) \leq \text{liminf } (\lambda n. \text{integral}^N M (u n))$
by $(\text{intro nn_integral_liminf}) \text{ auto}$
moreover have $\text{liminf } (\lambda n. \text{integral}^N M (u n)) \leq \text{limsup } (\lambda n. \text{integral}^N M (u n))$
by $(\text{intro Liminf_le_Limsup sequentially_bot})$
ultimately show $?thesis$
by $(\text{intro Liminf_eq_Limsup}) \text{ auto}$
qed

lemma $\text{inf_continuous_nn_integral}[\text{order_continuous_intros}]$:

assumes $f: \bigwedge y. \text{inf_continuous } (f y)$
assumes $[\text{measurable}]$: $\bigwedge x. (\lambda y. f y x) \in \text{borel_measurable } M$
assumes $\text{bnd: } \bigwedge x. (\int^+ y. f y x \partial M) \neq \infty$
shows $\text{inf_continuous } (\lambda x. (\int^+ y. f y x \partial M))$
unfolding $\text{inf_continuous_def}$
proof safe
fix $C :: \text{nat} \Rightarrow 'b$ **assume** $C: \text{decseq } C$

then show $(\int^+ y. f y (Inf (C ' UNIV)) \partial M) = (INF i. \int^+ y. f y (C i) \partial M)$
using *inf_continuous_mono*[OF f] *bnd*
by (*auto simp add: inf_continuousD*[OF f C] *fun_eq_iff monotone_def le_fun_def*
less_top
intro!: nn_integral_monotone_convergence_INF_decseq)
qed

lemma *nn_integral_null_set*:

assumes $N \in null_sets M$ **shows** $(\int^+ x. u x * indicator N x \partial M) = 0$
proof –
have $(\int^+ x. u x * indicator N x \partial M) = (\int^+ x. 0 \partial M)$
proof (*intro nn_integral_cong_AE AE_I*)
show $\{x \in space M. u x * indicator N x \neq 0\} \subseteq N$
by (*auto simp: indicator_def*)
show $(emeasure M) N = 0 \Rightarrow N \in sets M$
using *assms by auto*
qed
then show *?thesis by simp*
qed

lemma *nn_integral_0_iff*:

assumes $u [measurable]: u \in borel_measurable M$
shows $integral^N M u = 0 \iff emeasure M \{x \in space M. u x \neq 0\} = 0$
(is $_ \iff (emeasure M) ?A = 0$)
proof –
have *u_eq*: $(\int^+ x. u x * indicator ?A x \partial M) = integral^N M u$
by (*auto intro!: nn_integral_cong simp: indicator_def*)
show *?thesis*
proof
assume $(emeasure M) ?A = 0$
with *nn_integral_null_set*[of ?A M u] *u*
show $integral^N M u = 0$ **by** (*simp add: u_eq null_sets_def*)
next
assume $?: integral^N M u = 0$
let $?M = \lambda n. \{x \in space M. 1 \leq real (n::nat) * u x\}$
have $0 = (SUP n. (emeasure M) (?M n \cap ?A))$
proof –
{ fix $n :: nat$
have $emeasure M \{x \in ?A. 1 \leq of_nat n * u x\} \leq$
 $of_nat n * \int^+ x. u x * indicator ?A x \partial M$
by (*intro nn_integral_Markov_inequality*) *auto*
also have $\{x \in ?A. 1 \leq of_nat n * u x\} = (?M n \cap ?A)$
by (*auto simp: ennreal_of_nat_eq_real_of_nat u_eq **)
finally have $emeasure M (?M n \cap ?A) \leq 0$
by (*simp add: ennreal_of_nat_eq_real_of_nat u_eq **)
moreover have $0 \leq (emeasure M) (?M n \cap ?A)$ **using** *u by auto*
ultimately have $(emeasure M) (?M n \cap ?A) = 0$ **by auto** }
thus *?thesis by simp*
qed

```

also have ... = (emeasure M) (⋃ n. ?M n ∩ ?A)
proof (safe intro!: SUP_emeasure_incseq)
  fix n show ?M n ∩ ?A ∈ sets M
    using u by (auto intro!: sets.Int)
next
show incseq (λn. {x ∈ space M. 1 ≤ real n * u x} ∩ {x ∈ space M. u x ≠
0})
proof (safe intro!: incseq_SucI)
  fix n :: nat and x
  assume *: 1 ≤ real n * u x
  also have real n * u x ≤ real (Suc n) * u x
    by (auto intro!: mult_right_mono)
  finally show 1 ≤ real (Suc n) * u x by auto
qed
qed
also have ... = (emeasure M) {x ∈ space M. 0 < u x}
proof (safe intro!: arg_cong[where f=(emeasure M)])
  fix x assume 0 < u x and [simp, intro]: x ∈ space M
  show x ∈ (⋃ n. ?M n ∩ ?A)
  proof (cases u x rule: ennreal_cases)
    case (real r) with ⟨0 < u x⟩ have 0 < r by auto
    obtain j :: nat where 1 / r ≤ real j using real_arch_simple ..
    hence 1 / r * r ≤ real j * r unfolding mult_le_cancel_right using ⟨0 <
r⟩ by auto
    hence 1 ≤ real j * r using real ⟨0 < r⟩ by auto
    thus ?thesis using ⟨0 < r⟩ real
    by (auto simp: ennreal_of_nat_eq_real_of_nat ennreal_1[symmetric]
ennreal_mult[symmetric]
simp del: ennreal_1)
  qed (insert ⟨0 < u x⟩, auto simp: ennreal_mult_top)
  qed (auto simp: zero_less_iff_neq_zero)
  finally show emeasure M ?A = 0
    by (simp add: zero_less_iff_neq_zero)
  qed
qed

```

lemma *nn_integral_0_iff_AE*:

assumes *u*: $u \in \text{borel_measurable } M$

shows $\text{integral}^N M u = 0 \iff (\text{AE } x \text{ in } M. u x = 0)$

proof –

have $\text{sets: } \{x \in \text{space } M. u x \neq 0\} \in \text{sets } M$

using *u* by auto

show $\text{integral}^N M u = 0 \iff (\text{AE } x \text{ in } M. u x = 0)$

using *nn_integral_0_iff*[of *u*] *AE_iff_null*[OF *sets*] *u* by auto

qed

lemma *AE_iff_nn_integral*:

$\{x \in \text{space } M. P x\} \in \text{sets } M \implies (\text{AE } x \text{ in } M. P x) \iff \text{integral}^N M (\text{indicator } \{x. \neg P x\}) = 0$

by (subst nn_integral_0_iff_AE) (auto simp: indicator_def[abs_def])

lemma *nn_integral_less*:

assumes [measurable]: $f \in \text{borel_measurable } M$ $g \in \text{borel_measurable } M$

assumes $f: (\int^+ x. f x \partial M) \neq \infty$

assumes *ord*: $\text{AE } x \text{ in } M. f x \leq g x \neg (\text{AE } x \text{ in } M. g x \leq f x)$

shows $(\int^+ x. f x \partial M) < (\int^+ x. g x \partial M)$

proof –

have $0 < (\int^+ x. g x - f x \partial M)$

proof (*intro order_le_neq_trans notI*)

assume $0 = (\int^+ x. g x - f x \partial M)$

then have $\text{AE } x \text{ in } M. g x - f x = 0$

using *nn_integral_0_iff_AE*[of $\lambda x. g x - f x M$] **by** *simp*

with *ord*(1) **have** $\text{AE } x \text{ in } M. g x \leq f x$

by *eventually_elim* (auto simp: *ennreal_minus_eq_0*)

with *ord* **show** *False*

by *simp*

qed *simp*

also have $\dots = (\int^+ x. g x \partial M) - (\int^+ x. f x \partial M)$

using f **by** (*subst nn_integral_diff*) (auto simp: *ord*)

finally show *?thesis*

using f **by** (auto dest!: *ennreal_minus_pos_iff*[rotated] *simp: less_top*)

qed

lemma *nn_integral_subalgebra*:

assumes $f: f \in \text{borel_measurable } N$

and $N: \text{sets } N \subseteq \text{sets } M \text{ space } N = \text{space } M \wedge A. A \in \text{sets } N \implies \text{emeasure } N$
 $A = \text{emeasure } M A$

shows $\text{integral}^N N f = \text{integral}^N M f$

proof –

have [*simp*]: $\wedge f :: 'a \Rightarrow \text{ennreal}. f \in \text{borel_measurable } N \implies f \in \text{borel_measurable } M$

using N **by** (auto simp: *measurable_def*)

have [*simp*]: $\wedge P. (\text{AE } x \text{ in } N. P x) \implies (\text{AE } x \text{ in } M. P x)$

using N **by** (auto simp add: *eventually_ae_filter null_sets_def subset_eq*)

have [*simp*]: $\wedge A. A \in \text{sets } N \implies A \in \text{sets } M$

using N **by** *auto*

from f **show** *?thesis*

apply *induct*

apply (*simp_all* add: *nn_integral_add nn_integral_cmult nn_integral_monotone_convergence_SUP N image_comp*)

apply (auto intro!: *nn_integral_cong cong: nn_integral_cong simp: N(2)[symmetric]*)

done

qed

lemma *nn_integral_nat_function*:

fixes $f :: 'a \Rightarrow \text{nat}$

assumes $f \in \text{measurable } M$ (*count_space UNIV*)

shows $(\int^+ x. \text{of_nat } (f x) \partial M) = (\sum t. \text{emeasure } M \{x \in \text{space } M. t < f x\})$

proof –

```

define  $F$  where  $F\ i = \{x \in \text{space } M. i < f\ x\}$  for  $i$ 
with  $\text{assms}$  have  $[\text{measurable}]$ :  $\bigwedge i. F\ i \in \text{sets } M$ 
by  $\text{auto}$ 

{ fix  $x$  assume  $x \in \text{space } M$ 
have  $(\lambda i. \text{if } i < f\ x \text{ then } 1 \text{ else } 0)$   $\text{sums } (\text{of\_nat } (f\ x)::\text{real})$ 
using  $\text{sums\_If\_finite}[\text{of } \lambda i. i < f\ x\ \lambda_. 1::\text{real}]$  by  $\text{simp}$ 
then have  $(\lambda i. \text{ennreal } (\text{if } i < f\ x \text{ then } 1 \text{ else } 0))$   $\text{sums of\_nat}(f\ x)$ 
unfolding  $\text{ennreal\_of\_nat\_eq\_real\_of\_nat}$ 
by  $(\text{subst } \text{sums\_ennreal})$   $\text{auto}$ 
moreover have  $\bigwedge i. \text{ennreal } (\text{if } i < f\ x \text{ then } 1 \text{ else } 0) = \text{indicator } (F\ i)\ x$ 
using  $\langle x \in \text{space } M \rangle$  by  $(\text{simp } \text{add: one\_ennreal\_def } F\_def)$ 
ultimately have  $\text{of\_nat } (f\ x) = (\sum i. \text{indicator } (F\ i)\ x :: \text{ennreal})$ 
by  $(\text{simp } \text{add: sums\_iff})$  }
then have  $(\int^+ x. \text{of\_nat } (f\ x)\ \partial M) = (\int^+ x. (\sum i. \text{indicator } (F\ i)\ x)\ \partial M)$ 
by  $(\text{simp } \text{cong: nn\_integral\_cong})$ 
also have  $\dots = (\sum i. \text{emeasure } M\ (F\ i))$ 
by  $(\text{simp } \text{add: nn\_integral\_suminf})$ 
finally show  $?thesis$ 
by  $(\text{simp } \text{add: } F\_def)$ 

```

qed

theorem nn_integral_lfp :

```

assumes  $\text{sets}[\text{simp}]$ :  $\bigwedge s. \text{sets } (M\ s) = \text{sets } N$ 
assumes  $f$ :  $\text{sup\_continuous } f$ 
assumes  $g$ :  $\text{sup\_continuous } g$ 
assumes  $\text{meas}$ :  $\bigwedge F. F \in \text{borel\_measurable } N \implies f\ F \in \text{borel\_measurable } N$ 
assumes  $\text{step}$ :  $\bigwedge F\ s. F \in \text{borel\_measurable } N \implies \text{integral}^N (M\ s)\ (f\ F) = g$ 
 $(\lambda s. \text{integral}^N (M\ s)\ (F)\ s)$ 
shows  $(\int^+ \omega. \text{lfp } f\ \omega\ \partial M\ s) = \text{lfp } g\ s$ 
proof  $(\text{subst } \text{lfp\_transfer\_bounded}[\text{where } \alpha = \lambda F\ s. \int^+ x. F\ x\ \partial M\ s \text{ and } g = g \text{ and } f = f \text{ and } P = \lambda f. f \in \text{borel\_measurable } N, \text{ symmetric}]$ 
fix  $C :: \text{nat} \Rightarrow 'b \Rightarrow \text{ennreal}$  assume  $\text{incseq } C \bigwedge i. C\ i \in \text{borel\_measurable } N$ 
then show  $(\lambda s. \int^+ x. (\text{SUP } i. C\ i)\ x\ \partial M\ s) = (\text{SUP } i. (\lambda s. \int^+ x. C\ i\ x\ \partial M\ s))$ 
unfolding  $\text{SUP\_apply}[\text{abs\_def}]$ 
by  $(\text{subst } \text{nn\_integral\_monotone\_convergence\_SUP})$ 
 $(\text{auto } \text{simp: mono\_def } \text{fun\_eq\_iff } \text{intro!: arg\_cong2}[\text{where } f = \text{emeasure}]$ 
 $\text{cong: measurable\_cong\_sets})$ 
qed  $(\text{auto } \text{simp } \text{add: step } \text{le\_fun\_def } \text{SUP\_apply}[\text{abs\_def}] \text{ bot\_fun\_def } \text{bot\_ennreal}$ 
 $\text{intro!: meas } f\ g)$ 

```

theorem nn_integral_gfp :

```

assumes  $\text{sets}[\text{simp}]$ :  $\bigwedge s. \text{sets } (M\ s) = \text{sets } N$ 
assumes  $f$ :  $\text{inf\_continuous } f$  and  $g$ :  $\text{inf\_continuous } g$ 
assumes  $\text{meas}$ :  $\bigwedge F. F \in \text{borel\_measurable } N \implies f\ F \in \text{borel\_measurable } N$ 
assumes  $\text{bound}$ :  $\bigwedge F\ s. F \in \text{borel\_measurable } N \implies (\int^+ x. f\ F\ x\ \partial M\ s) < \infty$ 
assumes  $\text{non\_zero}$ :  $\bigwedge s. \text{emeasure } (M\ s)\ (\text{space } (M\ s)) \neq 0$ 
assumes  $\text{step}$ :  $\bigwedge F\ s. F \in \text{borel\_measurable } N \implies \text{integral}^N (M\ s)\ (f\ F) = g$ 

```

```

( $\lambda s. \text{integral}^N (M s) F$ ) s
  shows  $(\int^{+\omega}. \text{gfp } f \ \omega \ \partial M s) = \text{gfp } g s$ 
proof (subst gfp_transfer_bounded [where  $\alpha = \lambda F s. \int^{+x}. F x \ \partial M s$  and  $g = g$  and  $f = f$ 
  and  $P = \lambda F. F \in \text{borel\_measurable } N \wedge (\forall s. (\int^{+x}. F x \ \partial M s) < \infty)$ , symmetric])
  fix  $C :: \text{nat} \Rightarrow 'b \Rightarrow \text{ennreal}$  assume decseq  $C \wedge i. C i \in \text{borel\_measurable } N \wedge$ 
 $(\forall s. \text{integral}^N (M s) (C i) < \infty)$ 
  then show  $(\lambda s. \int^{+x}. (\text{INF } i. C i) x \ \partial M s) = (\text{INF } i. (\lambda s. \int^{+x}. C i x \ \partial M s))$ 
    unfolding INF_apply[abs_def]
    by (subst nn_integral_monotone_convergence_INF_decseq)
      (auto simp: mono_def fun_eq_iff intro!: arg_cong2[where f = emeasure])
  cong: measurable_cong_sets)
next
  show  $\bigwedge x. g x \leq (\lambda s. \text{integral}^N (M s) (f \text{ top}))$ 
    by (subst step)
      (auto simp add: top_fun_def less_le non_zero le_fun_def ennreal_top_mult
      cong del: if_weak_cong intro!: monoD[OF inf_continuous_mono[OF g], THEN le_funD])
next
  fix  $C$  assume  $\bigwedge i::\text{nat}. C i \in \text{borel\_measurable } N \wedge (\forall s. \text{integral}^N (M s) (C i) < \infty)$  decseq  $C$ 
  with bound show  $\text{Inf } (C \text{ ' UNIV}) \in \text{borel\_measurable } N \wedge (\forall s. \text{integral}^N (M s) (\text{Inf } (C \text{ ' UNIV})) < \infty)$ 
    unfolding INF_apply[abs_def]
    by (subst nn_integral_monotone_convergence_INF_decseq)
      (auto simp: INF_less_iff cong: measurable_cong_sets intro!: borel_measurable_INF)
next
  show  $\bigwedge x. x \in \text{borel\_measurable } N \wedge (\forall s. \text{integral}^N (M s) x < \infty) \implies$ 
 $(\lambda s. \text{integral}^N (M s) (f x)) = g (\lambda s. \text{integral}^N (M s) x)$ 
    by (subst step) auto
qed (insert bound, auto simp add: le_fun_def INF_apply[abs_def] top_fun_def intro!: meas f g)

```

Cauchy–Schwarz inequality for integral^N

lemma *sum_of_squares_ge_ennreal*:

fixes $a b :: \text{ennreal}$

shows $2 * a * b \leq a^2 + b^2$

proof (*cases a; cases b*)

fix $x y$

assume $xy: x \geq 0 \ y \geq 0$ **and** [*simp*]: $a = \text{ennreal } x \ b = \text{ennreal } y$

have $0 \leq (x - y)^2$

by *simp*

also have $\dots = x^2 + y^2 - 2 * x * y$

by (*simp add: algebra_simps power2_eq_square*)

finally have $2 * x * y \leq x^2 + y^2$

by *simp*

hence $\text{ennreal } (2 * x * y) \leq \text{ennreal } (x^2 + y^2)$

by (*intro ennreal_leI*)

```

thus ?thesis using xy
  by (simp add: ennreal_mult ennreal_power)
qed auto

lemma Cauchy_Schwarz_nn_integral:
  assumes [measurable]: f ∈ borel_measurable M g ∈ borel_measurable M
  shows (∫+x. f x * g x ∂M)2 ≤ (∫+x. f x ^ 2 ∂M) * (∫+x. g x ^ 2 ∂M)
proof (cases (∫+x. f x * g x ∂M) = 0)
  case False
  define F where F = nn_integral M (λx. f x ^ 2)
  define G where G = nn_integral M (λx. g x ^ 2)
  from False have ¬(AE x in M. f x = 0 ∨ g x = 0)
    by (auto simp: nn_integral_0_iff_AE)
  hence ¬(AE x in M. f x = 0) and ¬(AE x in M. g x = 0)
    by (auto intro: AE_disjI1 AE_disjI2)
  hence nz: F ≠ 0 G ≠ 0
    by (auto simp: nn_integral_0_iff_AE F_def G_def)

show ?thesis
proof (cases F = ∞ ∨ G = ∞)
  case True
  thus ?thesis using nz
    by (auto simp: F_def G_def)
next
  case False
  define F' where F' = ennreal (sqrt (enn2real F))
  define G' where G' = ennreal (sqrt (enn2real G))
  from False have fin: F < top G < top
    by (simp_all add: top.not_eq_extremum)
  have F'_sqr: F'^2 = F
    using False by (cases F) (auto simp: F'_def ennreal_power)
  have G'_sqr: G'^2 = G
    using False by (cases G) (auto simp: G'_def ennreal_power)
  have nz': F' ≠ 0 G' ≠ 0 and fin': F' ≠ ∞ G' ≠ ∞
    using F'_sqr G'_sqr nz fin by auto
  from fin' have fin'': F' < top G' < top
    by (auto simp: top.not_eq_extremum)

  have 2 * (F' / F') * (G' / G') * (∫+x. f x * g x ∂M) =
    F' * G' * (∫+x. 2 * (f x / F') * (g x / G') ∂M)
    using nz' fin''
    by (simp add: divide_ennreal_def algebra_simps ennreal_inverse_mult flip:
nn_integral_cmult)
  also have F' / F' = 1
    using nz' fin'' by simp
  also have G' / G' = 1
    using nz' fin'' by simp
  also have 2 * 1 * 1 = (2 :: ennreal) by simp
  also have F' * G' * (∫+x. 2 * (f x / F') * (g x / G') ∂M) ≤

```

```

      
$$F' * G' * (\int^+ x. (f x / F')^2 + (g x / G')^2 \partial M)$$

    by (intro mult_left_mono nn_integral_mono sum_of_squares_ge_ennreal)
  auto
  also have ... = F' * G' * (F / F'^2 + G / G'^2) using nz
    by (auto simp: nn_integral_add algebra_simps nn_integral_divide F_def
  G_def)
  also have F / F'^2 = 1
    using nz F'_sqr fin by simp
  also have G / G'^2 = 1
    using nz G'_sqr fin by simp
  also have F' * G' * (1 + 1) = 2 * (F' * G')
    by (simp add: mult_ac)
  finally have ( $\int^+ x. f x * g x \partial M$ ) ≤ F' * G'
    by (subst (asm) ennreal_mult_le_mult_iff) auto
  hence ( $\int^+ x. f x * g x \partial M$ )^2 ≤ (F' * G')^2
    by (intro power_mono_ennreal)
  also have ... = F * G
    by (simp add: algebra_simps F'_sqr G'_sqr)
  finally show ?thesis
    by (simp add: F_def G_def)
qed
qed auto

```

7.5.5 Integral under concrete measures

lemma nn_integral_mono_measure:

assumes sets M = sets N M ≤ N shows nn_integral M f ≤ nn_integral N f
 unfolding nn_integral_def

proof (intro SUP_subset_mono)

note ⟨sets M = sets N⟩[simp] ⟨sets M = sets N⟩[THEN sets_eq_imp_space_eq,
 simp]

show {g. simple_function M g ∧ g ≤ f} ⊆ {g. simple_function N g ∧ g ≤ f}
 by (simp add: simple_function_def)

show $\text{integral}^S M x \leq \text{integral}^S N x$ for x
 using le_measureD3[OF ⟨M ≤ N⟩]

by (auto simp add: simple_integral_def intro!: sum_mono mult_mono)

qed

lemma nn_integral_empty:

assumes space M = {}

shows nn_integral M f = 0

proof –

have ($\int^+ x. f x \partial M$) = ($\int^+ x. 0 \partial M$)

by (rule nn_integral_cong) (simp add: assms)

thus ?thesis by simp

qed

lemma nn_integral_bot[simp]: nn_integral bot f = 0

by (simp add: nn_integral_empty)

Distributions

lemma *nn_integral_distr*:

assumes $T: T \in \text{measurable } M \ M'$ **and** $f: f \in \text{borel_measurable } (\text{distr } M \ M' \ T)$

shows $\text{integral}^N (\text{distr } M \ M' \ T) f = (\int^+ x. f (T x) \ \partial M)$

using f

proof *induct*

case (*cong f g*)

with T **show** *?case*

apply (*subst nn_integral_cong[of _ f g]*)

apply *simp*

apply (*subst nn_integral_cong[of _ $\lambda x. f (T x)$ $\lambda x. g (T x)$]*)

apply (*simp add: measurable_def Pi_iff*)

apply *simp*

done

next

case (*set A*)

then have $eq: \bigwedge x. x \in \text{space } M \implies \text{indicator } A (T x) = \text{indicator } (T - ' A \cap \text{space } M) x$

by (*auto simp: indicator_def*)

from *set T show ?case*

by (*subst nn_integral_cong[OF eq]*)

(*auto simp add: emeasure_distr intro!: nn_integral_indicator[symmetric] measurable_sets*)

qed (*simp_all add: measurable_compose[OF T] T nn_integral_cmult nn_integral_add nn_integral_monotone_convergence_SUP le_fun_def incseq_def image_comp*)

Counting space

lemma *simple_function_count_space[simp]*:

simple_function (*count_space A*) $f \longleftrightarrow \text{finite } (f ' A)$

unfolding *simple_function_def* **by** *simp*

lemma *nn_integral_count_space*:

assumes $A: \text{finite } \{a \in A. 0 < f a\}$

shows $\text{integral}^N (\text{count_space } A) f = (\sum a | a \in A \wedge 0 < f a. f a)$

proof –

have $*$: $(\int^+ x. \max 0 (f x) \ \partial \text{count_space } A) =$

$(\int^+ x. (\sum a | a \in A \wedge 0 < f a. f a * \text{indicator } \{a\} x) \ \partial \text{count_space } A)$

by (*auto intro!: nn_integral_cong*

simp add: indicator_def of_bool_def if_distrib sum.If_cases[OF A] max_def le_less)

also have $\dots = (\sum a | a \in A \wedge 0 < f a. \int^+ x. f a * \text{indicator } \{a\} x \ \partial \text{count_space } A)$

by (*subst nn_integral_sum*) (*simp_all add: AE_count_space less_imp_le*)

also have $\dots = (\sum a | a \in A \wedge 0 < f a. f a)$

by (*auto intro!: sum.cong simp: one_enreal_def[symmetric] max_def*)

finally show *?thesis* **by** (*simp add: max.absorb2*)

qed

lemma *nn_integral_count_space_finite:*

finite A $\implies (\int^+ x. f x \partial \text{count_space } A) = (\sum a \in A. f a)$

by (*auto intro!: sum.mono_neutral_left simp: nn_integral_count_space_less_le*)

lemma *nn_integral_count_space':*

assumes *finite A* $\wedge x. x \in B \implies x \notin A \implies f x = 0$ $A \subseteq B$

shows $(\int^+ x. f x \partial \text{count_space } B) = (\sum x \in A. f x)$

proof –

have $(\int^+ x. f x \partial \text{count_space } B) = (\sum a \mid a \in B \wedge 0 < f a. f a)$

using *assms(2,3)*

by (*intro nn_integral_count_space_finite_subset[OF _ ‹finite A›] (auto simp: less_le)*)

also have $\dots = (\sum a \in A. f a)$

using *assms* **by** (*intro sum.mono_neutral_cong_left (auto simp: less_le)*)

finally show *?thesis* .

qed

lemma *nn_integral_bij_count_space:*

assumes *g: bij_betw g A B*

shows $(\int^+ x. f (g x) \partial \text{count_space } A) = (\int^+ x. f x \partial \text{count_space } B)$

using *g[THEN bij_betw_imp_funcset]*

by (*subst distr_bij_count_space[OF g, symmetric]*)

(*auto intro!: nn_integral_distr[symmetric]*)

lemma *nn_integral_indicator_finite:*

fixes *f :: 'a \Rightarrow ennreal*

assumes *f: finite A and [measurable]: $\wedge a. a \in A \implies \{a\} \in \text{sets } M$*

shows $(\int^+ x. f x * \text{indicator } A x \partial M) = (\sum x \in A. f x * \text{emeasure } M \{x\})$

proof –

from *f* **have** $(\int^+ x. f x * \text{indicator } A x \partial M) = (\int^+ x. (\sum a \in A. f a * \text{indicator } \{a\} x) \partial M)$

by (*intro nn_integral_cong (auto simp: indicator_def if_distrib[where f= $\lambda a.$ $x * a$ for x] sum.If_cases)*)

also have $\dots = (\sum a \in A. f a * \text{emeasure } M \{a\})$

by (*subst nn_integral_sum*) *auto*

finally show *?thesis* .

qed

lemma *nn_integral_count_space_nat:*

fixes *f :: nat \Rightarrow ennreal*

shows $(\int^+ i. f i \partial \text{count_space } UNIV) = (\sum i. f i)$

proof –

have $(\int^+ i. f i \partial \text{count_space } UNIV) =$

$(\int^+ i. (\sum j. f j * \text{indicator } \{j\} i) \partial \text{count_space } UNIV)$

proof (*intro nn_integral_cong*)

fix *i*

have $f i = (\sum j \in \{i\}. f j * \text{indicator } \{j\} i)$

```

    by simp
    also have ... = (∑ j. f j * indicator {j} i)
    by (rule suminf_finite[symmetric]) auto
    finally show f i = (∑ j. f j * indicator {j} i) .
  qed
  also have ... = (∑ j. (∫+ i. f j * indicator {j} i) ∂count_space UNIV)
  by (rule nn_integral_suminf) auto
  finally show ?thesis
  by simp
qed

```

lemma *nn_integral_enat_function*:

```

  assumes f: f ∈ measurable M (count_space UNIV)
  shows (∫+ x. ennreal_of_enat (f x) ∂M) = (∑ t. emeasure M {x ∈ space M.
t < f x})
  proof -
    define F where F i = {x ∈ space M. i < f x} for i :: nat
    with assms have [measurable]: ∧ i. F i ∈ sets M
    by auto

```

```

    { fix x assume x ∈ space M
      have (λ i :: nat. if i < f x then 1 else 0) sums ennreal_of_enat (f x)
      using sums_If_finite[of λ r. r < f x λ_. 1 :: ennreal]
      by (cases f x) (simp_all add: sums_def of_nat_tendsto_top_ennreal)
      also have (λ i. (if i < f x then 1 else 0)) = (λ i. indicator (F i) x)
      using ⟨x ∈ space M⟩ by (simp add: one_ennreal_def F_def fun_eq_iff)
      finally have ennreal_of_enat (f x) = (∑ i. indicator (F i) x)
      by (simp add: sums_iff) }
    then have (∫+ x. ennreal_of_enat (f x) ∂M) = (∫+ x. (∑ i. indicator (F i) x)
∂M)
    by (simp cong: nn_integral_cong)
    also have ... = (∑ i. emeasure M (F i))
    by (simp add: nn_integral_suminf)
    finally show ?thesis
    by (simp add: F_def)
  qed

```

lemma *nn_integral_count_space_nn_integral*:

```

  fixes f :: 'i ⇒ 'a ⇒ ennreal
  assumes countable I and [measurable]: ∧ i. i ∈ I ⇒ f i ∈ borel_measurable M
  shows (∫+ x. ∫+ i. f i x ∂count_space I ∂M) = (∫+ i. ∫+ x. f i x ∂M ∂count_space
I)
  proof cases
    assume finite I then show ?thesis
    by (simp add: nn_integral_count_space_finite nn_integral_sum)
  next
    assume infinite I
    then have [simp]: I ≠ {}
    by auto

```

note * = *bij_betw_from_nat_into*[*OF* ‹*countable I*› ‹*infinite I*›]
have **: $\bigwedge f. (\bigwedge i. 0 \leq f i) \implies (\int^+ i. f i \partial \text{count_space } I) = (\sum n. f (\text{from_nat_into } I n))$
by (*simp add: nn_integral_bij_count_space*[*symmetric, OF **] *nn_integral_count_space_nat*)
show ?thesis
by (*simp add: ** nn_integral_suminf_from_nat_into*)
qed

lemma *of_bool_Bex_eq_nn_integral*:

assumes *unique*: $\bigwedge x y. x \in X \implies y \in X \implies P x \implies P y \implies x = y$
shows *of_bool* ($\exists y \in X. P y$) = $(\int^+ y. \text{of_bool } (P y) \partial \text{count_space } X)$
proof *cases*
assume $\exists y \in X. P y$
then obtain *y* **where** $P y$ *y* $\in X$ **by** *auto*
then show ?thesis
by (*subst nn_integral_count_space'*[**where** $A = \{y\}$]) (*auto dest: unique*)
qed (*auto cong: nn_integral_cong_simp*)

lemma *emeasure_UN_countable*:

assumes *sets*[*measurable*]: $\bigwedge i. i \in I \implies X i \in \text{sets } M$ **and** *I*[*simp*]: *countable I*
assumes *disj*: *disjoint_family_on* $X I$
shows *emeasure* $M (\bigcup (X ' I)) = (\int^+ i. \text{emeasure } M (X i) \partial \text{count_space } I)$
proof –
have *eq*: $\bigwedge x. \text{indicator } (\bigcup (X ' I)) x = \int^+ i. \text{indicator } (X i) x \partial \text{count_space } I$
proof *cases*
fix *x* **assume** $x \in \bigcup (X ' I)$
then obtain *j* **where** $x \in X j$ $j \in I$
by *auto*
with *disj* **have** $\bigwedge i. i \in I \implies \text{indicator } (X i) x = (\text{indicator } \{j\} i :: \text{ennreal})$
by (*auto simp: disjoint_family_on_def split: split_indicator*)
with *x j* **show** ?thesis *x*
by (*simp cong: nn_integral_cong_simp*)
qed (*auto simp: nn_integral_0_iff AE*)

note *sets.countable_UN'*[*unfolded subset_eq, measurable*]
have *emeasure* $M (\bigcup (X ' I)) = (\int^+ x. \text{indicator } (\bigcup (X ' I)) x \partial M)$
by *simp*
also have $\dots = (\int^+ i. \int^+ x. \text{indicator } (X i) x \partial M \partial \text{count_space } I)$
by (*simp add: eq_nn_integral_count_space_nn_integral*)
finally show ?thesis
by (*simp cong: nn_integral_cong_simp*)
qed

lemma *emeasure_countable_singleton*:

assumes *sets*: $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ **and** *X*: *countable X*
shows *emeasure* $M X = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count_space } X)$
proof –
have *emeasure* $M (\bigcup i \in X. \{i\}) = (\int^+ x. \text{emeasure } M \{x\} \partial \text{count_space } X)$
using *assms* **by** (*intro emeasure_UN_countable*) (*auto simp: disjoint_family_on_def*)

also have $(\bigcup_{i \in X}. \{i\}) = X$ by *auto*
 finally show *?thesis* .

qed

lemma *measure_eqI_countable*:

assumes [*simp*]: sets $M = \text{Pow } A$ sets $N = \text{Pow } A$ and A : *countable* A

assumes *eq*: $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$

shows $M = N$

proof (rule *measure_eqI*)

fix X assume $X \in \text{sets } M$

then have $X: X \subseteq A$ by *auto*

moreover from $A \ X$ have *countable* X by (*auto dest: countable_subset*)

ultimately have

$\text{emeasure } M \ X = (\int^+ a. \text{emeasure } M \{a\} \ \partial \text{count_space } X)$

$\text{emeasure } N \ X = (\int^+ a. \text{emeasure } N \{a\} \ \partial \text{count_space } X)$

by (*auto intro!: emeasure_countable_singleton*)

moreover have $(\int^+ a. \text{emeasure } M \{a\} \ \partial \text{count_space } X) = (\int^+ a. \text{emeasure } N \{a\} \ \partial \text{count_space } X)$

using X by (*intro nn_integral_cong eq*) *auto*

ultimately show $\text{emeasure } M \ X = \text{emeasure } N \ X$

by *simp*

qed *simp*

lemma *measure_eqI_countable_AE*:

assumes [*simp*]: sets $M = \text{UNIV}$ sets $N = \text{UNIV}$

assumes *ae*: *AE* x in M . $x \in \Omega$ *AE* x in N . $x \in \Omega$ and [*simp*]: *countable* Ω

assumes *eq*: $\bigwedge x. x \in \Omega \implies \text{emeasure } M \{x\} = \text{emeasure } N \{x\}$

shows $M = N$

proof (rule *measure_eqI*)

fix A

have $\text{emeasure } N \ A = \text{emeasure } N \{x \in \Omega. x \in A\}$

using *ae* by (*intro emeasure_eq_AE*) *auto*

also have $\dots = (\int^+ x. \text{emeasure } N \{x\} \ \partial \text{count_space } \{x \in \Omega. x \in A\})$

by (*intro emeasure_countable_singleton*) *auto*

also have $\dots = (\int^+ x. \text{emeasure } M \{x\} \ \partial \text{count_space } \{x \in \Omega. x \in A\})$

by (*intro nn_integral_cong eq[symmetric]*) *auto*

also have $\dots = \text{emeasure } M \{x \in \Omega. x \in A\}$

by (*intro emeasure_countable_singleton[symmetric]*) *auto*

also have $\dots = \text{emeasure } M \ A$

using *ae* by (*intro emeasure_eq_AE*) *auto*

finally show $\text{emeasure } M \ A = \text{emeasure } N \ A$..

qed *simp*

lemma *nn_integral_monotone_convergence_SUP_nat*:

fixes $f :: 'a \Rightarrow \text{nat} \Rightarrow \text{ennreal}$

assumes *chain*: *Complete_Partial_Order.chain* (\leq) $(f \ ' Y)$

and *nonempty*: $Y \neq \{\}$

shows $(\int^+ x. (\text{SUP } i \in Y. f \ i \ x) \ \partial \text{count_space } \text{UNIV}) = (\text{SUP } i \in Y. (\int^+ x. f \ i \ x \ \partial \text{count_space } \text{UNIV}))$

```

(is ?lhs = ?rhs is integralN ?M _ = _)
proof (rule order_class.order.antisym)
  show ?rhs ≤ ?lhs
    by (auto intro!: SUP_least SUP_upper nn_integral_mono)
next
  have  $\exists g. \text{incseq } g \wedge \text{range } g \subseteq (\lambda i. f i x) \text{ ' } Y \wedge (\text{SUP } i \in Y. f i x) = (\text{SUP } i. g i)$ 
  for  $x$ 
    by (rule ennreal_Sup_countable_SUP) (simp add: nonempty)
  then obtain  $g$  where  $\text{incseq: } \bigwedge x. \text{incseq } (g x)$ 
    and  $\text{range: } \bigwedge x. \text{range } (g x) \subseteq (\lambda i. f i x) \text{ ' } Y$ 
    and  $\text{sup: } \bigwedge x. (\text{SUP } i \in Y. f i x) = (\text{SUP } i. g x i)$  by moura
  from  $\text{incseq}$  have  $\text{incseq': } \text{incseq } (\lambda i x. g x i)$ 
    by (blast intro: incseq_SucI le_funI dest: incseq_SucD)

  have  $?\text{lhs} = \int^+ x. (\text{SUP } i. g x i) \partial ?M$  by (simp add: sup)
  also have  $\dots = (\text{SUP } i. \int^+ x. g x i \partial ?M)$  using  $\text{incseq'}$ 
    by (rule nn_integral_monotone_convergence_SUP) simp
  also have  $\dots \leq (\text{SUP } i \in Y. \int^+ x. f i x \partial ?M)$ 
  proof (rule SUP_least)
    fix  $n$ 
    have  $\bigwedge x. \exists i. g x n = f i x \wedge i \in Y$  using  $\text{range}$  by blast
    then obtain  $I$  where  $I: \bigwedge x. g x n = f (I x) x \wedge x. I x \in Y$  by moura

    have  $(\int^+ x. g x n \partial \text{count\_space } UNIV) = (\sum x. g x n)$ 
      by (rule nn_integral_count_space_nat)
    also have  $\dots = (\text{SUP } m. \sum x < m. g x n)$ 
      by (rule suminf_eq_SUP)
    also have  $\dots \leq (\text{SUP } i \in Y. \int^+ x. f i x \partial ?M)$ 
  proof (rule SUP_mono)
    fix  $m$ 
    show  $\exists m' \in Y. (\sum x < m. g x n) \leq (\int^+ x. f m' x \partial ?M)$ 
  proof (cases  $m > 0$ )
    case False
      thus ?thesis using nonempty by auto
    next
    case True
      let  $?Y = I \text{ ' } \{..<m\}$ 
      have  $f \text{ ' } ?Y \subseteq f \text{ ' } Y$  using  $I$  by auto
      with  $\text{chain}$  have  $\text{chain' : Complete\_Partial\_Order.chain } (\leq) (f \text{ ' } ?Y)$  by (rule
      chain_subset)
      hence  $\text{Sup } (f \text{ ' } ?Y) \in f \text{ ' } ?Y$ 
      by (rule ccpo_class.in_chain_finite) (auto simp add: True lessThan_empty_iff)
      then obtain  $m'$  where  $m' < m$  and  $m': (\text{SUP } i \in ?Y. f i) = f (I m')$  by
      auto

      have  $I m' \in Y$  using  $I$  by blast
      have  $(\sum x < m. g x n) \leq (\sum x < m. f (I m') x)$ 
      proof (rule sum_mono)
        fix  $x$ 
        assume  $x \in \{..<m\}$ 

```

hence $x < m$ by *simp*
 have $g\ x\ n = f\ (I\ x)\ x$ by (*simp add: I*)
 also have $\dots \leq (SUP\ i \in ?Y. f\ i)\ x$ unfolding *Sup_fun_def image_image*
 using $\langle x \in \{..<m\} \rangle$ by (*rule Sup_upper [OF imageI]*)
 also have $\dots = f\ (I\ m')\ x$ unfolding *m'* by *simp*
 finally show $g\ x\ n \leq f\ (I\ m')\ x$.
 qed
 also have $\dots \leq (SUP\ m. (\sum\ x < m. f\ (I\ m')\ x))$
 by (*rule SUP_upper simp*)
 also have $\dots = (\sum\ x. f\ (I\ m')\ x)$
 by (*rule suminf_eq_SUP[symmetric]*)
 also have $\dots = (\int^+ x. f\ (I\ m')\ x\ \partial ?M)$
 by (*rule nn_integral_count_space_nat[symmetric]*)
 finally show *?thesis* using $\langle I\ m' \in Y \rangle$ by *blast*
 qed
 qed
 finally show $(\int^+ x. g\ x\ n\ \partial count_space\ UNIV) \leq \dots$.
 qed
 finally show *?lhs* \leq *?rhs* .
 qed

lemma *power_series_tendsto_at_left*:

assumes *nonneg*: $\bigwedge i. 0 \leq f\ i$ and *summable*: $\bigwedge z. 0 \leq z \implies z < 1 \implies$ *summable*
 ($\lambda n. f\ n * z^{\widehat{n}}$)

shows $(\lambda z. ennreal\ (\sum n. f\ n * z^{\widehat{n}})) \longrightarrow (\sum n. ennreal\ (f\ n))$ (*at_left*
 ($1::real$))

proof (*intro tendsto_at_left_sequentially*)

 show $0 < (1::real)$ by *simp*

 fix $S :: nat \Rightarrow real$ assume *S*: $\bigwedge n. S\ n < 1 \bigwedge n. 0 < S\ n \longrightarrow 1$ *incseq S*

 then have *S_nonneg*: $\bigwedge i. 0 \leq S\ i$ by (*auto intro: less_imp_le*)

have $(\lambda i. (\int^+ n. f\ n * S\ i^{\widehat{n}}\ \partial count_space\ UNIV)) \longrightarrow (\int^+ n. ennreal\ (f\ n)$
∂count_space UNIV)

proof (*rule nn_integral_LIMSEQ*)

 show *incseq* $(\lambda i n. ennreal\ (f\ n * S\ i^{\widehat{n}}))$

 using *S* by (*auto intro!: mult_mono power_mono nonneg ennreal_leI*
simp: incseq_def le_fun_def less_imp_le)

 fix *n* have $(\lambda i. ennreal\ (f\ n * S\ i^{\widehat{n}})) \longrightarrow ennreal\ (f\ n * 1^{\widehat{n}})$

 by (*intro tendsto_intros tendsto_ennrealI S*)

 then show $(\lambda i. ennreal\ (f\ n * S\ i^{\widehat{n}})) \longrightarrow ennreal\ (f\ n)$

 by *simp*

qed (*auto simp: S_nonneg intro!: mult_nonneg_nonneg nonneg*)

also have $(\lambda i. (\int^+ n. f\ n * S\ i^{\widehat{n}}\ \partial count_space\ UNIV)) = (\lambda i. \sum n. f\ n * S$
i^{\widehat{n}})

 by (*subst nn_integral_count_space_nat*)

 (*intro ext suminf_ennreal2 mult_nonneg_nonneg nonneg S_nonneg*
zero_le_power summable S) $+$

 also have $(\int^+ n. ennreal\ (f\ n)\ \partial count_space\ UNIV) = (\sum n. ennreal\ (f\ n))$

 by (*simp add: nn_integral_count_space_nat nonneg*)

finally show $(\lambda n. \text{ennreal } (\sum na. f na * S n \wedge na)) \longrightarrow (\sum n. \text{ennreal } (f n))$
 .
 qed

Measures with Restricted Space

lemma *simple_function_restrict_space_ennreal*:

fixes $f :: 'a \Rightarrow \text{ennreal}$
 assumes $\Omega \cap \text{space } M \in \text{sets } M$
 shows *simple_function* (*restrict_space* $M \ \Omega$) $f \longleftrightarrow \text{simple_function } M \ (\lambda x. f x * \text{indicator } \Omega \ x)$
 proof –
 { assume *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega)$)
 then have *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega) \cup \{0\}$) by *simp*
 then have *finite* $((\lambda x. f x * \text{indicator } \Omega \ x) \text{ 'space } M)$
 by (rule *rev_finite_subset*) (auto split: *split_indicator simp: space_restrict_space*)
 }
 moreover
 { assume *finite* $((\lambda x. f x * \text{indicator } \Omega \ x) \text{ 'space } M)$
 then have *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega)$)
 by (rule *rev_finite_subset*) (auto split: *split_indicator simp: space_restrict_space*)
 }
 ultimately show *thesis*
 unfolding
simple_function_iff_borel_measurable borel_measurable_restrict_space_iff_ennreal[*OF assms*]
 by *auto*
 qed

lemma *simple_function_restrict_space*:

fixes $f :: 'a \Rightarrow 'b::\text{real_normed_vector}$
 assumes $\Omega \cap \text{space } M \in \text{sets } M$
 shows *simple_function* (*restrict_space* $M \ \Omega$) $f \longleftrightarrow \text{simple_function } M \ (\lambda x. \text{indicator } \Omega \ x *_{\mathbb{R}} f x)$
 proof –
 { assume *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega)$)
 then have *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega) \cup \{0\}$) by *simp*
 then have *finite* $((\lambda x. \text{indicator } \Omega \ x *_{\mathbb{R}} f x) \text{ 'space } M)$
 by (rule *rev_finite_subset*) (auto split: *split_indicator simp: space_restrict_space*)
 }
 moreover
 { assume *finite* $((\lambda x. \text{indicator } \Omega \ x *_{\mathbb{R}} f x) \text{ 'space } M)$
 then have *finite* ($f \text{ 'space } (\text{restrict_space } M \ \Omega)$)
 by (rule *rev_finite_subset*) (auto split: *split_indicator simp: space_restrict_space*)
 }
 ultimately show *thesis*
 unfolding *simple_function_iff_borel_measurable borel_measurable_restrict_space_iff*[*OF assms*]
 by *auto*

qed

lemma *simple_integral_restrict_space*:

assumes $\Omega: \Omega \cap \text{space } M \in \text{sets } M$ *simple_function* (*restrict_space* $M \ \Omega$) f
shows *simple_integral* (*restrict_space* $M \ \Omega$) $f = \text{simple_integral } M \ (\lambda x. f \ x * \text{indicator } \Omega \ x)$
using *simple_function_restrict_space_ennreal*[*THEN iffD1*, *OF* Ω , *THEN simple_functionD*(1)]
by (*auto simp add: space_restrict_space emeasure_restrict_space*[*OF* Ω (1)] *le_infI2 simple_integral_def*
split: split_indicator split_indicator_asm
intro!: sum.mono_neutral_cong_left ennreal_mult_left_cong arg_cong2[**where** $f = \text{emeasure}$])

lemma *nn_integral_restrict_space*:

assumes Ω [*simp*]: $\Omega \cap \text{space } M \in \text{sets } M$
shows *nn_integral* (*restrict_space* $M \ \Omega$) $f = \text{nn_integral } M \ (\lambda x. f \ x * \text{indicator } \Omega \ x)$
proof –
let $?R = \text{restrict_space } M \ \Omega$ **and** $?X = \lambda M \ f. \ \{s. \ \text{simple_function } M \ s \wedge s \leq f \wedge (\forall x. s \ x < \text{top})\}$
have *integral^S* $?R \ ' \ ?X \ ?R \ f = \text{integral}^S \ M \ ' \ ?X \ M \ (\lambda x. f \ x * \text{indicator } \Omega \ x)$
proof (*safe intro!: image_eqI*)
fix s **assume** $s: \text{simple_function } ?R \ s \ s \leq f \ \forall x. s \ x < \text{top}$
from s **show** *integral^S* (*restrict_space* $M \ \Omega$) $s = \text{integral}^S \ M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$
by (*intro simple_integral_restrict_space*) *auto*
from s **show** *simple_function* $M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$
by (*simp add: simple_function_restrict_space_ennreal*)
from s **show** $(\lambda x. s \ x * \text{indicator } \Omega \ x) \leq (\lambda x. f \ x * \text{indicator } \Omega \ x)$
 $\wedge x. s \ x * \text{indicator } \Omega \ x < \text{top}$
by (*auto split: split_indicator simp: le_fun_def image_subset_iff*)
next
fix s **assume** $s: \text{simple_function } M \ s \ s \leq (\lambda x. f \ x * \text{indicator } \Omega \ x) \ \forall x. s \ x < \text{top}$
then have *simple_function* $M \ (\lambda x. s \ x * \text{indicator } (\Omega \cap \text{space } M) \ x)$ (**is** $?s'$)
by (*intro simple_function_mult simple_function_indicator*) *auto*
also have $?s' \longleftrightarrow \text{simple_function } M \ (\lambda x. s \ x * \text{indicator } \Omega \ x)$
by (*rule simple_function_cong*) (*auto split: split_indicator*)
finally show $\text{sf: simple_function } (\text{restrict_space } M \ \Omega) \ s$
by (*simp add: simple_function_restrict_space_ennreal*)

from s **have** $s_{\text{eq}}: s = (\lambda x. s \ x * \text{indicator } \Omega \ x)$
by (*auto simp add: fun_eq_iff le_fun_def image_subset_iff*
split: split_indicator split_indicator_asm
intro: antisym)

show *integral^S* $M \ s = \text{integral}^S \ (\text{restrict_space } M \ \Omega) \ s$
by (*subst s_eq*) (*rule simple_integral_restrict_space[symmetric, OF* Ω *sf]*)


```

show  $\bigwedge x. s\ x < \text{top}$ 
  using  $s$  by (auto simp: image_subset_iff)
from  $s$  show  $s \leq f$ 
  by (subst  $s\_eq$ ) (auto simp: image_subset_iff le_fun_def split: split_indicator
split_indicator_asm)
qed
then show ?thesis
  unfolding nn_integral_def_finite by (simp cong del: SUP_cong_simp)
qed

```

```

lemma nn_integral_count_space_indicator:
  assumes NO_MATCH (UNIV::'a set) (X::'a set)
  shows  $(\int^+ x. f\ x\ \partial\text{count\_space}\ X) = (\int^+ x. f\ x * \text{indicator}\ X\ x\ \partial\text{count\_space}\ \text{UNIV})$ 
  by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)

```

```

lemma nn_integral_count_space_eq:
   $(\bigwedge x. x \in A - B \implies f\ x = 0) \implies (\bigwedge x. x \in B - A \implies f\ x = 0) \implies$ 
   $(\int^+ x. f\ x\ \partial\text{count\_space}\ A) = (\int^+ x. f\ x\ \partial\text{count\_space}\ B)$ 
  by (auto simp: nn_integral_count_space_indicator intro!: nn_integral_cong split:
split_indicator)

```

```

lemma nn_integral_ge_point:
  assumes  $x \in A$ 
  shows  $p\ x \leq \int^+ x. p\ x\ \partial\text{count\_space}\ A$ 
proof -
  from  $\text{assms}$  have  $p\ x \leq \int^+ x. p\ x\ \partial\text{count\_space}\ \{x\}$ 
  by (auto simp add: nn_integral_count_space_finite_max_def)
  also have  $\dots = \int^+ x'. p\ x' * \text{indicator}\ \{x\}\ x'\ \partial\text{count\_space}\ A$ 
  using  $\text{assms}$  by (auto simp add: nn_integral_count_space_indicator indicator_def intro!: nn_integral_cong)
  also have  $\dots \leq \int^+ x. p\ x\ \partial\text{count\_space}\ A$ 
  by (rule nn_integral_mono) (simp add: indicator_def)
  finally show ?thesis .
qed

```

Measure spaces with an associated density

```

definition density :: 'a measure  $\Rightarrow$  ('a  $\Rightarrow$  ennreal)  $\Rightarrow$  'a measure where
  density  $M\ f = \text{measure\_of}\ (\text{space}\ M)\ (\text{sets}\ M)\ (\lambda A. \int^+ x. f\ x * \text{indicator}\ A\ x\ \partial M)$ 

```

```

lemma
  shows sets_density[simp, measurable_cong]: sets (density  $M\ f) = \text{sets}\ M$ 
  and space_density[simp]: space (density  $M\ f) = \text{space}\ M$ 
  by (auto simp: density_def)

```

```

lemma space_density_imp[measurable_dest]:

```

$\bigwedge x M f. x \in \text{space } (\text{density } M f) \implies x \in \text{space } M$ **by auto**

lemma

shows *measurable_density_eq1*[simp]: $g \in \text{measurable } (\text{density } M g f) M g' \longleftrightarrow g \in \text{measurable } M g M g'$

and *measurable_density_eq2*[simp]: $h \in \text{measurable } M h (\text{density } M h' f) \longleftrightarrow h \in \text{measurable } M h M h'$

and *simple_function_density_eq*[simp]: $\text{simple_function } (\text{density } M u f) u \longleftrightarrow \text{simple_function } M u u$

unfolding *measurable_def* *simple_function_def* **by** *simp_all*

lemma *density_cong*: $f \in \text{borel_measurable } M \implies f' \in \text{borel_measurable } M \implies (AE x \text{ in } M. f x = f' x) \implies \text{density } M f = \text{density } M f'$

unfolding *density_def* **by** (*auto intro!*: *measure_of_eq_nn_integral_cong_AE sets.space_closed*)

lemma *emeasure_density*:

assumes *f[measurable]*: $f \in \text{borel_measurable } M$ **and** *A[measurable]*: $A \in \text{sets } M$

shows *emeasure* ($\text{density } M f$) $A = (\int^+ x. f x * \text{indicator } A x \partial M)$

(*is* $_ = ?\mu A$)

unfolding *density_def*

proof (*rule emeasure_measure_of_sigma*)

show *sigma_algebra* ($\text{space } M$) ($\text{sets } M$) ..

show *positive* ($\text{sets } M$) $?\mu$

using *f* **by** (*auto simp: positive_def*)

show *countably_additive* ($\text{sets } M$) $?\mu$

proof (*intro countably_additiveI*)

fix $A :: \text{nat} \Rightarrow 'a \text{ set}$ **assume** $\text{range } A \subseteq \text{sets } M$

then have $\bigwedge i. A i \in \text{sets } M$ **by auto**

then have $*$: $\bigwedge i. (\lambda x. f x * \text{indicator } (A i) x) \in \text{borel_measurable } M$

by auto

assume *disj*: *disjoint_family* A

then have $(\sum n. ?\mu (A n)) = (\int^+ x. (\sum n. f x * \text{indicator } (A n) x) \partial M)$

using $f * \text{by}$ (*subst nn_integral_suminf*) *auto*

also have $(\int^+ x. (\sum n. f x * \text{indicator } (A n) x) \partial M) = (\int^+ x. f x * (\sum n. \text{indicator } (A n) x) \partial M)$

using f **by** (*auto intro!*: *ennreal_suminf_cmult nn_integral_cong_AE*)

also have $\dots = (\int^+ x. f x * \text{indicator } (\bigcup n. A n) x \partial M)$

unfolding *suminf_indicator[OF disj]* ..

finally show $(\sum i. \int^+ x. f x * \text{indicator } (A i) x \partial M) = \int^+ x. f x * \text{indicator } (\bigcup i. A i) x \partial M$.

qed

qed fact

lemma *null_sets_density_iff*:

assumes *f*: $f \in \text{borel_measurable } M$

shows $A \in \text{null_sets } (\text{density } M f) \longleftrightarrow A \in \text{sets } M \wedge (AE x \text{ in } M. x \in A \longrightarrow f x = 0)$

proof -

```

{ assume A ∈ sets M
  have (∫+x. f x * indicator A x ∂M) = 0 ↔ emeasure M {x ∈ space M. f x
* indicator A x ≠ 0} = 0
  using f ⟨A ∈ sets M⟩ by (intro nn_integral_0_iff) auto
  also have ... ↔ (AE x in M. f x * indicator A x = 0)
  using f ⟨A ∈ sets M⟩ by (intro AE_iff_measurable[OF _ refl, symmetric])
auto
  also have (AE x in M. f x * indicator A x = 0) ↔ (AE x in M. x ∈ A →
f x ≤ 0)
  by (auto simp add: indicator_def max_def split: if_split_asm)
  finally have (∫+x. f x * indicator A x ∂M) = 0 ↔ (AE x in M. x ∈ A →
f x ≤ 0) . }
with f show ?thesis
  by (simp add: null_sets_def emeasure_density cong: conj_cong)
qed

```

lemma *AE_density*:

```

assumes f: f ∈ borel_measurable M
shows (AE x in density M f. P x) ↔ (AE x in M. 0 < f x → P x)
proof
  assume AE x in density M f. P x
  with f obtain N where {x ∈ space M. ¬ P x} ⊆ N N ∈ sets M and ae: AE x
in M. x ∈ N → f x = 0
  by (auto simp: eventually_ae_filter null_sets_density_iff)
  then have AE x in M. x ∉ N → P x by auto
  with ae show AE x in M. 0 < f x → P x
  by (rule eventually_elim2) auto
next
  fix N assume ae: AE x in M. 0 < f x → P x
  then obtain N where {x ∈ space M. ¬ (0 < f x → P x)} ⊆ N N ∈ null_sets
M
  by (auto simp: eventually_ae_filter)
  then have *: {x ∈ space (density M f). ¬ P x} ⊆ N ∪ {x ∈ space M. f x = 0}
  N ∪ {x ∈ space M. f x = 0} ∈ sets M and ae2: AE x in M. x ∉ N
  using f by (auto simp: subset_eq zero_less_iff_neq_zero intro!: AE_not_in)
  show AE x in density M f. P x
  using ae2
  unfolding eventually_ae_filter[of _ density M f] Bex_def null_sets_density_iff[OF
f]
  by (intro exI[of _ N ∪ {x ∈ space M. f x = 0}] conjI *) (auto elim: eventu-
ally_elim2)
qed

```

lemma *nn_integral_density*:

```

assumes f: f ∈ borel_measurable M
assumes g: g ∈ borel_measurable M
shows integralN (density M f) g = (∫+x. f x * g x ∂M)
using g proof induct
  case (cong u v)

```

```

then show ?case
  apply (subst nn_integral_cong[OF cong(β)])
  apply (simp_all cong: nn_integral_cong)
  done
next
case (set A) then show ?case
  by (simp add: emeasure_density f)
next
case (mult u c)
moreover have  $\bigwedge x. f x * (c * u x) = c * (f x * u x)$  by (simp add: field_simps)
ultimately show ?case
  using f by (simp add: nn_integral_cmult)
next
case (add u v)
then have  $\bigwedge x. f x * (v x + u x) = f x * v x + f x * u x$ 
  by (simp add: distrib_left)
with add f show ?case
  by (auto simp add: nn_integral_add intro!: nn_integral_add[symmetric])
next
case (seq U)
have eq:  $AE\ x\ in\ M. f x * (SUP\ i. U\ i\ x) = (SUP\ i. f x * U\ i\ x)$ 
  by eventually_elim (simp add: SUP_mult_left_ennreal seq)
from seq f show ?case
  apply (simp add: nn_integral_monotone_convergence_SUP image_comp)
  apply (subst nn_integral_cong_AE[OF eq])
  apply (subst nn_integral_monotone_convergence_SUP_AE)
  apply (auto simp: incseq_def le_fun_def intro!: mult_left_mono)
  done
qed

```

lemma density_distr:

```

assumes [measurable]:  $f \in \text{borel\_measurable } N\ X \in \text{measurable } M\ N$ 
shows density (distr M N X)  $f = \text{distr } (\text{density } M\ (\lambda x. f\ (X\ x)))\ N\ X$ 
by (intro measure_eqI)
  (auto simp add: emeasure_density nn_integral_distr emeasure_distr
    split: split_indicator intro!: nn_integral_cong)

```

lemma emeasure_restricted:

```

assumes  $S: S \in \text{sets } M$  and  $X: X \in \text{sets } M$ 
shows emeasure (density M (indicator S))  $X = \text{emeasure } M\ (S \cap X)$ 

```

proof –

```

have emeasure (density M (indicator S))  $X = (\int^{+x}. \text{indicator } S\ x * \text{indicator } X\ x\ \partial M)$ 
  using S X by (simp add: emeasure_density)
also have  $\dots = (\int^{+x}. \text{indicator } (S \cap X)\ x\ \partial M)$ 
  by (auto intro!: nn_integral_cong simp: indicator_def)
also have  $\dots = \text{emeasure } M\ (S \cap X)$ 
  using S X by (simp add: sets.Int)
finally show ?thesis .

```

qed

lemma *measure_restricted*:

$S \in \text{sets } M \implies X \in \text{sets } M \implies \text{measure } (\text{density } M \text{ (indicator } S)) \ X = \text{measure } M \ (S \cap X)$

by (*simp add: emeasure_restricted measure_def*)

lemma (*in finite_measure*) *finite_measure_restricted*:

$S \in \text{sets } M \implies \text{finite_measure } (\text{density } M \text{ (indicator } S))$

by *standard (simp add: emeasure_restricted)*

lemma *emeasure_density_const*:

$A \in \text{sets } M \implies \text{emeasure } (\text{density } M \ (\lambda_. c)) \ A = c * \text{emeasure } M \ A$

by (*auto simp: nn_integral_cmult_indicator emeasure_density*)

lemma *measure_density_const*:

$A \in \text{sets } M \implies c \neq \infty \implies \text{measure } (\text{density } M \ (\lambda_. c)) \ A = \text{enn2real } c * \text{measure } M \ A$

by (*auto simp: emeasure_density_const measure_def enn2real_mult*)

lemma *density_density_eq*:

$f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies$

$\text{density } (\text{density } M \ f) \ g = \text{density } M \ (\lambda x. f \ x * g \ x)$

by (*auto intro!: measure_eqI simp: emeasure_density nn_integral_density ac_simps*)

lemma *distr_density_distr*:

assumes $T: T \in \text{measurable } M \ M'$ **and** $T': T' \in \text{measurable } M' \ M$

and *inv*: $\forall x \in \text{space } M. T' \ (T \ x) = x$

assumes $f: f \in \text{borel_measurable } M'$

shows $\text{distr } (\text{density } (\text{distr } M \ M' \ T) \ f) \ M \ T' = \text{density } M \ (f \circ T)$ (*is ?R = ?L*)

proof (*rule measure_eqI*)

fix A **assume** $A: A \in \text{sets } ?R$

{ **fix** x **assume** $x \in \text{space } M$

with *sets.sets_into_space*[*OF A*]

have *indicator* $(T' - ` A \cap \text{space } M')$ $(T \ x) = (\text{indicator } A \ x :: \text{ennreal})$

using T *inv* **by** (*auto simp: indicator_def measurable_space*) }

with $A \ T \ T' \ f$ **show** $\text{emeasure } ?R \ A = \text{emeasure } ?L \ A$

by (*simp add: measurable_comp emeasure_density emeasure_distr nn_integral_distr measurable_sets cong: nn_integral_cong*)

qed *simp*

lemma *density_density_divide*:

fixes $f \ g :: 'a \Rightarrow \text{real}$

assumes $f: f \in \text{borel_measurable } M \ A \ E \ x \ \text{in } M. 0 \leq f \ x$

assumes $g: g \in \text{borel_measurable } M \ A \ E \ x \ \text{in } M. 0 \leq g \ x$

assumes $ac: A \ E \ x \ \text{in } M. f \ x = 0 \longrightarrow g \ x = 0$

shows $\text{density } (\text{density } M \ f) \ (\lambda x. g \ x / f \ x) = \text{density } M \ g$

proof –

2274

```
have density M g = density M (λx. ennreal (f x) * ennreal (g x / f x))
  using f g ac by (auto intro!: density_cong measurable>If simp: ennreal_mult[symmetric])
then show ?thesis
  using f g by (subst density_density_eq) auto
qed
```

```
lemma density_1: density M (λ_. 1) = M
  by (intro measure_eqI) (auto simp: emeasure_density)
```

```
lemma emeasure_density_add:
  assumes X: X ∈ sets M
  assumes Mf[measurable]: f ∈ borel_measurable M
  assumes Mg[measurable]: g ∈ borel_measurable M
  shows emeasure (density M f) X + emeasure (density M g) X =
    emeasure (density M (λx. f x + g x)) X
  using assms
  apply (subst (1 2 3) emeasure_density, simp_all) []
  apply (subst nn_integral_add[symmetric], simp_all) []
  apply (intro nn_integral_cong, simp split: split_indicator)
  done
```

Point measure

definition *point_measure* :: 'a set \Rightarrow ('a \Rightarrow ennreal) \Rightarrow 'a measure **where**
point_measure A f = density (count_space A) f

```
lemma
  shows space_point_measure: space (point_measure A f) = A
    and sets_point_measure: sets (point_measure A f) = Pow A
  by (auto simp: point_measure_def)
```

```
lemma sets_point_measure_count_space[measurable_cong]: sets (point_measure
A f) = sets (count_space A)
  by (simp add: sets_point_measure)
```

```
lemma measurable_point_measure_eq1[simp]:
  g ∈ measurable (point_measure A f) M  $\longleftrightarrow$  g ∈ A  $\rightarrow$  space M
  unfolding point_measure_def by simp
```

```
lemma measurable_point_measure_eq2_finite[simp]:
  finite A  $\implies$ 
  g ∈ measurable M (point_measure A f)  $\longleftrightarrow$ 
  (g ∈ space M  $\rightarrow$  A  $\wedge$  ( $\forall a \in A. g - \{a\} \cap$  space M  $\in$  sets M))
  unfolding point_measure_def by (simp add: measurable_count_space_eq2)
```

```
lemma simple_function_point_measure[simp]:
  simple_function (point_measure A f) g  $\longleftrightarrow$  finite (g  $\text{'}$  A)
  by (simp add: point_measure_def)
```

lemma *emeasure_point_measure*:

assumes *A*: *finite* $\{a \in X. 0 < f\ a\}$ $X \subseteq A$

shows *emeasure* (*point_measure* *A* *f*) *X* = $(\sum a | a \in X \wedge 0 < f\ a. f\ a)$

proof –

have $\{a. (a \in X \longrightarrow a \in A \wedge 0 < f\ a) \wedge a \in X\} = \{a \in X. 0 < f\ a\}$

using $\langle X \subseteq A \rangle$ **by** *auto*

with *A* **show** *?thesis*

by (*simp* *add*: *emeasure_density nn_integral_count_space point_measure_def indicator_def of_bool_def*)

qed

lemma *emeasure_point_measure_finite*:

finite *A* $\implies X \subseteq A \implies$ *emeasure* (*point_measure* *A* *f*) *X* = $(\sum a \in X. f\ a)$

by (*subst* *emeasure_point_measure*) (*auto* *dest*: *finite_subset intro!*: *sum.mono_neutral_left simp*: *less_le*)

lemma *emeasure_point_measure_finite_if*:

finite *A* \implies *emeasure* (*point_measure* *A* *f*) *X* = (*if* $X \subseteq A$ *then* $\sum a \in X. f\ a$ *else* 0)

by (*simp* *add*: *emeasure_point_measure_finite emeasure_notin_sets sets_point_measure*)

lemma *measure_point_measure_finite_if*:

assumes *finite* *A* **and** $\bigwedge x. x \in A \implies f\ x \geq 0$

shows *measure* (*point_measure* *A* *f*) *X* = (*if* $X \subseteq A$ *then* $\sum a \in X. f\ a$ *else* 0)

by (*simp* *add*: *Sigma_Algebra.measure_def* *assms* *emeasure_point_measure_finite_if subset_eq sum_nonneg*)

lemma *emeasure_point_measure_finite2*:

$X \subseteq A \implies$ *finite* *X* \implies *emeasure* (*point_measure* *A* *f*) *X* = $(\sum a \in X. f\ a)$

by (*subst* *emeasure_point_measure*)

(*auto* *dest*: *finite_subset intro!*: *sum.mono_neutral_left simp*: *less_le*)

lemma *null_sets_point_measure_iff*:

$X \in$ *null_sets* (*point_measure* *A* *f*) $\iff X \subseteq A \wedge (\forall x \in X. f\ x = 0)$

by (*auto* *simp*: *AE_count_space null_sets_density_iff point_measure_def*)

lemma *AE_point_measure*:

(*AE* *x* *in* *point_measure* *A* *f*. *P* *x*) $\iff (\forall x \in A. 0 < f\ x \longrightarrow P\ x)$

unfolding *point_measure_def*

by (*subst* *AE_density*) (*auto* *simp*: *AE_density AE_count_space point_measure_def*)

lemma *nn_integral_point_measure*:

finite $\{a \in A. 0 < f\ a \wedge 0 < g\ a\} \implies$

integral^N (*point_measure* *A* *f*) *g* = $(\sum a | a \in A \wedge 0 < f\ a \wedge 0 < g\ a. f\ a * g\ a)$

unfolding *point_measure_def*

by (*subst* *nn_integral_density*)

(*simp_all* *add*: *nn_integral_density nn_integral_count_space ennreal_zero_less_mult_iff*)

lemma *nn_integral_point_measure_finite*:

$finite\ A \implies integral^N\ (point_measure\ A\ f)\ g = (\sum\ a \in A.\ f\ a * g\ a)$
by (*subst nn_integral_point_measure*) (*auto intro!: sum.mono_neutral_left simp: less_le*)

Uniform measure

definition *uniform_measure* $M\ A = density\ M\ (\lambda x.\ indicator\ A\ x / emeasure\ M\ A)$

lemma

shows *sets_uniform_measure*[*simp, measurable_cong*]: *sets* (*uniform_measure* $M\ A$) = *sets* M
and *space_uniform_measure*[*simp*]: *space* (*uniform_measure* $M\ A$) = *space* M
by (*auto simp: uniform_measure_def*)

lemma *emeasure_uniform_measure*[*simp*]:

assumes $A: A \in sets\ M$ **and** $B: B \in sets\ M$

shows *emeasure* (*uniform_measure* $M\ A$) $B = emeasure\ M\ (A \cap B) / emeasure\ M\ A$

proof –

from $A\ B$ **have** *emeasure* (*uniform_measure* $M\ A$) $B = (\int^{+x} (1 / emeasure\ M\ A) * indicator\ (A \cap B)\ x\ \partial M)$

by (*auto simp add: uniform_measure_def emeasure_density divide_ennreal_def split: split_indicator intro!: nn_integral_cong*)

also have $\dots = emeasure\ M\ (A \cap B) / emeasure\ M\ A$

using $A\ B$

by (*subst nn_integral_cmult_indicator*) (*simp_all add: sets.Int divide_ennreal_def mult.commute*)

finally show *?thesis* .

qed

lemma *measure_uniform_measure*[*simp*]:

assumes $A: emeasure\ M\ A \neq 0$ $emeasure\ M\ A \neq \infty$ **and** $B: B \in sets\ M$

shows *measure* (*uniform_measure* $M\ A$) $B = measure\ M\ (A \cap B) / measure\ M\ A$

using *emeasure_uniform_measure*[*OF emeasure_neq_0_sets* [*OF* $A(1)$]] B A

by (*cases emeasure\ M\ A emeasure\ M\ (A \cap B) rule: ennreal2_cases*)

(*simp_all add: measure_def divide_ennreal top_ennreal.rep_eq top_ereal_def ennreal_top_divide*)

lemma *AE_uniform_measureI*:

$A \in sets\ M \implies (AE\ x\ in\ M.\ x \in A \implies P\ x) \implies (AE\ x\ in\ uniform_measure\ M\ A.\ P\ x)$

unfolding *uniform_measure_def* **by** (*auto simp: AE_density divide_ennreal_def*)

lemma *emeasure_uniform_measure_1*:

$emeasure\ M\ S \neq 0 \implies emeasure\ M\ S \neq \infty \implies emeasure\ (uniform_measure\ M\ S)\ S = 1$

by (subst emeasure_uniform_measure)
 (simp_all add: emeasure_neq_0_sets emeasure_eq_ennreal_measure divide_ennreal
 zero_less_iff_neq_zero[symmetric])

lemma nn_integral_uniform_measure:

assumes f [measurable]: $f \in \text{borel_measurable } M$ and S [measurable]: $S \in \text{sets } M$
 shows $(\int^+ x. f x \partial \text{uniform_measure } M S) = (\int^+ x. f x * \text{indicator } S x \partial M) /$
 $\text{emeasure } M S$

proof –

{ assume $\text{emeasure } M S = \infty$
 then have ?thesis
 by (simp add: uniform_measure_def nn_integral_density f) }

moreover

{ assume [simp]: $\text{emeasure } M S = 0$
 then have $ae: AE x \text{ in } M. x \notin S$
 using sets.sets_into_space[OF S]
 by (subst AE_iff_measurable[OF _ refl]) (simp_all add: subset_eq cong:
 rev_conj_cong)

from ae have $(\int^+ x. \text{indicator } S x / 0 * f x \partial M) = 0$

by (subst nn_integral_0_iff_AE) auto

moreover from ae have $(\int^+ x. f x * \text{indicator } S x \partial M) = 0$

by (subst nn_integral_0_iff_AE) auto

ultimately have ?thesis

by (simp add: uniform_measure_def nn_integral_density f) }

moreover have $\text{emeasure } M S \neq 0 \implies \text{emeasure } M S \neq \infty \implies ?thesis$

unfolding uniform_measure_def

by (subst nn_integral_density)

(auto simp: ennreal_times_divide f nn_integral_divide[symmetric] mult.commute)

ultimately show ?thesis by blast

qed

lemma AE_uniform_measure:

assumes $\text{emeasure } M A \neq 0$ $\text{emeasure } M A < \infty$

shows $(AE x \text{ in } \text{uniform_measure } M A. P x) \longleftrightarrow (AE x \text{ in } M. x \in A \longrightarrow P x)$

proof –

have $A \in \text{sets } M$

using $\langle \text{emeasure } M A \neq 0 \rangle$ by (metis emeasure_notin_sets)

moreover have $\bigwedge x. 0 < \text{indicator } A x / \text{emeasure } M A \longleftrightarrow x \in A$

using *assms*

by (cases $\text{emeasure } M A$) (auto split: split_indicator simp: ennreal_zero_less_divide)

ultimately show ?thesis

unfolding uniform_measure_def by (simp add: AE_density)

qed

Null measure

lemma null_measure_eq_density: $\text{null_measure } M = \text{density } M (\lambda_. 0)$

by (intro measure_eqI) (simp_all add: emeasure_density)

lemma *nn_integral_null_measure*[simp]: $(\int^+ x. f x \partial \text{null_measure } M) = 0$
by (*auto simp add: nn_integral_def simple_integral_def SUP_constant bot_enreal_def le_fun_def*
intro!: exI[of _ $\lambda x. 0$])

lemma *density_null_measure*[simp]: $\text{density}(\text{null_measure } M) f = \text{null_measure } M$

proof (*intro measure_eqI*)

fix A **show** $\text{emeasure}(\text{density}(\text{null_measure } M) f) A = \text{emeasure}(\text{null_measure } M) A$

by (*simp add: density_def*) (*simp only: null_measure_def[symmetric] emeasure_null_measure*)

qed *simp*

Uniform count measure

definition *uniform_count_measure* $A = \text{point_measure } A (\lambda x. 1 / \text{card } A)$

lemma

shows *space_uniform_count_measure*: $\text{space}(\text{uniform_count_measure } A) = A$

and *sets_uniform_count_measure*: $\text{sets}(\text{uniform_count_measure } A) = \text{Pow } A$

unfolding *uniform_count_measure_def* **by** (*auto simp: space_point_measure sets_point_measure*)

lemma *sets_uniform_count_measure_count_space*[*measurable_cong*]:

$\text{sets}(\text{uniform_count_measure } A) = \text{sets}(\text{count_space } A)$

by (*simp add: sets_uniform_count_measure*)

lemma *emeasure_uniform_count_measure*:

$\text{finite } A \implies X \subseteq A \implies \text{emeasure}(\text{uniform_count_measure } A) X = \text{card } X / \text{card } A$

by (*simp add: emeasure_point_measure_finite uniform_count_measure_def divide_inverse ennreal_mult*

ennreal_of_nat_eq_real_of_nat)

lemma *emeasure_uniform_count_measure_if*:

$\text{finite } A \implies \text{emeasure}(\text{uniform_count_measure } A) X = (\text{if } X \subseteq A \text{ then } \text{card } X / \text{card } A \text{ else } 0)$

by (*simp add: emeasure_notin_sets emeasure_uniform_count_measure sets_uniform_count_measure*)

lemma *measure_uniform_count_measure*:

$\text{finite } A \implies X \subseteq A \implies \text{measure}(\text{uniform_count_measure } A) X = \text{card } X / \text{card } A$

by (*simp add: emeasure_point_measure_finite uniform_count_measure_def measure_def enn2real_mult*)

lemma *measure_uniform_count_measure_if*:

$\text{finite } A \implies \text{measure}(\text{uniform_count_measure } A) X = (\text{if } X \subseteq A \text{ then } \text{card } X$

```

/ card A else 0)
by (simp add: measure_uniform_count_measure measure_notin_sets sets_uniform_count_measure)

```

```

lemma space_uniform_count_measure_empty_iff [simp]:
  space (uniform_count_measure X) = {}  $\longleftrightarrow$  X = {}
by(simp add: space_uniform_count_measure)

```

```

lemma sets_uniform_count_measure_eq_UNIV [simp]:
  sets (uniform_count_measure UNIV) = UNIV  $\longleftrightarrow$  True
  UNIV = sets (uniform_count_measure UNIV)  $\longleftrightarrow$  True
by(simp_all add: sets_uniform_count_measure)

```

Scaled measure

```

lemma nn_integral_scale_measure:
  assumes f: f  $\in$  borel_measurable M
  shows nn_integral (scale_measure r M) f = r * nn_integral M f
  using f
proof induction
  case (cong f g)
  thus ?case
  by(simp add: cong.hyps space_scale_measure cong: nn_integral_cong_simp)
next
  case (mult f c)
  thus ?case
  by(simp add: nn_integral_cmult max_def mult.assoc mult.left_commute)
next
  case (add f g)
  thus ?case
  by(simp add: nn_integral_add distrib_left)
next
  case (seq U)
  thus ?case
  by(simp add: nn_integral_monotone_convergence_SUP SUP_mult_left_enreal
  image_comp)
qed simp
end

```

7.6 Binary Product Measure

```

theory Binary_Product_Measure
imports Nonnegative_Lebesgue_Integration
begin

```

```

lemma Pair_vimage_times[simp]: Pair x -' (A  $\times$  B) = (if x  $\in$  A then B else {})
  by auto

```

```

lemma rev_Pair_vimage_times[simp]: ( $\lambda x. (x, y)$ ) -' (A  $\times$  B) = (if y  $\in$  B then

```

A else $\{\}$)
 by *auto*

7.6.1 Binary products

definition *pair_measure* (**infixr** \otimes_M 80) **where**

$A \otimes_M B = \text{measure_of } (\text{space } A \times \text{space } B)$
 $\{a \times b \mid a \in \text{sets } A \wedge b \in \text{sets } B\}$
 $(\lambda X. \int^+ x. (\int^+ y. \text{indicator } X (x,y) \partial B) \partial A)$

lemma *pair_measure_closed*: $\{a \times b \mid a \in \text{sets } A \wedge b \in \text{sets } B\} \subseteq \text{Pow } (\text{space } A \times \text{space } B)$

using *sets.space_closed*[of A] *sets.space_closed*[of B] **by** *auto*

lemma *space_pair_measure*:

$\text{space } (A \otimes_M B) = \text{space } A \times \text{space } B$

unfolding *pair_measure_def* **using** *pair_measure_closed*[of A B]

by (*rule space_measure_of*)

lemma *SIGMA_Collect_eq*: $(\text{SIGMA } x:\text{space } M. \{y \in \text{space } N. P \ x \ y\}) = \{x \in \text{space } (M \otimes_M N). P \ (\text{fst } x) \ (\text{snd } x)\}$

by (*auto simp: space_pair_measure*)

lemma *sets_pair_measure*:

$\text{sets } (A \otimes_M B) = \text{sigma_sets } (\text{space } A \times \text{space } B) \{a \times b \mid a \in \text{sets } A \wedge b \in \text{sets } B\}$

unfolding *pair_measure_def* **using** *pair_measure_closed*[of A B]

by (*rule sets_measure_of*)

lemma *sets_pair_measure_cong*[*measurable_cong*, *cong*]:

$\text{sets } M1 = \text{sets } M1' \implies \text{sets } M2 = \text{sets } M2' \implies \text{sets } (M1 \otimes_M M2) = \text{sets } (M1' \otimes_M M2')$

unfolding *sets_pair_measure* **by** (*simp cong: sets_eq_imp_space_eq*)

lemma *pair_measureI*[*intro*, *simp*, *measurable*]:

$x \in \text{sets } A \implies y \in \text{sets } B \implies x \times y \in \text{sets } (A \otimes_M B)$

by (*auto simp: sets_pair_measure*)

lemma *sets_Pair*: $\{x\} \in \text{sets } M1 \implies \{y\} \in \text{sets } M2 \implies \{(x, y)\} \in \text{sets } (M1 \otimes_M M2)$

using *pair_measureI*[of $\{x\}$ $M1$ $\{y\}$ $M2$] **by** *simp*

lemma *measurable_pair_measureI*:

assumes 1: $f \in \text{space } M \rightarrow \text{space } M1 \times \text{space } M2$

assumes 2: $\bigwedge A \ B. A \in \text{sets } M1 \implies B \in \text{sets } M2 \implies f \ -' (A \times B) \cap \text{space } M \in \text{sets } M$

shows $f \in \text{measurable } M (M1 \otimes_M M2)$

unfolding *pair_measure_def* **using** 1 2

by (*intro measurable_measure_of*) (*auto dest: sets_sets_into_space*)

lemma *measurable_split_replace*[*measurable (raw)*]:
 $(\lambda x. f x (fst (g x)) (snd (g x))) \in measurable\ M\ N \implies (\lambda x. case_prod\ (f\ x)\ (g\ x)) \in measurable\ M\ N$
unfolding *split_beta'* .

lemma *measurable_Pair*[*measurable (raw)*]:
assumes *f*: $f \in measurable\ M\ M1$ **and** *g*: $g \in measurable\ M\ M2$
shows $(\lambda x. (f\ x, g\ x)) \in measurable\ M\ (M1 \otimes_M M2)$
proof (*rule measurable_pair_measureI*)
show $(\lambda x. (f\ x, g\ x)) \in space\ M \rightarrow space\ M1 \times space\ M2$
using *f g* **by** (*auto simp: measurable_def*)
fix *A B* **assume** ***: $A \in sets\ M1\ B \in sets\ M2$
have $(\lambda x. (f\ x, g\ x)) -' (A \times B) \cap space\ M = (f -' A \cap space\ M) \cap (g -' B \cap space\ M)$
by *auto*
also have $\dots \in sets\ M$
by (*rule sets.Int*) (*auto intro!: measurable_sets * f g*)
finally show $(\lambda x. (f\ x, g\ x)) -' (A \times B) \cap space\ M \in sets\ M$.
qed

lemma *measurable_fst*[*intro!, simp, measurable*]: $fst \in measurable\ (M1 \otimes_M M2)\ M1$
by (*auto simp: fst_vimage_eq_Times space_pair_measure sets.sets_into_space Times_Int_Times measurable_def*)

lemma *measurable_snd*[*intro!, simp, measurable*]: $snd \in measurable\ (M1 \otimes_M M2)\ M2$
by (*auto simp: snd_vimage_eq_Times space_pair_measure sets.sets_into_space Times_Int_Times measurable_def*)

lemma *measurable_Pair_compose_split*[*measurable_dest*]:
assumes *f*: $case_prod\ f \in measurable\ (M1 \otimes_M M2)\ N$
assumes *g*: $g \in measurable\ M\ M1$ **and** *h*: $h \in measurable\ M\ M2$
shows $(\lambda x. f\ (g\ x)\ (h\ x)) \in measurable\ M\ N$
using *measurable_compose[OF measurable_Pair f, OF g h]* **by** *simp*

lemma *measurable_Pair1_compose*[*measurable_dest*]:
assumes *f*: $(\lambda x. (f\ x, g\ x)) \in measurable\ M\ (M1 \otimes_M M2)$
assumes [*measurable*]: $h \in measurable\ N\ M$
shows $(\lambda x. f\ (h\ x)) \in measurable\ N\ M1$
using *measurable_compose[OF f measurable_fst]* **by** *simp*

lemma *measurable_Pair2_compose*[*measurable_dest*]:
assumes *f*: $(\lambda x. (f\ x, g\ x)) \in measurable\ M\ (M1 \otimes_M M2)$
assumes [*measurable*]: $h \in measurable\ N\ M$
shows $(\lambda x. g\ (h\ x)) \in measurable\ N\ M2$

using *measurable_compose*[*OF f measurable_snd*] **by** *simp*

lemma *measurable_pair*:

assumes $(fst \circ f) \in measurable\ M\ M1$ $(snd \circ f) \in measurable\ M\ M2$

shows $f \in measurable\ M\ (M1 \otimes_M M2)$

using *measurable_Pair*[*OF assms*] **by** *simp*

lemma

assumes $f[measurable]: f \in measurable\ M\ (N \otimes_M P)$

shows *measurable_fst'*: $(\lambda x. fst\ (f\ x)) \in measurable\ M\ N$

and *measurable_snd'*: $(\lambda x. snd\ (f\ x)) \in measurable\ M\ P$

by *simp_all*

lemma

assumes $f[measurable]: f \in measurable\ M\ N$

shows *measurable_fst''*: $(\lambda x. f\ (fst\ x)) \in measurable\ (M \otimes_M P)\ N$

and *measurable_snd''*: $(\lambda x. f\ (snd\ x)) \in measurable\ (P \otimes_M M)\ N$

by *simp_all*

lemma *sets_pair_in_sets*:

assumes $\bigwedge a\ b. a \in sets\ A \implies b \in sets\ B \implies a \times b \in sets\ N$

shows $sets\ (A \otimes_M B) \subseteq sets\ N$

unfolding *sets_pair_measure*

by (*intro sets.sigma_sets_subset'*) (*auto intro!: assms*)

lemma *sets_pair_eq_sets_fst_snd*:

$sets\ (A \otimes_M B) = sets\ (Sup\ \{vimage_algebra\ (space\ A \times space\ B)\ fst\ A,\ vimage_algebra\ (space\ A \times space\ B)\ snd\ B\})$

(**is** $?P = sets\ (Sup\ \{?fst,\ ?snd\})$)

proof –

{ **fix** $a\ b$ **assume** $ab: a \in sets\ A\ b \in sets\ B$

then have $a \times b = (fst\ -' a \cap (space\ A \times space\ B)) \cap (snd\ -' b \cap (space\ A \times space\ B))$

by (*auto dest: sets.sets_into_space*)

also have $\dots \in sets\ (Sup\ \{?fst,\ ?snd\})$

apply (*rule sets.Int*)

apply (*rule in_sets_Sup*)

apply *auto* []

apply (*rule insertI1*)

apply (*auto intro: ab in_vimage_algebra*) []

apply (*rule in_sets_Sup*)

apply *auto* []

apply (*rule insertI2*)

apply (*auto intro: ab in_vimage_algebra*)

done

finally have $a \times b \in sets\ (Sup\ \{?fst,\ ?snd\})$. }

moreover have $sets\ ?fst \subseteq sets\ (A \otimes_M B)$

by (*rule sets_image_in_sets*) (*auto simp: space_pair_measure[symmetric]*)

moreover have $sets\ ?snd \subseteq sets\ (A \otimes_M B)$

```

  by (rule sets_image_in_sets) (auto simp: space_pair_measure)
ultimately show ?thesis
  apply (intro antisym[of sets A for A] sets_Sup_in_sets sets_pair_in_sets)
  apply simp
  apply simp
  apply simp
  apply (elim disjE)
  apply (simp add: space_pair_measure)
  apply (simp add: space_pair_measure)
  apply (auto simp add: space_pair_measure)
done
qed

lemma measurable_pair_iff:
   $f \in \text{measurable } M (M1 \otimes_M M2) \longleftrightarrow (fst \circ f) \in \text{measurable } M M1 \wedge (snd \circ f) \in \text{measurable } M M2$ 
  by (auto intro: measurable_pair[of f M M1 M2])

lemma measurable_split_conv:
   $(\lambda(x, y). f x y) \in \text{measurable } A B \longleftrightarrow (\lambda x. f (fst x) (snd x)) \in \text{measurable } A B$ 
  by (intro arg_cong2[where f=( $\epsilon$ )] auto)

lemma measurable_pair_swap':  $(\lambda(x,y). (y, x)) \in \text{measurable } (M1 \otimes_M M2) (M2 \otimes_M M1)$ 
  by (auto intro!: measurable_Pair simp: measurable_split_conv)

lemma measurable_pair_swap:
  assumes  $f: f \in \text{measurable } (M1 \otimes_M M2) M$  shows  $(\lambda(x,y). f (y, x)) \in \text{measurable } (M2 \otimes_M M1) M$ 
  using measurable_comp[OF measurable_Pair f] by (auto simp: measurable_split_conv comp_def)

lemma measurable_pair_swap_iff:
   $f \in \text{measurable } (M2 \otimes_M M1) M \longleftrightarrow (\lambda(x,y). f (y,x)) \in \text{measurable } (M1 \otimes_M M2) M$ 
  by (auto dest: measurable_pair_swap)

lemma measurable_Pair1':  $x \in \text{space } M1 \implies \text{Pair } x \in \text{measurable } M2 (M1 \otimes_M M2)$ 
  by simp

lemma sets_Pair1[measurable (raw)]:
  assumes  $A: A \in \text{sets } (M1 \otimes_M M2)$  shows  $\text{Pair } x - ' A \in \text{sets } M2$ 
proof -
  have  $\text{Pair } x - ' A = (\text{if } x \in \text{space } M1 \text{ then } \text{Pair } x - ' A \cap \text{space } M2 \text{ else } \{\})$ 
  using A[THEN sets.sets_into_space] by (auto simp: space_pair_measure)
  also have  $\dots \in \text{sets } M2$ 
  using A by (auto simp add: measurable_Pair1' intro!: measurable_sets split: if_split_asm)

```

finally show *?thesis* .
qed

lemma *measurable_Pair2'*: $y \in \text{space } M2 \implies (\lambda x. (x, y)) \in \text{measurable } M1 (M1 \otimes_M M2)$
by (*auto intro!*: *measurable_Pair*)

lemma *sets_Pair2*: **assumes** $A: A \in \text{sets } (M1 \otimes_M M2)$ **shows** $(\lambda x. (x, y)) - ' A \in \text{sets } M1$

proof –

have $(\lambda x. (x, y)) - ' A = (\text{if } y \in \text{space } M2 \text{ then } (\lambda x. (x, y)) - ' A \cap \text{space } M1 \text{ else } \{\})$

using $A[\text{THEN sets.sets_into_space}]$ **by** (*auto simp*: *space_pair_measure*)

also have $\dots \in \text{sets } M1$

using A **by** (*auto simp add*: *measurable_Pair2'* *intro!*: *measurable_sets split*: *if_split_asm*)

finally show *?thesis* .

qed

lemma *measurable_Pair2*:

assumes $f: f \in \text{measurable } (M1 \otimes_M M2) M$ **and** $x: x \in \text{space } M1$

shows $(\lambda y. f (x, y)) \in \text{measurable } M2 M$

using *measurable_comp*[*OF measurable_Pair1' f, OF x*]

by (*simp add*: *comp_def*)

lemma *measurable_Pair1*:

assumes $f: f \in \text{measurable } (M1 \otimes_M M2) M$ **and** $y: y \in \text{space } M2$

shows $(\lambda x. f (x, y)) \in \text{measurable } M1 M$

using *measurable_comp*[*OF measurable_Pair2' f, OF y*]

by (*simp add*: *comp_def*)

lemma *Int_stable_pair_measure_generator*: *Int_stable* $\{a \times b \mid a \in \text{sets } A \wedge b \in \text{sets } B\}$

unfolding *Int_stable_def*

by safe (*auto simp add*: *Times_Int_Times*)

lemma (*in finite_measure*) *finite_measure_cut_measurable*:

assumes [*measurable*]: $Q \in \text{sets } (N \otimes_M M)$

shows $(\lambda x. \text{emeasure } M (\text{Pair } x - ' Q)) \in \text{borel_measurable } N$

(*is ?s Q \in _*)

using *Int_stable_pair_measure_generator pair_measure_closed assms*

unfolding *sets_pair_measure*

proof (*induct rule*: *sigma_sets_induct_disjoint*)

case (*compl A*)

with *sets.sets_into_space* **have** $\bigwedge x. \text{emeasure } M (\text{Pair } x - ' ((\text{space } N \times \text{space } M) - A)) =$

(*if x \in \text{space } N \text{ then } \text{emeasure } M (\text{space } M) - ?s A x \text{ else } 0*)

unfolding *sets_pair_measure*[*symmetric*]

by (*auto intro!*: *emeasure_compl simp*: *vimage_Diff sets_Pair1*)


```

with compl sets.top show ?case
  by (auto intro!: measurable>If simp: space_pair_measure)
next
case (union F)
then have  $\bigwedge x. \text{emeasure } M (\text{Pair } x -' (\bigcup i. F i)) = (\sum i. ?s (F i) x)$ 
  by (simp add: suminf_emeasure disjoint_family_on_vimageI subset_eq vimage_UN sets_pair_measure[symmetric])
  with union show ?case
  unfolding sets_pair_measure[symmetric] by simp
qed (auto simp add: if_distrib Int_def[symmetric] intro!: measurable>If)

lemma (in sigma_finite_measure) measurable_emeasure_Pair:
  assumes  $Q: Q \in \text{sets } (N \otimes_M M)$  shows  $(\lambda x. \text{emeasure } M (\text{Pair } x -' Q)) \in \text{borel\_measurable } N$  (is ?s  $Q \in \_$ )
proof -
  obtain  $F :: \text{nat} \Rightarrow 'a \text{ set}$  where  $F:$ 
    range  $F \subseteq \text{sets } M$ 
     $\bigcup (\text{range } F) = \text{space } M$ 
     $\bigwedge i. \text{emeasure } M (F i) \neq \infty$ 
    disjoint_family  $F$  by (blast intro: sigma_finite_disjoint)
  then have  $F\_sets: \bigwedge i. F i \in \text{sets } M$  by auto
  let ?C =  $\lambda x i. F i \cap \text{Pair } x -' Q$ 
  { fix i
    have [simp]:  $\text{space } N \times F i \cap \text{space } N \times \text{space } M = \text{space } N \times F i$ 
      using  $F$  sets.sets_into_space by auto
    let ?R = density  $M$  (indicator  $(F i)$ )
    have finite_measure ?R
      using  $F$  by (intro finite_measureI) (auto simp: emeasure_restricted_subset_eq)
    then have  $(\lambda x. \text{emeasure } ?R (\text{Pair } x -' (\text{space } N \times \text{space } ?R \cap Q))) \in \text{borel\_measurable } N$ 
      by (rule finite_measure.finite_measure_cut_measurable) (auto intro: Q)
    moreover have  $\bigwedge x. \text{emeasure } ?R (\text{Pair } x -' (\text{space } N \times \text{space } ?R \cap Q)) = \text{emeasure } M (F i \cap \text{Pair } x -' (\text{space } N \times \text{space } ?R \cap Q))$ 
      using  $Q$   $F\_sets$  by (intro emeasure_restricted) (auto intro: sets_Pair1)
    moreover have  $\bigwedge x. F i \cap \text{Pair } x -' (\text{space } N \times \text{space } ?R \cap Q) = ?C x i$ 
      using sets.sets_into_space[OF Q] by (auto simp: space_pair_measure)
    ultimately have  $(\lambda x. \text{emeasure } M (?C x i)) \in \text{borel\_measurable } N$ 
      by simp }
  moreover
  { fix x
    have  $(\sum i. \text{emeasure } M (?C x i)) = \text{emeasure } M (\bigcup i. ?C x i)$ 
      proof (intro suminf_emeasure)
        show range  $(?C x) \subseteq \text{sets } M$ 
          using  $F \langle Q \in \text{sets } (N \otimes_M M) \rangle$  by (auto intro!: sets_Pair1)
        have disjoint_family  $F$  using  $F$  by auto
        show disjoint_family  $(?C x)$ 
          by (rule disjoint_family_on_bisimulation[OF  $\langle \text{disjoint\_family } F \rangle$ ]) auto
      qed
  }

```

also have $(\bigcup i. ?C x i) = \text{Pair } x - ' Q$
using $F \text{ sets.sets_into_space}[OF \langle Q \in \text{sets } (N \otimes_M M) \rangle]$
by $(\text{auto simp: space_pair_measure})$
finally have $\text{emeasure } M (\text{Pair } x - ' Q) = (\sum i. \text{emeasure } M (?C x i))$
by simp }
ultimately show $?thesis$ **using** $\langle Q \in \text{sets } (N \otimes_M M) \rangle F_sets$
by auto
qed

lemma $(\text{in } \text{sigma_finite_measure}) \text{ measurable_emeasure}[\text{measurable } (raw)]:$
assumes $\text{space: } \bigwedge x. x \in \text{space } N \implies A x \subseteq \text{space } M$
assumes $A: \{x \in \text{space } (N \otimes_M M). \text{snd } x \in A (\text{fst } x)\} \in \text{sets } (N \otimes_M M)$
shows $(\lambda x. \text{emeasure } M (A x)) \in \text{borel_measurable } N$
proof –
from space have $\bigwedge x. x \in \text{space } N \implies \text{Pair } x - ' \{x \in \text{space } (N \otimes_M M). \text{snd } x \in A (\text{fst } x)\} = A x$
by $(\text{auto simp: space_pair_measure})$
with measurable_emeasure_Pair $[OF A]$ **show** $?thesis$
by $(\text{auto cong: measurable_cong})$
qed

lemma $(\text{in } \text{sigma_finite_measure}) \text{ emeasure_pair_measure:}$
assumes $X \in \text{sets } (N \otimes_M M)$
shows $\text{emeasure } (N \otimes_M M) X = (\int^+ x. \int^+ y. \text{indicator } X (x, y) \partial M \partial N)$
 $(\text{is } _ = ?\mu X)$
proof $(\text{rule emeasure_measure_of}[OF \text{pair_measure_def}])$
show $\text{positive } (\text{sets } (N \otimes_M M)) ?\mu$
by $(\text{auto simp: positive_def})$
have $\text{eq}[simp]: \bigwedge A x y. \text{indicator } A (x, y) = \text{indicator } (\text{Pair } x - ' A) y$
by $(\text{auto simp: indicator_def})$
show $\text{countably_additive } (\text{sets } (N \otimes_M M)) ?\mu$
proof $(\text{rule countably_additiveI})$
fix $F :: \text{nat} \Rightarrow ('b \times 'a) \text{ set}$ **assume** $F: \text{range } F \subseteq \text{sets } (N \otimes_M M) \text{ disjoint_family } F$
from F have $*$: $\bigwedge i. F i \in \text{sets } (N \otimes_M M)$ **by auto**
moreover have $\bigwedge x. \text{disjoint_family } (\lambda i. \text{Pair } x - ' F i)$
by $(\text{intro disjoint_family_on_bisimulation}[OF F(2)]) \text{ auto}$
moreover have $\bigwedge x. \text{range } (\lambda i. \text{Pair } x - ' F i) \subseteq \text{sets } M$
using F by $(\text{auto simp: sets_Pair1})$
ultimately show $(\sum n. ?\mu (F n)) = ?\mu (\bigcup i. F i)$
by $(\text{auto simp add: nn_integral_suminf}[symmetric] \text{vimage_UN suminf_emeasure}$
 $\text{intro!: nn_integral_cong nn_integral_indicator}[symmetric])$
qed
show $\{a \times b \mid a \in \text{sets } N \wedge b \in \text{sets } M\} \subseteq \text{Pow } (\text{space } N \times \text{space } M)$
using $\text{sets.space_closed}[of N] \text{sets.space_closed}[of M]$ **by auto**
qed fact

lemma $(\text{in } \text{sigma_finite_measure}) \text{ emeasure_pair_measure_alt:}$
assumes $X: X \in \text{sets } (N \otimes_M M)$

shows $\text{emeasure } (N \otimes_M M) X = (\int^+ x. \text{emeasure } M (\text{Pair } x -' X) \partial N)$
proof –
have $[\text{simp}]: \bigwedge x y. \text{indicator } X (x, y) = \text{indicator } (\text{Pair } x -' X) y$
by $(\text{auto simp: indicator_def})$
show $?thesis$
using X **by** $(\text{auto intro!: nn_integral_cong simp: emeasure_pair_measure sets_Pair1})$
qed

proposition $(\text{in } \text{sigma_finite_measure}) \text{emeasure_pair_measure_Times}$:
assumes $A: A \in \text{sets } N$ **and** $B: B \in \text{sets } M$
shows $\text{emeasure } (N \otimes_M M) (A \times B) = \text{emeasure } N A * \text{emeasure } M B$
proof –
have $\text{emeasure } (N \otimes_M M) (A \times B) = (\int^+ x. \text{emeasure } M B * \text{indicator } A x \partial N)$
using $A B$ **by** $(\text{auto intro!: nn_integral_cong simp: emeasure_pair_measure_alt})$
also have $\dots = \text{emeasure } M B * \text{emeasure } N A$
using A **by** $(\text{simp add: nn_integral_cmult_indicator})$
finally show $?thesis$
by $(\text{simp add: ac_simps})$
qed

lemma $(\text{in } \text{sigma_finite_measure}) \text{times_in_null_sets1}$ $[\text{intro}]$:
assumes $A \in \text{null_sets } N$ $B \in \text{sets } M$
shows $A \times B \in \text{null_sets } (N \otimes_M M)$
using assms **by** $(\text{simp add: null_sets_def emeasure_pair_measure_Times})$

lemma $(\text{in } \text{sigma_finite_measure}) \text{times_in_null_sets2}$ $[\text{intro}]$:
assumes $A \in \text{sets } N$ $B \in \text{null_sets } M$
shows $A \times B \in \text{null_sets } (N \otimes_M M)$
using assms **by** $(\text{simp add: null_sets_def emeasure_pair_measure_Times})$

7.6.2 Binary products of σ -finite emeasure spaces

locale $\text{pair_sigma_finite} = M1? : \text{sigma_finite_measure } M1 + M2? : \text{sigma_finite_measure } M2$
for $M1 :: 'a \text{ measure}$ **and** $M2 :: 'b \text{ measure}$

lemma $(\text{in } \text{pair_sigma_finite}) \text{measurable_emeasure_Pair1}$:
 $Q \in \text{sets } (M1 \otimes_M M2) \implies (\lambda x. \text{emeasure } M2 (\text{Pair } x -' Q)) \in \text{borel_measurable } M1$
using $M2.\text{measurable_emeasure_Pair}$.

lemma $(\text{in } \text{pair_sigma_finite}) \text{measurable_emeasure_Pair2}$:
assumes $Q: Q \in \text{sets } (M1 \otimes_M M2)$ **shows** $(\lambda y. \text{emeasure } M1 ((\lambda x. (x, y)) -' Q)) \in \text{borel_measurable } M2$
proof –
have $(\lambda(x, y). (y, x)) -' Q \cap \text{space } (M2 \otimes_M M1) \in \text{sets } (M2 \otimes_M M1)$
using $Q \text{measurable_pair_swap'}$ **by** $(\text{auto intro: measurable_sets})$

note $M1.measurable_emeasure_Pair$ [OF this]
moreover have $\bigwedge y. Pair\ y \text{ --}' ((\lambda(x, y). (y, x)) \text{ --}' Q \cap space\ (M2 \otimes_M M1))$
 $= (\lambda x. (x, y)) \text{ --}' Q$
using Q [THEN sets.sets_into_space] **by** (auto simp: space_pair_measure)
ultimately show ?thesis **by** simp
qed

proposition (in pair_sigma_finite) sigma_finite_up_in_pair_measure_generator:

defines $E \equiv \{A \times B \mid A\ B. A \in sets\ M1 \wedge B \in sets\ M2\}$

shows $\exists F::nat \Rightarrow ('a \times 'b)\ set. range\ F \subseteq E \wedge incseq\ F \wedge (\bigcup i. F\ i) = space\ M1 \times space\ M2 \wedge$

$(\forall i. emeasure\ (M1 \otimes_M M2)\ (F\ i) \neq \infty)$

proof –

obtain $F1$ **where** $F1: range\ F1 \subseteq sets\ M1$

$\bigcup (range\ F1) = space\ M1$

$\bigwedge i. emeasure\ M1\ (F1\ i) \neq \infty$

$incseq\ F1$

by (rule $M1.sigma_finite_incseq$) blast

obtain $F2$ **where** $F2: range\ F2 \subseteq sets\ M2$

$\bigcup (range\ F2) = space\ M2$

$\bigwedge i. emeasure\ M2\ (F2\ i) \neq \infty$

$incseq\ F2$

by (rule $M2.sigma_finite_incseq$) blast

from $F1\ F2$ **have** $space: space\ M1 = (\bigcup i. F1\ i)\ space\ M2 = (\bigcup i. F2\ i)$ **by** auto

let $?F = \lambda i. F1\ i \times F2\ i$

show ?thesis

proof (intro exI[of _ ?F] conjI allI)

show $range\ ?F \subseteq E$ **using** $F1\ F2$ **by** (auto simp: E_def) (metis range_subsetD)

next

have $space\ M1 \times space\ M2 \subseteq (\bigcup i. ?F\ i)$

proof (intro subsetI)

fix x **assume** $x \in space\ M1 \times space\ M2$

then obtain $i\ j$ **where** $fst\ x \in F1\ i\ snd\ x \in F2\ j$

by (auto simp: space)

then have $fst\ x \in F1\ (max\ i\ j)\ snd\ x \in F2\ (max\ j\ i)$

using $\langle incseq\ F1 \rangle\ \langle incseq\ F2 \rangle$ **unfolding** $incseq_def$

by (force split: split_max)+

then have $(fst\ x, snd\ x) \in F1\ (max\ i\ j) \times F2\ (max\ i\ j)$

by (intro SigmaI) (auto simp add: max_commute)

then show $x \in (\bigcup i. ?F\ i)$ **by** auto

qed

then show $(\bigcup i. ?F\ i) = space\ M1 \times space\ M2$

using space **by** (auto simp: space)

next

fix i **show** $incseq\ (\lambda i. F1\ i \times F2\ i)$

using $\langle incseq\ F1 \rangle\ \langle incseq\ F2 \rangle$ **unfolding** $incseq_Suc_iff$ **by** auto

next

fix i

from $F1\ F2$ **have** $F1\ i \in \text{sets } M1\ F2\ i \in \text{sets } M2$ **by** *auto*
with $F1\ F2$ **show** $\text{emeasure } (M1 \otimes_M M2) (F1\ i \times F2\ i) \neq \infty$
by (*auto simp add: emeasure_pair_measure_Times ennreal_mult_eq_top_iff*)
qed
qed

sublocale $\text{pair_sigma_finite} \subseteq P?$: $\text{sigma_finite_measure } M1 \otimes_M M2$

proof

obtain $F1 :: 'a\ \text{set set}$ **and** $F2 :: 'b\ \text{set set}$ **where**
 $\text{countable } F1 \wedge F1 \subseteq \text{sets } M1 \wedge \bigcup F1 = \text{space } M1 \wedge (\forall a \in F1. \text{emeasure } M1\ a \neq \infty)$
 $\text{countable } F2 \wedge F2 \subseteq \text{sets } M2 \wedge \bigcup F2 = \text{space } M2 \wedge (\forall a \in F2. \text{emeasure } M2\ a \neq \infty)$
using $M1.\text{sigma_finite_countable } M2.\text{sigma_finite_countable}$ **by** *auto*
then show
 $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (M1 \otimes_M M2) \wedge \bigcup A = \text{space } (M1 \otimes_M M2) \wedge$
 $(\forall a \in A. \text{emeasure } (M1 \otimes_M M2)\ a \neq \infty)$
by (*intro exI[of_ $(\lambda(a, b). a \times b)$ ' $(F1 \times F2)$] conjI*)
(auto simp: M2.emeasure_pair_measure_Times space_pair_measure_set_eq_iff subset_eq ennreal_mult_eq_top_iff)
qed

lemma $\text{sigma_finite_pair_measure}$:

assumes A : $\text{sigma_finite_measure } A$ **and** B : $\text{sigma_finite_measure } B$
shows $\text{sigma_finite_measure } (A \otimes_M B)$
proof –
interpret A : $\text{sigma_finite_measure } A$ **by** *fact*
interpret B : $\text{sigma_finite_measure } B$ **by** *fact*
interpret AB : $\text{pair_sigma_finite } A\ B$..
show *?thesis* ..
qed

lemma sets_pair_swap :

assumes $A \in \text{sets } (M1 \otimes_M M2)$
shows $(\lambda(x, y). (y, x)) \cdot A \in \text{sets } (M2 \otimes_M M1)$
using $\text{measurable_pair_swap' assms}$ **by** (*rule measurable_sets*)

lemma (**in** pair_sigma_finite) distr_pair_swap :

$M1 \otimes_M M2 = \text{distr } (M2 \otimes_M M1) (M1 \otimes_M M2) (\lambda(x, y). (y, x))$ (**is** $?P = ?D$)

proof –

let $?E = \{a \times b \mid a \in \text{sets } M1 \wedge b \in \text{sets } M2\}$
obtain $F :: \text{nat} \Rightarrow ('a \times 'b)\ \text{set}$ **where** $F: \text{range } F \subseteq ?E$
 $\text{incseq } F \cup (\text{range } F) = \text{space } M1 \times \text{space } M2 \forall i. \text{emeasure } (M1 \otimes_M M2) (F\ i) \neq \infty$
using $\text{sigma_finite_up_in_pair_measure_generator}$ **by** *auto*
show *?thesis*
proof (*rule measure_eqI_generator_eq[OF Int_stable_pair_measure_generator[of $M1\ M2$]]*)

```

show ?E ⊆ Pow (space ?P)
  using sets.space_closed[of M1] sets.space_closed[of M2] by (auto simp:
space_pair_measure)
show sets ?P = sigma_sets (space ?P) ?E
  by (simp add: sets_pair_measure space_pair_measure)
then show sets ?D = sigma_sets (space ?P) ?E
  by simp
from F show range F ⊆ ?E (⋃ i. F i) = space ?P ∧ i. emeasure ?P (F i) ≠
∞
  by (auto simp: space_pair_measure)
next
fix X assume X ∈ ?E
then obtain A B where X[simp]: X = A × B and A: A ∈ sets M1 and B:
B ∈ sets M2 by auto
  have (λ(y, x). (x, y)) -' X ∩ space (M2 ⊗M M1) = B × A
  using sets.sets_into_space[OF A] sets.sets_into_space[OF B] by (auto simp:
space_pair_measure)
  with A B show emeasure (M1 ⊗M M2) X = emeasure ?D X
  by (simp add: M2.emeasure_pair_measure_Times M1.emeasure_pair_measure_Times
emeasure_distr
measurable_pair_swap' ac_simps)

```

qed
qed

```

lemma (in pair_sigma_finite) emeasure_pair_measure_alt2:
assumes A: A ∈ sets (M1 ⊗M M2)
shows emeasure (M1 ⊗M M2) A = (∫+ y. emeasure M1 ((λx. (x, y)) -' A)
∂M2)
  (is _ = ?ν A)
proof -
  have [simp]: ∧y. (Pair y -' ((λ(x, y). (y, x)) -' A ∩ space (M2 ⊗M M1))) =
(λx. (x, y)) -' A
  using sets.sets_into_space[OF A] by (auto simp: space_pair_measure)
  show ?thesis using A
  by (subst distr_pair_swap)
  (simp_all del: vimage_Int add: measurable_sets[OF measurable_pair_swap']
M1.emeasure_pair_measure_alt emeasure_distr[OF measurable_pair_swap' A])

```

```

lemma (in pair_sigma_finite) AE_pair:
assumes AE x in (M1 ⊗M M2). Q x
shows AE x in M1. (AE y in M2. Q (x, y))
proof -
obtain N where N: N ∈ sets (M1 ⊗M M2) emeasure (M1 ⊗M M2) N = 0
{x ∈ space (M1 ⊗M M2). ¬ Q x} ⊆ N
  using assms unfolding eventually_ae_filter by auto
show ?thesis
proof (rule AE_I)

```

```

from  $N$  measurable_emeasure_Pair1[OF  $\langle N \in \text{sets } (M1 \otimes_M M2) \rangle$ ]
show emeasure  $M1$   $\{x \in \text{space } M1. \text{emeasure } M2 (\text{Pair } x -' N) \neq 0\} = 0$ 
  by (auto simp:  $M2.\text{emeasure\_pair\_measure\_alt\_nn\_integral\_0\_iff}$ )
show  $\{x \in \text{space } M1. \text{emeasure } M2 (\text{Pair } x -' N) \neq 0\} \in \text{sets } M1$ 
by (intro borel_measurable_eq measurable_emeasure_Pair1  $N \text{ sets.sets\_Collect\_neg}$ 
 $N$ ) simp
{ fix  $x$  assume  $x \in \text{space } M1$  emeasure  $M2$   $(\text{Pair } x -' N) = 0$ 
  have  $AE$   $y$  in  $M2$ .  $Q(x, y)$ 
  proof (rule  $AE\_I$ )
    show emeasure  $M2$   $(\text{Pair } x -' N) = 0$  by fact
    show  $\text{Pair } x -' N \in \text{sets } M2$  using  $N(I)$  by (rule sets_Pair1)
    show  $\{y \in \text{space } M2. \neg Q(x, y)\} \subseteq \text{Pair } x -' N$ 
      using  $N \langle x \in \text{space } M1 \rangle$  unfolding space_pair_measure by auto
    qed }
  then show  $\{x \in \text{space } M1. \neg (AE y \text{ in } M2. Q(x, y))\} \subseteq \{x \in \text{space } M1. \text{emeasure } M2 (\text{Pair } x -' N) \neq 0\}$ 
    by auto
  qed
qed

```

```

lemma (in pair_sigma_finite)  $AE\_pair\_measure$ :
  assumes  $\{x \in \text{space } (M1 \otimes_M M2). P x\} \in \text{sets } (M1 \otimes_M M2)$ 
  assumes  $ae$ :  $AE$   $x$  in  $M1$ .  $AE$   $y$  in  $M2$ .  $P(x, y)$ 
  shows  $AE$   $x$  in  $M1 \otimes_M M2$ .  $P x$ 
proof (subst  $AE\_iff\_measurable[OF\_refl]$ )
  show  $\{x \in \text{space } (M1 \otimes_M M2). \neg P x\} \in \text{sets } (M1 \otimes_M M2)$ 
    by (rule sets.sets_Collect) fact
  then have emeasure  $(M1 \otimes_M M2)$   $\{x \in \text{space } (M1 \otimes_M M2). \neg P x\} =$ 
     $(\int^+ x. \int^+ y. \text{indicator } \{x \in \text{space } (M1 \otimes_M M2). \neg P x\} (x, y) \partial M2 \partial M1)$ 
    by (simp add:  $M2.\text{emeasure\_pair\_measure}$ )
  also have  $\dots = (\int^+ x. \int^+ y. 0 \partial M2 \partial M1)$ 
    using  $ae$ 
    apply (safe intro!:  $nn\_integral\_cong\_AE$ )
    apply (intro  $AE\_I2$ )
    apply (safe intro!:  $nn\_integral\_cong\_AE$ )
    apply auto
    done
  finally show emeasure  $(M1 \otimes_M M2)$   $\{x \in \text{space } (M1 \otimes_M M2). \neg P x\} = 0$ 
by simp
qed

```

```

lemma (in pair_sigma_finite)  $AE\_pair\_iff$ :
   $\{x \in \text{space } (M1 \otimes_M M2). P(\text{fst } x)(\text{snd } x)\} \in \text{sets } (M1 \otimes_M M2) \implies$ 
   $(AE x \text{ in } M1. AE y \text{ in } M2. P x y) \longleftrightarrow (AE x \text{ in } (M1 \otimes_M M2). P(\text{fst } x)$ 
 $(\text{snd } x))$ 
  using  $AE\_pair$ [of  $\lambda x. P(\text{fst } x)(\text{snd } x)$ ]  $AE\_pair\_measure$ [of  $\lambda x. P(\text{fst } x)(\text{snd } x)$ ] by auto

```

```

lemma (in pair_sigma_finite)  $AE\_commute$ :

```

assumes $P: \{x \in \text{space } (M1 \otimes_M M2). P (\text{fst } x) (\text{snd } x)\} \in \text{sets } (M1 \otimes_M M2)$
shows $(\text{AE } x \text{ in } M1. \text{AE } y \text{ in } M2. P x y) \longleftrightarrow (\text{AE } y \text{ in } M2. \text{AE } x \text{ in } M1. P x y)$
proof –
interpret $Q: \text{pair_sigma_finite } M2 \ M1 \ ..$
have $[\text{simp}]: \bigwedge x. (\text{fst } (\text{case } x \text{ of } (x, y) \Rightarrow (y, x))) = \text{snd } x \wedge x. (\text{snd } (\text{case } x \text{ of } (x, y) \Rightarrow (y, x))) = \text{fst } x$
by *auto*
have $\{x \in \text{space } (M2 \otimes_M M1). P (\text{snd } x) (\text{fst } x)\} =$
 $(\lambda(x, y). (y, x)) -' \{x \in \text{space } (M1 \otimes_M M2). P (\text{fst } x) (\text{snd } x)\} \cap \text{space } (M2 \otimes_M M1)$
by $(\text{auto simp: space_pair_measure})$
also have $\dots \in \text{sets } (M2 \otimes_M M1)$
by $(\text{intro sets_pair_swap } P)$
finally show *?thesis*
apply $(\text{subst AE_pair_iff}[OF P])$
apply $(\text{subst distr_pair_swap})$
apply $(\text{subst AE_distr_iff}[OF \text{measurable_pair_swap}' P])$
apply $(\text{subst Q.AE_pair_iff})$
apply *simp_all*
done
qed

7.6.3 Fubinis theorem

lemma *measurable_compose_Pair1*:

$x \in \text{space } M1 \implies g \in \text{measurable } (M1 \otimes_M M2) \ L \implies (\lambda y. g (x, y)) \in \text{measurable } M2 \ L$
by *simp*

lemma $(\text{in } \text{sigma_finite_measure}) \ \text{borel_measurable_nn_integral_fst}$:

assumes $f: f \in \text{borel_measurable } (M1 \otimes_M M)$
shows $(\lambda x. \int^+ y. f (x, y) \ \partial M) \in \text{borel_measurable } M1$

using f **proof** *induct*

case $(\text{cong } u \ v)$

then have $\bigwedge w \ x. w \in \text{space } M1 \implies x \in \text{space } M \implies u (w, x) = v (w, x)$

by $(\text{auto simp: space_pair_measure})$

show *?case*

apply $(\text{subst measurable_cong})$

apply $(\text{rule nn_integral_cong})$

apply *fact+*

done

next

case $(\text{set } Q)$

have $[\text{simp}]: \bigwedge x \ y. \text{indicator } Q (x, y) = \text{indicator } (\text{Pair } x -' Q) \ y$

by $(\text{auto simp: indicator_def})$

have $\bigwedge x. x \in \text{space } M1 \implies \text{emeasure } M (\text{Pair } x -' Q) = \int^+ y. \text{indicator } Q (x, y) \ \partial M$

by $(\text{simp add: sets_Pair1}[OF \text{set}])$

from *this measurable_emeasure_Pair*[*OF set*] **show** *?case*
by (*rule measurable_cong*[*THEN iffD1*])
qed (*simp_all add: nn_integral_add nn_integral_cmult measurable_compose_Pair1*
nn_integral_monotone_convergence_SUP incseq_def le_fun_def
image_comp
cong: measurable_cong)

lemma (**in** *sigma_finite_measure*) *nn_integral_fst*:
assumes *f: f ∈ borel_measurable (M1 ⊗_M M)*
shows $(\int^+ x. \int^+ y. f(x, y) \partial M \partial M1) = \text{integral}^N (M1 \otimes_M M) f$ (**is** *?I f =*
—)
using *f proof induct*
case (*cong u v*)
then have *?I u = ?I v*
by (*intro nn_integral_cong*) (*auto simp: space_pair_measure*)
with *cong show ?case*
by (*simp cong: nn_integral_cong*)
qed (*simp_all add: emeasure_pair_measure nn_integral_cmult nn_integral_add*
nn_integral_monotone_convergence_SUP measurable_compose_Pair1
borel_measurable_nn_integral_fst nn_integral_mono incseq_def
le_fun_def image_comp
cong: nn_integral_cong)

lemma (**in** *sigma_finite_measure*) *borel_measurable_nn_integral*[*measurable (raw)*]:
case_prod f ∈ borel_measurable (N ⊗_M M) ⇒ (λx. ∫⁺ y. f x y ∂M) ∈
borel_measurable N
using *borel_measurable_nn_integral_fst*[*of case_prod f N*] **by** *simp*

proposition (**in** *pair_sigma_finite*) *nn_integral_snd*:
assumes *f[measurable]: f ∈ borel_measurable (M1 ⊗_M M2)*
shows $(\int^+ y. (\int^+ x. f(x, y) \partial M1) \partial M2) = \text{integral}^N (M1 \otimes_M M2) f$
proof —
note *measurable_pair_swap*[*OF f*]
from *M1.nn_integral_fst*[*OF this*]
have $(\int^+ y. (\int^+ x. f(x, y) \partial M1) \partial M2) = (\int^+ (x, y). f(y, x) \partial(M2 \otimes_M$
M1))
by *simp*
also have $(\int^+ (x, y). f(y, x) \partial(M2 \otimes_M M1)) = \text{integral}^N (M1 \otimes_M M2) f$
by (*subst distr_pair_swap*) (*auto simp add: nn_integral_distr intro!: nn_integral_cong*)
finally show *?thesis* .
qed

theorem (**in** *pair_sigma_finite*) *Fubini*:
assumes *f: f ∈ borel_measurable (M1 ⊗_M M2)*
shows $(\int^+ y. (\int^+ x. f(x, y) \partial M1) \partial M2) = (\int^+ x. (\int^+ y. f(x, y) \partial M2)$
∂M1)
unfolding *nn_integral_snd*[*OF assms*] *M2.nn_integral_fst*[*OF assms*] ..

theorem (**in** *pair_sigma_finite*) *Fubini'*:

assumes $f: \text{case_prod } f \in \text{borel_measurable } (M1 \otimes_M M2)$
 shows $(\int^+ y. (\int^+ x. f x y \partial M1) \partial M2) = (\int^+ x. (\int^+ y. f x y \partial M2) \partial M1)$
 using *Fubini*[*OF f*] by *simp*

7.6.4 Products on counting spaces, densities and distributions

proposition *sigma_prod*:

assumes $X_cover: \exists E \subseteq A. \text{countable } E \wedge X = \bigcup E$ and $A: A \subseteq \text{Pow } X$
 assumes $Y_cover: \exists E \subseteq B. \text{countable } E \wedge Y = \bigcup E$ and $B: B \subseteq \text{Pow } Y$
 shows $\text{sigma } X A \otimes_M \text{sigma } Y B = \text{sigma } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\}$
 (is $?P = ?S$)

proof (*rule measure_eqI*)

have [*simp*]: $\text{snd} \in X \times Y \rightarrow Y \text{fst} \in X \times Y \rightarrow X$

by *auto*

let $?XY = \{\{\text{fst} -' a \cap X \times Y \mid a. a \in A\}, \{\text{snd} -' b \cap X \times Y \mid b. b \in B\}\}$

have $\text{sets } ?P = \text{sets } (\text{SUP } xy \in ?XY. \text{sigma } (X \times Y) xy)$

by (*simp add: vimage_algebra_sigma_sets_pair_eq_sets_fst_snd A B*)

also have $\dots = \text{sets } (\text{sigma } (X \times Y) (\bigcup ?XY))$

by (*intro Sup_sigma_arg_cong[where f=sets] auto*)

also have $\dots = \text{sets } ?S$

proof (*intro arg_cong[where f=sets] sigma_eqI sigma_sets_eqI*)

show $\bigcup ?XY \subseteq \text{Pow } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\} \subseteq \text{Pow } (X \times Y)$

using *A B* by *auto*

next

interpret $XY: \text{sigma_algebra } X \times Y \text{sigma_sets } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\}$

using *A B* by (*intro sigma_algebra_sigma_sets auto*)

fix Z **assume** $Z \in \bigcup ?XY$

then show $Z \in \text{sigma_sets } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\}$

proof *safe*

fix a **assume** $a \in A$

from Y_cover **obtain** E **where** $E: E \subseteq B$ *countable E* and $Y = \bigcup E$

by *auto*

with $\langle a \in A \rangle A$ **have** $\text{eq: } \text{fst} -' a \cap X \times Y = (\bigcup e \in E. a \times e)$

by *auto*

show $\text{fst} -' a \cap X \times Y \in \text{sigma_sets } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\}$

using $\langle a \in A \rangle E$ **unfolding eq** by (*auto intro!: XY.countable_UN'*)

next

fix b **assume** $b \in B$

from X_cover **obtain** E **where** $E: E \subseteq A$ *countable E* and $X = \bigcup E$

by *auto*

with $\langle b \in B \rangle B$ **have** $\text{eq: } \text{snd} -' b \cap X \times Y = (\bigcup e \in E. e \times b)$

by *auto*

show $\text{snd} -' b \cap X \times Y \in \text{sigma_sets } (X \times Y) \{a \times b \mid a b. a \in A \wedge b \in B\}$

$B\}$

using $\langle b \in B \rangle E$ **unfolding eq** by (*auto intro!: XY.countable_UN'*)

qed

```

next
  fix Z assume Z ∈ {a × b | a b. a ∈ A ∧ b ∈ B}
  then obtain a b where Z = a × b and ab: a ∈ A b ∈ B
  by auto
  then have Z: Z = (fst -' a ∩ X × Y) ∩ (snd -' b ∩ X × Y)
  using A B by auto
  interpret XY: sigma_algebra X × Y sigma_sets (X × Y) (∪ ?XY)
  by (intro sigma_algebra_sigma_sets) auto
  show Z ∈ sigma_sets (X × Y) (∪ ?XY)
  unfolding Z by (rule XY.Int) (blast intro: ab)+
qed
finally show sets ?P = sets ?S .
next
  interpret finite_measure sigma X A for X A
  proof qed (simp add: emeasure_sigma)
  fix A assume A ∈ sets ?P then show emeasure ?P A = emeasure ?S A
  by (simp add: emeasure_pair_measure_alt emeasure_sigma)
qed

lemma sigma_sets_pair_measure_generator_finite:
  assumes finite A and finite B
  shows sigma_sets (A × B) { a × b | a b. a ⊆ A ∧ b ⊆ B } = Pow (A × B)
  (is sigma_sets ?prod ?sets = _)
proof safe
  have fin: finite (A × B) using assms by (rule finite_cartesian_product)
  fix x assume subset: x ⊆ A × B
  hence finite x using fin by (rule finite_subset)
  from this subset show x ∈ sigma_sets ?prod ?sets
  proof (induct x)
    case empty show ?case by (rule sigma_sets.Empty)
  next
    case (insert a x)
    hence {a} ∈ sigma_sets ?prod ?sets by auto
    moreover have x ∈ sigma_sets ?prod ?sets using insert by auto
    ultimately show ?case unfolding insert_is_Un[of a x] by (rule sigma_sets_Un)
  qed
next
  fix x a b
  assume x ∈ sigma_sets ?prod ?sets and (a, b) ∈ x
  from sigma_sets_into_sp[OF _ this(1)] this(2)
  show a ∈ A and b ∈ B by auto
qed

proposition sets_pair_eq:
  assumes Ea: Ea ⊆ Pow (space A) sets A = sigma_sets (space A) Ea
  and Ca: countable Ca Ca ⊆ Ea ∪ Ca = space A
  and Eb: Eb ⊆ Pow (space B) sets B = sigma_sets (space B) Eb
  and Cb: countable Cb Cb ⊆ Eb ∪ Cb = space B
  shows sets (A ⊗M B) = sets (sigma (space A × space B) { a × b | a b. a ∈

```

```

Ea ∧ b ∈ Eb }
(is _ = sets (sigma ?Ω ?E))
proof
  show sets (sigma ?Ω ?E) ⊆ sets (A ⊗M B)
    using Ea(1) Eb(1) by (subst sigma_le_sets) (auto simp: Ea(2) Eb(2))
  have ?E ⊆ Pow ?Ω
    using Ea(1) Eb(1) by auto
  then have E: a ∈ Ea ⇒ b ∈ Eb ⇒ a × b ∈ sets (sigma ?Ω ?E) for a b
    by auto
  have sets (A ⊗M B) ⊆ sets (Sup {vimage_algebra ?Ω fst A, vimage_algebra
?Ω snd B})
    unfolding sets_pair_eq_sets_fst_snd ..
  also have vimage_algebra ?Ω fst A = vimage_algebra ?Ω fst (sigma (space A)
Ea)
    by (intro vimage_algebra_cong[OF refl refl]) (simp add: Ea)
  also have ... = sigma ?Ω {fst -' A ∩ ?Ω |A. A ∈ Ea}
    by (intro Ea vimage_algebra_sigma) auto
  also have vimage_algebra ?Ω snd B = vimage_algebra ?Ω snd (sigma (space B)
Eb)
    by (intro vimage_algebra_cong[OF refl refl]) (simp add: Eb)
  also have ... = sigma ?Ω {snd -' A ∩ ?Ω |A. A ∈ Eb}
    by (intro Eb vimage_algebra_sigma) auto
  also have {sigma ?Ω {fst -' Aa ∩ ?Ω |Aa. Aa ∈ Ea}, sigma ?Ω {snd -' Aa ∩
?Ω |Aa. Aa ∈ Eb}} =
    sigma ?Ω ‘ {{fst -' Aa ∩ ?Ω |Aa. Aa ∈ Ea}, {snd -' Aa ∩ ?Ω |Aa. Aa ∈ Eb}}
    by auto
  also have sets (SUP S∈{{fst -' Aa ∩ ?Ω |Aa. Aa ∈ Ea}, {snd -' Aa ∩ ?Ω |Aa.
Aa ∈ Eb}}. sigma ?Ω S) =
    sets (sigma ?Ω (⋃ {{fst -' Aa ∩ ?Ω |Aa. Aa ∈ Ea}, {snd -' Aa ∩ ?Ω |Aa. Aa
∈ Eb}}))
    using Ea(1) Eb(1) by (intro sets_Sup_sigma) auto
  also have ... ⊆ sets (sigma ?Ω ?E)
proof (subst sigma_le_sets, safe intro!: space_in_measure_of)
  fix a assume a ∈ Ea
  then have fst -' a ∩ ?Ω = (⋃ b∈Cb. a × b)
    using Cb(3)[symmetric] Ea(1) by auto
  then show fst -' a ∩ ?Ω ∈ sets (sigma ?Ω ?E)
    using Cb ⟨a ∈ Ea⟩ by (auto intro!: sets.countable_UN' E)
  next
  fix b assume b ∈ Eb
  then have snd -' b ∩ ?Ω = (⋃ a∈Ca. a × b)
    using Ca(3)[symmetric] Eb(1) by auto
  then show snd -' b ∩ ?Ω ∈ sets (sigma ?Ω ?E)
    using Ca ⟨b ∈ Eb⟩ by (auto intro!: sets.countable_UN' E)
  qed
  finally show sets (A ⊗M B) ⊆ sets (sigma ?Ω ?E) .
  qed

```

proposition *borel_prod:*

```

(borel  $\otimes_M$  borel) = (borel :: ('a::second_countable_topology  $\times$  'b::second_countable_topology)
measure)
(is ?P = ?B)
proof -
  have ?B = sigma UNIV {A  $\times$  B | A B. open A  $\wedge$  open B}
    by (rule second_countable_borel_measurable[OF open_prod_generated])
  also have ... = ?P
    unfolding borel_def
    by (subst sigma_prod) (auto intro!: exI[of _ {UNIV}])
  finally show ?thesis ..
qed

```

proposition *pair_measure_count_space*:

```

assumes A: finite A and B: finite B
shows count_space A  $\otimes_M$  count_space B = count_space (A  $\times$  B) (is ?P =
?C)
proof (rule measure_eqI)
  interpret A: finite_measure count_space A by (rule finite_measure_count_space)
  fact
  interpret B: finite_measure count_space B by (rule finite_measure_count_space)
  fact
  interpret P: pair_sigma_finite count_space A count_space B ..
  show eq: sets ?P = sets ?C
    by (simp add: sets_pair_measure sigma_sets_pair_measure_generator_finite
A B)
  fix X assume X: X  $\in$  sets ?P
  with eq have X_subset: X  $\subseteq$  A  $\times$  B by simp
  with A B have fin_Pair:  $\bigwedge x. finite (Pair x -' X)$ 
    by (intro finite_subset[OF _ B]) auto
  have fin_X: finite X using X_subset by (rule finite_subset) (auto simp: A B)
  have card: 0 < card (Pair a -' X) if (a, b)  $\in$  X for a b
    using card_gt_0_iff fin_Pair that by auto
  then have emeasure ?P X =  $\int^+ x. emeasure (count_space B) (Pair x -' X)$ 
 $\partial count\_space A$ 
    by (simp add: B.emeasure_pair_measure_alt X)
  also have ... = emeasure ?C X
    apply (subst emeasure_count_space)
    using card X_subset A fin_Pair fin_X
    apply (auto simp add: nn_integral_count_space
of_nat_sum[symmetric] card_SigmaI[symmetric]
simp del: card_SigmaI
intro!: arg_cong[where f=card])
  done
  finally show emeasure ?P X = emeasure ?C X .
qed

```

lemma *emeasure_prod_count_space*:

```

assumes A: A  $\in$  sets (count_space UNIV  $\otimes_M$  M) (is A  $\in$  sets (?A  $\otimes_M$  ?B))

```

shows $\text{emeasure } (?A \otimes_M ?B) A = (\int^+ x. \int^+ y. \text{indicator } A (x, y) \partial ?B \partial ?A)$
by (*rule emeasure_measure_of[OF pair_measure_def]*)
(auto simp: countably_additive_def positive_def suminf_indicator A
nn_integral_suminf[symmetric] dest: sets_sets_into_space)

lemma *emeasure_prod_count_space_single[simp]:* $\text{emeasure } (\text{count_space } UNIV \otimes_M \text{count_space } UNIV) \{x\} = 1$

proof –

have [*simp*]: $\bigwedge a b x y. \text{indicator } \{(a, b)\} (x, y) = (\text{indicator } \{a\} x * \text{indicator } \{b\} y :: \text{ennreal})$

by (*auto split: split_indicator*)

show *?thesis*

by (*cases x (auto simp: emeasure_prod_count_space nn_integral_cmult sets_Pair)*)

qed

lemma *emeasure_count_space_prod_eq:*

fixes $A :: ('a \times 'b) \text{ set}$

assumes $A \in \text{sets } (\text{count_space } UNIV \otimes_M \text{count_space } UNIV)$ (**is** $A \in \text{sets } (?A \otimes_M ?B)$)

shows $\text{emeasure } (?A \otimes_M ?B) A = \text{emeasure } (\text{count_space } UNIV) A$

proof –

{ fix $A :: ('a \times 'b) \text{ set}$ **assume** *countable A*

then have $\text{emeasure } (?A \otimes_M ?B) (\bigcup a \in A. \{a\}) = (\int^+ a. \text{emeasure } (?A \otimes_M ?B) \{a\} \partial \text{count_space } A)$

by (*intro emeasure_UN_countable (auto simp: sets_Pair disjoint_family_on_def)*)

also have $\dots = (\int^+ a. \text{indicator } A a \partial \text{count_space } UNIV)$

by (*subst nn_integral_count_space_indicator auto*)

finally have $\text{emeasure } (?A \otimes_M ?B) A = \text{emeasure } (\text{count_space } UNIV) A$

by *simp }*

note $* = \text{this}$

show *?thesis*

proof *cases*

assume *finite A* **then show** *?thesis*

by (*intro * countable_finite*)

next

assume *infinite A*

then obtain C **where** *countable C and infinite C and $C \subseteq A$*

by (*auto dest: infinite_countable_subset'*)

with A **have** $\text{emeasure } (?A \otimes_M ?B) C \leq \text{emeasure } (?A \otimes_M ?B) A$

by (*intro emeasure_mono auto*)

also have $\text{emeasure } (?A \otimes_M ?B) C = \text{emeasure } (\text{count_space } UNIV) C$

using $\langle \text{countable } C \rangle$ **by** (*rule **)

finally show *?thesis*

using $\langle \text{infinite } C \rangle \langle \text{infinite } A \rangle$ **by** (*simp add: top_unique*)

qed

qed

lemma *nn_integral_count_space_prod_eq:*

$nn_integral (count_space UNIV \otimes_M count_space UNIV) f = nn_integral (count_space UNIV) f$

(is $nn_integral ?P f = _$)

proof cases

assume $cntbl: countable \{x. f x \neq 0\}$

have [$simp$]: $\bigwedge x. card (\{x\} \cap \{x. f x \neq 0\}) = (indicator \{x. f x \neq 0\} x :: ennreal)$

by ($auto split: split_indicator$)

have [$measurable$]: $\bigwedge y. (\lambda x. indicator \{y\} x) \in borel_measurable ?P$

by ($rule measurable_discrete_difference[of \lambda x. 0 _ borel \{y\} \lambda x. indicator \{y\} x \text{ for } y]$)

($auto intro: sets_Pair$)

have $(\int^{+x}. f x \partial ?P) = (\int^{+x}. \int^{+x'}. f x * indicator \{x\} x' \partial count_space \{x. f x \neq 0\} \partial ?P)$

by ($auto simp add: nn_integral_cmult nn_integral_indicator' intro!: nn_integral_cong split: split_indicator$)

also have $\dots = (\int^{+x}. \int^{+x'}. f x' * indicator \{x'\} x \partial count_space \{x. f x \neq 0\} \partial ?P)$

by ($auto intro!: nn_integral_cong split: split_indicator$)

also have $\dots = (\int^{+x'}. \int^{+x}. f x' * indicator \{x'\} x \partial ?P \partial count_space \{x. f x \neq 0\})$

by ($intro nn_integral_count_space_nn_integral cntbl$) $auto$

also have $\dots = (\int^{+x'}. f x' \partial count_space \{x. f x \neq 0\})$

by ($intro nn_integral_cong$) ($auto simp: nn_integral_cmult sets_Pair$)

finally show $?thesis$

by ($auto simp add: nn_integral_count_space_indicator intro!: nn_integral_cong split: split_indicator$)

next

{ **fix** x **assume** $f x \neq 0$

then have $(\exists r \geq 0. 0 < r \wedge f x = ennreal r) \vee f x = \infty$

by ($cases f x rule: ennreal_cases$) ($auto simp: less_le$)

then have $\exists n. ennreal (1 / real (Suc n)) \leq f x$

by ($auto elim!: nat_approx_posE intro!: less_imp_le$) }

note $* = this$

assume $cntbl: uncountable \{x. f x \neq 0\}$

also have $\{x. f x \neq 0\} = (\bigcup n. \{x. 1/Suc n \leq f x\})$

using $*$ **by** $auto$

finally obtain n **where** $infinite \{x. 1/Suc n \leq f x\}$

by ($meson countableI_type countable_UN uncountable_infinite$)

then obtain C **where** $C: C \subseteq \{x. 1/Suc n \leq f x\}$ **and** $countable C$ $infinite C$

by ($metis infinite_countable_subset'$)

have [$measurable$]: $C \in sets ?P$

using $sets.countable[OF _ \langle countable C \rangle, of ?P]$ **by** ($auto simp: sets_Pair$)

have $(\int^{+x}. ennreal (1/Suc n) * indicator C x \partial ?P) \leq nn_integral ?P f$

using C **by** ($intro nn_integral_mono$) ($auto split: split_indicator simp: zero_ereal_def[symmetric]$)

moreover have $(\int^{+x}. ennreal (1/Suc n) * indicator C x \partial ?P) = \infty$

```

using <infinite C> by (simp add: nn_integral_cmult emeasure_count_space_prod_eq
ennreal_mult_top)
moreover have ( $\int^+ x. \text{ennreal } (1/\text{Suc } n) * \text{indicator } C \ x \ \partial \text{count\_space UNIV}$ )
 $\leq$  nn_integral (count_space UNIV) f
using C by (intro nn_integral_mono) (auto split: split_indicator simp: zero_ereal_def[symmetric])
moreover have ( $\int^+ x. \text{ennreal } (1/\text{Suc } n) * \text{indicator } C \ x \ \partial \text{count\_space UNIV}$ )
 $= \infty$ 
using <infinite C> by (simp add: nn_integral_cmult ennreal_mult_top)
ultimately show ?thesis
by (simp add: top_unique)
qed

```

theorem pair_measure_density:

```

assumes f: f  $\in$  borel_measurable M1
assumes g: g  $\in$  borel_measurable M2
assumes sigma_finite_measure M2 sigma_finite_measure (density M2 g)
shows density M1 f  $\otimes_M$  density M2 g = density (M1  $\otimes_M$  M2) ( $\lambda(x,y). f \ x * g \ y$ ) (is ?L = ?R)
proof (rule measure_eqI)
interpret M2: sigma_finite_measure M2 by fact
interpret D2: sigma_finite_measure density M2 g by fact

```

fix A **assume** A: A \in sets ?L

```

with f g have ( $\int^+ x. f \ x * \int^+ y. g \ y * \text{indicator } A \ (x, y) \ \partial M2 \ \partial M1$ ) =
( $\int^+ x. \int^+ y. f \ x * g \ y * \text{indicator } A \ (x, y) \ \partial M2 \ \partial M1$ )
by (intro nn_integral_cong_AE)
(auto simp add: nn_integral_cmult[symmetric] ac_simps)

```

with A f g **show** emeasure ?L A = emeasure ?R A

```

by (simp add: D2.emeasure_pair_measure_emeasure_density nn_integral_density
M2.nn_integral_fst[symmetric]
cong: nn_integral_cong)

```

qed simp

lemma sigma_finite_measure_distr:

```

assumes sigma_finite_measure (distr M N f) and f: f  $\in$  measurable M N
shows sigma_finite_measure M

```

proof –

interpret sigma_finite_measure distr M N f **by** fact

obtain A **where** A: countable A A \subseteq sets (distr M N f)

$\bigcup A = \text{space } (\text{distr } M \ N \ f) \ \forall a \in A. \text{emeasure } (\text{distr } M \ N \ f) \ a \neq \infty$

using sigma_finite_countable **by** auto

show ?thesis

proof

show $\exists A. \text{countable } A \wedge A \subseteq \text{sets } M \wedge \bigcup A = \text{space } M \wedge (\forall a \in A. \text{emeasure } M \ a \neq \infty)$

using A f

by (intro exI[of _ ($\lambda a. f \ - \ 'a \ \cap \ \text{space } M$) 'A])

(auto simp: emeasure_distr_set_eq_iff_subset_eq intro: measurable_space)

qed

qed

lemma *pair_measure_distr*:

assumes *f*: $f \in \text{measurable } M \ S$ **and** *g*: $g \in \text{measurable } N \ T$
assumes *sigma_finite_measure* (*distr* $N \ T \ g$)
shows $\text{distr } M \ S \ f \ \otimes_M \ \text{distr } N \ T \ g = \text{distr } (M \ \otimes_M \ N) \ (S \ \otimes_M \ T) \ (\lambda(x, y). (f \ x, g \ y))$ (**is** $?P = ?D$)
proof (*rule* *measure_eqI*)
interpret *T*: *sigma_finite_measure* *distr* $N \ T \ g$ **by** *fact*
interpret *N*: *sigma_finite_measure* N **by** (*rule* *sigma_finite_measure_distr*)
fact+

fix *A* **assume** *A*: $A \in \text{sets } ?P$
with *f g* **show** *emeasure* $?P \ A = \text{emeasure } ?D \ A$
by (*auto simp add*: *N.emeasure_pair_measure_alt space_pair_measure_emeasure_distr*
 $T.emeasure_pair_measure_alt \ nn_integral_distr$
intro!: *nn_integral_cong arg_cong*[**where** $f = \text{emeasure } N$])

qed *simp*

lemma *pair_measure_eqI*:

assumes *sigma_finite_measure* $M1$ *sigma_finite_measure* $M2$
assumes *sets*: $\text{sets } (M1 \ \otimes_M \ M2) = \text{sets } M$
assumes *emeasure*: $\bigwedge A \ B. A \in \text{sets } M1 \implies B \in \text{sets } M2 \implies \text{emeasure } M1 \ A$
 $* \text{emeasure } M2 \ B = \text{emeasure } M \ (A \times B)$
shows $M1 \ \otimes_M \ M2 = M$
proof –
interpret *M1*: *sigma_finite_measure* $M1$ **by** *fact*
interpret *M2*: *sigma_finite_measure* $M2$ **by** *fact*
interpret *pair_sigma_finite* $M1 \ M2$..
let $?E = \{a \times b \mid a \in \text{sets } M1 \ \wedge \ b \in \text{sets } M2\}$
let $?P = M1 \ \otimes_M \ M2$
obtain *F* :: $\text{nat} \Rightarrow ('a \times 'b) \ \text{set}$ **where** *F*:
 $\text{range } F \subseteq ?E \ \text{incseq } F \ \cup \ (\text{range } F) = \text{space } M1 \times \text{space } M2 \ \forall i. \ \text{emeasure } ?P \ (F \ i) \neq \infty$
using *sigma_finite_up_in_pair_measure_generator*
by *blast*
show *?thesis*
proof (*rule* *measure_eqI_generator_eq*[*OF* *Int_stable_pair_measure_generator*[*of* $M1 \ M2$]])
show $?E \subseteq \text{Pow } (\text{space } ?P)$
using *sets.space_closed*[*of* $M1$] *sets.space_closed*[*of* $M2$] **by** (*auto simp*:
 $\text{space_pair_measure}$)
show $\text{sets } ?P = \text{sigma_sets } (\text{space } ?P) \ ?E$
by (*simp add*: *sets_pair_measure space_pair_measure*)
then show $\text{sets } M = \text{sigma_sets } (\text{space } ?P) \ ?E$
using *sets[symmetric]* **by** *simp*
next
show $\text{range } F \subseteq ?E \ (\cup i. F \ i) = \text{space } ?P \ \bigwedge i. \ \text{emeasure } ?P \ (F \ i) \neq \infty$

```

    using F by (auto simp: space_pair_measure)
  next
    fix X assume X ∈ ?E
    then obtain A B where X[simp]: X = A × B and A: A ∈ sets M1 and B:
B ∈ sets M2 by auto
    then have emeasure ?P X = emeasure M1 A * emeasure M2 B
      by (simp add: M2.emeasure_pair_measure_Times)
    also have ... = emeasure M (A × B)
      using A B emeasure by auto
    finally show emeasure ?P X = emeasure M X
      by simp
  qed
qed

```

lemma *sets_pair_countable*:

```

  assumes countable S1 countable S2
  assumes M: sets M = Pow S1 and N: sets N = Pow S2
  shows sets (M ⊗M N) = Pow (S1 × S2)
proof auto
  fix x a b assume x: x ∈ sets (M ⊗M N) (a, b) ∈ x
  from sets.sets_into_space[OF x(1)] x(2)
    sets_eq_imp_space_eq[of N count_space S2] sets_eq_imp_space_eq[of M
count_space S1] M N
  show a ∈ S1 b ∈ S2
    by (auto simp: space_pair_measure)
  next
  fix X assume X: X ⊆ S1 × S2
  then have countable X
    by (metis countable_subset ‹countable S1› ‹countable S2› countable_SIGMA)
  have X = (⋃ (a, b) ∈ X. {a} × {b}) by auto
  also have ... ∈ sets (M ⊗M N)
    using X
  by (safe intro!: sets.countable_UN' ‹countable X› subsetI pair_measureI) (auto
simp: M N)
  finally show X ∈ sets (M ⊗M N) .
qed

```

lemma *pair_measure_countable*:

```

  assumes countable S1 countable S2
  shows count_space S1 ⊗M count_space S2 = count_space (S1 × S2)
proof (rule pair_measure_eqI)
  show sigma_finite_measure (count_space S1) sigma_finite_measure (count_space
S2)
    using assms by (auto intro!: sigma_finite_measure_count_space_countable)
  show sets (count_space S1 ⊗M count_space S2) = sets (count_space (S1 ×
S2))
    by (subst sets_pair_countable[OF assms]) auto
  next
  fix A B assume A ∈ sets (count_space S1) B ∈ sets (count_space S2)

```

then show $\text{emeasure } (\text{count_space } S1) A * \text{emeasure } (\text{count_space } S2) B =$
 $\text{emeasure } (\text{count_space } (S1 \times S2)) (A \times B)$
by (*subst (1 2 3) emeasure_count_space*) (*auto simp: finite_cartesian_product_iff*
ennreal_mult_top ennreal_top_mult)
qed

proposition *nn_integral_fst_count_space*:

$(\int^+ x. \int^+ y. f(x, y) \partial \text{count_space } UNIV \partial \text{count_space } UNIV) = \text{integral}^N$
 $(\text{count_space } UNIV) f$
(is ?lhs = ?rhs)

proof(*cases*)

assume *: *countable* {*xy. f xy* $\neq 0$ }

let ?*A* = *fst* ' {*xy. f xy* $\neq 0$ }

let ?*B* = *snd* ' {*xy. f xy* $\neq 0$ }

from * **have** [*simp*]: *countable* ?*A* *countable* ?*B* **by**(*rule countable_image*)+

have ?*lhs* = $(\int^+ x. \int^+ y. f(x, y) \partial \text{count_space } UNIV \partial \text{count_space } ?A)$

by(*rule nn_integral_count_space_eq*)

(*auto simp add: nn_integral_0_iff_AE AE_count_space not_le intro: rev_image_eqI*)

also have ... = $(\int^+ x. \int^+ y. f(x, y) \partial \text{count_space } ?B \partial \text{count_space } ?A)$

by(*intro nn_integral_count_space_eq nn_integral_cong*)(*auto intro: rev_image_eqI*)

also have ... = $(\int^+ xy. f xy \partial \text{count_space } (?A \times ?B))$

by(*subst sigma_finite_measure.nn_integral_fst*)

(*simp_all add: sigma_finite_measure_count_space_countable pair_measure_countable*)

also have ... = ?*rhs*

by(*rule nn_integral_count_space_eq*)(*auto intro: rev_image_eqI*)

finally show ?*thesis* .

next

{ **fix** *xy* **assume** *f xy* $\neq 0$

then have $(\exists r \geq 0. 0 < r \wedge f xy = \text{ennreal } r) \vee f xy = \infty$

by (*cases f xy rule: ennreal_cases*) (*auto simp: less_le*)

then have $\exists n. \text{ennreal } (1 / \text{real } (\text{Suc } n)) \leq f xy$

by (*auto elim!: nat_approx_posE intro!: less_imp_le*) }

note * = *this*

assume *cntbl*: *uncountable* {*xy. f xy* $\neq 0$ }

also have {*xy. f xy* $\neq 0$ } = $(\bigcup n. \{xy. 1/\text{Suc } n \leq f xy\})$

using * **by** *auto*

finally obtain *n* **where** *infinite* {*xy. 1/Suc n* $\leq f xy$ }

by (*meson countableI_type countable_UN uncountable_infinite*)

then obtain *C* **where** *C*: $C \subseteq \{xy. 1/\text{Suc } n \leq f xy\}$ **and** *countable* *C* *infinite*
C

by (*metis infinite_countable_subset*)

have $\infty = (\int^+ xy. \text{ennreal } (1 / \text{Suc } n) * \text{indicator } C xy \partial \text{count_space } UNIV)$

using <*infinite C*> **by**(*simp add: nn_integral_cmult ennreal_mult_top*)

also have ... \leq ?*rhs* **using** *C*

by(*intro nn_integral_mono*)(*auto split: split_indicator*)

finally have ?*rhs* = ∞ **by** (*simp add: top_unique*)

moreover have ?*lhs* = ∞

```

proof(cases finite (fst ' C))
  case True
  then obtain x C' where x: x ∈ fst ' C
    and C': C' = fst -' {x} ∩ C
    and infinite C'
    using ⟨infinite C⟩ by(auto elim!: inf_img_fin_domE')
  from x C C' have **: C' ⊆ {xy. 1 / Suc n ≤ f xy} by auto

  from C' ⟨infinite C'⟩ have infinite (snd ' C')
    by(auto dest!: finite_imageD simp add: inj_on_def)
  then have ∞ = (∫+ y. ennreal (1 / Suc n) * indicator (snd ' C') y ∂count_space UNIV)
    by(simp add: nn_integral_cmult ennreal_mult_top)
  also have ... = (∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV)
    by(rule nn_integral_cong)(force split: split_indicator intro: rev_image_eqI simp add: C')
  also have ... = (∫+ x'. (∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV) * indicator {x} x' ∂count_space UNIV)
    by(simp add: one_ereal_def[symmetric])
  also have ... ≤ (∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C' (x, y) ∂count_space UNIV ∂count_space UNIV)
    by(rule nn_integral_mono)(simp split: split_indicator)
  also have ... ≤ ?lhs using **
    by(intro nn_integral_mono)(auto split: split_indicator)
  finally show ?thesis by (simp add: top_unique)
next
  case False
  define C' where C' = fst ' C
  have ∞ = ∫+ x. ennreal (1 / Suc n) * indicator C' x ∂count_space UNIV
    using C'_def False by(simp add: nn_integral_cmult ennreal_mult_top)
  also have ... = ∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C' x * indicator {SOME y. (x, y) ∈ C} y ∂count_space UNIV ∂count_space UNIV
    by(auto simp add: one_ereal_def[symmetric] max_def intro: nn_integral_cong)
  also have ... ≤ ∫+ x. ∫+ y. ennreal (1 / Suc n) * indicator C (x, y) ∂count_space UNIV ∂count_space UNIV
    by(intro nn_integral_mono)(auto simp add: C'_def split: split_indicator intro: someI)
  also have ... ≤ ?lhs using C
    by(intro nn_integral_mono)(auto split: split_indicator)
  finally show ?thesis by (simp add: top_unique)
qed
ultimately show ?thesis by simp
qed

```

proposition nn_integral_snd_count_space:

$$(\int^+ y. \int^+ x. f(x, y) \partial \text{count_space UNIV} \partial \text{count_space UNIV}) = \text{integral}^N(\text{count_space UNIV}) f$$

(is ?lhs = ?rhs)

proof –

have $?lhs = (\int^+ y. \int^+ x. (\lambda(y, x). f(x, y)) (y, x) \partial count_space UNIV)$
 $\partial count_space UNIV)$

by (*simp*)

also have $\dots = \int^+ yx. (\lambda(y, x). f(x, y)) yx \partial count_space UNIV$

by (*rule nn_integral_fst_count_space*)

also have $\dots = \int^+ xy. f xy \partial count_space ((\lambda(x, y). (y, x)) ' UNIV)$

by (*subst nn_integral_bij_count_space[OF inj_on_imp_bij_betw, symmetric]*)
(simp_all add: inj_on_def split_def)

also have $\dots = ?rhs$ **by** (*rule nn_integral_count_space_eq*) *auto*

finally show *?thesis* .

qed

lemma *measurable_pair_measure_countable1*:

assumes *countable A*

and [*measurable*]: $\bigwedge x. x \in A \implies (\lambda y. f(x, y)) \in measurable N K$

shows $f \in measurable (count_space A \otimes_M N) K$

using $_ _ assms(1)$

by (*rule measurable_compose_countable'[where f= $\lambda a b. f(a, snd b)$ and $g=fst$*
and $I=A$, *simplified*]) *simp_all*

7.6.5 Product of Borel spaces

theorem *borel_Times*:

fixes $A :: 'a::topological_space set$ **and** $B :: 'b::topological_space set$

assumes $A \in sets borel$ **and** $B \in sets borel$

shows $A \times B \in sets borel$

proof –

have $A \times B = (A \times UNIV) \cap (UNIV \times B)$

by *auto*

moreover

{ **have** $A \in sigma_sets UNIV \{S. open S\}$ **using** A **by** (*simp add: sets_borel*)

then have $A \times UNIV \in sets borel$

proof (*induct A*)

case (*Basic S*) **then show** *?case*

by (*auto intro!: borel_open open_Times*)

next

case (*Compl A*)

moreover have $*$: $(UNIV - A) \times UNIV = UNIV - (A \times UNIV)$

by *auto*

ultimately show *?case*

unfolding $*$ **by** *auto*

next

case (*Union A*)

moreover have $*$: $(\bigcup (A ' UNIV)) \times UNIV = \bigcup ((\lambda i. A i \times UNIV) ' UNIV)$

by *auto*

ultimately show *?case*

unfolding $*$ **by** *auto*

qed *simp* }

```

moreover
{ have  $B \in \text{sigma\_sets } UNIV \{S. \text{open } S\}$  using  $B$  by (simp add: sets_borel)
  then have  $UNIV \times B \in \text{sets borel}$ 
  proof (induct B)
    case (Basic S) then show ?case
      by (auto intro!: borel_open open_Times)
  next
    case (Compl B)
    moreover have *:  $UNIV \times (UNIV - B) = UNIV - (UNIV \times B)$ 
      by auto
    ultimately show ?case
      unfolding * by auto
  next
    case (Union B)
    moreover have *:  $UNIV \times (\bigcup (B \text{ ' } UNIV)) = \bigcup ((\lambda i. UNIV \times B i) \text{ ' } UNIV)$ 
      by auto
    ultimately show ?case
      unfolding * by auto
  qed simp }
ultimately show ?thesis
  by auto
qed

```

```

lemma finite_measure_pair_measure:
  assumes finite_measure M finite_measure N
  shows  $\text{finite\_measure } (N \otimes_M M)$ 
proof (rule finite_measureI)
  interpret  $M: \text{finite\_measure } M$  by fact
  interpret  $N: \text{finite\_measure } N$  by fact
  show  $\text{emeasure } (N \otimes_M M) (\text{space } (N \otimes_M M)) \neq \infty$ 
    by (auto simp: space_pair_measure M.emeasure_pair_measure_Times en-
nreal_mult_eq_top_iff)
qed

```

end

7.7 Finite Product Measure

```

theory Finite_Product_Measure
imports Binary_Product_Measure Function_Topology
begin

```

```

lemma Pi_choice:  $(\forall i \in I. \exists x \in F i. P i x) \longleftrightarrow (\exists f \in \text{Pi } I F. \forall i \in I. P i (f i))$ 
  by (metis Pi_iff)

```

```

lemma PiE_choice:  $(\forall i \in I. \exists x \in F i. P i x) \longleftrightarrow (\exists f \in \text{PiE } I F. \forall i \in I. P i (f i))$ 
  unfolding Pi_choice by (metis Int_iff PiE_def restrict_PiE restrict_apply)

```

```

lemma case_prod_const:  $(\lambda(i, j). c) = (\lambda_. c)$ 

```

by auto

7.7.1 More about Function restricted by *extensional*

definition

$\text{merge } I J = (\lambda(x, y) i. \text{if } i \in I \text{ then } x \ i \text{ else if } i \in J \text{ then } y \ i \text{ else undefined})$

lemma *merge_apply[simp]*:

$I \cap J = \{\} \implies i \in I \implies \text{merge } I J (x, y) \ i = x \ i$
 $I \cap J = \{\} \implies i \in J \implies \text{merge } I J (x, y) \ i = y \ i$
 $J \cap I = \{\} \implies i \in I \implies \text{merge } I J (x, y) \ i = x \ i$
 $J \cap I = \{\} \implies i \in J \implies \text{merge } I J (x, y) \ i = y \ i$
 $i \notin I \implies i \notin J \implies \text{merge } I J (x, y) \ i = \text{undefined}$

unfolding *merge_def* by auto

lemma *merge_commute*:

$I \cap J = \{\} \implies \text{merge } I J (x, y) = \text{merge } J I (y, x)$
 by (force simp: merge_def)

lemma *Pi_cancel_merge_range[simp]*:

$I \cap J = \{\} \implies x \in \text{Pi } I (\text{merge } I J (A, B)) \longleftrightarrow x \in \text{Pi } I A$
 $I \cap J = \{\} \implies x \in \text{Pi } I (\text{merge } J I (B, A)) \longleftrightarrow x \in \text{Pi } I A$
 $J \cap I = \{\} \implies x \in \text{Pi } I (\text{merge } I J (A, B)) \longleftrightarrow x \in \text{Pi } I A$
 $J \cap I = \{\} \implies x \in \text{Pi } I (\text{merge } J I (B, A)) \longleftrightarrow x \in \text{Pi } I A$
 by (auto simp: Pi_def)

lemma *Pi_cancel_merge[simp]*:

$I \cap J = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } I B \longleftrightarrow x \in \text{Pi } I B$
 $J \cap I = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } I B \longleftrightarrow x \in \text{Pi } I B$
 $I \cap J = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } J B \longleftrightarrow y \in \text{Pi } J B$
 $J \cap I = \{\} \implies \text{merge } I J (x, y) \in \text{Pi } J B \longleftrightarrow y \in \text{Pi } J B$
 by (auto simp: Pi_def)

lemma *extensional_merge[simp]*: $\text{merge } I J (x, y) \in \text{extensional } (I \cup J)$

by (auto simp: extensional_def)

lemma *restrict_merge[simp]*:

$I \cap J = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) \ I = \text{restrict } x \ I$
 $I \cap J = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) \ J = \text{restrict } y \ J$
 $J \cap I = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) \ I = \text{restrict } x \ I$
 $J \cap I = \{\} \implies \text{restrict } (\text{merge } I J (x, y)) \ J = \text{restrict } y \ J$
 by (auto simp: restrict_def)

lemma *split_merge*: $P (\text{merge } I J (x, y) \ i) \longleftrightarrow (i \in I \longrightarrow P (x \ i)) \wedge (i \in J - I \longrightarrow P (y \ i)) \wedge (i \notin I \cup J \longrightarrow P \ \text{undefined})$

unfolding *merge_def* by auto

lemma *PiE_cancel_merge[simp]*:

$I \cap J = \{\} \implies$

$\text{merge } I J (x, y) \in \text{Pi}_E (I \cup J) B \longleftrightarrow x \in \text{Pi } I B \wedge y \in \text{Pi } J B$
by (*auto simp: PiE_def restrict_Pi_cancel*)

lemma *merge_singleton[simp]*: $i \notin I \implies \text{merge } I \{i\} (x, y) = \text{restrict } (x(i := y$
 $i)) (\text{insert } i I)$

unfolding *merge_def* **by** (*auto simp: fun_eq_iff*)

lemma *extensional_merge_sub*: $I \cup J \subseteq K \implies \text{merge } I J (x, y) \in \text{extensional } K$

unfolding *merge_def extensional_def* **by** *auto*

lemma *merge_restrict[simp]*:

$\text{merge } I J (\text{restrict } x I, y) = \text{merge } I J (x, y)$

$\text{merge } I J (x, \text{restrict } y J) = \text{merge } I J (x, y)$

unfolding *merge_def* **by** *auto*

lemma *merge_x_x_eq_restrict[simp]*:

$\text{merge } I J (x, x) = \text{restrict } x (I \cup J)$

unfolding *merge_def* **by** *auto*

lemma *injective_vimage_restrict*:

assumes $J: J \subseteq I$

and sets: $A \subseteq (\prod_E i \in J. S i)$ $B \subseteq (\prod_E i \in J. S i)$ **and** $ne: (\prod_E i \in I. S i) \neq \{\}$

and eq: $(\lambda x. \text{restrict } x J) -' A \cap (\prod_E i \in I. S i) = (\lambda x. \text{restrict } x J) -' B \cap (\prod_E$
 $i \in I. S i)$

shows $A = B$

proof (*intro set_eqI*)

fix x

from ne **obtain** y **where** $y: \bigwedge i. i \in I \implies y i \in S i$ **by** *auto*

have $J \cap (I - J) = \{\}$ **by** *auto*

show $x \in A \longleftrightarrow x \in B$

proof *cases*

assume $x: x \in (\prod_E i \in J. S i)$

have $x \in A \longleftrightarrow \text{merge } J (I - J) (x, y) \in (\lambda x. \text{restrict } x J) -' A \cap (\prod_E i \in I.$
 $S i)$

using $y x \langle J \subseteq I \rangle \text{PiE_cancel_merge[of } J I - J x y S]$

by (*auto simp del: PiE_cancel_merge simp add: Un_absorb1*)

then show $x \in A \longleftrightarrow x \in B$

using $y x \langle J \subseteq I \rangle \text{PiE_cancel_merge[of } J I - J x y S]$

by (*auto simp del: PiE_cancel_merge simp add: Un_absorb1 eq*)

qed (*use sets in auto*)

qed

lemma *restrict_vimage*:

$I \cap J = \{\} \implies$

$(\lambda x. (\text{restrict } x I, \text{restrict } x J)) -' (\text{Pi}_E I E \times \text{Pi}_E J F) = \text{Pi } (I \cup J) (\text{merge}$
 $I J (E, F))$

by (*auto simp: restrict_Pi_cancel PiE_def*)

lemma *merge_vimage*:

$I \cap J = \{\} \implies \text{merge } I J -' \text{Pi}_E (I \cup J) E = \text{Pi } I E \times \text{Pi } J E$
by (auto simp: restrict_Pi_cancel PiE_def)

7.7.2 Finite product spaces

definition *prod_emb* **where**

$\text{prod_emb } I M K X = (\lambda x. \text{restrict } x K) -' X \cap (\prod_E i \in I. \text{space } (M i))$

lemma *prod_emb_iff*:

$f \in \text{prod_emb } I M K X \iff f \in \text{extensional } I \wedge (\text{restrict } f K \in X) \wedge (\forall i \in I. f i \in \text{space } (M i))$

unfolding *prod_emb_def* *PiE_def* **by** auto

lemma

shows *prod_emb_empty[simp]*: $\text{prod_emb } M L K \{\} = \{\}$

and *prod_emb_Un[simp]*: $\text{prod_emb } M L K (A \cup B) = \text{prod_emb } M L K A \cup \text{prod_emb } M L K B$

and *prod_emb_Int*: $\text{prod_emb } M L K (A \cap B) = \text{prod_emb } M L K A \cap \text{prod_emb } M L K B$

and *prod_emb_UN[simp]*: $\text{prod_emb } M L K (\bigcup i \in I. F i) = (\bigcup i \in I. \text{prod_emb } M L K (F i))$

and *prod_emb_INT[simp]*: $I \neq \{\} \implies \text{prod_emb } M L K (\bigcap i \in I. F i) = (\bigcap i \in I. \text{prod_emb } M L K (F i))$

and *prod_emb_Diff[simp]*: $\text{prod_emb } M L K (A - B) = \text{prod_emb } M L K A - \text{prod_emb } M L K B$

by (auto simp: *prod_emb_def*)

lemma *prod_emb_PiE*: $J \subseteq I \implies (\bigwedge i. i \in J \implies E i \subseteq \text{space } (M i)) \implies$

$\text{prod_emb } I M J (\prod_E i \in J. E i) = (\prod_E i \in I. \text{if } i \in J \text{ then } E i \text{ else } \text{space } (M i))$

by (force simp: *prod_emb_def* *PiE_iff* *if_split_mem2*)

lemma *prod_emb_PiE_same_index[simp]*:

$(\bigwedge i. i \in I \implies E i \subseteq \text{space } (M i)) \implies \text{prod_emb } I M I (\text{Pi}_E I E) = \text{Pi}_E I E$

by (auto simp: *prod_emb_def* *PiE_iff*)

lemma *prod_emb_trans[simp]*:

$J \subseteq K \implies K \subseteq L \implies \text{prod_emb } L M K (\text{prod_emb } K M J X) = \text{prod_emb } L M J X$

by (auto simp add: *Int_absorb1* *prod_emb_def* *PiE_def*)

lemma *prod_emb_Pi*:

assumes $X \in (\prod j \in J. \text{sets } (M j)) \quad J \subseteq K$

shows $\text{prod_emb } K M J (\text{Pi}_E J X) = (\prod_E i \in K. \text{if } i \in J \text{ then } X i \text{ else } \text{space } (M i))$

using *assms* *sets.space_closed*

by (auto simp: *prod_emb_def* *PiE_iff* *split: if_split_asm*) *blast+*

lemma *prod_emb_id*:

$B \subseteq (\prod_E i \in L. \text{space } (M i)) \implies \text{prod_emb } L M L B = B$

by (auto simp: prod_emb_def subset_eq extensional_restrict)

lemma prod_emb_mono:

$F \subseteq G \implies \text{prod_emb } A \ M \ B \ F \subseteq \text{prod_emb } A \ M \ B \ G$
by (auto simp: prod_emb_def)

definition PiM :: 'i set \Rightarrow ('i \Rightarrow 'a measure) \Rightarrow ('i \Rightarrow 'a) measure **where**

PiM I M = extend_measure ($\Pi_E \ i \in I. \text{space } (M \ i)$)
 $\{(J, X). (J \neq \{\} \vee I = \{\}) \wedge \text{finite } J \wedge J \subseteq I \wedge X \in (\Pi \ j \in J. \text{sets } (M \ j))\}$
 $(\lambda(J, X). \text{prod_emb } I \ M \ J \ (\Pi_E \ j \in J. X \ j))$
 $(\lambda(J, X). \prod \ j \in J \cup \{i \in I. \text{emeasure } (M \ i) \ (\text{space } (M \ i)) \neq 1\}. \text{if } j \in J \text{ then}$
 $\text{emeasure } (M \ j) \ (X \ j) \text{ else } \text{emeasure } (M \ j) \ (\text{space } (M \ j)))$

definition prod_algebra :: 'i set \Rightarrow ('i \Rightarrow 'a measure) \Rightarrow ('i \Rightarrow 'a) set set **where**

prod_algebra I M = $(\lambda(J, X). \text{prod_emb } I \ M \ J \ (\Pi_E \ j \in J. X \ j))$ '
 $\{(J, X). (J \neq \{\} \vee I = \{\}) \wedge \text{finite } J \wedge J \subseteq I \wedge X \in (\Pi \ j \in J. \text{sets } (M \ j))\}$

abbreviation

$Pi_M \ I \ M \equiv \text{PiM } I \ M$

syntax

$_ \text{PiM} :: \text{pttrn} \Rightarrow 'i \text{ set} \Rightarrow 'a \text{ measure} \Rightarrow ('i \Rightarrow 'a) \text{ measure} \ ((\exists \Pi_M _ \in _ / _)$
10)

translations

$\Pi_M \ x \in I. M == \text{CONST } \text{PiM } I \ (\%x. M)$

lemma extend_measure_cong:

assumes $\Omega = \Omega' \ I = I' \ G = G' \ \wedge i. i \in I' \implies \mu \ i = \mu' \ i$
shows $\text{extend_measure } \Omega \ I \ G \ \mu = \text{extend_measure } \Omega' \ I' \ G' \ \mu'$
unfolding extend_measure_def **by** (auto simp add: assms)

lemma Pi_cong_sets:

$\llbracket I = J; \wedge x. x \in I \implies M \ x = N \ x \rrbracket \implies \text{Pi } I \ M = \text{Pi } J \ N$
by auto

lemma PiM_cong:

assumes $I = J \ \wedge x. x \in I \implies M \ x = N \ x$
shows $\text{PiM } I \ M = \text{PiM } J \ N$
unfolding PiM_def

proof (rule extend_measure_cong, goal_cases)

case 1

show ?case **using** assms

by (subst assms(1), intro PiE_cong[of J $\lambda i. \text{space } (M \ i)$ $\lambda i. \text{space } (N \ i)$])

simp_all

next

case 2

have $\wedge K. K \subseteq J \implies (\Pi \ j \in K. \text{sets } (M \ j)) = (\Pi \ j \in K. \text{sets } (N \ j))$

using assms **by** (intro Pi_cong_sets) auto

thus ?case **by** (auto simp: assms)

```

next
  case 3
  show ?case using assms
    by (intro ext) (auto simp: prod_emb_def dest: PiE_mem)
next
  case (4 x)
  thus ?case using assms
    by (auto intro!: prod.cong split: if_split_asm)
qed

```

```

lemma prod_algebra_sets_into_space:
  prod_algebra I M  $\subseteq$  Pow ( $\prod_{E \ i \in I. \ space \ (M \ i)$ )
  by (auto simp: prod_emb_def prod_algebra_def)

```

```

lemma prod_algebra_eq_finite:
  assumes I: finite I
  shows prod_algebra I M =  $\{(\prod_{E \ i \in I. \ X \ i) \mid X. \ X \in (\prod_{j \in I. \ sets \ (M \ j))\}$  (is ?L
= ?R)
proof (intro iffI set_eqI)
  fix A assume A  $\in$  ?L
  then obtain J E where J: J  $\neq$  {}  $\vee$  I = {} finite J J  $\subseteq$  I  $\forall i \in J. \ E \ i \in sets$ 
(M i)
    and A: A = prod_emb I M J ( $\prod_{E \ j \in J. \ E \ j$ )
    by (auto simp: prod_algebra_def)
  let ?A =  $\prod_{E \ i \in I. \ if \ i \in J \ then \ E \ i \ else \ space \ (M \ i)$ 
  have A: A = ?A
    unfolding A using J by (intro prod_emb_PiE sets.sets_into_space) auto
  show A  $\in$  ?R unfolding A using J sets.top
    by (intro CollectI exI[of _  $\lambda i. \ if \ i \in J \ then \ E \ i \ else \ space \ (M \ i)$ ]) simp
next
  fix A assume A  $\in$  ?R
  then obtain X where A: A = ( $\prod_{E \ i \in I. \ X \ i$ ) and X: X  $\in$  ( $\prod_{j \in I. \ sets \ (M \ j)$ )
  by auto
  then have A: A = prod_emb I M I ( $\prod_{E \ i \in I. \ X \ i$ )
    by (simp add: prod_emb_PiE_same_index[OF sets.sets_into_space] Pi_iff)
  from X I show A  $\in$  ?L unfolding A
    by (auto simp: prod_algebra_def)
qed

```

```

lemma prod_algebraI:
  finite J  $\implies$  (J  $\neq$  {}  $\vee$  I = {})  $\implies$  J  $\subseteq$  I  $\implies$  ( $\bigwedge i. \ i \in J \implies \ E \ i \in sets \ (M \ i)$ )
 $\implies$  prod_emb I M J ( $\prod_{E \ j \in J. \ E \ j$ )  $\in$  prod_algebra I M
  by (auto simp: prod_algebra_def)

```

```

lemma prod_algebraI_finite:
  finite I  $\implies$  ( $\forall i \in I. \ E \ i \in sets \ (M \ i)$ )  $\implies$  (PiE I E)  $\in$  prod_algebra I M
  using prod_algebraI[of I I E M] prod_emb_PiE_same_index[of I E M, OF
sets.sets_into_space] by simp

```

lemma *Int_stable_PiE*: *Int_stable* {*PiE* *J E* | *E*. $\forall i \in I. E\ i \in \text{sets } (M\ i)$ }

proof (*safe intro!*: *Int_stableI*)

fix *E F* **assume** $\forall i \in I. E\ i \in \text{sets } (M\ i) \ \forall i \in I. F\ i \in \text{sets } (M\ i)$

then show $\exists G. PiE\ J\ E \cap PiE\ J\ F = PiE\ J\ G \wedge (\forall i \in I. G\ i \in \text{sets } (M\ i))$

by (*auto intro!*: *exI*[*of* _ $\lambda i. E\ i \cap F\ i$] *simp*: *PiE_Int*)

qed

lemma *prod_algebraE*:

assumes *A*: $A \in \text{prod_algebra } I\ M$

obtains *J E* **where** $A = \text{prod_emb } I\ M\ J\ (\Pi_{E\ j \in J. E\ j})$

finite *J* $J \neq \{\} \vee I = \{\} \ J \subseteq I \ \wedge i. i \in J \implies E\ i \in \text{sets } (M\ i)$

using *A* **by** (*auto simp*: *prod_algebra_def*)

lemma *prod_algebraE_all*:

assumes *A*: $A \in \text{prod_algebra } I\ M$

obtains *E* **where** $A = PiE\ I\ E\ E \in (\Pi\ i \in I. \text{sets } (M\ i))$

proof –

from *A* **obtain** *E J* **where** $A = \text{prod_emb } I\ M\ J\ (PiE\ J\ E)$

and *J*: $J \subseteq I$ **and** *E*: $E \in (\Pi\ i \in J. \text{sets } (M\ i))$

by (*auto simp*: *prod_algebra_def*)

from *E* **have** $\wedge i. i \in J \implies E\ i \subseteq \text{space } (M\ i)$

using *sets.sets_into_space* **by** *auto*

then have $A = (\Pi_{E\ i \in I. \text{if } i \in J \text{ then } E\ i \text{ else } \text{space } (M\ i)})$

using *A J* **by** (*auto simp*: *prod_emb_PiE*)

moreover have $(\lambda i. \text{if } i \in J \text{ then } E\ i \text{ else } \text{space } (M\ i)) \in (\Pi\ i \in I. \text{sets } (M\ i))$

using *sets.top E* **by** *auto*

ultimately show *?thesis* **using** *that* **by** *auto*

qed

lemma *Int_stable_prod_algebra*: *Int_stable* (*prod_algebra* *I M*)

unfolding *Int_stable_def*

proof *safe*

fix *A* **assume** $A \in \text{prod_algebra } I\ M$

from *prod_algebraE*[*OF this*] **obtain** *J E* **where** *A*:

$A = \text{prod_emb } I\ M\ J\ (PiE\ J\ E)$

finite *J*

$J \neq \{\} \vee I = \{\}$

$J \subseteq I$

$\wedge i. i \in J \implies E\ i \in \text{sets } (M\ i)$

by *auto*

fix *B* **assume** $B \in \text{prod_algebra } I\ M$

from *prod_algebraE*[*OF this*] **obtain** *K F* **where** *B*:

$B = \text{prod_emb } I\ M\ K\ (PiE\ K\ F)$

finite *K*

$K \neq \{\} \vee I = \{\}$

$K \subseteq I$

$\wedge i. i \in K \implies F\ i \in \text{sets } (M\ i)$

by *auto*

```

have  $A \cap B = \text{prod\_emb } I M (J \cup K) (\Pi_E i \in J \cup K. (\text{if } i \in J \text{ then } E i \text{ else } \text{space } (M i))) \cap$ 
  ( $\text{if } i \in K \text{ then } F i \text{ else } \text{space } (M i)$ )
unfolding  $A B$  using  $A(2,3,4) A(5)[\text{THEN sets.sets\_into\_space}] B(2,3,4)$ 
   $B(5)[\text{THEN sets.sets\_into\_space}]$ 
apply ( $\text{subst } (1\ 2\ 3) \text{prod\_emb\_PiE}$ )
apply ( $\text{simp\_all add: subset\_eq PiE\_Int}$ )
apply blast
apply ( $\text{intro PiE\_cong}$ )
apply auto
done
also have  $\dots \in \text{prod\_algebra } I M$ 
using  $A B$  by ( $\text{auto intro!: prod\_algebraI}$ )
finally show  $A \cap B \in \text{prod\_algebra } I M$  .
qed

```

```

proposition prod_algebra_mono:
assumes  $\text{space: } \bigwedge i. i \in I \implies \text{space } (E i) = \text{space } (F i)$ 
assumes  $\text{sets: } \bigwedge i. i \in I \implies \text{sets } (E i) \subseteq \text{sets } (F i)$ 
shows  $\text{prod\_algebra } I E \subseteq \text{prod\_algebra } I F$ 
proof
fix  $A$  assume  $A \in \text{prod\_algebra } I E$ 
then obtain  $J G$  where  $J: J \neq \{\} \vee I = \{\}$  finite  $J J \subseteq I$ 
  and  $A: A = \text{prod\_emb } I E J (\Pi_E i \in J. G i)$ 
  and  $G: \bigwedge i. i \in J \implies G i \in \text{sets } (E i)$ 
  by ( $\text{auto simp: prod\_algebra\_def}$ )
moreover
from  $\text{space}$  have  $(\Pi_E i \in I. \text{space } (E i)) = (\Pi_E i \in I. \text{space } (F i))$ 
  by ( $\text{rule PiE\_cong}$ )
with  $A$  have  $A = \text{prod\_emb } I F J (\Pi_E i \in J. G i)$ 
  by ( $\text{simp add: prod\_emb\_def}$ )
moreover
from  $\text{sets } G J$  have  $\bigwedge i. i \in J \implies G i \in \text{sets } (F i)$ 
  by auto
ultimately show  $A \in \text{prod\_algebra } I F$ 
  apply ( $\text{simp add: prod\_algebra\_def image\_iff}$ )
  apply ( $\text{intro exI[of\_ ] exI[of\_ ] conjI}$ )
  apply auto
done

```

qed

```

proposition prod_algebra_cong:
assumes  $I = J$  and  $(\bigwedge i. i \in I \implies \text{sets } (M i) = \text{sets } (N i))$ 
shows  $\text{prod\_algebra } I M = \text{prod\_algebra } J N$ 
by ( $\text{metis assms prod\_algebra\_mono sets\_eq imp\_space\_eq subsetI subset\_antisym}$ )

```

lemma *space_in_prod_algebra*: $(\Pi_E i \in I. \text{space } (M i)) \in \text{prod_algebra } I M$

proof *cases*

```

assume  $I = \{\}$  then show ?thesis
  by ( $\text{auto simp add: prod\_algebra\_def image\_iff prod\_emb\_def}$ )

```

```

next
  assume  $I \neq \{\}$ 
  then obtain  $i$  where  $i \in I$  by auto
  then have  $(\prod_{E} i \in I. \text{space } (M i)) = \text{prod\_emb } I M \{i\} (\prod_{E} i \in \{i\}. \text{space } (M i))$ 
    by (auto simp: prod_emb_def)
  then show ?thesis
    by (simp add:  $\langle i \in I \rangle \text{prod\_algebra } I$ )
qed

lemma space_PiM:  $\text{space } (\prod_{M} i \in I. M i) = (\prod_{E} i \in I. \text{space } (M i))$ 
  using prod_algebra_sets_into_space unfolding PiM_def prod_algebra_def by
(intro space_extend_measure) simp

lemma prod_emb_subset_PiM[simp]:  $\text{prod\_emb } I M K X \subseteq \text{space } (PiM I M)$ 
  by (auto simp: prod_emb_def space_PiM)

lemma space_PiM_empty_iff[simp]:  $\text{space } (PiM I M) = \{\} \iff (\exists i \in I. \text{space } (M i) = \{\})$ 
  by (auto simp: space_PiM PiE_eq_empty_iff)

lemma undefined_in_PiM_empty[simp]:  $(\lambda x. \text{undefined}) \in \text{space } (PiM \{\} M)$ 
  by (auto simp: space_PiM)

lemma sets_PiM:  $\text{sets } (\prod_{M} i \in I. M i) = \text{sigma\_sets } (\prod_{E} i \in I. \text{space } (M i))$ 
  (prod_algebra I M)
  using prod_algebra_sets_into_space unfolding PiM_def prod_algebra_def by
(intro sets_extend_measure) simp

proposition sets_PiM_single:  $\text{sets } (PiM I M) =$ 
   $\text{sigma\_sets } (\prod_{E} i \in I. \text{space } (M i)) \{ \{f \in \prod_{E} i \in I. \text{space } (M i). f i \in A\} \mid i A. i$ 
 $\in I \wedge A \in \text{sets } (M i)\}$ 
  (is  $\_ = \text{sigma\_sets } ?\Omega ?R$ )
  unfolding sets_PiM
proof (rule sigma_sets_eqI)
  interpret R: sigma_algebra ? $\Omega$  sigma_sets ? $\Omega$  ?R by (rule sigma_algebra_sigma_sets)
  auto
  fix A assume  $A \in \text{prod\_algebra } I M$ 
  from prod_algebraE[OF this] obtain J X where X:
     $A = \text{prod\_emb } I M J (PiE J X)$ 
    finite J
     $J \neq \{\} \vee I = \{\}$ 
     $J \subseteq I$ 
     $\bigwedge i. i \in J \implies X i \in \text{sets } (M i)$ 
  by auto
  show  $A \in \text{sigma\_sets } ?\Omega ?R$ 
proof cases
  assume  $I = \{\}$ 
  with X show ?thesis
    by (metis (no_types, lifting) PiE_cong R.top_empty_iff prod_emb_PiE

```

```

subset_eq)
next
  assume  $I \neq \{\}$ 
  with  $X$  have  $A = (\bigcap_{j \in J}. \{f \in (\prod_E i \in I. \text{space } (M i)). f j \in X j\})$ 
  by (auto simp: prod_emb_def)
  also have  $\dots \in \text{sigma\_sets } ?\Omega ?R$ 
  using  $X \langle I \neq \{\} \rangle$  by (intro R.finite_INT sigma_sets.Basic) auto
  finally show  $A \in \text{sigma\_sets } ?\Omega ?R$  .
qed
next
  fix  $A$  assume  $A \in ?R$ 
  then obtain  $i B$  where  $A: A = \{f \in \prod_E i \in I. \text{space } (M i). f i \in B\} \ i \in I \ B \in$ 
sets  $(M i)$ 
  by auto
  then have  $A = \text{prod\_emb } I \ M \ \{i\} \ (\prod_E i \in \{i\}. B)$ 
  by (auto simp: prod_emb_def)
  also have  $\dots \in \text{sigma\_sets } ?\Omega (\text{prod\_algebra } I \ M)$ 
  using  $A$  by (intro sigma_sets.Basic prod_algebraI) auto
  finally show  $A \in \text{sigma\_sets } ?\Omega (\text{prod\_algebra } I \ M)$  .
qed

```

lemma sets_PiM_eq_proj:

```

  assumes  $I \neq \{\}$ 
  shows sets  $(\text{PiM } I \ M) = \text{sets } (\text{SUP } i \in I. \text{vimage\_algebra } (\prod_E i \in I. \text{space } (M i))$ 
 $(\lambda x. x i) (M i))$ 
  (is ?lhs = ?rhs)
proof -
  have ?lhs =
    sigma_sets  $(\prod_E i \in I. \text{space } (M i)) \ \{\{f \in \prod_E i \in I. \text{space } (M i). f i \in A\} \mid i$ 
 $A. i \in I \wedge A \in \text{sets } (M i)\}$ 
  by (simp add: sets_PiM_single)
  also have  $\dots = \text{sigma\_sets } (\prod_E i \in I. \text{space } (M i))$ 
 $(\bigcup x \in I. \text{sets } (\text{vimage\_algebra } (\prod_E i \in I. \text{space } (M i)) (\lambda x a. x a x) (M$ 
 $x)))$ 
  apply (subst arg_cong [of __ Sup, OF image_cong, OF refl])
  apply (rule sets_vimage_algebra2)
  by (auto intro!: arg_cong2[where f=sigma_sets])
  also have  $\dots = \text{sigma\_sets } (\prod_E i \in I. \text{space } (M i))$ 
 $(\bigcup (\text{sets } '(\lambda i. \text{vimage\_algebra } (\prod_E i \in I. \text{space } (M i)) (\lambda x. x i) (M i)) ' I))$ 
  by simp
  also have  $\dots = ?rhs$ 
  by (subst sets_Sup_eq[where X= $\prod_E i \in I. \text{space } (M i)$ ]) (use assms in auto)
  finally show ?thesis .
qed

```

lemma

```

  shows space_PiM_empty:  $\text{space } (\text{PiM } \{\} \ M) = \{\lambda k. \text{undefined}\}$ 
  and sets_PiM_empty:  $\text{sets } (\text{PiM } \{\} \ M) = \{\{\}, \{\lambda k. \text{undefined}\}\}$ 
  by (simp_all add: space_PiM sets_PiM_single image_constant sigma_sets_empty_eq)

```

proposition *sets_PiM_sigma*:

assumes Ω_cover : $\bigwedge i. i \in I \implies \exists S \subseteq E. i. \text{countable } S \wedge \Omega i = \bigcup S$

assumes E : $\bigwedge i. i \in I \implies E i \subseteq Pow (\Omega i)$

assumes J : $\bigwedge j. j \in J \implies \text{finite } j \bigcup J = I$

defines $P \equiv \{ \{ f \in (\prod_{E i \in I. \Omega i}. \forall i \in j. f i \in A i) \mid A j. j \in J \wedge A \in Pi j E \}$

shows *sets* $(\prod_M i \in I. \text{sigma} (\Omega i) (E i)) = \text{sets} (\text{sigma} (\prod_{E i \in I. \Omega i} P)$

proof *cases*

assume $I = \{ \}$

with $\langle \bigcup J = I \rangle$ **have** $P = \{ \{ \lambda _. \text{undefined} \} \} \vee P = \{ \}$

by (*auto simp: P_def*)

with $\langle I = \{ \} \rangle$ **show** *?thesis*

by (*auto simp add: sets_PiM_empty sigma_sets_empty_eq*)

next

let $?F = \lambda i. \{ (\lambda x. x i) - ' A \cap Pi_E I \Omega \mid A. A \in E i \}$

assume $I \neq \{ \}$

then have *sets* $(Pi_M I (\lambda i. \text{sigma} (\Omega i) (E i))) =$

sets $(SUP i \in I. \text{vimage_algebra} (\prod_{E i \in I. \Omega i} (\lambda x. x i) (\text{sigma} (\Omega i) (E i))))$

by (*subst sets_PiM_eq_proj*) (*auto simp: space_measure_of_conv*)

also have $\dots = \text{sets} (SUP i \in I. \text{sigma} (Pi_E I \Omega) (?F i))$

using E **by** (*intro sets_SUP_cong arg_cong[where f=sets] vimage_algebra_sigma*)

auto

also have $\dots = \text{sets} (\text{sigma} (Pi_E I \Omega) (\bigcup i \in I. ?F i))$

using $\langle I \neq \{ \} \rangle$ **by** (*intro arg_cong[where f=sets] SUP_sigma_sigma*) *auto*

also have $\dots = \text{sets} (\text{sigma} (Pi_E I \Omega) P)$

proof (*intro arg_cong[where f=sets] sigma_eqI sigma_sets_eqI*)

show $(\bigcup i \in I. ?F i) \subseteq Pow (Pi_E I \Omega) P \subseteq Pow (Pi_E I \Omega)$

by (*auto simp: P_def*)

next

interpret P : *sigma_algebra* $\prod_{E i \in I. \Omega i}$ *sigma_sets* $(\prod_{E i \in I. \Omega i} P)$

by (*auto intro!: sigma_algebra_sigma_sets simp: P_def*)

fix Z **assume** $Z \in (\bigcup i \in I. ?F i)$

then obtain $i A$ **where** $i: i \in I A \in E i$ **and** Z_def : $Z = (\lambda x. x i) - ' A \cap Pi_E I \Omega$

by *auto*

from $\langle i \in I \rangle$ J **obtain** j **where** $j: i \in j j \in J j \subseteq I$ *finite* j

by *auto*

obtain S **where** $S: \bigwedge i. i \in j \implies S i \subseteq E i \bigwedge i. i \in j \implies \text{countable} (S i)$

$\bigwedge i. i \in j \implies \Omega i = \bigcup (S i)$

by (*metis subset_eq Omega_cover*) $\langle j \subseteq I \rangle$

define $A' n$ **where** $A' n = n(i := A)$ **for** n

then have $A'_i: \bigwedge n. A' n i = A$

by *simp*

{ fix n **assume** $n \in Pi_E (j - \{i\}) S$

then have $A' n \in Pi j E$

unfolding $Pi_E_def Pi_def$ **using** $S(1)$ **by** (*auto simp: A'_def*) $\langle A \in E i \rangle$

with $\langle j \in J \rangle$ **have** $\{ f \in Pi_E I \Omega. \forall i \in j. f i \in A' n i \} \in P$

by (*auto simp: P_def*) }


```

note A'_in_P = this

{ fix x assume x i ∈ A x ∈ Pi_E I Ω
  with S(3) ⟨j ⊆ I⟩ have ∀ i ∈ j. ∃ s ∈ S i. x i ∈ s
    by (auto simp: Pi_E_def Pi_def)
  then obtain s where s: ∧ i. i ∈ j ⇒ s i ∈ S i ∧ i. i ∈ j ⇒ x i ∈ s i
    by metis
  with ⟨x i ∈ A⟩ have ∃ n ∈ Pi_E (j - {i}) S. ∀ i ∈ j. x i ∈ A' n i
    by (intro ball[of _ restrict (s(i := A)) (j - {i})]) (auto simp: A'_def split:
if_splits) }
  then have Z = (⋃ n ∈ Pi_E (j - {i}) S. {f ∈ (Π_E i ∈ I. Ω i). ∀ i ∈ j. f i ∈ A' n i})
    unfolding Z_def
  by (auto simp add: set_eq_iff ball_conj_distrib ⟨i ∈ j⟩ A'_i dest: bspec[OF _
⟨i ∈ j⟩]
      cong: conj_cong)
  also have ... ∈ sigma_sets (Π_E i ∈ I. Ω i) P
    using ⟨finite j⟩ S(2)
  by (intro P.countable_UN' countable_Pi_E) (simp_all add: image_subset_iff
A'_in_P)
  finally show Z ∈ sigma_sets (Π_E i ∈ I. Ω i) P .
next
interpret F: sigma_algebra Π_E i ∈ I. Ω i sigma_sets (Π_E i ∈ I. Ω i) (⋃ i ∈ I.
?F i)
  by (auto intro!: sigma_algebra_sigma_sets)

fix b assume b ∈ P
then obtain A j where b: b = {f ∈ (Π_E i ∈ I. Ω i). ∀ i ∈ j. f i ∈ A i} j ∈ J A ∈
Pi j E
  by (auto simp: P_def)
show b ∈ sigma_sets (Pi_E I Ω) (⋃ i ∈ I. ?F i)
proof cases
  assume j = {}
  with b have b = (Π_E i ∈ I. Ω i)
    by auto
  then show ?thesis
    by blast
next
  assume j ≠ {}
  with J b(2,3) have eq: b = (⋂ i ∈ j. ((λx. x i) - ' A i ∩ Pi_E I Ω))
    unfolding b(1)
  by (auto simp: Pi_E_def Pi_def)
  show ?thesis
    unfolding eq using ⟨A ∈ Pi j E⟩ ⟨j ∈ J⟩ J(2)
  by (intro F.finite_INT J ⟨j ∈ J⟩ ⟨j ≠ {}⟩ sigma_sets.Basic) blast
qed
qed
finally show ?thesis .
qed

```

lemma *sets_PiM_in_sets*:

assumes *space*: $\text{space } N = (\prod_{E \ i \in I. \text{space } (M \ i)})$

assumes *sets*: $\bigwedge i \ A. \ i \in I \implies A \in \text{sets } (M \ i) \implies \{x \in \text{space } N. \ x \ i \in A\} \in \text{sets } N$

shows $\text{sets } (\prod_M \ i \in I. \ M \ i) \subseteq \text{sets } N$

unfolding *sets_PiM_single* *space[symmetric]*

by (*intro* *sets.sigma_sets_subset* *subsetI*) (*auto* *intro: sets*)

lemma *sets_PiM_cong[measurable_cong]*:

assumes $I = J \ \bigwedge i. \ i \in J \implies \text{sets } (M \ i) = \text{sets } (N \ i)$ **shows** $\text{sets } (PiM \ I \ M) = \text{sets } (PiM \ J \ N)$

using *assms* *sets_eq_imp_space_eq[OF assms(2)]* **by** (*simp* *add: sets_PiM_single* *cong: PiE_cong conj_cong*)

lemma *sets_PiM_I*:

assumes *finite* $J \ J \subseteq I \ \forall i \in J. \ E \ i \in \text{sets } (M \ i)$

shows $\text{prod_emb } I \ M \ J \ (\prod_{E \ j \in J. \ E \ j) \in \text{sets } (\prod_M \ i \in I. \ M \ i)$

proof *cases*

assume $J = \{\}$

then have $\text{prod_emb } I \ M \ J \ (\prod_{E \ j \in J. \ E \ j) = (\prod_{E \ j \in I. \ \text{space } (M \ j))$

by (*auto* *simp: prod_emb_def*)

then show *?thesis*

by (*auto* *simp* *add: sets_PiM_intro!: sigma_sets_top*)

next

assume $J \neq \{\}$ **with** *assms* **show** *?thesis*

by (*force* *simp* *add: sets_PiM_prod_algebra_def*)

qed

proposition *measurable_PiM*:

assumes *space*: $f \in \text{space } N \rightarrow (\prod_{E \ i \in I. \ \text{space } (M \ i)})$

assumes *sets*: $\bigwedge X \ J. \ J \neq \{\} \vee I = \{\} \implies \text{finite } J \implies J \subseteq I \implies (\bigwedge i. \ i \in J \implies X \ i \in \text{sets } (M \ i)) \implies$

$f \text{ --' } \text{prod_emb } I \ M \ J \ (PiE \ J \ X) \cap \text{space } N \in \text{sets } N$

shows $f \in \text{measurable } N \ (PiM \ I \ M)$

using *sets_PiM_prod_algebra_sets_into_space* *space*

proof (*rule* *measurable_sigma_sets*)

fix A **assume** $A \in \text{prod_algebra } I \ M$

from *prod_algebraE[OF this]* **obtain** $J \ X$ **where**

$A = \text{prod_emb } I \ M \ J \ (PiE \ J \ X)$

finite J

$J \neq \{\} \vee I = \{\}$

$J \subseteq I$

$\bigwedge i. \ i \in J \implies X \ i \in \text{sets } (M \ i)$

by *auto*

with *sets[of J X]* **show** $f \text{ --' } A \cap \text{space } N \in \text{sets } N$ **by** *auto*

qed

lemma *measurable_PiM_Collect*:

assumes *space*: $f \in \text{space } N \rightarrow (\prod_{E \ i \in I. \ \text{space } (M \ i)})$

```

assumes sets:  $\bigwedge X J. J \neq \{\} \vee I = \{\} \implies \text{finite } J \implies J \subseteq I \implies (\bigwedge i. i \in J \implies X i \in \text{sets } (M i)) \implies$ 
 $\{\omega \in \text{space } N. \forall i \in J. f \ \omega \ i \in X i\} \in \text{sets } N$ 
shows  $f \in \text{measurable } N \ (PiM \ I \ M)$ 
using sets_PiM_prod_algebra_sets_into_space_space
proof (rule measurable_sigma_sets)
fix  $A$  assume  $A \in \text{prod\_algebra } I \ M$ 
from prod_algebraE[OF this] obtain  $J \ X$ 
where  $X$ :
 $A = \text{prod\_emb } I \ M \ J \ (PiE \ J \ X)$ 
 $\text{finite } J$ 
 $J \neq \{\} \vee I = \{\}$ 
 $J \subseteq I$ 
 $\bigwedge i. i \in J \implies X i \in \text{sets } (M i)$ 
by auto
then have  $f - ' A \cap \text{space } N = \{\omega \in \text{space } N. \forall i \in J. f \ \omega \ i \in X i\}$ 
using space by (auto simp: prod_emb_def del: PiE_I)
also have  $\dots \in \text{sets } N$  using X(3,2,4,5) by (rule sets)
finally show  $f - ' A \cap \text{space } N \in \text{sets } N .$ 
qed

```

lemma measurable_PiM_single:

```

assumes space:  $f \in \text{space } N \rightarrow (\Pi_E \ i \in I. \text{space } (M i))$ 
assumes sets:  $\bigwedge A \ i. i \in I \implies A \in \text{sets } (M i) \implies \{\omega \in \text{space } N. f \ \omega \ i \in A\} \in \text{sets } N$ 
shows  $f \in \text{measurable } N \ (PiM \ I \ M)$ 
using sets_PiM_single
proof (rule measurable_sigma_sets)
fix  $A$  assume  $A \in \{\{f \in \Pi_E \ i \in I. \text{space } (M i). f \ i \in A\} \mid i \ A. i \in I \wedge A \in \text{sets } (M i)\}$ 
then obtain  $B \ i$  where  $A = \{f \in \Pi_E \ i \in I. \text{space } (M i). f \ i \in B\}$  and  $B: i \in I \ B \in \text{sets } (M i)$ 
by auto
with space have  $f - ' A \cap \text{space } N = \{\omega \in \text{space } N. f \ \omega \ i \in B\}$  by auto
also have  $\dots \in \text{sets } N$  using B by (rule sets)
finally show  $f - ' A \cap \text{space } N \in \text{sets } N .$ 
qed (auto simp: space)

```

lemma measurable_PiM_single':

```

assumes  $f: \bigwedge i. i \in I \implies f \ i \in \text{measurable } N \ (M i)$ 
and  $(\lambda \omega \ i. f \ i \ \omega) \in \text{space } N \rightarrow (\Pi_E \ i \in I. \text{space } (M i))$ 
shows  $(\lambda \omega \ i. f \ i \ \omega) \in \text{measurable } N \ (PiM \ I \ M)$ 
proof (rule measurable_PiM_single)
fix  $A \ i$  assume  $A: i \in I \ A \in \text{sets } (M i)$ 
then have  $\{\omega \in \text{space } N. f \ i \ \omega \in A\} = f \ i - ' A \cap \text{space } N$ 
by auto
then show  $\{\omega \in \text{space } N. f \ i \ \omega \in A\} \in \text{sets } N$ 
using A  $f$  by (auto intro!: measurable_sets)
qed fact

```

lemma *sets_PiM_I_finite*[*measurable*]:
assumes *finite I* **and** *sets: ($\bigwedge i. i \in I \implies E i \in \text{sets } (M i)$)*
shows $(\prod_{E j \in I. E j} \in \text{sets } (\prod_{M i \in I. M i})$
using *sets_PiM_I[of I I E M]* *sets.sets_into_space[OF sets]* $\langle \text{finite } I \rangle$ **sets by**
auto

lemma *measurable_component_singleton*[*measurable (raw)*]:
assumes $i \in I$ **shows** $(\lambda x. x i) \in \text{measurable } (Pi_M I M) (M i)$
proof (*unfold measurable_def, intro CollectI conjI ballI*)
fix *A* **assume** $A \in \text{sets } (M i)$
then have $(\lambda x. x i) - ' A \cap \text{space } (Pi_M I M) = \text{prod_emb } I M \{i\} (\prod_{E j \in \{i\}. A)$
using *sets.sets_into_space* $\langle i \in I \rangle$
by (*fastforce dest: Pi_mem simp: prod_emb_def space_PiM split: if_split_asm*)
then show $(\lambda x. x i) - ' A \cap \text{space } (Pi_M I M) \in \text{sets } (Pi_M I M)$
using $\langle A \in \text{sets } (M i) \rangle \langle i \in I \rangle$ **by** (*auto intro!: sets_PiM_I*)
qed (*use* $\langle i \in I \rangle$ **in** *auto simp: space_PiM*)

lemma *measurable_component_singleton'*[*measurable_dest*]:
assumes $f: f \in \text{measurable } N (Pi_M I M)$
assumes $g: g \in \text{measurable } L N$
assumes $i: i \in I$
shows $(\lambda x. (f (g x)) i) \in \text{measurable } L (M i)$
using *measurable_compose[OF measurable_compose[OF g f] measurable_component_singleton, OF i]* .

lemma *measurable_PiM_component_rev*:
 $i \in I \implies f \in \text{measurable } (M i) N \implies (\lambda x. f (x i)) \in \text{measurable } (Pi_M I M) N$
by *simp*

lemma *measurable_case_nat*[*measurable (raw)*]:
assumes [*measurable (raw)*]: $i = 0 \implies f \in \text{measurable } M N$
 $\bigwedge j. i = \text{Suc } j \implies (\lambda x. g x j) \in \text{measurable } M N$
shows $(\lambda x. \text{case_nat } (f x) (g x) i) \in \text{measurable } M N$
by (*cases i*) *simp_all*

lemma *measurable_case_nat'*[*measurable (raw)*]:
assumes *fg*[*measurable*]: $f \in \text{measurable } N M$ $g \in \text{measurable } N (\prod_{M i \in UNIV. M})$
shows $(\lambda x. \text{case_nat } (f x) (g x)) \in \text{measurable } N (\prod_{M i \in UNIV. M})$
using *fg*[*THEN measurable_space*]
by (*auto intro!: measurable_PiM_single' simp add: space_PiM PiE_iff split: nat.split*)

lemma *measurable_add_dim*[*measurable*]:
 $(\lambda(f, y). f(i := y)) \in \text{measurable } (Pi_M I M \otimes_M M i) (Pi_M (\text{insert } i I) M)$
(is $?f \in \text{measurable } ?P ?I)$
proof (*rule measurable_PiM_single*)

```

fix j A assume j: j ∈ insert i I and A: A ∈ sets (M j)
have {ω ∈ space ?P. (λ(f, x). fun_upd f i x) ω j ∈ A} =
  (if j = i then space (PiM I M) × A else ((λx. x j) ∘ fst) -‘ A ∩ space ?P)
  using sets.sets_into_space[OF A] by (auto simp add: space_pair_measure
space_PiM)
also have ... ∈ sets ?P
  using A j
  by (auto intro!: measurable_sets[OF measurable_comp, OF _ measurable_component_singleton])
finally show {ω ∈ space ?P. case_prod (λf. fun_upd f i) ω j ∈ A} ∈ sets ?P .
qed (auto simp: space_pair_measure space_PiM PiE_def)

```

proposition *measurable_fun_upd*:

```

assumes I: I = J ∪ {i}
assumes f[measurable]: f ∈ measurable N (PiM J M)
assumes h[measurable]: h ∈ measurable N (M i)
shows (λx. (f x) (i := h x)) ∈ measurable N (PiM I M)
proof (intro measurable_PiM_single)
  fix j assume j ∈ I then show (λω. ((f ω)(i := h ω)) j) ∈ measurable N (M j)
  unfolding I by (cases j = i) auto
next
  show (λx. (f x)(i := h x)) ∈ space N → (ΠE i ∈ I. space (M i))
  using I f[THEN measurable_space] h[THEN measurable_space]
  by (auto simp: space_PiM PiE_iff extensional_def)
qed

```

lemma *measurable_component_update*:

```

x ∈ space (PiM I M) ⇒ i ∉ I ⇒ (λv. x(i := v)) ∈ measurable (M i) (PiM
(insert i I) M)
by simp

```

lemma *measurable_merge[measurable]*:

```

merge I J ∈ measurable (PiM I M ⊗M PiM J M) (PiM (I ∪ J) M)
(is ?f ∈ measurable ?P ?U)
proof (rule measurable_PiM_single)
  fix i A assume A: A ∈ sets (M i) i ∈ I ∪ J
  then have {ω ∈ space ?P. merge I J ω i ∈ A} =
    (if i ∈ I then ((λx. x i) ∘ fst) -‘ A ∩ space ?P else ((λx. x i) ∘ snd) -‘ A ∩
space ?P)
  by (auto simp: merge_def)
  also have ... ∈ sets ?P
  using A
  by (auto intro!: measurable_sets[OF measurable_comp, OF _ measurable_component_singleton])
  finally show {ω ∈ space ?P. merge I J ω i ∈ A} ∈ sets ?P .
qed (auto simp: space_pair_measure space_PiM PiE_iff merge_def extensional_def)

```

lemma *measurable_restrict[measurable (raw)]*:

```

assumes X: ∧i. i ∈ I ⇒ X i ∈ measurable N (M i)
shows (λx. λi ∈ I. X i x) ∈ measurable N (PiM I M)
proof (rule measurable_PiM_single)

```

2322

```

fix A i assume A: i ∈ I A ∈ sets (M i)
then have {ω ∈ space N. (λi∈I. X i ω) i ∈ A} = X i -' A ∩ space N
  by auto
then show {ω ∈ space N. (λi∈I. X i ω) i ∈ A} ∈ sets N
  using A X by (auto intro!: measurable_sets)
next
show (λx. λi∈I. X i x) ∈ space N → (ΠE i∈I. space (M i))
  using X by (auto simp add: PiE_def dest: measurable_space)
qed

```

lemma *measurable_abs_UNIV*:

```

(∧n. (λω. f n ω) ∈ measurable M (N n)) ⇒ (λω n. f n ω) ∈ measurable M
(PiM UNIV N)
by (intro measurable_PiM_single) (auto dest: measurable_space)

```

lemma *measurable_restrict_subset*: $J \subseteq L \implies (\lambda f. \text{restrict } f J) \in \text{measurable}$
 $(Pi_M L M) (Pi_M J M)$

by (intro measurable_restrict measurable_component_singleton) auto

lemma *measurable_restrict_subset'*:

```

assumes J ⊆ L ∧x. x ∈ J ⇒ sets (M x) = sets (N x)
shows (λf. restrict f J) ∈ measurable (Pi_M L M) (Pi_M J N)
by (metis (no_types) assms measurable_cong_sets measurable_restrict_subset
sets_PiM_cong)

```

lemma *measurable_prod_emb*[intro, simp]:

$J \subseteq L \implies X \in \text{sets } (Pi_M J M) \implies \text{prod_emb } L M J X \in \text{sets } (Pi_M L M)$

unfolding prod_emb_def space_PiM[symmetric]

by (auto intro!: measurable_sets measurable_restrict measurable_component_singleton)

lemma *merge_in_prod_emb*:

assumes $y \in \text{space } (Pi_M I M)$ $x \in X$ **and** $X: X \in \text{sets } (Pi_M J M)$ **and** $J \subseteq I$

shows $\text{merge } J I (x, y) \in \text{prod_emb } I M J X$

using assms sets_into_space[OF X]

by (simp add: merge_def prod_emb_def subset_eq space_PiM PiE_def extensional_restrict Pi_iff

cong: if_cong restrict_cong)

(simp add: extensional_def)

lemma *prod_emb_eq_emptyD*:

assumes $J: J \subseteq I$ **and** $ne: \text{space } (Pi_M I M) \neq \{\}$ **and** $X: X \in \text{sets } (Pi_M J M)$

and $*$: $\text{prod_emb } I M J X = \{\}$

shows $X = \{\}$

proof safe

fix x **assume** $x \in X$

obtain ω **where** $\omega \in \text{space } (Pi_M I M)$

using ne **by** blast

from merge_in_prod_emb[OF this ⟨x∈X⟩ X J] * **show** $x \in \{\}$ **by** auto

qed

lemma *sets_in_Pi_aux*:

finite I $\implies (\bigwedge j. j \in I \implies \{x \in \text{space } (M j). x \in F j\} \in \text{sets } (M j)) \implies$
 $\{x \in \text{space } (PiM I M). x \in Pi I F\} \in \text{sets } (PiM I M)$
by (*simp add: subset_eq Pi_iff*)

lemma *sets_in_Pi*[*measurable (raw)*]:

finite I $\implies f \in \text{measurable } N (PiM I M) \implies$
 $(\bigwedge j. j \in I \implies \{x \in \text{space } (M j). x \in F j\} \in \text{sets } (M j)) \implies$
 $\text{Measurable.pred } N (\lambda x. f x \in Pi I F)$
unfolding *pred_def*
by (*rule measurable_sets_Collect[of f N PiM I M, OF _ sets_in_Pi_aux]*) *auto*

lemma *sets_in_extensional_aux*:

$\{x \in \text{space } (PiM I M). x \in \text{extensional } I\} \in \text{sets } (PiM I M)$
by (*smt (verit) PiE_iff mem_Collect_eq sets.top space_PiM subsetI subset_antisym*)

lemma *sets_in_extensional*[*measurable (raw)*]:

f $\in \text{measurable } N (PiM I M) \implies \text{Measurable.pred } N (\lambda x. f x \in \text{extensional } I)$
unfolding *pred_def*
by (*rule measurable_sets_Collect[of f N PiM I M, OF _ sets_in_extensional_aux]*)
auto

lemma *sets_PiM_I_countable*:

assumes *I*: *countable I* **and** *E*: $\bigwedge i. i \in I \implies E i \in \text{sets } (M i)$ **shows** $Pi_E I E \in \text{sets } (PiM I M)$

proof *cases*

assume $I \neq \{\}$

then have $Pi_E I E = (\bigcap i \in I. \text{prod_emb } I M \{i\} (Pi_E \{i\} E))$

using *E*[*THEN sets_sets_into_space*] **by** (*auto simp: PiE_iff prod_emb_def fun_eq_iff*)

also have $\dots \in \text{sets } (PiM I M)$

using $I \neq \{\}$ **by** (*simp add: E sets.countable_INT' sets_PiM_I subset_eq*)

finally show *?thesis* .

qed (*simp add: sets_PiM_empty*)

lemma *sets_PiM_D_countable*:

assumes *A*: $A \in PiM I M$

shows $\exists J \subseteq I. \exists X \in PiM J M. \text{countable } J \wedge A = \text{prod_emb } I M J X$

using *A*[*unfolded sets_PiM_single*]

proof *induction*

case (*Basic A*)

then obtain *i X* **where** $i \in I X \in \text{sets } (M i)$ **and** $A = \{f \in \Pi_E i \in I. \text{space } (M i). f i \in X\}$

by *auto*

then have $A = \text{prod_emb } I M \{i\} (\Pi_E _ \in \{i\}. X)$

by (*auto simp: prod_emb_def*)

then show *?case*

by (*intro exI[of _ {i}] conjI bexI[of _ $\Pi_E _ \in \{i\}. X$]*)

```

      (auto intro: countable_finite * sets_PiM_I_finite)
next
  case Empty then show ?case
    by (intro exI[of _ {}] conjI beX[of _ {}]) auto
next
  case (Compl A)
  then obtain J X where J  $\subseteq$  I X  $\in$  sets (PiM J M) countable J A = prod_emb
  I M J X
    by auto
  then show ?case
    by (intro exI[of _ J] beX[of _ space (PiM J M) - X] conjI)
      (auto simp add: space_PiM prod_emb_PiE intro!: sets_PiM_I_countable)
next
  case (Union K)
  obtain J X where J:  $\bigwedge i. J i \subseteq I \bigwedge i. \text{countable } (J i)$  and X:  $\bigwedge i. X i \in \text{sets}$ 
  (PiM (J i) M)
    and K:  $\bigwedge i. K i = \text{prod\_emb } I M (J i) (X i)$ 
    by (metis Union.IH)
  show ?case
  proof (intro exI beX conjI)
    show  $(\bigcup i. J i) \subseteq I$  countable  $(\bigcup i. J i)$ 
      using J by auto
    with J show  $\bigcup (K \text{ ' UNIV}) = \text{prod\_emb } I M (\bigcup i. J i) (\bigcup i. \text{prod\_emb } (\bigcup i.
  J i) M (J i) (X i))$ 
      by (simp add: K[abs_def] SUP_upper)
    qed(auto intro: X)
  qed

```

proposition *measure_eqI_PiM_finite:*

```

  assumes [simp]: finite I sets P = PiM I M sets Q = PiM I M
  assumes eq:  $\bigwedge A. (\bigwedge i. i \in I \implies A i \in \text{sets } (M i)) \implies P (PiE I A) = Q (PiE
  I A)$ 
  assumes A: range A  $\subseteq$  prod_algebra I M  $(\bigcup i. A i) = \text{space } (PiM I M) \bigwedge i::\text{nat.}$ 
  P (A i)  $\neq \infty$ 
  shows P = Q
proof (rule measure_eqI_generator_eq[OF Int_stable_prod_algebra prod_algebra_sets_into_space])
  show range A  $\subseteq$  prod_algebra I M  $(\bigcup i. A i) = (\prod_{E i \in I. \text{space } (M i)}) \bigwedge i. P
  (A i) \neq \infty$ 
    unfolding space_PiM[symmetric] by fact+
  fix X assume X  $\in$  prod_algebra I M
  then obtain J E where X: X = prod_emb I M J  $(\prod_{E j \in J. E j)$ 
    and J: finite J J  $\subseteq$  I  $\bigwedge j. j \in J \implies E j \in \text{sets } (M j)$ 
    by (force elim!: prod_algebraE)
  then show emeasure P X = emeasure Q X
    unfolding X by (subst (1 2) prod_emb_Pi) (auto simp: eq)
qed (simp_all add: sets_PiM)

```

proposition *measure_eqI_PiM_infinite:*

```

  assumes [simp]: sets P = PiM I M sets Q = PiM I M

```



```

assumes eq:  $\bigwedge A J. \text{finite } J \implies J \subseteq I \implies (\bigwedge i. i \in J \implies A \ i \in \text{sets } (M \ i))$ 
 $\implies$ 
   $P (\text{prod\_emb } I \ M \ J \ (Pi_E \ J \ A)) = Q (\text{prod\_emb } I \ M \ J \ (Pi_E \ J \ A))$ 
assumes A: finite_measure P
shows P = Q
proof (rule measure_eqI_generator_eq[OF Int_stable_prod_algebra prod_algebra_sets_into_space])
interpret finite_measure P by fact
define i where i = (SOME i. i  $\in$  I)
have i: I  $\neq$  {}  $\implies$  i  $\in$  I
  unfolding i_def by (rule someI_ex) auto
define A where A n =
  (if I = {} then prod_emb I M {} ( $\Pi_E \ i \in \{\}. \{\}$ ) else prod_emb I M {i} ( $\Pi_E \ i \in \{i\}. \text{space } (M \ i)$ ))
  for n :: nat
then show range A  $\subseteq$  prod_algebra I M
  using prod_algebraI[of {} I  $\lambda i. \text{space } (M \ i) \ M]$  by (auto intro!: prod_algebraI
i)
  have  $\bigwedge i. A \ i = \text{space } (PiM \ I \ M)$ 
  by (auto simp: prod_emb_def space_PiM PiE_iff A_def i_ex_in_conv[symmetric]
exI)
  then show ( $\bigcup i. A \ i$ ) = ( $\Pi_E \ i \in I. \text{space } (M \ i)$ )  $\wedge$  i. emeasure P (A i)  $\neq$   $\infty$ 
  by (auto simp: space_PiM)
next
fix X assume X: X  $\in$  prod_algebra I M
then obtain J E where X: X = prod_emb I M J ( $\Pi_E \ j \in J. E \ j$ )
  and J: finite J J  $\subseteq$  I  $\wedge$  j. j  $\in$  J  $\implies$  E j  $\in$  sets (M j)
  by (force elim!: prod_algebraE)
then show emeasure P X = emeasure Q X
  by (auto intro!: eq)
qed (auto simp: sets_PiM)

locale product_sigma_finite =
  fixes M :: 'i  $\Rightarrow$  'a measure
  assumes sigma_finite_measures:  $\bigwedge i. \text{sigma\_finite\_measure } (M \ i)$ 

sublocale product_sigma_finite  $\subseteq$  M?: sigma_finite_measure M i for i
  by (rule sigma_finite_measures)

locale finite_product_sigma_finite = product_sigma_finite M for M :: 'i  $\Rightarrow$  'a
measure +
  fixes I :: 'i set
  assumes finite_index: finite I

proposition (in finite_product_sigma_finite) sigma_finite_pairs:
   $\exists F :: 'i \Rightarrow \text{nat} \Rightarrow 'a \text{ set.}$ 
  ( $\forall i \in I. \text{range } (F \ i) \subseteq \text{sets } (M \ i)$ )  $\wedge$ 
  ( $\forall k. \forall i \in I. \text{emeasure } (M \ i) (F \ i \ k) \neq \infty$ )  $\wedge$  incseq ( $\lambda k. \Pi_E \ i \in I. F \ i \ k$ )  $\wedge$ 
  ( $\bigcup k. \Pi_E \ i \in I. F \ i \ k$ ) = space (PiM I M)
proof -

```

```

have  $\forall i::'i. \exists F::nat \Rightarrow 'a \text{ set. } range\ F \subseteq sets\ (M\ i) \wedge incseq\ F \wedge (\bigcup i. F\ i) =$ 
 $space\ (M\ i) \wedge (\forall k. emeasure\ (M\ i)\ (F\ k) \neq \infty)$ 
using M.sigma_finite_incseq by metis
then obtain  $F :: 'i \Rightarrow nat \Rightarrow 'a \text{ set}$ 
where  $\forall x. range\ (F\ x) \subseteq sets\ (M\ x) \wedge incseq\ (F\ x) \wedge \bigcup (range\ (F\ x)) = space$ 
 $(M\ x) \wedge (\forall k. emeasure\ (M\ x)\ (F\ x\ k) \neq \infty)$ 
by metis
then have  $F: \bigwedge i. range\ (F\ i) \subseteq sets\ (M\ i) \bigwedge i. incseq\ (F\ i) \bigwedge i. (\bigcup j. F\ i\ j) =$ 
 $space\ (M\ i) \bigwedge i\ k. emeasure\ (M\ i)\ (F\ i\ k) \neq \infty$ 
by auto
let  $?F = \lambda k. \Pi_E\ i \in I. F\ i\ k$ 
note space_PiM[simp]
show ?thesis
proof (intro exI[of _ F] conjI allI incseq_SucI set_eqI iffI ballI)
fix  $i$  show  $range\ (F\ i) \subseteq sets\ (M\ i)$  by fact
next
fix  $i\ k$  show  $emeasure\ (M\ i)\ (F\ i\ k) \neq \infty$  by fact
next
fix  $x$  assume  $x \in (\bigcup i. ?F\ i)$  with  $F(1)$  show  $x \in space\ (PiM\ I\ M)$ 
by (auto simp: PiE_def dest!: sets.sets_into_space)
next
fix  $f \in space\ (PiM\ I\ M)$ 
with  $Pi\_UN[OF\ finite\_index, of\ \lambda k\ i. F\ i\ k]\ F$ 
show  $f \in (\bigcup i. ?F\ i)$  by (auto simp: incseq_def PiE_def)
next
fix  $i$  show  $?F\ i \subseteq ?F\ (Suc\ i)$ 
using  $\langle \bigwedge i. incseq\ (F\ i) \rangle [THEN\ incseq\_SucD]$  by auto
qed
qed

```

```

lemma emeasure_PiM_empty[simp]:  $emeasure\ (PiM\ \{\}\ M)\ \{\lambda_.\ undefined\} = 1$ 
proof -
let  $?\mu = \lambda A. \text{if } A = \{\} \text{ then } 0 \text{ else } (1::ennreal)$ 
have  $emeasure\ (PiM\ \{\}\ M)\ (prod\_emb\ \{\}\ M\ \{\}\ (\Pi_E\ i \in \{\}. \{\})) = 1$ 
proof (subst emeasure_extend_measure_Pair[OF PiM_def])
show positive  $(PiM\ \{\}\ M)\ ?\mu$ 
by (auto simp: positive_def)
show countably_additive  $(PiM\ \{\}\ M)\ ?\mu$ 
by (rule sets.countably_additiveI_finite)
(auto simp: additive_def positive_def sets_PiM_empty space_PiM_empty
intro!:)
qed (auto simp: prod_emb_def)
also have  $(prod\_emb\ \{\}\ M\ \{\}\ (\Pi_E\ i \in \{\}. \{\})) = \{\lambda_.\ undefined\}$ 
by (auto simp: prod_emb_def)
finally show ?thesis
by simp
qed

```

```

lemma PiM_empty:  $PiM\ \{\}\ M = count\_space\ \{\lambda_.\ undefined\}$ 

```

by (rule measure_eqI) (auto simp add: sets_PiM_empty)

lemma (in product_sigma_finite) emeasure_PiM:

$finite\ I \implies (\bigwedge i. i \in I \implies A\ i \in sets\ (M\ i)) \implies emeasure\ (PiM\ I\ M)\ (PiE\ I\ A) = (\prod_{i \in I}. emeasure\ (M\ i)\ (A\ i))$

proof (induct I arbitrary: A rule: finite_induct)

case (insert i I)

interpret finite_product_sigma_finite M I **by** standard fact

have finite (insert i I) **using** <finite I> **by** auto

interpret I': finite_product_sigma_finite M insert i I **by** standard fact

let ?h = ($\lambda(f, y). f(i := y)$)

let ?P = $distr\ (PiM\ I\ M\ \otimes_M\ M\ i)\ (PiM\ (insert\ i\ I)\ M)\ ?h$

let ? μ = emeasure ?P

let ?I = $\{j \in insert\ i\ I. emeasure\ (M\ j)\ (space\ (M\ j)) \neq 1\}$

let ?f = $\lambda J\ E\ j. if\ j \in J\ then\ emeasure\ (M\ j)\ (E\ j)\ else\ emeasure\ (M\ j)\ (space\ (M\ j))$

have emeasure (PiM (insert i I) M) (prod_emb (insert i I) M (insert i I) (PiE (insert i I) A)) =

$(\prod_{i \in insert\ i\ I}. emeasure\ (M\ i)\ (A\ i))$

proof (subst emeasure_extend_measure_Pair[OF PiM_def])

fix J E **assume** $J \neq \{\}\ \vee\ insert\ i\ I = \{\} \wedge finite\ J \wedge J \subseteq insert\ i\ I \wedge E \in (\prod_{j \in J}. sets\ (M\ j))$

then have $J: J \neq \{\} \wedge finite\ J \wedge J \subseteq insert\ i\ I$ **and** $E: \forall j \in J. E\ j \in sets\ (M\ j)$

by auto

let ?p = prod_emb (insert i I) M J (PiE J E)

let ?p' = prod_emb I M (J - {i}) ($\Pi_E\ j \in J - \{i\}. E\ j$)

have ? μ ?p =

$emeasure\ (PiM\ I\ M\ \otimes_M\ (M\ i))\ (?h - ' ?p \cap space\ (PiM\ I\ M\ \otimes_M\ M\ i))$

by (intro emeasure_distr measurable_add_dim sets_PiM_I) fact+

also have ?h - ' ?p $\cap\ space\ (PiM\ I\ M\ \otimes_M\ M\ i) = ?p' \times (if\ i \in J\ then\ E\ i\ else\ space\ (M\ i))$

using J E [rule_format, THEN sets.sets_into_space]

by (force simp: space_pair_measure space_PiM prod_emb_iff PiE_def Pi_iff split: if_split_asm)

also have emeasure (PiM I M \otimes_M (M i)) (?p' \times (if i \in J then E i else space (M i))) =

$emeasure\ (PiM\ I\ M)\ ?p' * emeasure\ (M\ i)\ (if\ i \in J\ then\ (E\ i)\ else\ space\ (M\ i))$

i))

using J E **by** (intro M.emeasure_pair_measure_Times sets_PiM_I) auto

also have ?p' = ($\Pi_E\ j \in I. if\ j \in J - \{i\}\ then\ E\ j\ else\ space\ (M\ j)$)

using J E [rule_format, THEN sets.sets_into_space]

by (auto simp: prod_emb_iff PiE_def Pi_iff split: if_split_asm) blast+

also have emeasure (PiM I M) ($\Pi_E\ j \in I. if\ j \in J - \{i\}\ then\ E\ j\ else\ space\ (M\ j)$) =

$(\prod_{j \in I. if\ j \in J - \{i\}\ then\ emeasure\ (M\ j)\ (E\ j)\ else\ emeasure\ (M\ j)\ (space\ (M\ j))$)

using E **by** (subst insert) (auto intro!: prod.cong)

also have $(\prod_{j \in I}. \text{if } j \in J - \{i\} \text{ then } \text{emeasure } (M j) (E j) \text{ else } \text{emeasure } (M j) (\text{space } (M j))) *$
 $\text{emeasure } (M i) (\text{if } i \in J \text{ then } E i \text{ else } \text{space } (M i)) = (\prod_{j \in \text{insert } i I}. ?f J E j)$
using insert by $(\text{auto simp: mult.commute intro!: arg_cong2}[\text{where } f=(*)])$
 prod.cong
also have $\dots = (\prod_{j \in J \cup ?I}. ?f J E j)$
using insert(1,2) J E by $(\text{intro prod.mono_neutral_right}) \text{ auto}$
finally show $? \mu ?p = \dots$.

show $\text{prod_emb } (\text{insert } i I) M J (Pi_E J E) \in \text{Pow } (\prod_{E} i \in \text{insert } i I. \text{space } (M i))$
using $J E[\text{rule_format, THEN sets.sets_into_space}]$ **by** $(\text{auto simp: prod_emb_iff } Pi_E_def)$
next
show $\text{positive } (\text{sets } (Pi_M (\text{insert } i I) M)) ? \mu \text{ countably_additive } (\text{sets } (Pi_M (\text{insert } i I) M)) ? \mu$
using $\text{emeasure_positive}[\text{of } ?P] \text{ emeasure_countably_additive}[\text{of } ?P]$ **by** simp_all
next
show $(\text{insert } i I \neq \{\}) \vee \text{insert } i I = \{\} \wedge \text{finite } (\text{insert } i I) \wedge \text{insert } i I \subseteq \text{insert } i I \wedge A \in (\prod_{j \in \text{insert } i I}. \text{sets } (M j))$
using insert by auto
qed $(\text{auto intro!: prod.cong})$
with insert show $?case$
by $(\text{subst } (asm) \text{prod_emb_PiE_same_index}) (\text{auto intro!: sets.sets_into_space})$
qed simp

lemma $(\text{in product_sigma_finite}) PiM_eqI:$
assumes $I[\text{simp}]: \text{finite } I$ **and** $P: \text{sets } P = PiM I M$
assumes $eq: \bigwedge A. (\bigwedge i. i \in I \implies A i \in \text{sets } (M i)) \implies P (Pi_E I A) = (\prod_{i \in I}. \text{emeasure } (M i) (A i))$
shows $P = PiM I M$
proof –
interpret $\text{finite_product_sigma_finite } M I$
by standard fact
from $\text{sigma_finite_pairs}$ **obtain** C **where** $C:$
 $\forall i \in I. \text{range } (C i) \subseteq \text{sets } (M i) \forall k. \forall i \in I. \text{emeasure } (M i) (C i k) \neq \infty$
 $\text{incseq } (\lambda k. \prod_{E} i \in I. C i k) (\bigcup k. \prod_{E} i \in I. C i k) = \text{space } (Pi_M I M)$
by auto
show $?thesis$
proof $(\text{rule measure_eqI_PiM_finite}[\text{OF } I \text{ refl } P, \text{symmetric}])$
show $(\bigwedge i. i \in I \implies A i \in \text{sets } (M i)) \implies (Pi_M I M) (Pi_E I A) = P (Pi_E I A)$ **for** A
by $(\text{simp add: eq emeasure_PiM})$
define $A n = (\prod_{E} i \in I. C i n)$ **for** n
with C **show** $\text{range } A \subseteq \text{prod_algebra } I M \wedge i. \text{emeasure } (Pi_M I M) (A i) \neq \infty$
 $(\bigcup i. A i) = \text{space } (Pi_M I M)$
by $(\text{auto intro!: prod_algebraI_finite simp: emeasure_PiM subset_eq en-})$

nreal_prod_eq_top)

qed
qed

lemma (*in product_sigma_finite*) *sigma_finite*:

assumes *finite I*

shows *sigma_finite_measure (PiM I M)*

proof

interpret *finite_product_sigma_finite M I* **by** *standard fact*

obtain *F* **where** $F: \bigwedge j. \text{countable } (F j) \wedge \bigwedge j. f \in F j \implies f \in \text{sets } (M j)$

$\bigwedge j. f \in F j \implies \text{emeasure } (M j) f \neq \infty$ **and**

in_space: $\bigwedge j. \text{space } (M j) = \bigcup (F j)$

using *sigma_finite_countable* **by** (*metis subset_eq*)

moreover **have** $(\bigcup (PiE I ' PiE I F)) = \text{space } (PiM I M)$

using *in_space* **by** (*auto simp: space_PiM PiE_iff intro!: PiE_choice [THEN iffD1]*)

ultimately **show** $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (PiM I M) \wedge \bigcup A = \text{space } (PiM I M) \wedge (\forall a \in A. \text{emeasure } (PiM I M) a \neq \infty)$

by (*intro exI [of _ PiE I ' PiE I F]*)

(*auto intro!: countable_PiE sets_PiM_I_finite*

simp: PiE_iff emeasure_PiM finite_index ennreal_prod_eq_top)

qed

sublocale *finite_product_sigma_finite* \subseteq *sigma_finite_measure PiM I M*

using *sigma_finite [OF finite_index]* .

lemma (*in finite_product_sigma_finite*) *measure_times*:

$(\bigwedge i. i \in I \implies A i \in \text{sets } (M i)) \implies \text{emeasure } (PiM I M) (PiE I A) = (\prod_{i \in I} \text{emeasure } (M i) (A i))$

emeasure (M i) (A i))

using *emeasure_PiM [OF finite_index]* **by** *auto*

lemma (*in product_sigma_finite*) *nn_integral_empty*:

$0 \leq f (\lambda k. \text{undefined}) \implies \text{integral}^N (PiM \{ \} M) f = f (\lambda k. \text{undefined})$

by (*simp add: PiM_empty nn_integral_count_space_finite max.absorb2*)

lemma (*in product_sigma_finite*) *distr_merge*:

assumes *IJ [simp]: I \cap J = { }* **and** *fin: finite I finite J*

shows $\text{distr } (PiM I M \otimes_M PiM J M) (PiM (I \cup J) M) (\text{merge } I J) = PiM (I \cup J) M$

(*is ?D = ?P*)

proof (*rule PiM_eqI*)

interpret *I: finite_product_sigma_finite M I* **by** *standard fact*

interpret *J: finite_product_sigma_finite M J* **by** *standard fact*

fix *A* **assume** $A: \bigwedge i. i \in I \cup J \implies A i \in \text{sets } (M i)$

have $*$: $(\text{merge } I J - ' PiE (I \cup J) A \cap \text{space } (PiM I M \otimes_M PiM J M)) = PiE I A \times PiE J A$

using *A [THEN sets_into_space]* **by** (*auto simp: space_PiM space_pair_measure*)

from *A* **fin** **show** $\text{emeasure } (\text{distr } (PiM I M \otimes_M PiM J M) (PiM (I \cup J) M))$

$(\text{merge } I J) (P_{i_E} (I \cup J) A) =$
 $(\prod_{i \in I \cup J} \text{emeasure } (M i) (A i))$
by (*subst emeasure_distr*)
 $(\text{auto simp: } * J.\text{emeasure_pair_measure_Times } I.\text{measure_times } J.\text{measure_times}$
 $\text{prod.union_disjoint})$
qed (*use fin in simp_all*)

proposition (*in product_sigma_finite*) *product_nn_integral_fold*:

assumes $IJ: I \cap J = \{\}$ *finite I finite J*
and $f[\text{measurable}]: f \in \text{borel_measurable } (P_{i_M} (I \cup J) M)$
shows $\text{integral}^N (P_{i_M} (I \cup J) M) f = (\int^+ x. (\int^+ y. f (\text{merge } I J (x, y))) \partial(P_{i_M}$
 $J M)) \partial(P_{i_M} I M)$
(is ?lhs = ?rhs)

proof –

interpret $I: \text{finite_product_sigma_finite } M I$ **by** *standard fact*
interpret $J: \text{finite_product_sigma_finite } M J$ **by** *standard fact*
interpret $P: \text{pair_sigma_finite } P_{i_M} I M P_{i_M} J M$ **by** *standard*
have $P_borel: (\lambda x. f (\text{merge } I J x)) \in \text{borel_measurable } (P_{i_M} I M \otimes_M P_{i_M} J$
 $M)$
using *measurable_comp[OF measurable_merge f]* **by** (*simp add: comp_def*)
have $?lhs = \text{integral}^N (\text{distr } (P_{i_M} I M \otimes_M P_{i_M} J M) (P_{i_M} (I \cup J) M) (\text{merge}$
 $I J)) f$
by (*simp add: I.finite_index J.finite_index assms(1) distr_merge*)
also have $\dots = \int^+ x. f (\text{merge } I J x) \partial(P_{i_M} I M \otimes_M P_{i_M} J M)$
by (*simp add: nn_integral_distr*)
also have $\dots = ?rhs$
using $P.\text{Fubini } P.\text{nn_integral_snd}$ **by** *force*
finally show *?thesis* .
qed

lemma (*in product_sigma_finite*) *distr_singleton*:

$\text{distr } (P_{i_M} \{i\} M) (M i) (\lambda x. x i) = M i$ *(is ?D = _)*
proof (*intro measure_eqI[symmetric]*)
interpret $I: \text{finite_product_sigma_finite } M \{i\}$ **by** *standard simp*
fix A **assume** $A: A \in \text{sets } (M i)$
then have $(\lambda x. x i) -' A \cap \text{space } (P_{i_M} \{i\} M) = (\prod_{E} i \in \{i\}. A)$
using *sets.sets_into_space* **by** (*auto simp: space_PiM*)
then show $\text{emeasure } (M i) A = \text{emeasure } ?D A$
using $A I.\text{measure_times}[of \lambda_. A]$
by (*simp add: emeasure_distr measurable_component_singleton*)
qed *simp*

lemma (*in product_sigma_finite*) *product_nn_integral_singleton*:

assumes $f: f \in \text{borel_measurable } (M i)$
shows $\text{integral}^N (P_{i_M} \{i\} M) (\lambda x. f (x i)) = \text{integral}^N (M i) f$
proof –
interpret $I: \text{finite_product_sigma_finite } M \{i\}$ **by** *standard simp*
from f **show** *?thesis*
by (*metis distr_singleton insert_iff measurable_component_singleton nn_integral_distr*)

qed

proposition (in *product_sigma_finite*) *product_nn_integral_insert*:

assumes $I[simp]$: *finite* I $i \notin I$
and f : $f \in \text{borel_measurable } (Pi_M \text{ (insert } i \text{ I) } M)$
shows $\text{integral}^N (Pi_M \text{ (insert } i \text{ I) } M) f = (\int^+ x. (\int^+ y. f (x(i := y))) \partial(M i)) \partial(Pi_M I M)$
proof –
interpret I : *finite_product_sigma_finite* $M I$ **by** *standard auto*
interpret i : *finite_product_sigma_finite* $M \{i\}$ **by** *standard auto*
have IJ : $I \cap \{i\} = \{\}$ **and** $insert$: $I \cup \{i\} = insert\ i\ I$
using f **by** *auto*
show *?thesis*
unfolding *product_nn_integral_fold[OF IJ, unfolded insert, OF I(1) i.finite_index f]*
proof (*rule nn_integral_cong, subst product_nn_integral_singleton[symmetric]*)
fix x **assume** x : $x \in \text{space } (Pi_M I M)$
let $?f = \lambda y. f (x(i := y))$
show $?f \in \text{borel_measurable } (M i)$
using *measurable_comp[OF measurable_component_update f, OF x <i <notin I>]*
unfolding *comp_def* .
show $(\int^+ y. f (\text{merge } I \{i\} (x, y))) \partial Pi_M \{i\} M = (\int^+ y. f (x(i := y))) \partial Pi_M \{i\} M$
using x
by (*auto intro!: nn_integral_cong arg_cong[where f=f]*)
simp add: space_PiM extensional_def PiE_def

qed

qed

lemma (in *product_sigma_finite*) *product_nn_integral_insert_rev*:

assumes $I[simp]$: *finite* I $i \notin I$
and $[measurable]$: $f \in \text{borel_measurable } (Pi_M \text{ (insert } i \text{ I) } M)$
shows $\text{integral}^N (Pi_M \text{ (insert } i \text{ I) } M) f = (\int^+ y. (\int^+ x. f (x(i := y))) \partial(Pi_M I M)) \partial(M i)$
apply (*subst product_nn_integral_insert[OF assms]*)
apply (*rule pair_sigma_finite.Fubini'*)
apply (*simp add: local_sigma_finite pair_sigma_finite.intro sigma_finite_measures*)
apply *measurable*
done

lemma (in *product_sigma_finite*) *product_nn_integral_prod*:

assumes *finite* $I \wedge i. i \in I \implies f\ i \in \text{borel_measurable } (M i)$
shows $(\int^+ x. (\prod_{i \in I. f\ i (x\ i)}) \partial Pi_M I M) = (\prod_{i \in I. \text{integral}^N (M i) (f\ i))$
using *assms proof (induction I)*
case (*insert i I*)
note *insert.premis[measurable]*
note $\langle \text{finite } I \rangle$ [*intro, simp*]
interpret I : *finite_product_sigma_finite* $M I$ **by** *standard auto*
have $*$: $\bigwedge x\ y. (\prod_{j \in I. f\ j \text{ (if } j = i \text{ then } y \text{ else } x\ j)}) = (\prod_{j \in I. f\ j (x\ j)})$

```

    using insert by (auto intro!: prod.cong)
  have prod:  $\bigwedge J. J \subseteq \text{insert } i \ I \implies (\lambda x. (\prod_{i \in J}. f \ i \ (x \ i))) \in \text{borel\_measurable}$ 
    (PiM J M)
    using sets.sets_into_space insert
  by (intro borel_measurable_prod_ennreal
      measurable_comp[OF measurable_component_singleton, unfolded
    comp_def])
    auto
  then show ?case
    using product_nn_integral_insert[OF insert(1,2)]
  by (simp add: insert(2-) * nn_integral_multc nn_integral_cmult)
qed (simp add: space_PiM)

```

proposition (in *product_sigma_finite*) *product_nn_integral_pair*:

```

  assumes [measurable]: case_prod f  $\in \text{borel\_measurable}$  (M x  $\otimes_M$  M y)
  assumes xy:  $x \neq y$ 
  shows  $(\int^{+\sigma}. f \ (\sigma \ x) \ (\sigma \ y) \ \partial \text{PiM} \ \{x, y\} \ M) = (\int^{+z}. f \ (\text{fst } z) \ (\text{snd } z) \ \partial(M \ x$ 
 $\otimes_M \ M \ y))$ 
proof -
  interpret psm: pair_sigma_finite M x M y
  unfolding pair_sigma_finite_def using sigma_finite_measures by simp_all
  have  $\{x, y\} = \{y, x\}$  by auto
  also have  $(\int^{+\sigma}. f \ (\sigma \ x) \ (\sigma \ y) \ \partial \text{PiM} \ \{y, x\} \ M) = (\int^{+y}. \int^{+\sigma}. f \ (\sigma \ x) \ y \ \partial \text{PiM}$ 
 $\{x\} \ M \ \partial M \ y)$ 
    using xy by (subst product_nn_integral_insert_rev) simp_all
  also have ... =  $(\int^{+y}. \int^{+x}. f \ x \ y \ \partial M \ x \ \partial M \ y)$ 
    by (intro nn_integral_cong, subst product_nn_integral_singleton) simp_all
  also have ... =  $(\int^{+z}. f \ (\text{fst } z) \ (\text{snd } z) \ \partial(M \ x \otimes_M \ M \ y))$ 
    by (subst psm.nn_integral_snd[symmetric]) simp_all
  finally show ?thesis .
qed

```

lemma (in *product_sigma_finite*) *distr_component*:

```

  distr (M i) (PiM {i} M)  $(\lambda x. \lambda i \in \{i\}. x) = \text{PiM} \ \{i\} \ M$  (is ?D = ?P)
proof (intro PiM_eqI)
  fix A assume A:  $\bigwedge ia. ia \in \{i\} \implies A \ ia \in \text{sets} \ (M \ ia)$ 
  then have  $(\lambda x. \lambda i \in \{i\}. x) -' \text{PiE} \ \{i\} \ A \cap \text{space} \ (M \ i) = A \ i$ 
    by (fastforce dest: sets.sets_into_space)
  with A show emeasure (distr (M i) (PiM {i} M)  $(\lambda x. \lambda i \in \{i\}. x))$  (PiE {i} A)
    =  $(\prod_{i \in \{i\}}. \text{emeasure} \ (M \ i) \ (A \ i))$ 
    by (subst emeasure_distr) (auto intro!: sets_PiM_I_finite measurable_restrict)
qed simp_all

```

lemma (in *product_sigma_finite*)

```

  assumes IJ:  $I \cap J = \{\}$  finite I finite J and A:  $A \in \text{sets} \ (\text{PiM} \ (I \cup J) \ M)$ 
  shows emeasure_fold_integral:
    emeasure (PiM (I  $\cup$  J) M) A =  $(\int^{+x}. \text{emeasure} \ (\text{PiM} \ J \ M) \ ((\lambda y. \text{merge} \ I \ J$ 
 $(x, y)) -' A \cap \text{space} \ (\text{PiM} \ J \ M)) \ \partial \text{PiM} \ I \ M)$ 
    (is ?lhs = ?rhs)

```



```

and emeasure_fold_measurable:
  ( $\lambda x. \text{emeasure } (Pi_M J M) ((\lambda y. \text{merge } I J (x, y)) - 'A \cap \text{space } (Pi_M J M))$ )
 $\in \text{borel\_measurable } (Pi_M I M)$  (is ?B)
proof -
  interpret I: finite_product_sigma_finite M I by standard fact
  interpret J: finite_product_sigma_finite M J by standard fact
  interpret IJ: pair_sigma_finite Pi_M I M Pi_M J M ..
  have merge:  $\text{merge } I J - 'A \cap \text{space } (Pi_M I M \otimes_M Pi_M J M) \in \text{sets } (Pi_M I$ 
 $M \otimes_M Pi_M J M)$ 
    by (intro measurable_sets[OF _ A] measurable_merge assms)
  have ?lhs =  $\text{emeasure } (\text{distr } (Pi_M I M \otimes_M Pi_M J M) (Pi_M (I \cup J) M) (\text{merge}$ 
 $I J)) A$ 
    by (simp add: I.finite_index J.finite_index assms(1) distr_merge)
  also have ... =  $\text{emeasure } (Pi_M I M \otimes_M Pi_M J M) (\text{merge } I J - 'A \cap \text{space}$ 
 $(Pi_M I M \otimes_M Pi_M J M))$ 
    by (meson A emeasure_distr measurable_merge)
  also have ... = ?rhs
    apply (subst J.emeasure_pair_measure_alt[OF merge])
    by (auto intro!: nn_integral_cong arg_cong2[where f=emeasure] simp: space_pair_measure)
  finally show ?lhs = ?rhs .

show ?B
  using IJ.measurable_emeasure_Pair1[OF merge]
  by (simp add: vimage_comp comp_def space_pair_measure cong: measurable_cong)
qed

lemma sets_Collect_single:
   $i \in I \implies A \in \text{sets } (M i) \implies \{ x \in \text{space } (Pi_M I M). x i \in A \} \in \text{sets } (Pi_M I$ 
 $M)$ 
  by simp

lemma pair_measure_eq_distr_PiM:
  fixes M1 :: 'a measure and M2 :: 'a measure
  assumes sigma_finite_measure M1 sigma_finite_measure M2
  shows  $(M1 \otimes_M M2) = \text{distr } (Pi_M UNIV (case\_bool M1 M2)) (M1 \otimes_M M2)$ 
 $(\lambda x. (x \text{ True}, x \text{ False}))$ 
  (is ?P = ?D)
proof (rule pair_measure_eqI[OF assms])
  interpret B: product_sigma_finite case_bool M1 M2
  unfolding product_sigma_finite_def using assms by (auto split: bool.split)
  let ?B =  $Pi_M UNIV (case\_bool M1 M2)$ 

  have [simp]:  $\text{fst} \circ (\lambda x. (x \text{ True}, x \text{ False})) = (\lambda x. x \text{ True}) \text{snd} \circ (\lambda x. (x \text{ True}, x$ 
 $\text{ False})) = (\lambda x. x \text{ False})$ 
    by auto
  fix A B assume A:  $A \in \text{sets } M1$  and B:  $B \in \text{sets } M2$ 
  have  $\text{emeasure } M1 A * \text{emeasure } M2 B = (\prod_{i \in UNIV. \text{emeasure } (case\_bool$ 
 $M1 M2 i) (case\_bool A B i)}$ )
    by (simp add: UNIV_bool ac_simps)

```

```

also have ... = emeasure ?B (PiE UNIV (case_bool A B))
  using A B by (subst B.emeasure_PiM) (auto split: bool.split)
also have PiE UNIV (case_bool A B) = (λx. (x True, x False)) -‘ (A × B) ∩
space ?B
  using A[THEN sets.sets_into_space] B[THEN sets.sets_into_space]
by (auto simp: PiE_iff all_bool_eq space_PiM split: bool.split)
finally show emeasure M1 A * emeasure M2 B = emeasure ?D (A × B)
  using A B
    measurable_component_singleton[of True UNIV case_bool M1 M2]
    measurable_component_singleton[of False UNIV case_bool M1 M2]
by (subst emeasure_distr) (auto simp: measurable_pair_iff)
qed simp

```

```

lemma infprod_in_sets[intro]:
  fixes E :: nat ⇒ 'a set assumes E: ∧i. E i ∈ sets (M i)
  shows Pi UNIV E ∈ sets (ΠM i∈UNIV::nat set. M i)
proof -
  have Pi UNIV E = (∩i. prod_emb UNIV M {..i} (ΠE j∈{..i}. E j))
    using E E[THEN sets.sets_into_space]
  by (auto simp: prod_emb_def Pi_iff extensional_def)
  with E show ?thesis by auto
qed

```

7.7.3 Measurability

There are two natural sigma-algebras on a product space: the borel sigma algebra, generated by open sets in the product, and the product sigma algebra, countably generated by products of measurable sets along finitely many coordinates. The second one is defined and studied in `Finite_Product_Measure.thy`.

These sigma-algebras share a lot of natural properties (measurability of coordinates, for instance), but there is a fundamental difference: open sets are generated by arbitrary unions, not only countable ones, so typically many open sets will not be measurable with respect to the product sigma algebra (while all sets in the product sigma algebra are borel). The two sigma algebras coincide only when everything is countable (i.e., the product is countable, and the borel sigma algebra in the factor is countably generated).

In this paragraph, we develop basic measurability properties for the borel sigma algebra, and compare it with the product sigma algebra as explained above.

```

lemma measurable_product_coordinates [measurable (raw)]:
  (λx. x i) ∈ measurable borel borel
by (rule borel_measurable_continuous_onI[OF continuous_on_product_coordinates])

```

```

lemma measurable_product_then_coordinatewise:
  fixes f::'a ⇒ 'b ⇒ ('c::topological_space)
  assumes [measurable]: f ∈ borel_measurable M
  shows (λx. f x i) ∈ borel_measurable M

```

proof –
have $(\lambda x. f x i) = (\lambda y. y i)$ *of*
unfolding *comp_def* **by** *auto*
then show *?thesis* **by** *simp*
qed

To compare the Borel sigma algebra with the product sigma algebra, we give a presentation of the product sigma algebra that is more similar to the one we used above for the product topology.

lemma *sets_PiM_finite*:

$sets (Pi_M I M) = sigma_sets (\Pi_E i \in I. space (M i))$
 $\{(\Pi_E i \in I. X i) \mid X. (\forall i. X i \in sets (M i)) \wedge finite \{i. X i \neq space (M i)\}\}$

proof

have $\{(\Pi_E i \in I. X i) \mid X. (\forall i. X i \in sets (M i)) \wedge finite \{i. X i \neq space (M i)\}\}$
 $\subseteq sets (Pi_M I M)$

proof *clarify*

fix X **assume** $H: \forall i. X i \in sets (M i)$ *finite* $\{i. X i \neq space (M i)\}$

then have $*$: $X i \in sets (M i)$ **for** i **by** *simp*

define J **where** $J = \{i \in I. X i \neq space (M i)\}$

have *finite* J $J \subseteq I$ **unfolding** *J_def* **using** H **by** *auto*

define Y **where** $Y = (\Pi_E j \in J. X j)$

have *prod_emb* $I M J Y \in sets (Pi_M I M)$

unfolding *Y_def* **apply** (*rule* *sets_PiM_I*) **using** $\langle finite J \rangle \langle J \subseteq I \rangle *$ **by**
auto

moreover have *prod_emb* $I M J Y = (\Pi_E i \in I. X i)$

unfolding *prod_emb_def* *Y_def* *J_def* **using** H *sets.sets_into_space[OF *]*

by (*auto simp add: PiE_iff, blast*)

ultimately show $Pi_E I X \in sets (Pi_M I M)$ **by** *simp*

qed

then show $sigma_sets (\Pi_E i \in I. space (M i)) \{(\Pi_E i \in I. X i) \mid X. (\forall i. X i \in sets (M i)) \wedge finite \{i. X i \neq space (M i)\}\}$

$\subseteq sets (Pi_M I M)$

by (*metis (mono_tags, lifting) sets.sigma_sets_subset' sets.top_space_PiM*)

have $*$: $\exists X. \{f. (\forall i \in I. f i \in space (M i)) \wedge f \in extensional I \wedge f i \in A\} = Pi_E I X$
 $I X \wedge$

$(\forall i. X i \in sets (M i)) \wedge finite \{i. X i \neq space (M i)\}$

if $i \in I$ $A \in sets (M i)$ **for** i A

proof –

define X **where** $X = (\lambda j. if j = i then A else space (M j))$

have $\{f. (\forall i \in I. f i \in space (M i)) \wedge f \in extensional I \wedge f i \in A\} = Pi_E I X$

unfolding *X_def* **using** *sets.sets_into_space[OF <A \in sets (M i)> <i \in I>*

by (*auto simp add: PiE_iff extensional_def, metis subsetCE, metis*)

moreover have $X j \in sets (M j)$ **for** j

unfolding *X_def* **using** $\langle A \in sets (M i) \rangle$ **by** *auto*

moreover have *finite* $\{j. X j \neq space (M j)\}$

unfolding *X_def* **by** *simp*

ultimately show *?thesis* **by** *auto*

qed

show $\text{sets } (Pi_M I M) \subseteq \text{sigma_sets } (\Pi_{E i \in I}. \text{space } (M i)) \{(\Pi_{E i \in I}. X i) \mid X. (\forall i. X i \in \text{sets } (M i)) \wedge \text{finite } \{i. X i \neq \text{space } (M i)\}\}$
unfolding sets_PiM_single
by $(\text{intro sigma_sets_mono'}) (\text{auto simp add: PiE_iff } *)$
qed

lemma $\text{sets_PiM_subset_borel}$:

$\text{sets } (Pi_M UNIV (\lambda_. \text{borel})) \subseteq \text{sets borel}$

proof –

have $*: Pi_E UNIV X \in \text{sets borel}$ **if** $[\text{measurable}]: \bigwedge i. X i \in \text{sets borel}$ **finite** $\{i. X i \neq UNIV\}$ **for** $X::'a \Rightarrow 'b$ **set**

proof –

define I **where** $I = \{i. X i \neq UNIV\}$

have $\text{finite } I$ **unfolding** I_def **using** that **by** simp

have $Pi_E UNIV X = (\bigcap i \in I. (\lambda x. x i) - (X i) \cap \text{space borel}) \cap \text{space borel}$

unfolding I_def **by** auto

also have $\dots \in \text{sets borel}$

using that $\langle \text{finite } I \rangle$ **by** measurable

finally show $?thesis$ **by** simp

qed

then have $\{(\Pi_{E i \in UNIV}. X i) \mid X::('a \Rightarrow 'b \text{ set}). (\forall i. X i \in \text{sets borel}) \wedge \text{finite } \{i. X i \neq \text{space borel}\}\} \subseteq \text{sets borel}$

by auto

then show $?thesis$ **unfolding** $\text{sets_PiM_finite_space_borel}$

by $(\text{simp add: } * \text{ sets.sigma_sets_subset'})$

qed

proposition $\text{sets_PiM_equal_borel}$:

$\text{sets } (Pi_M UNIV (\lambda i::('a::\text{countable}). \text{borel}::('b::\text{second_countable_topology_measure}))) = \text{sets borel}$

proof

obtain $K::('a \Rightarrow 'b)$ **set set** **where** $K: \text{topological_basis } K$ **countable** K

$\bigwedge k. k \in K \implies \exists X. (k = Pi_E UNIV X) \wedge (\forall i. \text{open } (X i)) \wedge \text{finite } \{i. X i \neq UNIV\}$

using $\text{product_topology_countable_basis}$ **by** fast

have $*: k \in \text{sets } (Pi_M UNIV (\lambda_. \text{borel}))$ **if** $k \in K$ **for** k

proof –

obtain X **where** $H: k = Pi_E UNIV X \wedge i. \text{open } (X i)$ **finite** $\{i. X i \neq UNIV\}$

using $K(3)[OF \langle k \in K \rangle]$ **by** blast

show $?thesis$ **unfolding** $H(1)$ $\text{sets_PiM_finite_space_borel}$ **using** $\text{borel_open}[OF H(2)] H(3)$ **by** auto

qed

have $**:$ $U \in \text{sets } (Pi_M UNIV (\lambda_. \text{borel}))$ **if** $\text{open } U$ **for** $U::('a \Rightarrow 'b)$ **set**

proof –

obtain B **where** $B \subseteq K$ $U = (\bigcup B)$

using $\langle \text{open } U \rangle$ $\langle \text{topological_basis } K \rangle$ **by** $(\text{metis topological_basis_def})$

have $\text{countable } B$ **using** $\langle B \subseteq K \rangle$ $\langle \text{countable } K \rangle$ countable_subset **by** blast

moreover have $k \in \text{sets } (Pi_M UNIV (\lambda_. \text{borel}))$ **if** $k \in B$ **for** k

using $\langle B \subseteq K \rangle$ $*$ **that** **by** auto

```

ultimately show ?thesis unfolding ‹ $U = (\bigcup B)$ › by auto
qed
have sigma_sets UNIV (Collect open)  $\subseteq$  sets (Pi_M UNIV ( $\lambda i::'a$ . (borel::('b
measure))))
by (metis ** mem_Collect_eq open_UNIV sets.sigma_sets_subset' subsetI)
then show sets (borel::('a  $\Rightarrow$  'b) measure)  $\subseteq$  sets (Pi_M UNIV ( $\lambda\_$ . borel))
unfolding borel_def by auto
qed (simp add: sets_PiM_subset_borel)

```

```

lemma measurable_coordinatewise_then_product:
fixes f::'a  $\Rightarrow$  ('b::countable)  $\Rightarrow$  ('c::second_countable_topology)
assumes [measurable]:  $\bigwedge i$ . ( $\lambda x$ . f x i)  $\in$  borel_measurable M
shows f  $\in$  borel_measurable M
proof -
have f  $\in$  measurable M (Pi_M UNIV ( $\lambda\_$ . borel))
by (rule measurable_PiM_single', auto simp add: assms)
then show ?thesis using sets_PiM_equal_borel measurable_cong_sets by blast
qed

```

end

7.8 Caratheodory Extension Theorem

```

theory Caratheodory
imports Measure_Space
begin

```

Originally from the Hurd/Coble measure theory development, translated by Lawrence Paulson.

```

lemma suminf_ennreal_2dimen:
fixes f:: nat  $\times$  nat  $\Rightarrow$  ennreal
assumes  $\bigwedge m$ . g m = ( $\sum n$ . f (m,n))
shows ( $\sum i$ . f (prod_decode i)) = suminf g
proof -
have g_def: g = ( $\lambda m$ . ( $\sum n$ . f (m,n)))
using assms by (simp add: fun_eq_iff)
have reindex:  $\bigwedge B$ . ( $\sum x \in B$ . f (prod_decode x)) = sum f (prod_decode ' B)
by (simp add: sum.reindex[OF inj_prod_decode] comp_def)
have (SUP n.  $\sum i < n$ . f (prod_decode i)) = (SUP p  $\in$  UNIV  $\times$  UNIV.  $\sum i < fst$ 
p.  $\sum n < snd$  p. f (i, n))
proof (intro SUP_eq; clarsimp simp: sum.cartesian_product reindex)
fix n
let ?M =  $\lambda f$ . Suc (Max (f ' prod_decode ' {.. $n$ }))
{ fix a b x assume x < n and [symmetric]: (a, b) = prod_decode x
then have a < ?M fst b < ?M snd
by (auto intro!: Max_ge le_imp_less_Suc image_eqI) }
then have sum f (prod_decode ' {.. $n$ })  $\leq$  sum f ({.. $?M$  fst}  $\times$  {.. $?M$ 
snd})

```

```

    by (auto intro!: sum_mono2)
    then show  $\exists a b. \text{sum } f (\text{prod\_decode } \{..\<n\}) \leq \text{sum } f (\{..\<a\} \times \{..\<b\})$  by
  auto
  next
    fix a b
    let ?M = prod_decode ' {..Suc (Max (prod_encode ' ({..a}  $\times$  {..b}))) }
    { fix a' b' assume a' < a b' < b then have (a', b')  $\in$  ?M
      by (auto intro!: Max_ge le_imp_less_Suc image_eqI [where x=prod_encode
(a', b')]) }
    then have  $\text{sum } f (\{..\<a\} \times \{..\<b\}) \leq \text{sum } f ?M$ 
      by (auto intro!: sum_mono2)
    then show  $\exists n. \text{sum } f (\{..\<a\} \times \{..\<b\}) \leq \text{sum } f (\text{prod\_decode } \{..\<n\})$ 
      by auto
    qed
  also have ... = (SUP p.  $\sum i < p. \sum n. f (i, n)$ )
    unfolding suminf_sum[OF summableI, symmetric]
    by (simp add: suminf_eq_SUP SUP_PAIR swap[of _ {..fst _}])
  finally show ?thesis unfolding g_def
    by (simp add: suminf_eq_SUP)
  qed

```

7.8.1 Characterizations of Measures

definition *outer_measure_space* **where**

outer_measure_space $M f \iff \text{positive } M f \wedge \text{increasing } M f \wedge \text{countably_subadditive } M f$

Lambda Systems

definition *lambda_system* :: '*a* set \Rightarrow '*a* set set \Rightarrow ('*a* set \Rightarrow ennreal) \Rightarrow '*a* set set
where

lambda_system $\Omega M f = \{l \in M. \forall x \in M. f (l \cap x) + f ((\Omega - l) \cap x) = f x\}$

lemma (in algebra) *lambda_system_eq*:

lambda_system $\Omega M f = \{l \in M. \forall x \in M. f (x \cap l) + f (x - l) = f x\}$

proof –

have [simp]: $\bigwedge l x. l \in M \implies x \in M \implies (\Omega - l) \cap x = x - l$

by (metis Int_Diff Int_absorb1 Int_commute sets_into_space)

show ?thesis

by (auto simp add: lambda_system_def) (metis Int_commute)+

qed

lemma (in algebra) *lambda_system_empty*: $\text{positive } M f \implies \{\} \in \text{lambda_system } \Omega M f$

by (auto simp add: positive_def lambda_system_eq)

lemma *lambda_system_sets*: $x \in \text{lambda_system } \Omega M f \implies x \in M$

by (simp add: lambda_system_def)

lemma (in algebra) *lambda_system_Compl*:

```

fixes f:: 'a set  $\Rightarrow$  ennreal
assumes x: x  $\in$  lambda_system  $\Omega$  M f
shows  $\Omega - x \in$  lambda_system  $\Omega$  M f
proof -
  have x  $\subseteq$   $\Omega$ 
    by (metis sets_into_space lambda_system_sets)
  hence  $\Omega - (\Omega - x) = x$ 
    by (metis double_diff equalityE)
  with x show ?thesis
    by (force simp add: lambda_system_def ac_simps)
qed

```

lemma (in algebra) lambda_system_Int:

```

fixes f:: 'a set  $\Rightarrow$  ennreal
assumes xl: x  $\in$  lambda_system  $\Omega$  M f and yl: y  $\in$  lambda_system  $\Omega$  M f
shows x  $\cap$  y  $\in$  lambda_system  $\Omega$  M f
proof -
  from xl yl show ?thesis
proof (auto simp add: positive_def lambda_system_eq Int)
  fix u
  assume x: x  $\in$  M and y: y  $\in$  M and u: u  $\in$  M
    and fx:  $\forall z \in M. f(z \cap x) + f(z - x) = f z$ 
    and fy:  $\forall z \in M. f(z \cap y) + f(z - y) = f z$ 
  have u - x  $\cap$  y  $\in$  M
    by (metis Diff Diff_Int Un u x y)
  moreover
  have (u - (x  $\cap$  y))  $\cap$  y = u  $\cap$  y - x by blast
  moreover
  have u - x  $\cap$  y - y = u - y by blast
  ultimately
  have ey: f(u - x  $\cap$  y) = f(u  $\cap$  y - x) + f(u - y) using fy
    by force
  have f(u  $\cap$  (x  $\cap$  y)) + f(u - x  $\cap$  y)
    = (f(u  $\cap$  (x  $\cap$  y)) + f(u  $\cap$  y - x)) + f(u - y)
    by (simp add: ey ac_simps)
  also have ... = (f((u  $\cap$  y)  $\cap$  x) + f(u  $\cap$  y - x)) + f(u - y)
    by (simp add: Int_ac)
  also have ... = f(u  $\cap$  y) + f(u - y)
    using fx [THEN bspec, of u  $\cap$  y] Int y u
    by force
  also have ... = f u
    by (metis fy u)
  finally show f(u  $\cap$  (x  $\cap$  y)) + f(u - x  $\cap$  y) = f u .
qed
qed

```

lemma (in algebra) lambda_system_Un:

```

fixes f:: 'a set  $\Rightarrow$  ennreal
assumes xl: x  $\in$  lambda_system  $\Omega$  M f and yl: y  $\in$  lambda_system  $\Omega$  M f

```

shows $x \cup y \in \text{lambda_system } \Omega M f$
proof –
have $(\Omega - x) \cap (\Omega - y) \in M$
by (*metis Diff_Un Un compl_sets lambda_system_sets xl yl*)
moreover
have $x \cup y = \Omega - ((\Omega - x) \cap (\Omega - y))$
by *auto* (*metis subsetD lambda_system_sets sets_into_space xl yl*)+
ultimately show *?thesis*
by (*metis lambda_system_Compl lambda_system_Int xl yl*)
qed

lemma (*in algebra*) *lambda_system_algebra*:
positive M f \implies *algebra* Ω (*lambda_system* $\Omega M f$)
apply (*auto simp add: algebra_iff_Un*)
apply (*metis lambda_system_sets subsetD sets_into_space*)
apply (*metis lambda_system_empty*)
apply (*metis lambda_system_Compl*)
apply (*metis lambda_system_Un*)
done

lemma (*in algebra*) *lambda_system_strong_additive*:
assumes $z: z \in M$ **and** *disj*: $x \cap y = \{\}$
and *xl*: $x \in \text{lambda_system } \Omega M f$ **and** *yl*: $y \in \text{lambda_system } \Omega M f$
shows $f(z \cap (x \cup y)) = f(z \cap x) + f(z \cap y)$
proof –
have $z \cap x = (z \cap (x \cup y)) \cap x$ **using** *disj* **by** *blast*
moreover
have $z \cap y = (z \cap (x \cup y)) - x$ **using** *disj* **by** *blast*
moreover
have $(z \cap (x \cup y)) \in M$
by (*metis Int_Un lambda_system_sets xl yl z*)
ultimately show *?thesis* **using** *xl yl*
by (*simp add: lambda_system_eq*)
qed

lemma (*in algebra*) *lambda_system_additive*: *additive* (*lambda_system* $\Omega M f$) *f*
proof (*auto simp add: additive_def*)
fix x **and** y
assume *disj*: $x \cap y = \{\}$
and *xl*: $x \in \text{lambda_system } \Omega M f$ **and** *yl*: $y \in \text{lambda_system } \Omega M f$
hence $x \in M$ $y \in M$ **by** (*blast intro: lambda_system_sets*)+
thus $f(x \cup y) = f x + f y$
using *lambda_system_strong_additive* [*OF top disj xl yl*]
by (*simp add: Un*)
qed

lemma *lambda_system_increasing*: *increasing M f* \implies *increasing* (*lambda_system* $\Omega M f$) *f*
by (*simp add: increasing_def lambda_system_def*)

lemma *lambda_system_positive*: $positive\ M\ f \implies positive\ (\lambda_system\ \Omega\ M\ f)\ f$
by (*simp add: positive_def lambda_system_def*)

lemma (*in algebra*) *lambda_system_strong_sum*:
fixes $A::\ nat \Rightarrow 'a\ set$ **and** $f:: 'a\ set \Rightarrow ennreal$
assumes $f: positive\ M\ f$ **and** $a: a \in M$
and $A: range\ A \subseteq \lambda_system\ \Omega\ M\ f$
and $disj: disjoint_family\ A$
shows $(\sum\ i = 0..<n. f\ (a \cap A\ i)) = f\ (a \cap (\bigcup\ i \in \{0..<n\}. A\ i))$
proof (*induct n*)

case 0 **show** *?case* **using** f **by** (*simp add: positive_def*)
next
case (*Suc n*)
have $2: A\ n \cap \bigcup\ (A\ ' \{0..<n\}) = \{\}$ **using** *disj*
by (*force simp add: disjoint_family_on_def neq_iff*)
have $3: A\ n \in \lambda_system\ \Omega\ M\ f$ **using** A
by *blast*
interpret $l: algebra\ \Omega\ \lambda_system\ \Omega\ M\ f$
using f **by** (*rule lambda_system_algebra*)
have $4: \bigcup\ (A\ ' \{0..<n\}) \in \lambda_system\ \Omega\ M\ f$
using $A\ l.UNION_in_sets$ **by** *simp*
from *Suc.hyps* **show** *?case*
by (*simp add: atLeastLessThanSuc lambda_system_strong_additive [OF a 2 3 4]*)
qed

proposition (*in sigma_algebra*) *lambda_system_caratheodory*:
assumes $oms: outer_measure_space\ M\ f$
and $A: range\ A \subseteq \lambda_system\ \Omega\ M\ f$
and $disj: disjoint_family\ A$
shows $(\bigcup\ i. A\ i) \in \lambda_system\ \Omega\ M\ f \wedge (\sum\ i. f\ (A\ i)) = f\ (\bigcup\ i. A\ i)$

proof –
have $pos: positive\ M\ f$ **and** $inc: increasing\ M\ f$
and $csa: countably_subadditive\ M\ f$
by (*metis oms outer_measure_space_def*)
have $sa: subadditive\ M\ f$
by (*metis countably_subadditive_subadditive csa pos*)
have $A': \bigwedge S. A'S \subseteq (\lambda_system\ \Omega\ M\ f)$ **using** A
by *auto*
interpret $ls: algebra\ \Omega\ \lambda_system\ \Omega\ M\ f$
using pos **by** (*rule lambda_system_algebra*)
have $A'': range\ A \subseteq M$
by (*metis A image_subset_iff lambda_system_sets*)

have $U_in: (\bigcup\ i. A\ i) \in M$
by (*metis A'' countable_UN*)
have $U_eq: f\ (\bigcup\ i. A\ i) = (\sum\ i. f\ (A\ i))$

```

proof (rule antisym)
  show  $f (\bigcup i. A i) \leq (\sum i. f (A i))$ 
    using csa[unfolded countably_subadditive_def] A'' disj U_in by auto
  have dis:  $\bigwedge N. \text{disjoint\_family\_on } A \{..<N\}$  by (intro disjoint_family_on_mono[OF
  _ disj]) auto
  show  $(\sum i. f (A i)) \leq f (\bigcup i. A i)$ 
    using ls.additive_sum [OF lambda_system_positive[OF pos] lambda_system_additive
  _ A' dis] A''
    by (intro suminf_le_const[OF summableI]) (auto intro!: increasingD[OF inc]
  countable_UN)
  qed
  have  $f (a \cap (\bigcup i. A i)) + f (a - (\bigcup i. A i)) = f a$ 
    if a [iff]:  $a \in M$  for a
  proof (rule antisym)
    have range  $(\lambda i. a \cap A i) \subseteq M$  using A''
      by blast
    moreover
    have disjoint_family  $(\lambda i. a \cap A i)$  using disj
      by (auto simp add: disjoint_family_on_def)
    moreover
    have  $a \cap (\bigcup i. A i) \in M$ 
      by (metis Int U_in a)
    ultimately
    have  $f (a \cap (\bigcup i. A i)) \leq (\sum i. f (a \cap A i))$ 
      using csa[unfolded countably_subadditive_def, rule_format, of  $(\lambda i. a \cap A i)$ ]
      by (simp add: o_def)
    hence  $f (a \cap (\bigcup i. A i)) + f (a - (\bigcup i. A i)) \leq (\sum i. f (a \cap A i)) + f (a -$ 
   $(\bigcup i. A i))$ 
      by (rule add_right_mono)
    also have  $\dots \leq f a$ 
  proof (intro ennreal_suminf_bound_add)
    fix n
    have UNION_in:  $(\bigcup i \in \{0..<n\}. A i) \in M$ 
      by (metis A'' UNION_in_sets)
    have le_fa:  $f (\bigcup (A \ ' \{0..<n\}) \cap a) \leq f a$  using A''
      by (blast intro: increasingD [OF inc] A'' UNION_in_sets)
    have ls:  $(\bigcup i \in \{0..<n\}. A i) \in \text{lambda\_system } \Omega M f$ 
      using ls.UNION_in_sets by (simp add: A)
    hence eq_fa:  $f a = f (a \cap (\bigcup i \in \{0..<n\}. A i)) + f (a - (\bigcup i \in \{0..<n\}. A$ 
   $i))$ 
      by (simp add: lambda_system_eq UNION_in)
    have  $f (a - (\bigcup i. A i)) \leq f (a - (\bigcup i \in \{0..<n\}. A i))$ 
      by (blast intro: increasingD [OF inc] UNION_in U_in)
    thus  $(\sum i < n. f (a \cap A i)) + f (a - (\bigcup i. A i)) \leq f a$ 
      by (simp add: lambda_system_strong_sum_pos A disj eq_fa add_left_mono
  atLeast0LessThan[symmetric])
  qed
  finally show  $f (a \cap (\bigcup i. A i)) + f (a - (\bigcup i. A i)) \leq f a$ 
    by simp

```

```

next
  have  $f a \leq f (a \cap (\bigcup i. A i) \cup (a - (\bigcup i. A i)))$ 
    by (blast intro: increasingD [OF inc] U_in)
  also have  $\dots \leq f (a \cap (\bigcup i. A i)) + f (a - (\bigcup i. A i))$ 
    by (blast intro: subadditiveD [OF sa] U_in)
  finally show  $f a \leq f (a \cap (\bigcup i. A i)) + f (a - (\bigcup i. A i))$  .
qed
thus ?thesis
  by (simp add: lambda_system_eq sums_iff U_eq U_in)
qed

```

proposition (in *sigma_algebra*) *caratheodory_lemma*:

```

assumes oms: outer_measure_space M f
defines L  $\equiv$  lambda_system  $\Omega$  M f
shows measure_space  $\Omega$  L f
proof -
  have pos: positive M f
    by (metis oms outer_measure_space_def)
  have alg: algebra  $\Omega$  L
    using lambda_system_algebra [of f, OF pos]
    by (simp add: algebra_iff_Un L_def)
  then
  have sigma_algebra  $\Omega$  L
    using lambda_system_caratheodory [OF oms]
    by (simp add: sigma_algebra_disjoint_iff L_def)
  moreover
  have countably_additive L f positive L f
    using pos lambda_system_caratheodory [OF oms]
    by (auto simp add: lambda_system_sets L_def countably_additive_def positive_def)
  ultimately
  show ?thesis
    using pos by (simp add: measure_space_def)
qed

```

definition *outer_measure* :: 'a set set \Rightarrow ('a set \Rightarrow ennreal) \Rightarrow 'a set \Rightarrow ennreal
where

```

  outer_measure M f X =
    (INF A $\in$ {A. range A  $\subseteq$  M  $\wedge$  disjoint_family A  $\wedge$  X  $\subseteq$  ( $\bigcup$  i. A i)}.  $\sum$  i. f (A i))

```

lemma (in *ring_of_sets*) *outer_measure_agrees*:

```

assumes posf: positive M f and ca: countably_additive M f and s: s  $\in$  M
shows outer_measure M f s = f s
unfolding outer_measure_def
proof (safe intro!: antisym INF_greatest)
  fix A :: nat  $\Rightarrow$  'a set assume A: range A  $\subseteq$  M and dA: disjoint_family A and
  sA: s  $\subseteq$  ( $\bigcup$  x. A x)
  have inc: increasing M f

```

by (metis additive_increasing ca countably_additive_additive posf)
 have $f s = f (\bigcup i. A i \cap s)$
 using sA by (auto simp: Int_absorb1)
 also have $\dots = (\sum i. f (A i \cap s))$
 using sA dA A s
 by (intro ca[unfolded countably_additive_def, rule_format, symmetric])
 (auto simp: Int_absorb1 disjoint_family_on_def)
 also have $\dots \leq (\sum i. f (A i))$
 using A s by (auto intro!: suminf_le increasingD[OF inc])
 finally show $f s \leq (\sum i. f (A i))$.
 next
 have $(\sum i. f (if i = 0 then s else \{\})) \leq f s$
 using positiveD1[OF posf] by (subst suminf_finite[of \{0\}]) auto
 with s show $(\text{INF } A \in \{A. \text{range } A \subseteq M \wedge \text{disjoint_family } A \wedge s \subseteq \bigcup (A \text{ ` } UNIV)\}. \sum i. f (A i)) \leq f s$
 by (intro INF_lower2[of $\lambda i. \text{if } i = 0 \text{ then } s \text{ else } \{\}$])
 (auto simp: disjoint_family_on_def)
 qed

lemma outer_measure_empty:
 positive M f $\implies \{\} \in M \implies \text{outer_measure } M f \{\} = 0$
unfolding outer_measure_def
 by (intro antisym INF_lower2[of $\lambda _. \{\}$]) (auto simp: disjoint_family_on_def positive_def)

lemma (in ring_of_sets) positive_outer_measure:
 assumes positive M f shows positive (Pow Ω) (outer_measure M f)
unfolding positive_def by (auto simp: assms outer_measure_empty)

lemma (in ring_of_sets) increasing_outer_measure: increasing (Pow Ω) (outer_measure M f)
 by (force simp: increasing_def outer_measure_def intro!: INF_greatest intro: INF_lower)

lemma (in ring_of_sets) outer_measure_le:
 assumes pos: positive M f and inc: increasing M f and A: range A \subseteq M and X: X \subseteq ($\bigcup i. A i$)
 shows outer_measure M f X \leq ($\sum i. f (A i)$)
unfolding outer_measure_def
proof (safe intro!: INF_lower2[of disjointed A] del: subsetI)
 show dA: range (disjointed A) \subseteq M
 by (auto intro!: A range_disjointed_sets)
 have $\forall n. f (\text{disjointed } A n) \leq f (A n)$
 by (metis increasingD [OF inc] UNIV_I dA image_subset_iff disjointed_subset A)
 then show $(\sum i. f (\text{disjointed } A i)) \leq (\sum i. f (A i))$
 by (blast intro!: suminf_le)
 qed (auto simp: X UN_disjointed_eq disjoint_family_disjointed)

lemma (in ring_of_sets) outer_measure_close:

outer_measure M f X < e $\implies \exists A. \text{range } A \subseteq M \wedge \text{disjoint_family } A \wedge X \subseteq (\bigcup i. A i) \wedge (\sum i. f (A i)) < e$

unfolding outer_measure_def INF_less_iff **by** auto

lemma (in ring_of_sets) countably_subadditive_outer_measure:

assumes posf: positive M f **and** inc: increasing M f

shows countably_subadditive (Pow Ω) (outer_measure M f)

proof (simp add: countably_subadditive_def, safe)

fix A :: nat \implies _ **assume** A: range A \subseteq Pow (Ω) **and** sb: $(\bigcup i. A i) \subseteq \Omega$

let ?O = outer_measure M f

show ?O $(\bigcup i. A i) \leq (\sum n. ?O (A n))$

proof (rule ennreal_le_epsilon)

fix b **and** e :: real **assume** 0 < e $(\sum n. \text{outer_measure } M f (A n)) < \text{top}$

then have *: $\bigwedge n. \text{outer_measure } M f (A n) < \text{outer_measure } M f (A n) + e * (1/2)^{\wedge} \text{Suc } n$

by (auto simp add: less_top dest!: ennreal_suminf_lessD)

obtain B

where B: $\bigwedge n. \text{range } (B n) \subseteq M$

and sbB: $\bigwedge n. A n \subseteq (\bigcup i. B n i)$

and Ble: $\bigwedge n. (\sum i. f (B n i)) \leq ?O (A n) + e * (1/2)^{\wedge} \text{Suc } n$

by (metis less_imp_le outer_measure_close[OF *])

define C **where** C = case_prod B o prod_decode

from B **have** B_in_M: $\bigwedge i j. B i j \in M$

by (rule range_subsetD)

then have C: range C $\subseteq M$

by (auto simp add: C_def split_def)

have A_C: $(\bigcup i. A i) \subseteq (\bigcup i. C i)$

using sbB **by** (auto simp add: C_def subset_eq) (metis prod.case prod_encode_inverse)

have ?O $(\bigcup i. A i) \leq ?O (\bigcup i. C i)$

using A_C A_C **by** (intro increasing_outer_measure[THEN increasingD])

(auto dest!: sets_into_space)

also have ... $\leq (\sum i. f (C i))$

using C **by** (intro outer_measure_le[OF posf inc]) auto

also have ... = $(\sum n. \sum i. f (B n i))$

using B_in_M **unfolding** C_def comp_def **by** (intro suminf_ennreal_2dimen)

auto

also have ... $\leq (\sum n. ?O (A n) + e * (1/2)^{\wedge} \text{Suc } n)$

using B_in_M **by** (intro suminf_le suminf_nonneg allI Ble) auto

also have ... = $(\sum n. ?O (A n)) + (\sum n. \text{ennreal } e * \text{ennreal } ((1/2)^{\wedge} \text{Suc } n))$

using <0 < e> **by** (subst suminf_add[symmetric])

(auto simp del: ennreal_suminf_cmult simp add: en-

nreal_mult[symmetric])

also have ... = $(\sum n. ?O (A n)) + e$

unfolding ennreal_suminf_cmult

by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto

finally show ?O $(\bigcup i. A i) \leq (\sum n. ?O (A n)) + e$.

qed
qed

lemma (in ring_of_sets) outer_measure_space_outer_measure:
 positive $M f \implies$ increasing $M f \implies$ outer_measure_space (Pow Ω) (outer_measure $M f$)
 by (simp add: outer_measure_space_def
 positive_outer_measure increasing_outer_measure countably_subadditive_outer_measure)

lemma (in ring_of_sets) algebra_subset_lambda_system:
 assumes posf: positive $M f$ and inc: increasing $M f$
 and add: additive $M f$
 shows $M \subseteq$ lambda_system Ω (Pow Ω) (outer_measure $M f$)
proof (auto dest: sets_into_space
 simp add: algebra.lambda_system_eq [OF algebra_Pow])
 fix $x s$ assume $x: x \in M$ and $s: s \subseteq \Omega$
 have [simp]: $\bigwedge x. x \in M \implies s \cap (\Omega - x) = s - x$ using s
 by blast
 have outer_measure $M f (s \cap x) +$ outer_measure $M f (s - x) \leq$ outer_measure
 $M f s$
 unfolding outer_measure_def[of $M f s$]
proof (safe intro!: INF_greatest)
 fix $A :: \text{nat} \Rightarrow 'a \text{ set}$ assume $A: \text{disjoint_family } A$ range $A \subseteq M$ $s \subseteq (\bigcup i. A$
 $i)$
 have outer_measure $M f (s \cap x) \leq (\sum i. f (A i \cap x))$
 unfolding outer_measure_def
proof (safe intro!: INF_lower2[of $\lambda i. A i \cap x$])
 from $A(1)$ show disjoint_family $(\lambda i. A i \cap x)$
 by (rule disjoint_family_on_bisimulation) auto
 qed (insert $x A$, auto)
 moreover
 have outer_measure $M f (s - x) \leq (\sum i. f (A i - x))$
 unfolding outer_measure_def
proof (safe intro!: INF_lower2[of $\lambda i. A i - x$])
 from $A(1)$ show disjoint_family $(\lambda i. A i - x)$
 by (rule disjoint_family_on_bisimulation) auto
 qed (insert $x A$, auto)
 ultimately have outer_measure $M f (s \cap x) +$ outer_measure $M f (s - x) \leq$
 $(\sum i. f (A i \cap x)) + (\sum i. f (A i - x))$ by (rule add_mono)
 also have $\dots = (\sum i. f (A i \cap x) + f (A i - x))$
 using $A(2)$ x posf by (subst suminf_add) (auto simp: positive_def)
 also have $\dots = (\sum i. f (A i))$
 using $A x$
 by (subst add[THEN additiveD, symmetric])
 (auto intro!: arg_cong[where $f=\text{suminf}$] arg_cong[where $f=f$])
 finally show outer_measure $M f (s \cap x) +$ outer_measure $M f (s - x) \leq$
 $(\sum i. f (A i))$.
 qed
 moreover

```

have outer_measure M f s ≤ outer_measure M f (s ∩ x) + outer_measure M f
(s - x)
proof -
  have outer_measure M f s = outer_measure M f ((s ∩ x) ∪ (s - x))
    by (metis Un_Diff_Int Un_commute)
  also have ... ≤ outer_measure M f (s ∩ x) + outer_measure M f (s - x)
    apply (rule subadditiveD)
  apply (rule ring_of_sets.countably_subadditive_subadditive [OF ring_of_sets_Pow])
  apply (simp add: positive_def outer_measure_empty[OF posf])
  apply (rule countably_subadditive_outer_measure)
  using s by (auto intro!: posf inc)
  finally show ?thesis .
qed
ultimately
show outer_measure M f (s ∩ x) + outer_measure M f (s - x) = outer_measure
M f s
  by (rule order_antisym)
qed

```

```

lemma measure_down: measure_space Ω N μ ⇒ sigma_algebra Ω M ⇒ M ⊆
N ⇒ measure_space Ω M μ
  by (auto simp add: measure_space_def positive_def countably_additive_def sub-
set_eq)

```

7.8.2 Caratheodory's theorem

```

theorem (in ring_of_sets) caratheodory':
  assumes posf: positive M f and ca: countably_additive M f
  shows ∃μ :: 'a set ⇒ ennreal. (∀s ∈ M. μ s = f s) ∧ measure_space Ω
(sigma_sets Ω M) μ
proof -
  have inc: increasing M f
    by (metis additive_increasing ca countably_additive_additive posf)
  let ?O = outer_measure M f
  define ls where ls = lambda_system Ω (Pow Ω) ?O
  have mls: measure_space Ω ls ?O
    using sigma_algebra.caratheodory_lemma
      [OF sigma_algebra_Pow outer_measure_space_outer_measure [OF posf
inc]]
    by (simp add: ls_def)
  hence sls: sigma_algebra Ω ls
    by (simp add: measure_space_def)
  have M ⊆ ls
    by (simp add: ls_def)
    (metis ca posf inc countably_additive_additive algebra_subset_lambda_system)
  hence sgs_sb: sigma_sets (Ω) (M) ⊆ ls
    using sigma_algebra.sigma_sets_subset [OF sls, of M]
    by simp
  have measure_space Ω (sigma_sets Ω M) ?O

```

2348

by (rule *measure_down* [*OF mls*], rule *sigma_algebra_sigma_sets*)
 (simp_all add: *sgs_sb space_closed*)
 thus ?thesis using *outer_measure_agrees* [*OF posf ca*]
 by (intro *exI[of_ ?O]*) auto
 qed

lemma (in *ring_of_sets*) *caratheodory_empty_continuous*:
 assumes *f*: *positive M f additive M f* and *fin*: $\bigwedge A. A \in M \implies f A \neq \infty$
 assumes *cont*: $\bigwedge A. \text{range } A \subseteq M \implies \text{decseq } A \implies (\bigcap i. A i) = \{\} \implies (\lambda i. f (A i)) \longrightarrow 0$
 shows $\exists \mu :: 'a \text{ set} \Rightarrow \text{ennreal}. (\forall s \in M. \mu s = f s) \wedge \text{measure_space } \Omega$
 (*sigma_sets* ΩM) μ
proof (intro *caratheodory' empty_continuous_imp_countably_additive f*)
 show $\forall A \in M. f A \neq \infty$ using *fin* by auto
 qed (rule *cont*)

7.8.3 Volumes

definition *volume* :: $'a \text{ set set} \Rightarrow ('a \text{ set} \Rightarrow \text{ennreal}) \Rightarrow \text{bool}$ **where**
volume M f \longleftrightarrow
 $(f \{\} = 0) \wedge (\forall a \in M. 0 \leq f a) \wedge$
 $(\forall C \subseteq M. \text{disjoint } C \longrightarrow \text{finite } C \longrightarrow \bigcup C \in M \longrightarrow f (\bigcup C) = (\sum c \in C. f c))$

lemma *volumeI*:
 assumes $f \{\} = 0$
 assumes $\bigwedge a. a \in M \implies 0 \leq f a$
 assumes $\bigwedge C. C \subseteq M \implies \text{disjoint } C \implies \text{finite } C \implies \bigcup C \in M \implies f (\bigcup C) = (\sum c \in C. f c)$
 shows *volume M f*
 using *assms* by (auto simp: *volume_def*)

lemma *volume_positive*:
volume M f $\implies a \in M \implies 0 \leq f a$
 by (auto simp: *volume_def*)

lemma *volume_empty*:
volume M f $\implies f \{\} = 0$
 by (auto simp: *volume_def*)

proposition *volume_finite_additive*:
 assumes *volume M f*
 assumes *A*: $\bigwedge i. i \in I \implies A i \in M \text{ disjoint_family_on } A I \text{ finite } I \bigcup (A ' I) \in M$
 shows $f (\bigcup (A ' I)) = (\sum i \in I. f (A i))$
proof –
 have $A ' I \subseteq M \text{ disjoint } (A ' I) \text{ finite } (A ' I) \bigcup (A ' I) \in M$
 using *A* by (auto simp: *disjoint_family_on_disjoint_image*)
 with $\langle \text{volume } M f \rangle$ have $f (\bigcup (A ' I)) = (\sum a \in A ' I. f a)$
 unfolding *volume_def* by blast


```

also have ... = ( $\sum_{i \in I}. f(A\ i)$ )
proof (subst sum.reindex_nontrivial)
  fix i j assume i ∈ I j ∈ I i ≠ j A i = A j
  with ⟨disjoint_family_on A I⟩ have A i = {}
  by (auto simp: disjoint_family_on_def)
  then show f(A i) = 0
  using volume_empty[OF ⟨volume M f⟩] by simp
qed (auto intro: ⟨finite I⟩)
finally show f( $\bigcup(A\ 'I)$ ) = ( $\sum_{i \in I}. f(A\ i)$ )
  by simp
qed

lemma (in ring_of_sets) volume_additiveI:
  assumes pos:  $\bigwedge a. a \in M \implies 0 \leq \mu\ a$ 
  assumes [simp]:  $\mu\ \{\} = 0$ 
  assumes add:  $\bigwedge a\ b. a \in M \implies b \in M \implies a \cap b = \{\} \implies \mu(a \cup b) = \mu\ a + \mu\ b$ 
  shows volume M  $\mu$ 
proof (unfold volume_def, safe)
  fix C assume finite C C ⊆ M disjoint C
  then show  $\mu(\bigcup C) = \text{sum } \mu\ C$ 
  proof (induct C)
    case (insert c C)
    from insert(1,2,4,5) have  $\mu(\bigcup(\text{insert } c\ C)) = \mu\ c + \mu(\bigcup C)$ 
    by (auto intro!: add simp: disjoint_def)
    with insert show ?case
    by (simp add: disjoint_def)
  qed simp
qed fact+

proposition (in semiring_of_sets) extend_volume:
  assumes volume M  $\mu$ 
  shows  $\exists \mu'. \text{volume\_generated\_ring } \mu' \wedge (\forall a \in M. \mu'\ a = \mu\ a)$ 
proof -
  let ?R = generated_ring
  have  $\forall a \in ?R. \exists m. \exists C \subseteq M. a = \bigcup C \wedge \text{finite } C \wedge \text{disjoint } C \wedge m = (\sum_{c \in C}. \mu\ c)$ 
  by (auto simp: generated_ring_def)
  from bchoice[OF this] obtain  $\mu'$ 
  where  $\mu'_\text{spec}: \forall x \in \text{generated\_ring}. \exists C \subseteq M. x = \bigcup C \wedge \text{finite } C \wedge \text{disjoint } C \wedge \mu'\ x = \text{sum } \mu\ C$ 
  by blast
  { fix C assume C: C ⊆ M finite C disjoint C
    fix D assume D: D ⊆ M finite D disjoint D
    assume  $\bigcup C = \bigcup D$ 
    have  $(\sum_{d \in D}. \mu\ d) = (\sum_{d \in D}. \sum_{c \in C}. \mu(c \cap d))$ 
    proof (intro sum.cong refl)
      fix d assume d ∈ D
      have Un_eq_d:  $(\bigcup_{c \in C}. c \cap d) = d$ 

```

```

    using ⟨d ∈ D⟩ ⟨ $\bigcup C = \bigcup D$ ⟩ by auto
  moreover have  $\mu (\bigcup c \in C. c \cap d) = (\sum c \in C. \mu (c \cap d))$ 
  proof (rule volume_finite_additive)
    { fix c assume c ∈ C then show c ∩ d ∈ M
      using C D ⟨d ∈ D⟩ by auto }
    show  $(\bigcup a \in C. a \cap d) \in M$ 
      unfolding Un_eq_d using ⟨d ∈ D⟩ D by auto
    show disjoint_family_on (λa. a ∩ d) C
      using ⟨disjoint C⟩ by (auto simp: disjoint_family_on_def disjoint_def)
  qed fact+
  ultimately show  $\mu d = (\sum c \in C. \mu (c \cap d))$  by simp
  qed }
note split_sum = this

{ fix C assume C: C ⊆ M finite C disjoint C
  fix D assume D: D ⊆ M finite D disjoint D
  assume  $\bigcup C = \bigcup D$ 
  with split_sum[OF C D] split_sum[OF D C]
  have  $(\sum d \in D. \mu d) = (\sum c \in C. \mu c)$ 
    by (simp, subst sum.swap, simp add: ac_simps) }
note sum_eq = this

{ fix C assume C: C ⊆ M finite C disjoint C
  then have  $\bigcup C \in ?R$  by (auto simp: generated_ring_def)
  with μ'_spec[THEN bspec, of  $\bigcup C$ ]
  obtain D where
    D: D ⊆ M finite D disjoint D  $\bigcup C = \bigcup D$  and  $\mu' (\bigcup C) = (\sum d \in D. \mu d)$ 
    by auto
  with sum_eq[OF C D] have  $\mu' (\bigcup C) = (\sum c \in C. \mu c)$  by simp }
note μ' = this

show ?thesis
proof (intro exI conjI ring_of_sets.volume_additiveI[OF generating_ring] ballI)
  fix a assume a ∈ M with μ'[of {a}] show μ' a = μ a
    by (simp add: disjoint_def)
next
  fix a assume a ∈ ?R
  then obtain Ca where Ca: finite Ca disjoint Ca Ca ⊆ M a =  $\bigcup Ca$  ..
  with μ'[of Ca] ⟨volume M μ⟩[THEN volume_positive]
  show 0 ≤ μ' a
    by (auto intro!: sum_nonneg)
next
  show μ' {} = 0 using μ'[of {}] by auto
next
  fix a b assume a ∈ ?R b ∈ ?R
  then obtain Ca Cb
    where Ca: finite Ca disjoint Ca Ca ⊆ M a =  $\bigcup Ca$ 
      and Cb: finite Cb disjoint Cb Cb ⊆ M b =  $\bigcup Cb$ 
    by (meson generated_ringE)

```

```

assume  $a \cap b = \{\}$ 
with  $Ca\ Cb$  have  $Ca \cap Cb \subseteq \{\{\}\}$  by auto
then have  $C\_Int\_cases: Ca \cap Cb = \{\{\}\} \vee Ca \cap Cb = \{\}$  by auto

from  $\langle a \cap b = \{\} \rangle$  have  $\mu' (\bigcup (Ca \cup Cb)) = (\sum_{c \in Ca \cup Cb} \mu\ c)$ 
  using  $Ca\ Cb$  by (intro  $\mu'$ ) (auto intro!: disjoint_union)
also have  $\dots = (\sum_{c \in Ca \cup Cb} \mu\ c) + (\sum_{c \in Ca \cap Cb} \mu\ c)$ 
  using  $C\_Int\_cases$  volume_empty[OF  $\langle volume\ M\ \mu \rangle$ ] by (elim disjE)
simp_all
also have  $\dots = (\sum_{c \in Ca} \mu\ c) + (\sum_{c \in Cb} \mu\ c)$ 
  using  $Ca\ Cb$  by (simp add: sum.union_inter)
also have  $\dots = \mu'\ a + \mu'\ b$ 
  using  $Ca\ Cb$  by (simp add:  $\mu'$ )
finally show  $\mu' (a \cup b) = \mu'\ a + \mu'\ b$ 
  using  $Ca\ Cb$  by simp
qed
qed

```

Caratheodory on semirings

```

theorem (in semiring_of_sets) caratheodory:
  assumes pos: positive  $M\ \mu$  and ca: countably_additive  $M\ \mu$ 
  shows  $\exists \mu' :: 'a\ set \Rightarrow ennreal. (\forall s \in M. \mu'\ s = \mu\ s) \wedge measure\_space\ \Omega$ 
  (sigma_sets  $\Omega\ M)$   $\mu'$ 
proof -
  have volume  $M\ \mu$ 
  proof (rule volumeI)
    { fix  $a$  assume  $a \in M$  then show  $0 \leq \mu\ a$ 
      using pos unfolding positive_def by auto }
  note  $p = this$ 

  fix  $C$  assume sets_C:  $C \subseteq M \bigcup C \in M$  and disjoint  $C$  finite  $C$ 
  have  $\exists F'. bij\_betw\ F'\ \{..<card\ C\}\ C$ 
    by (rule finite_same_card_bij[OF  $\langle finite\ C \rangle$ ]) auto
  then obtain  $F'$  where bij_betw  $F'\ \{..<card\ C\}\ C$  ..
  then have  $F': C = F'\ \{..<card\ C\}$  inj_on  $F'\ \{..<card\ C\}$ 
    by (auto simp: bij_betw_def)
  { fix  $i\ j$  assume  $*$ :  $i < card\ C\ j < card\ C\ i \neq j$ 
    with  $F'$  have  $F'\ i \in C\ F'\ j \in C\ F'\ i \neq F'\ j$ 
      unfolding inj_on_def by auto
    with  $\langle disjoint\ C \rangle$  [THEN disjointD]
    have  $F'\ i \cap F'\ j = \{\}$ 
      by auto }
  note  $F'\_disj = this$ 
  define  $\bar{F}$  where  $\bar{F}\ i = (if\ i < card\ C\ then\ F'\ i\ else\ \{\})$  for  $i$ 
  then have disjoint_family  $\bar{F}$ 
    using  $F'\_disj$  by (auto simp: disjoint_family_on_def)
  moreover from  $F'$  have  $(\bigcup i. \bar{F}\ i) = \bigcup C$ 
    by (auto simp add:  $F\_def$  split: if_split_asm cong del: SUP_cong)

```

```

moreover have sets_F:  $\bigwedge i. F\ i \in M$ 
  using F' sets_C by (auto simp: F_def)
moreover note sets_C
ultimately have  $\mu (\bigcup C) = (\sum i. \mu (F\ i))$ 
  using ca[unfolded countably_additive_def, THEN spec, of F] by auto
also have ... =  $(\sum i < \text{card } C. \mu (F'\ i))$ 
proof -
  have ( $\lambda i. \text{if } i \in \{.. < \text{card } C\} \text{ then } \mu (F'\ i) \text{ else } 0$ ) sums  $(\sum i < \text{card } C. \mu (F'$ 
i))
    by (rule sums_If_finite_set) auto
  also have ( $\lambda i. \text{if } i \in \{.. < \text{card } C\} \text{ then } \mu (F'\ i) \text{ else } 0$ ) =  $(\lambda i. \mu (F\ i))$ 
    using pos by (auto simp: positive_def F_def)
  finally show  $(\sum i. \mu (F\ i)) = (\sum i < \text{card } C. \mu (F'\ i))$ 
    by (simp add: sums_iff)
qed
also have ... =  $(\sum c \in C. \mu\ c)$ 
  using F'(2) by (subst (2) F') (simp add: sum.reindex)
finally show  $\mu (\bigcup C) = (\sum c \in C. \mu\ c)$  .
next
show  $\mu \{\} = 0$ 
  using <positive M  $\mu$ > by (rule positiveD1)
qed
from extend_volume[OF this] obtain  $\mu\_r$  where
  V: volume_generated_ring  $\mu\_r \wedge a. a \in M \implies \mu\ a = \mu\_r\ a$ 
  by auto

interpret G: ring_of_sets  $\Omega$  generated_ring
  by (rule generating_ring)

have pos: positive_generated_ring  $\mu\_r$ 
  using V unfolding positive_def by (auto simp: positive_def intro!: volume_positive volume_empty)

have countably_additive_generated_ring  $\mu\_r$ 
proof (rule countably_additiveI)
  fix A' :: nat  $\Rightarrow$  'a set
  assume A': range A'  $\subseteq$  generated_ring disjoint_family A'
  and Un_A:  $(\bigcup i. A'\ i) \in$  generated_ring

  obtain C' where C': finite C' disjoint C' C'  $\subseteq M \cup (\text{range } A') = \bigcup C'$ 
  using generated_ringE[OF Un_A] by auto

  { fix c assume c  $\in C'$ 
    moreover define A where [abs_def]: A i = A' i  $\cap$  c for i
    ultimately have A: range A  $\subseteq$  generated_ring disjoint_family A
      and Un_A:  $(\bigcup i. A\ i) \in$  generated_ring
      using A' C'
      by (auto intro!: G.Int G.finite_Union intro: generated_ringI_Basic simp: disjoint_family_on_def)
  }

```

```

from  $A$   $C' \langle c \in C' \rangle$  have  $UN\_eq: (\bigcup i. A\ i) = c$ 
  by (auto simp: A_def)

have  $\forall i::nat. \exists f::nat \Rightarrow 'a\ set. \mu\_r (A\ i) = (\sum j. \mu\_r (f\ j)) \wedge disjoint\_family$ 
 $f \wedge \bigcup (range\ f) = A\ i \wedge (\forall j. f\ j \in M)$ 
  (is  $\forall i. ?P\ i$ )
proof
  fix  $i$ 
  from  $A$  have  $Ai: A\ i \in generated\_ring$  by auto
  from generated\_ringE[OF this] obtain  $C$ 
    where  $C: finite\ C\ disjoint\ C\ C \subseteq M\ A\ i = \bigcup C$  by auto
  have  $\exists F'. bij\_betw\ F' \{..<card\ C\}\ C$ 
    by (rule finite\_same\_card\_bij[OF  $\langle finite\ C \rangle$ ]) auto
  then obtain  $F$  where  $F: bij\_betw\ F \{..<card\ C\}\ C$  ..
  define  $f$  where [abs\_def]:  $f\ i = (if\ i < card\ C\ then\ F\ i\ else\ \{\})$  for  $i$ 
  then have  $f: bij\_betw\ f \{..<card\ C\}\ C$ 
    by (intro bij\_betw\_cong[THEN iffD1, OF  $F$ ]) auto
  with  $C$  have  $\forall j. f\ j \in M$ 
    by (auto simp: Pi\_iff\ f\_def\ dest!: bij\_betw\_imp\_funcset)
  moreover
  from  $f\ C$  have  $d\_f: disjoint\_family\_on\ f \{..<card\ C\}$ 
    by (intro disjoint\_image\_disjoint\_family\_on) (auto simp: bij\_betw\_def)
  then have disjoint\_family  $f$ 
    by (auto simp: disjoint\_family\_on\_def\ f\_def)
  moreover
  have  $Ai\_eq: A\ i = (\bigcup x<card\ C. f\ x)$ 
    using  $f\ C\ Ai$  unfolding bij\_betw\_def by auto
  then have  $\bigcup (range\ f) = A\ i$ 
    using  $f$  by (auto simp add: f\_def)
  moreover
  { have  $(\sum j. \mu\_r (f\ j)) = (\sum j. if\ j \in \{..<card\ C\}\ then\ \mu\_r (f\ j)\ else\ 0)$ 
    using volume\_empty[OF  $V(1)$ ] by (auto intro!: arg\_cong[where
 $f=suminf$ ] simp: f\_def)
    also have  $\dots = (\sum j<card\ C. \mu\_r (f\ j))$ 
      by (rule sums\_If\_finite\_set[THEN sums\_unique, symmetric]) simp
    also have  $\dots = \mu\_r (A\ i)$ 
      using  $C\ f$ [THEN bij\_betw\_imp\_funcset] unfolding  $Ai\_eq$ 
      by (intro volume\_finite\_additive[OF  $V(1)\ d\_f, symmetric$ ])
      (auto simp: Pi\_iff\ Ai\_eq\ intro: generated\_ringI\_Basic)
    finally have  $\mu\_r (A\ i) = (\sum j. \mu\_r (f\ j))$  .. }
  ultimately show  $?P\ i$ 
    by blast
qed
from choice[OF this] obtain  $f$ 
  where  $f: \forall x. \mu\_r (A\ x) = (\sum j. \mu\_r (f\ x\ j)) \wedge disjoint\_family (f\ x) \wedge \bigcup$ 
 $(range (f\ x)) = A\ x \wedge (\forall j. f\ x\ j \in M)$ 
  ..
  then have  $UN\_f\_eq: (\bigcup i. case\_prod\ f (prod\_decode\ i)) = (\bigcup i. A\ i)$ 
    unfolding  $UN\_extend\_simps\ surj\_prod\_decode$  by (auto simp: set\_eq\_iff)

```

```

have d: disjoint_family (λi. case_prod f (prod_decode i))
  unfolding disjoint_family_on_def
proof (intro ballI impI)
  fix m n :: nat assume m ≠ n
  then have neq: prod_decode m ≠ prod_decode n
    using inj_prod_decode[of UNIV] by (auto simp: inj_on_def)
  show case_prod f (prod_decode m) ∩ case_prod f (prod_decode n) = {}
  proof cases
    assume fst (prod_decode m) = fst (prod_decode n)
    then show ?thesis
      using neq f by (fastforce simp: disjoint_family_on_def)
  next
    assume neq: fst (prod_decode m) ≠ fst (prod_decode n)
    have case_prod f (prod_decode m) ⊆ A (fst (prod_decode m))
      case_prod f (prod_decode n) ⊆ A (fst (prod_decode n))
    using f[THEN spec, of fst (prod_decode m)]
    using f[THEN spec, of fst (prod_decode n)]
    by (auto simp: set_eq_iff)
    with f A neq show ?thesis
      by (fastforce simp: disjoint_family_on_def subset_eq set_eq_iff)
  qed
qed
from f have (∑ n. μ_r (A n)) = (∑ n. μ_r (case_prod f (prod_decode n)))
  by (intro suminf_ennreal_2dimen[symmetric] generated_ringI_Basic)
  (auto split: prod.split)
also have ... = (∑ n. μ (case_prod f (prod_decode n)))
  using f V(2) by (auto intro!: arg_cong[where f=suminf] split: prod.split)
also have ... = μ (⋃ i. case_prod f (prod_decode i))
  using f ⟨c ∈ C'⟩ C'
  by (intro ca[unfolded countably_additive_def, rule_format])
  (auto split: prod.split simp: UN_f_eq d UN_eq)
finally have (∑ n. μ_r (A' n ∩ c)) = μ c
  using UN_f_eq UN_eq by (simp add: A_def) }
note eq = this

have (∑ n. μ_r (A' n)) = (∑ n. ∑ c∈C'. μ_r (A' n ∩ c))
  using C' A'
  by (subst volume_finite_additive[symmetric, OF V(1)])
  (auto simp: disjoint_def disjoint_family_on_def
    intro!: G.Int G.finite_Union arg_cong[where f=λX. suminf (λi. μ_r
(X i))] ext
    intro: generated_ringI_Basic)
also have ... = (∑ c∈C'. ∑ n. μ_r (A' n ∩ c))
  using C' A'
  by (intro suminf_sum G.Int G.finite_Union) (auto intro: generated_ringI_Basic)
also have ... = (∑ c∈C'. μ_r c)
  using eq V C' by (auto intro!: sum.cong)
also have ... = μ_r (⋃ C')

```

```

using C' Un_A
by (subst volume_finite_additive[symmetric, OF V(1)])
    (auto simp: disjoint_family_on_def disjoint_def
      intro: generated_ringI_Basic)
finally show ( $\sum n. \mu_r (A' n) = \mu_r (\bigcup i. A' i)$ )
using C' by simp
qed
obtain  $\mu'$  where ( $\forall s \in \text{generated\_ring}. \mu' s = \mu_r s$ )  $\wedge$  measure_space  $\Omega$ 
(sigma_sets  $\Omega$  generated\_ring)  $\mu'$ 
using G.caratheodory'[OF pos <countably_additive generated_ring  $\mu_r$ >] by
auto
with V show ?thesis
unfolding sigma_sets_generated_ring_eq
by (intro exI[of _  $\mu'$ ]) (auto intro: generated_ringI_Basic)
qed

```

lemma *extend_measure_caratheodory*:

```

fixes G :: 'i  $\Rightarrow$  'a set
assumes M: M = extend_measure  $\Omega$  I G  $\mu$ 
assumes i  $\in$  I
assumes semiring_of_sets  $\Omega$  (G ' I)
assumes empty:  $\bigwedge i. i \in I \Rightarrow G i = \{\} \Rightarrow \mu i = 0$ 
assumes inj:  $\bigwedge i j. i \in I \Rightarrow j \in I \Rightarrow G i = G j \Rightarrow \mu i = \mu j$ 
assumes nonneg:  $\bigwedge i. i \in I \Rightarrow 0 \leq \mu i$ 
assumes add:  $\bigwedge A::\text{nat} \Rightarrow 'i. \bigwedge j. A \in \text{UNIV} \rightarrow I \Rightarrow j \in I \Rightarrow \text{disjoint\_family}$ 
(G o A)  $\Rightarrow$ 
( $\bigcup i. G (A i) = G j \Rightarrow (\sum n. \mu (A n)) = \mu j$ )
shows emeasure M (G i) =  $\mu i$ 

```

proof –

```

interpret semiring_of_sets  $\Omega$  G ' I
by fact
have  $\forall g \in G'I. \exists i \in I. g = G i$ 
by auto
then obtain sel where sel:  $\bigwedge g. g \in G ' I \Rightarrow sel g \in I \wedge g. g \in G ' I \Rightarrow G$ 
(sel g) = g
by metis

```

```

have  $\exists \mu'. (\forall s \in G ' I. \mu' s = \mu (sel s)) \wedge \text{measure\_space } \Omega$  (sigma_sets  $\Omega$  (G '
I))  $\mu'$ 

```

proof (*rule caratheodory*)

```

show positive (G ' I) ( $\lambda s. \mu (sel s)$ )

```

```

by (auto simp: positive_def intro!: empty sel nonneg)

```

```

show countably_additive (G ' I) ( $\lambda s. \mu (sel s)$ )

```

```

proof (rule countably_additiveI)

```

```

fix A :: nat  $\Rightarrow$  'a set assume range A  $\subseteq$  G ' I disjoint_family A ( $\bigcup i. A i$ )
 $\in$  G ' I

```

```

then show ( $\sum i. \mu (sel (A i)) = \mu (sel (\bigcup i. A i))$ )

```

```

by (intro add) (auto simp: sel_image_subset_iff_funcset comp_def Pi_iff)

```

```

intro!: sel)
qed
qed
then obtain  $\mu'$  where  $\mu': \forall s \in G \text{ ' } I. \mu' s = \mu (sel s) \text{ measure\_space } \Omega (sigma\_sets$ 
 $\Omega (G \text{ ' } I)) \mu'$ 
by metis

show ?thesis
proof (rule emeasure_extend_measure[OF M])
{ fix i assume  $i \in I$  then show  $\mu' (G i) = \mu i$ 
using  $\mu'$  by (auto intro!: inj sel) }
show  $G \text{ ' } I \subseteq Pow \Omega$ 
by (rule space_closed)
then show positive (sets M)  $\mu'$  countably_additive (sets M)  $\mu'$ 
using  $\mu'$  by (simp_all add: M sets_extend_measure measure_space_def)
qed fact
qed

proposition extend_measure_caratheodory_pair:
fixes  $G :: 'i \Rightarrow 'j \Rightarrow 'a \text{ set}$ 
assumes  $M: M = extend\_measure \Omega \{(a, b). P a b\} (\lambda(a, b). G a b) (\lambda(a, b).$ 
 $\mu a b)$ 
assumes  $P i j$ 
assumes semiring_of_sets  $\Omega \{G a b \mid a b. P a b\}$ 
assumes empty:  $\bigwedge i j. P i j \Rightarrow G i j = \{\} \Rightarrow \mu i j = 0$ 
assumes inj:  $\bigwedge i j k l. P i j \Rightarrow P k l \Rightarrow G i j = G k l \Rightarrow \mu i j = \mu k l$ 
assumes nonneg:  $\bigwedge i j. P i j \Rightarrow 0 \leq \mu i j$ 
assumes add:  $\bigwedge A::nat \Rightarrow 'i. \bigwedge B::nat \Rightarrow 'j. \bigwedge j k.$ 
 $(\bigwedge n. P (A n) (B n)) \Rightarrow P j k \Rightarrow disjoint\_family (\lambda n. G (A n) (B n)) \Rightarrow$ 
 $(\bigcup i. G (A i) (B i)) = G j k \Rightarrow (\sum n. \mu (A n) (B n)) = \mu j k$ 
shows emeasure M (G i j) =  $\mu i j$ 
proof -
have emeasure M (( $\lambda(a, b). G a b$ ) (i, j)) = ( $\lambda(a, b). \mu a b$ ) (i, j)
proof (rule extend_measure_caratheodory[OF M])
show semiring_of_sets  $\Omega ((\lambda(a, b). G a b) \text{ ' } \{(a, b). P a b\})$ 
using semiring by (simp add: image_def conj_commute)
next
fix  $A :: nat \Rightarrow ('i \times 'j)$  and j assume  $A \in UNIV \rightarrow \{(a, b). P a b\} j \in \{(a,$ 
 $b). P a b\}$ 
disjoint_family (( $\lambda(a, b). G a b$ )  $\circ A$ )
 $(\bigcup i. case A i of (a, b) \Rightarrow G a b) = (case j of (a, b) \Rightarrow G a b)$ 
then show  $(\sum n. case A n of (a, b) \Rightarrow \mu a b) = (case j of (a, b) \Rightarrow \mu a b)$ 
using add[of  $\lambda i. fst (A i) \lambda i. snd (A i) fst j snd j$ ]
by (simp add: split_beta' comp_def Pi_iff)
qed (auto split: prod.splits intro: assms)
then show ?thesis by simp
qed
end

```


7.9 Bochner Integration for Vector-Valued Functions

```
theory Bochner_Integration
  imports Finite_Product_Measure
begin
```

In the following development of the Bochner integral we use second countable topologies instead of separable spaces. A second countable topology is also separable.

```
proposition borel_measurable_implies_sequence_metric:
  fixes f :: 'a  $\Rightarrow$  'b :: {metric_space, second_countable_topology}
  assumes [measurable]: f  $\in$  borel_measurable M
  shows  $\exists F. (\forall i. \text{simple\_function } M (F i)) \wedge (\forall x \in \text{space } M. (\lambda i. F i x) \longrightarrow f x) \wedge$ 
     $(\forall i. \forall x \in \text{space } M. \text{dist } (F i x) z \leq 2 * \text{dist } (f x) z)$ 
```

```
proof -
```

```
  obtain D :: 'b set where countable D and D:  $\bigwedge X. \text{open } X \implies X \neq \{\} \implies$ 
 $\exists d \in D. d \in X$ 
  by (erule countable_dense_setE)
```

```
define e where e = from_nat_into D
{ fix n x
  obtain d where d  $\in$  D and d: d  $\in$  ball x (1 / Suc n)
  using D[of ball x (1 / Suc n)] by auto
  from  $\langle d \in D \rangle$  D[of UNIV]  $\langle$ countable D $\rangle$  obtain i where d = e i
  unfolding e_def by (auto dest: from_nat_into_surj)
  with d have  $\exists i. \text{dist } x (e i) < 1 / \text{Suc } n$ 
  by auto }
note e = this
```

```
define A where [abs_def]: A m n =
   $\{x \in \text{space } M. \text{dist } (f x) (e n) < 1 / (\text{Suc } m) \wedge 1 / (\text{Suc } m) \leq \text{dist } (f x) z\}$  for
m n
```

```
define B where [abs_def]: B m = disjointed (A m) for m
```

```
define m where [abs_def]: m N x = Max  $\{m. m \leq N \wedge x \in (\bigcup n \leq N. B m n)\}$ 
for N x
```

```
define F where [abs_def]: F N x =
  (if  $(\exists m \leq N. x \in (\bigcup n \leq N. B m n)) \wedge (\exists n \leq N. x \in B (m N x) n)$ 
  then e (LEAST n. x  $\in$  B (m N x) n) else z) for N x
```

```
have B_imp_A[intro, simp]:  $\bigwedge x m n. x \in B m n \implies x \in A m n$ 
  using disjointed_subset[of A m for m] unfolding B_def by auto
```

```
{ fix m
  have  $\bigwedge n. A m n \in \text{sets } M$ 
  by (auto simp: A_def)
  then have  $\bigwedge n. B m n \in \text{sets } M$ 
```

```

    using sets.range_disjointed_sets[of A m M] by (auto simp: B_def) }
  note this[measurable]

  { fix N i x assume  $\exists m \leq N. x \in (\bigcup_{n \leq N}. B\ m\ n)$ 
    then have  $m\ N\ x \in \{m :: nat. m \leq N \wedge x \in (\bigcup_{n \leq N}. B\ m\ n)\}$ 
      unfolding m_def by (intro Max_in) auto
    then have  $m\ N\ x \leq N \exists n \leq N. x \in B\ (m\ N\ x)\ n$ 
      by auto }
  note m = this

  { fix j N i x assume  $j \leq N\ i \leq N\ x \in B\ j\ i$ 
    then have  $j \leq m\ N\ x$ 
      unfolding m_def by (intro Max_ge) auto }
  note m_upper = this

show ?thesis
  unfolding simple_function_def
  proof (safe intro!: exI[of _ F])
    have [measurable]:  $\bigwedge i. F\ i \in \text{borel\_measurable}\ M$ 
      unfolding F_def m_def by measurable
    show  $\bigwedge x\ i. F\ i\ -\ '\{x\} \cap \text{space}\ M \in \text{sets}\ M$ 
      by measurable

  { fix i
    { fix n x assume  $x \in B\ (m\ i\ x)\ n$ 
      then have  $(LEAST\ n. x \in B\ (m\ i\ x)\ n) \leq n$ 
        by (intro Least_le)
      also assume  $n \leq i$ 
      finally have  $(LEAST\ n. x \in B\ (m\ i\ x)\ n) \leq i . }$ 
    then have  $F\ i\ '\text{space}\ M \subseteq \{z\} \cup e\ '\{.. i\}$ 
      by (auto simp: F_def)
    then show finite  $(F\ i\ '\text{space}\ M)$ 
      by (rule finite_subset) auto }

  { fix N i n x assume  $i \leq N\ n \leq N\ x \in B\ i\ n$ 
    then have 1:  $\exists m \leq N. x \in (\bigcup_{n \leq N}. B\ m\ n)$  by auto
    from m[OF this] obtain n where  $n: m\ N\ x \leq N\ n \leq N\ x \in B\ (m\ N\ x)\ n$ 
  by auto
  moreover
  define L where  $L = (LEAST\ n. x \in B\ (m\ N\ x)\ n)$ 
  have  $\text{dist}\ (f\ x)\ (e\ L) < 1 / \text{Suc}\ (m\ N\ x)$ 
  proof -
    have  $x \in B\ (m\ N\ x)\ L$ 
      using n(3) unfolding L_def by (rule LeastI)
    then have  $x \in A\ (m\ N\ x)\ L$ 
      by auto
    then show ?thesis
      unfolding A_def by simp
  qed

```

```

ultimately have  $\text{dist } (f x) (F N x) < 1 / \text{Suc } (m N x)$ 
  by (auto simp: F_def L_def) }
note * = this

fix x assume  $x \in \text{space } M$ 
show  $(\lambda i. F i x) \longrightarrow f x$ 
proof (cases  $f x = z$ )
  case True
  then have  $\bigwedge i n. x \notin A i n$ 
    unfolding A_def by auto
  then show ?thesis
    by (metis B_imp_A F_def LIMSEQ_def True dist_self)
next
  case False
  show ?thesis
  proof (rule tendstoI)
    fix e :: real assume  $0 < e$ 
    with  $\langle f x \neq z \rangle$  obtain n where  $1 / \text{Suc } n < e$   $1 / \text{Suc } n < \text{dist } (f x) z$ 
      by (metis dist_nz order_less_trans neq_iff nat_approx_posE)
    with  $\langle x \in \text{space } M \rangle \langle f x \neq z \rangle$  have  $x \in (\bigcup i. B n i)$ 
      unfolding A_def B_def UN_disjointed_eq using e by auto
    then obtain i where  $x \in B n i$  by auto

    show eventually  $(\lambda i. \text{dist } (F i x) (f x) < e)$  sequentially
      using eventually_ge_at_top[of  $\max n i$ ]
    proof eventually_elim
      fix j assume  $j: \max n i \leq j$ 
      with i have  $\text{dist } (f x) (F j x) < 1 / \text{Suc } (m j x)$ 
        by (intro *[OF _ _ i]) auto
      also have  $\dots \leq 1 / \text{Suc } n$ 
        using j m_upper[OF _ _ i]
        by (auto simp: field_simps)
      also note  $\langle 1 / \text{Suc } n < e \rangle$ 
      finally show  $\text{dist } (F j x) (f x) < e$ 
        by (simp add: less_imp_le dist_commute)
    qed
  qed
qed
fix i
{ fix n m assume  $x \in A n m$ 
  then have  $\text{dist } (e m) (f x) + \text{dist } (f x) z \leq 2 * \text{dist } (f x) z$ 
    unfolding A_def by (auto simp: dist_commute)
  also have  $\text{dist } (e m) z \leq \text{dist } (e m) (f x) + \text{dist } (f x) z$ 
    by (rule dist_triangle)
  finally (xtrans) have  $\text{dist } (e m) z \leq 2 * \text{dist } (f x) z . \}$ 
then show  $\text{dist } (F i x) z \leq 2 * \text{dist } (f x) z$ 
  unfolding F_def
  by (smt (verit, ccfv_threshold) LeastI2 B_imp_A dist_eq_0_iff zero_le_dist)
qed

```

qed

lemma

fixes $f :: 'a \Rightarrow 'b :: \text{semiring}_1$ assumes $\text{finite } A$
 shows $\text{sum_mult_indicator}[simp]: (\sum x \in A. f\ x * \text{indicator } (B\ x) (g\ x)) =$
 $(\sum x \in \{x \in A. g\ x \in B\ x\}. f\ x)$
 and $\text{sum_indicator_mult}[simp]: (\sum x \in A. \text{indicator } (B\ x) (g\ x) * f\ x) =$
 $(\sum x \in \{x \in A. g\ x \in B\ x\}. f\ x)$
 unfolding indicator_def
 using assms by ($\text{auto intro!}: \text{sum.mono_neutral_cong_right split}: \text{if_split_asm}$)

lemma $\text{borel_measurable_induct_real}[\text{consumes } 2, \text{case_names set mult add seq}]$:

fixes $P :: ('a \Rightarrow \text{real}) \Rightarrow \text{bool}$
 assumes $u: u \in \text{borel_measurable } M \wedge x. 0 \leq u\ x$
 assumes $\text{set}: \bigwedge A. A \in \text{sets } M \implies P (\text{indicator } A)$
 assumes $\text{mult}: \bigwedge u\ c. 0 \leq c \implies u \in \text{borel_measurable } M \implies (\bigwedge x. 0 \leq u\ x) \implies P\ u \implies P (\lambda x. c * u\ x)$
 assumes $\text{add}: \bigwedge u\ v. u \in \text{borel_measurable } M \implies (\bigwedge x. 0 \leq u\ x) \implies P\ u \implies v \in \text{borel_measurable } M \implies (\bigwedge x. 0 \leq v\ x) \implies (\bigwedge x. x \in \text{space } M \implies u\ x = 0 \vee v\ x = 0) \implies P\ v \implies P (\lambda x. v\ x + u\ x)$
 assumes $\text{seq}: \bigwedge U. (\bigwedge i. U\ i \in \text{borel_measurable } M) \implies (\bigwedge i\ x. 0 \leq U\ i\ x) \implies (\bigwedge i. P (U\ i)) \implies \text{incseq } U \implies (\bigwedge x. x \in \text{space } M \implies (\lambda i. U\ i\ x) \longrightarrow u\ x) \implies P\ u$

shows $P\ u$

proof -

have $(\lambda x. \text{ennreal } (u\ x)) \in \text{borel_measurable } M$ using u by auto

from $\text{borel_measurable_implies_simple_function_sequence}'[\text{OF this}]$

obtain U where $U: \bigwedge i. \text{simple_function } M (U\ i) \text{ incseq } U \wedge i\ x. U\ i\ x < \text{top}$
 and

$\text{sup}: \bigwedge x. (\text{SUP } i. U\ i\ x) = \text{ennreal } (u\ x)$

by blast

define U' where $[\text{abs_def}]: U'\ i\ x = \text{indicator } (\text{space } M)\ x * \text{enn2real } (U\ i\ x)$
 for $i\ x$

then have $U'\ \text{sf}[\text{measurable}]: \bigwedge i. \text{simple_function } M (U'\ i)$

using U by ($\text{auto intro!}: \text{simple_function_compose1}[\text{where } g = \text{enn2real}]$)

show $P\ u$

proof (rule seq)

show U' : $U'\ i \in \text{borel_measurable } M \wedge x. 0 \leq U'\ i\ x$ for i

using $U'\ \text{sf}$ by ($\text{auto simp}: U'\ \text{def borel_measurable_simple_function}$)

show $\text{incseq } U'$

using $U(2,3)$

by ($\text{auto simp}: \text{incseq_def le_fun_def image_iff eq_commute } U'\ \text{def indicator_def enn2real_mono}$)

fix x assume $x: x \in \text{space } M$

have $(\lambda i. U\ i\ x) \longrightarrow (\text{SUP } i. U\ i\ x)$

using $U(2)$ by (intro LIMSEQ_SUP) ($\text{auto simp}: \text{incseq_def le_fun_def}$)

```

moreover have  $(\lambda i. U\ i\ x) = (\lambda i. ennreal\ (U'\ i\ x))$ 
  using  $x\ U(3)$  by  $(auto\ simp:\ fun\_eq\_iff\ U'\_def\ image\_iff\ eq\_commute)$ 
moreover have  $(SUP\ i. U\ i\ x) = ennreal\ (u\ x)$ 
  using  $sup\ u(2)$  by  $(simp\ add:\ max\_def)$ 
ultimately show  $(\lambda i. U'\ i\ x) \longrightarrow u\ x$ 
  using  $u\ U'$  by  $simp$ 
next
fix  $i$ 
have  $U'\ i\ 'space\ M \subseteq enn2real\ '(U\ i\ 'space\ M)\ finite\ (U\ i\ 'space\ M)$ 
  unfolding  $U'\_def$  using  $U(1)$  by  $(auto\ dest:\ simple\_functionD)$ 
then have  $fin:\ finite\ (U'\ i\ 'space\ M)$ 
  by  $(metis\ finite\_subset\ finite\_imageI)$ 
moreover have  $\bigwedge z. \{y. U'\ i\ z = y \wedge y \in U'\ i\ 'space\ M \wedge z \in space\ M\} =$ 
 $(if\ z \in space\ M\ then\ \{U'\ i\ z\}\ else\ \{\})$ 
  by  $auto$ 
ultimately have  $U': (\lambda z. \sum y \in U'\ i\ 'space\ M. y * indicator\ \{x \in space\ M. U'\ i\ x = y\}\ z) = U'\ i$ 
  by  $(simp\ add:\ U'\_def\ fun\_eq\_iff)$ 
have  $\bigwedge x. x \in U'\ i\ 'space\ M \implies 0 \leq x$ 
  by  $(auto\ simp:\ U'\_def)$ 
with  $fin$  have  $P\ (\lambda z. \sum y \in U'\ i\ 'space\ M. y * indicator\ \{x \in space\ M. U'\ i\ x = y\}\ z)$ 
proof  $induct$ 
  case  $empty$  from  $set[of\ \{\}]\ show\ ?case$ 
  by  $(simp\ add:\ indicator\_def[abs\_def])$ 
next
case  $(insert\ x\ F)$ 
from  $insert.prem\ s$  have  $nonneg:\ x \geq 0 \wedge y. y \in F \implies y \geq 0$ 
  by  $simp\_all$ 
hence  $*$ :  $P\ (\lambda xa. x * indicat\_real\ \{x' \in space\ M. U'\ i\ x' = x\}\ xa)$ 
  by  $(intro\ mult\ set)\ auto$ 
have  $P\ (\lambda z. x * indicat\_real\ \{x' \in space\ M. U'\ i\ x' = x\}\ z +$ 
 $(\sum y \in F. y * indicat\_real\ \{x \in space\ M. U'\ i\ x = y\}\ z))$ 
  using  $insert(1-3)$ 
  by  $(intro\ add\ * \ sum\_nonneg\ mult\_nonneg\_nonneg)$ 
 $(auto\ simp:\ nonneg\ indicator\_def\ of\_bool\_def\ sum\_nonneg\_eq\_0\_iff)$ 
thus  $?case$ 
  using  $insert.hyps$  by  $(subst\ sum.insert)\ auto$ 
qed
with  $U'$  show  $P\ (U'\ i)$  by  $simp$ 
qed
qed

```

lemma $scaleR_cong_right:$

```

fixes  $x :: 'a :: real\_vector$ 
shows  $(x \neq 0 \implies r = p) \implies r *_{\mathbb{R}} x = p *_{\mathbb{R}} x$ 
by  $auto$ 

```

inductive $simple_bochner_integrable :: 'a\ measure \Rightarrow ('a \Rightarrow 'b::real_vector) \Rightarrow$

bool for $M f$ where

$simple_function\ M\ f \implies emeasure\ M\ \{y \in space\ M.\ f\ y \neq 0\} \neq \infty \implies$
 $simple_bochner_integrable\ M\ f$

lemma *simple_bochner_integrable_compose2*:

assumes p_0 : $p\ 0\ 0 = 0$

shows $simple_bochner_integrable\ M\ f \implies simple_bochner_integrable\ M\ g \implies$
 $simple_bochner_integrable\ M\ (\lambda x.\ p\ (f\ x)\ (g\ x))$

proof (*safe intro!*: *simple_bochner_integrable.intros elim!*: *simple_bochner_integrable.cases del: notI*)

assume sf : $simple_function\ M\ f\ simple_function\ M\ g$

then show $simple_function\ M\ (\lambda x.\ p\ (f\ x)\ (g\ x))$

by (*rule simple_function_compose2*)

from sf **have** [*measurable*]:

$f \in measurable\ M\ (count_space\ UNIV)$

$g \in measurable\ M\ (count_space\ UNIV)$

by (*auto intro: measurable_simple_function*)

assume fin : $emeasure\ M\ \{y \in space\ M.\ f\ y \neq 0\} \neq \infty\ emeasure\ M\ \{y \in space\ M.\ g\ y \neq 0\} \neq \infty$

have $emeasure\ M\ \{x \in space\ M.\ p\ (f\ x)\ (g\ x) \neq 0\} \leq$
 $emeasure\ M\ (\{x \in space\ M.\ f\ x \neq 0\} \cup \{x \in space\ M.\ g\ x \neq 0\})$

by (*intro emeasure_mono*) (*auto simp: p_0*)

also have $\dots \leq emeasure\ M\ \{x \in space\ M.\ f\ x \neq 0\} + emeasure\ M\ \{x \in space\ M.\ g\ x \neq 0\}$

by (*intro emeasure_subadditive*) *auto*

finally show $emeasure\ M\ \{y \in space\ M.\ p\ (f\ y)\ (g\ y) \neq 0\} \neq \infty$

using fin **by** (*auto simp: top_unique*)

qed

lemma *simple_function_finite_support*:

assumes f : $simple_function\ M\ f$ **and** fin : $(\int^+ x.\ f\ x\ \partial M) < \infty$ **and** nn : $\bigwedge x.\ 0 \leq f\ x$

shows $emeasure\ M\ \{x \in space\ M.\ f\ x \neq 0\} \neq \infty$

proof *cases*

from f **have** $meas[measurable]$: $f \in borel_measurable\ M$

by (*rule borel_measurable_simple_function*)

assume non_empty : $\exists x \in space\ M.\ f\ x \neq 0$

define m **where** $m = Min\ (f' \text{space}\ M - \{0\})$

have $m \in f' \text{space}\ M - \{0\}$

unfolding m_def **using** $f\ non_empty$ **by** (*intro Min_in*) (*auto simp: simple_function_def*)

then have m : $0 < m$

using nn **by** (*auto simp: less_le*)

```

from  $m$  have  $m * \text{emeasure } M \{x \in \text{space } M. 0 \neq f x\} =$ 
   $(\int^{+x}. m * \text{indicator } \{x \in \text{space } M. 0 \neq f x\} x \partial M)$ 
  using  $f$  by (intro nn_integral_cmult_indicator[symmetric]) auto
also have  $\dots \leq (\int^{+x}. f x \partial M)$ 
  using AE_space
proof (intro nn_integral_mono_AE, eventually_elim)
  fix  $x$  assume  $x \in \text{space } M$ 
  with  $nn$  show  $m * \text{indicator } \{x \in \text{space } M. 0 \neq f x\} x \leq f x$ 
    using  $f$  by (auto split: split_indicator simp: simple_function_def m_def)
qed
also note  $\langle \dots < \infty \rangle$ 
finally show ?thesis
  using  $m$  by (auto simp: ennreal_mult_less_top)
next
assume  $\neg (\exists x \in \text{space } M. f x \neq 0)$ 
with  $nn$  have  $*: \{x \in \text{space } M. f x \neq 0\} = \{\}$ 
  by auto
show ?thesis unfolding  $*$  by simp
qed

lemma simple_bochner_integrableI_bounded:
  assumes  $f: \text{simple\_function } M f$  and  $\text{fin}: (\int^{+x}. \text{norm } (f x) \partial M) < \infty$ 
  shows simple_bochner_integrable  $M f$ 
proof
  have  $\text{emeasure } M \{y \in \text{space } M. \text{ennreal } (\text{norm } (f y)) \neq 0\} \neq \infty$ 
    using simple_function_finite_support simple_function_compose1  $f \text{ fin}$  by force
  then show  $\text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty$  by simp
qed fact

definition simple_bochner_integral :: 'a measure  $\Rightarrow$  ('a  $\Rightarrow$  'b::real_vector)  $\Rightarrow$  'b
where
  simple_bochner_integral  $M f = (\sum y \in f' \text{space } M. \text{measure } M \{x \in \text{space } M. f x = y\} *_{\mathbb{R}} y)$ 

proposition simple_bochner_integral_partition:
  assumes  $f: \text{simple\_bochner\_integrable } M f$  and  $g: \text{simple\_function } M g$ 
  assumes  $\text{sub}: \bigwedge x y. x \in \text{space } M \Longrightarrow y \in \text{space } M \Longrightarrow g x = g y \Longrightarrow f x = f y$ 
  assumes  $v: \bigwedge x. x \in \text{space } M \Longrightarrow f x = v (g x)$ 
  shows simple_bochner_integral  $M f = (\sum y \in g' \text{space } M. \text{measure } M \{x \in \text{space } M. g x = y\} *_{\mathbb{R}} v y)$ 
  (is _ = ?r)
proof -
from  $f g$  have [simp]: finite  $(f' \text{space } M)$  finite  $(g' \text{space } M)$ 
  by (auto simp: simple_function_def elim: simple_bochner_integrable.cases)

from  $f$  have [measurable]:  $f \in \text{measurable } M$  (count_space UNIV)
  by (auto intro: measurable_simple_function elim: simple_bochner_integrable.cases)

from  $g$  have [measurable]:  $g \in \text{measurable } M$  (count_space UNIV)

```

```

by (auto intro: measurable_simple_function elim: simple_bochner_integrable.cases)

{ fix y assume y ∈ space M
  then have f ' space M ∩ {i. ∃ x ∈ space M. i = f x ∧ g y = g x} = {v (g y)}
    by (auto cong: sub simp: v[symmetric]) }
note eq = this

have simple_bochner_integral M f =
  (∑ y ∈ f ' space M. (∑ z ∈ g ' space M.
    if ∃ x ∈ space M. y = f x ∧ z = g x then measure M {x ∈ space M. g x = z}
  else 0) *R y)
unfolding simple_bochner_integral_def
proof (safe intro!: sum.cong scaleR_cong_right)
fix y assume y: y ∈ space M f y ≠ 0
have [simp]: g ' space M ∩ {z. ∃ x ∈ space M. f y = f x ∧ z = g x} =
  {z. ∃ x ∈ space M. f y = f x ∧ z = g x}
by auto
have eq: {x ∈ space M. f x = f y} =
  (∪ i ∈ {z. ∃ x ∈ space M. f y = f x ∧ z = g x}. {x ∈ space M. g x = i})
by (auto simp: eq_commute cong: sub rev_conj_cong)
have finite (g ' space M) by simp
then have finite {z. ∃ x ∈ space M. f y = f x ∧ z = g x}
by (rule rev_finite_subset) auto
moreover
{ fix x assume x ∈ space M f x = f y
  then have x ∈ space M f x ≠ 0
    using y by auto
  then have emeasure M {y ∈ space M. g y = g x} ≤ emeasure M {y ∈ space
M. f y ≠ 0}
    by (auto intro!: emeasure_mono cong: sub)
  then have emeasure M {x ∈ space M. g x = g x} < ∞
    using f by (auto simp: simple_bochner_integrable.simps less_top) }
ultimately
show measure M {x ∈ space M. f x = f y} =
  (∑ z ∈ g ' space M. if ∃ x ∈ space M. f y = f x ∧ z = g x then measure M {x
∈ space M. g x = z} else 0)
  apply (simp add: sum.If_cases eq)
  apply (subst measure_finite_Union[symmetric])
  apply (auto simp: disjoint_family_on_def less_top)
done
qed
also have ... = (∑ y ∈ f ' space M. (∑ z ∈ g ' space M.
  if ∃ x ∈ space M. y = f x ∧ z = g x then measure M {x ∈ space M. g x = z} *R
y else 0))
by (auto intro!: sum.cong simp: scaleR_sum_left)
also have ... = ?r
by (subst sum.swap)
  (auto intro!: sum.cong simp: sum.If_cases scaleR_sum_right[symmetric] eq)
finally show simple_bochner_integral M f = ?r .

```


qed

lemma *simple_bochner_integral_add*:

assumes *f*: *simple_bochner_integrable* *M* *f* **and** *g*: *simple_bochner_integrable* *M* *g*

shows *simple_bochner_integral* *M* ($\lambda x. f\ x + g\ x$) =

simple_bochner_integral *M* *f* + *simple_bochner_integral* *M* *g*

proof –

from *f* *g* **have** *simple_bochner_integral* *M* ($\lambda x. f\ x + g\ x$) =

($\sum_{y \in (\lambda x. (f\ x, g\ x)) \text{ ' } \textit{space } M. \textit{measure } M \{x \in \textit{space } M. (f\ x, g\ x) = y\}} *R$
(*fst* *y* + *snd* *y*))

by (*intro simple_bochner_integral_partition*)

(*auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases*)

moreover from *f* *g* **have** *simple_bochner_integral* *M* *f* =

($\sum_{y \in (\lambda x. (f\ x, g\ x)) \text{ ' } \textit{space } M. \textit{measure } M \{x \in \textit{space } M. (f\ x, g\ x) = y\}} *R$
fst *y*)

by (*intro simple_bochner_integral_partition*)

(*auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases*)

moreover from *f* *g* **have** *simple_bochner_integral* *M* *g* =

($\sum_{y \in (\lambda x. (f\ x, g\ x)) \text{ ' } \textit{space } M. \textit{measure } M \{x \in \textit{space } M. (f\ x, g\ x) = y\}} *R$
snd *y*)

by (*intro simple_bochner_integral_partition*)

(*auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases*)

ultimately show *?thesis*

by (*simp add: sum.distrib[symmetric] scaleR_add_right*)

qed

lemma *simple_bochner_integral_linear*:

assumes *linear* *f*

assumes *g*: *simple_bochner_integrable* *M* *g*

shows *simple_bochner_integral* *M* ($\lambda x. f\ (g\ x)$) = *f* (*simple_bochner_integral* *M* *g*)

proof –

interpret *linear* *f* **by** *fact*

from *g* **have** *simple_bochner_integral* *M* ($\lambda x. f\ (g\ x)$) =

($\sum_{y \in g \text{ ' } \textit{space } M. \textit{measure } M \{x \in \textit{space } M. g\ x = y\}} *R f\ y$)

by (*intro simple_bochner_integral_partition*)

(*auto simp: simple_bochner_integrable_compose2*[**where** *p*= $\lambda x y. f\ x$]

elim: simple_bochner_integrable.cases)

also have ... = *f* (*simple_bochner_integral* *M* *g*)

by (*simp add: simple_bochner_integral_def sum scale*)

finally show *?thesis* .

qed

lemma *simple_bochner_integral_minus*:

assumes *f*: *simple_bochner_integrable* *M* *f*

shows *simple_bochner_integral* *M* ($\lambda x. - f\ x$) = - *simple_bochner_integral* *M* *f*

proof –

2366

from *linear_uminus f* **show** *?thesis*
by (*rule simple_bochner_integral_linear*)
qed

lemma *simple_bochner_integral_diff*:
assumes *f: simple_bochner_integrable M f* **and** *g: simple_bochner_integrable M g*
shows *simple_bochner_integral M (λx. f x - g x) =*
simple_bochner_integral M f - simple_bochner_integral M g
unfolding *diff_conv_add_uminus* **using** *f g*
by (*subst simple_bochner_integral_add*)
(*auto simp: simple_bochner_integral_minus simple_bochner_integrable_compose2* [**where**
p=λx y. - y])

lemma *simple_bochner_integral_norm_bound*:
assumes *f: simple_bochner_integrable M f*
shows *norm (simple_bochner_integral M f) ≤ simple_bochner_integral M (λx.*
norm (f x))
proof -
have *norm (simple_bochner_integral M f) ≤*
*(∑ y∈f ' space M. norm (measure M {x ∈ space M. f x = y} *_R y))*
unfolding *simple_bochner_integral_def* **by** (*rule norm_sum*)
also have *... = (∑ y∈f ' space M. measure M {x ∈ space M. f x = y} *_R norm*
y)
by *simp*
also have *... = simple_bochner_integral M (λx. norm (f x))*
using *f*
by (*intro simple_bochner_integral_partition[symmetric]*)
(*auto intro: f simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases*)
finally show *?thesis .*
qed

lemma *simple_bochner_integral_nonneg[simp]*:
fixes *f :: 'a ⇒ real*
shows *(∧x. 0 ≤ f x) ⇒ 0 ≤ simple_bochner_integral M f*
by (*force simp: simple_bochner_integral_def intro: sum_nonneg*)

lemma *simple_bochner_integral_eq_nn_integral*:
assumes *f: simple_bochner_integrable M f* *∧x. 0 ≤ f x*
shows *simple_bochner_integral M f = (∫⁺x. f x ∂M)*
proof -
have *ennreal_cong_mult: (x ≠ 0 ⇒ y = z) ⇒ ennreal x * y = ennreal x * z*
for *x y z*
by *fastforce*

have [*measurable*]: *f ∈ borel_measurable M*
by (*meson borel_measurable_simple_function f(1) simple_bochner_integrable.cases*)

{ **fix** *y* **assume** *y: y ∈ space M f y ≠ 0*

```

have ennreal (measure M {x ∈ space M. f x = f y}) = emeasure M {x ∈ space
M. f x = f y}
proof (rule emeasure_eq_ennreal_measure[symmetric])
  have emeasure M {x ∈ space M. f x = f y} ≤ emeasure M {x ∈ space M. f
x ≠ 0}
  using y by (intro emeasure_mono) auto
  with f show emeasure M {x ∈ space M. f x = f y} ≠ top
  by (auto simp: simple_bochner_integrable.simps top_unique)
qed
moreover have {x ∈ space M. f x = f y} = (λx. ennreal (f x)) - ‘ {ennreal (f
y)} ∩ space M
  using f by auto
ultimately have ennreal (measure M {x ∈ space M. f x = f y}) =
emeasure M ((λx. ennreal (f x)) - ‘ {ennreal (f y)} ∩ space M) by simp }
with f have simple_bochner_integral M f = (∫S x. f x ∂M)
unfolding simple_integral_def
by (subst simple_bochner_integral_partition[OF f(1), where g=λx. ennreal (f
x) and v=enn2real])
  (auto intro: f simple_function_compose1 elim: simple_bochner_integrable.cases
  intro!: sum.cong ennreal_cong_mult
  simp: ac_simps ennreal_mult
  simp flip: sum_ennreal)
also have ... = (∫+ x. f x ∂M)
  using f
  by (metis nn_integral_eq_simple_integral simple_bochner_integrable.cases
simple_function_compose1)
finally show ?thesis .
qed

```

lemma simple_bochner_integral_bounded:

```

fixes f :: 'a ⇒ 'b::{real_normed_vector, second_countable_topology}
assumes f[measurable]: f ∈ borel_measurable M
assumes s: simple_bochner_integrable M s and t: simple_bochner_integrable M
t
shows ennreal (norm (simple_bochner_integral M s - simple_bochner_integral
M t)) ≤
(∫+ x. norm (f x - s x) ∂M) + (∫+ x. norm (f x - t x) ∂M)
(is ennreal (norm (?s - ?t)) ≤ ?S + ?T)
proof -
have [measurable]: s ∈ borel_measurable M t ∈ borel_measurable M
using s t by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

have ennreal (norm (?s - ?t)) = norm (simple_bochner_integral M (λx. s x -
t x))
  using s t by (subst simple_bochner_integral_diff) auto
also have ... ≤ simple_bochner_integral M (λx. norm (s x - t x))
  using simple_bochner_integrable_compose2[of (-) M s t] s t
  by (auto intro!: simple_bochner_integral_norm_bound)
also have ... = (∫+ x. norm (s x - t x) ∂M)

```

```

using simple_bochner_integrable_compose2[of  $\lambda x y. \text{norm } (x - y) M s t$ ] s t
by (auto intro!: simple_bochner_integral_eq_nn_integral)
also have ...  $\leq (\int^+ x. \text{ennreal } (\text{norm } (f x - s x)) + \text{ennreal } (\text{norm } (f x - t x)))$ 
 $\partial M$ 
proof -
  have  $\bigwedge x. x \in \text{space } M \implies$ 
     $\text{norm } (s x - t x) \leq \text{norm } (f x - s x) + \text{norm } (f x - t x)$ 
  by (metis dual_order.refl norm_diff_triangle_le norm_minus_commute)
  then show ?thesis
  by (metis (mono_tags, lifting) ennreal_leI ennreal_plus_nn_integral_mono
norm_ge_zero)
qed
also have ... = ?S + ?T
by (rule nn_integral_add) auto
finally show ?thesis .
qed

```

inductive has_bochner_integral :: 'a measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b::{real_normed_vector, second_countable_topology} \Rightarrow bool

```

for M f x where
  f  $\in$  borel_measurable M  $\implies$ 
    ( $\bigwedge i. \text{simple\_bochner\_integrable } M (s i)$ )  $\implies$ 
    ( $\lambda i. \int^+ x. \text{norm } (f x - s i x) \partial M \longrightarrow 0$ )  $\implies$ 
    ( $\lambda i. \text{simple\_bochner\_integral } M (s i) \longrightarrow x$ )  $\implies$ 
    has_bochner_integral M f x

```

lemma has_bochner_integral_cong:

```

assumes M = N  $\bigwedge x. x \in \text{space } N \implies f x = g x x = y$ 
shows has_bochner_integral M f x  $\longleftrightarrow$  has_bochner_integral N g y
unfolding has_bochner_integral.simps assms(1,3)
using assms(2) by (simp cong: measurable_cong_simp nn_integral_cong_simp)

```

lemma has_bochner_integral_cong_AE:

```

  f  $\in$  borel_measurable M  $\implies$  g  $\in$  borel_measurable M  $\implies$  (AE x in M. f x = g x)  $\implies$ 
    has_bochner_integral M f x  $\longleftrightarrow$  has_bochner_integral M g x
unfolding has_bochner_integral.simps
by (intro arg_cong[where f=Ex] ext conj_cong rev_conj_cong refl arg_cong[where
f= $\lambda x. x \longrightarrow 0$ ]
nn_integral_cong_AE)
  auto

```

lemma borel_measurable_has_bochner_integral:

```

  has_bochner_integral M f x  $\implies$  f  $\in$  borel_measurable M
by (rule has_bochner_integral.cases)

```

lemma borel_measurable_has_bochner_integral'[measurable_dest]:

```

  has_bochner_integral M f x  $\implies$  g  $\in$  measurable N M  $\implies$  ( $\lambda x. f (g x)$ )  $\in$ 
  borel_measurable N

```

using borel_measurable_has_bochner_integral[measurable] by measurable

lemma *has_bochner_integral_simple_bochner_integrable*:

simple_bochner_integrable M f \implies has_bochner_integral M f (simple_bochner_integral M f)

by (rule *has_bochner_integral.intros*[**where** *s= λ _. f*])
 (auto intro: *borel_measurable_simple_function*
elim: simple_bochner_integrable.cases
simp flip: zero_enreal_def)

lemma *has_bochner_integral_real_indicator*:

assumes [*measurable*]: *A \in sets M* **and** *A: emeasure M A $<$ ∞*

shows *has_bochner_integral M (indicator A) (measure M A)*

proof –

have *sbi: simple_bochner_integrable M (indicator A::'a \Rightarrow real)*

proof

have *{y \in space M. (indicator A y::real) \neq 0} = A*

using *sets.sets_into_space*[*OF* \langle *A \in sets M \rangle] **by** (auto split: *split_indicator*)*

then show *emeasure M {y \in space M. (indicator A y::real) \neq 0} \neq ∞*

using *A* **by** auto

qed (rule *simple_function_indicator assms*)+

moreover have *simple_bochner_integral M (indicator A) = measure M A*

using *simple_bochner_integral_eq_nn_integral*[*OF sbi*] *A*

by (*simp add: enreal_indicator emeasure_eq_enreal_measure*)

ultimately show *?thesis*

by (*metis has_bochner_integral_simple_bochner_integrable*)

qed

lemma *has_bochner_integral_add*[*intro*]:

has_bochner_integral M f x \implies has_bochner_integral M g y \implies

has_bochner_integral M (λ x. f x + g x) (x + y)

proof (*safe intro!*: *has_bochner_integral.intros elim!*: *has_bochner_integral.cases*)

fix *sf sg*

assume *f_sf: (λ i. $\int^+ x. norm (f x - sf i x) \partial M$) \longrightarrow 0*

assume *g_sg: (λ i. $\int^+ x. norm (g x - sg i x) \partial M$) \longrightarrow 0*

assume *sf: \forall i. simple_bochner_integrable M (sf i)*

and *sg: \forall i. simple_bochner_integrable M (sg i)*

then have [*measurable*]: \bigwedge i. *sf i \in borel_measurable M \wedge sg i \in borel_measurable M*

by (auto intro: *borel_measurable_simple_function elim: simple_bochner_integrable.cases*)

assume [*measurable*]: *f \in borel_measurable M g \in borel_measurable M*

show \bigwedge i. *simple_bochner_integrable M (λ x. sf i x + sg i x)*

using *sf sg* **by** (*simp add: simple_bochner_integrable_compose2*)

show (λ i. $\int^+ x. (norm (f x + g x - (sf i x + sg i x))) \partial M$) \longrightarrow 0

(**is** *?f \longrightarrow 0*)

proof (rule *tendsto_sandwich*)

```

show eventually ( $\lambda n. 0 \leq ?f n$ ) sequentially ( $\lambda_. 0$ )  $\longrightarrow 0$ 
  by auto
show eventually ( $\lambda i. ?f i \leq (\int^+ x. (\text{norm } (f x - sf i x)) \partial M) + \int^+ x. (\text{norm } (g x - sg i x)) \partial M$ ) sequentially
  (is eventually ( $\lambda i. ?f i \leq ?g i$ ) sequentially)
proof (intro always_eventually allI)
  fix  $i$  have  $?f i \leq (\int^+ x. (\text{norm } (f x - sf i x)) + \text{ennreal } (\text{norm } (g x - sg i x)) \partial M)$ 
    by (auto intro!: nn_integral_mono norm_diff_triangle_ineq
      simp flip: ennreal_plus)
  also have  $\dots = ?g i$ 
    by (intro nn_integral_add) auto
  finally show  $?f i \leq ?g i$  .
qed
show  $?g \longrightarrow 0$ 
  using tendsto_add[OF f_sf g_sg] by simp
qed
qed (auto simp: simple_bochner_integral_add tendsto_add)

```

lemma has_bochner_integral_bounded_linear:

assumes bounded_linear T

shows has_bochner_integral $M f x \implies$ has_bochner_integral $M (\lambda x. T (f x)) (T x)$

proof (safe intro!: has_bochner_integral.intros elim!: has_bochner_integral.cases)

interpret T : bounded_linear T **by** fact

have [measurable]: $T \in \text{borel_measurable borel}$

by (intro borel_measurable_continuous_onI T.continuous_on continuous_on_id)

assume [measurable]: $f \in \text{borel_measurable } M$

then **show** $(\lambda x. T (f x)) \in \text{borel_measurable } M$

by auto

fix s **assume** f_s : ($\lambda i. \int^+ x. \text{norm } (f x - s i x) \partial M$) $\longrightarrow 0$

assume s : $\forall i. \text{simple_bochner_integrable } M (s i)$

then **show** $\bigwedge i. \text{simple_bochner_integrable } M (\lambda x. T (s i x))$

by (auto intro: simple_bochner_integrable_compose2 T.zero)

have [measurable]: $\bigwedge i. s i \in \text{borel_measurable } M$

using s **by** (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

obtain K **where** $K: K > 0 \bigwedge x i. \text{norm } (T (f x) - T (s i x)) \leq \text{norm } (f x - s i x) * K$

using $T.pos_bounded$ **by** (auto simp: T.diff[symmetric])

show ($\lambda i. \int^+ x. \text{norm } (T (f x) - T (s i x)) \partial M$) $\longrightarrow 0$

(**is** $?f \longrightarrow 0$)

proof (rule tendsto_sandwich)

show eventually ($\lambda n. 0 \leq ?f n$) sequentially ($\lambda_. 0$) $\longrightarrow 0$

by auto

```

show eventually ( $\lambda i. ?f i \leq K * (\int^+ x. \text{norm } (f x - s i x) \partial M)$ ) sequentially
  (is eventually ( $\lambda i. ?f i \leq ?g i$ ) sequentially)
proof (intro always_eventually allI)
  fix  $i$  have  $?f i \leq (\int^+ x. \text{ennreal } K * \text{norm } (f x - s i x) \partial M)$ 
  using  $K$  by (intro nn_integral_mono) (auto simp: ac_simps ennreal_mult[symmetric])
  also have  $\dots = ?g i$ 
  using  $K$  by (intro nn_integral_cmult) auto
  finally show  $?f i \leq ?g i$  .
qed
show  $?g \longrightarrow 0$ 
  using ennreal_tendsto_cmult[OF _ f_s] by simp
qed

assume ( $\lambda i. \text{simple\_bochner\_integral } M (s i) \longrightarrow x$ )
with  $s$  show ( $\lambda i. \text{simple\_bochner\_integral } M (\lambda x. T (s i x)) \longrightarrow T x$ )
by (auto intro!: T.tendsto simp: simple_bochner_integral_linear T.linear_axioms)
qed

lemma has_bochner_integral_zero[intro]: has_bochner_integral  $M (\lambda x. 0) 0$ 
by (auto intro!: has_bochner_integral.intros[where  $s = \lambda \_ . 0$ ]
  simp: zero_ennreal_def[symmetric] simple_bochner_integrable_simps
  simple_bochner_integral_def image_constant_conv)

lemma has_bochner_integral_scaleR_left[intro]:
  ( $c \neq 0 \implies \text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M (\lambda x. f x$ 
 $*_R c) (x *_R c)$ )
by (cases  $c = 0$ ) (auto simp: has_bochner_integral_bounded_linear[OF bounded_linear_scaleR_left])

lemma has_bochner_integral_scaleR_right[intro]:
  ( $c \neq 0 \implies \text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M (\lambda x. c *_R$ 
 $f x) (c *_R x)$ )
by (cases  $c = 0$ ) (auto simp: has_bochner_integral_bounded_linear[OF bounded_linear_scaleR_right])

lemma has_bochner_integral_mult_left[intro]:
  fixes  $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$ 
  shows ( $c \neq 0 \implies \text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M$ 
 $(\lambda x. f x * c) (x * c)$ )
by (cases  $c = 0$ ) (auto simp: has_bochner_integral_bounded_linear[OF bounded_linear_mult_left])

lemma has_bochner_integral_mult_right[intro]:
  fixes  $c :: \_ :: \{\text{real\_normed\_algebra}, \text{second\_countable\_topology}\}$ 
  shows ( $c \neq 0 \implies \text{has\_bochner\_integral } M f x \implies \text{has\_bochner\_integral } M$ 
 $(\lambda x. c * f x) (c * x)$ )
by (cases  $c = 0$ ) (auto simp: has_bochner_integral_bounded_linear[OF bounded_linear_mult_right])

lemmas has_bochner_integral_divide =
  has_bochner_integral_bounded_linear[OF bounded_linear_divide]

lemma has_bochner_integral_divide_zero[intro]:

```

fixes $c :: _ :: \{real_normed_field, field, second_countable_topology\}$
shows $(c \neq 0 \implies has_bochner_integral\ M\ f\ x) \implies has_bochner_integral\ M$
 $(\lambda x. f\ x / c)\ (x / c)$
using $has_bochner_integral_divide$ **by** $(cases\ c = 0)\ auto$

lemma $has_bochner_integral_inner_left$ [intro]:
 $(c \neq 0 \implies has_bochner_integral\ M\ f\ x) \implies has_bochner_integral\ M\ (\lambda x. f\ x \cdot$
 $c)\ (x \cdot c)$
by $(cases\ c = 0)\ (auto\ simp: has_bochner_integral_bounded_linear[OF\ bounded_linear_inner_left])$

lemma $has_bochner_integral_inner_right$ [intro]:
 $(c \neq 0 \implies has_bochner_integral\ M\ f\ x) \implies has_bochner_integral\ M\ (\lambda x. c \cdot f$
 $x)\ (c \cdot x)$
by $(cases\ c = 0)\ (auto\ simp: has_bochner_integral_bounded_linear[OF\ bounded_linear_inner_right])$

lemmas $has_bochner_integral_minus =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_minus[OF\ bounded_linear_ident]]$

lemmas $has_bochner_integral_Re =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_Re]$

lemmas $has_bochner_integral_Im =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_Im]$

lemmas $has_bochner_integral_cnj =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_cnj]$

lemmas $has_bochner_integral_of_real =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_of_real]$

lemmas $has_bochner_integral_fst =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_fst]$

lemmas $has_bochner_integral_snd =$
 $has_bochner_integral_bounded_linear[OF\ bounded_linear_snd]$

lemma $has_bochner_integral_indicator$:
 $A \in sets\ M \implies emeasure\ M\ A < \infty \implies$
 $has_bochner_integral\ M\ (\lambda x. indicator\ A\ x *_{\mathbb{R}}\ c)\ (measure\ M\ A *_{\mathbb{R}}\ c)$
by $(intro\ has_bochner_integral_scaleR_left\ has_bochner_integral_real_indicator)$

lemma $has_bochner_integral_diff$:
 $has_bochner_integral\ M\ f\ x \implies has_bochner_integral\ M\ g\ y \implies$
 $has_bochner_integral\ M\ (\lambda x. f\ x - g\ x)\ (x - y)$
unfolding $diff_conv_add_uminus$
by $(intro\ has_bochner_integral_add\ has_bochner_integral_minus)$

lemma $has_bochner_integral_sum$:
 $(\bigwedge i. i \in I \implies has_bochner_integral\ M\ (f\ i)\ (x\ i)) \implies$
 $has_bochner_integral\ M\ (\lambda x. \sum_{i \in I}. f\ i\ x)\ (\sum_{i \in I}. x\ i)$
by $(induct\ I\ rule: infinite_finite_induct)\ auto$

proposition $has_bochner_integral_implies_finite_norm$:
 $has_bochner_integral\ M\ f\ x \implies (\int^+ x. norm\ (f\ x)\ \partial M) < \infty$
proof $(elim\ has_bochner_integral.cases)$


```

fix s v
assume [measurable]: f ∈ borel_measurable M and s:  $\bigwedge i$ . simple_bochner_integrable
M (s i) and
  lim_0:  $(\lambda i$ .  $\int^+ x$ . ennreal (norm (f x - s i x))  $\partial M$ )  $\longrightarrow$  0
from order_tendstoD[OF lim_0, of  $\infty$ ]
obtain i where f_s_fin:  $(\int^+ x$ . ennreal (norm (f x - s i x))  $\partial M$ )  $<$   $\infty$ 
  by (auto simp: eventually_sequentially)

have [measurable]:  $\bigwedge i$ . s i ∈ borel_measurable M
using s by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

define m where m = (if space M = {} then 0 else Max (( $\lambda x$ . norm (s i x)) 'space
M))
have finite (s i ' space M)
  using s by (auto simp: simple_function_def simple_bochner_integrable.simps)
then have finite (norm ' s i ' space M)
  by (rule finite_imageI)
then have  $\bigwedge x$ . x ∈ space M  $\implies$  norm (s i x)  $\leq$  m 0  $\leq$  m
  by (auto simp: m_def image_comp comp_def Max_ge_iff)
then have  $(\int^+ x$ . norm (s i x)  $\partial M$ )  $\leq$   $(\int^+ x$ . ennreal m * indicator {x ∈ space
M. s i x  $\neq$  0} x  $\partial M$ )
  by (auto split: split_indicator intro!: Max_ge nn_integral_mono simp:)
also have ...  $<$   $\infty$ 
  using s by (subst nn_integral_cmult_indicator) (auto simp:  $\langle$ 0  $\leq$  m $\rangle$  simple_bochner_integrable.simps ennreal_mult_less_top less_top)
  finally have s_fin:  $(\int^+ x$ . norm (s i x)  $\partial M$ )  $<$   $\infty$  .

have  $(\int^+ x$ . norm (f x)  $\partial M$ )  $\leq$   $(\int^+ x$ . ennreal (norm (f x - s i x)) + ennreal
(norm (s i x))  $\partial M$ )
  by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
  (metis add.commute norm_triangle_sub)
also have ... =  $(\int^+ x$ . norm (f x - s i x)  $\partial M$ ) +  $(\int^+ x$ . norm (s i x)  $\partial M$ )
  by (rule nn_integral_add) auto
also have ...  $<$   $\infty$ 
  using s_fin f_s_fin by auto
finally show  $(\int^+ x$ . ennreal (norm (f x))  $\partial M$ )  $<$   $\infty$  .
qed

```

proposition has_bochner_integral_norm_bound:

assumes i: has_bochner_integral M f x

shows norm x \leq $(\int^+ x$. norm (f x) ∂M)

using assms **proof**

fix s **assume**

x: $(\lambda i$. simple_bochner_integral M (s i)) \longrightarrow x (**is** ?s \longrightarrow x) **and**

s[simp]: $\bigwedge i$. simple_bochner_integrable M (s i) **and**

lim: $(\lambda i$. $\int^+ x$. ennreal (norm (f x - s i x)) ∂M) \longrightarrow 0 **and**

f[measurable]: f ∈ borel_measurable M

have [measurable]: $\bigwedge i$. s i ∈ borel_measurable M

```

using s by (auto simp: simple_bochner_integrable.simps intro: borel_measurable_simple_function)

show norm x ≤ (∫+x. norm (f x) ∂M)
proof (rule LIMSEQ_le)
  show (λi. ennreal (norm (?s i))) ⟶ norm x
    using x by (auto simp: tendsto_ennreal_iff intro: tendsto_intros)
  show ∃ N. ∀ n ≥ N. norm (?s n) ≤ (∫+x. norm (f x - s n x) ∂M) + (∫+x.
norm (f x) ∂M)
    (is ∃ N. ∀ n ≥ N. _ ≤ ?t n)
  proof (intro exI allI impI)
    fix n
    have ennreal (norm (?s n)) ≤ simple_bochner_integral M (λx. norm (s n x))
      by (auto intro!: simple_bochner_integral_norm_bound)
    also have ... = (∫+x. norm (s n x) ∂M)
      by (intro simple_bochner_integral_eq_nn_integral)
        (auto intro: s simple_bochner_integrable_compose2)
    also have ... ≤ (∫+x. ennreal (norm (f x - s n x)) + norm (f x) ∂M)
      by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
        (metis add.commute norm_minus_commute norm_triangle_sub)
    also have ... = ?t n
      by (rule nn_integral_add) auto
    finally show norm (?s n) ≤ ?t n .
  qed
  have ?t ⟶ 0 + (∫+x. ennreal (norm (f x)) ∂M)
    using has_bochner_integral_implies_finite_norm[OF i]
    by (intro tendsto_add tendsto_const lim)
  then show ?t ⟶ ∫+x. ennreal (norm (f x)) ∂M
    by simp
  qed
qed

```

lemma *has_bochner_integral_eq*:

```

has_bochner_integral M f x ⟹ has_bochner_integral M f y ⟹ x = y
proof (elim has_bochner_integral.cases)
  assume f[measurable]: f ∈ borel_measurable M

  fix s t
  assume (λi. ∫+x. norm (f x - s i x) ∂M) ⟶ 0 (is ?S ⟶ 0)
  assume (λi. ∫+x. norm (f x - t i x) ∂M) ⟶ 0 (is ?T ⟶ 0)
  assume s: ∧i. simple_bochner_integrable M (s i)
  assume t: ∧i. simple_bochner_integrable M (t i)

  have [measurable]: ∧i. s i ∈ borel_measurable M ∧i. t i ∈ borel_measurable M
    using s t by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

  let ?s = λi. simple_bochner_integral M (s i)
  let ?t = λi. simple_bochner_integral M (t i)
  assume ?s ⟶ x ?t ⟶ y
  then have (λi. norm (?s i - ?t i)) ⟶ norm (x - y)

```

```

    by (intro tendsto_intros)
  moreover
  have ( $\lambda i. \text{ennreal} (\text{norm} (?s\ i - ?t\ i)) \longrightarrow \text{ennreal } 0$ )
  proof (rule tendsto_sandwich)
    show eventually ( $\lambda i. 0 \leq \text{ennreal} (\text{norm} (?s\ i - ?t\ i))$ ) sequentially ( $\lambda \_. 0$ )
   $\longrightarrow \text{ennreal } 0$ 
    by auto

  show eventually ( $\lambda i. \text{norm} (?s\ i - ?t\ i) \leq ?S\ i + ?T\ i$ ) sequentially
    by (intro always_eventually_all simple_bochner_integral_bounded s t f)
  show ( $\lambda i. ?S\ i + ?T\ i \longrightarrow \text{ennreal } 0$ )
    using tendsto_add[OF  $\langle ?S \longrightarrow 0 \rangle \langle ?T \longrightarrow 0 \rangle$ ] by simp
  qed
  then have ( $\lambda i. \text{norm} (?s\ i - ?t\ i) \longrightarrow 0$ )
    by (simp flip: ennreal_0)
  ultimately have  $\text{norm} (x - y) = 0$ 
    by (rule LIMSEQ_unique)
  then show  $x = y$  by simp
  qed

lemma has_bochner_integral_AE:
  assumes f: has_bochner_integral M f x
    and g:  $g \in \text{borel\_measurable } M$ 
    and ae:  $AE\ x\ \text{in } M. f\ x = g\ x$ 
  shows has_bochner_integral M g x
  using f
proof (safe intro!: has_bochner_integral.intros elim!: has_bochner_integral.cases)
  fix s assume ( $\lambda i. \int^+ x. \text{ennreal} (\text{norm} (f\ x - s\ i\ x)) \partial M \longrightarrow 0$ )
  also have ( $\lambda i. \int^+ x. \text{ennreal} (\text{norm} (f\ x - s\ i\ x)) \partial M = (\lambda i. \int^+ x. \text{ennreal} (\text{norm} (g\ x - s\ i\ x)) \partial M)$ )
    using ae
  by (intro ext nn_integral_cong_AE, eventually_elim) simp
  finally show ( $\lambda i. \int^+ x. \text{ennreal} (\text{norm} (g\ x - s\ i\ x)) \partial M \longrightarrow 0$ ) .
  qed (auto intro: g)

lemma has_bochner_integral_eq_AE:
  assumes has_bochner_integral M f x and has_bochner_integral M g y
    and  $AE\ x\ \text{in } M. f\ x = g\ x$ 
  shows  $x = y$ 
  by (meson assms has_bochner_integral.simps has_bochner_integral_cong_AE
  has_bochner_integral_eq)

lemma simple_bochner_integrable_restrict_space:
  fixes f ::  $\_ \Rightarrow 'b::\text{real\_normed\_vector}$ 
  assumes  $\Omega: \Omega \cap \text{space } M \in \text{sets } M$ 
  shows  $\text{simple\_bochner\_integrable} (\text{restrict\_space } M\ \Omega)\ f \longleftrightarrow$ 
     $\text{simple\_bochner\_integrable } M\ (\lambda x. \text{indicator } \Omega\ x *_{\mathbb{R}} f\ x)$ 
  by (simp add: simple_bochner_integrable.simps space_restrict_space
  simple_function_restrict_space[OF  $\Omega$ ] emeasure_restrict_space[OF  $\Omega$ ] Col-
```

2376

lect_restrict

indicator_eq_0_iff_conj_left_commute)

lemma *simple_bochner_integral_restrict_space*:

fixes $f :: _ \Rightarrow 'b::\text{real_normed_vector}$

assumes $\Omega: \Omega \cap \text{space } M \in \text{sets } M$

assumes $f: \text{simple_bochner_integrable } (\text{restrict_space } M \ \Omega) \ f$

shows $\text{simple_bochner_integral } (\text{restrict_space } M \ \Omega) \ f =$

$\text{simple_bochner_integral } M \ (\lambda x. \text{indicator } \Omega \ x *_{\mathbb{R}} f \ x)$

proof –

have *finite* $((\lambda x. \text{indicator } \Omega \ x *_{\mathbb{R}} f \ x) \text{'space } M)$

using $f \ \text{simple_bochner_integrable_restrict_space}[OF \ \Omega, \text{ of } f]$

by $(\text{simp add: simple_bochner_integrable.simps simple_function_def})$

then show *?thesis*

by $(\text{auto simp: space_restrict_space measure_restrict_space}[OF \ \Omega(1)] \ \text{le_infI2}$

$\text{simple_bochner_integral_def Collect_restrict}$

$\text{split: split_indicator split_indicator_asm}$

$\text{intro! sum.mono_neutral_cong_left arg_cong2}[\mathbf{where} \ f=\text{measure}]$)

qed

context

notes $[[\text{inductive_internals}]]$

begin

inductive integrable for $M \ f$ **where**

$\text{has_bochner_integral } M \ f \ x \Longrightarrow \text{integrable } M \ f$

end

definition *lebesgue_integral* (*integral^L*) **where**

$\text{integral}^L \ M \ f = (\text{if } \exists x. \text{has_bochner_integral } M \ f \ x \text{ then } \text{THE } x. \text{has_bochner_integral } M \ f \ x \text{ else } 0)$

syntax

$_ \text{lebesgue_integral} :: \text{pttrn} \Rightarrow \text{real} \Rightarrow 'a \ \text{measure} \Rightarrow \text{real} \ (\int ((2 \ _./ \ _)/ \ \partial _))$
[60,61] 110)

translations

$\int x. f \ \partial M == \text{CONST lebesgue_integral } M \ (\lambda x. f)$

syntax

$_ \text{asciï_lebesgue_integral} :: \text{pttrn} \Rightarrow 'a \ \text{measure} \Rightarrow \text{real} \Rightarrow \text{real} \ ((3\text{LINT } (1_)/|(_)/ \ _))$
[0,110,60] 60)

translations

$\text{LINT } x|M. f == \text{CONST lebesgue_integral } M \ (\lambda x. f)$

lemma *has_bochner_integral_integral_eq*: $\text{has_bochner_integral } M \ f \ x \Longrightarrow \text{integral}^L \ M \ f = x$

by (*metis the_equality has_bochner_integral_eq lebesgue_integral_def*)

lemma *has_bochner_integral_integrable*:

$integrable\ M\ f \implies has_bochner_integral\ M\ f\ (integral^L\ M\ f)$

by (*auto simp: has_bochner_integral_integral_eq integrable.simps*)

lemma *has_bochner_integral_iff*:

$has_bochner_integral\ M\ f\ x \longleftrightarrow integrable\ M\ f \wedge integral^L\ M\ f = x$

by (*metis has_bochner_integral_integrable has_bochner_integral_integral_eq integrable.intros*)

lemma *simple_bochner_integrable_eq_integral*:

$simple_bochner_integrable\ M\ f \implies simple_bochner_integral\ M\ f = integral^L\ M\ f$

using *has_bochner_integral_simple_bochner_integrable*[of $M\ f$]

by (*simp add: has_bochner_integral_integral_eq*)

lemma *not_integrable_integral_eq*: $\neg integrable\ M\ f \implies integral^L\ M\ f = 0$

unfolding *integrable.simps lebesgue_integral_def* **by** (*auto intro!: arg_cong[where f=The]*)

lemma *integral_eq_cases*:

$integrable\ M\ f \longleftrightarrow integrable\ N\ g \implies$

$(integrable\ M\ f \implies integrable\ N\ g \implies integral^L\ M\ f = integral^L\ N\ g) \implies$

$integral^L\ M\ f = integral^L\ N\ g$

by (*metis not_integrable_integral_eq*)

lemma *borel_measurable_integrable*[*measurable_dest*]: $integrable\ M\ f \implies f \in borel_measurable\ M$

by (*auto elim: integrable.cases has_bochner_integral.cases*)

lemma *borel_measurable_integrable*'[*measurable_dest*]:

$integrable\ M\ f \implies g \in measurable\ N\ M \implies (\lambda x. f\ (g\ x)) \in borel_measurable\ N$

using *borel_measurable_integrable*[*measurable*] **by** *measurable*

lemma *integrable_cong*:

$M = N \implies (\bigwedge x. x \in space\ N \implies f\ x = g\ x) \implies integrable\ M\ f \longleftrightarrow integrable\ N\ g$

by (*simp cong: has_bochner_integral_cong add: integrable.simps*)

lemma *integrable_cong_AE*:

$f \in borel_measurable\ M \implies g \in borel_measurable\ M \implies AE\ x\ in\ M. f\ x = g\ x \implies$

$integrable\ M\ f \longleftrightarrow integrable\ M\ g$

unfolding *integrable.simps*

by (*intro has_bochner_integral_cong_AE arg_cong[where f=Ex] ext*)

lemma *integrable_cong_AE_imp*:

$integrable\ M\ g \implies f \in borel_measurable\ M \implies (AE\ x\ in\ M. g\ x = f\ x) \implies$

integrable $M f$
using *integrable_cong_AE*[of $f M g$] **by** (*auto simp: eq_commute*)

lemma *integral_cong*:

$M = N \implies (\bigwedge x. x \in \text{space } N \implies f x = g x) \implies \text{integral}^L M f = \text{integral}^L N g$
by (*simp cong: has_bochner_integral_cong cong del: if_weak_cong add: lebesgue_integral_def*)

lemma *integral_cong_AE*:

$f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies \text{AE } x \text{ in } M. f x = g x \implies$

$\text{integral}^L M f = \text{integral}^L M g$

unfolding *lebesgue_integral_def*

by (*rule arg_cong[where x=has_bochner_integral M f]*) (*intro has_bochner_integral_cong_AE ext*)

lemma *integrable_add*[*simp, intro*]: *integrable* $M f \implies \text{integrable } M g \implies \text{integrable } M (\lambda x. f x + g x)$

by (*auto simp: integrable.simps*)

lemma *integrable_zero*[*simp, intro*]: *integrable* $M (\lambda x. 0)$

by (*metis has_bochner_integral_zero integrable.simps*)

lemma *integrable_sum*[*simp, intro*]: $(\bigwedge i. i \in I \implies \text{integrable } M (f i)) \implies \text{integrable } M (\lambda x. \sum_{i \in I} f i x)$

by (*metis has_bochner_integral_sum integrable.simps*)

lemma *integrable_indicator*[*simp, intro*]: $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies \text{integrable } M (\lambda x. \text{indicator } A x *_R c)$

by (*metis has_bochner_integral_indicator integrable.simps*)

lemma *integrable_real_indicator*[*simp, intro*]: $A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies$

$\text{integrable } M (\text{indicator } A :: 'a \Rightarrow \text{real})$

by (*metis has_bochner_integral_real_indicator integrable.simps*)

lemma *integrable_diff*[*simp, intro*]: *integrable* $M f \implies \text{integrable } M g \implies \text{integrable } M (\lambda x. f x - g x)$

by (*auto simp: integrable.simps intro: has_bochner_integral_diff*)

lemma *integrable_bounded_linear*: *bounded_linear* $T \implies \text{integrable } M f \implies \text{integrable } M (\lambda x. T (f x))$

by (*auto simp: integrable.simps intro: has_bochner_integral_bounded_linear*)

lemma *integrable_scaleR_left*[*simp, intro*]: $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x *_R c)$

unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_scaleR_right*[*simp, intro*]: $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. c *_R f x)$

unfolding *integrable.simps* by *fastforce*

lemma *integrable_mult_left*[*simp, intro*]:
fixes $c :: _ :: \{\text{real_normed_algebra}, \text{second_countable_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x * c)$
unfolding *integrable.simps* by *fastforce*

lemma *integrable_mult_right*[*simp, intro*]:
fixes $c :: _ :: \{\text{real_normed_algebra}, \text{second_countable_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. c * f x)$
unfolding *integrable.simps* by *fastforce*

lemma *integrable_divide_zero*[*simp, intro*]:
fixes $c :: _ :: \{\text{real_normed_field}, \text{field}, \text{second_countable_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x / c)$
unfolding *integrable.simps* by *fastforce*

lemma *integrable_inner_left*[*simp, intro*]:
 $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. f x \cdot c)$
unfolding *integrable.simps* by *fastforce*

lemma *integrable_inner_right*[*simp, intro*]:
 $(c \neq 0 \implies \text{integrable } M f) \implies \text{integrable } M (\lambda x. c \cdot f x)$
unfolding *integrable.simps* by *fastforce*

lemmas *integrable_minus*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_minus*[*OF* *bounded_linear_ident*]]

lemmas *integrable_divide*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_divide*]

lemmas *integrable_Re*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_Re*]

lemmas *integrable_Im*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_Im*]

lemmas *integrable_cnj*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_cnj*]

lemmas *integrable_of_real*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_of_real*]

lemmas *integrable_fst*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_fst*]

lemmas *integrable_snd*[*simp, intro*] =
integrable_bounded_linear[*OF* *bounded_linear_snd*]

lemma *integral_zero*[*simp*]: $\text{integral}^L M (\lambda x. 0) = 0$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_zero*)

lemma *integral_add*[*simp*]: $\text{integrable } M f \implies \text{integrable } M g \implies$
 $\text{integral}^L M (\lambda x. f x + g x) = \text{integral}^L M f + \text{integral}^L M g$
by (*intro* *has_bochner_integral_integral_eq* *has_bochner_integral_add* *has_bochner_integral_integrable*)

lemma *integral_diff[simp]*: $\text{integrable } M f \implies \text{integrable } M g \implies$
 $\text{integral}^L M (\lambda x. f x - g x) = \text{integral}^L M f - \text{integral}^L M g$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_diff has_bochner_integral_integrable*)

lemma *integral_sum*: $(\bigwedge i. i \in I \implies \text{integrable } M (f i)) \implies$
 $\text{integral}^L M (\lambda x. \sum i \in I. f i x) = (\sum i \in I. \text{integral}^L M (f i))$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_sum has_bochner_integral_integrable*)

lemma *integral_sum'[simp]*: $(\bigwedge i. i \in I = \text{simp} \implies \text{integrable } M (f i)) \implies$
 $\text{integral}^L M (\lambda x. \sum i \in I. f i x) = (\sum i \in I. \text{integral}^L M (f i))$
unfolding *simp_implies_def* **by** (*rule integral_sum*)

lemma *integral_bounded_linear*: $\text{bounded_linear } T \implies \text{integrable } M f \implies$
 $\text{integral}^L M (\lambda x. T (f x)) = T (\text{integral}^L M f)$
by (*metis has_bochner_integral_bounded_linear has_bochner_integral_integrable has_bochner_integral_integral_eq*)

lemma *integral_bounded_linear'*:
assumes *T*: *bounded_linear T* **and** *T'*: *bounded_linear T'*
assumes *: $\neg (\forall x. T x = 0) \implies (\forall x. T' (T x) = x)$
shows $\text{integral}^L M (\lambda x. T (f x)) = T (\text{integral}^L M f)$
proof cases
assume $(\forall x. T x = 0)$ **then show** *?thesis*
by *simp*
next
assume **: $\neg (\forall x. T x = 0)$
show *?thesis*
proof cases
assume *integrable M f with T show* *?thesis*
by (*rule integral_bounded_linear*)
next
assume *not*: $\neg \text{integrable } M f$
moreover have $\neg \text{integrable } M (\lambda x. T (f x))$
by (*metis (full_types) ** T' integrable_bounded_linear integrable_cong not*)
ultimately show *?thesis*
using *T* **by** (*simp add: not_integrable_integral_eq linear_simps*)
qed
qed

lemma *integral_scaleR_left[simp]*: $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x *_{\mathbb{R}} c \partial M) = \text{integral}^L M f *_{\mathbb{R}} c$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_scaleR*)

lemma *integral_scaleR_right[simp]*: $(\int x. c *_{\mathbb{R}} f x \partial M) = c *_{\mathbb{R}} \text{integral}^L M f$
by (*rule integral_bounded_linear'[OF bounded_linear_scaleR_right bounded_linear_scaleR_right[of 1 / c]]*) *simp*

lemma *integral_mult_left[simp]*:
fixes *c* :: $_ :: \{\text{real_normed_algebra}, \text{second_countable_topology}\}$

shows $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x * c \ \partial M) = \text{integral}^L M f * c$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult_left*)

lemma *integral_mult_right[simp]*:

fixes $c :: _ :: \{\text{real_normed_algebra}, \text{second_countable_topology}\}$
shows $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. c * f x \ \partial M) = c * \text{integral}^L M f$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult_right*)

lemma *integral_mult_left_zero[simp]*:

fixes $c :: _ :: \{\text{real_normed_field}, \text{second_countable_topology}\}$
shows $(\int x. f x * c \ \partial M) = \text{integral}^L M f * c$
by (*rule integral_bounded_linear'[OF bounded_linear_mult_left bounded_linear_mult_left[of 1 / c]]*) *simp*

lemma *integral_mult_right_zero[simp]*:

fixes $c :: _ :: \{\text{real_normed_field}, \text{second_countable_topology}\}$
shows $(\int x. c * f x \ \partial M) = c * \text{integral}^L M f$
by (*rule integral_bounded_linear'[OF bounded_linear_mult_right bounded_linear_mult_right[of 1 / c]]*) *simp*

lemma *integral_inner_left[simp]*: $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. f x \cdot c \ \partial M) = \text{integral}^L M f \cdot c$

by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_inner_left*)

lemma *integral_inner_right[simp]*: $(c \neq 0 \implies \text{integrable } M f) \implies (\int x. c \cdot f x \ \partial M) = c \cdot \text{integral}^L M f$

by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_inner_right*)

lemma *integral_divide_zero[simp]*:

fixes $c :: _ :: \{\text{real_normed_field}, \text{field}, \text{second_countable_topology}\}$
shows $\text{integral}^L M (\lambda x. f x / c) = \text{integral}^L M f / c$
by (*rule integral_bounded_linear'[OF bounded_linear_divide bounded_linear_mult_left[of c]]*) *simp*

lemma *integral_minus[simp]*: $\text{integral}^L M (\lambda x. - f x) = - \text{integral}^L M f$

by (*rule integral_bounded_linear'[OF bounded_linear_minus[OF bounded_linear_ident] bounded_linear_minus[OF bounded_linear_ident]]*) *simp*

lemma *integral_complex_of_real[simp]*: $\text{integral}^L M (\lambda x. \text{complex_of_real } (f x)) = \text{of_real } (\text{integral}^L M f)$

by (*rule integral_bounded_linear'[OF bounded_linear_of_real bounded_linear_Re]*) *simp*

lemma *integral_cnj[simp]*: $\text{integral}^L M (\lambda x. \text{cnj } (f x)) = \text{cnj } (\text{integral}^L M f)$

by (*rule integral_bounded_linear'[OF bounded_linear_cnj bounded_linear_cnj]*) *simp*

lemmas *integral_divide[simp]* =

integral_bounded_linear[OF bounded_linear_divide]

```

lemmas integral_Re[simp] =
  integral_bounded_linear[OF bounded_linear_Re]
lemmas integral_Im[simp] =
  integral_bounded_linear[OF bounded_linear_Im]
lemmas integral_of_real[simp] =
  integral_bounded_linear[OF bounded_linear_of_real]
lemmas integral_fst[simp] =
  integral_bounded_linear[OF bounded_linear_fst]
lemmas integral_snd[simp] =
  integral_bounded_linear[OF bounded_linear_snd]

```

```

lemma integral_norm_bound_ennreal:
  integrable M f  $\implies$  norm (integralL M f)  $\leq$  ( $\int$  +x. norm (f x)  $\partial$ M)
  by (metis has_bochner_integral_integrable has_bochner_integral_norm_bound)

```

```

lemma integrableI_sequence:
  fixes f :: 'a  $\implies$  'b::{banach, second_countable_topology}
  assumes f[measurable]: f  $\in$  borel_measurable M
  assumes s:  $\bigwedge$ i. simple_bochner_integrable M (s i)
  assumes lim: ( $\lambda$ i.  $\int$  +x. norm (f x - s i x)  $\partial$ M)  $\longrightarrow$  0 (is ?S  $\longrightarrow$  0)
  shows integrable M f

```

```

proof -
  let ?s =  $\lambda$ n. simple_bochner_integral M (s n)

```

```

have  $\exists$ x. ?s  $\longrightarrow$  x

```

```

  unfolding convergent_eq_Cauchy

```

```

proof (rule metric_CauchyI)

```

```

  fix e :: real assume 0 < e

```

```

  then have 0 < ennreal (e / 2) by auto

```

```

  from order_tendstoD(2)[OF lim this]

```

```

  obtain M where M:  $\bigwedge$ n. M  $\leq$  n  $\implies$  ?S n < e / 2

```

```

  by (auto simp: eventually_sequentially)

```

```

  show  $\exists$ M.  $\forall$ m  $\geq$ M.  $\forall$ n  $\geq$ M. dist (?s m) (?s n) < e

```

```

  proof (intro exI allI impI)

```

```

    fix m n assume m: M  $\leq$  m and n: M  $\leq$  n

```

```

    have ?S n  $\neq$   $\infty$ 

```

```

      using M[OF n] by auto

```

```

    have norm (?s n - ?s m)  $\leq$  ?S n + ?S m

```

```

      by (intro simple_bochner_integral_bounded s f)

```

```

    also have ... < ennreal (e / 2) + e / 2

```

```

      by (intro add_strict_mono M n m)

```

```

    also have ... = e using <0<e> by (simp flip: ennreal_plus)

```

```

    finally show dist (?s n) (?s m) < e

```

```

      using <0<e> by (simp add: dist_norm ennreal_less_iff)

```

```

  qed

```

```

qed

```

```

then obtain x where ?s  $\longrightarrow$  x ..

```

```

show ?thesis

```

```

  by (rule, rule) fact+

```

qed

proposition *nn_integral_dominated_convergence_norm*:

fixes $u' :: _ \Rightarrow _ :: \{\text{real_normed_vector, second_countable_topology}\}$

assumes [*measurable*]:

$\bigwedge i. u\ i \in \text{borel_measurable } M\ u' \in \text{borel_measurable } M\ w \in \text{borel_measurable } M$

and *bound*: $\bigwedge j. \text{AE } x \text{ in } M. \text{norm } (u\ j\ x) \leq w\ x$

and $w: (\int^+ x. w\ x\ \partial M) < \infty$

and $u': \text{AE } x \text{ in } M. (\lambda i. u\ i\ x) \longrightarrow u'\ x$

shows $(\lambda i. (\int^+ x. \text{norm } (u'\ x - u\ i\ x)\ \partial M)) \longrightarrow 0$

proof –

have $\text{AE } x \text{ in } M. \forall j. \text{norm } (u\ j\ x) \leq w\ x$

unfolding *AE_all_countable* **by** *rule fact*

with u' **have** *bnd*: $\text{AE } x \text{ in } M. \forall j. \text{norm } (u'\ x - u\ j\ x) \leq 2 * w\ x$

proof (*eventually_elim, intro allI*)

fix $i\ x$ **assume** $(\lambda i. u\ i\ x) \longrightarrow u'\ x \forall j. \text{norm } (u\ j\ x) \leq w\ x \forall j. \text{norm } (u\ j\ x) \leq w\ x$

then have $\text{norm } (u'\ x) \leq w\ x \text{norm } (u\ i\ x) \leq w\ x$

by (*auto intro: LIMSEQ_le_const2 tendsto_norm*)

then have $\text{norm } (u'\ x) + \text{norm } (u\ i\ x) \leq 2 * w\ x$

by *simp*

also have $\text{norm } (u'\ x - u\ i\ x) \leq \text{norm } (u'\ x) + \text{norm } (u\ i\ x)$

by (*rule norm_triangle_ineq4*)

finally (*xtrans*) **show** $\text{norm } (u'\ x - u\ i\ x) \leq 2 * w\ x .$

qed

have $w_nonneg: \text{AE } x \text{ in } M. 0 \leq w\ x$

using *bound*[*of 0*] **by** (*auto intro: order_trans[OF norm_ge_zero]*)

have $(\lambda i. (\int^+ x. \text{norm } (u'\ x - u\ i\ x)\ \partial M)) \longrightarrow (\int^+ x. 0\ \partial M)$

proof (*rule nn_integral_dominated_convergence*)

show $(\int^+ x. 2 * w\ x\ \partial M) < \infty$

by (*rule nn_integral_mult_bounded_inf[OF _ w, of 2]*) (*insert w_nonneg, auto simp: ennreal_mult*)

show $\text{AE } x \text{ in } M. (\lambda i. \text{ennreal } (\text{norm } (u'\ x - u\ i\ x))) \longrightarrow 0$

using u'

proof *eventually_elim*

fix x **assume** $(\lambda i. u\ i\ x) \longrightarrow u'\ x$

from *tendsto_diff[OF tendsto_const[of $u'\ x$] this]*

show $(\lambda i. \text{ennreal } (\text{norm } (u'\ x - u\ i\ x))) \longrightarrow 0$

by (*simp add: tendsto_norm_zero_iff_flip: ennreal_0*)

qed

qed (*use bnd w_nonneg in auto*)

then show *?thesis* **by** *simp*

qed

proposition *integrableI_bounded*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$

assumes f [*measurable*]: $f \in \text{borel_measurable } M$ **and** $\text{fin}: (\int^+ x. \text{norm } (f\ x)\ \partial M)$

2384

< ∞

shows *integrable* $M f$

proof –

from *borel_measurable_implies_sequence_metric*[$OF f, of 0$] obtain s where

$s: \bigwedge i. \text{simple_function } M (s i)$ and

pointwise: $\bigwedge x. x \in \text{space } M \implies (\lambda i. s i x) \longrightarrow f x$ and

bound: $\bigwedge i x. x \in \text{space } M \implies \text{norm } (s i x) \leq 2 * \text{norm } (f x)$

by *simp metis*

show *?thesis*

proof (*rule integrableI_sequence*)

{ fix i

have $(\int^+ x. \text{norm } (s i x) \partial M) \leq (\int^+ x. \text{ennreal } (2 * \text{norm } (f x)) \partial M)$

by (*intro nn_integral_mono*) (*simp add: bound*)

also have $\dots = 2 * (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M)$

by (*simp add: ennreal_mult nn_integral_cmult*)

also have $\dots < \text{top}$

using *fin* by (*simp add: ennreal_mult_less_top*)

finally have $(\int^+ x. \text{norm } (s i x) \partial M) < \infty$

by *simp* }

note $\text{fin}_s = \text{this}$

show $\bigwedge i. \text{simple_bochner_integrable } M (s i)$

by (*rule simple_bochner_integrableI_bounded*) *fact+*

show $(\lambda i. \int^+ x. \text{ennreal } (\text{norm } (f x - s i x)) \partial M) \longrightarrow 0$

proof (*rule nn_integral_dominated_convergence_norm*)

show $\bigwedge j. AE x \text{ in } M. \text{norm } (s j x) \leq 2 * \text{norm } (f x)$

using *bound* by *auto*

show $\bigwedge i. s i \in \text{borel_measurable } M (\lambda x. 2 * \text{norm } (f x)) \in \text{borel_measurable}$

M

using s by (*auto intro: borel_measurable_simple_function*)

show $(\int^+ x. \text{ennreal } (2 * \text{norm } (f x)) \partial M) < \infty$

using *fin* by (*simp add: nn_integral_cmult ennreal_mult ennreal_mult_less_top*)

show $AE x \text{ in } M. (\lambda i. s i x) \longrightarrow f x$

using *pointwise* by *auto*

qed *fact*

qed *fact*

qed

lemma *integrableI_bounded_set*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$

assumes [*measurable*]: $A \in \text{sets } M f \in \text{borel_measurable } M$

assumes *finite*: $\text{emeasure } M A < \infty$

and *bnd*: $AE x \text{ in } M. x \in A \longrightarrow \text{norm } (f x) \leq B$

and *null*: $AE x \text{ in } M. x \notin A \longrightarrow f x = 0$

shows *integrable* $M f$

proof (*rule integrableI_bounded*)

{ fix $x :: 'b$ have $\text{norm } x \leq B \implies 0 \leq B$

```

    using norm_ge_zero[of x] by arith }
  with bnd null have  $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial M) \leq (\int^+ x. \text{ennreal}(\max 0 B) * \text{indicator } A x \partial M)$ 
  by (intro nn_integral_mono_AE) (auto split: split_indicator split_max)
  also have ... <  $\infty$ 
  using finite by (subst nn_integral_cmult_indicator) (auto simp: ennreal_mult_less_top)
  finally show  $(\int^+ x. \text{ennreal}(\text{norm}(f x)) \partial M) < \infty$  .
qed simp

```

lemma *integrableI_bounded_set_indicator*:

```

  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  shows  $A \in \text{sets } M \implies f \in \text{borel\_measurable } M \implies$ 
     $\text{emeasure } M A < \infty \implies (AE x \text{ in } M. x \in A \longrightarrow \text{norm}(f x) \leq B) \implies$ 
     $\text{integrable } M (\lambda x. \text{indicator } A x *_R f x)$ 
  by (rule integrableI_bounded_set[where A=A]) auto

```

lemma *integrableI_nonneg*:

```

  fixes f :: 'a  $\Rightarrow$  real
  assumes  $f \in \text{borel\_measurable } M \wedge AE x \text{ in } M. 0 \leq f x \wedge (\int^+ x. f x \partial M) < \infty$ 
  shows  $\text{integrable } M f$ 

```

proof –

```

  have  $(\int^+ x. \text{norm}(f x) \partial M) = (\int^+ x. f x \partial M)$ 
  using assms by (intro nn_integral_cong_AE) auto
  then show ?thesis
  using assms by (intro integrableI_bounded) auto

```

qed

lemma *integrable_iff_bounded*:

```

  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  shows  $\text{integrable } M f \iff f \in \text{borel\_measurable } M \wedge (\int^+ x. \text{norm}(f x) \partial M) < \infty$ 
  using integrableI_bounded[of f M] has_bochner_integral_implies_finite_norm[of M f]
  unfolding integrable.simps has_bochner_integral.simps[abs_def] by auto

```

lemma (in *finite_measure*) *square_integrable_imp_integrable*:

```

  fixes f :: 'a  $\Rightarrow$  'b::{second_countable_topology, banach, real_normed_div_algebra}
  assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
  assumes  $\text{integrable } M (\lambda x. f x ^ 2)$ 
  shows  $\text{integrable } M f$ 

```

proof –

```

  have less_top:  $\text{emeasure } M (\text{space } M) < \text{top}$ 
  using finite_emeasure_space by (meson top.not_eq_extremum)
  have nn_integral_M  $(\lambda x. \text{norm}(f x) ^ 2) \leq$ 
     $\text{nn\_integral } M (\lambda x. \text{norm}(f x) ^ 2) * \text{emeasure } M (\text{space } M)$ 
  using Cauchy_Schwarz_nn_integral[of  $\lambda x. \text{norm}(f x) M \lambda_. 1$ ]
  by (simp add: ennreal_power)
  also have ... <  $\infty$ 
  using assms(2) less_top

```

by (*subst (asm) integrable_iff_bounded*) (*auto simp: norm_power ennreal_mult_less_top*)
finally have *nn_integral M (λx. norm (f x)) < ∞*
by (*simp add: power_less_top_ennreal*)
thus *?thesis*
by (*simp add: integrable_iff_bounded*)
qed

lemma *integrable_bound*:

fixes *f :: 'a ⇒ 'b::{banach, second_countable_topology}*
and *g :: 'a ⇒ 'c::{banach, second_countable_topology}*
shows *integrable M f ⇒ g ∈ borel_measurable M ⇒ (AE x in M. norm (g x)*
 \leq *norm (f x)) ⇒*
integrable M g
unfolding *integrable_iff_bounded*
by (*smt (verit) AE_cong ennreal_le_iff nn_integral_mono_AE norm_ge_zero*
order_less_subst2 linorder_not_le order_less_le_trans)

lemma *integrable_mult_indicator*:

fixes *f :: 'a ⇒ 'b::{banach, second_countable_topology}*
shows *A ∈ sets M ⇒ integrable M f ⇒ integrable M (λx. indicator A x *_R f*
x)
by (*rule integrable_bound[of M f]*) (*auto split: split_indicator*)

lemma *integrable_real_mult_indicator*:

fixes *f :: 'a ⇒ real*
shows *A ∈ sets M ⇒ integrable M f ⇒ integrable M (λx. f x * indicator A x)*
using *integrable_mult_indicator[of A M f]* **by** (*simp add: mult_ac*)

lemma *integrable_abs[*simp, intro*]*:

fixes *f :: 'a ⇒ real*
assumes [*measurable*]: *integrable M f* **shows** *integrable M (λx. |f x|)*
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm[*simp, intro*]*:

fixes *f :: 'a ⇒ 'b::{banach, second_countable_topology}*
assumes [*measurable*]: *integrable M f* **shows** *integrable M (λx. norm (f x))*
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm_cancel*:

fixes *f :: 'a ⇒ 'b::{banach, second_countable_topology}*
assumes [*measurable*]: *integrable M (λx. norm (f x))* *f ∈ borel_measurable M*
shows *integrable M f*
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm_iff*:

fixes *f :: 'a ⇒ 'b::{banach, second_countable_topology}*
shows *f ∈ borel_measurable M ⇒ integrable M (λx. norm (f x)) ↔ integrable*
M f

by (auto intro: integrable_norm_cancel)

lemma *integrable_abs_cancel*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes [measurable]: $\text{integrable } M (\lambda x. |f x|) f \in \text{borel_measurable } M$ shows $\text{integrable } M f$

using *assms* by (rule *integrable_bound*) auto

lemma *integrable_abs_iff*:

fixes $f :: 'a \Rightarrow \text{real}$

shows $f \in \text{borel_measurable } M \implies \text{integrable } M (\lambda x. |f x|) \longleftrightarrow \text{integrable } M f$

by (auto intro: *integrable_abs_cancel*)

lemma *integrable_max*[*simp*, *intro*]:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $fg[\text{measurable}]$: $\text{integrable } M f \text{ integrable } M g$

shows $\text{integrable } M (\lambda x. \max (f x) (g x))$

using *integrable_add*[*OF integrable_norm*[*OF fg(1)*] *integrable_norm*[*OF fg(2)*]]

by (rule *integrable_bound*) auto

lemma *integrable_min*[*simp*, *intro*]:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $fg[\text{measurable}]$: $\text{integrable } M f \text{ integrable } M g$

shows $\text{integrable } M (\lambda x. \min (f x) (g x))$

using *integrable_add*[*OF integrable_norm*[*OF fg(1)*] *integrable_norm*[*OF fg(2)*]]

by (rule *integrable_bound*) auto

lemma *integral_minus_iff*[*simp*]:

$\text{integrable } M (\lambda x. -f x) :: 'a :: \{\text{banach}, \text{second_countable_topology}\} \longleftrightarrow \text{integrable } M f$

unfolding *integrable_iff_bounded*

by (auto)

lemma *integrable_indicator_iff*:

$\text{integrable } M (\text{indicator } A :: _ \Rightarrow \text{real}) \longleftrightarrow A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M (A \cap \text{space } M) < \infty$

by (*simp add: integrable_iff_bounded borel_measurable_indicator_iff ennreal_indicator nn_integral_indicator'*)

cong: *conj_cong*)

lemma *integral_indicator*[*simp*]: $\text{integral}^L M (\text{indicator } A) = \text{measure } M (A \cap \text{space } M)$

proof *cases*

assume *: $A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M (A \cap \text{space } M) < \infty$

have $\text{integral}^L M (\text{indicator } A) = \text{integral}^L M (\text{indicator } (A \cap \text{space } M))$

by (*metis* (*no_types*, *lifting*) *Int_iff_indicator_simps integral_cong*)

also have ... = $\text{measure } M (A \cap \text{space } M)$

using * by (*intro has_bochner_integral_integral_eq has_bochner_integral_real_indicator*) auto

```

finally show ?thesis .
next
  assume *:  $\neg (A \cap \text{space } M \in \text{sets } M \wedge \text{emeasure } M (A \cap \text{space } M) < \infty)$ 
  have  $\text{integral}^L M (\text{indicator } A) = \text{integral}^L M (\text{indicator } (A \cap \text{space } M)) :: \_ \Rightarrow$ 
  real
    by (intro integral_cong) (auto split: split_indicator)
  also have  $\dots = 0$ 
  using * by (subst not_integrable_integral_eq) (auto simp: integrable_indicator_iff)
  also have  $\dots = \text{measure } M (A \cap \text{space } M)$ 
  using * by (auto simp: measure_def emeasure_notin_sets not_less top_unique)
  finally show ?thesis .
qed

```

```

lemma integrable_discrete_difference_aux:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $f$ : integrable  $M$   $f$  and  $X$ : countable  $X$ 
  assumes  $\text{null}$ :  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes  $\text{sets}$ :  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes  $\text{eq}$ :  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows integrable  $M$   $g$ 
  unfolding integrable_iff_bounded
proof
  have  $f_{\text{meas}}$ :  $f \in \text{borel\_measurable } M (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) < \infty$ 
    using  $f$  integrable_iff_bounded by auto
  then show  $g \in \text{borel\_measurable } M$ 
    using measurable_discrete_difference[where  $X=X$ ]
    by (smt (verit) UNIV_I X eq_sets space_borel)
  have  $AE$   $x$  in  $M$ .  $x \notin X$ 
    using AE_discrete_difference  $X$  null sets by blast
  with  $f_{\text{meas}}$  show  $(\int^+ x. \text{ennreal } (\text{norm } (g x)) \partial M) < \infty$ 
    by (metis (mono_tags, lifting) AE_I2 AE_mp eq_nn_integral_cong_AE)
qed

```

```

lemma integrable_discrete_difference:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $X$ : countable  $X$ 
  assumes  $\text{null}$ :  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes  $\text{sets}$ :  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes  $\text{eq}$ :  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows integrable  $M$   $f \longleftrightarrow$  integrable  $M$   $g$ 
  by (metis X eq_integrable_discrete_difference_aux null sets)

```

```

lemma integral_discrete_difference:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
  assumes  $X$ : countable  $X$ 
  assumes  $\text{null}$ :  $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$ 
  assumes  $\text{sets}$ :  $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$ 
  assumes  $\text{eq}$ :  $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$ 
  shows  $\text{integral}^L M f = \text{integral}^L M g$ 

```


proof (rule integral_eq_cases)
show eq: integrable M f \longleftrightarrow integrable M g
 by (rule integrable_discrete_difference[where X=X]) fact+

assume f: integrable M f
show integral^L M f = integral^L M g
proof (rule integral_cong_AE)
show f \in borel_measurable M g \in borel_measurable M
 using f eq **by** (auto intro: borel_measurable_integrable)
have AE x in M. x \notin X
 by (rule AE_discrete_difference) fact+
with AE_space **show** AE x in M. f x = g x
 by eventually_elim fact

qed

qed

lemma has_bochner_integral_discrete_difference:
fixes f :: 'a \Rightarrow 'b::{banach, second_countable_topology}
assumes countable X
assumes $\bigwedge x. x \in X \implies \text{emeasure } M \{x\} = 0$
assumes $\bigwedge x. x \in X \implies \{x\} \in \text{sets } M$
assumes $\bigwedge x. x \in \text{space } M \implies x \notin X \implies f x = g x$
shows has_bochner_integral M f x \longleftrightarrow has_bochner_integral M g x
 by (metis assms has_bochner_integral_iff_integrable_discrete_difference integral_discrete_difference)

lemma

fixes f :: 'a \Rightarrow 'b::{banach, second_countable_topology} **and** w :: 'a \Rightarrow real
assumes f \in borel_measurable M $\bigwedge i. s i \in$ borel_measurable M integrable M w
assumes lim: AE x in M. $(\lambda i. s i x) \longrightarrow f x$
assumes bound: $\bigwedge i. AE x in M. \text{norm } (s i x) \leq w x$
shows integrable_dominated_convergence: integrable M f
 and integrable_dominated_convergence2: $\bigwedge i. \text{integrable } M (s i)$
and integral_dominated_convergence: $(\lambda i. \text{integral}^L M (s i)) \longrightarrow \text{integral}^L M f$

proof –

have w_nonneg: AE x in M. $0 \leq w x$
 using bound[of 0] **by** eventually_elim (auto intro: norm_ge_zero order_trans)
then **have** $(\int^{+x}. w x \partial M) = (\int^{+x}. \text{norm } (w x) \partial M)$
 by (intro nn_integral_cong_AE) auto
with $\langle \text{integrable } M w \rangle$ **have** w: $w \in \text{borel_measurable } M (\int^{+x}. w x \partial M) < \infty$
unfolding integrable_iff_bounded **by** auto

show int_s: $\bigwedge i. \text{integrable } M (s i)$
unfolding integrable_iff_bounded

proof

fix i

have $(\int^{+x}. \text{ennreal } (\text{norm } (s i x)) \partial M) \leq (\int^{+x}. w x \partial M)$
 using bound[of i] w_nonneg **by** (intro nn_integral_mono_AE) auto

with w **show** $(\int^+ x. \text{ennreal } (\text{norm } (s \ i \ x)) \ \partial M) < \infty$ **by** *auto*
qed *fact*

have all_bound : $AE \ x \ \text{in } M. \forall i. \text{norm } (s \ i \ x) \leq w \ x$
using $\text{bound unfolding AE_all_countable}$ **by** *auto*

show int_f : *integrable* $M \ f$
unfolding $\text{integrable_iff_bounded}$

proof

have $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M) \leq (\int^+ x. w \ x \ \partial M)$

using $\text{all_bound lim } w \ \text{nonneg}$

proof (*intro nn_integral_mono_AE, eventually_elim*)

fix x **assume** $\forall i. \text{norm } (s \ i \ x) \leq w \ x \ (\lambda i. s \ i \ x) \longrightarrow f \ x \ 0 \leq w \ x$

then **show** $\text{ennreal } (\text{norm } (f \ x)) \leq \text{ennreal } (w \ x)$

by (*metis LIMSEQ_le_const2 ennreal_leI tendsto_norm*)

qed

with w **show** $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M) < \infty$ **by** *auto*

qed *fact*

have $(\lambda n. \text{ennreal } (\text{norm } (\text{integral}^L \ M \ (s \ n) - \text{integral}^L \ M \ f))) \longrightarrow \text{ennreal } 0$
(is $?d \longrightarrow \text{ennreal } 0)$

proof (*rule tendsto_sandwich*)

show *eventually* $(\lambda n. \text{ennreal } 0 \leq ?d \ n) \ \text{sequentially } (\lambda _. \text{ennreal } 0) \longrightarrow$
 $\text{ennreal } 0$ **by** *auto*

show *eventually* $(\lambda n. ?d \ n \leq (\int^+ x. \text{norm } (s \ n \ x - f \ x) \ \partial M)) \ \text{sequentially}$

proof (*intro always_eventually allI*)

fix n

have $?d \ n = \text{norm } (\text{integral}^L \ M \ (\lambda x. s \ n \ x - f \ x))$

using int_f int_s **by** *simp*

also **have** $\dots \leq (\int^+ x. \text{norm } (s \ n \ x - f \ x) \ \partial M)$

by (*intro int_f int_s integrable_diff integral_norm_bound_ennreal*)

finally **show** $?d \ n \leq (\int^+ x. \text{norm } (s \ n \ x - f \ x) \ \partial M)$.

qed

show $(\lambda n. \int^+ x. \text{norm } (s \ n \ x - f \ x) \ \partial M) \longrightarrow \text{ennreal } 0$

unfolding ennreal_0

apply (*subst norm_minus_commute*)

proof (*rule nn_integral_dominated_convergence_norm[where w=w]*)

show $\bigwedge n. s \ n \in \text{borel_measurable } M$

using $\text{int_s unfolding integrable_iff_bounded}$ **by** *auto*

qed *fact+*

qed

then **have** $(\lambda n. \text{integral}^L \ M \ (s \ n) - \text{integral}^L \ M \ f) \longrightarrow 0$

by (*simp add: tendsto_norm_zero_iff del: ennreal_0*)

from $\text{tendsto_add}[OF \ \text{this tendsto_const}[of \ \text{integral}^L \ M \ f]]$

show $(\lambda i. \text{integral}^L \ M \ (s \ i)) \longrightarrow \text{integral}^L \ M \ f$ **by** *simp*

qed

context

fixes $s :: \text{real} \Rightarrow 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$ **and** $w :: 'a \Rightarrow$

```

real
  and f :: 'a ⇒ 'b and M
  assumes f ∈ borel_measurable M ∧ t. s t ∈ borel_measurable M integrable M w
  assumes lim: AE x in M. ((λi. s i x) ⟶ f x) at_top
  assumes bound: ∀ F i in at_top. AE x in M. norm (s i x) ≤ w x
begin

lemma integral_dominated_convergence_at_top: ((λt. integralL M (s t)) ⟶
integralL M f) at_top
proof (rule tendsto_at_topI_sequentially)
  fix X :: nat ⇒ real assume X: filterlim X at_top sequentially
  from filterlim_iff[THEN iffD1, OF this, rule_format, OF bound]
  obtain N where w: ∧n. N ≤ n ⟹ AE x in M. norm (s (X n) x) ≤ w x
  by (auto simp: eventually_sequentially)

  show (λn. integralL M (s (X n))) ⟶ integralL M f
  proof (rule LIMSEQ_offset, rule integral_dominated_convergence)
    show AE x in M. norm (s (X (n + N)) x) ≤ w x for n
    by (rule w) auto
    show AE x in M. (λn. s (X (n + N)) x) ⟶ f x
    using lim
  proof eventually_elim
    fix x assume ((λi. s i x) ⟶ f x) at_top
    then show (λn. s (X (n + N)) x) ⟶ f x
    by (intro LIMSEQ_ignore_initial_segment filterlim_compose[OF _ X])
  qed
  qed fact+
qed

lemma integrable_dominated_convergence_at_top: integrable M f
proof -
  from bound obtain N where w: ∧n. N ≤ n ⟹ AE x in M. norm (s n x) ≤
w x
  by (auto simp: eventually_at_top_linorder)
  show ?thesis
  proof (rule integrable_dominated_convergence)
    show AE x in M. norm (s (N + i) x) ≤ w x for i :: nat
    by (intro w) auto
    show AE x in M. (λi. s (N + real i) x) ⟶ f x
    using lim
  proof eventually_elim
    fix x assume ((λi. s i x) ⟶ f x) at_top
    then show (λn. s (N + n) x) ⟶ f x
    by (rule filterlim_compose)
    (auto intro!: filterlim_tendsto_add_at_top filterlim_real_sequentially)
  qed
  qed fact+
qed

```

end

lemma *integrable_mult_left_iff* [simp]:
fixes $f :: 'a \Rightarrow \text{real}$
shows $\text{integrable } M (\lambda x. c * f x) \longleftrightarrow c = 0 \vee \text{integrable } M f$
using *integrable_mult_left*[of $c M f$] *integrable_mult_left*[of $1 / c M \lambda x. c * f x$]
by (cases $c = 0$) auto

lemma *integrable_mult_right_iff* [simp]:
fixes $f :: 'a \Rightarrow \text{real}$
shows $\text{integrable } M (\lambda x. f x * c) \longleftrightarrow c = 0 \vee \text{integrable } M f$
using *integrable_mult_left_iff* [of $M c f$] **by** (simp add: mult.commute)

lemma *integrableI_nn_integral_finite*:
assumes [*measurable*]: $f \in \text{borel_measurable } M$
and *nonneg*: $AE x \text{ in } M. 0 \leq f x$
and *finite*: $(\int^+ x. f x \partial M) = \text{ennreal } x$
shows $\text{integrable } M f$
proof (rule *integrableI_bounded*)
have $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) = (\int^+ x. \text{ennreal } (f x) \partial M)$
using *nonneg* **by** (intro *nn_integral_cong_AE*) auto
with *finite* **show** $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) < \infty$
by auto
qed *simp*

lemma *integral_nonneg_AE*:
fixes $f :: 'a \Rightarrow \text{real}$
assumes *nonneg*: $AE x \text{ in } M. 0 \leq f x$
shows $0 \leq \text{integral}^L M f$
proof *cases*
assume $f: \text{integrable } M f$
then **have** [*measurable*]: $f \in M \rightarrow_M \text{borel}$
by auto
have $(\lambda x. \text{max } 0 (f x)) \in M \rightarrow_M \text{borel} \wedge x. 0 \leq \text{max } 0 (f x) \text{ integrable } M (\lambda x. \text{max } 0 (f x))$
using f **by** auto
from *this* **have** $0 \leq \text{integral}^L M (\lambda x. \text{max } 0 (f x))$
proof (*induction rule: borel_measurable_induct_real*)
case (add $f g$)
then **have** $\text{integrable } M f \text{ integrable } M g$
by (auto intro!: *integrable_bound*[OF *add.prem*s])
with *add* **show** ?*case*
by (simp add: *nn_integral_add*)
next
case (seq U)
show ?*case*
proof (rule *LIMSEQ_le_const*)
have $U_le: x \in \text{space } M \implies U i x \leq \text{max } 0 (f x) \text{ for } x i$

```

    using seq by (intro incseq_le) (auto simp: incseq_def le_fun_def)
  with seq nonneg show  $(\lambda i. \text{integral}^L M (U i)) \longrightarrow \text{LINT } x | M. \max 0 (f$ 
x)
    by (intro integral_dominated_convergence)
      (simp_all, fastforce)
  have integrable M (U i) for i
    using seq.premis by (rule integrable_bound) (insert U_le seq, auto)
  with seq show  $\exists N. \forall n \geq N. 0 \leq \text{integral}^L M (U n)$ 
    by auto
  qed
  qed (auto)
  also have  $\dots = \text{integral}^L M f$ 
    using nonneg by (auto intro!: integral_cong_AE)
  finally show ?thesis .
  qed (simp add: not_integrable_integral_eq)

lemma integral_nonneg[simp]:
  fixes f :: 'a  $\Rightarrow$  real
  shows  $(\bigwedge x. x \in \text{space } M \implies 0 \leq f x) \implies 0 \leq \text{integral}^L M f$ 
  by (intro integral_nonneg_AE) auto

proposition nn_integral_eq_integral:
  assumes f: integrable M f
  assumes nonneg: AE x in M.  $0 \leq f x$ 
  shows  $(\int^+ x. f x \partial M) = \text{integral}^L M f$ 
proof -
  { fix f :: 'a  $\Rightarrow$  real assume f:  $f \in \text{borel\_measurable } M \bigwedge x. 0 \leq f x \text{ integrable } M f$ 
  then have  $(\int^+ x. f x \partial M) = \text{integral}^L M f$ 
  proof (induct rule: borel_measurable_induct_real)
    case (set A) then show ?case
      by (simp add: integrable_indicator_iff ennreal_indicator emeasure_eq ennreal_measure)
  next
    case (mult f c) then show ?case
      by (auto simp: nn_integral_cmult ennreal_mult integral_nonneg_AE)
  next
    case (add g f)
    then have integrable M f integrable M g
      by (auto intro!: integrable_bound[OF add.premis])
    with add show ?case
      by (simp add: nn_integral_add integral_nonneg_AE)
  next
    case (seq U)
    show ?case
      proof (rule LIMSEQ_unique)
        have U_le_f:  $x \in \text{space } M \implies U i x \leq f x$  for x i
          using seq by (intro incseq_le) (auto simp: incseq_def le_fun_def)
        have int_U:  $\bigwedge i. \text{integrable } M (U i)$ 
          using seq f U_le_f by (intro integrable_bound[OF f(3)]) auto
      
```

```

from  $U\_le\_f$  seq have  $(\lambda i. \text{integral}^L M (U i)) \longrightarrow \text{integral}^L M f$ 
  by (intro integral_dominated_convergence) auto
then show  $(\lambda i. \text{ennreal} (\text{integral}^L M (U i))) \longrightarrow \text{ennreal} (\text{integral}^L M f)$ 
using seq_f_int_U by (simp add: f_integral_nonneg_AE)
have  $(\lambda i. \int^+ x. U i x \partial M) \longrightarrow \int^+ x. f x \partial M$ 
using seq_U_le_ff
by (intro nn_integral_dominated_convergence[where  $w=f$ ]) (auto simp: integrable_iff_bounded)
then show  $(\lambda i. \int x. U i x \partial M) \longrightarrow \int^+ x. f x \partial M$ 
using seq_int_U by simp
qed
qed }
from this[of  $\lambda x. \max 0 (f x)$ ] assms have  $(\int^+ x. \max 0 (f x) \partial M) = \text{integral}^L M (\lambda x. \max 0 (f x))$ 
by simp
also have  $\dots = \text{integral}^L M f$ 
using assms by (auto intro!: integral_cong_AE simp: integral_nonneg_AE)
also have  $(\int^+ x. \max 0 (f x) \partial M) = (\int^+ x. f x \partial M)$ 
using assms by (auto intro!: nn_integral_cong_AE simp: max_def)
finally show ?thesis .
qed

```

lemma *nn_integral_eq_integrable*:

```

assumes  $f \in M \rightarrow_M \text{borel AE } x \text{ in } M. 0 \leq f x$  and  $0 \leq x$ 
shows  $(\int^+ x. f x \partial M) = \text{ennreal } x \iff (\text{integrable } M f \wedge \text{integral}^L M f = x)$ 
by (metis assms ennreal_inj_integrableI_nn_integral_finite_integral_nonneg_AE nn_integral_eq_integral)

```

lemma

```

fixes  $f :: \_ \Rightarrow \_ \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
assumes integrable[measurable]:  $\bigwedge i. \text{integrable } M (f i)$ 
and summable:  $\text{AE } x \text{ in } M. \text{summable } (\lambda i. \text{norm } (f i x))$ 
and sums:  $\text{summable } (\lambda i. (\int x. \text{norm } (f i x) \partial M))$ 
shows integrable_suminf:  $\text{integrable } M (\lambda x. (\sum i. f i x))$  (is integrable M ?S)
and sums_integral:  $(\lambda i. \text{integral}^L M (f i)) \text{sums } (\int x. (\sum i. f i x) \partial M)$  (is ?f sums ?x)
and integral_suminf:  $(\int x. (\sum i. f i x) \partial M) = (\sum i. \text{integral}^L M (f i))$ 
and summable_integral:  $\text{summable } (\lambda i. \text{integral}^L M (f i))$ 

```

proof –

```

have 1:  $\text{integrable } M (\lambda x. \sum i. \text{norm } (f i x))$ 
proof (rule integrableI_bounded)
have  $\bigwedge x. \text{summable } (\lambda i. \text{norm } (f i x)) \implies$ 
   $\text{ennreal} (\text{norm } (\sum i. \text{norm } (f i x))) = (\sum i. \text{ennreal} (\text{norm } (f i x)))$ 
by (simp add: suminf_nonneg_ennreal_suminf_neq_top)
then have  $(\int^+ x. \text{ennreal} (\text{norm } (\sum i. \text{norm } (f i x))) \partial M) = (\int^+ x. (\sum i. \text{ennreal} (\text{norm } (f i x)))) \partial M$ 
using local.summable by (intro nn_integral_cong_AE) auto
also have  $\dots = (\sum i. \int^+ x. \text{norm } (f i x) \partial M)$ 

```

```

    by (intro nn_integral_suminf) auto
  also have ... = ( $\sum i.$  ennreal ( $\int x.$  norm ( $f i x$ )  $\partial M$ ))
    by (intro arg_cong[where f=suminf] ext nn_integral_eq_integral_integrable_norm integrable) auto
  finally show ( $\int^+ x.$  ennreal (norm ( $\sum i.$  norm ( $f i x$ )))  $\partial M$ )  $< \infty$ 
    by (simp add: sums_ennreal_suminf_neq_top_less_top[symmetric] integral_nonneg_AE)
  qed simp

have 2: AE x in M. ( $\lambda n.$   $\sum i < n.$   $f i x$ )  $\longrightarrow$  ( $\sum i.$   $f i x$ )
  using summable by eventually_elim (auto intro: summable_LIMSEQ summable_norm_cancel)

have 3:  $\bigwedge j.$  AE x in M. norm ( $\sum i < j.$   $f i x$ )  $\leq$  ( $\sum i.$  norm ( $f i x$ ))
  using summable
proof eventually_elim
  fix j x assume [simp]: summable ( $\lambda i.$  norm ( $f i x$ ))
  have norm ( $\sum i < j.$   $f i x$ )  $\leq$  ( $\sum i < j.$  norm ( $f i x$ )) by (rule norm_sum)
  also have ...  $\leq$  ( $\sum i.$  norm ( $f i x$ ))
    using sum_le_suminf[of  $\lambda i.$  norm ( $f i x$ )] unfolding sums_iff by auto
  finally show norm ( $\sum i < j.$   $f i x$ )  $\leq$  ( $\sum i.$  norm ( $f i x$ )) by simp
qed

note ibl = integrable_dominated_convergence[OF _ _ 1 2 3]
note int = integral_dominated_convergence[OF _ _ 1 2 3]

show integrable M ?S
  by (rule ibl) measurable

show ?f sums ?x unfolding sums_def
  using int by (simp add: integrable)
then show ?x = suminf ?f summable ?f
  unfolding sums_iff by auto
qed

proposition integral_norm_bound [simp]:
  fixes f ::  $\_ \Rightarrow 'a$  :: {banach, second_countable_topology}
  shows norm (integralL M f)  $\leq$  ( $\int x.$  norm (f x)  $\partial M$ )
proof (cases integrable M f)
  case True then show ?thesis
    using nn_integral_eq_integral[of M  $\lambda x.$  norm (f x)] integral_norm_bound_ennreal[of M f]
    by (simp add: integral_nonneg_AE)
  next
  case False
  then have norm (integralL M f) = 0 by (simp add: not_integrable_integral_eq)
  moreover have ( $\int x.$  norm (f x)  $\partial M$ )  $\geq$  0 by auto
  ultimately show ?thesis by simp
qed

```

proposition *integral_abs_bound* [simp]:

fixes $f :: 'a \Rightarrow \text{real}$ **shows** $\text{abs} (\int x. f x \partial M) \leq (\int x. |f x| \partial M)$
using *integral_norm_bound*[of $M f$] **by** *auto*

lemma *integral_eq_nn_integral*:

assumes $f \in \text{borel_measurable } M$
assumes $AE\ x\ \text{in } M. 0 \leq f\ x$
shows $\text{integral}^L M f = \text{enn2real} (\int^+ x. \text{ennreal} (f\ x) \partial M)$
by (*metis* *assms* *enn2real_ennreal_enn2real_top_infinity_ennreal_def* *integrableI_nonneg* *integral_nonneg_AE* *less_top_nn_integral_eq_integral* *not_integrable_integral_eq*)

lemma *enn2real_nn_integral_eq_integral*:

assumes $AE\ x\ \text{in } M. f\ x = \text{ennreal} (g\ x)$ **and** $nn: AE\ x\ \text{in } M. 0 \leq g\ x$
and $g \in M \rightarrow_M \text{borel}$
shows $\text{enn2real} (\int^+ x. f\ x \partial M) = (\int x. g\ x \partial M)$
by (*metis* *assms* *integral_eq_nn_integral* *nn_nn_integral_cong_AE*)

lemma *has_bochner_integral_nn_integral*:

assumes $f \in \text{borel_measurable } M$ $AE\ x\ \text{in } M. 0 \leq f\ x \leq x$
assumes $(\int^+ x. f\ x \partial M) = \text{ennreal } x$
shows *has_bochner_integral* $M f\ x$
by (*metis* *assms* *has_bochner_integral_iff_nn_integral_eq_integrable*)

lemma *integrableI_simple_bochner_integrable*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second_countable_topology}\}$
shows *simple_bochner_integrable* $M f \implies \text{integrable } M f$
using *has_bochner_integral_simple_bochner_integrable* *integrable.intros* **by** *blast*

proposition *integrable_induct*[*consumes 1*, *case_names* *base* *add* *lim*, *induct* *pred*: *integrable*]:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second_countable_topology}\}$
assumes *integrable* $M f$
assumes *base*: $\bigwedge A\ c. A \in \text{sets } M \implies \text{emeasure } M A < \infty \implies P (\lambda x. \text{indicator } A\ x *_{\mathbb{R}} c)$
assumes *add*: $\bigwedge f\ g. \text{integrable } M f \implies P f \implies \text{integrable } M g \implies P g \implies P (\lambda x. f\ x + g\ x)$
assumes *lim*: $\bigwedge f\ s. (\bigwedge i. \text{integrable } M (s\ i)) \implies (\bigwedge i. P (s\ i)) \implies (\bigwedge x. x \in \text{space } M \implies (\lambda i. s\ i\ x) \longrightarrow f\ x) \implies (\bigwedge i\ x. x \in \text{space } M \implies \text{norm} (s\ i\ x) \leq 2 * \text{norm} (f\ x)) \implies \text{integrable } M f \implies P f$
shows $P f$

proof –

from $\langle \text{integrable } M f \rangle$ **have** $f: f \in \text{borel_measurable } M (\int^+ x. \text{norm} (f\ x) \partial M) < \infty$

unfolding *integrable_iff_bounded* **by** *auto*

from *borel_measurable_implies_sequence_metric*[*OF* $f(1)$]

obtain s **where** $s: \bigwedge i. \text{simple_function } M (s\ i) \bigwedge x. x \in \text{space } M \implies (\lambda i. s\ i\ x) \longrightarrow f\ x$


```

 $\bigwedge i x. x \in \text{space } M \implies \text{norm } (s \ i \ x) \leq 2 * \text{norm } (f \ x)$ 
unfolding norm_conv_dist by metis

{ fix f A
  have [simp]:  $P (\lambda x. 0)$ 
    using base[of {} undefined] by simp
  have ( $\bigwedge i::'b. i \in A \implies \text{integrable } M (f \ i::'a \Rightarrow 'b)$ )  $\implies$ 
    ( $\bigwedge i. i \in A \implies P (f \ i)$ )  $\implies P (\lambda x. \sum i \in A. f \ i \ x)$ 
  by (induct A rule: infinite_finite_induct) (auto intro!: add) }
note sum = this

define s' where [abs_def]:  $s' \ i \ z = \text{indicator } (\text{space } M) \ z *_{\mathbb{R}} s \ i \ z$  for i z
then have s'_eq_s:  $\bigwedge i x. x \in \text{space } M \implies s' \ i \ x = s \ i \ x$ 
  by simp

have sf[measurable]:  $\bigwedge i. \text{simple\_function } M (s' \ i)$ 
  unfolding s'_def using s(1)
  by (intro simple_function_compose2[where h=(*_R)] simple_function_indicator)
auto

{ fix i
  have  $\bigwedge z. \{y. s' \ i \ z = y \wedge y \in s' \ i \ \text{space } M \wedge y \neq 0 \wedge z \in \text{space } M\} =$ 
    (if  $z \in \text{space } M \wedge s' \ i \ z \neq 0$  then  $\{s' \ i \ z\}$  else  $\{\}$ )
  by (auto simp: s'_def split: split_indicator)
  then have  $\bigwedge z. s' \ i = (\lambda z. \sum y \in s' \ i \ \text{space } M - \{0\}. \text{indicator } \{x \in \text{space } M. s' \ i \ x = y\} z *_{\mathbb{R}} y)$ 
  using sf by (auto simp: fun_eq_iff simple_function_def s'_def) }
note s'_eq = this

show  $P \ f$ 
proof (rule lim)
  fix i

  have  $(\int^+ x. \text{norm } (s' \ i \ x) \ \partial M) \leq (\int^+ x. \text{ennreal } (2 * \text{norm } (f \ x)) \ \partial M)$ 
  using s by (intro nn_integral_mono) (auto simp: s'_eq_s)
  also have  $\dots < \infty$ 
  using f by (simp add: nn_integral_cmult ennreal_mult_less_top ennreal_mult)
  finally have sbi: simple_bochner_integrable  $M (s' \ i)$ 
  using sf by (intro simple_bochner_integrableI_bounded) auto
  then show integrable  $M (s' \ i)$ 
  by (rule integrableI_simple_bochner_integrable)

  { fix x assume  $x \in \text{space } M \ s' \ i \ x \neq 0$ 
    then have emeasure  $M \ \{y \in \text{space } M. s' \ i \ y = s' \ i \ x\} \leq \text{emeasure } M \ \{y \in \text{space } M. s' \ i \ y \neq 0\}$ 
    by (intro emeasure_mono) auto
    also have  $\dots < \infty$ 
    using sbi by (auto elim: simple_bochner_integrable.cases simp: less_top)
    finally have emeasure  $M \ \{y \in \text{space } M. s' \ i \ y = s' \ i \ x\} \neq \infty$  by simp }

```

```

then show  $P (s' i)$ 
  by (subst  $s'_eq$ ) (auto intro!: sum base simp: less_top)

fix  $x$  assume  $x \in \text{space } M$  with  $s$  show  $(\lambda i. s' i x) \longrightarrow f x$ 
  by (simp add:  $s'_eq_s$ )
show  $\text{norm } (s' i x) \leq 2 * \text{norm } (f x)$ 
  using  $\langle x \in \text{space } M \rangle s$  by (simp add:  $s'_eq_s$ )
qed fact
qed

```

```

lemma integral_eq_zero_AE:
  ( $AE x \text{ in } M. f x = 0$ )  $\implies$   $\text{integral}^L M f = 0$ 
  using integral_cong_AE[of  $f M \lambda_. 0$ ]
  by (cases integrable  $M f$ ) (simp_all add: not_integrable_integral_eq)

```

```

lemma integral_nonneg_eq_0_iff_AE:
  fixes  $f :: \_ \Rightarrow \text{real}$ 
  assumes  $f[\text{measurable}]$ : integrable  $M f$  and nonneg:  $AE x \text{ in } M. 0 \leq f x$ 
  shows  $\text{integral}^L M f = 0 \iff (AE x \text{ in } M. f x = 0)$ 
proof
  assume  $\text{integral}^L M f = 0$ 
  then have  $\text{integral}^N M f = 0$ 
    using nn_integral_eq_integral[OF  $f$  nonneg] by simp
  then have  $AE x \text{ in } M. \text{ennreal } (f x) \leq 0$ 
    by (simp add: nn_integral_0_iff_AE)
  with nonneg show  $AE x \text{ in } M. f x = 0$ 
    by auto
qed (auto simp: integral_eq_zero_AE)

```

```

lemma integral_mono_AE:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes integrable  $M f$  integrable  $M g$   $AE x \text{ in } M. f x \leq g x$ 
  shows  $\text{integral}^L M f \leq \text{integral}^L M g$ 
proof -
  have  $0 \leq \text{integral}^L M (\lambda x. g x - f x)$ 
    using assms by (intro integral_nonneg_AE integrable_diff assms) auto
  also have  $\dots = \text{integral}^L M g - \text{integral}^L M f$ 
    by (intro integral_diff assms)
  finally show ?thesis by simp
qed

```

```

lemma integral_mono:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  shows integrable  $M f \implies$  integrable  $M g \implies (\bigwedge x. x \in \text{space } M \implies f x \leq g x) \implies$ 
 $\text{integral}^L M f \leq \text{integral}^L M g$ 
  by (intro integral_mono_AE) auto

```

```

lemma integral_norm_bound_integral:

```

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{banach,second_countable_topology}
assumes integrable M f integrable M g  $\wedge x. x \in \text{space } M \implies \text{norm}(f x) \leq g x$ 
shows norm  $(\int x. f x \partial M) \leq (\int x. g x \partial M)$ 
by (smt (verit, best) assms integrable_norm integral_mono integral_norm_bound)

```

lemma *integral_abs_bound_integral*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  real
assumes integrable M f integrable M g  $\wedge x. x \in \text{space } M \implies |f x| \leq g x$ 
shows  $|\int x. f x \partial M| \leq (\int x. g x \partial M)$ 
by (metis integral_norm_bound_integral assms real_norm_def)

```

The next two statements are useful to bound Lebesgue integrals, as they avoid one integrability assumption. The price to pay is that the upper function has to be nonnegative, but this is often true and easy to check in computations.

lemma *integral_mono_AE'*:

```

fixes f::_  $\Rightarrow$  real
assumes integrable M f AE x in M. g x  $\leq$  f x AE x in M.  $0 \leq$  f x
shows  $(\int x. g x \partial M) \leq (\int x. f x \partial M)$ 
by (metis assms integral_mono_AE integral_nonneg_AE not_integrable_integral_eq)

```

lemma *integral_mono'*:

```

fixes f::_  $\Rightarrow$  real
assumes integrable M f  $\wedge x. x \in \text{space } M \implies g x \leq f x \wedge x. x \in \text{space } M \implies 0 \leq f x$ 
shows  $(\int x. g x \partial M) \leq (\int x. f x \partial M)$ 
by (rule integral_mono_AE', insert assms, auto)

```

lemma (in *finite_measure*) *integrable_measure*:

```

assumes I: disjoint_family_on X I countable I
shows integrable (count_space I)  $(\lambda i. \text{measure } M (X i))$ 

```

proof –

```

have  $(\int^{+i. \text{measure } M (X i) \partial \text{count\_space } I} = (\int^{+i. \text{measure } M (if X i \in \text{sets } M \text{ then } X i \text{ else } \{\}) \partial \text{count\_space } I}$ 

```

```

by (auto intro!: nn_integral_cong measure_notin_sets)

```

```

also have  $\dots = \text{measure } M (\bigcup i \in I. if X i \in \text{sets } M \text{ then } X i \text{ else } \{\})$ 

```

```

using I unfolding emeasure_eq_measure[symmetric]

```

```

by (subst emeasure_UN_countable) (auto simp: disjoint_family_on_def)

```

```

finally show ?thesis

```

```

by (auto intro!: integrableI_bounded)

```

qed

lemma *nn_integral_nonneg_infinite*:

```

fixes f::'a  $\Rightarrow$  real
assumes f  $\in$  borel_measurable M  $\neg$  integrable M f AE x in M. f x  $\geq$  0
shows  $(\int^{+x. f x \partial M}) = \infty$ 
using assms integrableI_nonneg less_top by auto

```

lemma *integral_real_bounded*:

2400

```

    assumes  $0 \leq r$   $\text{integral}^N M f \leq \text{ennreal } r$ 
    shows  $\text{integral}^L M f \leq r$ 
  proof cases
    assume [simp]:  $\text{integrable } M f$ 

    have  $\text{integral}^L M (\lambda x. \max 0 (f x)) = \text{integral}^N M (\lambda x. \max 0 (f x))$ 
      by (intro nn_integral_eq_integral[symmetric]) auto
    also have  $\dots = \text{integral}^N M f$ 
      by (intro nn_integral_cong) (simp add: max_def ennreal_neg)
    also have  $\dots \leq r$ 
      by fact
    finally have  $\text{integral}^L M (\lambda x. \max 0 (f x)) \leq r$ 
      using  $\langle 0 \leq r \rangle$  by simp

    moreover have  $\text{integral}^L M f \leq \text{integral}^L M (\lambda x. \max 0 (f x))$ 
      by (rule integral_mono_AE) auto
    ultimately show ?thesis
      by simp
  next
    assume  $\neg \text{integrable } M f$  then show ?thesis
      using  $\langle 0 \leq r \rangle$  by (simp add: not_integrable_integral_eq)
  qed

lemma integrable_MIN:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \text{real}$ 
  shows  $\llbracket \text{finite } I; I \neq \{\}; \bigwedge i. i \in I \implies \text{integrable } M (f i) \rrbracket$ 
     $\implies \text{integrable } M (\lambda x. \text{MIN } i \in I. f i x)$ 
  by (induct rule: finite_ne_induct) simp+

lemma integrable_MAX:
  fixes  $f :: \_ \Rightarrow \_ \Rightarrow \text{real}$ 
  shows  $\llbracket \text{finite } I; I \neq \{\}; \bigwedge i. i \in I \implies \text{integrable } M (f i) \rrbracket$ 
     $\implies \text{integrable } M (\lambda x. \text{MAX } i \in I. f i x)$ 
  by (induct rule: finite_ne_induct) simp+

theorem integral_Markov_inequality:
  assumes [measurable]:  $\text{integrable } M u$  and AE  $x$  in  $M. 0 \leq u x < (c :: \text{real})$ 
  shows  $(\text{emeasure } M) \{x \in \text{space } M. u x \geq c\} \leq (1/c) * (\int x. u x \partial M)$ 
  proof -
    have  $(\int^+ x. \text{ennreal}(u x) * \text{indicator } (\text{space } M) x \partial M) \leq (\int^+ x. u x \partial M)$ 
      by (rule nn_integral_mono_AE, auto simp:  $\langle c > 0 \rangle$  less_eq_real_def)
    also have  $\dots = (\int x. u x \partial M)$ 
      by (rule nn_integral_eq_integral, auto simp: assms)
    finally have  $*$ :  $(\int^+ x. \text{ennreal}(u x) * \text{indicator } (\text{space } M) x \partial M) \leq (\int x. u x \partial M)$ 
      by simp

    have  $\{x \in \text{space } M. u x \geq c\} = \{x \in \text{space } M. \text{ennreal}(1/c) * u x \geq 1\} \cap (\text{space } M)$ 

```

```

using <c>0> by (auto simp: ennreal_mult'[symmetric])
then have emeasure M {x ∈ space M. u x ≥ c} = emeasure M ({x ∈ space M.
ennreal(1/c) * u x ≥ 1})
by simp
also have ... ≤ ennreal(1/c) * (∫+ x. ennreal(u x) * indicator (space M) x
∂M)
by (rule nn_integral_Markov_inequality) (auto simp: assms)
also have ... ≤ ennreal(1/c) * (∫ x. u x ∂M)
by (rule mult_left_mono) (use * <c > 0> in auto)
finally show ?thesis
using <0 < c> by (simp add: ennreal_mult'[symmetric])
qed

```

theorem *integral_Markov_inequality_measure*:

```

assumes [measurable]: integrable M u and A ∈ sets M and AE x in M. 0 ≤ u
x 0 < (c::real)
shows measure M {x ∈ space M. u x ≥ c} ≤ (∫ x. u x ∂M) / c
proof -
have le: emeasure M {x ∈ space M. u x ≥ c} ≤ ennreal ((1/c) * (∫ x. u x ∂M))
by (rule integral_Markov_inequality) (use assms in auto)
also have ... < top
by auto
finally have ennreal (measure M {x ∈ space M. u x ≥ c}) = emeasure M {x ∈ space
M. u x ≥ c}
by (intro emeasure_eq_ennreal_measure [symmetric]) auto
also note le
finally show ?thesis
by (simp add: assms divide_nonneg_pos integral_nonneg_AE)
qed

```

theorem (*in finite_measure*) *second_moment_method*:

```

assumes [measurable]: f ∈ M →M borel
assumes integrable M (λx. f x ^ 2)
defines μ ≡ lebesgue_integral M f
assumes a > 0
shows measure M {x ∈ space M. |f x| ≥ a} ≤ lebesgue_integral M (λx. f x ^ 2)
/ a2
proof -
have integrable: integrable M f
using assms by (blast dest: square_integrable_imp_integrable)
have {x ∈ space M. |f x| ≥ a} = {x ∈ space M. f x ^ 2 ≥ a2}
using <a > 0> by (simp flip: abs_le_square_iff)
hence measure M {x ∈ space M. |f x| ≥ a} = measure M {x ∈ space M. f x ^ 2 ≥
a2}
by simp
also have ... ≤ lebesgue_integral M (λx. f x ^ 2) / a2
using assms by (intro integral_Markov_inequality_measure) auto
finally show ?thesis .
qed

```

lemma *integral_ineq_eq_0_then_AE*:

fixes $f :: _ \Rightarrow \text{real}$

assumes $AE\ x\ \text{in}\ M. f\ x \leq g\ x\ \text{integrable}\ M\ f\ \text{integrable}\ M\ g$

$$(\int x. f\ x\ \partial M) = (\int x. g\ x\ \partial M)$$

shows $AE\ x\ \text{in}\ M. f\ x = g\ x$

proof –

define h **where** $h = (\lambda x. g\ x - f\ x)$

have $AE\ x\ \text{in}\ M. h\ x = 0$

apply (*subst integral_nonneg_eq_0_iff_AE[symmetric]*)

unfolding h_def **using** *assms* **by** *auto*

then show *?thesis* **unfolding** h_def **by** *auto*

qed

lemma *not_AE_zero_int_E*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $AE\ x\ \text{in}\ M. f\ x \geq 0\ (\int x. f\ x\ \partial M) > 0$

and [*measurable*]: $f \in \text{borel_measurable}\ M$

shows $\exists A\ e. A \in \text{sets}\ M \wedge e > 0 \wedge \text{emeasure}\ M\ A > 0 \wedge (\forall x \in A. f\ x \geq e)$

proof (*rule not_AE_zero_E, auto simp: assms*)

assume $*$: $AE\ x\ \text{in}\ M. f\ x = 0$

have $(\int x. f\ x\ \partial M) = (\int x. 0\ \partial M)$

by (*rule integral_cong_AE, auto simp: **)

then have $(\int x. f\ x\ \partial M) = 0$ **by** *simp*

then show *False* **using** *assms(2)* **by** *simp*

qed

proposition *tendsto_L1_int*:

fixes $u :: _ \Rightarrow _ \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$

assumes [*measurable*]: $\bigwedge n. \text{integrable}\ M\ (u\ n)\ \text{integrable}\ M\ f$

and $((\lambda n. (\int^+ x. \text{norm}(u\ n\ x - f\ x)\ \partial M)) \longrightarrow 0)\ F$

shows $((\lambda n. (\int x. u\ n\ x\ \partial M)) \longrightarrow (\int x. f\ x\ \partial M))\ F$

proof –

have $((\lambda n. \text{norm}((\int x. u\ n\ x\ \partial M) - (\int x. f\ x\ \partial M))) \longrightarrow (0 :: \text{ennreal}))\ F$

proof (*rule tendsto_sandwich[of $\lambda _ . 0$, where $?h = \lambda n. (\int^+ x. \text{norm}(u\ n\ x - f\ x)\ \partial M)$], auto simp: assms*)

{

fix n

have $(\int x. u\ n\ x\ \partial M) - (\int x. f\ x\ \partial M) = (\int x. u\ n\ x - f\ x\ \partial M)$

by (*simp add: assms*)

then have $\text{norm}((\int x. u\ n\ x\ \partial M) - (\int x. f\ x\ \partial M)) = \text{norm}(\int x. u\ n\ x - f\ x\ \partial M)$

by *auto*

also have $\dots \leq (\int x. \text{norm}(u\ n\ x - f\ x)\ \partial M)$

by (*rule integral_norm_bound*)

finally have $\text{ennreal}(\text{norm}((\int x. u\ n\ x\ \partial M) - (\int x. f\ x\ \partial M))) \leq (\int x. \text{norm}(u\ n\ x - f\ x)\ \partial M)$

by *simp*

also have $\dots = (\int^+ x. \text{norm}(u\ n\ x - f\ x)\ \partial M)$

```

      by (simp add: assms nn_integral_eq_integral)
      finally have norm((∫ x. u n x ∂M) - (∫ x. f x ∂M)) ≤ (∫+x. norm(u n x
- f x) ∂M)
      by simp
    }
    then show eventually (λn. norm((∫ x. u n x ∂M) - (∫ x. f x ∂M)) ≤ (∫+x.
norm(u n x - f x) ∂M)) F
      by auto
  qed
  then have ((λn. norm((∫ x. u n x ∂M) - (∫ x. f x ∂M))) → 0) F
    by (simp flip: ennreal_0)
  then have ((λn. ((∫ x. u n x ∂M) - (∫ x. f x ∂M))) → 0) F
    using tendsto_norm_zero_iff by blast
  then show ?thesis
    using Lim_null by auto
qed

```

The next lemma asserts that, if a sequence of functions converges in L^1 , then it admits a subsequence that converges almost everywhere.

proposition *tendsto_L1_AE_subseq*:

```

  fixes u :: nat ⇒ 'a ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: ∧n. integrable M (u n)
    and (λn. (∫ x. norm(u n x) ∂M)) → 0
  shows ∃ r::nat⇒nat. strict_mono r ∧ (AE x in M. (λn. u (r n) x) → 0)
proof -
  {
    fix k
    have eventually (λn. (∫ x. norm(u n x) ∂M) < (1/2)~k) sequentially
      using order_tendstoD(2)[OF assms(2)] by auto
    with eventually_elim2[OF eventually_gt_at_top[of k] this]
    have ∃ n>k. (∫ x. norm(u n x) ∂M) < (1/2)~k
      by (metis eventually_False_sequentially)
  }
  then have ∃ r. ∀ n. True ∧ (r (Suc n) > r n ∧ (∫ x. norm(u (r (Suc n)) x) ∂M)
< (1/2)~(r n))
    by (intro dependent_nat_choice, auto)
  then obtain r0 where r0: strict_mono r0 ∧ n. (∫ x. norm(u (r0 (Suc n)) x)
∂M) < (1/2)~(r0 n)
    by (auto simp: strict_mono_Suc_iff)
  define r where r = (λn. r0(n+1))
  have strict_mono r unfolding r_def using r0(1) by (simp add: strict_mono_Suc_iff)
  have I: (∫+x. norm(u (r n) x) ∂M) < ennreal((1/2)~n) for n
  proof -
    have r0 n ≥ n using ⟨strict_mono r0⟩ by (simp add: seq_suble)
    have (1/2::real)~(r0 n) ≤ (1/2)~n by (rule power_decreasing[OF ⟨r0 n ≥
n⟩], auto)
    then have (∫ x. norm(u (r0 (Suc n)) x) ∂M) < (1/2)~n
      using r0(2) less_le_trans by blast
    then have (∫ x. norm(u (r n) x) ∂M) < (1/2)~n

```

unfolding r_def **by** *auto*
moreover have $(\int^+ x. \text{norm}(u (r\ n) x) \partial M) = (\int x. \text{norm}(u (r\ n) x) \partial M)$
by (*rule nn_integral_eq_integral*, *auto simp: integrable_norm[OF assms(1)[of r n]]*)
ultimately show *?thesis* **by** (*auto intro: ennreal_lessI*)
qed

have $AE\ x\ \text{in}\ M. \limsup (\lambda n. \text{ennreal} (\text{norm}(u (r\ n) x))) \leq 0$

proof (*rule AE_upper_bound_inf_ennreal*)

fix $e::\text{real}$ **assume** $e > 0$

define A **where** $A = (\lambda n. \{x \in \text{space } M. \text{norm}(u (r\ n) x) \geq e\})$

have A_meas [*measurable*]: $\bigwedge n. A\ n \in \text{sets } M$ **unfolding** A_def **by** *auto*

have A_bound : $\text{emeasure } M (A\ n) < (1/e) * \text{ennreal}((1/2)^{\wedge}n)$ **for** n

proof –

have $*$: $\text{indicator } (A\ n) x \leq (1/e) * \text{ennreal}(\text{norm}(u (r\ n) x))$ **for** x

using $\langle 0 < e \rangle$ **by** (*cases* $x \in A\ n$) (*auto simp: A_def ennreal_mult[symmetric]*)

have $\text{emeasure } M (A\ n) = (\int^+ x. \text{indicator } (A\ n) x \partial M)$ **by** *auto*

also have $\dots \leq (\int^+ x. (1/e) * \text{ennreal}(\text{norm}(u (r\ n) x)) \partial M)$

by (*meson * nn_integral_mono*)

also have $\dots = (1/e) * (\int^+ x. \text{norm}(u (r\ n) x) \partial M)$

using $\langle e > 0 \rangle$ **by** (*force simp add: intro: nn_integral_cmult*)

also have $\dots < (1/e) * \text{ennreal}((1/2)^{\wedge}n)$

using $I[\text{of } n] \langle e > 0 \rangle$ **by** (*intro ennreal_mult_strict_left_mono*) *auto*

finally show *?thesis* **by** *simp*

qed

have A_fin : $\text{emeasure } M (A\ n) < \infty$ **for** n

using $\langle e > 0 \rangle A_bound[\text{of } n]$

by (*auto simp: ennreal_mult_less_top[symmetric]*)

have A_sum : $\text{summable } (\lambda n. \text{measure } M (A\ n))$

proof (*rule summable_comparison_test'*)

have $\text{summable } (\lambda n. (1/2::\text{real})^{\wedge}n)$

by (*simp add: summable_geometric*)

then show $\text{summable } (\lambda n. (1/e) * (1/2)^{\wedge}n)$ **using** *summable_mult* **by** *blast*

fix $n::\text{nat}$ **assume** $n \geq 0$

have $\text{norm}(\text{measure } M (A\ n)) = \text{measure } M (A\ n)$ **by** *simp*

also have $\dots = \text{enn2real}(\text{emeasure } M (A\ n))$ **unfolding** *measure_def* **by**

simp

also have $\dots < \text{enn2real}((1/e) * (1/2)^{\wedge}n)$

using $A_bound[\text{of } n] \langle \text{emeasure } M (A\ n) < \infty \rangle \langle 0 < e \rangle$

by (*auto simp: emeasure_eq_ennreal_measure ennreal_mult[symmetric] ennreal_less_iff*)

also have $\dots = (1/e) * (1/2)^{\wedge}n$

using $\langle 0 < e \rangle$ **by** *auto*

finally show $\text{norm}(\text{measure } M (A\ n)) \leq (1/e) * (1/2)^{\wedge}n$ **by** *simp*

qed

have $AE\ x\ \text{in}\ M. \text{eventually } (\lambda n. x \in \text{space } M - A\ n)$ *sequentially*

by (*rule borel_cantelli_AE1[OF A_meas A_fin A_sum]*)


```

moreover
{
  fix  $x$  assume eventually  $(\lambda n. x \in \text{space } M - A \ n)$  sequentially
  moreover have  $\text{norm}(u \ (r \ n) \ x) \leq \text{ennreal } e$  if  $x \in \text{space } M - A \ n$  for  $n$ 
    using that unfolding A_def by (auto intro: ennreal_leI)
  ultimately have eventually  $(\lambda n. \text{norm}(u \ (r \ n) \ x) \leq \text{ennreal } e)$  sequentially
    by (simp add: eventually_mono)
  then have  $\text{limsup } (\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))) \leq e$ 
    by (simp add: Limsup_bounded)
}
ultimately show AE x in M. limsup  $(\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))) \leq 0 +$ 
ennreal e
  by auto
qed
moreover
{
  fix  $x$  assume  $x: \text{limsup } (\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))) \leq 0$ 
  moreover have  $\text{liminf } (\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))) \leq 0$ 
    by (metis x Liminf_le_Limsup le_zero_eq sequentially_bot)
  ultimately have  $(\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))) \longrightarrow 0$ 
    using tendsto_Limsup[of sequentially  $\lambda n. \text{ennreal } (\text{norm}(u \ (r \ n) \ x))$ ] by auto
  then have  $(\lambda n. \text{norm}(u \ (r \ n) \ x)) \longrightarrow 0$ 
    by (simp flip: ennreal_0)
  then have  $(\lambda n. u \ (r \ n) \ x) \longrightarrow 0$ 
    by (simp add: tendsto_norm_zero_iff)
}
ultimately have AE x in M.  $(\lambda n. u \ (r \ n) \ x) \longrightarrow 0$  by auto
then show ?thesis using <strict_mono r> by auto
qed

```

7.9.1 Restricted measure spaces

lemma *integrable_restrict_space:*

```

fixes  $f :: 'a \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
assumes  $\Omega[\text{simp}]: \Omega \cap \text{space } M \in \text{sets } M$ 
shows integrable  $(\text{restrict\_space } M \ \Omega) \ f \iff \text{integrable } M \ (\lambda x. \text{indicator } \Omega \ x$ 
*R f x)
unfolding integrable_iff_bounded
  borel_measurable_restrict_space_iff[OF  $\Omega$ ]
  nn_integral_restrict_space[OF  $\Omega$ ]
by (simp add: ac_simps ennreal_indicator ennreal_mult)

```

lemma *integral_restrict_space:*

```

fixes  $f :: 'a \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
assumes  $\Omega[\text{simp}]: \Omega \cap \text{space } M \in \text{sets } M$ 
shows  $\text{integral}^L (\text{restrict\_space } M \ \Omega) \ f = \text{integral}^L M \ (\lambda x. \text{indicator } \Omega \ x *_{R} f$ 
x)
proof (rule integral_eq_cases)
  assume integrable  $(\text{restrict\_space } M \ \Omega) \ f$ 

```

```

then show ?thesis
proof induct
  case (base A c) then show ?case
    by (simp add: indicator_inter_arith[symmetric] sets_restrict_space_iff
        emeasure_restrict_space Int_absorb1 measure_restrict_space)
next
  case (add g f) then show ?case
    by (simp add: scaleR_add_right integrable_restrict_space)
next
  case (lim f s)
  show ?case
  proof (rule LIMSEQ_unique)
  show ( $\lambda i.$  integralL (restrict_space M  $\Omega$ ) (s i))  $\longrightarrow$  integralL (restrict_space
M  $\Omega$ ) f
    using lim by (intro integral_dominated_convergence[where w= $\lambda x.$  2 *
norm (f x)]) simp_all

    show ( $\lambda i.$  integralL (restrict_space M  $\Omega$ ) (s i))  $\longrightarrow$  ( $\int x.$  indicator  $\Omega$  x
*_R f x  $\partial M$ )
      unfolding lim
      using lim
      by (intro integral_dominated_convergence[where w= $\lambda x.$  2 * norm (indicator
 $\Omega$  x *_R f x)])
        (auto simp: space_restrict_space integrable_restrict_space simp del:
norm_scaleR
split: split_indicator)

    qed
  qed
qed (simp add: integrable_restrict_space)

lemma integral_empty:
assumes space M = {}
shows integralL M f = 0
by (metis AE_I2 assms empty_iff integral_eq_zero_AE)

```

7.9.2 Measure spaces with an associated density

```

lemma integrable_density:
fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology} and g :: 'a  $\Rightarrow$  real
assumes [measurable]: f  $\in$  borel_measurable M g  $\in$  borel_measurable M
and nn: AE x in M. 0  $\leq$  g x
shows integrable (density M g) f  $\longleftrightarrow$  integrable M ( $\lambda x.$  g x *_R f x)
unfolding integrable_iff_bounded using nn
apply (simp add: nn_integral_density_less_top[symmetric])
apply (intro arg_cong2[where f=(=)] refl nn_integral_cong_AE)
apply (auto simp: ennreal_mult)
done

```

```

lemma integral_density:

```

```

fixes  $f :: 'a \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$  and  $g :: 'a \Rightarrow \text{real}$ 
assumes  $f: f \in \text{borel\_measurable } M$ 
and  $g[\text{measurable}]: g \in \text{borel\_measurable } M \text{ AE } x \text{ in } M. 0 \leq g \ x$ 
shows  $\text{integral}^L (\text{density } M \ g) \ f = \text{integral}^L M (\lambda x. g \ x *_{\mathbb{R}} f \ x)$ 
proof (rule integral_eq_cases)
assume  $\text{integrable } (\text{density } M \ g) \ f$ 
then show ?thesis
proof induct
case (base  $A \ c$ )
then have  $[\text{measurable}]: A \in \text{sets } M \text{ by auto}$ 

have  $\text{int}: \text{integrable } M (\lambda x. g \ x * \text{indicator } A \ x)$ 
using  $g \ \text{base } \text{integrable\_density[of indicator } A :: 'a \Rightarrow \text{real } M \ g]$  by simp
then have  $\text{integral}^L M (\lambda x. g \ x * \text{indicator } A \ x) = (\int^+ x. \text{ennreal } (g \ x * \text{indicator } A \ x) \ \partial M)$ 
using  $g \ \text{by } (\text{subst } \text{nn\_integral\_eq\_integral}) \ \text{auto}$ 
also have  $\dots = (\int^+ x. \text{ennreal } (g \ x) * \text{indicator } A \ x \ \partial M)$ 
by (intro nn\_integral\_cong) (auto split: split\_indicator)
also have  $\dots = \text{emeasure } (\text{density } M \ g) \ A$ 
by (rule emeasure\_density[symmetric]) auto
also have  $\dots = \text{ennreal } (\text{measure } (\text{density } M \ g) \ A)$ 
using base by (auto intro: emeasure\_eq\_ennreal\_measure)
also have  $\dots = \text{integral}^L (\text{density } M \ g) (\text{indicator } A)$ 
using base by simp
finally show ?case
using base g
apply (simp add: int integral\_nonneg\_AE)
apply (subst (asm) ennreal\_inj)
apply (auto intro!: integral\_nonneg\_AE)
done
next
case (add  $f \ h$ )
then have  $[\text{measurable}]: f \in \text{borel\_measurable } M \ h \in \text{borel\_measurable } M$ 
by (auto dest!: borel\_measurable\_integrable)
from add g show ?case
by (simp add: scaleR\_add\_right integrable\_density)
next
case (lim  $f \ s$ )
have  $[\text{measurable}]: f \in \text{borel\_measurable } M \ \wedge \ i. s \ i \in \text{borel\_measurable } M$ 
using lim(1,5)[THEN borel\_measurable\_integrable] by auto

show ?case
proof (rule LIMSEQ\_unique)
show  $(\lambda i. \text{integral}^L M (\lambda x. g \ x *_{\mathbb{R}} s \ i \ x)) \longrightarrow \text{integral}^L M (\lambda x. g \ x *_{\mathbb{R}} f \ x)$ 
proof (rule integral\_dominated\_convergence)
show  $\text{integrable } M (\lambda x. 2 * \text{norm } (g \ x *_{\mathbb{R}} f \ x))$ 
by (intro integrable\_mult\_right integrable\_norm integrable\_density[THEN iffD1] lim g) auto

```

```

show  $AE\ x\ in\ M. (\lambda i. g\ x\ *_R\ s\ i\ x) \longrightarrow g\ x\ *_R\ f\ x$ 
  using  $lim(3)$  by (auto intro!:  $tendsto\_scaleR\ AE\_I2[of\ M]$ )
show  $\bigwedge i. AE\ x\ in\ M. norm\ (g\ x\ *_R\ s\ i\ x) \leq 2 * norm\ (g\ x\ *_R\ f\ x)$ 
  using  $lim(4)$   $g$  by (auto intro!:  $AE\_I2[of\ M]\ mult\_left\_mono\ simp$ :
field\_simps)
qed auto
show  $(\lambda i. integral^L\ M\ (\lambda x. g\ x\ *_R\ s\ i\ x)) \longrightarrow integral^L\ (density\ M\ g)\ f$ 
  unfolding  $lim(2)[symmetric]$ 
  by (rule integral\_dominated\_convergence[where w= $\lambda x. 2 * norm\ (f\ x)$ ])
  (use lim in auto)
qed
qed
qed (simp add: f g integrable\_density)

```

lemma

```

fixes  $g :: 'a \Rightarrow real$ 
assumes  $f \in borel\_measurable\ M\ AE\ x\ in\ M. 0 \leq f\ x\ g \in borel\_measurable\ M$ 
shows  $integral\_real\_density: integral^L\ (density\ M\ f)\ g = (\int\ x. f\ x\ * g\ x\ \partial M)$ 
  and  $integrable\_real\_density: integrable\ (density\ M\ f)\ g \longleftrightarrow integrable\ M\ (\lambda x. f\ x\ * g\ x)$ 
using assms integral\_density[of g M f] integrable\_density[of g M f] by auto

```

lemma *has_bochner_integral_density:*

```

fixes  $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$  and  $g :: 'a \Rightarrow real$ 
shows  $f \in borel\_measurable\ M \implies g \in borel\_measurable\ M \implies (AE\ x\ in\ M. 0 \leq g\ x) \implies$ 
   $has\_bochner\_integral\ M\ (\lambda x. g\ x\ *_R\ f\ x)\ x \implies has\_bochner\_integral\ (density\ M\ g)\ f\ x$ 
by (simp add: has_bochner\_integral\_iff\_integrable\_density\_integral\_density)

```

7.9.3 Distributions

lemma *integrable_distr_eq:*

```

fixes  $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$ 
assumes [measurable]:  $g \in measurable\ M\ N\ f \in borel\_measurable\ N$ 
shows  $integrable\ (distr\ M\ N\ g)\ f \longleftrightarrow integrable\ M\ (\lambda x. f\ (g\ x))$ 
unfolding integrable\_iff\_bounded by (simp_all add: nn\_integral\_distr)

```

lemma *integrable_distr:*

```

fixes  $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$ 
shows  $T \in measurable\ M\ M' \implies integrable\ (distr\ M\ M'\ T)\ f \implies integrable\ M\ (\lambda x. f\ (T\ x))$ 
by (metis integrable\_distr\_eq integrable\_iff\_bounded measurable\_distr\_eq1)

```

lemma *integral_distr:*

```

fixes  $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$ 
assumes  $g[measurable]: g \in measurable\ M\ N$  and  $f: f \in borel\_measurable\ N$ 
shows  $integral^L\ (distr\ M\ N\ g)\ f = integral^L\ M\ (\lambda x. f\ (g\ x))$ 
proof (rule integral\_eq\_cases)

```

```

assume integrable (distr M N g) f
then show ?thesis
proof induct
  case (base A c)
  then have [measurable]: A ∈ sets N by auto
  from base have int: integrable (distr M N g) (λa. indicator A a *R c)
  by (intro integrable_indicator)

  have integralL (distr M N g) (λa. indicator A a *R c) = measure (distr M N
g) A *R c
  using base by auto
  also have ... = measure M (g -' A ∩ space M) *R c
  by (subst measure_distr) auto
  also have ... = integralL M (λa. indicator (g -' A ∩ space M) a *R c)
  using base by (auto simp: emeasure_distr)
  also have ... = integralL M (λa. indicator A (g a) *R c)
  using int base by (intro integral_cong_AE) (auto simp: emeasure_distr split:
split_indicator)
  finally show ?case .
next
case (add f h)
then have [measurable]: f ∈ borel_measurable N h ∈ borel_measurable N
by (auto dest!: borel_measurable_integrable)
from add g show ?case
by (simp add: scaleR_add_right integrable_distr_eq)
next
case (lim f s)
have [measurable]: f ∈ borel_measurable N ∧ i. s i ∈ borel_measurable N
using lim(1,5)[THEN borel_measurable_integrable] by auto

show ?case
proof (rule LIMSEQ_unique)
  show (λi. integralL M (λx. s i (g x))) ⟶ integralL M (λx. f (g x))
  proof (rule integral_dominated_convergence)
    show integrable M (λx. 2 * norm (f (g x)))
    using lim by (auto simp: integrable_distr_eq)
    show AE x in M. (λi. s i (g x)) ⟶ f (g x)
    using lim(3) g[THEN measurable_space] by auto
    show ∧i. AE x in M. norm (s i (g x)) ≤ 2 * norm (f (g x))
    using lim(4) g[THEN measurable_space] by auto
  qed auto
  show (λi. integralL M (λx. s i (g x))) ⟶ integralL (distr M N g) f
  unfolding lim(2)[symmetric]
  by (rule integral_dominated_convergence[where w=λx. 2 * norm (f x)])
  (use lim in auto)
qed
qed
qed (simp add: f g integrable_distr_eq)

```

lemma *has_bochner_integral_distr*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second_countable_topology}\}$
shows $f \in \text{borel_measurable } N \implies g \in \text{measurable } M N \implies$
 $\text{has_bochner_integral } M (\lambda x. f (g x)) x \implies \text{has_bochner_integral } (\text{distr } M N$
 $g) f x$
by (*simp add: has_bochner_integral_iff integrable_distr_eq integral_distr*)

7.9.4 Lebesgue integration on *count_space*

lemma *integrable_count_space*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second_countable_topology}\}$
shows $\text{finite } X \implies \text{integrable } (\text{count_space } X) f$
by (*auto simp: nn_integral_count_space_integrable_iff_bounded*)

lemma *measure_count_space[simp]*:
 $B \subseteq A \implies \text{finite } B \implies \text{measure } (\text{count_space } A) B = \text{card } B$
unfolding *measure_def* **by** (*subst emeasure_count_space*) *auto*

lemma *lebesgue_integral_count_space_finite_support*:
assumes $f: \text{finite } \{a \in A. f a \neq 0\}$
shows $(\int x. f x \partial \text{count_space } A) = (\sum a \mid a \in A \wedge f a \neq 0. f a)$
proof –
have $\text{eq}: \bigwedge x. x \in A \implies (\sum a \mid x = a \wedge a \in A \wedge f a \neq 0. f a) = (\sum x \in \{x\}. f x)$
by (*intro sum.mono_neutral_cong_left*) *auto*

have $(\int x. f x \partial \text{count_space } A) = (\int x. (\sum a \mid a \in A \wedge f a \neq 0. \text{indicator } \{a\}$
 $x *_R f a) \partial \text{count_space } A)$
by (*intro integral_cong_refl*) (*simp add: f eq*)
also have $\dots = (\sum a \mid a \in A \wedge f a \neq 0. \text{measure } (\text{count_space } A) \{a\} *_R f a)$
by (*subst integral_sum*) (*auto intro!: sum.cong*)
finally show *?thesis*
by *auto*
qed

lemma *lebesgue_integral_count_space_finite*: $\text{finite } A \implies (\int x. f x \partial \text{count_space } A) = (\sum a \in A. f a)$
by (*subst lebesgue_integral_count_space_finite_support*)
(auto intro!: sum.mono_neutral_cong_left)

lemma *integrable_count_space_nat_iff*:
fixes $f :: \text{nat} \Rightarrow _ :: \{\text{banach}, \text{second_countable_topology}\}$
shows $\text{integrable } (\text{count_space } \text{UNIV}) f \iff \text{summable } (\lambda x. \text{norm } (f x))$
by (*auto simp: integrable_iff_bounded nn_integral_count_space_nat ennreal_suminf_neq_top*
intro: summable_suminf_not_top)

lemma *sums_integral_count_space_nat*:
fixes $f :: \text{nat} \Rightarrow _ :: \{\text{banach}, \text{second_countable_topology}\}$
assumes $*$: $\text{integrable } (\text{count_space } \text{UNIV}) f$
shows $f \text{ sums } (\text{integral}^L (\text{count_space } \text{UNIV}) f)$

proof –

let $?f = \lambda n i. \text{indicator } \{n\} i *_{\mathbb{R}} f i$
have $f': \bigwedge n i. ?f n i = \text{indicator } \{n\} i *_{\mathbb{R}} f n$
by (*auto simp: fun_eq_iff split: split_indicator*)

have $(\lambda i. \int n. ?f i n \partial \text{count_space UNIV}) \text{ sums } \int n. (\sum i. ?f i n) \partial \text{count_space UNIV}$

proof (*rule sums_integral*)

show $\bigwedge i. \text{integrable } (\text{count_space UNIV}) (?f i)$
using * **by** (*intro integrable_mult_indicator auto*)
show $AE n \text{ in } \text{count_space UNIV}. \text{summable } (\lambda i. \text{norm } (?f i n))$
using *summable_finite*[of $\{n\}$ $\lambda i. \text{norm } (?f i n)$ **for** n] **by** *simp*
show $\text{summable } (\lambda i. \int n. \text{norm } (?f i n) \partial \text{count_space UNIV})$
using * **by** (*subst f'*) (*simp add: integrable_count_space_nat_iff*)

qed

also have $(\int n. (\sum i. ?f i n) \partial \text{count_space UNIV}) = (\int n. f n \partial \text{count_space UNIV})$

using *suminf_finite*[of $\{n\}$ $\lambda i. ?f i n$ **for** n] **by** (*auto intro!: integral_cong*)

also have $(\lambda i. \int n. ?f i n \partial \text{count_space UNIV}) = f$

by (*subst f'*) *simp*

finally show *?thesis* .

qed

lemma *integral_count_space_nat*:

fixes $f :: \text{nat} \Rightarrow _ :: \{\text{banach}, \text{second_countable_topology}\}$

shows $\text{integrable } (\text{count_space UNIV}) f \implies \text{integral}^L (\text{count_space UNIV}) f = (\sum x. f x)$

using *sums_integral_count_space_nat sums_unique* **by** *auto*

lemma *integrable_bij_count_space*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second_countable_topology}\}$

assumes $g: \text{bij_betw } g A B$

shows $\text{integrable } (\text{count_space } A) (\lambda x. f (g x)) \longleftrightarrow \text{integrable } (\text{count_space } B) f$

unfolding *integrable_iff_bounded* **by** (*subst nn_integral_bij_count_space[OF g]*) *auto*

lemma *integral_bij_count_space*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second_countable_topology}\}$

assumes $g: \text{bij_betw } g A B$

shows $\text{integral}^L (\text{count_space } A) (\lambda x. f (g x)) = \text{integral}^L (\text{count_space } B) f$

using *g[THEN bij_betw_imp_funcset] distr_bij_count_space [OF g]*

by (*metis borel_measurable_count_space integral_distr measurable_count_space_eq1 space_count_space*)

lemma *has_bochner_integral_count_space_nat*:

fixes $f :: \text{nat} \Rightarrow _ :: \{\text{banach}, \text{second_countable_topology}\}$

shows $\text{has_bochner_integral } (\text{count_space UNIV}) f x \implies f \text{ sums } x$

unfolding *has_bochner_integral_iff* **by** (*auto intro!: sums_integral_count_space_nat*)

7.9.5 Point measure

lemma *lebesgue_integral_point_measure_finite*:
fixes $g :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$
shows $\text{finite } A \implies (\bigwedge a. a \in A \implies 0 \leq f a) \implies$
 $\text{integral}^L (\text{point_measure } A f) g = (\sum_{a \in A}. f a *_{\mathbb{R}} g a)$
by (*simp add: lebesgue_integral_count_space_finite AE_count_space integral_density point_measure_def*)

proposition *integrable_point_measure_finite*:
fixes $g :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$ **and** $f :: 'a \Rightarrow \text{real}$
assumes *finite A*
shows *integrable (point_measure A f) g*
proof –
have *integrable (density (count_space A) ($\lambda x. \text{ennreal } (\text{max } 0 (f x))$)) g*
using *assms*
by (*simp add: integrable_count_space integrable_density*)
then show *?thesis*
by (*smt (verit) AE_I2 borel_measurable_count_space_density_cong ennreal_neg point_measure_def*)
qed

7.9.6 Lebesgue integration on null_measure

lemma *has_bochner_integral_null_measure_iff*[*iff*]:
 $\text{has_bochner_integral } (\text{null_measure } M) f 0 \iff f \in \text{borel_measurable } M$
by (*auto simp: has_bochner_integral.simps simple_bochner_integral_def[abs_def]*
intro!: exI[of _ $\lambda n x. 0$] simple_bochner_integrable.intros)

lemma *integrable_null_measure_iff*[*iff*]: $\text{integrable } (\text{null_measure } M) f \iff f \in \text{borel_measurable } M$
by (*auto simp: integrable.simps*)

lemma *integral_null_measure*[*simp*]: $\text{integral}^L (\text{null_measure } M) f = 0$
using *integral_norm_bound_ennreal not_integrable_integral_eq* **by** *fastforce*

7.9.7 Legacy lemmas for the real-valued Lebesgue integral

theorem *real_lebesgue_integral_def*:
assumes $f[\text{measurable}]$: *integrable M f*
shows $\text{integral}^L M f = \text{enn2real } (\int^+ x. f x \partial M) - \text{enn2real } (\int^+ x. \text{ennreal } (- f x) \partial M)$
proof –
have $\text{integral}^L M f = \text{integral}^L M (\lambda x. \text{max } 0 (f x) - \text{max } 0 (- f x))$
by (*auto intro!: arg_cong[where f=integral^L M]*)
also have $\dots = \text{integral}^L M (\lambda x. \text{max } 0 (f x)) - \text{integral}^L M (\lambda x. \text{max } 0 (- f x))$
by (*intro integral_diff integrable_max integrable_minus integrable_zero f*)
also have $\text{integral}^L M (\lambda x. \text{max } 0 (f x)) = \text{enn2real } (\int^+ x. \text{ennreal } (f x) \partial M)$
by (*subst integral_eq_nn_integral*) (*auto intro!: arg_cong[where f=enn2real]*)

nn_integral_cong_simp: max_def ennreal_neg
also have $\text{integral}^L M (\lambda x. \max 0 (- f x)) = \text{enn2real} (\int^+ x. \text{ennreal} (- f x) \partial M)$
by (*subst integral_eq_nn_integral*) (*auto intro!: arg_cong[where f=enn2real]*)
nn_integral_cong_simp: max_def ennreal_neg
finally show ?thesis .
qed

theorem *real_integrable_def*:

integrable M f \longleftrightarrow $f \in \text{borel_measurable } M \wedge$
 $(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty \wedge (\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$
unfolding *integrable_iff_bounded*
proof (*safe del: notI*)
assume *: $(\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M) < \infty$
have $(\int^+ x. \text{ennreal} (f x) \partial M) \leq (\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M)$
by (*intro nn_integral_mono*) *auto*
also note *
finally show $(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty$
by *simp*
have $(\int^+ x. \text{ennreal} (- f x) \partial M) \leq (\int^+ x. \text{ennreal} (\text{norm} (f x)) \partial M)$
by (*intro nn_integral_mono*) *auto*
also note *
finally show $(\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$
by *simp*
next
assume [*measurable*]: $f \in \text{borel_measurable } M$
assume *fin*: $(\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty \wedge (\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$
have $(\int^+ x. \text{norm} (f x) \partial M) = (\int^+ x. \text{ennreal} (f x) + \text{ennreal} (- f x) \partial M)$
by (*intro nn_integral_cong*) (*auto simp: abs_real_def ennreal_neg*)
also have ... = $(\int^+ x. \text{ennreal} (f x) \partial M) + (\int^+ x. \text{ennreal} (- f x) \partial M)$
by (*intro nn_integral_add*) *auto*
also have ... $< \infty$
using *fin* **by** (*auto simp: less_top*)
finally show $(\int^+ x. \text{norm} (f x) \partial M) < \infty$.
qed

lemma *integrableD[dest]*:

assumes *integrable M f*
shows $f \in \text{borel_measurable } M \wedge (\int^+ x. \text{ennreal} (f x) \partial M) \neq \infty \wedge (\int^+ x. \text{ennreal} (- f x) \partial M) \neq \infty$
using *assms* **unfolding** *real_integrable_def* **by** *auto*

lemma *integrableE*:

assumes *integrable M f*
obtains $r \ q$ **where** $0 \leq r \ 0 \leq q$
 $(\int^+ x. \text{ennreal} (f x) \partial M) = \text{ennreal } r$
 $(\int^+ x. \text{ennreal} (- f x) \partial M) = \text{ennreal } q$
 $f \in \text{borel_measurable } M \wedge \text{integral}^L M f = r - q$
using *assms* **unfolding** *real_integrable_def real_lebesgue_integral_def* [*OF assms*]

by (cases rule: ennreal2_cases[of ($\int^+ x. \text{ennreal } (-f x) \partial M$) ($\int^+ x. \text{ennreal } (f x) \partial M$)]) auto

lemma *integral_monotone_convergence_nonneg*:

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $i: \bigwedge i. \text{integrable } M (f i)$ and $\text{mono}: AE x \text{ in } M. \text{mono } (\lambda n. f n x)$

and $\text{pos}: \bigwedge i. AE x \text{ in } M. 0 \leq f i x$

and $\text{lim}: AE x \text{ in } M. (\lambda i. f i x) \longrightarrow u x$

and $\text{ilim}: (\lambda i. \text{integral}^L M (f i)) \longrightarrow x$

and $u: u \in \text{borel_measurable } M$

shows *integrable* $M u$

and *integral*^L $M u = x$

proof –

have $nn: AE x \text{ in } M. \forall i. 0 \leq f i x$

using pos **unfolding** *AE_all_countable* **by** auto

with lim have $u_nn: AE x \text{ in } M. 0 \leq u x$

by *eventually_elim* (auto intro: *LIMSEQ_le_const*)

have [*simp*]: $0 \leq x$

by (intro *LIMSEQ_le_const*[OF *ilim*] *allI exI impI integral_nonneg_AE pos*)

have $(\int^+ x. \text{ennreal } (u x) \partial M) = (\text{SUP } n. (\int^+ x. \text{ennreal } (f n x) \partial M))$

proof (*subst nn_integral_monotone_convergence_SUP_AE*[*symmetric*])

fix i

from $\text{mono } nn$ **show** $AE x \text{ in } M. \text{ennreal } (f i x) \leq \text{ennreal } (f (\text{Suc } i) x)$

by *eventually_elim* (auto *simp*: *mono_def*)

show $(\lambda x. \text{ennreal } (f i x)) \in \text{borel_measurable } M$

using i **by** auto

next

show $(\int^+ x. \text{ennreal } (u x) \partial M) = \int^+ x. (\text{SUP } i. \text{ennreal } (f i x)) \partial M$

proof (*rule nn_integral_cong_AE*)

show $AE x \text{ in } M. \text{ennreal } (u x) = (\text{SUP } i. \text{ennreal } (f i x))$

using *lim mono nn u_nn*

by *eventually_elim* (*simp add*: *LIMSEQ_unique*[OF *LIMSEQ_SUP*]
incseq_def)

qed

qed

also have $\dots = \text{ennreal } x$

using *mono i nn* **unfolding** *nn_integral_eq_integral*[OF i *pos*]

by (*subst LIMSEQ_unique*[OF *LIMSEQ_SUP*]) (auto *simp*: *mono_def integral_nonneg_AE pos intro*!: *integral_mono_AE ilim*)

finally have $(\int^+ x. \text{ennreal } (u x) \partial M) = \text{ennreal } x$.

moreover have $(\int^+ x. \text{ennreal } (-u x) \partial M) = 0$

using $u u_nn$ **by** (*subst nn_integral_0_iff_AE*) (auto *simp*: *ennreal_neg*)

ultimately show *integrable* $M u$ *integral*^L $M u = x$

by (auto *simp*: *real_integrable_def real_lebesgue_integral_def u*)

qed

lemma

fixes $f :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes $f: \bigwedge i. \text{integrable } M (f i)$ and $\text{mono}: AE x \text{ in } M. \text{mono } (\lambda n. f n x)$

```

and lim:  $\text{AE } x \text{ in } M. (\lambda i. f \ i \ x) \longrightarrow u \ x$ 
and ilim:  $(\lambda i. \text{integral}^L M (f \ i)) \longrightarrow x$ 
and u:  $u \in \text{borel\_measurable } M$ 
shows integrable_monotone_convergence: integrable M u
  and integral_monotone_convergence:  $\text{integral}^L M \ u = x$ 
  and has_bochner_integral_monotone_convergence: has_bochner_integral M u
x
proof -
  have 1:  $\bigwedge i. \text{integrable } M (\lambda x. f \ i \ x - f \ 0 \ x)$ 
    using f by auto
  have 2:  $\text{AE } x \text{ in } M. \text{mono } (\lambda n. f \ n \ x - f \ 0 \ x)$ 
    using mono by (auto simp: mono_def le_fun_def)
  have 3:  $\bigwedge n. \text{AE } x \text{ in } M. 0 \leq f \ n \ x - f \ 0 \ x$ 
    using mono by (auto simp: field_simps mono_def le_fun_def)
  have 4:  $\text{AE } x \text{ in } M. (\lambda i. f \ i \ x - f \ 0 \ x) \longrightarrow u \ x - f \ 0 \ x$ 
    using lim by (auto intro!: tendsto_diff)
  have 5:  $(\lambda i. (\int x. f \ i \ x - f \ 0 \ x \ \partial M)) \longrightarrow x - \text{integral}^L M (f \ 0)$ 
    using f ilim by (auto intro!: tendsto_diff)
  have 6:  $(\lambda x. u \ x - f \ 0 \ x) \in \text{borel\_measurable } M$ 
    using f[of 0] u by auto
  note diff = integral_monotone_convergence_nonneg[OF 1 2 3 4 5 6]
  have integrable M  $(\lambda x. (u \ x - f \ 0 \ x) + f \ 0 \ x)$ 
    using diff(1) f by (rule integrable_add)
  with diff(2) f show integrable M u  $\text{integral}^L M \ u = x$ 
    by auto
  then show has_bochner_integral M u x
    by (metis has_bochner_integral_integrable)
qed

```

```

lemma integral_norm_eq_0_iff:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes f[measurable]: integrable M f
  shows  $(\int x. \text{norm } (f \ x) \ \partial M) = 0 \iff \text{emeasure } M \ \{x \in \text{space } M. f \ x \neq 0\} = 0$ 
proof -
  have  $(\int^{+x}. \text{norm } (f \ x) \ \partial M) = (\int x. \text{norm } (f \ x) \ \partial M)$ 
    using f by (intro nn_integral_eq_integral integrable_norm) auto
  then have  $(\int x. \text{norm } (f \ x) \ \partial M) = 0 \iff (\int^{+x}. \text{norm } (f \ x) \ \partial M) = 0$ 
    by simp
  also have  $\dots \iff \text{emeasure } M \ \{x \in \text{space } M. \text{ennreal } (\text{norm } (f \ x)) \neq 0\} = 0$ 
    by (intro nn_integral_0_iff) auto
  finally show ?thesis
    by simp
qed

```

```

lemma integral_0_iff:
  fixes f :: 'a  $\Rightarrow$  real
  shows integrable M f  $\implies (\int x. |f \ x| \ \partial M) = 0 \iff \text{emeasure } M \ \{x \in \text{space } M. f \ x \neq 0\} = 0$ 
    using integral_norm_eq_0_iff[of M f] by simp

```

lemma (in *finite_measure*) *integrable_const*[*intro!*, *simp*]: *integrable* M ($\lambda x. a$)
using *integrable_indicator*[of space M M a] **by** (*simp cong: integrable_cong add: less_top[symmetric]*)

lemma *lebesgue_integral_const*[*simp*]:
fixes $a :: 'a :: \{banach, second_countable_topology\}$
shows $(\int x. a \ \partial M) = \text{measure } M \ (\text{space } M) *_{\mathbb{R}} a$
proof –
{ assume *emeasure* M (*space* M) = ∞ $a \neq 0$
then have *?thesis*
by (*auto simp: not_integrable_integral_eq ennreal_mult_less_top measure_def integrable_iff_bounded*) }
moreover
{ assume $a = 0$ **then have** *?thesis* **by** *simp* }
moreover
{ assume *emeasure* M (*space* M) $\neq \infty$
interpret *finite_measure* M
proof **qed** *fact*
have $(\int x. a \ \partial M) = (\int x. \text{indicator } (\text{space } M) \ x *_{\mathbb{R}} a \ \partial M)$
by (*intro integral_cong auto*)
also have $\dots = \text{measure } M \ (\text{space } M) *_{\mathbb{R}} a$
by (*simp add: less_top[symmetric]*)
finally have *?thesis* . }
ultimately show *?thesis* **by** *blast*
qed

lemma (in *finite_measure*) *integrable_const_bound*:
fixes $f :: 'a \Rightarrow 'b :: \{banach, second_countable_topology\}$
shows *AE* x in $M. \text{norm } (f \ x) \leq B \implies f \in \text{borel_measurable } M \implies \text{integrable } M \ f$
using *integrable_bound*[OF *integrable_const*[of B], of f]
by (*smt (verit, ccfv_SIG) eventually_elim2 real_norm_def*)

lemma (in *finite_measure*) *integral_bounded_eq_bound_then_AE*:
assumes *AE* x in $M. f \ x \leq (c :: \text{real})$
integrable $M \ f$ $(\int x. f \ x \ \partial M) = c * \text{measure } M \ (\text{space } M)$
shows *AE* x in $M. f \ x = c$
using *assms* **by** (*intro integral_ineq_eq_0_then_AE*) *auto*

lemma *integral_indicator_finite_real*:
fixes $f :: 'a \Rightarrow \text{real}$
assumes [*simp*]: *finite* A
assumes [*measurable*]: $\bigwedge a. a \in A \implies \{a\} \in \text{sets } M$
assumes *finite*: $\bigwedge a. a \in A \implies \text{emeasure } M \ \{a\} < \infty$
shows $(\int x. f \ x * \text{indicator } A \ x \ \partial M) = (\sum a \in A. f \ a * \text{measure } M \ \{a\})$
proof –
have $(\int x. f \ x * \text{indicator } A \ x \ \partial M) = (\int x. (\sum a \in A. f \ a * \text{indicator } \{a\} \ x) \ \partial M)$
proof (*intro integral_cong refl*)

```

fix  $x$  show  $f x * \text{indicator } A x = (\sum a \in A. f a * \text{indicator } \{a\} x)$ 
  by (auto split: split_indicator simp: eq_commute[of x] cong: conj_cong)
qed
also have  $\dots = (\sum a \in A. f a * \text{measure } M \{a\})$ 
  using finite by (subst integral_sum) (auto)
finally show ?thesis .
qed

lemma (in finite_measure) ennreal_integral_real:
  assumes [measurable]:  $f \in \text{borel\_measurable } M$ 
  assumes ae:  $A E x \text{ in } M. f x \leq \text{ennreal } B \ 0 \leq B$ 
  shows  $\text{ennreal } (\int x. \text{enn2real } (f x) \ \partial M) = (\int ^+ x. f x \ \partial M)$ 
proof (subst nn_integral_eq_integral[symmetric])
  show integrable  $M (\lambda x. \text{enn2real } (f x))$ 
  using ae by (intro integrable_const_bound[where B=B]) (auto simp: enn2real_leI)
  show  $(\int ^+ x. \text{ennreal } (\text{enn2real } (f x)) \ \partial M) = \text{integral}^N M f$ 
  using ae by (intro nn_integral_cong_AE) (auto simp: le_less_trans[OF _
ennreal_less_top])
qed auto

lemma (in finite_measure) integral_less_AE:
  fixes  $X Y :: 'a \Rightarrow \text{real}$ 
  assumes int: integrable  $M X$  integrable  $M Y$ 
  assumes A:  $(\text{emeasure } M) A \neq 0 \ A \in \text{sets } M \ A E x \text{ in } M. x \in A \longrightarrow X x \neq Y x$ 
  assumes gt:  $A E x \text{ in } M. X x \leq Y x$ 
  shows  $\text{integral}^L M X < \text{integral}^L M Y$ 
proof -
  have  $\text{integral}^L M X \leq \text{integral}^L M Y$ 
  using gt int by (intro integral_mono_AE) auto
moreover
  have  $\text{integral}^L M X \neq \text{integral}^L M Y$ 
proof
  assume eq:  $\text{integral}^L M X = \text{integral}^L M Y$ 
  have  $\text{integral}^L M (\lambda x. |Y x - X x|) = \text{integral}^L M (\lambda x. Y x - X x)$ 
  using gt int by (intro integral_cong_AE) auto
  also have  $\dots = 0$ 
  using eq int by simp
  finally have  $(\text{emeasure } M) \{x \in \text{space } M. Y x - X x \neq 0\} = 0$ 
  using int by (simp add: integral_0_iff)
moreover
  have  $(\int ^+ x. \text{indicator } A x \ \partial M) \leq (\int ^+ x. \text{indicator } \{x \in \text{space } M. Y x - X x \neq 0\} x \ \partial M)$ 
  using A by (intro nn_integral_mono_AE) auto
  then have  $(\text{emeasure } M) A \leq (\text{emeasure } M) \{x \in \text{space } M. Y x - X x \neq 0\}$ 
  using int A by (simp add: integrable_def)
  ultimately have  $\text{emeasure } M A = 0$ 
  by simp
  with  $\langle (\text{emeasure } M) A \neq 0 \rangle$  show False by auto
qed

```

ultimately show ?thesis by auto
qed

lemma (in finite_measure) integral_less_AE_space:
fixes X Y :: 'a \Rightarrow real
assumes int: integrable M X integrable M Y
assumes gt: AE x in M. X x < Y x emeasure M (space M) \neq 0
shows $\text{integral}^L M X < \text{integral}^L M Y$
using gt by (intro integral_less_AE[OF int, where A=space M]) auto

lemma tendsto_integral_at_top:
fixes f :: real \Rightarrow 'a::{banach, second_countable_topology}
assumes [measurable_cong]: sets M = sets borel and f[measurable]: integrable M f
shows $((\lambda y. \int x. \text{indicator } \{.. y\} x *_R f x \partial M) \longrightarrow \int x. f x \partial M)$ at_top
proof (rule tendsto_at_topI_sequentially)
fix X :: nat \Rightarrow real assume filterlim X at_top sequentially
show $(\lambda n. \int x. \text{indicator } \{..X n\} x *_R f x \partial M) \longrightarrow \text{integral}^L M f$
proof (rule integral_dominated_convergence)
show integrable M $(\lambda x. \text{norm } (f x))$
by (rule integrable_norm) fact
show AE x in M. $(\lambda n. \text{indicator } \{..X n\} x *_R f x) \longrightarrow f x$
proof
fix x
from <filterlim X at_top sequentially>
have eventually $(\lambda n. x \leq X n)$ sequentially
unfolding filterlim_at_top_ge[where c=x] by auto
then show $(\lambda n. \text{indicator } \{..X n\} x *_R f x) \longrightarrow f x$
by (intro tendsto_eventually) (auto split: split_indicator elim!: eventually_mono)
qed
fix n show AE x in M. $\text{norm } (\text{indicator } \{..X n\} x *_R f x) \leq \text{norm } (f x)$
by (auto split: split_indicator)
qed auto
qed

lemma
fixes f :: real \Rightarrow real
assumes M: sets M = sets borel
assumes nonneg: AE x in M. $0 \leq f x$
assumes borel: $f \in \text{borel_measurable borel}$
assumes int: $\bigwedge y. \text{integrable } M (\lambda x. f x * \text{indicator } \{.. y\} x)$
assumes conv: $((\lambda y. \int x. f x * \text{indicator } \{.. y\} x \partial M) \longrightarrow x)$ at_top
shows has_bochner_integral_monotone_convergence_at_top: has_bochner_integral M f x
and integrable_monotone_convergence_at_top: integrable M f
and integrable_monotone_convergence_at_top: $\text{integral}^L M f = x$
proof –
from nonneg have AE x in M. mono $(\lambda n::nat. f x * \text{indicator } \{..real n\} x)$

```

  by (auto split: split_indicator intro!: monoI)
  { fix x have eventually ( $\lambda n. f x * indicator \{..real n\} x = f x$ ) sequentially
    by (rule eventually_sequentiallyI[of nat [x]])
      (auto split: split_indicator simp: nat_le_iff ceiling_le_iff) }
  from filterlim_cong[OF refl refl this]
  have AE x in M. ( $\lambda i. f x * indicator \{..real i\} x$ )  $\longrightarrow$  f x
    by simp
  have ( $\lambda i. \int x. f x * indicator \{..real i\} x \partial M$ )  $\longrightarrow$  x
    using conv_filterlim_real_sequentially by (rule filterlim_compose)
  have M_measure[simp]: borel_measurable M = borel_measurable borel
    using M by (simp add: sets_eq_imp_space_eq measurable_def)
  have f  $\in$  borel_measurable M
    using borel by simp
  show has_bochner_integral M f x
    by (rule has_bochner_integral_monotone_convergence) fact+
  then show integrable M f integralL M f = x
    by (auto simp: _has_bochner_integral_iff)
qed

```

7.9.8 Product measure

lemma (in sigma_finite_measure) borel_measurable_lebesgue_integrable[measurable (raw)]:

```

  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_::\{banach, second\_countable\_topology\}$ 
  assumes [measurable]: case_prod f  $\in$  borel_measurable ( $N \otimes_M M$ )
  shows Measurable.pred N ( $\lambda x. integrable M (f x)$ )

```

proof –

```

  have [simp]:  $\bigwedge x. x \in space N \implies integrable M (f x) \longleftrightarrow (\int^+ y. norm (f x y) \partial M) < \infty$ 

```

```

  unfolding integrable_iff_bounded by simp

```

```

  show ?thesis

```

```

  by (simp cong: measurable_cong)

```

qed

lemma (in sigma_finite_measure) measurable_measure[measurable (raw)]:

```

  ( $\bigwedge x. x \in space N \implies A x \subseteq space M$ )  $\implies$ 
  { $x \in space (N \otimes_M M). snd x \in A (fst x)$ }  $\in$  sets ( $N \otimes_M M$ )  $\implies$ 
  ( $\lambda x. measure M (A x)$ )  $\in$  borel_measurable N

```

```

  unfolding measure_def by (intro measurable_emeasure borel_measurable_enn2real)
  auto

```

proposition (in sigma_finite_measure) borel_measurable_lebesgue_integral[measurable (raw)]:

```

  fixes f ::  $\_ \Rightarrow \_ \Rightarrow \_::\{banach, second\_countable\_topology\}$ 
  assumes f[measurable]: case_prod f  $\in$  borel_measurable ( $N \otimes_M M$ )
  shows ( $\lambda x. \int y. f x y \partial M$ )  $\in$  borel_measurable N

```

proof –

```

  from borel_measurable_implies_sequence_metric[OF f, of 0]

```

```

  obtain s where s:  $\bigwedge i. simple\_function (N \otimes_M M) (s i)$ 

```

```

and  $\forall x \in \text{space } (N \otimes_M M). (\lambda i. s\ i\ x) \longrightarrow (\text{case } x \text{ of } (x, y) \Rightarrow f\ x\ y)$ 
and  $\forall i. \forall x \in \text{space } (N \otimes_M M). \text{dist } (s\ i\ x)\ 0 \leq 2 * \text{dist } (\text{case } x \text{ of } (x, xa) \Rightarrow$ 
 $f\ x\ xa)\ 0$ 
by auto
then have *:
 $\bigwedge x\ y. x \in \text{space } N \Rightarrow y \in \text{space } M \Rightarrow (\lambda i. s\ i\ (x, y)) \longrightarrow f\ x\ y$ 
 $\bigwedge i\ x\ y. x \in \text{space } N \Rightarrow y \in \text{space } M \Rightarrow \text{norm } (s\ i\ (x, y)) \leq 2 * \text{norm } (f\ x\ y)$ 
by (auto simp: space_pair_measure)

have [measurable]:  $\bigwedge i. s\ i \in \text{borel\_measurable } (N \otimes_M M)$ 
by (rule borel\_measurable\_simple\_function fact)

have  $\bigwedge i. s\ i \in \text{measurable } (N \otimes_M M)$  (count_space UNIV)
by (rule measurable\_simple\_function fact)

define  $f'$  where [abs_def]:  $f'\ i\ x =$ 
  (if integrable  $M$  ( $f\ x$ ) then simple_bochner_integral  $M$  ( $\lambda y. s\ i\ (x, y)$ ) else  $0$ )
for  $i\ x$ 

{ fix  $i\ x$  assume  $x \in \text{space } N$ 
  then have simple_bochner_integral  $M$  ( $\lambda y. s\ i\ (x, y)$ ) =
    ( $\sum z \in s\ i\ '(\text{space } N \times \text{space } M). \text{measure } M \{y \in \text{space } M. s\ i\ (x, y) = z\}$ 
 $*_R\ z$ )
  using  $s[THEN\ \text{simple\_functionD}(1)]$ 
  unfolding simple_bochner_integral_def
  by (intro sum.mono_neutral_cong_left)
    (auto simp: eq_commute space_pair_measure_image_iff cong: conj_cong)
}
note  $eq = \text{this}$ 

show ?thesis
proof (rule borel\_measurable_LIMSEQ_metric)
  fix  $i$  show  $f'\ i \in \text{borel\_measurable } N$ 
  unfolding  $f'\_def$  by (simp_all add: eq cong: measurable_cong if_cong)
next
  fix  $x$  assume  $x: x \in \text{space } N$ 
  { assume  $int\_f: \text{integrable } M$  ( $f\ x$ )
    have  $int\_2f: \text{integrable } M$  ( $\lambda y. 2 * \text{norm } (f\ x\ y)$ )
      by (intro integrable_norm integrable_mult_right int_f)
    have ( $\lambda i. \text{integral}^L\ M$  ( $\lambda y. s\ i\ (x, y)$ ))  $\longrightarrow \text{integral}^L\ M$  ( $f\ x$ )
      proof (rule integral_dominated_convergence)
        from  $int\_f$  show  $f\ x \in \text{borel\_measurable } M$  by auto
        show  $\bigwedge i. (\lambda y. s\ i\ (x, y)) \in \text{borel\_measurable } M$ 
          using  $x$  by simp
        show AE  $xa$  in  $M. (\lambda i. s\ i\ (x, xa)) \longrightarrow f\ x\ xa$ 
          using  $x *$  by auto
        show  $\bigwedge i. \text{AE } xa \text{ in } M. \text{norm } (s\ i\ (x, xa)) \leq 2 * \text{norm } (f\ x\ xa)$ 
          using  $x *$  by auto
      qed fact
  }

```



```

moreover
{ fix  $i$ 
  have  $\text{simple\_bochner\_integrable } M (\lambda y. s\ i\ (x, y))$ 
  proof ( $\text{rule simple\_bochner\_integrableI\_bounded}$ )
    have  $(\lambda y. s\ i\ (x, y)) \text{ ' space } M \subseteq s\ i\ \text{' (space } N \times \text{space } M)$ 
    using  $x$  by  $\text{auto}$ 
    then show  $\text{simple\_function } M (\lambda y. s\ i\ (x, y))$ 
    using  $\text{simple\_functionD}(1)[OF\ s(1),\ of\ i]\ x$ 
    by ( $\text{intro simple\_function\_borel\_measurable}$ )
      ( $\text{auto simp: space\_pair\_measure dest: finite\_subset}$ )
    have  $(\int^+ y. \text{ennreal } (\text{norm } (s\ i\ (x, y))))\ \partial M \leq (\int^+ y. 2 * \text{norm } (f\ x\ y))\ \partial M$ 
 $\partial M$ )
    using  $x$  by ( $\text{intro nn\_integral\_mono}$ )  $\text{auto}$ 
    also have  $(\int^+ y. 2 * \text{norm } (f\ x\ y))\ \partial M < \infty$ 
    using  $\text{int\_2f\_unfolding\_integrable\_iff\_bounded by simp}$ 
    finally show  $(\int^+ xa. \text{ennreal } (\text{norm } (s\ i\ (x, xa))))\ \partial M < \infty$  .
  qed
  then have  $\text{integral}^L\ M (\lambda y. s\ i\ (x, y)) = \text{simple\_bochner\_integral } M (\lambda y.$ 
 $s\ i\ (x, y))$ 
  by ( $\text{rule simple\_bochner\_integrable\_eq\_integral[symmetric] }$ )
  ultimately have  $(\lambda i. \text{simple\_bochner\_integral } M (\lambda y. s\ i\ (x, y))) \longrightarrow$ 
 $\text{integral}^L\ M (f\ x)$ 
  by  $\text{simp }$ 
  then
  show  $(\lambda i. f'\ i\ x) \longrightarrow \text{integral}^L\ M (f\ x)$ 
  unfolding  $f'\ \text{def}$ 
  by ( $\text{cases integrable } M (f\ x)$ ) ( $\text{simp\_all add: not\_integrable\_integral\_eq}$ )
  qed
qed

```

```

lemma (in  $\text{pair\_sigma\_finite}$ )  $\text{integrable\_product\_swap}$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $\text{integrable } (M1 \otimes_M M2)\ f$ 
  shows  $\text{integrable } (M2 \otimes_M M1)\ (\lambda(x,y). f\ (y,x))$ 
  by ( $\text{smt (verit) assms distr\_pair\_swap integrable\_cong integrable\_distr measur-$ 
 $\text{able\_pair\_swap' prod.case\_distrib split\_cong}$ )

```

```

lemma (in  $\text{pair\_sigma\_finite}$ )  $\text{integrable\_product\_swap\_iff}$ :
  fixes  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  shows  $\text{integrable } (M2 \otimes_M M1)\ (\lambda(x,y). f\ (y,x)) \longleftrightarrow \text{integrable } (M1 \otimes_M M2)$ 
 $f$ 
proof -
  interpret  $Q: \text{pair\_sigma\_finite } M2\ M1 \ ..$ 
  from  $Q.\text{integrable\_product\_swap}[of\ \lambda(x,y). f\ (y,x)]\ \text{integrable\_product\_swap}[of\ f]$ 
  show  $?thesis$  by  $\text{auto}$ 
qed

```

```

lemma (in  $\text{pair\_sigma\_finite}$ )  $\text{integral\_product\_swap}$ :

```

fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $f: f \in \text{borel_measurable } (M1 \otimes_M M2)$
shows $(\int (x,y). f (y,x) \partial(M2 \otimes_M M1)) = \text{integral}^L (M1 \otimes_M M2) f$
by (*smt* (*verit*) *distr_pair_swap* *f* *integral_cong* *integral_distr* *measurable_pair_swap'* *prod.case_distrib* *split_cong*)

theorem (*in pair_sigma_finite*) *Fubini_integrable*:

fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $f[\text{measurable}]: f \in \text{borel_measurable } (M1 \otimes_M M2)$
and $\text{integ1}: \text{integrable } M1 (\lambda x. \int y. \text{norm } (f (x, y)) \partial M2)$
and $\text{integ2}: \text{AE } x \text{ in } M1. \text{integrable } M2 (\lambda y. f (x, y))$
shows $\text{integrable } (M1 \otimes_M M2) f$
proof (*rule* *integrableI_bounded*)
have $(\int^+ p. \text{norm } (f p) \partial(M1 \otimes_M M2)) = (\int^+ x. (\int^+ y. \text{norm } (f (x, y)) \partial M2) \partial M1)$
by (*simp* *add: M2.nn_integral_fst* [*symmetric*])
also have $\dots = (\int^+ x. |\int y. \text{norm } (f (x, y)) \partial M2| \partial M1)$
apply (*intro* *nn_integral_cong_AE*)
using *integ2*
proof *eventually_elim*
fix x **assume** $\text{integrable } M2 (\lambda y. f (x, y))$
then have $f: \text{integrable } M2 (\lambda y. \text{norm } (f (x, y)))$
by *simp*
then have $(\int^+ y. \text{ennreal } (\text{norm } (f (x, y))) \partial M2) = \text{ennreal } (LINT y | M2. \text{norm } (f (x, y)))$
by (*rule* *nn_integral_eq_integral*) *simp*
also have $\dots = \text{ennreal } |LINT y | M2. \text{norm } (f (x, y))|$
using f **by** *simp*
finally show $(\int^+ y. \text{ennreal } (\text{norm } (f (x, y))) \partial M2) = \text{ennreal } |LINT y | M2. \text{norm } (f (x, y))|$
qed
also have $\dots < \infty$
using *integ1* **by** (*simp* *add: integrable_iff_bounded* *integral_nonneg_AE*)
finally show $(\int^+ p. \text{norm } (f p) \partial(M1 \otimes_M M2)) < \infty$
qed *fact*

lemma (*in pair_sigma_finite*) *emeasure_pair_measure_finite*:

assumes $A: A \in \text{sets } (M1 \otimes_M M2)$ **and** *finite*: $\text{emeasure } (M1 \otimes_M M2) A < \infty$
shows $\text{AE } x \text{ in } M1. \text{emeasure } M2 \{y \in \text{space } M2. (x, y) \in A\} < \infty$
proof –
from $M2. \text{emeasure_pair_measure_alt}[OF A]$ *finite*
have $(\int^+ x. \text{emeasure } M2 (\text{Pair } x - ' A) \partial M1) \neq \infty$
by *simp*
then have $\text{AE } x \text{ in } M1. \text{emeasure } M2 (\text{Pair } x - ' A) \neq \infty$
by (*rule* *nn_integral_PInf_AE*[*rotated*]) (*intro* $M2. \text{measurable_emeasure_Pair } A$)
moreover have $\bigwedge x. x \in \text{space } M1 \implies \text{Pair } x - ' A = \{y \in \text{space } M2. (x, y) \in A\}$

using *sets.sets_into_space*[*OF A*] **by** (*auto simp: space_pair_measure*)
ultimately show *?thesis* **by** (*auto simp: less_top*)
qed

lemma (*in pair_sigma_finite*) *AE_integrable_fst'*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $f[\text{measurable}]$: *integrable* ($M1 \otimes_M M2$) f
shows *AE* x *in* $M1$. *integrable* $M2$ ($\lambda y. f(x, y)$)
proof –
have ($\int^+ x. (\int^+ y. \text{norm}(f(x, y)) \partial M2) \partial M1$) = ($\int^+ x. \text{norm}(f x) \partial(M1 \otimes_M M2)$)
by (*rule* $M2$.*nn_integral_fst*) *simp*
also have ($\int^+ x. \text{norm}(f x) \partial(M1 \otimes_M M2)$) $\neq \infty$
using f **unfolding** *integrable_iff_bounded* **by** *simp*
finally have *AE* x *in* $M1$. ($\int^+ y. \text{norm}(f(x, y)) \partial M2$) $\neq \infty$
by (*intro* $nn_integral_PInf_AE$ $M2$.*borel_measurable_nn_integral*)
(*auto simp: measurable_split_conv*)
with *AE_space* **show** *?thesis*
by *eventually_elim*
(*auto simp: integrable_iff_bounded measurable_compose*[*OF* $_$ *borel_measurable_integrable*[*OF*
 f]] *less_top*)
qed

lemma (*in pair_sigma_finite*) *integrable_fst'*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $f[\text{measurable}]$: *integrable* ($M1 \otimes_M M2$) f
shows *integrable* $M1$ ($\lambda x. \int y. f(x, y) \partial M2$)
unfolding *integrable_iff_bounded*
proof
show ($\lambda x. \int y. f(x, y) \partial M2$) \in *borel_measurable* $M1$
by (*rule* $M2$.*borel_measurable_lebesgue_integral*) *simp*
have ($\int^+ x. \text{ennreal}(\text{norm}(\int y. f(x, y) \partial M2)) \partial M1$) \leq ($\int^+ x. (\int^+ y. \text{norm}(f(x, y)) \partial M2) \partial M1$)
using *AE_integrable_fst'*[*OF f*] **by** (*auto intro!*: *nn_integral_mono_AE_integral_norm_bound_ennreal*)
also have ($\int^+ x. (\int^+ y. \text{norm}(f(x, y)) \partial M2) \partial M1$) = ($\int^+ x. \text{norm}(f x) \partial(M1 \otimes_M M2)$)
by (*rule* $M2$.*nn_integral_fst*) *simp*
also have ($\int^+ x. \text{norm}(f x) \partial(M1 \otimes_M M2)$) $< \infty$
using f **unfolding** *integrable_iff_bounded* **by** *simp*
finally show ($\int^+ x. \text{ennreal}(\text{norm}(\int y. f(x, y) \partial M2)) \partial M1$) $< \infty$.
qed

proposition (*in pair_sigma_finite*) *integral_fst'*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes f : *integrable* ($M1 \otimes_M M2$) f
shows ($\int x. (\int y. f(x, y) \partial M2) \partial M1$) = *integral* ^{L} ($M1 \otimes_M M2$) f
using f **proof** *induct*
case (*base A c*)

```

have A[measurable]: A ∈ sets (M1 ⊗M M2) by fact

have eq: ⋀ x y. x ∈ space M1 ⇒ indicator A (x, y) = indicator {y ∈ space M2.
(x, y) ∈ A} y
using sets.sets_into_space[OF A] by (auto split: split_indicator simp: space_pair_measure)

have int_A: integrable (M1 ⊗M M2) (indicator A :: _ ⇒ real)
using base by (rule integrable_real_indicator)

have (∫ x. ∫ y. indicator A (x, y) *R c ∂M2 ∂M1) = (∫ x. measure M2 {y ∈ space
M2. (x, y) ∈ A} *R c ∂M1)
proof (intro integral_cong_AE, simp, simp)
from AE_integrable_fst'[OF int_A] AE_space
show AE x in M1. (∫ y. indicator A (x, y) *R c ∂M2) = measure M2 {y ∈ space
M2. (x, y) ∈ A} *R c
by eventually_elim (simp add: eq_integrable_indicator_iff)
qed
also have ... = measure (M1 ⊗M M2) A *R c
proof (subst integral_scaleR_left)
have (∫+x. ennreal (measure M2 {y ∈ space M2. (x, y) ∈ A}) ∂M1) =
(∫+x. emeasure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1)
using emeasure_pair_measure_finite[OF base]
by (intro nn_integral_cong_AE, eventually_elim) (simp add: emeasure_eq_ennreal_measure)
also have ... = emeasure (M1 ⊗M M2) A
using sets.sets_into_space[OF A]
by (subst M2.emeasure_pair_measure_alt)
(auto intro!: nn_integral_cong arg_cong[where f=emeasure M2] simp:
space_pair_measure)
finally have *: (∫+x. ennreal (measure M2 {y ∈ space M2. (x, y) ∈ A}) ∂M1)
= emeasure (M1 ⊗M M2) A .

from base * show integrable M1 (λx. measure M2 {y ∈ space M2. (x, y) ∈
A})
by (simp add: integrable_iff_bounded)
then have (∫ x. measure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1) =
(∫+x. ennreal (measure M2 {y ∈ space M2. (x, y) ∈ A}) ∂M1)
by (rule nn_integral_eq_integral[symmetric]) simp
also note *
finally show (∫ x. measure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1) *R c =
measure (M1 ⊗M M2) A *R c
using base by (simp add: emeasure_eq_ennreal_measure)
qed
also have ... = (∫ a. indicator A a *R c ∂(M1 ⊗M M2))
using base by simp
finally show ?case .
next
case (add f g)
then have [measurable]: f ∈ borel_measurable (M1 ⊗M M2) g ∈ borel_measurable
(M1 ⊗M M2)

```

```

  by auto
  have AE x in M1. LINT y|M2. f (x, y) + g (x, y) =
    (LINT y|M2. f (x, y)) + (LINT y|M2. g (x, y))
  using AE_integrable_fst'[OF add(1)] AE_integrable_fst'[OF add(3)]
  by eventually_elim simp
  then have (∫ x. ∫ y. f (x, y) + g (x, y) ∂M2 ∂M1) =
    (∫ x. (∫ y. f (x, y) ∂M2) + (∫ y. g (x, y) ∂M2) ∂M1)
  by (intro integral_cong_AE) auto
  also have ... = (∫ x. f x ∂(M1 ⊗M M2)) + (∫ x. g x ∂(M1 ⊗M M2))
  using integrable_fst'[OF add(1)] integrable_fst'[OF add(3)] add(2,4) by simp
  finally show ?case
    using add by simp
next
  case (lim f s)
  then have [measurable]: f ∈ borel_measurable (M1 ⊗M M2) ∧ i. s i ∈ borel_measurable
  (M1 ⊗M M2)
  by auto

  show ?case
  proof (rule LIMSEQ_unique)
    show (λi. integralL (M1 ⊗M M2) (s i)) ⟶ integralL (M1 ⊗M M2) f
  proof (rule integral_dominated_convergence)
    show integrable (M1 ⊗M M2) (λx. 2 * norm (f x))
    using lim(5) by auto
  qed (insert lim, auto)
  have (λi. ∫ x. ∫ y. s i (x, y) ∂M2 ∂M1) ⟶ ∫ x. ∫ y. f (x, y) ∂M2 ∂M1
  proof (rule integral_dominated_convergence)
    have AE x in M1. ∀ i. integrable M2 (λy. s i (x, y))
    unfolding AE_all_countable using AE_integrable_fst'[OF lim(1)] ..
    with AE_space AE_integrable_fst'[OF lim(5)]
    show AE x in M1. (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
  proof eventually_elim
    fix x assume x: x ∈ space M1 and
      s: ∀ i. integrable M2 (λy. s i (x, y)) and f: integrable M2 (λy. f (x, y))
    show (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
  proof (rule integral_dominated_convergence)
    show integrable M2 (λy. 2 * norm (f (x, y)))
    using f by auto
    show AE xa in M2. (λi. s i (x, xa)) ⟶ f (x, xa)
    using x lim(3) by (auto simp: space_pair_measure)
    show ∧ i. AE xa in M2. norm (s i (x, xa)) ≤ 2 * norm (f (x, xa))
    using x lim(4) by (auto simp: space_pair_measure)
  qed (insert x, measurable)
  qed
  show integrable M1 (λx. (∫ y. 2 * norm (f (x, y)) ∂M2))
  by (intro integrable_mult_right integrable_norm integrable_fst' lim)
  fix i show AE x in M1. norm (∫ y. s i (x, y) ∂M2) ≤ (∫ y. 2 * norm (f
  (x, y)) ∂M2)
  using AE_space AE_integrable_fst'[OF lim(1), of i] AE_integrable_fst'[OF

```

```

lim(5)]
  proof eventually_elim
    fix x assume x: x ∈ space M1
    and s: integrable M2 (λy. s i (x, y)) and f: integrable M2 (λy. f (x, y))
    from s have norm (∫ y. s i (x, y) ∂M2) ≤ (∫ +y. norm (s i (x, y)) ∂M2)
    by (rule integral_norm_bound_ennreal)
    also have ... ≤ (∫ +y. 2 * norm (f (x, y)) ∂M2)
    using x lim by (auto intro!: nn_integral_mono simp: space_pair_measure)
    also have ... = (∫ y. 2 * norm (f (x, y)) ∂M2)
    using f by (intro nn_integral_eq_integral) auto
    finally show norm (∫ y. s i (x, y) ∂M2) ≤ (∫ y. 2 * norm (f (x, y))
∂M2)
      by simp
    qed
  qed simp_all
  then show (λi. integralL (M1 ⊗M M2) (s i)) → ∫ x. ∫ y. f (x, y) ∂M2
∂M1
    using lim by simp
  qed
qed

```

```

lemma (in pair_sigma_finite)
  fixes f :: _ ⇒ _ ⇒ _ :: {banach, second_countable_topology}
  assumes f: integrable (M1 ⊗M M2) (case_prod f)
  shows AE_integrable_fst: AE x in M1. integrable M2 (λy. f x y) (is ?AE)
    and integrable_fst: integrable M1 (λx. ∫ y. f x y ∂M2) (is ?INT)
    and integral_fst: (∫ x. (∫ y. f x y ∂M2) ∂M1) = integralL (M1 ⊗M M2) (λ(x,
y). f x y) (is ?EQ)
  using AE_integrable_fst'[OF f] integrable_fst'[OF f] integral_fst'[OF f] by auto

```

```

lemma (in pair_sigma_finite)
  fixes f :: _ ⇒ _ ⇒ _ :: {banach, second_countable_topology}
  assumes f[measurable]: integrable (M1 ⊗M M2) (case_prod f)
  shows AE_integrable_snd: AE y in M2. integrable M1 (λx. f x y) (is ?AE)
    and integrable_snd: integrable M2 (λy. ∫ x. f x y ∂M1) (is ?INT)
    and integral_snd: (∫ y. (∫ x. f x y ∂M1) ∂M2) = integralL (M1 ⊗M M2)
(case_prod f) (is ?EQ)

```

```

proof -
  interpret Q: pair_sigma_finite M2 M1 ..
  have Q_int: integrable (M2 ⊗M M1) (λ(x, y). f y x)
    using f unfolding integrable_product_swap_iff[symmetric] by simp
  show ?AE using Q.AE_integrable_fst'[OF Q_int] by simp
  show ?INT using Q.integrable_fst'[OF Q_int] by simp
  show ?EQ using Q.integral_fst'[OF Q_int]
    using integral_product_swap[of case_prod f] by simp
  qed

```

```

proposition (in pair_sigma_finite) Fubini_integral:
  fixes f :: _ ⇒ _ ⇒ _ :: {banach, second_countable_topology}

```

assumes f : integrable $(M1 \otimes_M M2)$ (case_prod f)
 shows $(\int y. (\int x. f\ x\ y\ \partial M1)\ \partial M2) = (\int x. (\int y. f\ x\ y\ \partial M2)\ \partial M1)$
 unfolding integral_snd[OF assms] integral_fst[OF assms] ..

lemma (in product_sigma_finite) product_integral_singleton:
 fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
 shows $f \in \text{borel_measurable } (M\ i) \implies (\int x. f\ (x\ i)\ \partial Pi_M\ \{i\}\ M) = \text{integral}^L$
 $(M\ i)\ f$
 by (metis (no_types) distr_singleton insert_iff integral_distr measurable_component_singleton)

lemma (in product_sigma_finite) product_integral_fold:
 fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
 assumes $IJ[\text{simp}]: I \cap J = \{\}$ and $\text{fin}: \text{finite } I\ \text{finite } J$
 and f : integrable $(Pi_M\ (I \cup J)\ M)\ f$
 shows $\text{integral}^L\ (Pi_M\ (I \cup J)\ M)\ f = (\int x. (\int y. f\ (\text{merge } I\ J\ (x, y))\ \partial Pi_M\ J$
 $M)\ \partial Pi_M\ I\ M)$
 proof -
 interpret I : finite_product_sigma_finite $M\ I$ by standard fact
 interpret J : finite_product_sigma_finite $M\ J$ by standard fact
 have finite $(I \cup J)$ using fin by auto
 interpret IJ : finite_product_sigma_finite $M\ I \cup J$ by standard fact
 interpret P : pair_sigma_finite $Pi_M\ I\ M\ Pi_M\ J\ M$..
 let $?M = \text{merge } I\ J$
 let $?f = \lambda x. f\ (?M\ x)$
 from f have $f_borel: f \in \text{borel_measurable } (Pi_M\ (I \cup J)\ M)$
 by auto
 have $P_borel: (\lambda x. f\ (\text{merge } I\ J\ x)) \in \text{borel_measurable } (Pi_M\ I\ M \otimes_M Pi_M\ J$
 $M)$
 using measurable_comp[OF measurable_merge f_borel] by (simp add: comp_def)
 have $f_int: \text{integrable } (Pi_M\ I\ M \otimes_M Pi_M\ J\ M)\ ?f$
 by (rule integrable_distr[OF measurable_merge]) (simp add: distr_merge[OF
 $IJ\ \text{fin}]\ f)$
 have $LINT\ x|(Pi_M\ I\ M \otimes_M Pi_M\ J\ M). f\ (\text{merge } I\ J\ x) =$
 $LINT\ x|Pi_M\ I\ M. LINT\ y|Pi_M\ J\ M. f\ (\text{merge } I\ J\ (x, y))$
 by (simp add: $P.\text{integral_fst}'[\text{symmetric, OF } f_int]$)
 then show $?thesis$
 apply (subst distr_merge[symmetric, OF $IJ\ \text{fin}$])
 by (simp add: integral_distr[OF measurable_merge f_borel])
 qed

lemma (in product_sigma_finite) product_integral_insert:
 fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
 assumes I : finite $I\ i \notin I$
 and f : integrable $(Pi_M\ (\text{insert } i\ I)\ M)\ f$
 shows $\text{integral}^L\ (Pi_M\ (\text{insert } i\ I)\ M)\ f = (\int x. (\int y. f\ (x(i:=y))\ \partial M\ i)\ \partial Pi_M\ I$
 $M)$
 proof -
 have $\text{integral}^L\ (Pi_M\ (\text{insert } i\ I)\ M)\ f = \text{integral}^L\ (Pi_M\ (I \cup \{i\})\ M)\ f$
 by simp

```

also have ... = (∫ x. (∫ y. f (merge I {i} (x,y)) ∂Pi_M {i} M) ∂Pi_M I M)
  using f I by (intro product_integral_fold) auto
also have ... = (∫ x. (∫ y. f (x(i := y)) ∂M i) ∂Pi_M I M)
proof (rule integral_cong[OF refl], subst product_integral_singleton[symmetric])
  fix x assume x: x ∈ space (Pi_M I M)
  have f_borel: f ∈ borel_measurable (Pi_M (insert i I) M)
    using f by auto
  show (λy. f (x(i := y))) ∈ borel_measurable (M i)
    using measurable_comp[OF measurable_component_update f_borel, OF x ⟨i
  ∉ I⟩]
  unfolding comp_def .
  from x I show (∫ y. f (merge I {i} (x,y)) ∂Pi_M {i} M) = (∫ xa. f (x(i :=
  xa i)) ∂Pi_M {i} M)
  by (auto intro!: integral_cong arg_cong[where f=f] simp: merge_def space_PiM
  extensional_def PiE_def)
qed
finally show ?thesis .
qed

```

```

lemma (in product_sigma_finite) product_integrable_prod:
  fixes f :: 'i ⇒ 'a ⇒ _ :: {real_normed_field,banach,second_countable_topology}
  assumes [simp]: finite I and integrable: ∧i. i ∈ I ⇒ integrable (M i) (f i)
  shows integrable (Pi_M I M) (λx. (∏ i∈I. f i (x i))) (is_integrable _ ?f)
proof (unfold integrable_iff_bounded, intro conjI)
  interpret finite_product_sigma_finite M I by standard fact

```

```

  show ?f ∈ borel_measurable (Pi_M I M)
    using assms by simp
  have (∫+ x. ennreal (norm (∏ i∈I. f i (x i))) ∂Pi_M I M) =
    (∫+ x. (∏ i∈I. ennreal (norm (f i (x i)))) ∂Pi_M I M)
    by (simp add: prod_norm_prod_ennreal)
  also have ... = (∏ i∈I. ∫+ x. ennreal (norm (f i x)) ∂M i)
    using assms by (intro product_nn_integral_prod) auto
  also have ... < ∞
    using integrable by (simp add: less_top[symmetric] ennreal_prod_eq_top in-
  tegrable_iff_bounded)
  finally show (∫+ x. ennreal (norm (∏ i∈I. f i (x i))) ∂Pi_M I M) < ∞ .
qed

```

```

lemma (in product_sigma_finite) product_integral_prod:
  fixes f :: 'i ⇒ 'a ⇒ _ :: {real_normed_field,banach,second_countable_topology}
  assumes finite I and integrable: ∧i. i ∈ I ⇒ integrable (M i) (f i)
  shows (∫ x. (∏ i∈I. f i (x i)) ∂Pi_M I M) = (∏ i∈I. integralL (M i) (f i))
using assms proof induct
  case empty
  interpret finite_measure Pi_M {} M
  by rule (simp add: space_PiM)
  show ?case by (simp add: space_PiM measure_def)
next

```



```

case (insert i I)
then have iI: finite (insert i I) by auto
then have prod:  $\bigwedge J. J \subseteq \text{insert } i \ I \implies$ 
  integrable (Pi_M J M) ( $\lambda x. (\prod_{i \in J}. f \ i \ (x \ i))$ )
  by (intro product_integrable_prod insert(4)) (auto intro: finite_subset)
interpret I: finite_product_sigma_finite M I by standard fact
have *:  $\bigwedge x \ y. (\prod_{j \in I}. f \ j \ (\text{if } j = i \ \text{then } y \ \text{else } x \ j)) = (\prod_{j \in I}. f \ j \ (x \ j))$ 
  using <i  $\notin$  I> by (auto intro!: prod.cong)
show ?case
  unfolding product_integral_insert[OF insert(1,2) prod[OF subset_refl]]
  by (simp add: * insert prod subset_insertI)
qed

```

```

lemma integrable_subalgebra:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes borel: f  $\in$  borel_measurable N
  and N: sets N  $\subseteq$  sets M space N = space M  $\bigwedge$  A. A  $\in$  sets N  $\implies$  emeasure N
  A = emeasure M A
  shows integrable N f  $\longleftrightarrow$  integrable M f (is ?P)
proof -
  have f  $\in$  borel_measurable M
  using assms by (auto simp: measurable_def)
  with assms show ?thesis
  using assms by (auto simp: integrable_iff_bounded nn_integral_subalgebra)
qed

```

```

lemma integral_subalgebra:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes borel: f  $\in$  borel_measurable N
  and N: sets N  $\subseteq$  sets M space N = space M  $\bigwedge$  A. A  $\in$  sets N  $\implies$  emeasure N
  A = emeasure M A
  shows integralL N f = integralL M f
proof cases
  assume integrable N f
  then show ?thesis
  proof induct
    case base with assms show ?case by (auto simp: subset_eq measure_def)
  next
    case (add f g)
    then have ( $\int a. f \ a + g \ a \ \partial N$ ) = integralL M f + integralL M g
      by simp
    also have ... = ( $\int a. f \ a + g \ a \ \partial M$ )
      using add_integrable_subalgebra[OF _ N, of f] integrable_subalgebra[OF _
  N, of g] by simp
    finally show ?case .
  next
    case (lim f s)
    then have M:  $\bigwedge i. \text{integrable } M \ (s \ i) \ \text{integrable } M \ f$ 
      using integrable_subalgebra[OF _ N, of f] integrable_subalgebra[OF _ N, of

```

2430

```
s i for i] by simp_all
  show ?case
  proof (intro LIMSEQ_unique)
    show  $(\lambda i. \text{integral}^L N (s i)) \longrightarrow \text{integral}^L N f$ 
      apply (rule integral_dominated_convergence[where  $w = \lambda x. 2 * \text{norm } (f$ 
 $x)$ ])
    using lim by auto
    show  $(\lambda i. \text{integral}^L N (s i)) \longrightarrow \text{integral}^L M f$ 
      unfolding lim
      apply (rule integral_dominated_convergence[where  $w = \lambda x. 2 * \text{norm } (f$ 
 $x)$ ])
    using lim M N by auto
  qed
  qed
qed (simp add: not_integrable_integral_eq integrable_subalgebra[OF assms])

hide_const (open) simple_bochner_integral
hide_const (open) simple_bochner_integrable

end
```

7.10 Complete Measures

```
theory Complete_Measure
  imports Bochner_Integration
begin

locale complete_measure =
  fixes M :: 'a measure
  assumes complete:  $\bigwedge A B. B \subseteq A \implies A \in \text{null\_sets } M \implies B \in \text{sets } M$ 

definition
  split_completion M A p = (if A  $\in$  sets M then p = (A, { }) else
     $\exists N'. A = \text{fst } p \cup \text{snd } p \wedge \text{fst } p \cap \text{snd } p = \{ \} \wedge \text{fst } p \in \text{sets } M \wedge \text{snd } p \subseteq N' \wedge$ 
 $N' \in \text{null\_sets } M$ )

definition
  main_part M A = fst (Eps (split_completion M A))

definition
  null_part M A = snd (Eps (split_completion M A))

definition completion :: 'a measure  $\Rightarrow$  'a measure where
  completion M = measure_of (space M) { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$ 
 $\text{null\_sets } M \wedge N \subseteq N' \}$ 
  (emeasure M  $\circ$  main_part M)

lemma completion_into_space:
  { S  $\cup$  N | S N N'. S  $\in$  sets M  $\wedge$  N'  $\in$  null_sets M  $\wedge$  N  $\subseteq$  N' }  $\subseteq$  Pow (space
```

M)

using *sets.sets_into_space* **by** *auto*

lemma *space_completion[simp]*: *space (completion M) = space M*

unfolding *completion_def* **using** *space_measure_of[OF completion_into_space]*
by *simp*

lemma *completionI*:

assumes $A = S \cup N$ $N \subseteq N'$ $N' \in \text{null_sets } M$ $S \in \text{sets } M$

shows $A \in \{ S \cup N \mid S \in \text{sets } M \wedge N' \in \text{null_sets } M \wedge N \subseteq N' \}$

using *assms* **by** *auto*

lemma *completionE*:

assumes $A \in \{ S \cup N \mid S \in \text{sets } M \wedge N' \in \text{null_sets } M \wedge N \subseteq N' \}$

obtains S N N' **where** $A = S \cup N$ $N \subseteq N'$ $N' \in \text{null_sets } M$ $S \in \text{sets } M$

using *assms* **by** *auto*

lemma *sigma_algebra_completion*:

sigma_algebra (space M) { S \cup N \mid S \in sets M \wedge N' \in null_sets M \wedge N \subseteq N' }

(is sigma_algebra _ ?A)

unfolding *sigma_algebra_iff2*

proof (*intro conjI ballI allI impI*)

show $?A \subseteq \text{Pow (space M)}$

using *sets.sets_into_space* **by** *auto*

next

show $\{ \} \in ?A$ **by** *auto*

next

let $?C = \text{space } M$

fix A **assume** $A \in ?A$

then obtain S N N'

where $A = S \cup N$ $N \subseteq N'$ $N' \in \text{null_sets } M$ $S \in \text{sets } M$

by (*rule completionE*)

then show $\text{space } M - A \in ?A$

by (*intro completionI[of _ (?C - S) \cap (?C - N') (?C - S) \cap N' \cap (?C - N)]*) *auto*

next

fix $A :: \text{nat} \Rightarrow 'a$ **set** **assume** $A: \text{range } A \subseteq ?A$

then have $\forall n. \exists S$ N $N'. A \ n = S \cup N$ $\wedge S \in \text{sets } M$ $\wedge N' \in \text{null_sets } M$ $\wedge N \subseteq N'$

by (*auto simp: image_subset_iff*)

then obtain S N N' **where** $\forall x. A \ x = S \cup N$ $\wedge S \in \text{sets } M$ $\wedge N' \in \text{null_sets } M$ $\wedge N \subseteq N'$

by *metis*

then show $\bigcup (A \text{ ' UNIV}) \in ?A$

using *null_sets_UN[of N]*

by (*intro completionI[of _ \bigcup (S \text{ ' UNIV}) \bigcup (N \text{ ' UNIV}) \bigcup (N' \text{ ' UNIV})]*) *auto*

qed

lemma *sets_completion*:

$sets (completion M) = \{ S \cup N \mid S N N'. S \in sets M \wedge N' \in null_sets M \wedge N \subseteq N' \}$
using *sigma_algebra.sets_measure_of_eq*[*OF sigma_algebra_completion*]
by (*simp add: completion_def*)

lemma *sets_completionE*:

assumes $A \in sets (completion M)$
obtains $S N N'$ **where** $A = S \cup N N \subseteq N' N' \in null_sets M S \in sets M$
using *assms unfolding sets_completion by auto*

lemma *sets_completionI*:

assumes $A = S \cup N N \subseteq N' N' \in null_sets M S \in sets M$
shows $A \in sets (completion M)$
using *assms unfolding sets_completion by auto*

lemma *sets_completionI_sets*[*intro, simp*]:

$A \in sets M \implies A \in sets (completion M)$
unfolding *sets_completion by force*

lemma *measurable_completion*: $f \in M \rightarrow_M N \implies f \in completion M \rightarrow_M N$

by (*auto simp: measurable_def*)

lemma *null_sets_completion*:

assumes $N' \in null_sets M N \subseteq N'$ **shows** $N \in sets (completion M)$
using *assms by (intro sets_completionI[of N {} N N']) auto*

lemma *split_completion*:

assumes $A \in sets (completion M)$
shows *split_completion* $M A (main_part M A, null_part M A)$

proof *cases*

assume $A \in sets M$ **then show** *?thesis*

by (*simp add: split_completion_def[abs_def] main_part_def null_part_def*)

next

assume $nA: A \notin sets M$

show *?thesis*

unfolding *main_part_def null_part_def if_not_P*[*OF nA*]

proof (*rule someI2_ex*)

from *assms obtain* $S N N'$ **where** $A = S \cup N N \subseteq N' N' \in null_sets M$
 $S \in sets M$

by (*blast intro: sets_completionE*)

let $?P = (S, N - S)$

show $\exists p. split_completion M A p$

unfolding *split_completion_def if_not_P*[*OF nA*] **using** A

proof (*intro exI conjI*)

show $A = fst ?P \cup snd ?P$ **using** A **by** *auto*

show $snd ?P \subseteq N'$ **using** A **by** *auto*

qed *auto*

qed *auto*

qed

lemma *sets_restrict_space_subset*:
assumes $S: S \in \text{sets } (\text{completion } M)$
shows $\text{sets } (\text{restrict_space } (\text{completion } M) S) \subseteq \text{sets } (\text{completion } M)$
by (*metis* *assms* *sets.Int_space_eq2* *sets_restrict_space_iff* *subsetI*)

lemma
assumes $S \in \text{sets } (\text{completion } M)$
shows *main_part_sets*[*intro*, *simp*]: $\text{main_part } M S \in \text{sets } M$
and *main_part_null_part_Un*[*simp*]: $\text{main_part } M S \cup \text{null_part } M S = S$
and *main_part_null_part_Int*[*simp*]: $\text{main_part } M S \cap \text{null_part } M S = \{\}$
using *split_completion*[*OF* *assms*]
by (*auto* *simp*: *split_completion_def* *split*: *if_split_asm*)

lemma *main_part*[*simp*]: $S \in \text{sets } M \implies \text{main_part } M S = S$
using *split_completion*[*of* $S M$]
by (*auto* *simp*: *split_completion_def* *split*: *if_split_asm*)

lemma *null_part*:
assumes $S \in \text{sets } (\text{completion } M)$ **shows** $\exists N. N \in \text{null_sets } M \wedge \text{null_part } M S \subseteq N$
using *split_completion*[*OF* *assms*] **by** (*auto* *simp*: *split_completion_def* *split*: *if_split_asm*)

lemma *null_part_sets*[*intro*, *simp*]:
assumes $S \in \text{sets } M$ **shows** $\text{null_part } M S \in \text{sets } M$ *emeasure* M ($\text{null_part } M S$) = 0
proof –
have $S: S \in \text{sets } (\text{completion } M)$ **using** *assms* **by** *auto*
have $*$: $S - \text{main_part } M S \in \text{sets } M$ **using** *assms* **by** *auto*
from *main_part_null_part_Un*[*OF* S] *main_part_null_part_Int*[*OF* S]
have $S - \text{main_part } M S = \text{null_part } M S$ **by** *auto*
with $*$ **show** $\text{sets: null_part } M S \in \text{sets } M$ **by** *auto*
from *null_part*[*OF* S] **obtain** N **where** $N \in \text{null_sets } M \wedge \text{null_part } M S \subseteq N$..
with *emeasure_eq_0*[*of* N $\text{null_part } M S$] *sets*
show *emeasure* M ($\text{null_part } M S$) = 0 **by** *auto*
qed

lemma *emeasure_main_part_UN*:
fixes $S :: \text{nat} \Rightarrow 'a \text{ set}$
assumes $\text{range } S \subseteq \text{sets } (\text{completion } M)$
shows *emeasure* M ($\text{main_part } M (\bigcup i. (S i))$) = *emeasure* M ($\bigcup i. \text{main_part } M (S i)$)
proof –
have $S: \bigwedge i. S i \in \text{sets } (\text{completion } M)$ **using** *assms* **by** *auto*
then have $UN: (\bigcup i. S i) \in \text{sets } (\text{completion } M)$ **by** *auto*
have $\forall i. \exists N. N \in \text{null_sets } M \wedge \text{null_part } M (S i) \subseteq N$

```

    using null_part[OF S] by auto
    then obtain N where N:  $\forall x. N x \in \text{null\_sets } M \wedge \text{null\_part } M (S x) \subseteq N x$ 
  by metis
    then have UN_N:  $(\bigcup i. N i) \in \text{null\_sets } M$  by (intro null_sets_UN) auto
    from S have  $(\bigcup i. S i) \in \text{sets } (\text{completion } M)$  by auto
    from null_part[OF this] obtain N' where N':  $N' \in \text{null\_sets } M \wedge \text{null\_part } M (\bigcup (\text{range } S)) \subseteq N' ..$ 
    let ?N =  $(\bigcup i. N i) \cup N'$ 
    have null_set:  $?N \in \text{null\_sets } M$  using N' UN_N by (intro null_sets.Un) auto
    have main_part M  $(\bigcup i. S i) \cup ?N = (\text{main\_part } M (\bigcup i. S i) \cup \text{null\_part } M (\bigcup i. S i)) \cup ?N$ 
    using N' by auto
    also have ... =  $(\bigcup i. \text{main\_part } M (S i) \cup \text{null\_part } M (S i)) \cup ?N$ 
    unfolding main_part_null_part_Un[OF S] main_part_null_part_Un[OF UN] by auto
    also have ... =  $(\bigcup i. \text{main\_part } M (S i)) \cup ?N$ 
    using N by auto
    finally have *:  $\text{main\_part } M (\bigcup i. S i) \cup ?N = (\bigcup i. \text{main\_part } M (S i)) \cup ?N$ 
  .
  have emeasure M  $(\text{main\_part } M (\bigcup i. S i)) = \text{emeasure } M (\text{main\_part } M (\bigcup i. S i) \cup ?N)$ 
    using null_set UN by (intro emeasure_Un_null_set[symmetric]) auto
    also have ... =  $\text{emeasure } M ((\bigcup i. \text{main\_part } M (S i)) \cup ?N)$ 
    unfolding * ..
    also have ... =  $\text{emeasure } M (\bigcup i. \text{main\_part } M (S i))$ 
    using null_set S by (intro emeasure_Un_null_set) auto
    finally show ?thesis .
qed

```

```

lemma emeasure_completion[simp]:
  assumes S:  $S \in \text{sets } (\text{completion } M)$ 
  shows emeasure  $(\text{completion } M) S = \text{emeasure } M (\text{main\_part } M S)$ 
proof (subst emeasure_measure_of[OF completion_def completion_into_space])
  let ? $\mu$  =  $\text{emeasure } M \circ \text{main\_part } M$ 
  show  $S \in \text{sets } (\text{completion } M) \text{ ?}\mu S = \text{emeasure } M (\text{main\_part } M S)$  using S
  by simp_all
  show positive  $(\text{sets } (\text{completion } M)) \text{ ?}\mu$ 
    by (simp add: positive_def)
  show countably_additive  $(\text{sets } (\text{completion } M)) \text{ ?}\mu$ 
  proof (intro countably_additiveI)
    fix A ::  $\text{nat} \Rightarrow 'a \text{ set}$  assume A:  $\text{range } A \subseteq \text{sets } (\text{completion } M) \text{ disjoint\_family } A$ 
    have disjoint_family  $(\lambda i. \text{main\_part } M (A i))$ 
    proof (intro disjoint_family_on_bisimulation[OF A(2)])
      fix n m assume  $A n \cap A m = \{\}$ 
      then have  $(\text{main\_part } M (A n) \cup \text{null\_part } M (A n)) \cap (\text{main\_part } M (A m) \cup \text{null\_part } M (A m)) = \{\}$ 
      using A by (subst (1 2) main_part_null_part_Un) auto
    end
  end

```

```

    then show  $\text{main\_part } M (A n) \cap \text{main\_part } M (A m) = \{\}$  by auto
  qed
  then have  $(\sum n. \text{emeasure } M (\text{main\_part } M (A n))) = \text{emeasure } M (\bigcup i. \text{main\_part } M (A i))$ 
    using A by (auto intro!: suminf_emeasure)
  then show  $(\sum n. ?\mu (A n)) = ?\mu (\bigcup (A \text{ ' UNIV}))$ 
    by (simp add: completion_def_emeasure_main_part_UN[OF A(1)])
  qed
qed

```

```

lemma measure_completion[simp]:  $S \in \text{sets } M \implies \text{measure } (\text{completion } M) S = \text{measure } M S$ 
  unfolding measure_def by auto

```

```

lemma emeasure_completion_UN:
   $\text{range } S \subseteq \text{sets } (\text{completion } M) \implies$ 
   $\text{emeasure } (\text{completion } M) (\bigcup i::\text{nat}. (S i)) = \text{emeasure } M (\bigcup i. \text{main\_part } M (S i))$ 
  by (subst emeasure_completion) (auto simp add: emeasure_main_part_UN)

```

```

lemma emeasure_completion_Un:
  assumes  $S: S \in \text{sets } (\text{completion } M)$  and  $T: T \in \text{sets } (\text{completion } M)$ 
  shows  $\text{emeasure } (\text{completion } M) (S \cup T) = \text{emeasure } M (\text{main\_part } M S \cup \text{main\_part } M T)$ 
  proof (subst emeasure_completion)
    have  $UN: (\bigcup i. \text{binary } (\text{main\_part } M S) (\text{main\_part } M T) i) = (\bigcup i. \text{main\_part } M (\text{binary } S T i))$ 
      unfolding binary_def by (auto split: if_split_asm)
    show  $\text{emeasure } M (\text{main\_part } M (S \cup T)) = \text{emeasure } M (\text{main\_part } M S \cup \text{main\_part } M T)$ 
      using emeasure_main_part_UN[of binary S T M] assms
      by (simp add: range_binary_eq, simp add: Un_range_binary UN)
  qed (auto intro: S T)

```

```

lemma sets_completionI_sub:
  assumes  $N: N' \in \text{null\_sets } M \subseteq N'$ 
  shows  $N \in \text{sets } (\text{completion } M)$ 
  using assms by (intro sets_completionI[of _  $\{\}$   $N N'$ ]) auto

```

```

lemma completion_ex_simple_function:
  assumes  $f: \text{simple\_function } (\text{completion } M) f$ 
  shows  $\exists f'. \text{simple\_function } M f' \wedge (\text{AE } x \text{ in } M. f x = f' x)$ 
  proof -
    let  $?F = \lambda x. f - \{x\} \cap \text{space } M$ 
    have  $F: \bigwedge x. ?F x \in \text{sets } (\text{completion } M)$  and  $\text{fin: finite } (f \text{ ' space } M)$ 
      using simple_functionD[OF f] simple_functionD[OF f] by simp_all
    have  $\forall x. \exists N. N \in \text{null\_sets } M \wedge \text{null\_part } M (?F x) \subseteq N$ 
      using F null_part by auto
    from choice[OF this] obtain  $N$  where

```

```

  N:  $\bigwedge x. \text{null\_part } M \text{ } (?F x) \subseteq N x \bigwedge x. N x \in \text{null\_sets } M$  by auto
let ?N =  $\bigcup_{x \in f' \text{space } M}. N x$ 
let ?f' =  $\lambda x. \text{if } x \in ?N \text{ then undefined else } f x$ 
have sets: ?N  $\in \text{null\_sets } M$  using N fin by (intro null_sets.finite_UN) auto
show ?thesis unfolding simple_function_def
proof (safe intro!: exI[of _ ?f'])
  have ?f' ' space M  $\subseteq f' \text{space } M \cup \{\text{undefined}\}$  by auto
  from finite_subset[OF this] simple_functionD(1)[OF f]
  show finite (?f' ' space M) by auto
next
fix x assume x  $\in \text{space } M$ 
have ?f' - ' {?f' x}  $\cap \text{space } M =$ 
  (if x  $\in ?N$  then ?F undefined  $\cup ?N$ 
  else if f x = undefined then ?F (f x)  $\cup ?N$ 
  else ?F (f x) - ?N)
using N(2) sets.sets_into_space by (auto split: if_split_asm simp: null_sets_def)
moreover { fix y have ?F y  $\cup ?N \in \text{sets } M$ 
proof cases
  assume y: y  $\in f' \text{space } M$ 
  have ?F y  $\cup ?N = (\text{main\_part } M \text{ } (?F y) \cup \text{null\_part } M \text{ } (?F y)) \cup ?N$ 
  using main_part_null_part_Un[OF F] by auto
  also have ... = main_part M (?F y)  $\cup ?N$ 
  using y N by auto
  finally show ?thesis
  using F sets by auto
next
  assume y  $\notin f' \text{space } M$  then have ?F y = {} by auto
  then show ?thesis using sets by auto
qed }
moreover {
have ?F (f x) - ?N = main_part M (?F (f x))  $\cup \text{null\_part } M \text{ } (?F (f x)) -$ 
?N
  using main_part_null_part_Un[OF F] by auto
  also have ... = main_part M (?F (f x)) - ?N
  using N  $\langle x \in \text{space } M \rangle$  by auto
  finally have ?F (f x) - ?N  $\in \text{sets } M$ 
  using F sets by auto }
ultimately show ?f' - ' {?f' x}  $\cap \text{space } M \in \text{sets } M$  by auto
next
show  $\exists x \text{ in } M. f x = ?f' x$ 
by (rule AE_I', rule sets) auto
qed
qed

```

lemma completion_ex_borel_measurable:

fixes g :: 'a $\Rightarrow \text{ennreal}$

assumes g: g $\in \text{borel_measurable } (\text{completion } M)$

shows $\exists g' \in \text{borel_measurable } M. (\exists x \text{ in } M. g x = g' x)$

proof -


```

obtain f :: nat  $\Rightarrow$  'a  $\Rightarrow$  ennreal
  where *:  $\bigwedge i.$  simple_function (completion M) (f i)
  and **:  $\bigwedge x.$  (SUP i. f i x) = g x
  using g[THEN borel_measurable_implies_simple_function_sequence] by metis
  from *[THEN completion_ex_simple_function]
  have  $\forall i.$   $\exists f'.$  simple_function M f'  $\wedge$  (AE x in M. f i x = f' x) ..
  then obtain f'
    where sf:  $\bigwedge i.$  simple_function M (f' i)
    and AE:  $\forall i.$  AE x in M. f i x = f' i x
    by metis
  show ?thesis
  proof (intro bexI)
    from AE[unfolded AE_all_countable[symmetric]]
    show AE x in M. g x = (SUP i. f' i x) (is AE x in M. g x = ?f x)
    proof (elim AE_mp, safe intro!: AE_I2)
      fix x assume eq:  $\forall i.$  f i x = f' i x
      have g x = (SUP i. f i x) by (auto simp: ** split: split_max)
      with eq show g x = ?f x by auto
    qed
    show ?f  $\in$  borel_measurable M
    using sf[THEN borel_measurable_simple_function] by auto
  qed
qed

lemma null_sets_completionI:  $N \in$  null_sets M  $\implies$   $N \in$  null_sets (completion M)
  by (auto simp: null_sets_def)

lemma AE_completion: (AE x in M. P x)  $\implies$  (AE x in completion M. P x)
  unfolding eventually_ae_filter by (auto intro: null_sets_completionI)

lemma null_sets_completion_iff:  $N \in$  sets M  $\implies$   $N \in$  null_sets (completion M)
 $\iff$   $N \in$  null_sets M
  by (auto simp: null_sets_def)

lemma sets_completion_AE: (AE x in M.  $\neg$  P x)  $\implies$  Measurable.pred (completion M) P
  unfolding pred_def sets_completion eventually_ae_filter
  by auto

lemma null_sets_completion_iff2:
   $A \in$  null_sets (completion M)  $\iff$  ( $\exists N' \in$  null_sets M.  $A \subseteq N'$ )
proof safe
  assume A  $\in$  null_sets (completion M)
  then have A: A  $\in$  sets (completion M) and main_part M A  $\in$  null_sets M
    by (auto simp: null_sets_def)
  moreover obtain N where N  $\in$  null_sets M null_part M A  $\subseteq$  N
    using null_part[OF A] by auto
  ultimately show  $\exists N' \in$  null_sets M.  $A \subseteq N'$ 

```

2438

```

proof (intro beXI)
  show  $A \subseteq N \cup \text{main\_part } M \ A$ 
    using  $\langle \text{null\_part } M \ A \subseteq N \rangle$  by (subst main_part_null_part_Un[OF A,
symmetric]) auto
  qed auto
next
  fix  $N$  assume  $N \in \text{null\_sets } M \ A \subseteq N$ 
  then have  $A \in \text{sets } (\text{completion } M)$  and  $N: N \in \text{sets } M \ A \subseteq N \ \text{emeasure } M \ N$ 
   $= 0$ 
    by (auto intro: null_sets_completion)
  moreover have  $\text{emeasure } (\text{completion } M) \ A = 0$ 
    using  $N$  by (intro emeasure_eq_0[of N _ A]) auto
  ultimately show  $A \in \text{null\_sets } (\text{completion } M)$ 
    by auto
qed

```

```

lemma null_sets_completion_subset:
   $B \subseteq A \implies A \in \text{null\_sets } (\text{completion } M) \implies B \in \text{null\_sets } (\text{completion } M)$ 
  unfolding null_sets_completion_iff2 by auto

```

interpretation completion: complete_measure completion M **for** M

```

proof
  show  $B \subseteq A \implies A \in \text{null\_sets } (\text{completion } M) \implies B \in \text{sets } (\text{completion } M)$ 
  for  $B \ A$ 
    using null_sets_completion_subset[of B A M] by (simp add: null_sets_def)
qed

```

```

lemma null_sets_restrict_space:
   $\Omega \in \text{sets } M \implies A \in \text{null\_sets } (\text{restrict\_space } M \ \Omega) \iff A \subseteq \Omega \wedge A \in \text{null\_sets } M$ 
  by (auto simp: null_sets_def emeasure_restrict_space sets_restrict_space)

```

lemma completion_ex_borel_measurable_real:

```

  fixes  $g :: 'a \Rightarrow \text{real}$ 
  assumes  $g: g \in \text{borel\_measurable } (\text{completion } M)$ 
  shows  $\exists g' \in \text{borel\_measurable } M. (\text{AE } x \text{ in } M. g \ x = g' \ x)$ 
proof -
  have  $(\lambda x. \text{ennreal } (g \ x)) \in \text{completion } M \rightarrow_M \text{borel } (\lambda x. \text{ennreal } (- g \ x)) \in$ 
   $\text{completion } M \rightarrow_M \text{borel}$ 
    using  $g$  by auto
  from this[THEN completion_ex_borel_measurable]
  obtain  $pf \ nf :: 'a \Rightarrow \text{ennreal}$ 
    where [measurable]:  $nf \in M \rightarrow_M \text{borel}$   $pf \in M \rightarrow_M \text{borel}$ 
    and  $ae: \text{AE } x \text{ in } M. pf \ x = \text{ennreal } (g \ x) \ \text{AE } x \text{ in } M. nf \ x = \text{ennreal } (- g \ x)$ 
    by (auto simp: eq_commute)
  then have  $\text{AE } x \text{ in } M. pf \ x = \text{ennreal } (g \ x) \wedge nf \ x = \text{ennreal } (- g \ x)$ 
    by auto
  then obtain  $N$  where  $N \in \text{null\_sets } M \ \{x \in \text{space } M. pf \ x \neq \text{ennreal } (g \ x) \wedge$ 
   $nf \ x \neq \text{ennreal } (- g \ x)\} \subseteq N$ 

```

```

  by (auto elim!: AE_E)
show ?thesis
proof
  let ?F =  $\lambda x. \text{indicator } (\text{space } M - N) x * (\text{enn2real } (pf x) - \text{enn2real } (nf x))$ 
  show ?F  $\in M \rightarrow_M \text{borel}$ 
    using  $\langle N \in \text{null\_sets } M \rangle$  by auto
  show AE x in M.  $g x = ?F x$ 
    using  $\langle N \in \text{null\_sets } M \rangle$  [THEN AE_not_in] ae AE_space
    apply eventually_elim
    subgoal for x
      by (cases 0::real  $g x$  rule: linorder_le_cases) (auto simp: ennreal_neg)
    done
qed
qed

lemma simple_function_completion: simple_function M f  $\implies$  simple_function
(completion M) f
  by (simp add: simple_function_def)

lemma simple_integral_completion:
  simple_function M f  $\implies$  simple_integral (completion M) f = simple_integral M
f
  unfolding simple_integral_def by simp

lemma nn_integral_completion: nn_integral (completion M) f = nn_integral M
f
  unfolding nn_integral_def
proof (safe intro!: SUP_eq)
  fix s assume s: simple_function (completion M) s and  $s \leq f$ 
  then obtain s' where s': simple_function M s' AE x in M.  $s x = s' x$ 
    by (auto dest: completion_ex_simple_function)
  then obtain N where N:  $N \in \text{null\_sets } M \{x \in \text{space } M. s x \neq s' x\} \subseteq N$ 
    by (auto elim!: AE_E)
  then have ae_N: AE x in M.  $(s x \neq s' x \longrightarrow x \in N) \wedge x \notin N$ 
    by (auto dest: AE_not_in)
  define s'' where s'' x = (if  $x \in N$  then 0 else s x) for x
  then have ae_s_eq_s'': AE x in completion M.  $s x = s'' x$ 
    using s' ae_N by (intro AE_completion) auto
  have s'': simple_function M s''
proof (subst simple_function_cong)
  show  $t \in \text{space } M \implies s'' t = (\text{if } t \in N \text{ then } 0 \text{ else } s' t)$  for t
    using N by (auto simp: s''_def dest: sets.sets_into_space)
  show simple_function M  $(\lambda t. \text{if } t \in N \text{ then } 0 \text{ else } s' t)$ 
    unfolding s''_def [abs_def] using N by (auto intro!: simple_function_If s')
qed

show  $\exists j \in \{g. \text{simple\_function } M g \wedge g \leq f\}. \text{integral}^S (\text{completion } M) s \leq$ 
 $\text{integral}^S M j$ 
  proof (safe intro!: bexI [of _ s''])

```

```

have  $\text{integral}^S (\text{completion } M) s = \text{integral}^S (\text{completion } M) s''$ 
by (intro simple_integral_cong_AE s simple_function_completion s'' ae_s_eq_s'')
then show  $\text{integral}^S (\text{completion } M) s \leq \text{integral}^S M s''$ 
  using  $s''$  by (simp add: simple_integral_completion)
from  $\langle s \leq f \rangle$  show  $s'' \leq f$ 
  unfolding  $s''\_def\ le\_fun\_def$  by auto
qed fact
next
fix  $s$  assume simple_function  $M s s \leq f$ 
then show  $\exists j \in \{g. \text{simple\_function } (\text{completion } M) g \wedge g \leq f\}. \text{integral}^S M s$ 
 $\leq \text{integral}^S (\text{completion } M) j$ 
by (intro bexI[of _ s]) (auto simp: simple_integral_completion simple_function_completion)
qed

```

lemma *integrable_completion*:

```

fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
shows  $f \in M \rightarrow_M \text{borel} \implies \text{integrable } (\text{completion } M) f \longleftrightarrow \text{integrable } M f$ 
by (rule integrable_subalgebra[symmetric]) auto

```

lemma *integral_completion*:

```

fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second\_countable\_topology}\}$ 
assumes  $f: f \in M \rightarrow_M \text{borel}$  shows  $\text{integral}^L (\text{completion } M) f = \text{integral}^L M f$ 
by (rule integral_subalgebra[symmetric]) (auto intro: f)

```

locale *semifinite_measure* =

```

fixes  $M :: 'a \text{ measure}$ 
assumes semifinite:
 $\bigwedge A. A \in \text{sets } M \implies \text{emeasure } M A = \infty \implies \exists B \in \text{sets } M. B \subseteq A \wedge \text{emeasure } M B < \infty$ 

```

locale *locally_determined_measure* = *semifinite_measure* +

```

assumes locally_determined:
 $\bigwedge A. A \subseteq \text{space } M \implies (\bigwedge B. B \in \text{sets } M \implies \text{emeasure } M B < \infty \implies A \cap B \in \text{sets } M) \implies A \in \text{sets } M$ 

```

locale *cld_measure* =

```

 $\text{complete\_measure } M + \text{locally\_determined\_measure } M$  for  $M :: 'a \text{ measure}$ 

```

definition *outer_measure_of* :: $'a \text{ measure} \Rightarrow 'a \text{ set} \Rightarrow \text{ennreal}$

```

where  $\text{outer\_measure\_of } M A = (\text{INF } B \in \{B \in \text{sets } M. A \subseteq B\}. \text{emeasure } M B)$ 

```

lemma *outer_measure_of_eq[simp]*: $A \in \text{sets } M \implies \text{outer_measure_of } M A = \text{emeasure } M A$

```

by (auto simp: outer_measure_of_def intro!: INF_eqI emeasure_mono)

```

lemma *outer_measure_of_mono*: $A \subseteq B \implies \text{outer_measure_of } M A \leq \text{outer_measure_of } M B$

```

unfolding outer_measure_of_def by (intro INF_superset_mono) auto

lemma outer_measure_of_attain:
  assumes  $A \subseteq \text{space } M$ 
  shows  $\exists E \in \text{sets } M. A \subseteq E \wedge \text{outer\_measure\_of } M A = \text{emeasure } M E$ 
proof -
  have  $\text{emeasure } M \{B \in \text{sets } M. A \subseteq B\} \neq \{\}$ 
    using  $\langle A \subseteq \text{space } M \rangle$  by auto
  from ennreal_Inf_countable_INF[OF this]
  obtain f
    where  $f: \text{range } f \subseteq \text{emeasure } M \{B \in \text{sets } M. A \subseteq B\}$  decseq f
    and  $\text{outer\_measure\_of } M A = (\text{INF } i. f i)$ 
  unfolding outer_measure_of_def by auto
  have  $\exists E. \forall n. (E n \in \text{sets } M \wedge A \subseteq E n \wedge \text{emeasure } M (E n) \leq f n) \wedge E (\text{Suc } n) \subseteq E n$ 
    proof (rule dependent_nat_choice)
      show  $\exists x. x \in \text{sets } M \wedge A \subseteq x \wedge \text{emeasure } M x \leq f 0$ 
        using  $f(1)$  by (fastforce simp: image_subset_iff image_iff intro: eq_refl[OF sym])
      next
        fix  $E n$  assume  $E \in \text{sets } M \wedge A \subseteq E \wedge \text{emeasure } M E \leq f n$ 
        moreover obtain  $F$  where  $F \in \text{sets } M \wedge A \subseteq F \wedge f (\text{Suc } n) = \text{emeasure } M F$ 
          using  $f(1)$  by (auto simp: image_subset_iff image_iff)
        ultimately show  $\exists y. (y \in \text{sets } M \wedge A \subseteq y \wedge \text{emeasure } M y \leq f (\text{Suc } n)) \wedge y \subseteq E$ 
          by (auto intro!: exI[of _  $F \cap E$ ] emeasure_mono)
    qed
  then obtain  $E$ 
    where  $[\text{simp}]: \bigwedge n. E n \in \text{sets } M$ 
    and  $\bigwedge n. A \subseteq E n$ 
    and  $le\_f: \bigwedge n. \text{emeasure } M (E n) \leq f n$ 
    and decseq  $E$ 
    by (auto simp: decseq_Suc_iff)
  show ?thesis
proof cases
  assume  $fin: \exists i. \text{emeasure } M (E i) < \infty$ 
  show ?thesis
    proof (intro bestI[of  $\bigcap i. E i$ ] conjI)
      show  $A \subseteq (\bigcap i. E i) \wedge (\bigcap i. E i) \in \text{sets } M$ 
        using  $\langle \bigwedge n. A \subseteq E n \rangle$  by auto

      have  $(\text{INF } i. \text{emeasure } M (E i)) \leq \text{outer\_measure\_of } M A$ 
        unfolding  $\langle \text{outer\_measure\_of } M A = (\text{INF } n. f n) \rangle$ 
        by (intro INF_superset_mono le_f) auto
      moreover have  $\text{outer\_measure\_of } M A \leq (\text{INF } i. \text{outer\_measure\_of } M (E i))$ 
        by (intro INF_greatest outer_measure_of_mono  $\langle \bigwedge n. A \subseteq E n \rangle$ )
      ultimately have  $\text{outer\_measure\_of } M A = (\text{INF } i. \text{emeasure } M (E i))$ 
        by auto
    qed

```

```

    also have ... = emeasure M ( $\bigcap i. E i$ )
    using fin by (intro INF_emeasure_decseq' <decseq E>) (auto simp: less_top)
    finally show outer_measure_of M A = emeasure M ( $\bigcap i. E i$ ) .
  qed
next
  assume  $\nexists i. \text{emeasure } M (E i) < \infty$ 
  then have  $f n = \infty$  for  $n$ 
    using le_f by (auto simp: not_less_top_unique)
  moreover have  $\exists E \in \text{sets } M. A \subseteq E \wedge f 0 = \text{emeasure } M E$ 
    using f by auto
  ultimately show ?thesis
    unfolding <outer_measure_of M A = (INF n. f n)> by simp
  qed
qed

lemma SUP_outer_measure_of_incseq:
  assumes A:  $\bigwedge n. A n \subseteq \text{space } M$  and incseq A
  shows (SUP n. outer_measure_of M (A n)) = outer_measure_of M ( $\bigcup i. A i$ )
proof (rule antisym)
  obtain E
    where E:  $\bigwedge n. E n \in \text{sets } M \wedge n. A n \subseteq E n \wedge n. \text{outer\_measure\_of } M (A n)$ 
    = emeasure M (E n)
    using outer_measure_of_attain[OF A] by metis

  define F where  $F n = (\bigcap i \in \{n ..\}. E i)$  for  $n$ 
  with E have F: incseq F  $\bigwedge n. F n \in \text{sets } M$ 
    by (auto simp: incseq_def)
  have  $A n \subseteq F n$  for  $n$ 
    using incseqD[OF <incseq A>, of n] < $\bigwedge n. A n \subseteq E n$ > by (auto simp: F_def)

  have eq: outer_measure_of M (A n) = outer_measure_of M (F n) for  $n$ 
  proof (intro antisym)
    have outer_measure_of M (F n)  $\leq$  outer_measure_of M (E n)
      by (intro outer_measure_of_mono) (auto simp add: F_def)
    with E show outer_measure_of M (F n)  $\leq$  outer_measure_of M (A n)
      by auto
    show outer_measure_of M (A n)  $\leq$  outer_measure_of M (F n)
      by (intro outer_measure_of_mono <A n  $\subseteq$  F n>)
  qed

  have outer_measure_of M ( $\bigcup n. A n$ )  $\leq$  outer_measure_of M ( $\bigcup n. F n$ )
    using < $\bigwedge n. A n \subseteq F n$ > by (intro outer_measure_of_mono) auto
  also have ... = (SUP n. emeasure M (F n))
    using F by (simp add: SUP_emeasure_incseq_subset_eq)
  finally show outer_measure_of M ( $\bigcup n. A n$ )  $\leq$  (SUP n. outer_measure_of M
(A n))
    by (simp add: eq F)
qed (auto intro: SUP_least_outer_measure_of_mono)

```

definition *measurable_envelope* :: 'a measure \Rightarrow 'a set \Rightarrow 'a set \Rightarrow bool
where *measurable_envelope* M A E \longleftrightarrow
 $(A \subseteq E \wedge E \in \text{sets } M \wedge (\forall F \in \text{sets } M. \text{emeasure } M (F \cap E) = \text{outer_measure_of } M (F \cap A)))$

lemma *measurable_envelopeD*:
assumes *measurable_envelope* M A E
shows $A \subseteq E$
and $E \in \text{sets } M$
and $\bigwedge F. F \in \text{sets } M \implies \text{emeasure } M (F \cap E) = \text{outer_measure_of } M (F \cap A)$
and $A \subseteq \text{space } M$
using *assms sets.sets_into_space[of E]* **by** (*auto simp: measurable_envelope_def*)

lemma *measurable_envelopeD1*:
assumes $E: \text{measurable_envelope } M A E$ **and** $F: F \in \text{sets } M \ F \subseteq E - A$
shows $\text{emeasure } M F = 0$
proof –
have $\text{emeasure } M F = \text{emeasure } M (F \cap E)$
using F **by** (*intro arg_cong2[where f=emeasure]*) *auto*
also have $\dots = \text{outer_measure_of } M (F \cap A)$
using *measurable_envelopeD[OF E]* $\langle F \in \text{sets } M \rangle$ **by** (*auto simp: measurable_envelope_def*)
also have $\dots = \text{outer_measure_of } M \ \{\}$
using $\langle F \subseteq E - A \rangle$ **by** (*intro arg_cong2[where f=outer_measure_of]*) *auto*
finally show $\text{emeasure } M F = 0$
by *simp*
qed

lemma *measurable_envelope_eq1*:
assumes $A \subseteq E \ E \in \text{sets } M$
shows $\text{measurable_envelope } M A E \longleftrightarrow (\forall F \in \text{sets } M. F \subseteq E - A \longrightarrow \text{emeasure } M F = 0)$
proof *safe*
assume $*$: $\forall F \in \text{sets } M. F \subseteq E - A \longrightarrow \text{emeasure } M F = 0$
show *measurable_envelope* M A E
unfolding *measurable_envelope_def*
proof (*rule ccontr, auto simp add: $\langle E \in \text{sets } M \rangle \langle A \subseteq E \rangle$*)
fix F **assume** $F \in \text{sets } M \ \text{emeasure } M (F \cap E) \neq \text{outer_measure_of } M (F \cap A)$
then have $\text{outer_measure_of } M (F \cap A) < \text{emeasure } M (F \cap E)$
using *outer_measure_of_mono[of F \cap A F \cap E M]* $\langle A \subseteq E \rangle \langle E \in \text{sets } M \rangle$
by (*auto simp: less_le*)
then obtain G **where** $G: G \in \text{sets } M \ F \cap A \subseteq G$ **and** *less*: $\text{emeasure } M G < \text{emeasure } M (E \cap F)$
unfolding *outer_measure_of_def INF_less_iff* **by** (*auto simp: ac_simps*)
have *le*: $\text{emeasure } M (G \cap E \cap F) \leq \text{emeasure } M G$
using $\langle E \in \text{sets } M \rangle \langle G \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$ **by** (*auto intro!: measure_mono*)

```

from  $G$  have  $E \cap F - G \in \text{sets } M$   $E \cap F - G \subseteq E - A$ 
  using  $\langle F \in \text{sets } M \rangle \langle E \in \text{sets } M \rangle$  by auto
with  $*$  have  $0 = \text{emeasure } M (E \cap F - G)$ 
  by auto
also have  $E \cap F - G = E \cap F - (G \cap E \cap F)$ 
  by auto
also have  $\text{emeasure } M (E \cap F - (G \cap E \cap F)) = \text{emeasure } M (E \cap F) -$ 
 $\text{emeasure } M (G \cap E \cap F)$ 
  using  $\langle E \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$  le less  $G$  by (intro emeasure_Diff) (auto simp: top_unique)
  also have  $\dots > 0$ 
  using le less by (intro diff_gr0_enreal) auto
  finally show False by auto
qed
qed (rule measurable_envelopeD1)

```

lemma *measurable_envelopeD2*:

```

assumes  $E: \text{measurable\_envelope } M A E$  shows  $\text{emeasure } M E = \text{outer\_measure\_of } M A$ 
proof -
from  $\langle \text{measurable\_envelope } M A E \rangle$  have  $\text{emeasure } M (E \cap E) = \text{outer\_measure\_of } M (E \cap A)$ 
  by (auto simp: measurable_envelope_def)
with measurable_envelopeD[OF E] show  $\text{emeasure } M E = \text{outer\_measure\_of } M A$ 
  by (auto simp: Int_absorb1)
qed

```

lemma *measurable_envelope_eq2*:

```

assumes  $A \subseteq E$   $E \in \text{sets } M$   $\text{emeasure } M E < \infty$ 
shows  $\text{measurable\_envelope } M A E \longleftrightarrow (\text{emeasure } M E = \text{outer\_measure\_of } M A)$ 
proof safe
assume  $*$ :  $\text{emeasure } M E = \text{outer\_measure\_of } M A$ 
show measurable_envelope  $M A E$ 
  unfolding measurable_envelope_eq1[OF  $\langle A \subseteq E \rangle \langle E \in \text{sets } M \rangle$ ]
proof (intro conjI ballI impI assms)
  fix  $F$  assume  $F: F \in \text{sets } M$   $F \subseteq E - A$ 
  with  $\langle E \in \text{sets } M \rangle$  have  $le: \text{emeasure } M F \leq \text{emeasure } M E$ 
  by (intro emeasure_mono) auto
  from  $F$   $\langle A \subseteq E \rangle$  have  $\text{outer\_measure\_of } M A \leq \text{outer\_measure\_of } M (E - F)$ 
  by (intro outer_measure_of_mono) auto
  then have  $\text{emeasure } M E - 0 \leq \text{emeasure } M (E - F)$ 
  using  $*$   $\langle E \in \text{sets } M \rangle \langle F \in \text{sets } M \rangle$  by simp
  also have  $\dots = \text{emeasure } M E - \text{emeasure } M F$ 
  using  $\langle E \in \text{sets } M \rangle \langle \text{emeasure } M E < \infty \rangle$   $F$  le by (intro emeasure_Diff) (auto simp: top_unique)

```


finally show $\text{emeasure } M F = 0$
using $\text{ennreal_mono_minus_cancel}$ [of $\text{emeasure } M E 0 \text{emeasure } M F$] *le*
assms **by** *auto*
qed
qed (*auto intro: measurable_envelopeD2*)

lemma *measurable_envelopeI_countable*:

fixes $A :: \text{nat} \Rightarrow 'a \text{ set}$
assumes $E: \bigwedge n. \text{measurable_envelope } M (A n) (E n)$
shows $\text{measurable_envelope } M (\bigcup n. A n) (\bigcup n. E n)$
proof (*subst measurable_envelope_eq1*)
show $(\bigcup n. A n) \subseteq (\bigcup n. E n) \ (\bigcup n. E n) \in \text{sets } M$
using $\text{measurable_envelopeD}(1,2)$ [OF E] **by** *auto*
show $\forall F \in \text{sets } M. F \subseteq (\bigcup n. E n) - (\bigcup n. A n) \longrightarrow \text{emeasure } M F = 0$
proof *safe*
fix F **assume** $F: F \in \text{sets } M \ F \subseteq (\bigcup n. E n) - (\bigcup n. A n)$
then have $F \cap E n \in \text{sets } M \ F \cap E n \subseteq E n - A n \ F \subseteq (\bigcup n. E n)$ **for** n
using $\text{measurable_envelopeD}(1,2)$ [OF E] **by** *auto*
then have $\text{emeasure } M (\bigcup n. F \cap E n) = 0$
by (*intro emeasure_UN_eq_0 measurable_envelopeD1*[OF E]) *auto*
then show $\text{emeasure } M F = 0$
using $\langle F \subseteq (\bigcup n. E n) \rangle$ **by** (*auto simp: Int_absorb2*)
qed
qed

lemma *measurable_envelopeI_countable_cover*:

fixes A **and** $C :: \text{nat} \Rightarrow 'a \text{ set}$
assumes $C: A \subseteq (\bigcup n. C n) \ \bigwedge n. C n \in \text{sets } M \ \bigwedge n. \text{emeasure } M (C n) < \infty$
shows $\exists E \subseteq (\bigcup n. C n). \text{measurable_envelope } M A E$
proof –
have $A \cap C n \subseteq \text{space } M$ **for** n
using $\langle C n \in \text{sets } M \rangle$ **by** (*auto dest: sets.sets_into_space*)
then have $\forall n. \exists E \in \text{sets } M. A \cap C n \subseteq E \ \wedge \ \text{outer_measure_of } M (A \cap C n)$
 $= \text{emeasure } M E$
using $\text{outer_measure_of_attain}$ [of $A \cap C n \ M$ **for** n] **by** *auto*
then obtain E
where $E: \bigwedge n. E n \in \text{sets } M \ \bigwedge n. A \cap C n \subseteq E n$
and $\text{eq}: \bigwedge n. \text{outer_measure_of } M (A \cap C n) = \text{emeasure } M (E n)$
by *metis*

have $\text{outer_measure_of } M (A \cap C n) \leq \text{outer_measure_of } M (E n \cap C n)$ **for**
 n
using E **by** (*intro outer_measure_of_mono*) *auto*
moreover have $\text{outer_measure_of } M (E n \cap C n) \leq \text{outer_measure_of } M (E$
 $n)$ **for** n
by (*intro outer_measure_of_mono*) *auto*
ultimately have $\text{eq}: \text{outer_measure_of } M (A \cap C n) = \text{emeasure } M (E n \cap C$
 $n)$ **for** n
using $E \ C$ **by** (*intro antisym*) (*auto simp: eq*)

```

{ fix n
  have outer_measure_of M (A ∩ C n) ≤ outer_measure_of M (C n)
    by (intro outer_measure_of_mono) simp
  also have ... < ∞
    using assms by auto
  finally have emeasure M (E n ∩ C n) < ∞
    using eq by simp }
then have measurable_envelope M (⋃ n. A ∩ C n) (⋃ n. E n ∩ C n)
  using E C by (intro measurable_envelopeI_countable measurable_envelope_eq2[THEN
iffD2]) (auto simp: eq)
with ⟨A ⊆ (⋃ n. C n)⟩ show ?thesis
  by (intro exI[of _ (⋃ n. E n ∩ C n)]) (auto simp add: Int_absorb2)
qed

```

```

lemma (in complete_measure) complete_sets_sandwich:
  assumes [measurable]: A ∈ sets M C ∈ sets M and subset: A ⊆ B B ⊆ C
    and measure: emeasure M A = emeasure M C emeasure M A < ∞
  shows B ∈ sets M
proof -
  have B - A ∈ sets M
  proof (rule complete)
    show B - A ⊆ C - A
      using subset by auto
    show C - A ∈ null_sets M
      using measure subset by (simp add: emeasure_Diff null_setsI)
  qed
  then have A ∪ (B - A) ∈ sets M
    by measurable
  also have A ∪ (B - A) = B
    using ⟨A ⊆ B⟩ by auto
  finally show ?thesis .
qed

```

```

lemma (in complete_measure) complete_sets_sandwich_fmeasurable:
  assumes [measurable]: A ∈ fmeasurable M C ∈ fmeasurable M and subset: A ⊆
B B ⊆ C
    and measure: measure M A = measure M C
  shows B ∈ fmeasurable M
proof (rule fmeasurableI2)
  show B ⊆ C C ∈ fmeasurable M by fact+
  show B ∈ sets M
  proof (rule complete_sets_sandwich)
    show A ∈ sets M C ∈ sets M A ⊆ B B ⊆ C
      using assms by auto
    show emeasure M A < ∞
      using ⟨A ∈ fmeasurable M⟩ by (auto simp: fmeasurable_def)
    show emeasure M A = emeasure M C
      using assms by (simp add: emeasure_eq_measure2)
  qed

```

qed
qed

lemma *AE_completion_iff*: $(AE\ x\ in\ completion\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ M.\ P\ x)$
proof

assume *AE x in completion M. P x*
 then obtain *N* **where** $N \in null_sets\ (completion\ M)$ **and** $P: \{x \in space\ M.\ \neg P\ x\} \subseteq N$
 unfolding *eventually_ae_filter* **by** *auto*
 then obtain *N'* **where** $N': N' \in null_sets\ M$ **and** $N \subseteq N'$
 unfolding *null_sets_completion_iff2* **by** *auto*
 from $P\ \langle N \subseteq N' \rangle$ **have** $\{x \in space\ M.\ \neg P\ x\} \subseteq N'$
 by *auto*
 with *N'* **show** *AE x in M. P x*
 unfolding *eventually_ae_filter* **by** *auto*
qed (*rule AE_completion*)

lemma *null_part_null_sets*: $S \in completion\ M \implies null_part\ M\ S \in null_sets\ (completion\ M)$
by (*auto dest!: null_part_intro: null_sets_completionI null_sets_completion_subset*)

lemma *AE_notin_null_part*: $S \in completion\ M \implies (AE\ x\ in\ M.\ x \notin null_part\ M\ S)$
by (*auto dest!: null_part_null_sets AE_not_in_simp: AE_completion_iff*)

lemma *completion_upper*:
 assumes *A: A \in sets (completion M)*
 shows $\exists A' \in sets\ M.\ A \subseteq A' \wedge emeasure\ (completion\ M)\ A = emeasure\ M\ A'$
proof –
 from *AE_notin_null_part[OF A]* **obtain** *N* **where** $N: N \in null_sets\ M$ $null_part\ M\ A \subseteq N$
 unfolding *eventually_ae_filter* **using** *null_part_null_sets[OF A, THEN null_setsD2, THEN sets.sets_into_space]* **by** *auto*
 show *?thesis*
 proof (*intro bexI conjI*)
 show $A \subseteq main_part\ M\ A \cup N$
 using $\langle null_part\ M\ A \subseteq N \rangle$ **by** (*subst main_part_null_part_Un[symmetric, OF A]*) *auto*
 show $emeasure\ (completion\ M)\ A = emeasure\ M\ (main_part\ M\ A \cup N)$
 using $A\ \langle N \in null_sets\ M \rangle$ **by** (*simp add: emeasure_Un_null_set*)
 qed (*use A N in auto*)
qed

lemma *AE_in_main_part*:
 assumes *A: A \in completion M* **shows** $AE\ x\ in\ M.\ x \in main_part\ M\ A \longleftrightarrow x \in A$
 using *AE_notin_null_part[OF A]*
 by (*subst (2) main_part_null_part_Un[symmetric, OF A]*) *auto*

lemma *completion_density_eq*:
assumes *ae*: $AE\ x\ in\ M.\ 0 < f\ x$ **and** [*measurable*]: $f \in M \rightarrow_M\ borel$
shows $completion\ (density\ M\ f) = density\ (completion\ M)\ f$
proof (*intro\ measure_eqI*)
have $N' \in sets\ M \wedge (AE\ x \in N' \ in\ M.\ f\ x = 0) \longleftrightarrow N' \in null_sets\ M$ **for** N'
proof *safe*
assume $N': N' \in sets\ M$ **and** *ae_N'*: $AE\ x \in N' \ in\ M.\ f\ x = 0$
from *ae_N'* **ae** **have** $AE\ x \ in\ M.\ x \notin N'$
by *eventually_elim\ auto*
then show $N' \in null_sets\ M$
using N' **by** (*simp\ add: AE_iff_null_sets*)
next
assume $N': N' \in null_sets\ M$ **then show** $N' \in sets\ M\ AE\ x \in N' \ in\ M.\ f\ x = 0$
using *ae\ AE_not_in[OF\ N']* **by** (*auto\ simp: less_le*)
qed
then show *sets_eq*: $sets\ (completion\ (density\ M\ f)) = sets\ (density\ (completion\ M)\ f)$
by (*simp\ add: sets_completion_null_sets_density_iff*)

fix A **assume** $A: \langle A \in completion\ (density\ M\ f) \rangle$
moreover
have $A \in completion\ M$
using A **unfolding** *sets_eq* **by** *simp*
moreover
have *main_part* $(density\ M\ f)\ A \in M$
using A *main_part_sets*[*of\ A\ density\ M\ f*] **unfolding** *sets_density\ sets_eq* **by**
simp
moreover have $AE\ x \ in\ M.\ x \in main_part\ (density\ M\ f)\ A \longleftrightarrow x \in A$
using *AE_in_main_part*[*OF\ \langle A \in completion\ (density\ M\ f) \rangle*] *ae* **by** (*auto\ simp\ add: AE_density*)
ultimately show $emeasure\ (completion\ (density\ M\ f))\ A = emeasure\ (density\ (completion\ M)\ f)\ A$
by (*auto\ simp\ add: emeasure_density\ measurable_completion\ nn_integral_completion\ intro!: nn_integral_cong_AE*)
qed

lemma *null_sets_subset*: $B \in null_sets\ M \implies A \in sets\ M \implies A \subseteq B \implies A \in null_sets\ M$
using *emeasure_mono*[*of\ A\ B\ M*] **by** (*simp\ add: null_sets_def*)

lemma (*in\ complete_measure*) *complete2*: $A \subseteq B \implies B \in null_sets\ M \implies A \in null_sets\ M$
using *complete*[*of\ A\ B*] *null_sets_subset*[*of\ B\ M\ A*] **by** *simp*

lemma (*in\ complete_measure*) *AE_iff_null_sets*: $(AE\ x \ in\ M.\ P\ x) \longleftrightarrow \{x \in space\ M.\ \neg\ P\ x\} \in null_sets\ M$
unfolding *eventually_ae_filter* **by** (*auto\ intro: complete2*)

lemma (in complete_measure) null_sets_iff_AE: $A \in \text{null_sets } M \longleftrightarrow ((AE \ x \text{ in } M. x \notin A) \wedge A \subseteq \text{space } M)$
unfolding AE_iff_null_sets **by** (auto cong: rev_conj_cong dest: sets.sets_into_space simp: subset_eq)

lemma (in complete_measure) in_sets_AE:
assumes ae: $AE \ x \text{ in } M. x \in A \longleftrightarrow x \in B$ **and** A: $A \in \text{sets } M$ **and** B: $B \subseteq \text{space } M$
shows $B \in \text{sets } M$
proof –
have $(AE \ x \text{ in } M. x \notin B - A \wedge x \notin A - B)$
using ae **by** eventually_elim auto
then have $B - A \in \text{null_sets } M \wedge A - B \in \text{null_sets } M$
using A B **unfolding** null_sets_iff_AE **by** (auto dest: sets.sets_into_space)
then have $A \cup (B - A) - (A - B) \in \text{sets } M$
using A **by** blast
also have $A \cup (B - A) - (A - B) = B$
by auto
finally show $B \in \text{sets } M$.
qed

lemma (in complete_measure) vimage_null_part_null_sets:
assumes f: $f \in M \rightarrow_M N$ **and** eq: $\text{null_sets } N \subseteq \text{null_sets } (\text{distr } M \ N \ f)$
and A: $A \in \text{completion } N$
shows $f - ' \text{null_part } N \ A \cap \text{space } M \in \text{null_sets } M$
proof –
obtain N' **where** $N' \in \text{null_sets } N \wedge \text{null_part } N \ A \subseteq N'$
using null_part[OF A] **by** auto
then have $N' \in \text{null_sets } (\text{distr } M \ N \ f)$
using eq **by** auto
show ?thesis
proof (rule complete2)
show $f - ' \text{null_part } N \ A \cap \text{space } M \subseteq f - ' N' \cap \text{space } M$
using $\langle \text{null_part } N \ A \subseteq N' \rangle$ **by** auto
show $f - ' N' \cap \text{space } M \in \text{null_sets } M$
using f N' **by** (auto simp: null_sets_def emeasure_distr)
qed
qed

lemma (in complete_measure) vimage_null_part_sets:
 $f \in M \rightarrow_M N \implies \text{null_sets } N \subseteq \text{null_sets } (\text{distr } M \ N \ f) \implies A \in \text{completion } N \implies$
 $f - ' \text{null_part } N \ A \cap \text{space } M \in \text{sets } M$
using vimage_null_part_null_sets[of f N A] **by** auto

lemma (in complete_measure) measurable_completion2:
assumes f: $f \in M \rightarrow_M N$ **and** null_sets: $\text{null_sets } N \subseteq \text{null_sets } (\text{distr } M \ N \ f)$
shows $f \in M \rightarrow_M \text{completion } N$

proof (*rule measurableI*)
show $x \in \text{space } M \implies f x \in \text{space } (\text{completion } N)$ **for** x
using $f[\text{THEN measurable_space}]$ **by** *auto*
fix A **assume** $A: A \in \text{sets } (\text{completion } N)$
have $f -' A \cap \text{space } M = (f -' \text{main_part } N A \cap \text{space } M) \cup (f -' \text{null_part } N A \cap \text{space } M)$
using $\text{main_part_null_part_Un}[OF A]$ **by** *auto*
then show $f -' A \cap \text{space } M \in \text{sets } M$
using $f A \text{ null_sets}$ **by** (*auto intro: vimage_null_part_sets measurable_sets*)
qed

lemma (*in complete_measure*) *completion_distr_eq*:
assumes $X: X \in M \rightarrow_M N$ **and** $\text{null_sets}: \text{null_sets } (\text{distr } M N X) = \text{null_sets } N$

shows $\text{completion } (\text{distr } M N X) = \text{distr } M (\text{completion } N) X$

proof (*rule measure_eqI*)
show $\text{eq}: \text{sets } (\text{completion } (\text{distr } M N X)) = \text{sets } (\text{distr } M (\text{completion } N) X)$
by (*simp add: sets_completion null_sets*)

fix A **assume** $A: A \in \text{completion } (\text{distr } M N X)$
moreover have $A': A \in \text{completion } N$
using A **by** (*simp add: eq*)
moreover have $\text{main_part } (\text{distr } M N X) A \in \text{sets } N$
using $\text{main_part_sets}[OF A]$ **by** *simp*
moreover have $X -' A \cap \text{space } M = (X -' \text{main_part } (\text{distr } M N X) A \cap \text{space } M) \cup (X -' \text{null_part } (\text{distr } M N X) A \cap \text{space } M)$
using $\text{main_part_null_part_Un}[OF A]$ **by** *auto*
moreover have $X -' \text{null_part } (\text{distr } M N X) A \cap \text{space } M \in \text{null_sets } M$
using $X A$ **by** (*intro vimage_null_part_null_sets*) (*auto cong: distr_cong*)
ultimately show $\text{emeasure } (\text{completion } (\text{distr } M N X)) A = \text{emeasure } (\text{distr } M (\text{completion } N) X) A$
using X **by** (*auto simp: emeasure_distr measurable_completion null_sets measurable_completion2*
intro!: emeasure_Un_null_set[symmetric])

qed

lemma *distr_completion*: $X \in M \rightarrow_M N \implies \text{distr } (\text{completion } M) N X = \text{distr } M N X$

by (*intro measure_eqI*) (*auto simp: emeasure_distr measurable_completion*)

proposition (*in complete_measure*) *fmeasurable_inner_outer*:

$S \in \text{fmeasurable } M \iff$

$(\forall e > 0. \exists T \in \text{fmeasurable } M. \exists U \in \text{fmeasurable } M. T \subseteq S \wedge S \subseteq U \wedge |\text{measure } M T - \text{measure } M U| < e)$

$(\text{is } _ \iff ?\text{approx})$

proof *safe*

let $?\mu = \text{measure } M$ **let** $?D = \lambda T U. |?\mu T - ?\mu U|$

assume $?\text{approx}$

have $\exists A. \forall n. (\text{fst } (A n) \in \text{fmeasurable } M \wedge \text{snd } (A n) \in \text{fmeasurable } M \wedge \text{fst}$

```

(A n) ⊆ S ∧ S ⊆ snd (A n) ∧
  ?D (fst (A n)) (snd (A n)) < 1/Suc n) ∧ (fst (A n) ⊆ fst (A (Suc n)) ∧ snd
(A (Suc n)) ⊆ snd (A n))
  (is ∃ A. ∀ n. ?P n (A n) ∧ ?Q (A n) (A (Suc n)))
proof (intro dependent_nat_choice)
  show ∃ A. ?P 0 A
    using ‹?approx›[THEN spec, of 1] by auto
next
  fix A n assume ?P n A
  moreover
  from ‹?approx›[THEN spec, of 1/Suc (Suc n)]
  obtain T U where *: T ∈ fmeasurable M U ∈ fmeasurable M T ⊆ S S ⊆ U
  ?D T U < 1 / Suc (Suc n)
    by auto
  ultimately have ?μ T ≤ ?μ (T ∪ fst A) ?μ (U ∩ snd A) ≤ ?μ U
    ?μ T ≤ ?μ U ?μ (T ∪ fst A) ≤ ?μ (U ∩ snd A)
    by (auto intro!: measure_mono_fmeasurable)
  then have ?D (T ∪ fst A) (U ∩ snd A) ≤ ?D T U
    by auto
  also have ?D T U < 1/Suc (Suc n) by fact
  finally show ∃ B. ?P (Suc n) B ∧ ?Q A B
    using ‹?P n A› *
    by (intro exI[of _ (T ∪ fst A, U ∩ snd A)] conjI) auto
qed
then obtain A
  where lm: ⋀n. fst (A n) ∈ fmeasurable M ⋀n. snd (A n) ∈ fmeasurable M
    and set_bound: ⋀n. fst (A n) ⊆ S ⋀n. S ⊆ snd (A n)
    and mono: ⋀n. fst (A n) ⊆ fst (A (Suc n)) ⋀n. snd (A (Suc n)) ⊆ snd (A
n)
    and bound: ⋀n. ?D (fst (A n)) (snd (A n)) < 1/Suc n
    by metis

have INT_sA: (⋂ n. snd (A n)) ∈ fmeasurable M
  using lm by (intro fmeasurable_INT[of _ 0]) auto
have UN_fA: (⋃ n. fst (A n)) ∈ fmeasurable M
  using lm order_trans[OF set_bound(1) set_bound(2)[of 0]] by (intro fmea-
surable_UN[of _ _ snd (A 0)]) auto

have (λn. ?μ (fst (A n)) - ?μ (snd (A n))) ⟶ 0
  using bound
  by (subst tendsto_rabs_zero_iff[symmetric])
  (intro tendsto_sandwich[OF __ tendsto_const LIMSEQ_inverse_real_of_nat];
  auto intro!: always_eventually_less_imp_le simp: divide_inverse)
moreover
have (λn. ?μ (fst (A n)) - ?μ (snd (A n))) ⟶ ?μ (⋃ n. fst (A n)) - ?μ
(⋂ n. snd (A n))
proof (intro tendsto_diff Lim_measure_incseq Lim_measure_decseq)
  show range (λi. fst (A i)) ⊆ sets M range (λi. snd (A i)) ⊆ sets M
  incseq (λi. fst (A i)) decseq (λn. snd (A n))

```

using *mono lm* **by** (*auto simp: incseq_Suc_iff decseq_Suc_iff intro!: measure_mono_fmeasurable*)
show $\text{emeasure } M (\bigcup x. \text{fst } (A x)) \neq \infty$ $\text{emeasure } M (\text{snd } (A n)) \neq \infty$ **for** n
using *lm(2)[of n] UN_fA* **by** (*auto simp: fmeasurable_def*)
qed
ultimately have $0 = ?\mu (\bigcup n. \text{fst } (A n)) - ?\mu (\bigcap n. \text{snd } (A n))$
by (*rule LIMSEQ_unique*)

show $S \in \text{fmeasurable } M$
using *UN_fA INT_sA*
proof (*rule complete_sets_sandwich_fmeasurable*)
show $(\bigcup n. \text{fst } (A n)) \subseteq S$ $S \subseteq (\bigcap n. \text{snd } (A n))$
using *set_bound* **by** *auto*
show $?\mu (\bigcup n. \text{fst } (A n)) = ?\mu (\bigcap n. \text{snd } (A n))$
using *eq* **by** *auto*
qed
qed (*auto intro!: bexI[of _ S]*)

lemma (*in complete_measure*) *fmeasurable_measure_inner_outer*:

$(S \in \text{fmeasurable } M \wedge m = \text{measure } M S) \longleftrightarrow$
 $(\forall e > 0. \exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e < \text{measure } M T) \wedge$
 $(\forall e > 0. \exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U < m + e)$
(is ?lhs = ?rhs)

proof

assume *RHS: ?rhs*

then have $T: \bigwedge e. 0 < e \longrightarrow (\exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e < \text{measure } M T)$

and $U: \bigwedge e. 0 < e \longrightarrow (\exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U < m + e)$

by *auto*

have $S \in \text{fmeasurable } M$

proof (*subst fmeasurable_inner_outer, safe*)

fix $e::\text{real}$ **assume** $0 < e$

with *RHS* **obtain** $T U$ **where** $T: T \in \text{fmeasurable } M$ $T \subseteq S$ $m - e/2 < \text{measure } M T$

and $U: U \in \text{fmeasurable } M$ $S \subseteq U$ $\text{measure } M U < m + e/2$

by (*meson half_gt_zero*) $+$

moreover have $\text{measure } M U - \text{measure } M T < (m + e/2) - (m - e/2)$

by (*intro diff_strict_mono*) *fact+*

moreover have $\text{measure } M T \leq \text{measure } M U$

using $T U$ **by** (*intro measure_mono_fmeasurable*) *auto*

ultimately show $\exists T \in \text{fmeasurable } M. \exists U \in \text{fmeasurable } M. T \subseteq S \wedge S \subseteq U$
 $\wedge |\text{measure } M T - \text{measure } M U| < e$

apply (*rule_tac bexI[OF _ $\langle T \in \text{fmeasurable } M \rangle$]*)

apply (*rule_tac bexI[OF _ $\langle U \in \text{fmeasurable } M \rangle$]*)

by *auto*

qed

moreover have $m = \text{measure } M S$

using $\langle S \in \text{fmeasurable } M \rangle$ U [*of measure M S - m*] T [*of m - measure M S*]

by (cases m measure M S rule: linorder_cases)
 (auto simp: not_le[symmetric] measure_mono_fmeasurable)
 ultimately show ?lhs
 by simp
 qed (auto intro!: bexI[of _ S])

lemma (in complete_measure) null_sets_outer:
 $S \in \text{null_sets } M \iff (\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T < e)$
proof –
 have $S \in \text{null_sets } M \iff (S \in \text{fmeasurable } M \wedge 0 = \text{measure } M S)$
 by (auto simp: null_sets_def emeasure_eq_measure2 intro: fmeasurableI) (simp
 add: measure_def)
 also have ... = $(\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T < e)$
 unfolding fmeasurable_measure_inner_outer by auto
 finally show ?thesis .
 qed

lemma (in complete_measure) fmeasurable_measure_inner_outer_le:
 $(S \in \text{fmeasurable } M \wedge m = \text{measure } M S) \iff$
 $(\forall e > 0. \exists T \in \text{fmeasurable } M. T \subseteq S \wedge m - e \leq \text{measure } M T) \wedge$
 $(\forall e > 0. \exists U \in \text{fmeasurable } M. S \subseteq U \wedge \text{measure } M U \leq m + e)$ (is ?T1)
and null_sets_outer_le:
 $S \in \text{null_sets } M \iff (\forall e > 0. \exists T \in \text{fmeasurable } M. S \subseteq T \wedge \text{measure } M T \leq$
 $e)$ (is ?T2)
proof –
 have $e > 0 \wedge m - e/2 \leq t \implies m - e < t$
 $e > 0 \wedge t \leq m + e/2 \implies t < m + e$
 $e > 0 \iff e/2 > 0$
 for e t
 by auto
 then show ?T1 ?T2
 unfolding fmeasurable_measure_inner_outer null_sets_outer
 by (meson dense le_less_trans less_imp_le)+
 qed

lemma (in cld_measure)notin_sets_outer_measure_of_cover:
assumes $E: E \subseteq \text{space } M \ E \notin \text{sets } M$
shows $\exists B \in \text{sets } M. 0 < \text{emeasure } M B \wedge \text{emeasure } M B < \infty \wedge$
 $\text{outer_measure_of } M (B \cap E) = \text{emeasure } M B \wedge \text{outer_measure_of } M (B -$
 $E) = \text{emeasure } M B$
proof –
from locally_determined[OF $\langle E \subseteq \text{space } M \rangle \langle E \notin \text{sets } M \rangle$]
obtain F
where [measurable]: $F \in \text{sets } M$ **and** $\text{emeasure } M F < \infty \ E \cap F \notin \text{sets } M$
by blast
then obtain H H'
where H: $\text{measurable_envelope } M (F \cap E) H$ **and** H': $\text{measurable_envelope}$
 $M (F - E) H'$
using measurable_envelopeI_countable_cover[of F \cap E λ _. F M]

```

    measurable_envelopeI_countable_cover[of F - E λ_. F M]
  by auto
  note measurable_envelopeD(2)[OF H', measurable] measurable_envelopeD(2)[OF
H, measurable]

  from measurable_envelopeD(1)[OF H'] measurable_envelopeD(1)[OF H]
  have subset: F - H' ⊆ F ∩ E F ∩ E ⊆ F ∩ H
    by auto
  moreover define G where G = (F ∩ H) - (F - H')
  ultimately have G: G = F ∩ H ∩ H'
    by auto
  have emeasure M (F ∩ H) ≠ 0
  proof
    assume emeasure M (F ∩ H) = 0
    then have F ∩ H ∈ null_sets M
      by auto
    with ⟨E ∩ F ∉ sets M⟩ show False
      using complete[OF ⟨F ∩ E ⊆ F ∩ H⟩] by (auto simp: Int_commute)
  qed
  moreover
  have emeasure M (F - H') ≠ emeasure M (F ∩ H)
  proof
    assume emeasure M (F - H') = emeasure M (F ∩ H)
    with ⟨E ∩ F ∉ sets M⟩ emeasure_mono[of F ∩ H F M] ⟨emeasure M F < ∞⟩
    have F ∩ E ∈ sets M
      by (intro complete_sets_sandwich[OF _ _ subset]) auto
    with ⟨E ∩ F ∉ sets M⟩ show False
      by (simp add: Int_commute)
  qed
  moreover have emeasure M (F - H') ≤ emeasure M (F ∩ H)
    using subset by (intro emeasure_mono) auto
  ultimately have emeasure M G ≠ 0
    unfolding G_def using subset
    by (subst emeasure_Diff) (auto simp: top_unique diff_eq_0_iff_enreal)
  show ?thesis
  proof (intro beXI conjI)
    have emeasure M G ≤ emeasure M F
      unfolding G by (auto intro!: emeasure_mono)
    with ⟨emeasure M F < ∞⟩ show 0 < emeasure M G emeasure M G < ∞
      using ⟨emeasure M G ≠ 0⟩ by (auto simp: zero_less_iff_neq_zero)
    show [measurable]: G ∈ sets M
      unfolding G by auto
  have emeasure M G = outer_measure_of M (F ∩ H' ∩ (F ∩ E))
    using measurable_envelopeD(3)[OF H, of F ∩ H'] unfolding G by (simp
add: ac_simps)
    also have ... ≤ outer_measure_of M (G ∩ E)
      using measurable_envelopeD(1)[OF H] by (intro outer_measure_of_mono)
    (auto simp: G)

```

```

finally show outer_measure_of M (G ∩ E) = emeasure M G
using outer_measure_of_mono[of G ∩ E G M] by auto

have emeasure M G = outer_measure_of M (F ∩ H ∩ (F - E))
using measurable_envelopeD(3)[OF H', of F ∩ H] unfolding G by (simp
add: ac_simps)
also have ... ≤ outer_measure_of M (G - E)
using measurable_envelopeD(1)[OF H'] by (intro outer_measure_of_mono)
(auto simp: G)
finally show outer_measure_of M (G - E) = emeasure M G
using outer_measure_of_mono[of G - E G M] by auto
qed
qed

```

The following theorem is a specialization of D.H. Fremlin, Measure Theory vol 4I (413G). We only show one direction and do not use a inner regular family K .

```

lemma (in cld_measure) borel_measurable_cld:
fixes f :: 'a ⇒ real
assumes ⋀A a b. A ∈ sets M ⇒ 0 < emeasure M A ⇒ emeasure M A < ∞
⇒ a < b ⇒
  min (outer_measure_of M {x∈A. f x ≤ a}) (outer_measure_of M {x∈A. b
≤ f x}) < emeasure M A
shows f ∈ M →M borel
proof (rule ccontr)
let ?E = λa. {x∈space M. f x ≤ a} and ?F = λa. {x∈space M. a ≤ f x}

assume f ∉ M →M borel
then obtain a where ?E a ∉ sets M
unfolding borel_measurable_iff_le by blast
from notin_sets_outer_measure_of_cover[OF _ this]
obtain K
where K: K ∈ sets M 0 < emeasure M K emeasure M K < ∞
and eq1: outer_measure_of M (K ∩ ?E a) = emeasure M K
and eq2: outer_measure_of M (K - ?E a) = emeasure M K
by auto
then have me_K: measurable_envelope M (K ∩ ?E a) K
by (subst measurable_envelope_eq2) auto

define b where b n = a + inverse (real (Suc n)) for n
have (SUP n. outer_measure_of M (K ∩ ?F (b n))) = outer_measure_of M
(⋃n. K ∩ ?F (b n))
proof (intro SUP_outer_measure_of_incseq)
have x ≤ y ⇒ b y ≤ b x for x y
by (auto simp: b_def field_simps)
then show incseq (λn. K ∩ {x ∈ space M. b n ≤ f x})
by (auto simp: incseq_def intro: order_trans)
qed auto
also have (⋃n. K ∩ ?F (b n)) = K - ?E a

```

```

proof –
  have  $b \longrightarrow a$ 
    unfolding  $b\_def$  by (rule LIMSEQ_inverse_real_of_nat_add)
  then have  $\forall n. \neg b n \leq f x \implies f x \leq a$  for  $x$ 
    by (rule LIMSEQ_le_const) (auto intro: less_imp_le simp: not_le)
  moreover have  $\neg b n \leq a$  for  $n$ 
    by (auto simp: b_def)
  ultimately show ?thesis
    using  $\langle K \in sets\ M \rangle [THEN\ sets.sets\_into\_space]$  by (auto simp: subset_eq
intro: order_trans)
  qed
  finally have  $0 < (SUP\ n. outer\_measure\_of\ M\ (K \cap ?F\ (b\ n)))$ 
    using  $K$  by (simp add: eq2)
  then obtain  $n$  where  $pos\_b: 0 < outer\_measure\_of\ M\ (K \cap ?F\ (b\ n))$  and  $a < b\ n$ 
    unfolding less_SUP_iff by (auto simp: b_def)
  from measurable_envelopeI_countable_cover[ $of\ K \cap ?F\ (b\ n)\ \lambda_.\ K\ M$ ]  $K$ 
  obtain  $K'$  where  $K' \subseteq K$  and  $me\_K': measurable\_envelope\ M\ (K \cap ?F\ (b\ n))\ K'$ 
    by auto
  then have  $K'\_le\_K: emeasure\ M\ K' \leq emeasure\ M\ K$ 
    by (intro emeasure_mono  $K$ )
  have  $K' \in sets\ M$ 
    using  $me\_K'$  by (rule measurable_envelopeD)

  have  $min\ (outer\_measure\_of\ M\ \{x \in K'.\ f\ x \leq a\})\ (outer\_measure\_of\ M\ \{x \in K'.\ b\ n \leq f\ x\}) < emeasure\ M\ K'$ 
    proof (rule assms)
      show  $0 < emeasure\ M\ K'\ emeasure\ M\ K' < \infty$ 
        using measurable_envelopeD2[ $OF\ me\_K'$ ]  $pos\_b\ K\ K'\_le\_K$  by auto
      qed fact+
    also have  $\{x \in K'.\ f\ x \leq a\} = K' \cap (K \cap ?E\ a)$ 
      using  $\langle K' \in sets\ M \rangle [THEN\ sets.sets\_into\_space]$   $\langle K' \subseteq K \rangle$  by auto
    also have  $\{x \in K'.\ b\ n \leq f\ x\} = K' \cap (K \cap ?F\ (b\ n))$ 
      using  $\langle K' \in sets\ M \rangle [THEN\ sets.sets\_into\_space]$   $\langle K' \subseteq K \rangle$  by auto
    finally have  $min\ (emeasure\ M\ K)\ (emeasure\ M\ K') < emeasure\ M\ K'$ 
      unfolding
        measurable_envelopeD(3)[ $OF\ me\_K\ \langle K' \in sets\ M \rangle, symmetric$ ]
        measurable_envelopeD(3)[ $OF\ me\_K'\ \langle K' \in sets\ M \rangle, symmetric$ ]
      using  $\langle K' \subseteq K \rangle$  by (simp add: Int_absorb1 Int_absorb2)
    with  $K'\_le\_K$  show False
      by (auto simp: min_def split: if_split_asm)
    qed
  end

```

7.11 Regularity of Measures

theory *Regularity*

```
imports Measure_Space Borel_Space
begin
```

```
theorem
```

```
  fixes M::'a::{second_countable_topology, complete_space} measure
  assumes sb: sets M = sets borel
  assumes emeasure M (space M)  $\neq$   $\infty$ 
  assumes B  $\in$  sets borel
  shows inner_regular: emeasure M B =
    (SUP K  $\in$  {K. K  $\subseteq$  B  $\wedge$  compact K}. emeasure M K) (is ?inner B)
  and outer_regular: emeasure M B =
    (INF U  $\in$  {U. B  $\subseteq$  U  $\wedge$  open U}. emeasure M U) (is ?outer B)
proof -
  have Us: UNIV = space M by (metis assms(1) sets_eq_imp_space_eq space_borel)
  hence sU: space M = UNIV by simp
  interpret finite_measure M by rule fact
  have approx_inner:  $\bigwedge A. A \in$  sets M  $\implies$ 
    ( $\bigwedge e. e > 0 \implies \exists K. K \subseteq A \wedge$  compact K  $\wedge$  emeasure M A  $\leq$  emeasure M K
  + ennreal e)  $\implies$  ?inner A
    by (rule ennreal_approx_SUP)
    (force intro!: emeasure_mono simp: compact_imp_closed emeasure_eq_measure)+
  have approx_outer:  $\bigwedge A. A \in$  sets M  $\implies$ 
    ( $\bigwedge e. e > 0 \implies \exists B. A \subseteq B \wedge$  open B  $\wedge$  emeasure M B  $\leq$  emeasure M A +
  ennreal e)  $\implies$  ?outer A
    by (rule ennreal_approx_INF)
    (force intro!: emeasure_mono simp: emeasure_eq_measure sb)+
  from countable_dense_setE obtain X :: 'a set
    where X: countable X  $\wedge$  Y :: 'a set. open Y  $\implies$  Y  $\neq$  {}  $\implies \exists d \in X. d \in Y$ 
    by auto
  {
    fix r::real assume r > 0 hence  $\bigwedge y. \text{open } (\text{ball } y \text{ } r) \wedge y. \text{ball } y \text{ } r \neq \{\}$  by auto
    with X(2)[OF this]
    have x: space M = ( $\bigcup x \in X. \text{cball } x \text{ } r$ )
      by (auto simp add: sU) (metis dist_commute order_less_imp_le)
    let ?U =  $\bigcup k. (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)$ 
    have ( $\lambda k. \text{emeasure } M (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)$ )  $\longrightarrow$  M
      ?U
      by (rule Lim_emeasure_incseq) (auto intro!: borel_closed bexI simp: inc-
      seq_def Us sb)
    also have ?U = space M
    proof safe
      fix x from X(2)[OF open_ball[of x r]]  $\langle r > 0 \rangle$  obtain d where d:  $d \in X \text{ } d \in$ 
      ball x r by auto
      show x  $\in$  ?U
        using X(1) d
        by simp (auto intro!: exI [where x = to_nat_on X d] simp: dist_commute
        Bex_def)
    qed (simp add: sU)
    finally have ( $\lambda k. M (\bigcup n \in \{0..k\}. \text{cball } (\text{from\_nat\_into } X \text{ } n) \text{ } r)$ )  $\longrightarrow$  M
```

```

(space M) .
} note M_space = this
{
  fix e :: real and n :: nat assume e > 0 n > 0
  hence 1/n > 0 e * 2 powr - n > 0 by (auto)
  from M_space[OF ‹1/n>0›
  have (λk. measure M (∪ i∈{0..k}. cball (from_nat_into X i) (1/real n)))
  → measure M (space M)
  unfolding emeasure_eq_measure by (auto)
  from metric_LIMSEQ_D[OF this ‹0 < e * 2 powr - n›]
  obtain k where dist (measure M (∪ i∈{0..k}. cball (from_nat_into X i)
  (1/real n))) (measure M (space M)) <
  e * 2 powr - n
  by auto
  hence measure M (∪ i∈{0..k}. cball (from_nat_into X i) (1/real n)) ≥
  measure M (space M) - e * 2 powr - real n
  by (auto simp: dist_real_def)
  hence ∃ k. measure M (∪ i∈{0..k}. cball (from_nat_into X i) (1/real n)) ≥
  measure M (space M) - e * 2 powr - real n ..
} note k=this
hence ∀ e∈{0<..}. ∀ (n::nat)∈{0<..}. ∃ k.
  measure M (∪ i∈{0..k}. cball (from_nat_into X i) (1/real n)) ≥ measure M
(space M) - e * 2 powr - real n
by blast
then obtain k where k: ∀ e∈{0<..}. ∀ n∈{0<..}. measure M (space M) - e *
2 powr - real (n::nat)
≤ measure M (∪ i∈{0..k e n}. cball (from_nat_into X i) (1 / n))
by metis
hence k: ∧ e n. e > 0 ⇒ n > 0 ⇒ measure M (space M) - e * 2 powr - n
≤ measure M (∪ i∈{0..k e n}. cball (from_nat_into X i) (1 / n))
unfolding Ball_def by blast
have approx_space:
  ∃ K ∈ {K. K ⊆ space M ∧ compact K}. emeasure M (space M) ≤ emeasure
M K + ennreal e
(is ?thesis e) if 0 < e for e :: real
proof -
  define B where [abs_def]:
    B n = (∪ i∈{0..k e (Suc n)}. cball (from_nat_into X i) (1 / Suc n)) for n
  have ∧ n. closed (B n) by (auto simp: B_def)
  hence [simp]: ∧ n. B n ∈ sets M by (simp add: sb)
  from k[OF ‹e > 0› zero_less_Suc]
  have ∧ n. measure M (space M) - measure M (B n) ≤ e * 2 powr - real (Suc
n)
  by (simp add: algebra_simps B_def finite_measure_compl)
  hence B_compl_le: ∧ n::nat. measure M (space M - B n) ≤ e * 2 powr -
real (Suc n)
  by (simp add: finite_measure_compl)
  define K where K = (∩ n. B n)
  from ‹closed (B _)› have closed K by (auto simp: K_def)

```

```

hence [simp]:  $K \in \text{sets } M$  by (simp add: sb)
have  $\text{measure } M (\text{space } M) - \text{measure } M K = \text{measure } M (\text{space } M - K)$ 
  by (simp add: finite_measure_compl)
also have  $\dots = \text{emeasure } M (\bigcup n. \text{space } M - B n)$  by (auto simp:  $K\_def$ 
emeasure_eq_measure)
also have  $\dots \leq (\sum n. \text{emeasure } M (\text{space } M - B n))$ 
  by (rule emeasure_subadditive_countably) (auto simp: summable_def)
also have  $\dots \leq (\sum n. \text{ennreal } (e * 2^{\text{pow } n} - \text{real } (\text{Suc } n)))$ 
  using  $B\_compl\_le$  by (intro suminf_le) (simp_all add: emeasure_eq_measure
ennreal_leI)
also have  $\dots \leq (\sum n. \text{ennreal } (e * (1 / 2)^{\wedge \text{Suc } n}))$ 
  by (simp add: pow_minus pow_realpow field_simps del: of_nat_Suc)
also have  $\dots = \text{ennreal } e * (\sum n. \text{ennreal } ((1 / 2)^{\wedge \text{Suc } n}))$ 
  unfolding ennreal_power[symmetric]
  using  $\langle 0 < e \rangle$ 
by (simp add: ac_simps ennreal_mult' divide_ennreal[symmetric] divide_ennreal_def
ennreal_power[symmetric])
also have  $\dots = e$ 
  by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
finally have  $\text{measure } M (\text{space } M) \leq \text{measure } M K + e$ 
  using  $\langle 0 < e \rangle$  by simp
hence  $\text{emeasure } M (\text{space } M) \leq \text{emeasure } M K + e$ 
  using  $\langle 0 < e \rangle$  by (simp add: emeasure_eq_measure flip: ennreal_plus)
moreover have compact  $K$ 
  unfolding compact_eq_totally_bounded
proof safe
  show complete  $K$  using  $\langle \text{closed } K \rangle$  by (simp add: complete_eq_closed)
  fix  $e'::\text{real}$  assume  $0 < e'$ 
  then obtain  $n$  where  $n: 1 / \text{real } (\text{Suc } n) < e'$  by (rule nat_approx_posE)
  let  $?k = \text{from\_nat\_into } X \text{ ' } \{0..k \ e (\text{Suc } n)\}$ 
  have finite  $?k$  by simp
  moreover have  $K \subseteq (\bigcup x \in ?k. \text{ball } x \ e')$  unfolding  $K\_def \ B\_def$  using  $n$ 
by force
  ultimately show  $\exists k. \text{finite } k \wedge K \subseteq (\bigcup x \in k. \text{ball } x \ e')$  by blast
qed
ultimately
show  $?thesis$  by (auto simp: sU)
qed
{ fix  $A::'a \text{ set}$  assume closed  $A$  hence  $A \in \text{sets borel}$  by (simp add: compact_imp_closed)
hence [simp]:  $A \in \text{sets } M$  by (simp add: sb)
have  $?inner \ A$ 
proof (rule approx_inner)
  fix  $e::\text{real}$  assume  $e > 0$ 
  from approx_space[OF this] obtain  $K$  where
     $K: K \subseteq \text{space } M$  compact  $K$   $\text{emeasure } M (\text{space } M) \leq \text{emeasure } M K + e$ 
  by (auto simp: emeasure_eq_measure)
  hence [simp]:  $K \in \text{sets } M$  by (simp add: sb compact_imp_closed)
  have  $\text{measure } M A - \text{measure } M (A \cap K) = \text{measure } M (A - A \cap K)$ 

```

```

    by (subst finite_measure_Diff) auto
  also have  $A - A \cap K = A \cup K - K$  by auto
  also have  $\text{measure } M \dots = \text{measure } M (A \cup K) - \text{measure } M K$ 
    by (subst finite_measure_Diff) auto
  also have  $\dots \leq \text{measure } M (\text{space } M) - \text{measure } M K$ 
    by (simp add: emeasure_eq_measure sU sb finite_measure_mono)
  also have  $\dots \leq e$ 
    using  $K \langle 0 < e \rangle$  by (simp add: emeasure_eq_measure flip: ennreal_plus)
  finally have  $\text{emeasure } M A \leq \text{emeasure } M (A \cap K) + \text{ennreal } e$ 
    using  $\langle 0 < e \rangle$  by (simp add: emeasure_eq_measure algebra_simps flip:
ennreal_plus)
  moreover have  $A \cap K \subseteq A$  compact  $(A \cap K)$  using  $\langle \text{closed } A \rangle \langle \text{compact } K \rangle$  by auto
  ultimately show  $\exists K \subseteq A. \text{compact } K \wedge \text{emeasure } M A \leq \text{emeasure } M K + \text{ennreal } e$ 
    by blast
qed simp
have ?outer A
proof cases
  assume  $A \neq \{\}$ 
  let ?G =  $\lambda d. \{x. \text{infdist } x A < d\}$ 
  {
    fix d
    have  $?G d = (\lambda x. \text{infdist } x A) -' \{.. < d\}$  by auto
    also have open ...
      by (intro continuous_open_vimage) (auto intro!: continuous_infdist
continuous_ident)
    finally have open (?G d) .
  } note open_G = this
  from in_closed_iff_infdist_zero[OF  $\langle \text{closed } A \rangle \langle A \neq \{\} \rangle$ ]
  have  $A = \{x. \text{infdist } x A = 0\}$  by auto
  also have  $\dots = (\bigcap i. ?G (1/\text{real } (\text{Suc } i)))$ 
  proof (auto simp del: of_nat_Suc, rule ccontr)
    fix x
    assume  $\text{infdist } x A \neq 0$ 
    then have  $\text{pos: infdist } x A > 0$  using infdist_nonneg[of x A] by simp
    then obtain n where  $n: 1 / \text{real } (\text{Suc } n) < \text{infdist } x A$  by (rule
nat_approx_posE)
    assume  $\forall i. \text{infdist } x A < 1 / \text{real } (\text{Suc } i)$ 
    then have  $\text{infdist } x A < 1 / \text{real } (\text{Suc } n)$  by auto
    with n show False by simp
  qed
  also have  $M \dots = (\text{INF } n. \text{emeasure } M (?G (1 / \text{real } (\text{Suc } n))))$ 
  proof (rule INF_emeasure_decseq[symmetric], safe)
    fix  $i::\text{nat}$ 
    from open_G[of  $1 / \text{real } (\text{Suc } i)$ ]
    show  $?G (1 / \text{real } (\text{Suc } i)) \in \text{sets } M$  by (simp add: sb borel_open)
  next
    show decseq  $(\lambda i. \{x. \text{infdist } x A < 1 / \text{real } (\text{Suc } i)\})$ 

```



```

      by (auto intro: less_trans intro!: divide_strict_left_mono
          simp: decseq_def le_eq_less_or_eq)
    qed simp
  finally
  have  $\text{emeasure } M A = (\text{INF } n. \text{emeasure } M \{x. \text{infdist } x A < 1 / \text{real } (Suc\ n)\})$  .
  moreover
  have  $\dots \geq (\text{INF } U \in \{U. A \subseteq U \wedge \text{open } U\}. \text{emeasure } M U)$ 
  proof (intro INF_mono)
    fix m
    have  $?G (1 / \text{real } (Suc\ m)) \in \{U. A \subseteq U \wedge \text{open } U\}$  using open_G by
  auto
  moreover have  $M (?G (1 / \text{real } (Suc\ m))) \leq M (?G (1 / \text{real } (Suc\ m)))$ 
  by simp
  ultimately show  $\exists U \in \{U. A \subseteq U \wedge \text{open } U\}. \text{emeasure } M U \leq \text{emeasure } M \{x. \text{infdist } x A < 1 / \text{real } (Suc\ m)\}$ 
    by blast
  qed
  moreover
  have  $\text{emeasure } M A \leq (\text{INF } U \in \{U. A \subseteq U \wedge \text{open } U\}. \text{emeasure } M U)$ 
  by (rule INF_greatest) (auto intro!: emeasure_mono simp: sb)
  ultimately show ?thesis by simp
  qed (auto intro!: INF_eqI)
  note  $\langle ?inner\ A \rangle \langle ?outer\ A \rangle$  }
  note closed_in_D = this
  from  $\langle B \in \text{sets borel} \rangle$ 
  have Int_stable (Collect closed) Collect closed  $\subseteq$  Pow UNIV  $B \in \text{sigma\_sets UNIV (Collect closed)}$ 
  by (auto simp: Int_stable_def borel_eq_closed)
  then show ?inner B ?outer B
  proof (induct B rule: sigma_sets_induct_disjoint)
    case empty
    { case 1 show ?case by (intro SUP_eqI[symmetric]) auto }
    { case 2 show ?case by (intro INF_eqI[symmetric]) (auto elim!: meta_allE[of _ {}]) }
  }
  next
  case (basic B)
  { case 1 from basic closed_in_D show ?case by auto }
  { case 2 from basic closed_in_D show ?case by auto }
  next
  case (compl B)
  note inner = compl(2) and outer = compl(3)
  from compl have [simp]:  $B \in \text{sets } M$  by (auto simp: sb borel_eq_closed)
  case 2
  have  $M (\text{space } M - B) = M (\text{space } M) - \text{emeasure } M B$  by (auto simp: emeasure_compl)
  also have  $\dots = (\text{INF } K \in \{K. K \subseteq B \wedge \text{compact } K\}. M (\text{space } M) - M K)$ 
  by (subst ennreal_SUP_const_minus) (auto simp: less_top[symmetric] inner)
  also have  $\dots = (\text{INF } U \in \{U. U \subseteq B \wedge \text{compact } U\}. M (\text{space } M - U))$ 

```

```

    by (auto simp add: emeasure_compl sb compact_imp_closed)
  also have ... ≥ (INF U∈{U. U ⊆ B ∧ closed U}. M (space M - U))
    by (rule INF_superset_mono) (auto simp add: compact_imp_closed)
  also have (INF U∈{U. U ⊆ B ∧ closed U}. M (space M - U)) =
    (INF U∈{U. space M - B ⊆ U ∧ open U}. emeasure M U)
  apply (rule arg_cong [of _ _ Inf])
  using sU
  apply (auto simp add: image_iff)
  apply (rule exI [of _ UNIV - y for y])
  apply safe
  apply (auto simp add: double_diff)
  done
  finally have
    (INF U∈{U. space M - B ⊆ U ∧ open U}. emeasure M U) ≤ emeasure M
(space M - B) .
  moreover have
    (INF U∈{U. space M - B ⊆ U ∧ open U}. emeasure M U) ≥ emeasure M
(space M - B)
  by (auto simp: sb sU intro!: INF_greatest emeasure_mono)
  ultimately show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF
sb])

  case 1
  have M (space M - B) = M (space M) - emeasure M B by (auto simp:
emeasure_compl)
  also have ... = (SUP U∈{U. B ⊆ U ∧ open U}. M (space M) - M U)
    unfolding outer by (subst ennreal_INF_const_minus) auto
  also have ... = (SUP U∈{U. B ⊆ U ∧ open U}. M (space M - U))
    by (auto simp add: emeasure_compl sb compact_imp_closed)
  also have ... = (SUP K∈{K. K ⊆ space M - B ∧ closed K}. emeasure M
K)
    unfolding SUP_image [of _ λu. space M - u _, symmetric, unfolded
comp_def]
    apply (rule arg_cong [of _ _ Sup])
    using sU apply (auto intro!: imageI)
    done
  also have ... = (SUP K∈{K. K ⊆ space M - B ∧ compact K}. emeasure M
K)
  proof (safe intro!: antisym SUP_least)
    fix K assume closed K K ⊆ space M - B
    from closed_in_D[OF ⟨closed K⟩]
    have K_inner: emeasure M K = (SUP K∈{Ka. Ka ⊆ K ∧ compact Ka}.
emeasure M K) by simp
    show emeasure M K ≤ (SUP K∈{K. K ⊆ space M - B ∧ compact K}.
emeasure M K)
      unfolding K_inner using ⟨K ⊆ space M - B⟩
      by (auto intro!: SUP_upper SUP_least)
  qed (fastforce intro!: SUP_least SUP_upper simp: compact_imp_closed)
  finally show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF

```

```

sb])
next
  case (union D)
  then have range D  $\subseteq$  sets M by (auto simp: sb borel_eq_closed)
  with union have M[symmetric]:  $(\sum i. M (D i)) = M (\bigcup i. D i)$  by (intro
suminf_emeasure)
  also have  $(\lambda n. \sum i < n. M (D i)) \longrightarrow (\sum i. M (D i))$ 
  by (intro summable_LIMSEQ) auto
  finally have measure_LIMSEQ:  $(\lambda n. \sum i < n. \text{measure } M (D i)) \longrightarrow \text{measure } M (\bigcup i. D i)$ 
  by (simp add: emeasure_eq_measure sum_nonneg)
  have  $(\bigcup i. D i) \in \text{sets } M$  using  $\langle \text{range } D \subseteq \text{sets } M \rangle$  by auto

  case 1
  show ?case
  proof (rule approx_inner)
    fix e::real assume e > 0
    with measure_LIMSEQ
    have  $\exists n0. \forall n \geq n0. |(\sum i < n. \text{measure } M (D i)) - \text{measure } M (\bigcup x. D x)| <$ 
e/2
  by (auto simp: lim_sequentially dist_real_def simp del: less_divide_eq_numeral1)
  hence  $\exists n0. |(\sum i < n0. \text{measure } M (D i)) - \text{measure } M (\bigcup x. D x)| < e/2$ 
by auto
  then obtain n0 where n0:  $|(\sum i < n0. \text{measure } M (D i)) - \text{measure } M (\bigcup i. D i)| < e/2$ 
  unfolding choice_iff by blast
  have ennreal  $(\sum i < n0. \text{measure } M (D i)) = (\sum i < n0. M (D i))$ 
  by (auto simp add: emeasure_eq_measure)
  also have  $\dots \leq (\sum i. M (D i))$  by (rule sum_le_suminf) auto
  also have  $\dots = M (\bigcup i. D i)$  by (simp add: M)
  also have  $\dots = \text{measure } M (\bigcup i. D i)$  by (simp add: emeasure_eq_measure)
  finally have n0:  $\text{measure } M (\bigcup i. D i) - (\sum i < n0. \text{measure } M (D i)) < e/2$ 
  using n0 by (auto simp: sum_nonneg)
  have  $\forall i. \exists K. K \subseteq D i \wedge \text{compact } K \wedge \text{emeasure } M (D i) \leq \text{emeasure } M K + e/(2*\text{Suc } n0)$ 
  proof
    fix i
    from  $\langle 0 < e \rangle$  have  $0 < e/(2*\text{Suc } n0)$  by simp
    have  $\text{emeasure } M (D i) = (\text{SUP } K \in \{K. K \subseteq (D i) \wedge \text{compact } K\}. \text{emeasure } M K)$ 
  using union by blast
  from SUP_approx_ennreal[OF  $\langle 0 < e/(2*\text{Suc } n0) \rangle$  _ this]
  show  $\exists K. K \subseteq D i \wedge \text{compact } K \wedge \text{emeasure } M (D i) \leq \text{emeasure } M K + e/(2*\text{Suc } n0)$ 
  by (auto simp: emeasure_eq_measure intro: less_imp_le compact_empty)
  qed
  then obtain K where K:  $\bigwedge i. K i \subseteq D i \wedge \bigwedge i. \text{compact } (K i)$ 
 $\bigwedge i. \text{emeasure } M (D i) \leq \text{emeasure } M (K i) + e/(2*\text{Suc } n0)$ 
  unfolding choice_iff by blast

```

```

let ?K =  $\bigcup_{i \in \{..<n0\}}$ . K i
have disjoint_family_on K {..<n0} using K <disjoint_family D>
  unfolding disjoint_family_on_def by blast
hence mK: measure M ?K =  $(\sum_{i < n0}$ . measure M (K i)) using K
by (intro finite_measure_finite_Union) (auto simp: sb_compact_imp_closed)
have measure M ( $\bigcup i$ . D i) <  $(\sum_{i < n0}$ . measure M (D i)) + e/2 using n0
by simp
  also have  $(\sum_{i < n0}$ . measure M (D i))  $\leq$   $(\sum_{i < n0}$ . measure M (K i)) +
e/(2*Suc n0))
  using K <0 < e>
  by (auto intro: sum_mono simp: emeasure_eq_measure simp flip: ennreal_plus)
  also have ... =  $(\sum_{i < n0}$ . measure M (K i)) +  $(\sum_{i < n0}$ . e/(2*Suc n0))
  by (simp add: sum.distrib)
  also have ...  $\leq$   $(\sum_{i < n0}$ . measure M (K i)) + e / 2 using <0 < e>
  by (auto simp: field_simps intro!: mult_left_mono)
  finally
  have measure M ( $\bigcup i$ . D i) <  $(\sum_{i < n0}$ . measure M (K i)) + e / 2 + e / 2
  by auto
  hence M ( $\bigcup i$ . D i) < M ?K + e
  using <0 < e> by (auto simp: mK emeasure_eq_measure sum_nonneg ennreal_less_iff simp flip: ennreal_plus)
  moreover
  have ?K  $\subseteq$  ( $\bigcup i$ . D i) using K by auto
  moreover
  have compact ?K using K by auto
  ultimately
  have ?K  $\subseteq$  ( $\bigcup i$ . D i)  $\wedge$  compact ?K  $\wedge$  emeasure M ( $\bigcup i$ . D i)  $\leq$  emeasure M
?K + ennreal e by simp
  thus  $\exists K \subseteq \bigcup i$ . D i. compact K  $\wedge$  emeasure M ( $\bigcup i$ . D i)  $\leq$  emeasure M K +
ennreal e ..
qed fact
case 2
show ?case
proof (rule approx_outer[OF <( $\bigcup i$ . D i)  $\in$  sets M>])
  fix e::real assume e > 0
  have  $\forall i::nat$ .  $\exists U$ . D i  $\subseteq$  U  $\wedge$  open U  $\wedge$  e/(2 powr Suc i) > emeasure M U
– emeasure M (D i)
  proof
    fix i::nat
    from <0 < e> have 0 < e/(2 powr Suc i) by simp
    have emeasure M (D i) = (INF U  $\in$  {U. (D i)  $\subseteq$  U  $\wedge$  open U}. emeasure
M U)
    using union by blast
    from INF_approx_ennreal[OF <0 < e/(2 powr Suc i)> this]
    show  $\exists U$ . D i  $\subseteq$  U  $\wedge$  open U  $\wedge$  e/(2 powr Suc i) > emeasure M U –
emeasure M (D i)
    using <0 < e>
    by (auto simp: emeasure_eq_measure sum_nonneg ennreal_less_iff

```

```

ennreal_minus
  finite_measure_mono sb
  simp flip: ennreal_plus)
qed
then obtain U where U:  $\bigwedge i. D\ i \subseteq U\ i \wedge i. \text{open}\ (U\ i)$ 
   $\bigwedge i. e/(2\ \text{powr}\ \text{Suc}\ i) > \text{emeasure}\ M\ (U\ i) - \text{emeasure}\ M\ (D\ i)$ 
  unfolding choice_iff by blast
let ?U =  $\bigcup i. U\ i$ 
have ennreal (measure M ?U - measure M ( $\bigcup i. D\ i$ )) = M ?U - M ( $\bigcup i.$ 
D i)
  using U(1,2)
  by (subst ennreal_minus[symmetric])
  (auto intro!: finite_measure_mono simp: sb emeasure_eq_measure)
also have ... = M (?U - ( $\bigcup i. D\ i$ )) using U  $\langle \bigcup i. D\ i \in \text{sets}\ M \rangle$ 
  by (subst emeasure_Diff) (auto simp: sb)
also have ...  $\leq$  M ( $\bigcup i. U\ i - D\ i$ ) using U  $\langle \text{range}\ D \subseteq \text{sets}\ M \rangle$ 
  by (intro emeasure_mono) (auto simp: sb intro!: sets.countable_nat_UN
sets.Diff)
also have ...  $\leq$  ( $\sum i. M\ (U\ i - D\ i)$ ) using U  $\langle \text{range}\ D \subseteq \text{sets}\ M \rangle$ 
  by (intro emeasure_subadditive_countably) (auto intro!: sets.Diff simp: sb)
also have ...  $\leq$  ( $\sum i. \text{ennreal}\ e/(2\ \text{powr}\ \text{Suc}\ i)$ ) using U  $\langle \text{range}\ D \subseteq \text{sets}$ 
M  $\rangle$ 
  using  $\langle 0 < e \rangle$ 
  by (intro suminf_le, subst emeasure_Diff)
  (auto simp: emeasure_Diff emeasure_eq_measure sb ennreal_minus
  finite_measure_mono divide_ennreal ennreal_less_iff
  intro: less_imp_le)
also have ...  $\leq$  ( $\sum n. \text{ennreal}\ (e * (1 / 2) ^ \text{Suc}\ n)$ )
  using  $\langle 0 < e \rangle$ 
  by (simp add: powr_minus powr_realpow field_simps divide_ennreal del:
of_nat_Suc)
also have ... = ennreal e * ( $\sum n. \text{ennreal}\ ((1 / 2) ^ \text{Suc}\ n)$ )
  unfolding ennreal_power[symmetric]
  using  $\langle 0 < e \rangle$ 
  by (simp add: ac_simps ennreal_mult' divide_ennreal[symmetric] di-
vide_ennreal_def
  ennreal_power[symmetric])
also have ... = ennreal e
  by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
finally have emeasure M ?U  $\leq$  emeasure M ( $\bigcup i. D\ i$ ) + ennreal e
  using  $\langle 0 < e \rangle$  by (simp add: emeasure_eq_measure flip: ennreal_plus)
moreover
have ( $\bigcup i. D\ i$ )  $\subseteq$  ?U using U by auto
moreover
have open ?U using U by auto
ultimately
have ( $\bigcup i. D\ i$ )  $\subseteq$  ?U  $\wedge$  open ?U  $\wedge$  emeasure M ?U  $\leq$  emeasure M ( $\bigcup i. D$ 
i) + ennreal e by simp
thus  $\exists B. (\bigcup i. D\ i) \subseteq B \wedge \text{open}\ B \wedge \text{emeasure}\ M\ B \leq \text{emeasure}\ M\ (\bigcup i. D$ 

```

2466

```
i) + ennreal e ..
  qed
  qed
qed

end
```

7.12 Lebesgue Measure

```
theory Lebesgue_Measure
imports
  Finite_Product_Measure
  Caratheodory
  Complete_Measure
  Summation_Tests
  Regularity
begin

lemma measure_eqI_lessThan:
  fixes M N :: real measure
  assumes sets: sets M = sets borel sets N = sets borel
  assumes fin:  $\bigwedge x. \text{emeasure } M \{x <..\} < \infty$ 
  assumes  $\bigwedge x. \text{emeasure } M \{x <..\} = \text{emeasure } N \{x <..\}$ 
  shows M = N
proof (rule measure_eqI_generator_eq_countable)
  let ?LT =  $\lambda a::\text{real}. \{a <..\}$  let ?E = range ?LT
  show Int_stable ?E
    by (auto simp: Int_stable_def lessThan_Int_lessThan)

  show ?E  $\subseteq$  Pow UNIV sets M = sigma_sets UNIV ?E sets N = sigma_sets
  UNIV ?E
    unfolding sets borel_Ioi by auto

  show ?LT'Rats  $\subseteq$  ?E ( $\bigcup i \in \text{Rats}. ?LT i$ ) = UNIV  $\bigwedge a. a \in ?LT'Rats \implies \text{emeasure}$ 
  M a  $\neq \infty$ 
    using fin by (auto intro: Rats_no_bot_less simp: less_top)
qed (auto intro: assms countable_rat)
```

7.12.1 Measures defined by monotonous functions

Every right-continuous and nondecreasing function gives rise to a measure on the reals:

```
definition interval_measure :: (real  $\Rightarrow$  real)  $\Rightarrow$  real measure where
  interval_measure F =
    extend_measure UNIV  $\{(a, b). a \leq b\}$  ( $\lambda(a, b). \{a <..b\}$ ) ( $\lambda(a, b). \text{ennreal } (F$ 
  b - F a))
```

```
lemma emeasure_interval_measure_Ioc:
```

```

assumes  $a < b$ 
assumes  $mono\_F: \bigwedge x y. x \leq y \implies F x \leq F y$ 
assumes  $right\_cont\_F: \bigwedge a. continuous (at\_right a) F$ 
shows  $emeasure (interval\_measure F) \{a <.. b\} = F b - F a$ 
proof (rule  $extend\_measure\_caratheodory\_pair[OF interval\_measure\_def \langle a \leq b \rangle]$ )
  show  $semiring\_of\_sets UNIV \{\{a <.. b\} \mid a b :: real. a \leq b\}$ 
  proof (unfold  $locales, safe$ )
    fix  $a b c d :: real$  assume  $*$ :  $a \leq b c \leq d$ 
    then show  $\exists C \subseteq \{\{a <.. b\} \mid a b. a \leq b\}. finite C \wedge disjoint C \wedge \{a <.. b\} - \{c <.. d\} = \bigcup C$ 
    proof  $cases$ 
      let  $?C = \{\{a <.. b\}\}$ 
      assume  $b < c \vee d \leq a \vee d \leq c$ 
      with  $*$  have  $?C \subseteq \{\{a <.. b\} \mid a b. a \leq b\} \wedge finite ?C \wedge disjoint ?C \wedge \{a <.. b\} - \{c <.. d\} = \bigcup ?C$ 
      by ( $auto simp add: disjoint\_def$ )
      thus  $?thesis ..$ 
    next
      let  $?C = \{\{a <.. c\}, \{d <.. b\}\}$ 
      assume  $\neg (b < c \vee d \leq a \vee d \leq c)$ 
      with  $*$  have  $?C \subseteq \{\{a <.. b\} \mid a b. a \leq b\} \wedge finite ?C \wedge disjoint ?C \wedge \{a <.. b\} - \{c <.. d\} = \bigcup ?C$ 
      by ( $auto simp add: disjoint\_def Ioc\_inj (metis linear)+$ )
      thus  $?thesis ..$ 
    qed
  qed ( $auto simp: Ioc\_inj, metis linear$ )
next
  fix  $l r :: nat \implies real$  and  $a b :: real$ 
  assume  $l\_r[simp]: \bigwedge n. l n \leq r n$  and  $a \leq b$  and  $disj: disjoint\_family (\lambda n. \{l n <.. r n\})$ 
  assume  $lr\_eq\_ab: (\bigcup i. \{l i <.. r i\}) = \{a <.. b\}$ 

  have [ $intro, simp$ ]:  $\bigwedge a b. a \leq b \implies F a \leq F b$ 
  by ( $auto intro!: l\_r mono\_F$ )

  { fix  $S :: nat set$  assume  $finite S$ 
    moreover note  $\langle a \leq b \rangle$ 
    moreover have  $\bigwedge i. i \in S \implies \{l i <.. r i\} \subseteq \{a <.. b\}$ 
    unfolding  $lr\_eq\_ab[symmetric]$  by  $auto$ 
    ultimately have  $(\sum_{i \in S}. F (r i) - F (l i)) \leq F b - F a$ 
    proof ( $induction S arbitrary: a rule: finite\_psubset\_induct$ )
      case ( $psubset S$ )
      show  $?case$ 
      proof  $cases$ 
        assume  $\exists i \in S. l i < r i$ 
        with  $\langle finite S \rangle$  have  $Min (l \ ' \{i \in S. l i < r i\}) \in l \ ' \{i \in S. l i < r i\}$ 
        by ( $intro Min\_in$ )  $auto$ 
        then obtain  $m$  where  $m: m \in S l m < r m l m = Min (l \ ' \{i \in S. l i < r$ 

```

```

i})
  by fastforce

  have ( $\sum_{i \in S}. F(r\ i) - F(l\ i) = (F(r\ m) - F(l\ m)) + (\sum_{i \in S - \{m\}}. F(r\ i) - F(l\ i))$ )
  F (r i) - F (l i)
    using m_psubset by (intro sum.remove) auto
  also have ( $\sum_{i \in S - \{m\}}. F(r\ i) - F(l\ i) \leq F\ b - F(r\ m)$ )
  proof (intro psubset.IH)
    show  $S - \{m\} \subset S$ 
      using ⟨m ∈ S⟩ by auto
    show  $r\ m \leq b$ 
      using psubset.prem(2)[OF ⟨m ∈ S⟩] ⟨l m < r m⟩ by auto
  next
  fix i assume  $i \in S - \{m\}$ 
  then have  $i: i \in S\ i \neq m$  by auto
  { assume  $i': l\ i < r\ i\ l\ i < r\ m$ 
    with ⟨finite S⟩ i m have  $l\ m \leq l\ i$ 
      by auto
    with  $i'$  have  $\{l\ i <.. r\ i\} \cap \{l\ m <.. r\ m\} \neq \{\}$ 
      by auto
    then have False
      using disjoint_family_onD[OF disj, of i m] i by auto }
  then have  $l\ i \neq r\ i \implies r\ m \leq l\ i$ 
    unfolding not_less[symmetric] using l_r[of i] by auto
  then show  $\{l\ i <.. r\ i\} \subseteq \{r\ m <.. b\}$ 
    using psubset.prem(2)[OF ⟨i ∈ S⟩] by auto
  qed
  also have  $F(r\ m) - F(l\ m) \leq F(r\ m) - F\ a$ 
    using psubset.prem(2)[OF ⟨m ∈ S⟩] ⟨l m < r m⟩
    by (auto simp add: Ioc_subset_iff intro!: mono_F)
  finally show ?case
    by (auto intro: add_mono)
  qed (auto simp add: ⟨a ≤ b⟩ less_le)
qed }
note claim1 = this

```

```

{ fix S u v and l r :: nat ⇒ real
  assume finite S ∧  $i. i \in S \implies l\ i < r\ i$  {u..v} ⊆ (⋃ i ∈ S. {l i <.. < r i})
  then have  $F\ v - F\ u \leq (\sum_{i \in S}. F(r\ i) - F(l\ i))$ 
  proof (induction arbitrary: v u rule: finite_psubset_induct)
    case (psubset S)
    show ?case
    proof cases
      assume  $S = \{\}$  then show ?case
        using psubset by (simp add: mono_F)
    next
      assume  $S \neq \{\}$ 

```



```

then obtain j where j ∈ S
  by auto

let ?R = r j < u ∨ l j > v ∨ (∃ i ∈ S - {j}. l i ≤ l j ∧ r j ≤ r i)
show ?case
proof cases
  assume ?R
  with ⟨j ∈ S⟩ psubset.prem1 have {u..v} ⊆ (⋃ i ∈ S - {j}. {l i <..< r i})
  apply (simp add: subset_eq Ball_def Bex_def)
  by (metis order_le_less_trans order_less_le_trans order_less_not_sym)
  with ⟨j ∈ S⟩ have F v - F u ≤ (∑ i ∈ S - {j}. F (r i) - F (l i))
  by (intro psubset) auto
  also have ... ≤ (∑ i ∈ S. F (r i) - F (l i))
  using psubset.prem1
  by (intro sum_mono2 psubset) (auto intro: less_imp_le)
  finally show ?thesis .
next
  assume ¬ ?R
  then have j: u ≤ r j ∧ l j ≤ v ∧ i. i ∈ S - {j} ⇒ r i < r j ∨ l i > l j
  by (auto simp: not_less)
  let ?S1 = {i ∈ S. l i < l j}
  let ?S2 = {i ∈ S. r i > r j}
  have *: ?S1 ∩ ?S2 = {}
  using j by (fastforce simp add: disjoint_iff)
  have (∑ i ∈ S. F (r i) - F (l i)) ≥ (∑ i ∈ ?S1 ∪ ?S2 ∪ {j}. F (r i) - F
(l i))
  using ⟨j ∈ S⟩ ⟨finite S⟩ psubset.prem1 j
  by (intro sum_mono2) (auto intro: less_imp_le)
  also have (∑ i ∈ ?S1 ∪ ?S2 ∪ {j}. F (r i) - F (l i)) =
  (∑ i ∈ ?S1. F (r i) - F (l i)) + (∑ i ∈ ?S2. F (r i) - F (l i)) + (F (r
j) - F (l j))
  using psubset(1) by (simp add: * sum.union_disjoint disjoint_iff_not_equal)
  also (xtrans) have (∑ i ∈ ?S1. F (r i) - F (l i)) ≥ F (l j) - F u
  using ⟨j ∈ S⟩ ⟨finite S⟩ psubset.prem1 j
  apply (intro psubset.IH psubset)
  apply (auto simp: subset_eq Ball_def)
  apply (metis less_le_trans not_le)
  done
  also (xtrans) have (∑ i ∈ ?S2. F (r i) - F (l i)) ≥ F v - F (r j)
  using ⟨j ∈ S⟩ ⟨finite S⟩ psubset.prem1 j
  apply (intro psubset.IH psubset)
  apply (auto simp: subset_eq Ball_def)
  apply (metis le_less_trans not_le)
  done
  finally (xtrans) show ?case
  by (auto simp: add_mono)
qed
qed
qed }

```

note *claim2 = this*

```

have ennreal (F b - F a) ≤ (∑ i. ennreal (F (r i) - F (l i)))
proof (rule ennreal_le_epsilon)
  fix epsilon :: real assume egt0: epsilon > 0
  have ∀ i. ∃ d > 0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
  proof
    fix i
    note right_cont_F [of r i]
    then have ∃ d > 0. F (r i + d) - F (r i) < epsilon / 2^(i + 2)
      by (simp add: continuous_at_right_real_increasing egt0)
    thus ∃ d > 0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
      by force
  qed
then obtain delta where
  delta_i_gt0: ∧ i. delta i > 0 and
  delta_i_prop: ∧ i. F (r i + delta i) < F (r i) + epsilon / 2^(i+2)
  by metis
have ∃ a' > a. F a' - F a < epsilon / 2
  using right_cont_F [of a]
  by (metis continuous_at_right_real_increasing egt0 half_gt_zero less_add_same_cancel1
mono_F)
then obtain a' where a'_lea [arith]: a' > a and
  a'_prop: F a' - F a < epsilon / 2
  by auto
define S' where S' = {i. l i < r i}
obtain S :: nat set where S ⊆ S' and finS: finite S
  and Sprop: {a'..b} ⊆ (∪ i ∈ S. {l i <.. < r i + delta i})
proof (rule compactE_image)
  show compact {a'..b}
    by (rule compact_Icc)
  show ∧ i. i ∈ S' ⇒ open ({l i <.. < r i + delta i}) by auto
  have {a'..b} ⊆ {a <.. b}
    by auto
  also have {a <.. b} = (∪ i ∈ S'. {l i <.. r i})
    unfolding lr_eq_ab[symmetric] by (fastforce simp add: S'_def intro:
less_le_trans)
  also have ... ⊆ (∪ i ∈ S'. {l i <.. < r i + delta i})
  by (intro UN_mono; simp add: add commute add_strict_increasing delta_i_gt0
subset_iff)
  finally show {a'..b} ⊆ (∪ i ∈ S'. {l i <.. < r i + delta i}) .
qed
with S'_def have Sprop2: ∧ i. i ∈ S ⇒ l i < r i by auto
obtain n where Sbound: ∧ i. i ∈ S ⇒ i ≤ n
  using Max_ge finS by blast
have F b - F a = (F b - F a') + (F a' - F a)
  by auto
also have ... ≤ (F b - F a') + epsilon / 2

```

```

    using a_prop by (intro add_left_mono) simp
  also have ... ≤ (∑ i∈S. F (r i) - F (l i)) + epsilon / 2 + epsilon / 2
  proof -
    have F b - F a' ≤ (∑ i∈S. F (r i + delta i) - F (l i))
      using claim2 l_r Sprop by (simp add: deltai_gt0 finS add.commute
add_strict_increasing)
    also have ... ≤ (∑ i ∈ S. F(r i) - F(l i) + epsilon / 2^(i+2))
      by (smt (verit) sum_mono deltai_prop)
    also have ... = (∑ i ∈ S. F(r i) - F(l i)) +
      (epsilon / 4) * (∑ i ∈ S. (1 / 2)^i) (is _ = ?t + _)
      by (subst sum.distrib) (simp add: field_simps sum_distrib_left)
    also have ... ≤ ?t + (epsilon / 4) * (∑ i < Suc n. (1 / 2)^i)
      using egt0 Sbound by (intro add_left_mono mult_left_mono sum_mono2)
  force+
    also have ... ≤ ?t + (epsilon / 2)
      using egt0 by (simp add: geometric_sum add_left_mono mult_left_mono)
    finally have F b - F a' ≤ (∑ i∈S. F (r i) - F (l i)) + epsilon / 2
      by simp
    then show ?thesis
      by linarith
  qed
  also have ... = (∑ i∈S. F (r i) - F (l i)) + epsilon
    by auto
  also have ... ≤ (∑ i≤n. F (r i) - F (l i)) + epsilon
    using finS Sbound Sprop by (auto intro!: add_right_mono sum_mono2)
  finally have ennreal (F b - F a) ≤ (∑ i≤n. ennreal (F (r i) - F (l i))) +
  epsilon
    using egt0 by (simp add: sum_nonneg flip: ennreal_plus)
  then show ennreal (F b - F a) ≤ (∑ i. ennreal (F (r i) - F (l i))) + (epsilon
  :: real)
    by (rule order_trans) (auto intro!: add_mono sum_le_suminf simp del:
sum_ennreal)
  qed
  moreover have (∑ i. ennreal (F (r i) - F (l i))) ≤ ennreal (F b - F a)
    using ⟨a ≤ b⟩ by (auto intro!: suminf_le_const ennreal_le_iff[THEN iffD2]
claim1)
  ultimately show (∑ n. ennreal (F (r n) - F (l n))) = ennreal (F b - F a)
    by (rule antisym[rotated])
  qed (auto simp: Ioc_inj mono_F)

```

lemma *measure_interval_measure_Ioc*:

assumes $a \leq b$ **and** $\bigwedge x y. x \leq y \implies F x \leq F y$ **and** $\bigwedge a. \text{continuous (at_right } a) F$

shows $\text{measure (interval_measure } F) \{a <.. b\} = F b - F a$

unfolding *measure_def*

by (*simp add: assms emeasure_interval_measure_Ioc*)

lemma *emeasure_interval_measure_Ioc_eq*:

$(\bigwedge x y. x \leq y \implies F x \leq F y) \implies (\bigwedge a. \text{continuous (at_right } a) F) \implies$

$emeasure (interval_measure F) \{a <.. b\} = (if a \leq b \text{ then } F b - F a \text{ else } 0)$
using $emeasure_interval_measure_Ioc$ $[of a b F]$ **by** $auto$

lemma $sets_interval_measure$ $[simp, measurable_cong]$:

$sets (interval_measure F) = sets \text{ borel}$

apply $(simp \text{ add: } sets_extend_measure \text{ interval_measure_def } \text{ borel_sigma_sets_Ioc}$
 $image_def \text{ split: } \text{ prod.split})$

by $(metis \text{ greaterThanAtMost_empty } \text{ nle_le})$

lemma $space_interval_measure$ $[simp]$: $space (interval_measure F) = UNIV$

by $(simp \text{ add: } interval_measure_def \text{ space_extend_measure})$

lemma $emeasure_interval_measure_Icc$:

assumes $a \leq b$

assumes $mono_F: \bigwedge x y. x \leq y \implies F x \leq F y$

assumes $cont_F: \text{continuous_on } UNIV F$

shows $emeasure (interval_measure F) \{a .. b\} = F b - F a$

proof $(rule \text{ tendsto_unique})$

$\{ \text{fix } a b :: \text{real } \text{assume } a \leq b \text{ then have } emeasure (interval_measure F) \{a$
 $<.. b\} = F b - F a$

using $cont_F$

by $(subst \text{ emeasure_interval_measure_Ioc})$

$(auto \text{ intro: } \text{mono_F } \text{continuous_within_subset } \text{simp: } \text{continuous_on_eq_continuous_within})$

$\}$

note $* = \text{this}$

let $?F = interval_measure F$

show $((\lambda a. F b - F a) \longrightarrow emeasure ?F \{a..b\}) (at_left a)$

proof $(rule \text{ tendsto_at_left_sequentially})$

show $a - 1 < a$ **by** $simp$

fix X **assume** $X: \bigwedge n. X n < a \text{ incseq } X X \longrightarrow a$

then have $emeasure (interval_measure F) \{X n <.. b\} \neq \infty$ **for** n

by $(smt (verit) * \langle a \leq b \rangle \text{ ennreal_neq_top_infinity_ennreal_def})$

with X **have** $(\lambda n. emeasure ?F \{X n <.. b\}) \longrightarrow emeasure ?F (\bigcap n. \{X n$
 $<.. b\})$

by $(intro \text{ Lim_emeasure_decseq; force } \text{simp: } \text{decseq_def } \text{incseq_def } \text{emea-}$
 $\text{sure_interval_measure_Ioc } *)$

also have $(\bigcap n. \{X n <.. b\}) = \{a..b\}$

apply $auto$

apply $(rule \text{ LIMSEQ_le_const2}[OF \langle X \longrightarrow a \rangle])$

using less_eq_real_def **apply** presburger

using $X(1)$ $\text{order_less_le_trans}$ **by** blast

also have $(\lambda n. emeasure ?F \{X n <.. b\}) = (\lambda n. F b - F (X n))$

using $\langle \bigwedge n. X n < a \rangle \langle a \leq b \rangle$ **by** $(subst *) (auto \text{ intro: } \text{less_imp_le}$

$\text{less_le_trans})$

finally show $(\lambda n. F b - F (X n)) \longrightarrow emeasure ?F \{a..b\} .$

qed

show $((\lambda a. \text{ennreal } (F b - F a)) \longrightarrow F b - F a) (at_left a)$

by $(rule \text{ continuous_on_tendsto_compose}[\text{where } g = \lambda x. x \text{ and } s = UNIV])$

(*auto simp: continuous_on_enreal continuous_on_diff cont_F*)
qed (*rule trivial_limit_at_left_real*)

lemma *sigma_finite_interval_measure:*

assumes *mono_F*: $\bigwedge x y. x \leq y \implies F x \leq F y$
assumes *right_cont_F*: $\bigwedge a. \text{continuous}(\text{at_right } a) F$
shows *sigma_finite_measure* (*interval_measure F*)
apply *unfold_locales*
apply (*intro exI[of_* $(\lambda(a, b). \{a <.. b\})$ ‘ $(\mathbf{Q} \times \mathbf{Q})$ ’)
apply (*auto intro!*: *Rats_no_top_le Rats_no_bot_less countable_rat simp: emeasure_interval_measure_Ioc_eq[OF assms]*)
done

7.12.2 Lebesgue-Borel measure

definition *lborel* :: (*'a* :: *euclidean_space*) *measure* **where**

lborel = *distr* ($\Pi_M b \in \text{Basis}. \text{interval_measure}(\lambda x. x)$) *borel* ($\lambda f. \sum b \in \text{Basis}. f b *_R b$)

abbreviation *lebesgue* :: (*'a* :: *euclidean_space*) *measure*

where *lebesgue* \equiv *completion lborel*

abbreviation *lebesgue_on* :: (*'a* *set*) \Rightarrow (*'a* :: *euclidean_space*) *measure*

where *lebesgue_on* $\Omega \equiv$ *restrict_space* (*completion lborel*) Ω

lemma *lebesgue_on_mono:*

assumes *major*: *AE* *x* *in lebesgue_on S. P x* **and** *minor*: $\bigwedge x. \llbracket P x; x \in S \rrbracket \implies Q x$

shows *AE x in lebesgue_on S. Q x*

proof –

have *AE a in lebesgue_on S. P a* \longrightarrow *Q a*

using *minor space_restrict_space* **by** *fastforce*

then show *?thesis*

using *major* **by** *auto*

qed

lemma *integral_eq_zero_null_sets:*

assumes *S* \in *null_sets lebesgue*

shows $\text{integral}^L(\text{lebesgue_on } S) f = 0$

proof (*rule integral_eq_zero_AE*)

show *AE x in lebesgue_on S. f x = 0*

by (*metis* (*no_types, lifting*) *assms AE_not_in lebesgue_on_mono null_setsD2 null_sets_restrict_space order_refl*)

qed

lemma

shows *sets_lborel*[*simp, measurable_cong*]: *sets lborel* = *sets borel*

and *space_lborel*[*simp*]: *space lborel* = *space borel*

and *measurable_lborel1*[*simp*]: *measurable M lborel* = *measurable M borel*

and *measurable_lborel2*[simp]: *measurable lborel M = measurable borel M*
by (*simp_all add: lborel_def*)

lemma *space_lebesgue_on* [simp]: *space (lebesgue_on S) = S*
by (*simp add: space_restrict_space*)

lemma *sets_lebesgue_on_refl* [iff]: *S ∈ sets (lebesgue_on S)*
by (*metis inf_top.right_neutral sets.top space_borel space_completion space_lborel space_restrict_space*)

lemma *Compl_in_sets_lebesgue*: *−A ∈ sets lebesgue ↔ A ∈ sets lebesgue*
by (*metis Compl_eq_Diff_UNIV double_compl space_borel space_completion space_lborel Sigma_Algebra.sets.compl_sets*)

lemma *measurable_lebesgue_cong*:
assumes $\bigwedge x. x \in S \implies f x = g x$
shows *f ∈ measurable (lebesgue_on S) M ↔ g ∈ measurable (lebesgue_on S) M*
by (*metis (mono_tags, lifting) IntD1 assms measurable_cong_simp space_restrict_space*)

lemma *lebesgue_on_UNIV_eq*: *lebesgue_on UNIV = lebesgue*
by (*simp add: emeasure_restrict_space measure_eqI*)

lemma *integral_restrict_Int*:
fixes *f :: 'a::euclidean_space ⇒ 'b::euclidean_space*
assumes *S ∈ sets lebesgue T ∈ sets lebesgue*
shows *integral^L (lebesgue_on T) (λx. if x ∈ S then f x else 0) = integral^L (lebesgue_on (S ∩ T)) f*
proof −
have $(\lambda x. \text{indicat_real } T x *_R (\text{if } x \in S \text{ then } f x \text{ else } 0)) = (\lambda x. \text{indicat_real } (S \cap T) x *_R f x)$
by (*force simp: indicator_def*)
then show *?thesis*
by (*simp add: assms sets.Int Bochner_Integration.integral_restrict_space*)
qed

lemma *integral_restrict*:
fixes *f :: 'a::euclidean_space ⇒ 'b::euclidean_space*
assumes *S ⊆ T S ∈ sets lebesgue T ∈ sets lebesgue*
shows *integral^L (lebesgue_on T) (λx. if x ∈ S then f x else 0) = integral^L (lebesgue_on S) f*
using *integral_restrict_Int [of S T f] assms*
by (*simp add: Int_absorb2*)

lemma *integral_restrict_UNIV*:
fixes *f :: 'a::euclidean_space ⇒ 'b::euclidean_space*
assumes *S ∈ sets lebesgue*
shows *integral^L lebesgue (λx. if x ∈ S then f x else 0) = integral^L (lebesgue_on S) f*

using *integral_restrict_Int* [of *S UNIV f*] *assms*
 by (*simp add: lebesgue_on_UNIV_eq*)

lemma *integrable_lebesgue_on_empty* [*iff*]:
 fixes *f* :: 'a::euclidean_space \Rightarrow 'b::{second_countable_topology,banach}
 shows *integrable* (*lebesgue_on* {}) *f*
 by (*simp add: integrable_restrict_space*)

lemma *integral_lebesgue_on_empty* [*simp*]:
 fixes *f* :: 'a::euclidean_space \Rightarrow 'b::{second_countable_topology,banach}
 shows *integral*^L (*lebesgue_on* {}) *f* = 0
 by (*simp add: Bochner_Integration.integral_empty*)

lemma *has_bochner_integral_restrict_space*:
 fixes *f* :: 'a \Rightarrow 'b::{banach, second_countable_topology}
 assumes $\Omega: \Omega \cap \text{space } M \in \text{sets } M$
 shows *has_bochner_integral* (*restrict_space* *M* Ω) *f* *i*
 \longleftrightarrow *has_bochner_integral* *M* ($\lambda x. \text{indicator } \Omega x *_{\mathbb{R}} f x$) *i*
 by (*simp add: integrable_restrict_space* [*OF assms*] *integral_restrict_space* [*OF assms*] *has_bochner_integral_iff*)

lemma *integrable_restrict_UNIV*:
 fixes *f* :: 'a::euclidean_space \Rightarrow 'b::{banach, second_countable_topology}
 assumes *S*: *S* \in *sets lebesgue*
 shows *integrable lebesgue* ($\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$) \longleftrightarrow *integrable* (*lebesgue_on* *S*) *f*
 using *has_bochner_integral_restrict_space* [of *S lebesgue f*] *assms*
 by (*simp add: integrable_simps indicator_scaleR_eq_if*)

lemma *integral_mono_lebesgue_on_AE*:
 fixes *f*::_ \Rightarrow *real*
 assumes *f*: *integrable* (*lebesgue_on* *T*) *f*
 and *gf*: *AE* *x* *in* (*lebesgue_on* *S*). *g* *x* \leq *f* *x*
 and *f0*: *AE* *x* *in* (*lebesgue_on* *T*). $0 \leq f x$
 and *S* \subseteq *T* and *S*: *S* \in *sets lebesgue* and *T*: *T* \in *sets lebesgue*
 shows $(\int x. g x \partial(\text{lebesgue_on } S)) \leq (\int x. f x \partial(\text{lebesgue_on } T))$
proof –
 have $(\int x. g x \partial(\text{lebesgue_on } S)) = (\int x. (\text{if } x \in S \text{ then } g x \text{ else } 0) \partial \text{lebesgue})$
 by (*simp add: Lebesgue_Measure.integral_restrict_UNIV S*)
 also have $\dots \leq (\int x. (\text{if } x \in T \text{ then } f x \text{ else } 0) \partial \text{lebesgue})$
proof (*rule Bochner_Integration.integral_mono_AE'*)
 show *integrable lebesgue* ($\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$)
 by (*simp add: integrable_restrict_UNIV T f*)
 show *AE* *x* *in* *lebesgue*. $(\text{if } x \in S \text{ then } g x \text{ else } 0) \leq (\text{if } x \in T \text{ then } f x \text{ else } 0)$
 using *assms* by (*auto simp: AE_restrict_space_iff*)
 show *AE* *x* *in* *lebesgue*. $0 \leq (\text{if } x \in T \text{ then } f x \text{ else } 0)$
 using *f0* by (*simp add: AE_restrict_space_iff T*)
qed
 also have $\dots = (\int x. f x \partial(\text{lebesgue_on } T))$
 using *Lebesgue_Measure.integral_restrict_UNIV T* by *blast*

finally show ?thesis .
qed

7.12.3 Borel measurability

lemma borel_measurable_if_I:

fixes f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space

assumes f: f \in borel_measurable (lebesgue_on S) and S: S \in sets lebesgue

shows (λx . if x \in S then f x else 0) \in borel_measurable lebesgue

proof -

have eq: {x. x \notin S} \cup {x. f x \in Y} = {x. x \notin S} \cup {x. f x \in Y} \cap S for Y

by blast

show ?thesis

using f S

apply (simp add: vimage_def in_borel_measurable_borel Ball_def)

apply (elim all_forward imp_forward asm_rl)

apply (simp only: Collect_conj_eq Collect_disj_eq imp_conv_disj_eq)

apply (auto simp: Compl_eq [symmetric] Compl_in_sets_lebesgue sets_restrict_space_iff)

done

qed

lemma borel_measurable_if_D:

fixes f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space

assumes (λx . if x \in S then f x else 0) \in borel_measurable lebesgue

shows f \in borel_measurable (lebesgue_on S)

using assms by (smt (verit) measurable_lebesgue_cong measurable_restrict_space1)

lemma borel_measurable_if:

fixes f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space

assumes S \in sets lebesgue

shows (λx . if x \in S then f x else 0) \in borel_measurable lebesgue \longleftrightarrow f \in borel_measurable (lebesgue_on S)

using assms borel_measurable_if_D borel_measurable_if_I by blast

lemma borel_measurable_if_lebesgue_on:

fixes f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space

assumes S \in sets lebesgue T \in sets lebesgue S \subseteq T

shows (λx . if x \in S then f x else 0) \in borel_measurable (lebesgue_on T) \longleftrightarrow f \in borel_measurable (lebesgue_on S)

(is ?lhs = ?rhs)

proof

assume ?lhs then show ?rhs

using measurable_restrict_mono [OF _ $\langle S \subseteq T \rangle$]

by (subst measurable_lebesgue_cong [where g = (λx . if x \in S then f x else 0)]) auto

next

assume ?rhs then show ?lhs

by (simp add: $\langle S \in$ sets lebesgue \rangle borel_measurable_if_I measurable_restrict_space1)

qed

lemma *borel_measurable_vimage_halfspace_component_lt*:
 $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i < a\} \in \text{sets } (\text{lebesgue_on } S))$
by (*force simp add: space_restrict_space trans [OF borel_measurable_iff_halfspace_less]*)

lemma *borel_measurable_vimage_halfspace_component_ge*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
shows $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i \geq a\} \in \text{sets } (\text{lebesgue_on } S))$
by (*force simp add: space_restrict_space trans [OF borel_measurable_iff_halfspace_ge]*)

lemma *borel_measurable_vimage_halfspace_component_gt*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
shows $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i > a\} \in \text{sets } (\text{lebesgue_on } S))$
by (*force simp add: space_restrict_space trans [OF borel_measurable_iff_halfspace_greater]*)

lemma *borel_measurable_vimage_halfspace_component_le*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
shows $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall a \ i. \ i \in \text{Basis} \longrightarrow \{x \in S. f \ x \cdot i \leq a\} \in \text{sets } (\text{lebesgue_on } S))$
by (*force simp add: space_restrict_space trans [OF borel_measurable_iff_halfspace_le]*)

lemma
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
shows *borel_measurable_vimage_open_interval*:
 $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall a \ b. \ \{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue_on } S))$ (**is** *?thesis1*)
and *borel_measurable_vimage_open*:
 $f \in \text{borel_measurable } (\text{lebesgue_on } S) \iff$
 $(\forall T. \ \text{open } T \longrightarrow \{x \in S. f \ x \in T\} \in \text{sets } (\text{lebesgue_on } S))$ (**is** *?thesis2*)

proof –
have $\{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue_on } S)$ **if** $f \in \text{borel_measurable } (\text{lebesgue_on } S)$ **for** $a \ b$

proof –
have $S = S \cap \text{space lebesgue}$
by *simp*
then have $S \cap (f - \text{' box } a \ b) \in \text{sets } (\text{lebesgue_on } S)$
by (*metis (no_types) box_borel_in_borel_measurable_borel_inf_sup_aci(1) space_restrict_space that*)
then show *?thesis*
by (*simp add: Collect_conj_eq vimage_def*)

qed

moreover

have $\{x \in S. f \ x \in T\} \in \text{sets } (\text{lebesgue_on } S)$
if $T: \bigwedge a \ b. \ \{x \in S. f \ x \in \text{box } a \ b\} \in \text{sets } (\text{lebesgue_on } S)$ **open** T **for** T

proof –

obtain \mathcal{D} **where** *countable* \mathcal{D} **and** $\mathcal{D}: \bigwedge X. \ X \in \mathcal{D} \implies \exists a \ b. \ X = \text{box } a \ b \cup \mathcal{D}$

2478

```

= T
  using open_countable_Union_open_box that ⟨open T⟩ by metis
  then have eq: {x ∈ S. f x ∈ T} = (⋃ U ∈ D. {x ∈ S. f x ∈ U})
    by blast
  have {x ∈ S. f x ∈ U} ∈ sets (lebesgue_on S) if U ∈ D for U
    using that T D by blast
  then show ?thesis
    by (auto simp: eq intro: Sigma_Algebra.sets.countable_UN' [OF ⟨countable
D⟩])
  qed
  moreover
  have eq: {x ∈ S. f x · i < a} = {x ∈ S. f x ∈ {y. y · i < a}} for i a
    by auto
  have f ∈ borel_measurable (lebesgue_on S)
    if ∧ T. open T ⇒ {x ∈ S. f x ∈ T} ∈ sets (lebesgue_on S)
    by (metis (no_types) eq borel_measurable_vimage_halfspace_component_lt
open_halfspace_component_lt that)
  ultimately show ?thesis1 ?thesis2
    by blast+
  qed

```

lemma borel_measurable_vimage_closed:

```

  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  shows f ∈ borel_measurable (lebesgue_on S) ↔
    (∀ T. closed T ⇒ {x ∈ S. f x ∈ T} ∈ sets (lebesgue_on S))

```

proof –

```

  have eq: {x ∈ S. f x ∈ T} = S - (S ∩ f -' (- T)) for T
    by auto
  show ?thesis

```

unfolding borel_measurable_vimage_open eq

```

  by (metis Diff_Diff_Int closed_Compl diff_eq open_Compl sets.Diff sets_lebesgue_on_refl
vimage_Compl)

```

qed

lemma borel_measurable_vimage_closed_interval:

```

  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  shows f ∈ borel_measurable (lebesgue_on S) ↔
    (∀ a b. {x ∈ S. f x ∈ cbox a b} ∈ sets (lebesgue_on S))
    (is ?lhs = ?rhs)

```

proof

```

  assume ?lhs then show ?rhs

```

```

  using borel_measurable_vimage_closed by blast

```

next

```

  assume RHS: ?rhs

```

```

  have {x ∈ S. f x ∈ T} ∈ sets (lebesgue_on S) if open T for T

```

proof –

```

  obtain D where countable D and D: D ⊆ Pow T ∧ X. X ∈ D ⇒ ∃ a b. X
= cbox a b ∪ D = T

```

```

  using open_countable_Union_open_cbox that ⟨open T⟩ by metis

```

```

then have eq:  $\{x \in S. f x \in T\} = (\bigcup U \in \mathcal{D}. \{x \in S. f x \in U\})$ 
by blast
have  $\{x \in S. f x \in U\} \in \text{sets } (\text{lebesgue\_on } S)$  if  $U \in \mathcal{D}$  for  $U$ 
using that  $\mathcal{D}$  by (metis RHS)
then show ?thesis
by (auto simp: eq intro: Sigma_Algebra.sets.countable_UN' [OF ‹countable
 $\mathcal{D}$ ›])
qed
then show ?lhs
by (simp add: borel_measurable_vimage_open)
qed

```

lemma borel_measurable_vimage_borel:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S) \iff$ 
   $(\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f x \in T\} \in \text{sets } (\text{lebesgue\_on } S))$ 
  (is ?lhs = ?rhs)

```

proof

```

assume  $f: ?lhs$ 
then show ?rhs
using measurable_sets [OF  $f$ ]
by (simp add: Collect_conj_eq inf_sup_aci(1) space_restrict_space vimage_def)
qed (simp add: borel_measurable_vimage_open_interval)

```

lemma lebesgue_measurable_vimage_borel:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
assumes  $f \in \text{borel\_measurable lebesgue } T \in \text{sets borel}$ 
shows  $\{x. f x \in T\} \in \text{sets lebesgue}$ 
using assms borel_measurable_vimage_borel [of  $f$  UNIV] by auto

```

lemma borel_measurable_lebesgue_preimage_borel:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
shows  $f \in \text{borel\_measurable lebesgue} \iff$ 
   $(\forall T. T \in \text{sets borel} \longrightarrow \{x. f x \in T\} \in \text{sets lebesgue})$ 
by (smt (verit, best) Collect_cong UNIV_I borel_measurable_vimage_borel lebesgue_on_UNIV_eq)

```

7.12.4 Measurability of continuous functions

lemma continuous_imp_measurable_on_sets_lebesgue:

```

assumes  $f: \text{continuous\_on } S$  and  $S: S \in \text{sets lebesgue}$ 
shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } S)$ 
by (metis borel_measurable_continuous_on_restrict borel_measurable_subalgebra
 $f$ 
  lebesgue_on_UNIV_eq mono_restrict_space sets_completionI_sets sets_lborel
  space_borel
  space_lebesgue_on space_restrict_space subsetI)

```

lemma id_borel_measurable_lebesgue [iff]: $\text{id} \in \text{borel_measurable lebesgue}$

2480

by (simp add: measurable_completion)

lemma *id_borel_measurable_lebesgue_on* [iff]: $id \in \text{borel_measurable } (\text{lebesgue_on } S)$

by (simp add: measurable_completion measurable_restrict_space1)

context

begin

interpretation *sigma_finite_measure_interval_measure* ($\lambda x. x$)

by (rule *sigma_finite_interval_measure*) auto

interpretation *finite_product_sigma_finite* $\lambda_. \text{interval_measure } (\lambda x. x)$ *Basis*

proof qed *simp*

lemma *lborel_eq_real*: $\text{lborel} = \text{interval_measure } (\lambda x. x)$

unfolding *lborel_def Basis_real_def*

using *distr_id*[of *interval_measure* ($\lambda x. x$)]

by (subst *distr_component*[*symmetric*])

(*simp_all* add: *distr_distr comp_def del: distr_id cong: distr_cong*)

lemma *lborel_eq*: $\text{lborel} = \text{distr } (\prod_M b \in \text{Basis. } \text{lborel}) \text{ borel } (\lambda f. \sum b \in \text{Basis. } f b *_R b)$

by (subst *lborel_def*) (*simp* add: *lborel_eq_real*)

lemma *nn_integral_lborel_prod*:

assumes [*measurable*]: $\bigwedge b. b \in \text{Basis} \implies f b \in \text{borel_measurable borel}$

assumes *nn*[*simp*]: $\bigwedge b x. b \in \text{Basis} \implies 0 \leq f b x$

shows $(\int^+ x. (\prod b \in \text{Basis. } f b (x \cdot b)) \partial \text{lborel}) = (\prod b \in \text{Basis. } (\int^+ x. f b x \partial \text{lborel}))$

by (*simp* add: *lborel_def nn_integral_distr product_nn_integral_prod product_nn_integral_singleton*)

lemma *emeasure_lborel_Icc*[*simp*]:

fixes $l u :: \text{real}$

assumes [*simp*]: $l \leq u$

shows $\text{emeasure lborel } \{l .. u\} = u - l$

by (*simp* add: *emeasure_interval_measure_Icc lborel_eq_real*)

lemma *emeasure_lborel_Icc_eq*: $\text{emeasure lborel } \{l .. u\} = \text{ennreal } (\text{if } l \leq u \text{ then } u - l \text{ else } 0)$

by *simp*

lemma *emeasure_lborel_cbox*[*simp*]:

assumes [*simp*]: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$

shows $\text{emeasure lborel } (\text{cbox } l u) = (\prod b \in \text{Basis. } (u - l) \cdot b)$

proof –

have $(\lambda x. \prod b \in \text{Basis. } \text{indicator } \{l \cdot b .. u \cdot b\} (x \cdot b) :: \text{ennreal}) = \text{indicator } (\text{cbox } l u)$

by (*auto simp: fun_eq_iff cbox_def split: split_indicator*)

then have $\text{emeasure lborel } (\text{cbox } l u) = (\int^+ x. (\prod b \in \text{Basis. } \text{indicator } \{l \cdot b .. u \cdot b\}$

$(x \cdot b) \partial \text{lborel}$
 by *simp*
 also have $\dots = (\prod_{b \in \text{Basis.}} (u - l) \cdot b)$
 by (*subst nn_integral_lborel_prod*) (*simp_all add: prod_enreal_inner_diff_left*)
 finally show ?thesis .
 qed

lemma *AE_lborel_singleton*: *AE* x in *lborel*::'*a*::*euclidean_space* *measure*. $x \neq c$
 using *SOME_Basis AE_discrete_difference* [of $\{c\}$ *lborel*] *emeasure_lborel_cbox*
 [of c c]
 by (*auto simp add: power_0_left*)

lemma *emeasure_lborel_Ioo*[*simp*]:
 assumes [*simp*]: $l \leq u$
 shows *emeasure* *lborel* $\{l <..< u\} = \text{enreal } (u - l)$
proof –
 have *emeasure* *lborel* $\{l <..< u\} = \text{emeasure *lborel* $\{l .. u\}$
 using *AE_lborel_singleton*[of u] *AE_lborel_singleton*[of l] **by** (*intro* *emeasure_eq_AE*) *auto*
 then show ?thesis
 by *simp*
 qed$

lemma *emeasure_lborel_Ioc*[*simp*]:
 assumes [*simp*]: $l \leq u$
 shows *emeasure* *lborel* $\{l <.. u\} = \text{enreal } (u - l)$
 by (*simp add: emeasure_interval_measure_Ioc lborel_eq_real*)

lemma *emeasure_lborel_Ico*[*simp*]:
 assumes [*simp*]: $l \leq u$
 shows *emeasure* *lborel* $\{l ..< u\} = \text{enreal } (u - l)$
proof –
 have *emeasure* *lborel* $\{l ..< u\} = \text{emeasure *lborel* $\{l .. u\}$
 using *AE_lborel_singleton*[of u] *AE_lborel_singleton*[of l] **by** (*intro* *emeasure_eq_AE*) *auto*
 then show ?thesis
 by *simp*
 qed$

lemma *emeasure_lborel_box*[*simp*]:
 assumes [*simp*]: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$
 shows *emeasure* *lborel* $(\text{box } l \ u) = (\prod_{b \in \text{Basis.}} (u - l) \cdot b)$
proof –
 have $(\lambda x. \prod_{b \in \text{Basis.}} \text{indicator } \{l \cdot b <..< u \cdot b\} (x \cdot b) :: \text{enreal}) = \text{indicator}$
 $(\text{box } l \ u)$
 by (*auto simp: fun_eq_iff box_def split: split_indicator*)
 then have *emeasure* *lborel* $(\text{box } l \ u) = (\int^+ x. (\prod_{b \in \text{Basis.}} \text{indicator } \{l \cdot b <..<$
 $u \cdot b\} (x \cdot b)) \partial \text{lborel})$
 by *simp*

2482

also have $\dots = (\prod_{b \in \text{Basis}} (u - l) \cdot b)$
by (*subst nn_integral_lborel_prod*) (*simp_all add: prod_enreal_inner_diff_left*)
finally show *?thesis* .

qed

lemma *emeasure_lborel_cbox_eq*:
 $\text{emeasure lborel (cbox l u)} = (\text{if } \forall b \in \text{Basis}. l \cdot b \leq u \cdot b \text{ then } \prod_{b \in \text{Basis}} (u - l) \cdot b \text{ else } 0)$
using *box_eq_empty(2)* [*THEN iffD2, of u l*] **by** (*auto simp: not_le*)

lemma *emeasure_lborel_box_eq*:
 $\text{emeasure lborel (box l u)} = (\text{if } \forall b \in \text{Basis}. l \cdot b \leq u \cdot b \text{ then } \prod_{b \in \text{Basis}} (u - l) \cdot b \text{ else } 0)$
using *box_eq_empty(1)* [*THEN iffD2, of u l*] **by** (*auto simp: not_le dest!: less_imp_le*)
force

lemma *emeasure_lborel_singleton[simp]*: $\text{emeasure lborel } \{x\} = 0$
using *emeasure_lborel_cbox[of x x] nonempty_Basis*
by (*auto simp del: emeasure_lborel_cbox nonempty_Basis*)

lemma *emeasure_lborel_cbox_finite*: $\text{emeasure lborel (cbox a b)} < \infty$
by (*auto simp: emeasure_lborel_cbox_eq*)

lemma *emeasure_lborel_box_finite*: $\text{emeasure lborel (box a b)} < \infty$
by (*auto simp: emeasure_lborel_box_eq*)

lemma *emeasure_lborel_ball_finite*: $\text{emeasure lborel (ball c r)} < \infty$
by (*metis bounded_ball bounded_subset_cbox_symmetric cbox_borel emeasure_lborel_cbox_finite emeasure_mono order_le_less_trans sets_lborel*)

lemma *emeasure_lborel_cball_finite*: $\text{emeasure lborel (cball c r)} < \infty$
by (*metis bounded_cball bounded_subset_cbox_symmetric cbox_borel emeasure_lborel_cbox_finite emeasure_mono order_le_less_trans sets_lborel*)

lemma *fmeasurable_cbox [iff]*: $\text{cbox a b} \in \text{fmeasurable lborel}$
and *fmeasurable_box [iff]*: $\text{box a b} \in \text{fmeasurable lborel}$
by (*auto simp: fmeasurable_def emeasure_lborel_box_eq emeasure_lborel_cbox_eq*)

lemma
fixes $l u :: \text{real}$
assumes [*simp*]: $l \leq u$
shows *measure_lborel_Icc[simp]*: $\text{measure lborel } \{l .. u\} = u - l$
and *measure_lborel_Ico[simp]*: $\text{measure lborel } \{l ..< u\} = u - l$
and *measure_lborel_Ioc[simp]*: $\text{measure lborel } \{l <.. u\} = u - l$
and *measure_lborel_Ioo[simp]*: $\text{measure lborel } \{l <..< u\} = u - l$
by (*simp_all add: measure_def*)

lemma

assumes [simp]: $\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b$
shows *measure_lborel_box*[simp]: $\text{measure lborel} (\text{box } l \ u) = \prod_{b \in \text{Basis}. (u - l) \cdot b)$
and *measure_lborel_cbox*[simp]: $\text{measure lborel} (\text{cbox } l \ u) = \prod_{b \in \text{Basis}. (u - l) \cdot b)$
by (*simp_all add: measure_def inner_diff_left prod_nonneg*)

lemma *measure_lborel_cbox_eq*:
 $\text{measure lborel} (\text{cbox } l \ u) = (\text{if } \forall b \in \text{Basis}. l \cdot b \leq u \cdot b \text{ then } \prod_{b \in \text{Basis}. (u - l) \cdot b \text{ else } 0)$
using *box_eq_empty(2)*[THEN *iffD2*, of *u l*] **by** (*auto simp: not_le*)

lemma *measure_lborel_box_eq*:
 $\text{measure lborel} (\text{box } l \ u) = (\text{if } \forall b \in \text{Basis}. l \cdot b \leq u \cdot b \text{ then } \prod_{b \in \text{Basis}. (u - l) \cdot b \text{ else } 0)$
using *box_eq_empty(1)*[THEN *iffD2*, of *u l*] **by** (*auto simp: not_le dest!: less_imp_le*)
force

lemma *measure_lborel_singleton*[simp]: $\text{measure lborel} \{x\} = 0$
by (*simp add: measure_def*)

lemma *sigma_finite_lborel*: *sigma_finite_measure lborel*

proof

show $\exists A::'a \text{ set set. countable } A \wedge A \subseteq \text{sets lborel} \wedge \bigcup A = \text{space lborel} \wedge (\forall a \in A. \text{emeasure lborel } a \neq \infty)$

by (*intro exI[of _ range ($\lambda n::\text{nat}. \text{box} (- \text{real } n *_{\mathbb{R}} \text{One}) (\text{real } n *_{\mathbb{R}} \text{One}))]$*)
(auto simp: emeasure_lborel_cbox_eq UN_box_eq_UNIV))

qed

end

lemma *emeasure_lborel_UNIV* [simp]: $\text{emeasure lborel} (\text{UNIV}::'a::\text{euclidean_space set}) = \infty$

proof -

{ **fix** *n::nat*

let *?Ba = Basis :: 'a set*

have $\text{real } n \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real } n$

by (*simp add: mult_le_cancel_right1*)

also

have $\dots \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real} (\text{Suc } n) \wedge \text{card } ?Ba$

apply (*rule mult_left_mono*)

apply (*metis DIM_positive One_nat_def less_eq_Suc_le less_imp_le of_nat_le_iff of_nat_power_self_le_power zero_less_Suc*)

apply (*simp*)

done

finally have $\text{real } n \leq (2::\text{real}) \wedge \text{card } ?Ba * \text{real} (\text{Suc } n) \wedge \text{card } ?Ba .$

} **note** [*intro!*] = *this*

show *?thesis*

unfolding *UN_box_eq_UNIV*[*symmetric*]

```

apply (subst SUP_emeasure_incseq[symmetric])
apply (auto simp: incseq_def subset_box inner_add_left
  simp del: Sup_eq_top_iff SUP_eq_top_iff
  intro!: ennreal_SUP_eq_top)
done
qed

```

```

lemma emeasure_lborel_countable:
  fixes A :: 'a::euclidean_space set
  assumes countable A
  shows emeasure_lborel A = 0
proof -
  have A  $\subseteq$  ( $\bigcup i.$  {from_nat_into A i}) using from_nat_into_surj assms by
  force
  then have emeasure_lborel A  $\leq$  emeasure_lborel ( $\bigcup i.$  {from_nat_into A i})
  by (intro emeasure_mono) auto
  also have emeasure_lborel ( $\bigcup i.$  {from_nat_into A i}) = 0
  by (rule emeasure_UN_eq_0) auto
  finally show ?thesis
  by simp
qed

```

```

lemma countable_imp_null_set_lborel: countable A  $\implies$  A  $\in$  null_sets_lborel
  by (simp add: null_sets_def emeasure_lborel_countable sets.countable)

```

```

lemma finite_imp_null_set_lborel: finite A  $\implies$  A  $\in$  null_sets_lborel
  by (intro countable_imp_null_set_lborel countable_finite)

```

```

lemma insert_null_sets_iff [simp]: insert a N  $\in$  null_sets_lebesgue  $\iff$  N  $\in$ 
  null_sets_lebesgue
  by (meson completion.complete2 finite.simps finite_imp_null_set_lborel null_sets.insert_in_sets

    null_sets_completionI subset_insertI)

```

```

lemma insert_null_sets_lebesgue_on_iff [simp]:
  assumes a  $\in$  S S  $\in$  sets_lebesgue
  shows insert a N  $\in$  null_sets (lebesgue_on S)  $\iff$  N  $\in$  null_sets (lebesgue_on
  S)
  by (simp add: assms null_sets_restrict_space)

```

```

lemma lborel_neq_count_space[simp]:
  fixes A :: ('a::ordered_euclidean_space) set
  shows lborel  $\neq$  count_space A
  by (metis finite.simps finite_imp_null_set_lborel insert_not_empty null_sets_count_space
  singleton_iff)

```

```

lemma mem_closed_if_AE_lebesgue_open:
  assumes open S closed C
  assumes AE x  $\in$  S in lebesgue. x  $\in$  C

```



```

assumes  $x \in S$ 
shows  $x \in C$ 
proof (rule ccontr)
  assume  $x \notin C$ 
  with  $\text{openE}[of\ S - C]$  assms
  obtain  $e$  where  $e: 0 < e \text{ ball } x\ e \subseteq S - C$ 
    by blast
  then obtain  $a\ b$  where  $\text{box: } x \in \text{box } a\ b \text{ box } a\ b \subseteq S - C$ 
    by (metis rational_boxes order_trans)
  then have  $0 < \text{emeasure lebesgue } (\text{box } a\ b)$ 
    by (auto simp: emeasure_lborel_box_eq mem_box algebra_simps intro!: prod_pos)
  also have  $\dots \leq \text{emeasure lebesgue } (S - C)$ 
    using assms box
    by (auto intro!: emeasure_mono)
  also have  $\dots = 0$ 
    using assms
    by (auto simp: eventually_ae_filter completion.complete2_set_diff_eq null_setsD1)
  finally show False by simp
qed

```

```

lemma mem_closed_if_AE_lebesgue:  $\text{closed } C \implies (\text{AE } x \text{ in lebesgue. } x \in C) \implies x \in C$ 
  using mem_closed_if_AE_lebesgue_open[OF open_UNIV] by simp

```

7.12.5 Affine transformation on the Lebesgue-Borel

```

lemma lborel_eqI:
  fixes  $M :: 'a::\text{euclidean\_space}$  measure
  assumes emeasure_eq:  $\bigwedge l\ u. (\bigwedge b. b \in \text{Basis} \implies l \cdot b \leq u \cdot b) \implies \text{emeasure } M$ 
   $(\text{box } l\ u) = (\prod_{b \in \text{Basis}} (u - l) \cdot b)$ 
  assumes sets_eq:  $\text{sets } M = \text{sets borel}$ 
  shows  $\text{lborel} = M$ 
proof (rule measure_eqI_generator_eq)
  let  $?E = \text{range } (\lambda(a, b). \text{box } a\ b::'a \text{ set})$ 
  show Int_stable  $?E$ 
    by (auto simp: Int_stable_def box_Int_box)

  show  $?E \subseteq \text{Pow UNIV sets lborel} = \text{sigma\_sets UNIV } ?E \text{ sets } M = \text{sigma\_sets UNIV } ?E$ 
    by (simp_all add: borel_eq_box sets_eq)

  let  $?A = \lambda n::\text{nat. box } (- (\text{real } n *_{\mathbb{R}} \text{One})) (\text{real } n *_{\mathbb{R}} \text{One}) :: 'a \text{ set}$ 
  show  $\text{range } ?A \subseteq ?E \ (\bigcup i. ?A\ i) = \text{UNIV}$ 
    unfolding UN_box_eq_UNIV by auto
  show  $\text{emeasure lborel } (?A\ i) \neq \infty$  for  $i$  by auto
  show  $\text{emeasure lborel } X = \text{emeasure } M\ X$  if  $X \in ?E$  for  $X$ 
    using that box_eq_empty(1) by (fastforce simp: emeasure_eq emeasure_lborel_box_eq)
qed

```

lemma *lborel_affine_euclidean*:

fixes $c :: 'a::\text{euclidean_space} \Rightarrow \text{real}$ **and** t

defines $T\ x \equiv t + (\sum_{j \in \text{Basis}} (c\ j * (x \cdot j)) *_{\mathbb{R}} j)$

assumes $c: \bigwedge j. j \in \text{Basis} \implies c\ j \neq 0$

shows $\text{lborel} = \text{density} (\text{distr}\ \text{lborel}\ \text{borel}\ T) (\lambda_. (\prod_{j \in \text{Basis}} |c\ j|))$ (**is** $_ = ?D$)

proof (*rule lborel_eqI*)

let $?B = \text{Basis} :: 'a\ \text{set}$

fix $l\ u$ **assume** $le: \bigwedge b. b \in ?B \implies l \cdot b \leq u \cdot b$

have [*measurable*]: $T \in \text{borel} \rightarrow_M \text{borel}$

by (*simp add: T_def[abs_def]*)

have *eq*: $T - ' \text{box}\ l\ u = \text{box}$

$(\sum_{j \in \text{Basis}} (((\text{if}\ 0 < c\ j\ \text{then}\ l - t\ \text{else}\ u - t) \cdot j) / c\ j) *_{\mathbb{R}} j)$

$(\sum_{j \in \text{Basis}} (((\text{if}\ 0 < c\ j\ \text{then}\ u - t\ \text{else}\ l - t) \cdot j) / c\ j) *_{\mathbb{R}} j)$

using c **by** (*auto simp: box_def T_def field_simps inner_simps divide_less_eq*)

with $le\ c$ **show** $\text{emeasure}\ ?D (\text{box}\ l\ u) = (\prod_{b \in ?B} (u - l) \cdot b)$

by (*auto simp: emeasure_density emeasure_distr nn_integral_multc emeasure_lborel_box_eq inner_simps*

field_split_simps ennreal_mult'[symmetric] prod_nonneg

prod.distrib[symmetric]

intro!: prod.cong)

qed *simp*

lemma *lborel_affine*:

fixes $t :: 'a::\text{euclidean_space}$

shows $c \neq 0 \implies \text{lborel} = \text{density} (\text{distr}\ \text{lborel}\ \text{borel}\ (\lambda x. t + c *_{\mathbb{R}} x)) (\lambda_. |c|^{\wedge \text{DIM}}('a))$

using *lborel_affine_euclidean*[**where** $c = \lambda_. 'a. c$ **and** $t = t$]

unfolding *scaleR_scaleR[symmetric] scaleR_sum_right[symmetric] euclidean_representation prod_constant* **by** *simp*

lemma *lborel_real_affine*:

$c \neq 0 \implies \text{lborel} = \text{density} (\text{distr}\ \text{lborel}\ \text{borel}\ (\lambda x. t + c * x)) (\lambda_. \text{ennreal} (\text{abs}\ c))$

using *lborel_affine*[*of* $c\ t$] **by** *simp*

lemma *AE_lborel_affine*:

fixes $P :: \text{real} \Rightarrow \text{bool}$

shows $c \neq 0 \implies \text{Measurable.pred}\ \text{borel}\ P \implies \text{AE}\ x\ \text{in}\ \text{lborel}. P\ x \implies \text{AE}\ x\ \text{in}\ \text{lborel}. P (t + c * x)$

by (*subst lborel_real_affine*[**where** $t = - t / c$ **and** $c = 1 / c$])

(*simp_all add: AE_density AE_distr_iff field_simps*)

lemma *nn_integral_real_affine*:

fixes $c :: \text{real}$ **assumes** [*measurable*]: $f \in \text{borel_measurable}\ \text{borel}$ **and** $c: c \neq 0$

shows $(\int^{+x}. f\ x\ \partial \text{lborel}) = |c| * (\int^{+x}. f (t + c * x)\ \partial \text{lborel})$

by (*subst lborel_real_affine*[*OF* c , *of* t])

(*simp add: nn_integral_density nn_integral_distr nn_integral_cmult*)

lemma *lborel_integrable_real_affine*:

```

fixes f :: real  $\Rightarrow$  'a :: {banach, second_countable_topology}
assumes f: integrable lborel f
shows c  $\neq$  0  $\implies$  integrable lborel ( $\lambda x. f (t + c * x)$ )
using f f [THEN borel_measurable_integrable] unfolding integrable_iff_bounded
by (subst (asm) nn_integral_real_affine[where c=c and t=t]) (auto simp: en-
nreal_mult_less_top)

```

```

lemma lborel_integrable_real_affine_iff:
fixes f :: real  $\Rightarrow$  'a :: {banach, second_countable_topology}
shows c  $\neq$  0  $\implies$  integrable lborel ( $\lambda x. f (t + c * x)$ )  $\longleftrightarrow$  integrable lborel f
using
  lborel_integrable_real_affine[of f c t]
  lborel_integrable_real_affine[of  $\lambda x. f (t + c * x)$  1/c -t/c]
by (auto simp add: field_simps)

```

```

lemma lborel_integral_real_affine:
fixes f :: real  $\Rightarrow$  'a :: {banach, second_countable_topology} and c :: real
assumes c: c  $\neq$  0 shows ( $\int x. f x \partial$  lborel) = |c| *R ( $\int x. f (t + c * x) \partial$  lborel)
proof cases
  assume f[measurable]: integrable lborel f then show ?thesis
  using c f f [THEN borel_measurable_integrable] f [THEN lborel_integrable_real_affine,
of c t]
  by (subst lborel_real_affine[OF c, of t])
  (simp add: integral_density_integral_distr)
next
  assume  $\neg$  integrable lborel f with c show ?thesis
  by (simp add: lborel_integrable_real_affine_iff not_integrable_integral_eq)
qed

```

```

lemma
fixes c :: 'a::euclidean_space  $\Rightarrow$  real and t
assumes c:  $\bigwedge j. j \in \text{Basis} \implies c j \neq 0$ 
defines T == ( $\lambda x. t + (\sum j \in \text{Basis}. (c j * (x \cdot j)) *R j)$ )
shows lebesgue_affine_euclidean: lebesgue = density (distr lebesgue lebesgue T)
( $\lambda_. (\prod j \in \text{Basis}. |c j|)$ ) (is _ = ?D)
and lebesgue_affine_measurable: T  $\in$  lebesgue  $\rightarrow_M$  lebesgue
proof -
  have T_borel[measurable]: T  $\in$  borel  $\rightarrow_M$  borel
  by (auto simp: T_def[abs_def])
  { fix A :: 'a set assume A: A  $\in$  sets borel
  then have emeasure lborel A = 0  $\longleftrightarrow$  emeasure (density (distr lborel borel T)
( $\lambda_. (\prod j \in \text{Basis}. |c j|)$ )) A = 0
  unfolding T_def using c by (subst lborel_affine_euclidean[symmetric]) auto
  also have ...  $\longleftrightarrow$  emeasure (distr lebesgue lborel T) A = 0
  using A c by (simp add: distr_completion_emeasure_density_nn_integral_mult
prod_nonneg cong: distr_cong)
  finally have emeasure lborel A = 0  $\longleftrightarrow$  emeasure (distr lebesgue lborel T) A
= 0 . }
  then have eq: null_sets lborel = null_sets (distr lebesgue lborel T)

```

```

    by (auto simp: null_sets_def)

show  $T \in \text{lebesgue} \rightarrow_M \text{lebesgue}$ 
  by (simp add: completion.measurable_completion2 eq measurable_completion)

have  $\text{lebesgue} = \text{completion} (\text{density} (\text{distr lborel borel } T) (\lambda_. (\prod_{j \in \text{Basis}} |c_j|)))$ 
  using  $c$  by (subst lborel_affine_euclidean[of  $c$   $t$ ]) (simp_all add:  $T\_def$ [ $\text{abs\_def}$ ])
  also have  $\dots = \text{density} (\text{completion} (\text{distr lebesgue lborel } T)) (\lambda_. (\prod_{j \in \text{Basis}} |c_j|))$ 
  using  $c$  by (auto intro!: always_eventually_prod_pos completion_density_eq
  simp: distr_completion cong: distr_cong)
  also have  $\dots = \text{density} (\text{distr lebesgue lebesgue } T) (\lambda_. (\prod_{j \in \text{Basis}} |c_j|))$ 
  by (subst completion_completion_distr_eq) (auto simp: eq measurable_completion)
  finally show  $\text{lebesgue} = \text{density} (\text{distr lebesgue lebesgue } T) (\lambda_. (\prod_{j \in \text{Basis}} |c_j|))$  .
qed

```

corollary *lebesgue_real_affine*:

```

 $c \neq 0 \implies \text{lebesgue} = \text{density} (\text{distr lebesgue lebesgue} (\lambda x. t + c * x)) (\lambda_. \text{ennreal } (\text{abs } c))$ 
  using lebesgue_affine_euclidean [where  $c = \lambda x :: \text{real}. c$ ] by simp

```

lemma *nn_integral_real_affine_lebesgue*:

```

fixes  $c :: \text{real}$  assumes  $f[\text{measurable}]$ :  $f \in \text{borel\_measurable lebesgue}$  and  $c: c \neq 0$ 

```

```

shows  $(\int^+ x. f x \partial \text{lebesgue}) = \text{ennreal } |c| * (\int^+ x. f(t + c * x) \partial \text{lebesgue})$ 

```

proof –

```

have  $(\int^+ x. f x \partial \text{lebesgue}) = (\int^+ x. f x \partial \text{density} (\text{distr lebesgue lebesgue} (\lambda x. t + c * x)) (\lambda x. \text{ennreal } |c|))$ 

```

```

  using lebesgue_real_affine  $c$  by auto

```

```

  also have  $\dots = \int^+ x. \text{ennreal } |c| * f x \partial \text{distr lebesgue lebesgue} (\lambda x. t + c * x)$ 

```

```

  by (subst nn_integral_density) auto

```

```

  also have  $\dots = \text{ennreal } |c| * \text{integral}^N (\text{distr lebesgue lebesgue} (\lambda x. t + c * x))$ 

```

f

```

  using  $f$  measurable_distr_eq1 nn_integral_cmult by blast

```

```

  also have  $\dots = |c| * (\int^+ x. f(t + c * x) \partial \text{lebesgue})$ 

```

```

  using lebesgue_affine_measurable [where  $c = \lambda x :: \text{real}. c$ ]

```

```

  by (subst nn_integral_distr) (force+)

```

```

  finally show ?thesis .

```

qed

lemma *lebesgue_measurable_scaling*[*measurable*]: $(*_R) x \in \text{lebesgue} \rightarrow_M \text{lebesgue}$

proof *cases*

```

  assume  $x = 0$ 

```

```

  then have  $(*_R) x = (\lambda x. 0 :: 'a)$ 

```

```

    by (auto simp: fun_eq_iff)

```

```

  then show ?thesis by auto

```

next

```

assume  $x \neq 0$  then show ?thesis
  using lebesgue_affine_measurable[of  $\lambda_. x 0$ ]
  unfolding scaleR_scaleR[symmetric] scaleR_sum_right[symmetric] euclidean_representation
  by (auto simp add: ac_simps)
qed

```

lemma

```

fixes  $m :: \text{real}$  and  $\delta :: 'a :: \text{euclidean\_space}$ 
defines  $T\ r\ d\ x \equiv r *_{\mathbb{R}} x + d$ 
shows emeasure_lebesgue_affine: emeasure lebesgue  $(T\ m\ \delta\ 'S) = |m| \wedge \text{DIM}('a)$ 
* emeasure lebesgue  $S$  (is ?e)
  and measure_lebesgue_affine: measure lebesgue  $(T\ m\ \delta\ 'S) = |m| \wedge \text{DIM}('a)$ 
* measure lebesgue  $S$  (is ?m)
proof -
  show ?e
  proof cases
    assume  $m = 0$  then show ?thesis
      by (simp add: image_constant_conv T_def[abs_def])
    next
      let ?T =  $T\ m\ \delta$  and ?T' =  $T\ (1 / m)\ (-((1/m) *_{\mathbb{R}} \delta))$ 
      assume  $m \neq 0$ 
      then have s_comp_s: ?T'  $\circ$  ?T = id ?T  $\circ$  ?T' = id
      by (auto simp: T_def[abs_def] fun_eq_iff scaleR_add_right scaleR_diff_right)
      then have inv ?T' = ?T bij ?T'
      by (auto intro: inv_unique_comp o_bij)
      then have eq:  $T\ m\ \delta\ 'S = T\ (1 / m)\ ((-1/m) *_{\mathbb{R}} \delta)\ -' S \cap \text{space lebesgue}$ 
      using bij_vimage_eq_inv_image[OF <bij ?T'>, of S] by auto

      have trans_eq_T:  $(\lambda x. \delta + (\sum_{j \in \text{Basis.}} (m * (x \cdot j)) *_{\mathbb{R}} j)) = T\ m\ \delta$  for  $m\ \delta$ 
      unfolding T_def[abs_def] scaleR_scaleR[symmetric] scaleR_sum_right[symmetric]
      by (auto simp add: euclidean_representation ac_simps)

      have T[measurable]:  $T\ r\ d \in \text{lebesgue} \rightarrow_M \text{lebesgue}$  for  $r\ d$ 
      using lebesgue_affine_measurable[of  $\lambda_. r\ d$ ]
      by (cases  $r = 0$ ) (auto simp: trans_eq_T T_def[abs_def])

      show ?thesis
      proof cases
        assume  $S \in \text{sets lebesgue}$  with  $\langle m \neq 0 \rangle$  show ?thesis
          unfolding eq
          apply (subst lebesgue_affine_euclidean[of  $\lambda_. m\ \delta$ ])
          apply (simp_all add: emeasure_density trans_eq_T nn_integral_cmult
emeasure_distr
              del: space_completion emeasure_completion)
          apply (simp add: vimage_comp s_comp_s)
          done
      next
        assume  $S \notin \text{sets lebesgue}$ 
        moreover have ?T 'S  $\notin \text{sets lebesgue}$ 

```

```

proof
  assume ?T ' S ∈ sets lebesgue
  then have ?T -' (?T ' S) ∩ space lebesgue ∈ sets lebesgue
    by (rule measurable_sets[OF T])
  also have ?T -' (?T ' S) ∩ space lebesgue = S
    by (simp add: vimage_comp s_comp_s eq)
  finally show False using ⟨S ∉ sets lebesgue⟩ by auto
qed
ultimately show ?thesis
  by (simp add: emeasure_notin_sets)
qed
qed
show ?m
  unfolding measure_def ⟨?e⟩ by (simp add: enn2real_mult prod_nonneg)
qed

lemma lebesgue_real_scale:
  assumes c ≠ 0
  shows lebesgue = density (distr lebesgue lebesgue (λx. c * x)) (λx. ennreal |c|)
  using assms by (subst lebesgue_affine_euclidean[of λ_. c 0]) simp_all

lemma lborel_has_bochner_integral_real_affine_iff:
  fixes x :: 'a :: {banach, second_countable_topology}
  shows c ≠ 0 ⇒
    has_bochner_integral lborel f x ⟷
    has_bochner_integral lborel (λx. f (t + c * x)) (x /R |c|)
  unfolding has_bochner_integral_iff lborel_integrable_real_affine_iff
  by (simp_all add: lborel_integral_real_affine[symmetric] divideR_right cong:
  conj_cong)

lemma lborel_distr_uminus: distr lborel borel uminus = (lborel :: real measure)
  by (subst lborel_real_affine[of -1 0])
  (auto simp: density_1 one_ennreal_def[symmetric])

lemma lborel_distr_mult:
  assumes (c::real) ≠ 0
  shows distr lborel borel ((* c) = density lborel (λ_. inverse |c|)
proof -
  have distr lborel borel ((* c) = distr lborel lborel ((* c) by (simp cong: distr_cong)
  also from assms have ... = density lborel (λ_. inverse |c|)
    by (subst lborel_real_affine[of inverse c 0]) (auto simp: o_def distr_density_distr)
  finally show ?thesis .
qed

lemma lborel_distr_mult':
  assumes (c::real) ≠ 0
  shows lborel = density (distr lborel borel ((* c)) (λ_. |c|)
proof -
  have lborel = density lborel (λ_. 1) by (rule density_1[symmetric])

```

also from *assms* **have** $(\lambda_. 1 :: \text{ennreal}) = (\lambda_. \text{inverse } |c| * |c|)$ **by** (*intro ext*)
simp
also have $\text{density lborel } \dots = \text{density } (\text{density lborel } (\lambda_. \text{inverse } |c|)) (\lambda_. |c|)$
by (*subst density_density_eq*) (*auto simp: ennreal_mult*)
also from *assms* **have** $\text{density lborel } (\lambda_. \text{inverse } |c|) = \text{distr lborel borel } ((*) c)$
by (*rule lborel_distr_mult[symmetric]*)
finally show *?thesis* .
qed

lemma *lborel_distr_plus*:
fixes $c :: 'a::\text{euclidean_space}$
shows $\text{distr lborel borel } ((+) c) = \text{lborel}$
by (*subst lborel_affine[of 1 c]*, *auto simp: density_1*)

interpretation *lborel*: *sigma_finite_measure lborel*
by (*rule sigma_finite_lborel*)

interpretation *lborel_pair*: *pair_sigma_finite lborel lborel ..*

lemma *lborel_prod*:
 $\text{lborel } \otimes_M \text{lborel} = (\text{lborel} :: ('a::\text{euclidean_space} \times 'b::\text{euclidean_space}) \text{measure})$
proof (*rule lborel_eqI[symmetric]*, *clarify*)
fix $la\ ua :: 'a$ **and** $lb\ ub :: 'b$
assume $lu: \bigwedge a\ b. (a, b) \in \text{Basis} \implies (la, lb) \cdot (a, b) \leq (ua, ub) \cdot (a, b)$
have [*simp*]:
 $\bigwedge b. b \in \text{Basis} \implies la \cdot b \leq ua \cdot b$
 $\bigwedge b. b \in \text{Basis} \implies lb \cdot b \leq ub \cdot b$
 $\text{inj_on } (\lambda u. (u, 0)) \text{Basis } \text{inj_on } (\lambda u. (0, u)) \text{Basis}$
 $(\lambda u. (u, 0)) ' \text{Basis} \cap (\lambda u. (0, u)) ' \text{Basis} = \{\}$
 $\text{box } (la, lb) (ua, ub) = \text{box } la\ ua \times \text{box } lb\ ub$
using $lu[\text{of } _ 0]$ $lu[\text{of } 0]$ **by** (*auto intro!: inj_onI simp add: Basis_prod_def ball_Un box_def*)
show $\text{emeasure } (\text{lborel } \otimes_M \text{lborel}) (\text{box } (la, lb) (ua, ub)) =$
 $\text{ennreal } (\text{prod } ((\cdot) ((ua, ub) - (la, lb))) \text{Basis})$
by (*simp add: lborel_emeasure_pair_measure_Times Basis_prod_def prod.union_disjoint prod.reindex ennreal_mult inner_diff_left prod_nonneg*)
qed (*simp add: borel_prod[symmetric]*)

lemma *lborelD_Collect[measurable (raw)]*: $\{x \in \text{space borel}. P\ x\} \in \text{sets borel} \implies$
 $\{x \in \text{space lborel}. P\ x\} \in \text{sets lborel}$
by *simp*

lemma *lborelD[measurable (raw)]*: $A \in \text{sets borel} \implies A \in \text{sets lborel}$
by *simp*

lemma *emeasure_bounded_finite*:
assumes *bounded A* **shows** $\text{emeasure lborel } A < \infty$

proof –

obtain $a \subseteq b$ **where** $A \subseteq \text{cbox } a \ b$
by (*meson bounded_subset_cbox_symmetric* $\langle \text{bounded } A \rangle$)
then have $\text{emeasure lborel } A \leq \text{emeasure lborel } (\text{cbox } a \ b)$
by (*intro emeasure_mono*) *auto*
then show *?thesis*
by (*auto simp: emeasure_lborel_cbox_eq prod_nonneg less_top[symmetric]*
top_unique split: if_split_asm)
qed

lemma *emeasure_compact_finite*: $\text{compact } A \implies \text{emeasure lborel } A < \infty$
using *emeasure_bounded_finite*[of A] **by** (*auto intro: compact_imp_bounded*)

lemma *borel_integrable_compact*:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$
assumes $\text{compact } S \ \text{continuous_on } S \ f$
shows $\text{integrable lborel } (\lambda x. \text{indicator } S \ x \ *_{\mathbb{R}} \ f \ x)$
proof *cases*
assume $S \neq \{\}$
have $\text{continuous_on } S \ (\lambda x. \text{norm } (f \ x))$
using *assms* **by** (*intro continuous_intros*)
from *continuous_attains_sup*[OF $\langle \text{compact } S \rangle \langle S \neq \{\} \rangle$] *this*
obtain M **where** $M: \bigwedge x. x \in S \implies \text{norm } (f \ x) \leq M$
by *auto*
show *?thesis*
proof (*rule integrable_bound*)
show $\text{integrable lborel } (\lambda x. \text{indicator } S \ x \ * \ M)$
using *assms* **by** (*auto intro!: emeasure_compact_finite borel_compact integrable_mult_left*)
show $(\lambda x. \text{indicator } S \ x \ *_{\mathbb{R}} \ f \ x) \in \text{borel_measurable lborel}$
using *assms* **by** (*auto intro!: borel_measurable_continuous_on_indicator borel_compact*)
show $\text{AE } x \ \text{in } \text{lborel}. \text{norm } (\text{indicator } S \ x \ *_{\mathbb{R}} \ f \ x) \leq \text{norm } (\text{indicator } S \ x \ * \ M)$
by (*auto split: split_indicator simp: abs_real_def dest!: M*)
qed
qed *simp*

lemma *borel_integrable_atLeastAtMost*:

fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $f: \bigwedge x. a \leq x \implies x \leq b \implies \text{isCont } f \ x$
shows $\text{integrable lborel } (\lambda x. f \ x \ * \ \text{indicator } \{a .. b\} \ x)$ (*is integrable _ ?f*)
proof –
have $\text{integrable lborel } (\lambda x. \text{indicator } \{a .. b\} \ x \ *_{\mathbb{R}} \ f \ x)$
proof (*rule borel_integrable_compact*)
from f **show** $\text{continuous_on } \{a..b\} \ f$
by (*auto intro: continuous_at_imp_continuous_on*)
qed *simp*
then show *?thesis*
by (*auto simp: mult.commute*)

qed

7.12.6 Lebesgue measurable sets

abbreviation *lmeasurable* :: 'a::euclidean_space set set

where

lmeasurable \equiv *fmeasurable* *lebesgue*

lemma *not_measurable_UNIV* [*simp*]: *UNIV* \notin *lmeasurable*

by (*simp* *add*: *fmeasurable_def*)

lemma *lmeasurable_iff_integrable*:

S \in *lmeasurable* \iff *integrable* *lebesgue* (*indicator* *S* :: 'a::euclidean_space \Rightarrow *real*)

by (*auto* *simp*: *fmeasurable_def* *integrable_iff_bounded* *borel_measurable_indicator_iff* *ennreal_indicator*)

lemma *lmeasurable_cbox* [*iff*]: *cbox* *a* *b* \in *lmeasurable*

and *lmeasurable_box* [*iff*]: *box* *a* *b* \in *lmeasurable*

by (*auto* *simp*: *fmeasurable_def* *emeasure_lborel_box_eq* *emeasure_lborel_cbox_eq*)

lemma

fixes *a*::*real*

shows *lmeasurable_interval* [*iff*]: $\{a..b\} \in$ *lmeasurable* $\{a <..<b\} \in$ *lmeasurable*

by (*metis* *box_real* *lmeasurable_box* *lmeasurable_cbox*)**+**

lemma *fmeasurable_compact*: *compact* *S* \implies *S* \in *fmeasurable* *lborel*

using *emeasure_compact_finite*[*of* *S*] **by** (*intro* *fmeasurableI*) (*auto* *simp*: *borel_compact*)

lemma *lmeasurable_compact*: *compact* *S* \implies *S* \in *lmeasurable*

using *fmeasurable_compact* **by** (*force* *simp*: *fmeasurable_def*)

lemma *measure_frontier*:

bounded *S* \implies *measure* *lebesgue* (*frontier* *S*) = *measure* *lebesgue* (*closure* *S*) - *measure* *lebesgue* (*interior* *S*)

using *closure_subset* *interior_subset*

by (*auto* *simp*: *frontier_def* *fmeasurable_compact* *intro!*: *measurable_measure_Diff*)

lemma *lmeasurable_closure*:

bounded *S* \implies *closure* *S* \in *lmeasurable*

by (*simp* *add*: *lmeasurable_compact*)

lemma *lmeasurable_frontier*:

bounded *S* \implies *frontier* *S* \in *lmeasurable*

by (*simp* *add*: *compact_frontier_bounded* *lmeasurable_compact*)

lemma *lmeasurable_open*: *bounded* *S* \implies *open* *S* \implies *S* \in *lmeasurable*

using *emeasure_bounded_finite*[*of* *S*] **by** (*intro* *fmeasurableI*) (*auto* *simp*: *borel_open*)

2494

lemma *lmeasurable_ball* [*simp*]: $\text{ball } a \ r \in \text{lmeasurable}$
by (*simp add: lmeasurable_open*)

lemma *lmeasurable_cball* [*simp*]: $\text{cball } a \ r \in \text{lmeasurable}$
by (*simp add: lmeasurable_compact*)

lemma *lmeasurable_interior*: $\text{bounded } S \implies \text{interior } S \in \text{lmeasurable}$
by (*simp add: bounded_interior lmeasurable_open*)

lemma *null_sets_cbox_Diff_box*: $\text{cbox } a \ b - \text{box } a \ b \in \text{null_sets lborel}$
by (*simp add: emeasure_Diff emeasure_lborel_box_eq emeasure_lborel_cbox_eq null_setsI subset_box*)

lemma *bounded_set_imp_lmeasurable*:
assumes $\text{bounded } S \ S \in \text{sets lebesgue}$ shows $S \in \text{lmeasurable}$
by (*metis assms bounded_Un emeasure_bounded_finite emeasure_completion fmeasurableI main_part_null_part_Un*)

lemma *finite_measure_lebesgue_on*: $S \in \text{lmeasurable} \implies \text{finite_measure } (\text{lebesgue_on } S)$
by (*auto simp: finite_measureI fmeasurable_def emeasure_restrict_space*)

lemma *integrable_const_ivl* [*iff*]:
fixes $a::'a::\text{ordered_euclidean_space}$
shows $\text{integrable } (\text{lebesgue_on } \{a..b\}) (\lambda x. c)$
by (*metis cbox_interval finite_measure.integrable_const finite_measure_lebesgue_on lmeasurable_cbox*)

7.12.7 Translation preserves Lebesgue measure

lemma *sigma_sets_image*:
assumes $S: S \in \text{sigma_sets } \Omega \ M$ and $M \subseteq \text{Pow } \Omega \ f' \ \Omega = \Omega \ \text{inj_on } f \ \Omega$
and $M: \bigwedge y. y \in M \implies f' \ y \in M$
shows $(f' \ S) \in \text{sigma_sets } \Omega \ M$
using S

proof (*induct S rule: sigma_sets.induct*)

case (*Basic a*) then show ?case

by (*simp add: M*)

next

case (*Empty*) then show ?case

by (*simp add: sigma_sets.Empty*)

next

case (*Compl a*)

with *assms* show ?case

by (*metis inj_on_image_set_diff sigma_sets.Compl sigma_sets_into_sp*)

next

case (*Union a*) then show ?case

by (*metis image_UN sigma_sets.simps*)

qed

lemma *null_sets_translation*:
assumes $N \in \text{null_sets } \text{lborel}$ **shows** $\{x. x - a \in N\} \in \text{null_sets } \text{lborel}$
proof –
have [*simp*]: $(\lambda x. x + a) \text{ ` } N = \{x. x - a \in N\}$
by *force*
show *?thesis*
using *assms emeasure_lebesgue_affine [of 1 a N]* **by** (*auto simp: null_sets_def*)
qed

lemma *lebesgue_sets_translation*:
fixes $f :: 'a \Rightarrow 'a::\text{euclidean_space}$
assumes $S: S \in \text{sets } \text{lebesgue}$
shows $((\lambda x. a + x) \text{ ` } S) \in \text{sets } \text{lebesgue}$
proof –
have *im_eq*: $(+) a \text{ ` } A = \{x. x - a \in A\}$ **for** A
by *force*
have $((\lambda x. a + x) \text{ ` } S) = ((\lambda x. -a + x) \text{ ` } S) \cap (\text{space } \text{lebesgue})$
using *image_iff* **by** *fastforce*
also **have** $\dots \in \text{sets } \text{lebesgue}$
proof (*rule measurable_sets [OF measurableI assms]*)
fix $A :: 'b \text{ set}$
assume $A: A \in \text{sets } \text{lebesgue}$
have *vim_eq*: $(\lambda x. x - a) \text{ ` } A = (+) a \text{ ` } A$ **for** A
by *force*
have $\exists s n N'. (+) a \text{ ` } (S \cup N) = s \cup n \wedge s \in \text{sets } \text{borel} \wedge N' \in \text{null_sets } \text{lborel} \wedge n \subseteq N'$
if $S \in \text{sets } \text{borel}$ **and** $N' \in \text{null_sets } \text{lborel}$ **and** $N \subseteq N'$ **for** $S N N'$
proof (*intro exI conjI*)
show $(+) a \text{ ` } (S \cup N) = (\lambda x. a + x) \text{ ` } S \cup (\lambda x. a + x) \text{ ` } N$
by *auto*
show $(\lambda x. a + x) \text{ ` } N' \in \text{null_sets } \text{lborel}$
using *that* **by** (*auto simp: null_sets_translation im_eq*)
qed (*use that im_eq in auto*)
with A **have** $(\lambda x. x - a) \text{ ` } A \in \text{sets } \text{lebesgue}$
by (*force simp: vim_eq completion_def intro!: sigma_sets_image*)
then **show** $(+) (- a) \text{ ` } A \cap \text{space } \text{lebesgue} \in \text{sets } \text{lebesgue}$
by (*auto simp: vimage_def im_eq*)
qed *auto*
finally **show** *?thesis* .
qed

lemma *measurable_translation*:
 $S \in \text{lmeasurable} \implies ((+) a \text{ ` } S) \in \text{lmeasurable}$
using *emeasure_lebesgue_affine [of 1 a S]*
by (*smt (verit, best) add commute ennreal_1 fmeasurable_def image_cong lambda_one*)
lebesgue_sets_translation mem_Collect_eq power_one scaleR_one)

lemma *measurable_translation_subtract*:

$S \in \text{lmeasurable} \implies ((\lambda x. x - a) \text{ ` } S) \in \text{lmeasurable}$
using *measurable_translation* [of $S - a$] **by** (*simp cong: image_cong_simp*)

lemma *measure_translation*:

$\text{measure lebesgue } ((+) a \text{ ` } S) = \text{measure lebesgue } S$
using *measure_lebesgue_affine* [of $1 a S$] **by** (*simp add: ac_simps cong: image_cong_simp*)

lemma *measure_translation_subtract*:

$\text{measure lebesgue } ((\lambda x. x - a) \text{ ` } S) = \text{measure lebesgue } S$
using *measure_translation* [of $- a$] **by** (*simp cong: image_cong_simp*)

7.12.8 A nice lemma for negligibility proofs

lemma *summable_iff_suminf_neq_top*: $(\bigwedge n. f n \geq 0) \implies \neg \text{summable } f \implies$
 $(\sum i. \text{ennreal } (f i)) = \text{top}$
by (*metis summable_suminf_not_top*)

proposition *starlike_negligible_bounded_gmeasurable*:

fixes $S :: 'a :: \text{euclidean_space set}$
assumes $S \in \text{sets lebesgue}$ **and** *bounded* S
and *eq1*: $\bigwedge c x. \llbracket (c *_{\mathbb{R}} x) \in S; 0 \leq c; x \in S \rrbracket \implies c = 1$
shows $S \in \text{null_sets lebesgue}$

proof –

obtain M **where** $0 < M$ $S \subseteq \text{ball } 0 M$
using $\langle \text{bounded } S \rangle$ **by** (*auto dest: bounded_subset_ballD*)

let $?f = \lambda n. \text{root } DIM('a) (Suc n)$

have $?f n *_{\mathbb{R}} x \in S \implies x \in (*_{\mathbb{R}}) (1 / ?f n) \text{ ` } S$ **for** $x n$

by (*rule image_eqI*[of $__ ?f n *_{\mathbb{R}} x$]) *auto*

then have *vimage_eq_image*: $(*_{\mathbb{R}}) (?f n) \text{ ` } S = (*_{\mathbb{R}}) (1 / ?f n) \text{ ` } S$ **for** n
by *auto*

have *eq*: $(1 / ?f n) \wedge DIM('a) = 1 / Suc n$ **for** n

by (*simp add: field_simps*)

{ **fix** $n x$ **assume** $x: \text{root } DIM('a) (1 + \text{real } n) *_{\mathbb{R}} x \in S$

have $1 * \text{norm } x \leq \text{root } DIM('a) (1 + \text{real } n) * \text{norm } x$

by (*rule mult_mono*) *auto*

also have $\dots < M$

using $x \langle S \subseteq \text{ball } 0 M \rangle$ **by** *auto*

finally have $\text{norm } x < M$ **by** *simp* }

note *less_M = this*

have $(\sum n. \text{ennreal } (1 / Suc n)) = \text{top}$

using *not_summable_harmonic*[**where** $'a = \text{real}$] *summable_Suc_iff*[**where**
 $f = \lambda n. 1 / (\text{real } n)$]

by (intro summable_iff_suminf_neq_top) (auto simp add: inverse_eq_divide)
then have $\text{top} * \text{emeasure lebesgue } S = (\sum n. (1 / ?f n)^{\text{DIM}(a)} * \text{emeasure lebesgue } S)$
unfolding ennreal_suminf_multc_eq **by** simp
also have $\dots = (\sum n. \text{emeasure lebesgue } ((*_R) (?f n) -' S))$
unfolding vimage_eq_image **using** emeasure_lebesgue_affine[of 1 / ?f n 0 S
for n] **by** simp
also have $\dots = \text{emeasure lebesgue } (\bigcup n. (*_R) (?f n) -' S)$
proof (intro suminf_emeasure)
show disjoint_family $(\lambda n. (*_R) (?f n) -' S)$
unfolding disjoint_family_on_def
proof safe
fix m n :: nat **and** x **assume** $m \neq n$?f m *_R x ∈ S ?f n *_R x ∈ S
with eq1[of ?f m / ?f n ?f n *_R x] **show** $x \in \{\}$
by auto
qed
have $(*_R) (?f i) -' S \in \text{sets lebesgue for } i$
using measurable_sets[OF lebesgue_measurable_scaling[of ?f i] S] **by** auto
then show $\text{range } (\lambda i. (*_R) (?f i) -' S) \subseteq \text{sets lebesgue}$
by auto
qed
also have $\dots \leq \text{emeasure lebesgue } (\text{ball } 0 M :: 'a \text{ set})$
using less_M **by** (intro emeasure_mono) auto
also have $\dots < \text{top}$
using lmeasurable_ball **by** (auto simp: fmeasurable_def)
finally have $\text{emeasure lebesgue } S = 0$
by (simp add: ennreal_top_mult_split: if_split_asm)
then show $S \in \text{null_sets lebesgue}$
unfolding null_sets_def **using** $\langle S \in \text{sets lebesgue} \rangle$ **by** auto
qed

corollary starlike_negligible_compact:

compact S $\implies (\bigwedge c x. [(c *_R x) \in S; 0 \leq c; x \in S] \implies c = 1) \implies S \in \text{null_sets lebesgue}$

using starlike_negligible_bounded_gmeasurable[of S] **by** (auto simp: compact_eq_bounded_closed)

proposition outer_regular_lborel_le:

assumes B[measurable]: $B \in \text{sets borel}$ **and** $0 < (e::\text{real})$

obtains U **where** open U $B \subseteq U$ **and** $\text{emeasure lborel } (U - B) \leq e$

proof –

let ?μ = emeasure lborel

let ?B = $\lambda n::\text{nat}. \text{ball } 0 n :: 'a \text{ set}$

let ?e = $\lambda n. e * ((1/2)^{\text{Suc } n})$

have $\forall n. \exists U. \text{open } U \wedge ?B n \cap B \subseteq U \wedge ?\mu (U - B) < ?e n$

proof

fix n :: nat

let ?A = density lborel (indicator (?B n))

have $\text{emeasure}_A: X \in \text{sets borel} \implies \text{emeasure } ?A X = ?\mu (?B n \cap X)$ **for** X

by (auto simp: emeasure_density_borel_measurable_indicator indicator_inter_arith[symmetric])

```

have finite_A: emeasure ?A (space ?A) ≠ ∞
  using emeasure_bounded_finite[of ?B n] by (auto simp: emeasure_A)
interpret A: finite_measure ?A
  by rule fact
have emeasure ?A B + ?e n > (INF U∈{U. B ⊆ U ∧ open U}. emeasure ?A
U)
  using ‹0 < e› by (auto simp: outer_regular[OF finite_A B, symmetric])
  then obtain U where U: B ⊆ U open U and muU: ?μ (?B n ∩ B) + ?e n
> ?μ (?B n ∩ U)
  unfolding INF_less_iff by (auto simp: emeasure_A)
moreover
{ have ?μ ((?B n ∩ U) - B) = ?μ ((?B n ∩ U) - (?B n ∩ B))
  using U by (intro arg_cong[where f=?μ]) auto
  also have ... = ?μ (?B n ∩ U) - ?μ (?B n ∩ B)
  using U A.emeasure_finite[of B]
  by (intro emeasure_Diff) (auto simp del: A.emeasure_finite simp: emea-
sure_A)
  also have ... < ?e n
  using U muU A.emeasure_finite[of B]
  by (subst minus_less_iff_ennreal)
  (auto simp del: A.emeasure_finite simp: emeasure_A less_top ac_simps
intro!: emeasure_mono)
  finally have ?μ ((?B n ∩ U) - B) < ?e n . }
ultimately show ∃ U. open U ∧ ?B n ∩ B ⊆ U ∧ ?μ (U - B) < ?e n
  by (intro exI[of _ ?B n ∩ U]) auto
qed
then obtain U
  where U: ∧n. open (U n) ∧n. ?B n ∩ B ⊆ U n ∧n. ?μ (U n - B) < ?e n
  by metis
show ?thesis
proof
{ fix x assume x ∈ B
  moreover
  obtain n where norm x < real n
    using reals_Archimedean2 by blast
  ultimately have x ∈ (⋃ n. U n)
    using U(2)[of n] by auto }
note * = this
then show open (⋃ n. U n) B ⊆ (⋃ n. U n)
  using U by auto
have ?μ (⋃ n. U n - B) ≤ (∑ n. ?μ (U n - B))
  using U(1) by (intro emeasure_subadditive_countably) auto
also have ... ≤ (∑ n. ennreal (?e n))
  using U(3) by (intro suminf_le) (auto intro: less_imp_le)
also have ... = ennreal (e * 1)
  using ‹0 < e› by (intro suminf_ennreal_eq sums_mult power_half_series)
auto
  finally show emeasure lborel ((⋃ n. U n) - B) ≤ ennreal e

```

by simp
qed
qed

lemma outer_regular_lborel:

assumes $B: B \in \text{sets borel}$ and $0 < (e::\text{real})$

obtains U where $\text{open } U \ B \subseteq U \ \text{emeasure lborel } (U - B) < e$

proof -

obtain U where $U: \text{open } U \ B \subseteq U$ and $\text{emeasure lborel } (U - B) \leq e/2$

using outer_regular_lborel_le [OF B, of e/2] $\langle e > 0 \rangle$

by force

moreover have $\text{ennreal } (e/2) < \text{ennreal } e$

using $\langle e > 0 \rangle$ by (simp add: ennreal_lessI)

ultimately have $\text{emeasure lborel } (U - B) < e$

by auto

with U show ?thesis

using that by auto

qed

lemma completion_upper:

assumes $A: A \in \text{sets } (\text{completion } M)$

obtains A' where $A \subseteq A' \ A' \in \text{sets } M \ A' - A \in \text{null_sets } (\text{completion } M)$
 $\text{emeasure } (\text{completion } M) \ A = \text{emeasure } M \ A'$

proof -

from $AE_notin_null_part$ [OF A] obtain N where $N: N \in \text{null_sets } M \ \text{null_part } M \ A \subseteq N$

by (meson assms null_part)

let $?A' = \text{main_part } M \ A \cup N$

show ?thesis

proof

show $A \subseteq ?A'$

using $\langle \text{null_part } M \ A \subseteq N \rangle$ assms main_part_null_part_Un by blast

have $\text{main_part } M \ A \subseteq A$

using assms main_part_null_part_Un by auto

then have $?A' - A \subseteq N$

by blast

with N show $?A' - A \in \text{null_sets } (\text{completion } M)$

by (blast intro: null_sets_completionI completion.complete_measure_axioms complete_measure.complete2)

show $\text{emeasure } (\text{completion } M) \ A = \text{emeasure } M \ (\text{main_part } M \ A \cup N)$

using $A \langle N \in \text{null_sets } M \rangle$ by (simp add: emeasure_Un_null_set)

qed (use A N in auto)

qed

lemma sets_lebesgue_outer_open:

fixes $e::\text{real}$

assumes $S: S \in \text{sets lebesgue}$ and $e > 0$

obtains T where $\text{open } T \ S \subseteq T \ (T - S) \in \text{lmeasurable } \text{emeasure lebesgue } (T - S) < \text{ennreal } e$

2500

proof –

```
obtain  $S'$  where  $S' \subseteq S$   $S' \in \text{sets borel}$ 
  and  $\text{null}: S' - S \in \text{null\_sets lebesgue}$ 
  and  $\text{em}: \text{emeasure lebesgue } S = \text{emeasure lborel } S'$ 
  using  $\text{completion\_upper}[of S \text{ lborel}] S$  by  $\text{auto}$ 
then have  $f_{S'}: S' \in \text{sets borel}$ 
  using  $S$  by  $(\text{auto simp: fmeasurable\_def})$ 
with  $\text{outer\_regular\_lborel}[OF \_ \langle 0 < e \rangle]$ 
obtain  $U$  where  $U: \text{open } U$   $S' \subseteq U$   $\text{emeasure lborel } (U - S') < e$ 
  by  $\text{blast}$ 
show  $\text{thesis}$ 
proof
  show  $\text{open } U$   $S \subseteq U$ 
    using  $f_{S'} U S'$  by  $\text{auto}$ 
have  $(U - S) = (U - S') \cup (S' - S)$ 
  using  $S' U$  by  $\text{auto}$ 
then have  $\text{eq}: \text{emeasure lebesgue } (U - S) = \text{emeasure lborel } (U - S')$ 
  using  $\text{null}$  by  $(\text{simp add: } U(1) \text{ emeasure\_Un\_null\_set } f_{S'} \text{ sets.Diff})$ 
have  $(U - S) \in \text{sets lebesgue}$ 
  by  $(\text{simp add: } S U(1) \text{ sets.Diff})$ 
then show  $(U - S) \in \text{lmeasurable}$ 
  unfolding  $\text{fmeasurable\_def}$  using  $U(3) \text{ eq less\_le\_trans}$  by  $\text{fastforce}$ 
with  $\text{eq } U$  show  $\text{emeasure lebesgue } (U - S) < e$ 
  by  $(\text{simp add: eq})$ 
qed
qed
```

lemma $\text{sets_lebesgue_inner_closed}$:

```
fixes  $e::\text{real}$ 
assumes  $S \in \text{sets lebesgue}$   $e > 0$ 
obtains  $T$  where  $\text{closed } T$   $T \subseteq S$   $S - T \in \text{lmeasurable}$   $\text{emeasure lebesgue } (S - T) < \text{ennreal } e$ 
proof –
  have  $-S \in \text{sets lebesgue}$ 
    using  $\text{assms}$  by  $(\text{simp add: Compl\_in\_sets\_lebesgue})$ 
then obtain  $T$  where  $\text{open } T$   $-S \subseteq T$ 
  and  $T: (T - -S) \in \text{lmeasurable}$   $\text{emeasure lebesgue } (T - -S) < e$ 
  using  $\text{sets\_lebesgue\_outer\_open}$   $\text{assms}$  by  $\text{blast}$ 
show  $\text{thesis}$ 
proof
  show  $\text{closed } (-T)$ 
    using  $\langle \text{open } T \rangle$  by  $\text{blast}$ 
show  $-T \subseteq S$ 
    using  $\langle -S \subseteq T \rangle$  by  $\text{auto}$ 
show  $S - (-T) \in \text{lmeasurable}$   $\text{emeasure lebesgue } (S - (-T)) < e$ 
  using  $T$  by  $(\text{auto simp: Int\_commute})$ 
qed
qed
```


lemma *lebesgue_openin*:

$\llbracket \text{openin } (\text{top_of_set } S) \ T; \ S \in \text{sets lebesgue} \rrbracket \implies T \in \text{sets lebesgue}$
by (*metis borel_open openin_open sets.Int sets_completionI_sets sets_lborel*)

lemma *lebesgue_closedin*:

$\llbracket \text{closedin } (\text{top_of_set } S) \ T; \ S \in \text{sets lebesgue} \rrbracket \implies T \in \text{sets lebesgue}$
by (*metis borel_closed closedin_closed sets.Int sets_completionI_sets sets_lborel*)

7.12.9 F _sigma and G _delta sets.

inductive *fsigma* :: '*a*::topological_space set \Rightarrow bool **where**

$(\bigwedge n::\text{nat. closed } (F \ n)) \implies \text{fsigma } (\bigcup (F \ ' \ \text{UNIV}))$

inductive *gdelta* :: '*a*::topological_space set \Rightarrow bool **where**

$(\bigwedge n::\text{nat. open } (F \ n)) \implies \text{gdelta } (\bigcap (F \ ' \ \text{UNIV}))$

lemma *fsigma_UNIV* [*iff*]: *fsigma* (UNIV :: '*a*::real_inner set)

proof –

have (UNIV :: '*a* set) = $(\bigcup i. \text{cball } 0 \ (\text{of_nat } i))$

by (*auto simp: real_arch_simple*)

then show *?thesis*

by (*metis closed_cball fsigma.intros*)

qed

lemma *fsigma_Union_compact*:

fixes *S* :: '*a*::{real_normed_vector,heine_borel} set

shows *fsigma* *S* $\longleftrightarrow (\exists F::\text{nat} \Rightarrow 'a \text{ set. range } F \subseteq \text{Collect compact} \wedge S = \bigcup (F \ ' \ \text{UNIV}))$

proof *safe*

assume *fsigma* *S*

then obtain *F* :: *nat* \Rightarrow '*a* set **where** *F*: range *F* \subseteq Collect closed *S* = $\bigcup (F \ ' \ \text{UNIV})$

by (*meson fsigma.cases image_subsetI mem_Collect_eq*)

then have $\exists D::\text{nat} \Rightarrow 'a \text{ set. range } D \subseteq \text{Collect compact} \wedge \bigcup (D \ ' \ \text{UNIV}) = F$
i for i

using *closed_Union_compact_subsets* [*of F i*]

by (*metis image_subsetI mem_Collect_eq range_subsetD*)

then obtain *D* :: *nat* \Rightarrow *nat* \Rightarrow '*a* set

where *D*: $\bigwedge i. \text{range } (D \ i) \subseteq \text{Collect compact} \wedge \bigcup ((D \ i) \ ' \ \text{UNIV}) = F \ i$

by *metis*

let *?DD* = $\lambda n. (\lambda (i,j). D \ i \ j)$ (*prod_decode n*)

show $\exists F::\text{nat} \Rightarrow 'a \text{ set. range } F \subseteq \text{Collect compact} \wedge S = \bigcup (F \ ' \ \text{UNIV})$

proof (*intro exI conjI*)

show range *?DD* \subseteq Collect compact

using *D* **by** *clarsimp* (*metis mem_Collect_eq rangeI split_conv subsetCE surj_pair*)

show *S* = $\bigcup (\text{range } ?DD)$

proof

show *S* $\subseteq \bigcup (\text{range } ?DD)$

2502

```

    using D F
    by clarsimp (metis UN_iff old.prod.case prod_decode_inverse prod_encode_eq)
    show  $\bigcup (\text{range } ?DD) \subseteq S$ 
    using D F by fastforce
  qed
qed
next
  fix F :: nat  $\Rightarrow$  'a set
  assume range F  $\subseteq$  Collect compact and  $S = \bigcup (F \text{ ' UNIV})$ 
  then show fsigma ( $\bigcup (F \text{ ' UNIV})$ )
    by (simp add: compact_imp_closed fsigma.intros image_subset_iff)
  qed

```

```

lemma gdelta_imp_fsigma: gdelta S  $\implies$  fsigma ( $- S$ )
proof (induction rule: gdelta.induct)
  case (1 F)
  have  $- \bigcap (F \text{ ' UNIV}) = (\bigcup i. -(F i))$ 
    by auto
  then show ?case
    by (simp add: fsigma.intros closed_Compl 1)
  qed

```

```

lemma fsigma_imp_gdelta: fsigma S  $\implies$  gdelta ( $- S$ )
proof (induction rule: fsigma.induct)
  case (1 F)
  have  $- \bigcup (F \text{ ' UNIV}) = (\bigcap i. -(F i))$ 
    by auto
  then show ?case
    by (simp add: 1 gdelta.intros open_closed)
  qed

```

```

lemma gdelta_complement: gdelta( $- S$ )  $\longleftrightarrow$  fsigma S
  using fsigma_imp_gdelta gdelta_imp_fsigma by force

```

```

lemma lebesgue_set_almost_fsigma:
  assumes S  $\in$  sets lebesgue
  obtains C T where fsigma C T  $\in$  null_sets lebesgue C  $\cup$  T = S disjnt C T
proof -
  { fix n::nat
    obtain T where closed T T  $\subseteq$  S S-T  $\in$  lmeasurable emeasure lebesgue (S -
    T) < ennreal (1 / Suc n)
    using sets_lebesgue_inner_closed [OF assms]
    by (metis of_nat_0_less_iff zero_less_Suc zero_less_divide_1_iff)
    then have  $\exists T. \text{closed } T \wedge T \subseteq S \wedge S - T \in \text{lmeasurable} \wedge \text{measure lebesgue}
    (S-T) < 1 / \text{Suc } n$ 
    by (metis emeasure_eq_measure2 ennreal_leI not_le)
  }
  then obtain F where F:  $\bigwedge n::nat. \text{closed } (F n) \wedge F n \subseteq S \wedge S - F n \in
  \text{lmeasurable} \wedge \text{measure lebesgue } (S - F n) < 1 / \text{Suc } n$ 

```

```

  by metis
let ?C =  $\bigcup (F \text{ ' } UNIV)$ 
show thesis
proof
  show fsigma ?C
    using F by (simp add: fsigma.intros)
  show  $(S - ?C) \in \text{null\_sets lebesgue}$ 
  proof (clarsimp simp add: completion.null_sets_outer_le)
    fix e :: real
    assume  $0 < e$ 
    then obtain n where  $n: 1 / \text{Suc } n < e$ 
      using nat_approx_posE by metis
    show  $\exists T \in \text{lmeasurable}. S - (\bigcup x. F x) \subseteq T \wedge \text{measure lebesgue } T \leq e$ 
    proof (intro bexI conjI)
      show  $\text{measure lebesgue } (S - F n) \leq e$ 
        by (meson F n less_trans not_le order.asym)
      qed (use F in auto)
    qed
  qed
show ?C  $\cup (S - ?C) = S$ 
  using F by blast
show disjnt ?C  $(S - ?C)$ 
  by (auto simp: disjnt_def)
qed
qed

lemma lebesgue_set_almost_gdelta:
  assumes  $S \in \text{sets lebesgue}$ 
  obtains C T where  $gdelta C T \in \text{null\_sets lebesgue } S \cup T = C \text{ disjnt } S T$ 
proof -
  have  $-S \in \text{sets lebesgue}$ 
    using assms Compl_in_sets_lebesgue by blast
  then obtain C T where  $C: \text{fsigma } C T \in \text{null\_sets lebesgue } C \cup T = -S$ 
    disjnt C T
    using lebesgue_set_almost_fsigma by metis
  show thesis
  proof
    show  $gdelta (-C)$ 
      by (simp add:  $\langle \text{fsigma } C \rangle \text{fsigma\_imp\_gdelta}$ )
    show  $S \cup T = -C \text{ disjnt } S T$ 
      using C by (auto simp: disjnt_def)
    qed (use C in auto)
  qed
qed

end

```

7.13 Tagged Divisions for Henstock-Kurzweil Integration

```
theory Tagged_Division
  imports Topology_Euclidean_Space
begin
```

```
lemma sum_Sigma_product:
  assumes finite S
    and  $\bigwedge i. i \in S \implies \text{finite } (T\ i)$ 
  shows  $(\sum_{i \in S} \text{sum } (x\ i) (T\ i)) = (\sum_{(i, j) \in \text{Sigma } S\ T. x\ i\ j})$ 
  using assms
  by induction (auto simp: sum.Sigma)
```

```
lemmas scaleR_simps = scaleR_zero_left scaleR_minus_left scaleR_left_diff_distrib
  scaleR_zero_right scaleR_minus_right scaleR_right_diff_distrib scaleR_eq_0_iff
  scaleR_cancel_left scaleR_cancel_right scaleR_add_right scaleR_add_left real_vector_class.scaleR_c
```

7.13.1 Some useful lemmas about intervals

```
lemma interior_subset_union_intervals:
  fixes a b c d
  defines  $i \equiv \text{cbox } a\ b$ 
  defines  $j \equiv \text{cbox } c\ d$ 
  assumes interior  $j \neq \{\}$ 
    and  $i \subseteq j \cup S$ 
    and interior  $i \cap \text{interior } j = \{\}$ 
  shows interior  $i \subseteq \text{interior } S$ 
  by (smt (verit, del_insts) IntI Int_interval_mixed_eq_empty UnE assms empty_iff
  interior_cbox interior_maximal interior_subset open_interior subset_eq)
```

```
lemma interior_Union_subset_cbox:
  assumes finite  $\mathcal{F}$ 
  assumes  $\mathcal{F}: \bigwedge S. S \in \mathcal{F} \implies \exists a\ b. S = \text{cbox } a\ b \wedge S. S \in \mathcal{F} \implies \text{interior } S \subseteq T$ 
    and  $T: \text{closed } T$ 
  shows interior  $(\bigcup \mathcal{F}) \subseteq T$ 
```

```
proof -
  have clo[simp]:  $S \in \mathcal{F} \implies \text{closed } S$  for  $S$ 
    using  $\mathcal{F}$  by auto
  define E where  $E = \{s \in \mathcal{F}. \text{interior } s = \{\}\}$ 
  then have finite E  $E \subseteq \{s \in \mathcal{F}. \text{interior } s = \{\}\}$ 
    using  $\langle \text{finite } \mathcal{F} \rangle$  by auto
  then have interior  $(\bigcup \mathcal{F}) = \text{interior } (\bigcup (\mathcal{F} - E))$ 
  proof (induction E rule: finite_subset_induct')
    case (insert s f')
    have interior  $(\bigcup (\mathcal{F} - \text{insert } s\ f') \cup s) = \text{interior } (\bigcup (\mathcal{F} - \text{insert } s\ f'))$ 
      using insert.hyps  $\langle \text{finite } \mathcal{F} \rangle$  by (intro interior_closed_Un_empty_interior)
  auto
  also have  $\bigcup (\mathcal{F} - \text{insert } s\ f') \cup s = \bigcup (\mathcal{F} - f')$ 
```

```

    using insert.hyps by auto
  finally show ?case
    by (simp add: insert.IH)
qed simp
also have ...  $\subseteq \bigcup (\mathcal{F} - E)$ 
  by (rule interior_subset)
also have ...  $\subseteq T$ 
proof (rule Union_least)
  fix s assume s  $\in \mathcal{F} - E$ 
  with  $\mathcal{F}$ [of s] obtain a b where s: s  $\in \mathcal{F}$  s = cbox a b box a b  $\neq \{\}$ 
    by (fastforce simp: E_def)
  have closure (interior s)  $\subseteq$  closure T
    by (intro closure_mono  $\mathcal{F}$   $\langle s \in \mathcal{F} \rangle$ )
  with s  $\langle$ closed T $\rangle$  show s  $\subseteq T$ 
    by simp
qed
finally show ?thesis .
qed

```

lemma *Int_interior_Union_intervals:*

```

  [[finite  $\mathcal{F}$ ; open S;  $\bigwedge T. T \in \mathcal{F} \implies \exists a b. T = \text{cbox } a b; \bigwedge T. T \in \mathcal{F} \implies S \cap$ 
  (interior T) =  $\{\}$ ]]
   $\implies S \cap \text{interior } (\bigcup \mathcal{F}) = \{\}$ 
  using interior_Union_subset_cbox[of  $\mathcal{F}$  UNIV - S] by auto

```

lemma *interval_split:*

```

  fixes a :: 'a::euclidean_space
  assumes k  $\in$  Basis
  shows
    cbox a b  $\cap \{x. x \cdot k \leq c\} = \text{cbox } a (\sum_{i \in \text{Basis}. (\text{if } i = k \text{ then } \min (b \cdot k) c \text{ else } b \cdot i) *_{\mathbb{R}} i)$ 
    cbox a b  $\cap \{x. x \cdot k \geq c\} = \text{cbox } (\sum_{i \in \text{Basis}. (\text{if } i = k \text{ then } \max (a \cdot k) c \text{ else } a \cdot i) *_{\mathbb{R}} i) b$ 
  using assms by (rule_tac set_eqI; auto simp: mem_box)+

```

lemma *interval_not_empty:* $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies \text{cbox } a b \neq \{\}$

by (simp add: box_ne_empty)

7.13.2 Bounds on intervals where they exist

definition *interval_upperbound* :: ('a::euclidean_space) set \Rightarrow 'a
 where *interval_upperbound* s = $(\sum_{i \in \text{Basis}. (\text{SUP } x \in s. x \cdot i) *_{\mathbb{R}} i)$

definition *interval_lowerbound* :: ('a::euclidean_space) set \Rightarrow 'a
 where *interval_lowerbound* s = $(\sum_{i \in \text{Basis}. (\text{INF } x \in s. x \cdot i) *_{\mathbb{R}} i)$

lemma *interval_upperbound[simp]:*

```

 $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies$ 
  interval_upperbound (cbox a b) = (b::'a::euclidean_space)

```

unfolding *interval_upperbound_def euclidean_representation_sum cbox_def*
by (*safe intro!: cSup_eq*) *auto*

lemma *interval_lowerbound[simp]*:

$\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies$

$\text{interval_lowerbound} (\text{cbox } a \ b) = (a :: 'a :: \text{euclidean_space})$

unfolding *interval_lowerbound_def euclidean_representation_sum cbox_def*
by (*safe intro!: cInf_eq*) *auto*

lemmas *interval_bounds = interval_upperbound interval_lowerbound*

lemma

fixes *X::real set*

shows *interval_upperbound_real[simp]*: $\text{interval_upperbound } X = \text{Sup } X$

and *interval_lowerbound_real[simp]*: $\text{interval_lowerbound } X = \text{Inf } X$

by (*auto simp: interval_upperbound_def interval_lowerbound_def*)

lemma *interval_bounds'[simp]*:

assumes $\text{cbox } a \ b \neq \{\}$

shows $\text{interval_upperbound} (\text{cbox } a \ b) = b$

and $\text{interval_lowerbound} (\text{cbox } a \ b) = a$

using *assms unfolding box_ne_empty by auto*

lemma *interval_upperbound_Times*:

assumes $A \neq \{\}$ **and** $B \neq \{\}$

shows $\text{interval_upperbound} (A \times B) = (\text{interval_upperbound } A, \text{interval_upperbound } B)$

proof –

from *assms have fst_image_times': $A = \text{fst } '(A \times B)$ by simp*

have $(\sum i \in \text{Basis}. (\text{SUP } x \in A \times B. x \cdot (i, 0)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{SUP } x \in A. x \cdot i) *_{\mathbb{R}} i)$

by (*subst (2) fst_image_times'*) (*simp del: fst_image_times add: image_comp inner_Pair_0*)

moreover from *assms have snd_image_times': $B = \text{snd } '(A \times B)$ by simp*

have $(\sum i \in \text{Basis}. (\text{SUP } x \in A \times B. x \cdot (0, i)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{SUP } x \in B. x \cdot i) *_{\mathbb{R}} i)$

by (*subst (2) snd_image_times'*) (*simp del: snd_image_times add: image_comp inner_Pair_0*)

ultimately show *?thesis unfolding interval_upperbound_def*

by (*subst sum_Basis_prod_eq*) (*auto simp add: sum_prod*)

qed

lemma *interval_lowerbound_Times*:

assumes $A \neq \{\}$ **and** $B \neq \{\}$

shows $\text{interval_lowerbound} (A \times B) = (\text{interval_lowerbound } A, \text{interval_lowerbound } B)$

proof –

from *assms have fst_image_times': $A = \text{fst } '(A \times B)$ by simp*

have $(\sum i \in \text{Basis}. (\text{INF } x \in A \times B. x \cdot (i, 0)) *_{\mathbb{R}} i) = (\sum i \in \text{Basis}. (\text{INF } x \in A. x$

$\cdot i) *_R i)$
by (subst (2) fst_image_times') (simp del: fst_image_times add: image_comp inner_Pair_0)
moreover from *assms* **have** snd_image_times': $B = \text{snd } '(A \times B)$ **by** *simp*
have $(\sum_{i \in \text{Basis.}} (\text{INF } x \in A \times B. x \cdot (0, i)) *_R i) = (\sum_{i \in \text{Basis.}} (\text{INF } x \in B. x \cdot i) *_R i)$
by (subst (2) snd_image_times') (simp del: snd_image_times add: image_comp inner_Pair_0)
ultimately show ?thesis **unfolding** interval_lowerbound_def
by (subst sum_Basis_prod_eq) (auto simp add: sum_prod)
qed

7.13.3 The notion of a gauge — simply an open set containing the point

definition *gauge* $\gamma \longleftrightarrow (\forall x. x \in \gamma x \wedge \text{open } (\gamma x))$

lemma *gaugeI*:
assumes $\bigwedge x. x \in \gamma x$ **and** $\bigwedge x. \text{open } (\gamma x)$
shows *gauge* γ
using *assms* **unfolding** *gauge_def* **by** *auto*

lemma *gaugeD[dest]*:
assumes *gauge* γ **shows** $x \in \gamma x$
and $\text{open } (\gamma x)$
using *assms* **unfolding** *gauge_def* **by** *auto*

lemma *gauge_ball_dependent*: $\forall x. 0 < e x \implies \text{gauge } (\lambda x. \text{ball } x (e x))$
unfolding *gauge_def* **by** *auto*

lemma *gauge_ball[intro]*: $0 < e \implies \text{gauge } (\lambda x. \text{ball } x e)$
unfolding *gauge_def* **by** *auto*

lemma *gauge_trivial[intro!]*: *gauge* $(\lambda x. \text{ball } x 1)$
by *auto*

lemma *gauge_Int[intro]*: *gauge* $\gamma 1 \implies \text{gauge } \gamma 2 \implies \text{gauge } (\lambda x. \gamma 1 x \cap \gamma 2 x)$
unfolding *gauge_def* **by** *auto*

lemma *gauge_reflect*:
fixes $\gamma :: 'a :: \text{euclidean_space} \Rightarrow 'a \text{ set}$
shows *gauge* $\gamma \implies \text{gauge } (\lambda x. \text{uminus } ' \gamma (- x))$
by (*metis* (*mono_tags*, *lifting*) *gauge_def imageI open_negations minus_minus*)

lemma *gauge_Inter*:
assumes *finite* S
and $\bigwedge u. u \in S \implies \text{gauge } (f u)$
shows *gauge* $(\lambda x. \bigcap \{f u x \mid u. u \in S\})$
proof –

2508

```

have *:  $\bigwedge x. \{f\ u\ x \mid u. u \in S\} = (\lambda u. f\ u\ x) \text{ ` } S$ 
  by auto
show ?thesis
  unfolding gauge_def unfolding *
  by (simp add: assms gaugeD open_INT)
qed

```

```

lemma gauge_existence_lemma:
   $(\forall x. \exists d :: \text{real}. p\ x \longrightarrow 0 < d \wedge q\ d\ x) \longleftrightarrow (\forall x. \exists d > 0. p\ x \longrightarrow q\ d\ x)$ 
  by (metis zero_less_one)

```

7.13.4 Attempt a systematic general set of "offset" results for components

```

lemma gauge_modify:
  assumes  $(\forall S. \text{open } S \longrightarrow \text{open } \{x. f(x) \in S\})$  gauge d
  shows gauge  $(\lambda x. \{y. f\ y \in d\ (f\ x)\})$ 
  using assms unfolding gauge_def by force

```

7.13.5 Divisions

definition *division_of* (infixl *division'_of* 40)

where

```

s division_of i  $\longleftrightarrow$ 
  finite s  $\wedge$ 
   $(\forall K \in s. K \subseteq i \wedge K \neq \{\}) \wedge (\exists a\ b. K = \text{cbox } a\ b) \wedge$ 
   $(\forall K1 \in s. \forall K2 \in s. K1 \neq K2 \longrightarrow \text{interior}(K1) \cap \text{interior}(K2) = \{\}) \wedge$ 
   $(\bigcup s = i)$ 

```

lemma *division_ofD[dest]*:

```

assumes s division_of i
shows finite s
  and  $\bigwedge K. K \in s \Longrightarrow K \subseteq i$ 
  and  $\bigwedge K. K \in s \Longrightarrow K \neq \{\}$ 
  and  $\bigwedge K. K \in s \Longrightarrow \exists a\ b. K = \text{cbox } a\ b$ 
  and  $\bigwedge K1\ K2. K1 \in s \Longrightarrow K2 \in s \Longrightarrow K1 \neq K2 \Longrightarrow \text{interior}(K1) \cap \text{interior}(K2) = \{\}$ 
  and  $\bigcup s = i$ 
using assms unfolding division_of_def by auto

```

lemma *division_ofI*:

```

assumes finite s
  and  $\bigwedge K. K \in s \Longrightarrow K \subseteq i$ 
  and  $\bigwedge K. K \in s \Longrightarrow K \neq \{\}$ 
  and  $\bigwedge K. K \in s \Longrightarrow \exists a\ b. K = \text{cbox } a\ b$ 
  and  $\bigwedge K1\ K2. K1 \in s \Longrightarrow K2 \in s \Longrightarrow K1 \neq K2 \Longrightarrow \text{interior } K1 \cap \text{interior } K2 = \{\}$ 
  and  $\bigcup s = i$ 
shows s division_of i

```


using *assms* **unfolding** *division_of_def* **by** *auto*

lemma *division_of_finite*: $s \text{ division_of } i \implies \text{finite } s$
by *auto*

lemma *division_of_self*[*intro*]: $\text{cbox } a \ b \neq \{\} \implies \{\text{cbox } a \ b\} \text{ division_of } (\text{cbox } a \ b)$
unfolding *division_of_def* **by** *auto*

lemma *division_of_trivial*[*simp*]: $s \text{ division_of } \{\} \longleftrightarrow s = \{\}$
unfolding *division_of_def* **by** *auto*

lemma *division_of_sing*[*simp*]:
 $s \text{ division_of } \text{cbox } a \ (a::'a::\text{euclidean_space}) \longleftrightarrow s = \{\text{cbox } a \ a\}$
(is *?l* = *?r*)

proof

assume *?l*

have $x = \{a\}$ **if** $x \in s$ **for** x

by (*metis* $\langle s \text{ division_of } \text{cbox } a \ a \rangle$ *cbox_idem* *division_ofD(2)* *division_ofD(3)* *subset_singletonD* *that*)

moreover have $s \neq \{\}$

using $\langle s \text{ division_of } \text{cbox } a \ a \rangle$ **by** *auto*

ultimately show *?r*

unfolding *cbox_idem* **by** *auto*

qed (*use* *cbox_idem* **in** *blast*)

lemma *elementary_empty*: **obtains** p **where** $p \text{ division_of } \{\}$
by *simp*

lemma *elementary_interval*: **obtains** p **where** $p \text{ division_of } (\text{cbox } a \ b)$
by (*metis* *division_of_trivial* *division_of_self*)

lemma *division_contains*: $s \text{ division_of } i \implies \forall x \in i. \exists k \in s. x \in k$
unfolding *division_of_def* **by** *auto*

lemma *forall_in_division*:
 $d \text{ division_of } i \implies (\forall x \in d. P \ x) \longleftrightarrow (\forall a \ b. \text{cbox } a \ b \in d \longrightarrow P \ (\text{cbox } a \ b))$
unfolding *division_of_def* **by** *fastforce*

lemma *cbox_division_memE*:
assumes \mathcal{D} : $K \in \mathcal{D}$ $\mathcal{D} \text{ division_of } S$
obtains $a \ b$ **where** $K = \text{cbox } a \ b$ $K \neq \{\}$ $K \subseteq S$
using *assms* **unfolding** *division_of_def* **by** *metis*

lemma *division_of_subset*:
assumes $p \text{ division_of } (\bigcup p)$
and $q \subseteq p$
shows $q \text{ division_of } (\bigcup q)$
proof (*rule* *division_ofI*)

2510

```

    show finite q
      using assms finite_subset by blast
next
  fix k
  assume k ∈ q
  show k ⊆ ⋃ q
    using ⟨k ∈ q⟩ by auto
  show ∃ a b. k = cbox a b k ≠ {}
    using assms ⟨k ∈ q⟩ by blast+
next
  fix k1 k2
  assume k1 ∈ q k2 ∈ q k1 ≠ k2
  then show interior k1 ∩ interior k2 = {}
    using assms by blast
qed auto

lemma division_of_union_self[intro]: p division_of s ⇒ p division_of (⋃ p)
  by blast

lemma division_inter:
  fixes s1 s2 :: 'a::euclidean_space set
  assumes p1 division_of s1
    and p2 division_of s2
  shows {k1 ∩ k2 | k1 k2. k1 ∈ p1 ∧ k2 ∈ p2 ∧ k1 ∩ k2 ≠ {}} division_of (s1
  ∩ s2)
  (is ?A' division_of _)
proof -
  let ?A = {s. s ∈ (λ(k1,k2). k1 ∩ k2) ‘ (p1 × p2) ∧ s ≠ {}}
  have *: ?A' = ?A by auto
  show ?thesis
    unfolding *
  proof (rule division_ofI)
    have ?A ⊆ (λ(x, y). x ∩ y) ‘ (p1 × p2)
      by auto
    moreover have finite (p1 × p2)
      using assms unfolding division_of_def by auto
    ultimately show finite ?A by auto
  have *: ⋂ s. ⋃ {x ∈ s. x ≠ {}} = ⋃ s
    by auto
  show UA: ⋃ ?A = s1 ∩ s2
    unfolding *
    using division_ofD(6)[OF assms(1)] and division_ofD(6)[OF assms(2)] by
  auto
  {
    fix k
    assume kA: k ∈ ?A
    then obtain k1 k2 where k: k = k1 ∩ k2 k1 ∈ p1 k2 ∈ p2 k ≠ {}
      by auto
    then show k ≠ {}
  }

```

```

    by auto
  show  $k \subseteq s1 \cap s2$ 
    using  $UA\ kA$  by blast
  show  $\exists a\ b. k = \text{cbox } a\ b$ 
    using  $k$  by (metis (no_types, lifting) Int_interval assms division_ofD(4))
}
fix  $k1\ k2$ 
assume  $k1 \in ?A$ 
then obtain  $x1\ y1$  where  $k1: k1 = x1 \cap y1\ x1 \in p1\ y1 \in p2\ k1 \neq \{\}$ 
  by auto
assume  $k2 \in ?A$ 
then obtain  $x2\ y2$  where  $k2: k2 = x2 \cap y2\ x2 \in p1\ y2 \in p2\ k2 \neq \{\}$ 
  by auto
assume  $k1 \neq k2$ 
then show  $\text{interior } k1 \cap \text{interior } k2 = \{\}$ 
  unfolding  $k1\ k2$ 
  using assms division_ofD(5)  $k1\ k2$  by auto
qed
qed

```

```

lemma division_inter_1:
  assumes  $d$  division_of  $i$ 
    and  $\text{cbox } a\ (b::'a::\text{euclidean\_space}) \subseteq i$ 
  shows  $\{\text{cbox } a\ b \cap k \mid k. k \in d \wedge \text{cbox } a\ b \cap k \neq \{\}\}$  division_of  $(\text{cbox } a\ b)$ 
proof (cases  $\text{cbox } a\ b = \{\}$ )
  case True
  show ?thesis
    unfolding True and division_of_trivial by auto
  next
  case False
  have *:  $\text{cbox } a\ b \cap i = \text{cbox } a\ b$  using assms(2) by auto
  show ?thesis
    using division_inter[OF division_of_self[OF False] assms(1)]
    unfolding * by auto
qed

```

```

lemma elementary_Int:
  fixes  $s\ t :: 'a::\text{euclidean\_space}$  set
  assumes  $p1$  division_of  $s$  and  $p2$  division_of  $t$ 
  shows  $\exists p. p$  division_of  $(s \cap t)$ 
using assms division_inter by blast

```

```

lemma elementary_Inter:
  assumes finite  $\mathcal{F}$ 
    and  $\mathcal{F} \neq \{\}$ 
    and  $\forall s \in \mathcal{F}. \exists p. p$  division_of  $(s::('a::\text{euclidean\_space})\ \text{set})$ 
  shows  $\exists p. p$  division_of  $(\bigcap \mathcal{F})$ 
  using assms
proof (induct  $\mathcal{F}$  rule: finite_induct)

```

2512

```

case (insert x F)
then show ?case
  by (metis cInf_singleton complete_lattice_class.Inf_insert elementary_Int insert_iff)
qed auto

```

```

lemma division_disjoint_union:
  assumes p1 division_of s1
    and p2 division_of s2
    and interior s1 ∩ interior s2 = {}
  shows (p1 ∪ p2) division_of (s1 ∪ s2)
proof (rule division_ofI)
  note d1 = division_ofD[OF assms(1)]
  note d2 = division_ofD[OF assms(2)]
  fix k1 k2
  assume k1 ∈ p1 ∪ p2 k2 ∈ p1 ∪ p2 k1 ≠ k2
  with assms show interior k1 ∩ interior k2 = {}
  by (smt (verit, best) IntE Un_iff disjoint_iff_not_equal division_ofD(2) division_ofD(5) inf.orderE interior_Int)
qed (use division_ofD assms in auto)

```

```

lemma partial_division_extend_1:
  fixes a b c d :: 'a::euclidean_space
  assumes incl: cbox c d ⊆ cbox a b
    and nonempty: cbox c d ≠ {}
  obtains p where p division_of (cbox a b) cbox c d ∈ p
proof
  let ?B =  $\lambda f::'a \Rightarrow 'a \times 'a.$ 
    cbox ( $\sum i \in \text{Basis}. (\text{fst } (f i) \cdot i) *_R i$ ) ( $\sum i \in \text{Basis}. (\text{snd } (f i) \cdot i) *_R i$ )
  define p where p = ?B ‘ (Basis  $\rightarrow_E$  {(a, c), (c, d), (d, b)})

  show cbox c d ∈ p
    unfolding p_def
    by (auto simp add: box_eq_empty cbox_def intro!: image_eqI [where x =  $\lambda (i::'a) \in \text{Basis}. (c, d)$ ])
  have ord: a · i ≤ c · i c · i ≤ d · i d · i ≤ b · i if i ∈ Basis for i
    using incl nonempty that
    unfolding box_eq_empty subset_box by (auto simp: not_le)

  show p division_of (cbox a b)
proof (rule division_ofI)
  show finite p
    unfolding p_def by (auto intro!: finite_PiE)
  {
  fix K
  assume K ∈ p
  then obtain f where f: f ∈ Basis  $\rightarrow_E$  {(a, c), (c, d), (d, b)} and k: K =
  ?B f
    by (auto simp: p_def)
  }

```

```

then show  $\exists a b. K = \text{cbox } a b$ 
  by auto
{
  fix l
  assume  $l \in p$ 
  then obtain g where  $g: g \in \text{Basis} \rightarrow_E \{(a, c), (c, d), (d, b)\}$  and  $l: l =$ 
?B g
    by (auto simp: p_def)
  assume  $l \neq K$ 
  have  $\exists i \in \text{Basis}. f i \neq g i$ 
    using  $\langle l \neq K \rangle l k f g$  by auto
  then obtain i where  $*: i \in \text{Basis} f i \neq g i ..$ 
  then have  $f i = (a, c) \vee f i = (c, d) \vee f i = (d, b)$ 
     $g i = (a, c) \vee g i = (c, d) \vee g i = (d, b)$ 
    using f g by (auto simp: PiE_iff)
  with * ord[of i] show  $\text{interior } l \cap \text{interior } K = \{\}$ 
    by (auto simp add: l k disjoint_interval intro!: bexI[of _ i])
}
have  $a \cdot i \leq \text{fst } (f i) \cdot i \text{ snd } (f i) \cdot i \leq b \cdot i \text{ fst } (f i) \cdot i \leq \text{snd } (f i) \cdot i$ 
  if  $i \in \text{Basis}$  for i
proof -
  have  $f i = (a, c) \vee f i = (c, d) \vee f i = (d, b)$ 
    using that f by (auto simp: PiE_iff)
  with that ord[of i]
  show  $a \cdot i \leq \text{fst } (f i) \cdot i \text{ snd } (f i) \cdot i \leq b \cdot i \text{ fst } (f i) \cdot i \leq \text{snd } (f i) \cdot i$ 
    by auto
qed
then show  $K \neq \{\}$   $K \subseteq \text{cbox } a b$ 
  by (auto simp add: k box_eq_empty subset_box not_less)
}
moreover
have  $\exists k \in p. x \in k$  if  $x: x \in \text{cbox } a b$  for x
proof -
  have  $\exists l. x \cdot i \in \{\text{fst } l \cdot i .. \text{snd } l \cdot i\} \wedge l \in \{(a, c), (c, d), (d, b)\}$  if  $i \in$ 
Basis for i
  proof -
    have  $(a \cdot i \leq x \cdot i \wedge x \cdot i \leq c \cdot i) \vee (c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i) \vee$ 
       $(d \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$ 
      using that x ord[of i]
      by (auto simp: cbox_def)
    then show  $\exists l. x \cdot i \in \{\text{fst } l \cdot i .. \text{snd } l \cdot i\} \wedge l \in \{(a, c), (c, d), (d, b)\}$ 
      by auto
  qed
  then obtain f where
     $f: \forall i \in \text{Basis}. x \cdot i \in \{\text{fst } (f i) \cdot i .. \text{snd } (f i) \cdot i\} \wedge f i \in \{(a, c), (c, d), (d, b)\}$ 
    by metis
  moreover from f have  $x \in ?B (\text{restrict } f \text{ Basis}) \text{ restrict } f \text{ Basis} \in \text{Basis} \rightarrow_E$ 
 $\{(a,c), (c,d), (d,b)\}$ 
    by (auto simp: mem_box)

```

```

ultimately show ?thesis
  unfolding p_def by blast
qed
ultimately show  $\bigcup p = \text{cbox } a \ b$ 
  by auto
qed
qed

proposition partial_division_extend_interval:
  assumes  $p \text{ division\_of } (\bigcup p) (\bigcup p) \subseteq \text{cbox } a \ b$ 
  obtains  $q$  where  $p \subseteq q$   $q \text{ division\_of cbox } a \ b$  ( $b::'a::\text{euclidean\_space}$ )
proof (cases  $p = \{\}$ )
  case True
  then show ?thesis
    using elementary_interval that by auto
next
  case False
  note  $p = \text{division\_ofD}[OF \text{assms}(1)]$ 
  have  $\forall k \in p. \exists q. q \text{ division\_of cbox } a \ b \wedge k \in q$ 
  proof
    fix  $k$ 
    assume  $kp: k \in p$ 
    obtain  $c \ d$  where  $k: k = \text{cbox } c \ d$ 
      using  $p(4)[OF kp]$  by blast
    have  $*$ :  $\text{cbox } c \ d \subseteq \text{cbox } a \ b \ \text{cbox } c \ d \neq \{\}$ 
      using  $p(2,3)[OF kp, \text{unfolded } k]$  using  $\text{assms}(2)$ 
      by (blast intro: order.trans)+
    obtain  $q$  where  $q \text{ division\_of cbox } a \ b \ \text{cbox } c \ d \in q$ 
      by (rule partial_division_extend_1[OF *])
    then show  $\exists q. q \text{ division\_of cbox } a \ b \wedge k \in q$ 
      unfolding  $k$  by auto
  qed
  then obtain  $q$  where  $q: \bigwedge x. x \in p \implies q \text{ division\_of cbox } a \ b \wedge x. x \in p \implies$ 
 $x \in q$ 
    by metis
  have  $q \text{ division\_of } \bigcup (q \ x)$  if  $x: x \in p$  for  $x$ 
    using  $q(1) \ x$  by blast
  then have  $di: \bigwedge x. x \in p \implies \exists d. d \text{ division\_of } \bigcup (q \ x - \{x\})$ 
    by (meson Diff_subset division_of_subset)
  have  $\forall s \in (\lambda i. \bigcup (q \ i - \{i\})) \ ' p. \exists d. d \text{ division\_of } s$ 
    using  $di$  by blast
  then obtain  $d$  where  $d: d \text{ division\_of } \bigcap ((\lambda i. \bigcup (q \ i - \{i\})) \ ' p)$ 
    by (meson False elementary_Inter finite_imageI image_is_empty p(1))
  have  $d \cup p \text{ division\_of cbox } a \ b$ 
  proof -
    have  $te: \bigwedge S \ f \ T. S \neq \{\} \implies \forall i \in S. f \ i \cup i = T \implies T = \bigcap (f \ ' S) \cup \bigcup S$  by
  auto
    have  $\text{cbox\_eq}: \text{cbox } a \ b = \bigcap ((\lambda i. \bigcup (q \ i - \{i\})) \ ' p) \cup \bigcup p$ 
  proof (rule  $te[OF \text{False}]$ , clarify)

```

```

    fix i
    assume i ∈ p
    then show  $\bigcup (q\ i - \{i\}) \cup i = \text{cbox } a\ b$ 
      by (metis Un_commute complete_lattice_class.Sup_insert division_ofD(6)
insert_Diff q)
    qed
    have [simp]: interior  $(\bigcap_{i \in p}. \bigcup (q\ i - \{i\})) \cap \text{interior } K = \{\}$  if  $K: K \in p$ 
  for K
    proof -
    note qk = division_ofD[OF q(1)[OF K]]
    have *:  $\bigwedge U\ T\ S. T \cap S = \{\} \implies U \subseteq S \implies U \cap T = \{\}$ 
      by auto
    show ?thesis
    proof (rule *[OF Int_interior_Union_intervals])
      show  $\bigwedge T. T \in q\ K - \{K\} \implies \text{interior } K \cap \text{interior } T = \{\}$ 
        using K q(2) qk(5) by auto
      show interior  $(\bigcap_{i \in p}. \bigcup (q\ i - \{i\})) \subseteq \text{interior } (\bigcup (q\ K - \{K\}))$ 
        by (meson INF_lower K interior_mono)
    qed (use qk in auto)
    qed
    show d  $\cup p$  division_of (cbox a b)
    by (simp add: Int_interior_Union_intervals assms(1) cbox_eq d division_disjoint_union
p(1) p(4))
    qed
    then show ?thesis
      by (meson Un_upper2 that)
  qed

```

```

lemma elementary_bounded[dest]:
  shows p division_of S  $\implies$  bounded S
  unfolding division_of_def by (metis bounded_Union bounded_cbox)

```

```

lemma elementary_subset_cbox:
  p division_of S  $\implies$   $\exists a\ b. S \subseteq \text{cbox } a\ b$ 
  by (meson bounded_subset_cbox_symmetric elementary_bounded)

```

```

proposition division_union_intervals_exists:
  assumes cbox a b  $\neq \{\}$ 
  obtains p where (insert (cbox a b) p) division_of (cbox a b  $\cup$  cbox c d)
proof (cases cbox c d =  $\{\}$ )
  case True
  with assms that show ?thesis by force
next
  case False
  show ?thesis
  proof (cases cbox a b  $\cap$  cbox c d =  $\{\}$ )
    case True
    then show ?thesis
      by (metis that False assms division_disjoint_union division_of_self in-

```

```

sert_is_Un interior_Int interior_empty)
next
  case False
  obtain u v where uv: cbox a b  $\cap$  cbox c d = cbox u v
  unfolding Int_interval by auto
  have uv_sub: cbox u v  $\subseteq$  cbox c d using uv by auto
  obtain p where pd: p division_of cbox c d and cbox u v  $\in$  p
  by (rule partial_division_extend_1[OF uv_sub False[unfolded uv]])
  note p = this division_ofD[OF pd]
  have interior (cbox a b  $\cap$   $\bigcup$  (p - {cbox u v})) = interior(cbox u v)  $\cap$  interior
  ( $\bigcup$  (p - {cbox u v}))
  by (metis Diff_subset Int_absorb1 Int_assoc Sup_subset_mono interior_Int
  p(8) uv)
  also have ... = {}
  using p(6) p(7)[OF p(2)] <finite p>
  by (intro Int_interior_Union_intervals) auto
  finally have disj: interior (cbox a b)  $\cap$  interior ( $\bigcup$  (p - {cbox u v})) = {}
  by simp
  have cbe: cbox a b  $\cup$  cbox c d = cbox a b  $\cup$   $\bigcup$  (p - {cbox u v})
  using p(8) unfolding uv[symmetric] by auto
  have insert (cbox a b) (p - {cbox u v}) division_of cbox a b  $\cup$   $\bigcup$  (p - {cbox u
  v})
  by (metis Diff_subset assms disj division_disjoint_union division_of_self
  division_of_subset insert_is_Un p(8) pd)
  with that[of p - {cbox u v}] show ?thesis
  by (simp add: cbe)
qed
qed

```

lemma *division_of_Union:*

```

assumes finite f
  and  $\bigwedge p. p \in f \implies p$  division_of ( $\bigcup p$ )
  and  $\bigwedge k1 k2. k1 \in \bigcup f \implies k2 \in \bigcup f \implies k1 \neq k2 \implies$  interior k1  $\cap$  interior
  k2 = {}
shows  $\bigcup f$  division_of  $\bigcup$  ( $\bigcup f$ )
using assms by (auto intro!: division_ofI)

```

lemma *elementary_union_interval:*

```

fixes a b :: 'a::euclidean_space
assumes p division_of  $\bigcup p$ 
obtains q where q division_of (cbox a b  $\cup$   $\bigcup p$ )
proof (cases p = {}  $\vee$  cbox a b = {})
  case True
  obtain p where p division_of (cbox a b)
  by (rule elementary_interval)
  then show thesis
  using True assms that by auto
next
  case False

```



```

then have  $p \neq \{\}$   $cbox\ a\ b \neq \{\}$ 
  by auto
note  $pdiv = division\_ofD[OF\ assms]$ 
show  $?thesis$ 
proof (cases interior (cbox a b) =  $\{\}$ )
  case True
    show  $?thesis$ 
    by (metis True assms division_disjoint_union elementary_interval inf_bot_left
that)
  next
    case nonempty: False
    have  $\forall K \in p. \exists q. (insert\ (cbox\ a\ b)\ q)\ division\_of\ (cbox\ a\ b \cup K)$ 
      by (metis  $\langle cbox\ a\ b \neq \{\} \rangle$  division_union_intervals_exists pdiv(4))
    then obtain  $q$  where  $\forall x \in p. insert\ (cbox\ a\ b)\ (q\ x)\ division\_of\ (cbox\ a\ b) \cup$ 
x
      by metis
    note  $q = division\_ofD[OF\ this[rule\_format]]$ 
    let  $?D = \bigcup \{insert\ (cbox\ a\ b)\ (q\ K) \mid K. K \in p\}$ 
    show  $thesis$ 
    proof (rule that[OF division_ofI])
      have  $*$ :  $\{insert\ (cbox\ a\ b)\ (q\ K) \mid K. K \in p\} = (\lambda K. insert\ (cbox\ a\ b)\ (q\ K))$ 
      ‘ p
      by auto
    show finite  $?D$ 
      using  $*\ pdiv(1)\ q(1)$  by auto
    have  $\bigcup ?D = (\bigcup x \in p. \bigcup (insert\ (cbox\ a\ b)\ (q\ x)))$ 
      by auto
    also have  $\dots = \bigcup \{cbox\ a\ b \cup t \mid t. t \in p\}$ 
      using  $q(6)$  by auto
    also have  $\dots = cbox\ a\ b \cup \bigcup p$ 
      using  $\langle p \neq \{\} \rangle$  by auto
    finally show  $\bigcup ?D = cbox\ a\ b \cup \bigcup p$  .
    show  $K \subseteq cbox\ a\ b \cup \bigcup p\ K \neq \{\}$  if  $K \in ?D$  for  $K$ 
      using  $q$  that by blast+
    show  $\exists a\ b. K = cbox\ a\ b$  if  $K \in ?D$  for  $K$ 
      using  $q(4)$  that by auto
  next
    fix  $K'\ K$ 
    assume  $K: K \in ?D$  and  $K': K' \in ?D\ K \neq K'$ 
    obtain  $x$  where  $x: K \in insert\ (cbox\ a\ b)\ (q\ x)\ x \in p$ 
      using  $K$  by auto
    obtain  $x'$  where  $x': K' \in insert\ (cbox\ a\ b)\ (q\ x')\ x' \in p$ 
      using  $K'$  by auto
    show interior  $K \cap$  interior  $K' = \{\}$ 
    proof (cases  $x = x' \vee K = cbox\ a\ b \vee K' = cbox\ a\ b$ )
      case True
        then show  $?thesis$ 
          using True  $K'\ q(5)\ x'\ x$  by auto
      next

```

```

case False
then have as':  $K \neq \text{cbox } a \ b$   $K' \neq \text{cbox } a \ b$ 
  by auto
obtain c d where  $K: K = \text{cbox } c \ d$ 
  using  $q(4) \ x$  by blast
have  $\text{interior } K \cap \text{interior } (\text{cbox } a \ b) = \{\}$ 
  using as'  $K'(2) \ q(5) \ x$  by blast
then have  $\text{interior } K \subseteq \text{interior } x$ 
by (metis  $\langle \text{interior } (\text{cbox } a \ b) \neq \{\} \rangle K \ q(2) \ x$  interior_subset_union_intervals)
moreover
obtain c' d' where  $K': K' = \text{cbox } c' \ d'$ 
  using  $q(4) \ x'(1) \ x'(2)$  by presburger
have  $\text{interior } K' \cap \text{interior } (\text{cbox } a \ b) = \{\}$ 
  using as'(2)  $q(5) \ x'$  by blast
then have  $\text{interior } K' \subseteq \text{interior } x'$ 
  by (metis nonempty  $c' \ d'$  interior_subset_union_intervals  $q(2) \ x'$ )
moreover have  $\text{interior } x \cap \text{interior } x' = \{\}$ 
  by (meson False assms division_ofD(5)  $x'(2) \ x(2)$ )
ultimately show ?thesis
  using  $\langle \text{interior } K \subseteq \text{interior } x \rangle \langle \text{interior } K' \subseteq \text{interior } x' \rangle$  by auto
qed
qed
qed
qed

```

```

lemma elementary_unions_intervals:
  assumes fin: finite  $\mathcal{F}$ 
    and  $\bigwedge s. s \in \mathcal{F} \implies \exists a \ b. s = \text{cbox } a \ (b::'a::\text{euclidean\_space})$ 
  obtains p where p division_of  $(\bigcup \mathcal{F})$ 
proof -
  have  $\exists p. p \ \text{division\_of} \ (\bigcup \mathcal{F})$ 
proof (induct_tac  $\mathcal{F}$  rule:finite_subset_induct)
  show  $\exists p. p \ \text{division\_of} \ \bigcup \{\}$  using elementary_empty by auto
next
  fix x F
  assume as: finite  $F \ x \notin F \ \exists p. p \ \text{division\_of} \ \bigcup F \ x \in F$ 
  then obtain a b where  $x: x = \text{cbox } a \ b$ 
    by (meson assms(2))
  then show  $\exists p. p \ \text{division\_of} \ \bigcup (\text{insert } x \ F)$ 
    using elementary_union_interval by (metis Union_insert as(3) division_ofD(6))
qed (use assms in auto)
then show ?thesis
  using that by auto
qed

```

```

lemma elementary_union:
  assumes ps division_of  $S$  pt division_of  $T$ 

```

```

obtains  $p$  where  $p$  division_of  $(S \cup T)$ 
proof -
  have  $S \cup T = \bigcup ps \cup \bigcup pt$ 
    using assms unfolding division_of_def by auto
  with elementary_unions_intervals[of ps  $\cup$  pt] assms
  show ?thesis
    by (metis Un_iff Union_Un_distrib division_of_def finite_Un that)
qed

lemma partial_division_extend:
  fixes  $T :: 'a::euclidean\_space$  set
  assumes  $p$  division_of  $S$ 
    and  $q$  division_of  $T$ 
    and  $S \subseteq T$ 
  obtains  $r$  where  $p \subseteq r$  and  $r$  division_of  $T$ 
proof -
  note  $divp = \text{division\_ofD}[OF\ assms(1)]$  and  $divq = \text{division\_ofD}[OF\ assms(2)]$ 
  obtain  $a\ b$  where  $ab: T \subseteq \text{cbox } a\ b$ 
    using elementary_subset_cbox[OF assms(2)] by auto
  obtain  $r1$  where  $p \subseteq r1$   $r1$  division_of  $(\text{cbox } a\ b)$ 
    using assms
  by (metis ab dual_order.trans partial_division_extend_interval divp(6))
  note  $r1 = \text{this division\_ofD}[OF\ \text{this}(2)]$ 
  obtain  $p'$  where  $p'$  division_of  $\bigcup(r1 - p)$ 
    by (metis Diff_subset division_of_subset r1(2) r1(8))
  then obtain  $r2$  where  $r2: r2$  division_of  $(\bigcup(r1 - p)) \cap (\bigcup q)$ 
    by (metis assms(2) divq(6) elementary_Int)
  {
    fix  $x$ 
    assume  $x: x \in T\ x \notin S$ 
    then obtain  $R$  where  $r: R \in r1\ x \in R$ 
      unfolding  $r1$  using  $ab$ 
      by (meson division_contains r1(2) subsetCE)
    moreover have  $R \notin p$ 
      using  $divp(6)\ r(2)\ x(2)$  by blast
    ultimately have  $x \in \bigcup(r1 - p)$  by auto
  }
  then have  $Teq: T = \bigcup p \cup (\bigcup(r1 - p) \cap \bigcup q)$ 
    unfolding  $divp\ divq$  using assms(3) by auto
  have interior  $S \cap \text{interior } (\bigcup(r1 - p)) = \{\}$ 
proof (rule Int_interior_Union_intervals)
  have  $*$ :  $\bigwedge S. (\bigwedge x. x \in S \implies \text{False}) \implies S = \{\}$ 
    by auto
  show interior  $S \cap \text{interior } m = \{\}$  if  $m \in r1 - p$  for  $m$ 
proof -
  have interior  $m \cap \text{interior } (\bigcup p) = \{\}$ 
  proof (rule Int_interior_Union_intervals)
  show  $\bigwedge T. T \in p \implies \text{interior } m \cap \text{interior } T = \{\}$ 
    by (metis DiffD1 DiffD2 that r1(1) r1(7) rev_subsetD)

```

```

qed (use divp in auto)
then show interior  $S \cap$  interior  $m = \{\}$ 
  unfolding divp by auto
qed
qed (use r1 in auto)
then have interior  $S \cap$  interior  $(\bigcup(r1-p) \cap (\bigcup q)) = \{\}$ 
  using interior_subset by auto
then have div:  $p \cup r2$  division_of  $\bigcup p \cup \bigcup(r1 - p) \cap \bigcup q$ 
  by (simp add: assms(1) division_disjoint_union divp(6) r2)
show ?thesis
  by (metis Teq div sup_ge1 that)
qed

```

```

lemma division_split:
  assumes p division_of (cbox a b)
  and k:  $k \in$  Basis
  shows  $\{l \cap \{x. x \cdot k \leq c\} \mid l. l \in p \wedge l \cap \{x. x \cdot k \leq c\} \neq \{\}\}$  division_of (cbox a
  b  $\cap \{x. x \cdot k \leq c\}$ )
  (is ?p1 division_of ?I1)
  and  $\{l \cap \{x. x \cdot k \geq c\} \mid l. l \in p \wedge l \cap \{x. x \cdot k \geq c\} \neq \{\}\}$  division_of (cbox a
  b  $\cap \{x. x \cdot k \geq c\}$ )
  (is ?p2 division_of ?I2)
proof (rule_tac[!] division_ofI)
  note  $p =$  division_ofD[OF assms(1)]
  show finite ?p1 finite ?p2
  using p(1) by auto
  show  $\bigcup ?p1 = ?I1 \bigcup ?p2 = ?I2$ 
  unfolding p(6)[symmetric] by auto
  {
  fix K
  assume  $K \in ?p1$ 
  then obtain l where  $l: K = l \cap \{x. x \cdot k \leq c\} \mid l \in p \mid l \cap \{x. x \cdot k \leq c\} \neq \{\}$ 
  by blast
  obtain u v where  $uv: l =$  cbox u v
  using assms(1) l(2) by blast
  show  $K \subseteq ?I1$ 
  using l p(2) uv by force
  show  $K \neq \{\}$ 
  by (simp add: l)
  have  $\exists a b. \text{cbox } u v \cap \{x. x \cdot k \leq c\} =$  cbox a b
  unfolding interval_split[OF k] by (auto intro: order.trans)
  then show  $\exists a b. K =$  cbox a b
  by (simp add: l(1) uv)
  fix K'
  assume  $K' \in ?p1$ 
  then obtain l' where  $l': K' = l' \cap \{x. x \cdot k \leq c\} \mid l' \in p \mid l' \cap \{x. x \cdot k \leq c\} \neq \{\}$ 
  by blast
  }

```

```

  assume  $K \neq K'$ 
  then show  $\text{interior } K \cap \text{interior } K' = \{\}$ 
    unfolding  $l\ l'$  using  $p(5)[OF\ l(2)\ l'(2)]$  by auto
  }
  {
    fix  $K$ 
    assume  $K \in ?p2$ 
    then obtain  $l$  where  $l: K = l \cap \{x. c \leq x \cdot k\}$   $l \in p\ l \cap \{x. c \leq x \cdot k\} \neq \{\}$ 
      by blast
    obtain  $u\ v$  where  $uv: l = \text{cbox } u\ v$ 
      using  $l(2)\ p(4)$  by blast
    show  $K \subseteq ?I2$ 
      using  $l\ p(2)\ uv$  by force
    show  $K \neq \{\}$ 
      by (simp add:  $l$ )
    have  $\exists a\ b. \text{cbox } u\ v \cap \{x. c \leq x \cdot k\} = \text{cbox } a\ b$ 
      unfolding  $\text{interval\_split}[OF\ k]$  by (auto intro:  $\text{order.trans}$ )
    then show  $\exists a\ b. K = \text{cbox } a\ b$ 
      by (simp add:  $l(1)\ uv$ )
    fix  $K'$ 
    assume  $K' \in ?p2$ 
    then obtain  $l'$  where  $l': K' = l' \cap \{x. c \leq x \cdot k\}$   $l' \in p\ l' \cap \{x. c \leq x \cdot k\} \neq \{\}$ 
      by blast
    assume  $K \neq K'$ 
    then show  $\text{interior } K \cap \text{interior } K' = \{\}$ 
      unfolding  $l\ l'$  using  $p(5)[OF\ l(2)\ l'(2)]$  by auto
  }
}
qed

```

7.13.6 Tagged (partial) divisions

definition $\text{tagged_partial_division_of}$ (**infixr** $\text{tagged}'_partial'_division'$ of 40)
 where $s\ \text{tagged_partial_division_of}\ i \iff$
 $\text{finite } s \wedge$
 $(\forall x\ K. (x, K) \in s \implies x \in K \wedge K \subseteq i \wedge (\exists a\ b. K = \text{cbox } a\ b)) \wedge$
 $(\forall x1\ K1\ x2\ K2. (x1, K1) \in s \wedge (x2, K2) \in s \wedge (x1, K1) \neq (x2, K2) \implies$
 $\text{interior } K1 \cap \text{interior } K2 = \{\})$

lemma $\text{tagged_partial_division_ofD}$:

assumes $s\ \text{tagged_partial_division_of}\ i$

shows $\text{finite } s$

and $\bigwedge x\ K. (x, K) \in s \implies x \in K$

and $\bigwedge x\ K. (x, K) \in s \implies K \subseteq i$

and $\bigwedge x\ K. (x, K) \in s \implies \exists a\ b. K = \text{cbox } a\ b$

and $\bigwedge x1\ K1\ x2\ K2. (x1, K1) \in s \implies$

$(x2, K2) \in s \implies (x1, K1) \neq (x2, K2) \implies \text{interior } K1 \cap \text{interior } K2 = \{\}$

using assms unfolding $\text{tagged_partial_division_of_def}$ by blast+

definition *tagged_division_of* (**infixr** *tagged'_division'_of* 40)

where $s \text{ tagged_division_of } i \longleftrightarrow s \text{ tagged_partial_division_of } i \wedge (\bigcup \{K. \exists x. (x, K) \in s\} = i)$

lemma *tagged_division_of_finite*: $s \text{ tagged_division_of } i \implies \text{finite } s$

unfolding *tagged_division_of_def* *tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_of*:

$s \text{ tagged_division_of } i \longleftrightarrow$

$\text{finite } s \wedge$

$(\forall x K. (x, K) \in s \longrightarrow x \in K \wedge K \subseteq i \wedge (\exists a b. K = \text{cbox } a \ b)) \wedge$

$(\forall x1 K1 x2 K2. (x1, K1) \in s \wedge (x2, K2) \in s \wedge (x1, K1) \neq (x2, K2) \longrightarrow$

$\text{interior } K1 \cap \text{interior } K2 = \{\}) \wedge$

$(\bigcup \{K. \exists x. (x, K) \in s\} = i)$

unfolding *tagged_division_of_def* *tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_ofI*:

assumes *finite s*

and $\bigwedge x K. (x, K) \in s \implies x \in K$

and $\bigwedge x K. (x, K) \in s \implies K \subseteq i$

and $\bigwedge x K. (x, K) \in s \implies \exists a b. K = \text{cbox } a \ b$

and $\bigwedge x1 K1 x2 K2. (x1, K1) \in s \implies (x2, K2) \in s \implies (x1, K1) \neq (x2, K2)$

\implies

$\text{interior } K1 \cap \text{interior } K2 = \{\}$

and $(\bigcup \{K. \exists x. (x, K) \in s\} = i)$

shows *s tagged_division_of i*

unfolding *tagged_division_of*

using *assms* **by** *fastforce*

lemma *tagged_division_ofD[dest]*:

assumes *s tagged_division_of i*

shows *finite s*

and $\bigwedge x K. (x, K) \in s \implies x \in K$

and $\bigwedge x K. (x, K) \in s \implies K \subseteq i$

and $\bigwedge x K. (x, K) \in s \implies \exists a b. K = \text{cbox } a \ b$

and $\bigwedge x1 K1 x2 K2. (x1, K1) \in s \implies (x2, K2) \in s \implies (x1, K1) \neq (x2, K2)$

\implies

$\text{interior } K1 \cap \text{interior } K2 = \{\}$

and $(\bigcup \{K. \exists x. (x, K) \in s\} = i)$

using *assms* **unfolding** *tagged_division_of* **by** *blast+*

lemma *division_of_tagged_division*:

assumes *s tagged_division_of i*

shows $(\text{snd } 's) \text{ division_of } i$

proof (*rule division_ofI*)

note *assm = tagged_division_ofD[OF assms]*

show $\bigcup (\text{snd } 's) = i \text{ finite } (\text{snd } 's)$

using *assm* **by** *auto*

fix *K*

```

  assume k:  $K \in \text{snd } 's$ 
  then show  $K \subseteq i \ K \neq \{\}$   $\exists a \ b. K = \text{cbox } a \ b$ 
    using asm by fastforce+
  fix  $K'$ 
  assume k':  $K' \in \text{snd } 's \ K \neq K'$ 
  then show  $\text{interior } K \cap \text{interior } K' = \{\}$ 
    by (metis (no_types, lifting) assms imageE k prod.collapse tagged_division_ofD(5))
qed

```

```

lemma partial_division_of_tagged_division:
  assumes s tagged_partial_division_of i
  shows  $(\text{snd } 's) \ \text{division\_of} \ \bigcup (\text{snd } 's)$ 
proof (rule division_ofI)
  note asm = tagged_partial_division_ofD[OF assms]
  show finite  $(\text{snd } 's) \ \bigcup (\text{snd } 's) = \bigcup (\text{snd } 's)$ 
    using asm by auto
  fix  $K$ 
  assume  $K: K \in \text{snd } 's$ 
  then show  $K \neq \{\} \ \exists a \ b. K = \text{cbox } a \ b \ K \subseteq \bigcup (\text{snd } 's)$ 
    using asm by fastforce+
  fix  $K'$ 
  assume  $K' \in \text{snd } 's \ K \neq K'$ 
  then show  $\text{interior } K \cap \text{interior } K' = \{\}$ 
    using asm(5)  $K$  by force
qed

```

```

lemma tagged_partial_division_subset:
  assumes s tagged_partial_division_of i and  $t \subseteq s$ 
  shows t tagged_partial_division_of i
  using assms finite_subset[OF assms(2)]
  unfolding tagged_partial_division_of_def
  by blast

```

```

lemma tag_in_interval:  $p \ \text{tagged\_division\_of} \ i \implies (x, k) \in p \implies x \in i$ 
  by auto

```

```

lemma tagged_division_of_empty:  $\{\} \ \text{tagged\_division\_of} \ \{\}$ 
  unfolding tagged_division_of by auto

```

```

lemma tagged_partial_division_of_trivial[simp]:  $p \ \text{tagged\_partial\_division\_of} \ \{\}$ 
 $\longleftrightarrow p = \{\}$ 
  unfolding tagged_partial_division_of_def by auto

```

```

lemma tagged_division_of_trivial[simp]:  $p \ \text{tagged\_division\_of} \ \{\} \longleftrightarrow p = \{\}$ 
  unfolding tagged_division_of by auto

```

```

lemma tagged_division_of_self:  $x \in \text{cbox } a \ b \implies \{(x, \text{cbox } a \ b)\} \ \text{tagged\_division\_of}$ 
 $(\text{cbox } a \ b)$ 
  by (rule tagged_division_ofI) auto

```

lemma *tagged_division_of_self_real*: $x \in \{a .. b :: \text{real}\} \implies \{(x, \{a .. b\})\}$ *tagged_division_of* $\{a .. b\}$
by (*metis* *box_real*(2) *tagged_division_of_self*)

lemma *tagged_division_Un*:
assumes $p1$ *tagged_division_of* $s1$
and $p2$ *tagged_division_of* $s2$
and *interior* $s1 \cap \text{interior } s2 = \{\}$
shows $(p1 \cup p2)$ *tagged_division_of* $(s1 \cup s2)$
proof (*rule* *tagged_division_ofI*)
note $p1 = \text{tagged_division_ofD}[OF \text{ assms}(1)]$
note $p2 = \text{tagged_division_ofD}[OF \text{ assms}(2)]$
show *finite* $(p1 \cup p2)$
using $p1(1)$ $p2(1)$ **by** *auto*
show $\bigcup \{k. \exists x. (x, k) \in p1 \cup p2\} = s1 \cup s2$
using $p1(6)$ $p2(6)$ **by** *blast*
fix $x K$
assume $xK: (x, K) \in p1 \cup p2$
show $x \in K \exists a b. K = \text{cbox } a b K \subseteq s1 \cup s2$
using xK $p1$ $p2$ **by** *auto*
fix $x' K'$
assume $xk': (x', K') \in p1 \cup p2$ $(x, K) \neq (x', K')$
have $*$: $\bigwedge a b. a \subseteq s1 \implies b \subseteq s2 \implies \text{interior } a \cap \text{interior } b = \{\}$
using *assms*(3) *interior_mono* **by** *blast*
with *assms* **show** *interior* $K \cap \text{interior } K' = \{\}$
by (*metis* *Un_iff_inf_commute* $p1(3)$ $p2(3)$ *tagged_division_ofD*(5) xK xk')
qed

lemma *tagged_division_Union*:
assumes *finite* I
and *tag*: $\bigwedge i. i \in I \implies \text{pfn } i$ *tagged_division_of* i
and *disj*: $\bigwedge i1 i2. \llbracket i1 \in I; i2 \in I; i1 \neq i2 \rrbracket \implies \text{interior}(i1) \cap \text{interior}(i2) = \{\}$
shows $\bigcup (\text{pfn } ' I)$ *tagged_division_of* $(\bigcup I)$
proof (*rule* *tagged_division_ofI*)
note *assm* = *tagged_division_ofD*[*OF* *tag*]
show *finite* $(\bigcup (\text{pfn } ' I))$
using *assms* **by** *auto*
have $\bigcup \{k. \exists x. (x, k) \in \bigcup (\text{pfn } ' I)\} = \bigcup ((\lambda i. \bigcup \{k. \exists x. (x, k) \in \text{pfn } i\}) ' I)$
by *blast*
also have $\dots = \bigcup I$
using *assm*(6) **by** *auto*
finally show $\bigcup \{k. \exists x. (x, k) \in \bigcup (\text{pfn } ' I)\} = \bigcup I$.
fix $x k$
assume $xk: (x, k) \in \bigcup (\text{pfn } ' I)$
then obtain i **where** $i: i \in I$ $(x, k) \in \text{pfn } i$
by *auto*
show $x \in k \exists a b. k = \text{cbox } a b k \subseteq \bigcup I$


```

  using assm(2-4)[OF i] using i(1) by auto
fix x' k'
assume xk':  $(x', k') \in \bigcup (pfn \text{ ' } I) (x, k) \neq (x', k')$ 
then obtain i' where i':  $i' \in I (x', k') \in pfn \text{ ' } i'$ 
  by auto
have *:  $\bigwedge a b. i \neq i' \implies a \subseteq i \implies b \subseteq i' \implies interior \ a \cap interior \ b = \{\}$ 
  using i(1) i'(1) disj interior_mono by blast
show  $interior \ k \cap interior \ k' = \{\}$ 
proof (cases i = i')
  case True then show ?thesis
    using assm(5) i' i xk'(2) by blast
  next
  case False then show ?thesis
    using * assm(3) i' i by auto
qed
qed

```

```

lemma tagged_partial_division_of_Union_self:
assumes p tagged_partial_division_of s
shows p tagged_division_of ( $\bigcup (snd \text{ ' } p)$ )
using tagged_partial_division_ofD[OF assms]
by (intro tagged_division_ofI) auto

```

```

lemma tagged_division_of_union_self:
assumes p tagged_division_of s
shows p tagged_division_of ( $\bigcup (snd \text{ ' } p)$ )
using tagged_division_ofD[OF assms]
by (intro tagged_division_ofI) auto

```

```

lemma tagged_division_Un_interval:
assumes p1 tagged_division_of ( $cbox \ a \ b \cap \{x. x \cdot k \leq (c::real)\}$ )
  and p2 tagged_division_of ( $cbox \ a \ b \cap \{x. x \cdot k \geq c\}$ )
  and k: k \in Basis
shows (p1  $\cup$  p2) tagged_division_of ( $cbox \ a \ b$ )
proof -
  have *:  $cbox \ a \ b = (cbox \ a \ b \cap \{x. x \cdot k \leq c\}) \cup (cbox \ a \ b \cap \{x. x \cdot k \geq c\})$ 
    by auto
  have  $interior \ (cbox \ a \ b \cap \{x. x \cdot k \leq c\}) \cap interior \ (cbox \ a \ b \cap \{x. c \leq x \cdot k\})$ 
    =  $\{\}$ 
    using k by (force simp: interval_split[OF k] box_def)
  with assms show ?thesis
    by (metis * tagged_division_Un)
qed

```

```

lemma tagged_division_Un_interval_real:
fixes a :: real
assumes p1 tagged_division_of ( $\{a .. b\} \cap \{x. x \cdot k \leq (c::real)\}$ )
  and p2 tagged_division_of ( $\{a .. b\} \cap \{x. x \cdot k \geq c\}$ )
  and k: k \in Basis

```

shows $(p1 \cup p2)$ *tagged_division_of* $\{a .. b\}$
using *assms* **by** $(metis\ box_real(2)\ tagged_division_Un_interval)$

lemma *tagged_division_split_left_inj*:
assumes $d: d$ *tagged_division_of* i
and *tags*: $(x1, K1) \in d$ $(x2, K2) \in d$
and $K1 \neq K2$
and *eq*: $K1 \cap \{x. x \cdot k \leq c\} = K2 \cap \{x. x \cdot k \leq c\}$
shows *interior* $(K1 \cap \{x. x \cdot k \leq c\}) = \{\}$
by $(smt\ (verit)\ Int_Un_eq(1)\ Un_Int_distrib\ interior_Int\ prod.inject\ sup_bot.right_neutral\ tagged_division_ofD(5)\ assms)$

lemma *tagged_division_split_right_inj*:
assumes $d: d$ *tagged_division_of* i
and *tags*: $(x1, K1) \in d$ $(x2, K2) \in d$
and $K1 \neq K2$
and *eq*: $K1 \cap \{x. x \cdot k \geq c\} = K2 \cap \{x. x \cdot k \geq c\}$
shows *interior* $(K1 \cap \{x. x \cdot k \geq c\}) = \{\}$
by $(smt\ (verit)\ Int_Un_eq(1)\ Un_Int_distrib\ interior_Int\ prod.inject\ sup_bot.right_neutral\ tagged_division_ofD(5)\ assms)$

lemma $(in\ comm_monoid_set)$ *over_tagged_division_lemma*:

assumes p *tagged_division_of* i
and $\bigwedge u\ v. box\ u\ v = \{\} \implies d\ (cbox\ u\ v) = \mathbf{1}$
shows $F\ (\lambda(_, k). d\ k)\ p = F\ d\ (snd\ ' p)$

proof –

have $*$: $(\lambda(_, k). d\ k) = d \circ snd$
by $(simp\ add: fun_eq_iff)$
note *assm* = *tagged_division_ofD[OF assms(1)]*
show *?thesis*

unfolding $*$

proof $(rule\ reindex_nontrivial[symmetric])$

show *finite* p

using *assm* **by** *auto*

fix $x\ y$

assume $x \in p\ y \in p\ x \neq y\ snd\ x = snd\ y$

obtain $a\ b$ **where** *ab*: $snd\ x = cbox\ a\ b$

using *assm(4)[of fst x snd x]* $\langle x \in p \rangle$ **by** *auto*

have $(fst\ x, snd\ y) \in p\ (fst\ x, snd\ y) \neq y$

using $\langle x \in p \rangle\ \langle x \neq y \rangle\ \langle snd\ x = snd\ y \rangle$ *[symmetric]* **by** *auto*

with $\langle x \in p \rangle\ \langle y \in p \rangle$ **have** *box* $a\ b = \{\}$

by $(metis\ \langle snd\ x = snd\ y \rangle\ ab\ assm(5)\ inf.idem\ interior_cbox\ prod.collapse)$

then **show** $d\ (snd\ x) = \mathbf{1}$

by $(simp\ add: ab\ assms(2))$

qed

qed

7.13.7 Functions closed on boxes: morphisms from boxes to monoids

This auxiliary structure is used to sum up over the elements of a division. Main theorem is *operative_division*. Instances for the monoid are *'a option*, *real*, and *bool*.

Using additivity of lifted function to encode definedness. **definition**

lift_option :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a option \Rightarrow 'b option \Rightarrow 'c option

where

lift_option f a' b' = Option.bind a' (λa . Option.bind b' (λb . Some (f a b)))

lemma *lift_option_simps*[simp]:

lift_option f (Some a) (Some b) = Some (f a b)

lift_option f None b' = None

lift_option f a' None = None

by (auto simp: *lift_option_def*)

lemma *comm_monoid_lift_option*:

assumes *comm_monoid* f z

shows *comm_monoid* (*lift_option* f) (Some z)

proof –

from *assms* **interpret** *comm_monoid* f z .

show *?thesis*

by *standard* (auto simp: *lift_option_def* *ac_simps* *split*: *bind_split*)

qed

lemma *comm_monoid_set_and*: *comm_monoid_set* HOL.conj True

by (*simp* *add*: *comm_monoid_set.intro* *conj*.*comm_monoid_axioms*)

Misc lemma *interval_real_split*:

$\{a .. b::real\} \cap \{x. x \leq c\} = \{a .. \min b c\}$

$\{a .. b\} \cap \{x. c \leq x\} = \{\max a c .. b\}$

by *auto*

lemma *bgauged_existence_lemma*: $(\forall x \in s. \exists d::real. 0 < d \wedge q d x) \longleftrightarrow (\forall x.$

$\exists d > 0. x \in s \longrightarrow q d x)$

by (*meson* *zero_less_one*)

Division points definition *division_points* (*k*::('a::euclidean_space) set) *d* =

$\{(j,x). j \in \text{Basis} \wedge (\text{interval_lowerbound } k) \cdot j < x \wedge x < (\text{interval_upperbound } k) \cdot j \wedge$

$(\exists i \in d. (\text{interval_lowerbound } i) \cdot j = x \vee (\text{interval_upperbound } i) \cdot j = x)\}$

lemma *division_points_finite*:

fixes *i* :: 'a::euclidean_space set

assumes *d* *division_of* *i*

shows *finite* (*division_points* *i* *d*)
proof –
note *assm* = *division_ofD*[*OF* *assms*]
let *?M* = $\lambda j. \{(j,x) \mid x. (\text{interval_lowerbound } i) \cdot j < x \wedge x < (\text{interval_upperbound } i) \cdot j \wedge$
 $(\exists i \in d. (\text{interval_lowerbound } i) \cdot j = x \vee (\text{interval_upperbound } i) \cdot j = x)\}$
have *: *division_points* *i* *d* = $\bigcup (\text{?M } \text{'Basis})$
unfolding *division_points_def* **by** *auto*
show *?thesis*
unfolding * **using** *assm* **by** *auto*
qed

lemma *division_points_subset*:
fixes *a* :: '*a*::*euclidean_space*
assumes *d* *division_of* (*cbox* *a* *b*)
and $\forall i \in \text{Basis}. a \cdot i < b \cdot i \quad a \cdot k < c < b \cdot k$
and *k*: *k* \in *Basis*
shows *division_points* (*cbox* *a* *b* $\cap \{x. x \cdot k \leq c\}$) $\{l \cap \{x. x \cdot k \leq c\} \mid l. l \in d \wedge$
 $l \cap \{x. x \cdot k \leq c\} \neq \{\}\} \subseteq$
division_points (*cbox* *a* *b*) *d* (**is** *?t1*)
and *division_points* (*cbox* *a* *b* $\cap \{x. x \cdot k \geq c\}$) $\{l \cap \{x. x \cdot k \geq c\} \mid l. l \in d \wedge$
 $\neg(l \cap \{x. x \cdot k \geq c\} = \{\})\} \subseteq$
division_points (*cbox* *a* *b*) *d* (**is** *?t2*)

proof –
note *assm* = *division_ofD*[*OF* *assms*(1)]
have *: $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$
 $\forall i \in \text{Basis}. a \cdot i \leq (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \min(b \cdot k) \ c \text{ else } b \cdot i) *_{\mathbb{R}} i) \cdot i$
 $\forall i \in \text{Basis}. (\sum i \in \text{Basis}. (\text{if } i = k \text{ then } \max(a \cdot k) \ c \text{ else } a \cdot i) *_{\mathbb{R}} i) \cdot i \leq b \cdot i$
 $\min(b \cdot k) \ c = c \ \max(a \cdot k) \ c = c$
using *assms* **using** *less_imp_le* **by** *auto*
have $\exists i \in d. \text{interval_lowerbound } i \cdot x = y \vee \text{interval_upperbound } i \cdot x = y$
if $a \cdot x < y < (\text{if } x = k \text{ then } c \text{ else } b \cdot x)$
 $\text{interval_lowerbound } i \cdot x = y \vee \text{interval_upperbound } i \cdot x = y$
 $i = l \cap \{x. x \cdot k \leq c\} \ l \in d \ l \cap \{x. x \cdot k \leq c\} \neq \{\}$
 $x \in \text{Basis}$ **for** *i* *l* *x* *y*

proof –
obtain *u* *v* **where** *l*: *l* = *cbox* *u* *v*
using $\langle l \in d \rangle$ *assms*(1) **by** *blast*
have $\forall i \in \text{Basis}. u \cdot i \leq v \cdot i$
using *l* **using** *that*(6) **unfolding** *box_ne_empty*[*symmetric*] **by** *auto*
then **show** *?thesis*
using *that* $\langle x \in \text{Basis} \rangle$ **unfolding** *l* *interval_split*[*OF* *k*]
by (*force* *split*: *if_split_asm*)

qed
moreover **have** $\bigwedge x \ y. \llbracket y < (\text{if } x = k \text{ then } c \text{ else } b \cdot x) \rrbracket \implies y < b \cdot x$
using $\langle c < b \cdot k \rangle$ **by** (*auto* *split*: *if_split_asm*)
ultimately **show** *?t1*
unfolding *division_points_def* *interval_split*[*OF* *k*, *of* *a* *b*]
unfolding *interval_bounds*[*OF* *(1)] *interval_bounds*[*OF* *(2)] *interval_bounds*[*OF*

```

*(3)]
  unfolding * by force
  have  $\bigwedge x y i l. (if\ x = k\ then\ c\ else\ a \cdot x) < y \implies a \cdot x < y$ 
    using  $\langle a \cdot k < c \rangle$  by (auto split: if_split_asm)
  moreover have  $\exists i \in d. interval\_lowerbound\ i \cdot x = y \vee interval\_upperbound\ i \cdot x = y$ 
  if (if  $x = k$  then  $c$  else  $a \cdot x$ )  $< y$   $y < b \cdot x$ 
    interval_lowerbound  $i \cdot x = y \vee interval\_upperbound\ i \cdot x = y$ 
     $i = l \cap \{x. c \leq x \cdot k\} \ l \in d \ l \cap \{x. c \leq x \cdot k\} \neq \{\}$ 
     $x \in Basis$  for  $x\ y\ i\ l$ 
  proof -
    obtain  $u\ v$  where  $l = cbox\ u\ v$ 
      using  $\langle l \in d \rangle$  assm(4) by blast
    have  $\forall i \in Basis. u \cdot i \leq v \cdot i$ 
      using  $l$  using that(6) unfolding box_ne_empty[symmetric] by auto
    then show  $\exists i \in d. interval\_lowerbound\ i \cdot x = y \vee interval\_upperbound\ i \cdot x = y$ 
      = y
      using that  $\langle x \in Basis \rangle$  unfolding  $l$  interval_split[OF  $k$ ]
      by (force split: if_split_asm)
  qed
  ultimately show ?t2
    unfolding division_points_def interval_split[OF  $k$ , of  $a\ b$ ]
    unfolding interval_bounds[OF *(1)] interval_bounds[OF *(2)] interval_bounds[OF *(3)]
  *(3)]
    unfolding *
    by force
  qed

lemma division_points_subset:
  fixes  $a :: 'a::euclidean\_space$ 
  assumes  $d: d\ division\_of\ (cbox\ a\ b)$ 
    and  $altb: \forall i \in Basis. a \cdot i < b \cdot i$   $a \cdot k < c$   $c < b \cdot k$ 
    and  $l \in d$ 
    and  $disj: interval\_lowerbound\ l \cdot k = c \vee interval\_upperbound\ l \cdot k = c$ 
    and  $k: k \in Basis$ 
  shows  $division\_points\ (cbox\ a\ b \cap \{x. x \cdot k \leq c\}) \{l \cap \{x. x \cdot k \leq c\} \mid l. l \in d \wedge l \cap \{x. x \cdot k \leq c\} \neq \{\}\} \subset$ 
     $division\_points\ (cbox\ a\ b)\ d$  (is ?D1  $\subset$  ?D)
    and  $division\_points\ (cbox\ a\ b \cap \{x. x \cdot k \geq c\}) \{l \cap \{x. x \cdot k \geq c\} \mid l. l \in d \wedge l \cap \{x. x \cdot k \geq c\} \neq \{\}\} \subset$ 
     $division\_points\ (cbox\ a\ b)\ d$  (is ?D2  $\subset$  ?D)
  proof -
    have  $ab: \forall i \in Basis. a \cdot i \leq b \cdot i$ 
      using  $altb$  by (auto intro!: less_imp_le)
    obtain  $u\ v$  where  $l = cbox\ u\ v$ 
      using  $d$   $\langle l \in d \rangle$  by blast
    have  $uv: \forall i \in Basis. u \cdot i \leq v \cdot i$   $\forall i \in Basis. a \cdot i \leq u \cdot i \wedge v \cdot i \leq b \cdot i$ 
      apply (metis assms(5) box_ne_empty(1) cbox_division_memE  $d\ l$ )
      by (metis assms(5) box_ne_empty(1) cbox_division_memE  $d\ l$  subset_box(1))
  
```

```

have *: interval_upperbound (cbox a b  $\cap$  {x. x · k ≤ interval_upperbound l · k})
· k = interval_upperbound l · k
      interval_upperbound (cbox a b  $\cap$  {x. x · k ≤ interval_lowerbound l · k}) ·
k = interval_lowerbound l · k
  unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
  using uv[rule_format, of k] ab k
  by auto
have  $\exists x. x \in ?D - ?D1$ 
  using assms(3-)
  unfolding division_points_def interval_bounds[OF ab]
  by (force simp add: *)
moreover have  $?D1 \subseteq ?D$ 
  by (auto simp add: assms division_points_subset)
ultimately show  $?D1 \subset ?D$ 
  by blast
have *: interval_lowerbound (cbox a b  $\cap$  {x. x · k ≥ interval_lowerbound l · k})
· k = interval_lowerbound l · k
      interval_lowerbound (cbox a b  $\cap$  {x. x · k ≥ interval_upperbound l · k}) · k =
interval_upperbound l · k
  unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
  using uv[rule_format, of k] ab k
  by auto
have  $\exists x. x \in ?D - ?D2$ 
  using assms(3-)
  unfolding division_points_def interval_bounds[OF ab]
  by (force simp add: *)
moreover have  $?D2 \subseteq ?D$ 
  by (auto simp add: assms division_points_subset)
ultimately show  $?D2 \subset ?D$ 
  by blast
qed

```

lemma *division_split_left_inj*:

```

fixes S :: 'a::euclidean_space set
assumes div:  $\mathcal{D}$  division_of S
  and eq:  $K1 \cap \{x::'a. x \cdot k \leq c\} = K2 \cap \{x. x \cdot k \leq c\}$ 
  and  $K1 \in \mathcal{D} \ K2 \in \mathcal{D} \ K1 \neq K2$ 
shows interior ( $K1 \cap \{x. x \cdot k \leq c\}$ ) = {}
proof -
  have interior  $K2 \cap \{x. x \cdot k \leq c\} = \text{interior } K2 \cap \text{interior } \{x. x \cdot k \leq c\}$ 
  by (metis (no_types) eq interior_Int)
moreover have  $\bigwedge A. \text{interior } A \cap \text{interior } K2 = \{ \} \vee A = K2 \vee A \notin \mathcal{D}$ 
  by (meson div  $\langle K2 \in \mathcal{D} \rangle$  division_of_def)
ultimately show ?thesis
  using  $\langle K1 \in \mathcal{D} \rangle \langle K1 \neq K2 \rangle$  by auto
qed

```

lemma *division_split_right_inj*:

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $div: \mathcal{D}\ division\_of\ S$ 
  and  $eq: K1 \cap \{x::'a.\ x \cdot k \geq c\} = K2 \cap \{x.\ x \cdot k \geq c\}$ 
  and  $K1 \in \mathcal{D}\ K2 \in \mathcal{D}\ K1 \neq K2$ 
shows  $interior\ (K1 \cap \{x.\ x \cdot k \geq c\}) = \{\}$ 
proof -
  have  $interior\ K2 \cap interior\ \{a.\ a \cdot k \geq c\} = interior\ K1 \cap interior\ \{a.\ a \cdot k \geq c\}$ 
  by  $(metis\ (no\_types)\ eq\ interior\_Int)$ 
  moreover have  $\bigwedge A.\ interior\ A \cap interior\ K2 = \{\} \vee A = K2 \vee A \notin \mathcal{D}$ 
  by  $(meson\ div\ \langle K2 \in \mathcal{D} \rangle\ division\_of\_def)$ 
  ultimately show  $?thesis$ 
  using  $\langle K1 \in \mathcal{D} \rangle\ \langle K1 \neq K2 \rangle$  by  $auto$ 
qed

```

```

lemma  $interval\_doublesplit:$ 
fixes  $a :: 'a::euclidean\_space$ 
assumes  $k \in Basis$ 
shows  $cbox\ a\ b \cap \{x.\ |x \cdot k - c| \leq (e::real)\} =$ 
   $cbox\ (\sum_{i \in Basis.\ (if\ i = k\ then\ max\ (a \cdot i)\ (c - e)\ else\ a \cdot i)\ *R\ i}$ 
   $(\sum_{i \in Basis.\ (if\ i = k\ then\ min\ (b \cdot i)\ (c + e)\ else\ b \cdot i)\ *R\ i}$ 
proof -
  have  $*$ :  $\bigwedge x\ c\ e::real.\ |x - c| \leq e \iff x \geq c - e \wedge x \leq c + e$ 
  by  $auto$ 
  have  $**$ :  $\bigwedge s\ P\ Q.\ s \cap \{x.\ P\ x \wedge Q\ x\} = (s \cap \{x.\ Q\ x\}) \cap \{x.\ P\ x\}$ 
  by  $blast$ 
  show  $?thesis$ 
  unfolding  $*$   $**$   $interval\_split[OF\ assms]$  by  $(rule\ refl)$ 
qed

```

```

lemma  $division\_doublesplit:$ 
fixes  $a :: 'a::euclidean\_space$ 
assumes  $p\ division\_of\ (cbox\ a\ b)$ 
  and  $k: k \in Basis$ 
shows  $(\lambda l.\ l \cap \{x.\ |x \cdot k - c| \leq e\})\ ' \{l \in p.\ l \cap \{x.\ |x \cdot k - c| \leq e\} \neq \{\}\}$ 
   $division\_of\ (cbox\ a\ b \cap \{x.\ |x \cdot k - c| \leq e\})$ 
proof -
  have  $**$ :  $\bigwedge p\ q\ p'\ q'.\ p\ division\_of\ q \implies p = p' \implies q = q' \implies p'\ division\_of\ q'$ 
  by  $auto$ 
  note  $division\_split(1)[OF\ assms,\ where\ c=c+e,\ unfolded\ interval\_split[OF\ k]]$ 
  note  $division\_split(2)[OF\ this,\ where\ c=c-e\ and\ k=k,\ OF\ k]$ 
  then show  $?thesis$ 
  apply  $(rule\ **)$ 
  subgoal
  apply  $(simp\ add:\ abs\_diff\_le\_iff\ field\_simps\ Collect\_conj\_eq\ setcompr\_eq\_image$ 
   $[symmetric]\ cong:\ image\_cong\_simp)$ 
  apply  $(rule\ equalityI)$ 
  apply  $blast$ 
  apply  $clarsimp$ 

```

2532

```

apply (rename_tac S)
apply (rule_tac x=S  $\cap$  {x. c + e  $\geq$  x  $\cdot$  k} in exI)
apply auto
done
by (simp add: interval_split k interval_doublesplit)
qed

```

```

Operative locale operative = comm_monoid_set +
  fixes g :: 'b::euclidean_space set  $\Rightarrow$  'a
  assumes box_empty_imp:  $\bigwedge a b$ . box a b = {}  $\implies$  g (cbox a b) = 1
  and Basis_imp:  $\bigwedge a b c k$ . k  $\in$  Basis  $\implies$  g (cbox a b) = g (cbox a b  $\cap$  {x. x  $\cdot$  k  $\leq$  c}) * g (cbox a b  $\cap$  {x. x  $\cdot$  k  $\geq$  c})
begin

```

```

lemma empty [simp]: g {} = 1
  by (metis box_empty_imp box_subset_cbox empty_as_interval subset_empty)

```

lemma division:

$F g d = g$ (cbox a b) **if** d division_of (cbox a b)

proof –

define C **where** [abs_def]: C = card (division_points (cbox a b) d)

then show ?thesis

using that **proof** (induction C arbitrary: a b d rule: less_induct)

case (less a b d)

show ?case

proof (cases box a b = {})

case True

{ **fix** k **assume** k \in d

then obtain a' b' **where** k: k = cbox a' b'

using division_ofD(4)[OF less.prem] **by** blast

then have cbox a' b' \subseteq cbox a b

using <k \in d> less.prem **by** blast

then have box a' b' \subseteq box a b

unfolding subset_box **by** auto

then have g k = 1

using box_empty_imp [of a' b'] k **by** (simp add: True)

}

with True **show** F g d = g (cbox a b)

by (auto intro!: neutral simp: box_empty_imp)

next

case False

then have ab: $\forall i \in$ Basis. a \cdot i < b \cdot i **and** ab': $\forall i \in$ Basis. a \cdot i \leq b \cdot i

by (auto simp: box_ne_empty)

show F g d = g (cbox a b)

proof (cases division_points (cbox a b) d = {})

case True

{ **fix** u v **and** j :: 'b

assume j: j \in Basis **and** as: cbox u v \in d

then have cbox u v \neq {}


```

    using less.premis by blast
  then have uv:  $\forall i \in \text{Basis}. u \cdot i \leq v \cdot i \wedge u \cdot j \leq v \cdot j$ 
    using j unfolding box_ne_empty by auto
  have *:  $\bigwedge p r Q. \neg j \in \text{Basis} \vee p \vee r \vee (\forall x \in d. Q x) \implies p \vee r \vee Q$  (cbox
u v)
    using as j by auto
  have (j, u·j)  $\notin$  division_points (cbox a b) d
    (j, v·j)  $\notin$  division_points (cbox a b) d using True by auto
  note this[unfolded de_Morgan_conj division_points_def mem_Collect_eq
split_conv interval_bounds[OF ab'] bex_simps]
  note *[OF this(1)] *[OF this(2)] note this[unfolded interval_bounds[OF
uv(1)]]
  moreover
  have a·j  $\leq$  u·j v·j  $\leq$  b·j
    by (meson as division_ofD(2) j less.premis subset_box(1) uv(1))+
  ultimately have u·j = a·j  $\wedge$  v·j = a·j  $\vee$  u·j = b·j  $\wedge$  v·j = b·j  $\vee$  u·j =
a·j  $\wedge$  v·j = b·j
    using uv(2) by force
}
  then have d':  $\forall i \in d. \exists u v. i = \text{cbox } u v \wedge$ 
    ( $\forall j \in \text{Basis}. u \cdot j = a \cdot j \wedge v \cdot j = a \cdot j \vee u \cdot j = b \cdot j \wedge v \cdot j = b \cdot j \vee u \cdot j = a \cdot j \wedge$ 
v·j = b·j)
  unfolding forall_in_division[OF less.premis] by blast
  have (1/2) *R (a+b)  $\in$  cbox a b
    unfolding mem_box using ab by (auto simp: inner_simps)
  note this[unfolded division_ofD(6)[OF <d division_of cbox a b>,symmetric]
Union_iff]
  then obtain i where i:  $i \in d$  (1 / 2) *R (a + b)  $\in$  i ..
  obtain u v where uv:  $i = \text{cbox } u v$ 
     $\forall j \in \text{Basis}. u \cdot j = a \cdot j \wedge v \cdot j = a \cdot j \vee$ 
 $u \cdot j = b \cdot j \wedge v \cdot j = b \cdot j \vee$ 
 $u \cdot j = a \cdot j \wedge v \cdot j = b \cdot j$ 
    using d' i(1) by auto
  have cbox a b  $\in$  d
  proof -
  have u = a v = b
    unfolding euclidean_eq_iff[where 'a='b]
  proof safe
  fix j :: 'b
  assume j:  $j \in \text{Basis}$ 
  note i(2)[unfolded uv mem_box]
  then show  $u \cdot j = a \cdot j$  and  $v \cdot j = b \cdot j$ 
    by (smt (verit) False box_midpoint j mem_box(1) uv(2))+
  qed
  then have i = cbox a b using uv by auto
  then show ?thesis using i by auto
  qed
  then have deq:  $d = \text{insert } (\text{cbox } a b) (d - \{\text{cbox } a b\})$ 
    by auto

```

```

have F g (d - {cbox a b}) = 1
proof (intro neutral ballI)
  fix x
  assume x: x ∈ d - {cbox a b}
  then have x ∈ d
    by auto note d'[rule_format, OF this]
  then obtain u v where uv: x = cbox u v
    by (metis Basis. u · j = a · j ∧ v · j = a · j ∨
      u · j = b · j ∧ v · j = b · j ∨
      u · j = a · j ∧ v · j = b · j)
  by blast
  have u ≠ a ∨ v ≠ b
    using x[unfolded uv] by auto
  then obtain j where u · j ≠ a · j ∨ v · j ≠ b · j and j: j ∈ Basis
    unfolding euclidean_eq_iff[where 'a='b] by auto
  then have u · j = v · j
    using uv(2)[rule_format, OF j] by auto
  then show g x = 1
    by (smt (verit) box_empty_imp box_eq_empty(1) j uv(1))
qed
then show F g d = g (cbox a b)
by (metis deq_division_of_finite insert_Diff_single insert_remove less.prem
right_neutral)
next
case False
then have ∃ x. x ∈ division_points (cbox a b) d
  by auto
then obtain k c
  where k ∈ Basis interval_lowerbound (cbox a b) · k < c
    c < interval_upperbound (cbox a b) · k
    ∃ i ∈ d. interval_lowerbound i · k = c ∨ interval_upperbound i · k = c
  unfolding division_points_def by auto
then obtain j where a · k < c < b · k
  and j ∈ d and j: interval_lowerbound j · k = c ∨ interval_upperbound
j · k = c
  by (metis division_of_trivial empty_iff interval_bounds' less.prem)
let ?lec = {x. x · k ≤ c} let ?gec = {x. x · k ≥ c}
define d1 where d1 = {l ∩ ?lec | l. l ∈ d ∧ l ∩ ?lec ≠ {}}
define d2 where d2 = {l ∩ ?gec | l. l ∈ d ∧ l ∩ ?gec ≠ {}}
define cb where cb = (∑ i ∈ Basis. (if i = k then c else b · i) *R i)
define ca where ca = (∑ i ∈ Basis. (if i = k then c else a · i) *R i)
have division_points (cbox a b ∩ ?lec) {l ∩ ?lec | l. l ∈ d ∧ l ∩ ?lec ≠ {}}
  ⊂ division_points (cbox a b) d
  by (rule division_points_subset[OF ⟨d division_of cbox a b⟩ ab ⟨a · k <
c⟩ ⟨c < b · k⟩ ⟨j ∈ d⟩ j ⟨k ∈ Basis⟩])
with division_points_finite[OF ⟨d division_of cbox a b⟩]
have card
  (division_points (cbox a b ∩ ?lec) {l ∩ ?lec | l. l ∈ d ∧ l ∩ ?lec ≠ {}})
  < card (division_points (cbox a b) d)

```

```

    by (rule psubset_card_mono)
    moreover have division_points (cbox a b  $\cap$   $\{x. c \leq x \cdot k\}$ )  $\{l \cap \{x. c \leq x \cdot k\} \mid l. l \in d \wedge l \cap \{x. c \leq x \cdot k\} \neq \{\}\}$ 
       $\subset$  division_points (cbox a b) d
    by (rule division_points_psubset[OF  $\langle d \text{ division\_of cbox a b} \rangle ab \langle a \cdot k < c \rangle \langle c < b \cdot k \rangle \langle j \in d \rangle j \langle k \in \text{Basis} \rangle$ ])
    with division_points_finite[OF  $\langle d \text{ division\_of cbox a b} \rangle$ ]
    have card (division_points (cbox a b  $\cap$  ?gec)  $\{l \cap ?gec \mid l. l \in d \wedge l \cap ?gec \neq \{\}\}$ )
       $<$  card (division_points (cbox a b) d)
    by (rule psubset_card_mono)
    ultimately have *:  $F g d1 = g (cbox a b \cap ?lec) F g d2 = g (cbox a b \cap ?gec)$ 
    unfolding interval_split[OF  $\langle k \in \text{Basis} \rangle$ ]
    apply (rule_tac[!] less.hyps)
    using division_split[OF  $\langle d \text{ division\_of cbox a b} \rangle$ , where  $k=k$  and  $c=c$ ]
     $\langle k \in \text{Basis} \rangle$ 
    by (simp_all add: interval_split d1_def d2_def division_points_finite[OF  $\langle d \text{ division\_of cbox a b} \rangle$ ])
    have f:k_le:  $g (l \cap ?lec) = \mathbf{1}$ 
      if  $l \in d \ y \in d \ l \cap ?lec = y \cap ?lec \ l \neq y$  for  $l \ y$ 
    proof -
      obtain  $u \ v$  where  $leq: l = cbox \ u \ v$ 
      using  $\langle l \in d \rangle$  less.prem by auto
      have interior (cbox  $u \ v \cap ?lec$ ) =  $\{\}$ 
      using that division_split_left_inj leq less.prem by blast
      then show ?thesis
      unfolding leq interval_split [OF  $\langle k \in \text{Basis} \rangle$ ]
      by (auto intro: box_empty_imp)
    qed
    have f:k_ge:  $g (l \cap \{x. x \cdot k \geq c\}) = \mathbf{1}$ 
      if  $l \in d \ y \in d \ l \cap ?gec = y \cap ?gec \ l \neq y$  for  $l \ y$ 
    proof -
      obtain  $u \ v$  where  $leq: l = cbox \ u \ v$ 
      using  $\langle l \in d \rangle$  less.prem by auto
      have interior (cbox  $u \ v \cap ?gec$ ) =  $\{\}$ 
      using that division_split_right_inj leq less.prem by blast
      then show ?thesis
      unfolding leq interval_split[OF  $\langle k \in \text{Basis} \rangle$ ]
      by (auto intro: box_empty_imp)
    qed
    have d1_alt:  $d1 = (\lambda l. l \cap ?lec) \cdot \{l \in d. l \cap ?lec \neq \{\}\}$ 
      using d1_def by auto
    have d2_alt:  $d2 = (\lambda l. l \cap ?gec) \cdot \{l \in d. l \cap ?gec \neq \{\}\}$ 
      using d2_def by auto
    have  $g (cbox a b) = F g d1 * F g d2$  (is  $\_ = ?prev$ )
      unfolding * using  $\langle k \in \text{Basis} \rangle$ 
      by (auto dest: Basis_imp)
    also have  $F g d1 = F (\lambda l. g (l \cap ?lec)) d$ 

```

```

    unfolding d1_alt using division_of_finite[OF less.prem] fvk_le
  by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
also have F g d2 = F (λl. g (l ∩ ?gec)) d
    unfolding d2_alt using division_of_finite[OF less.prem] fvk_ge
  by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
also have *: ∀x∈d. g x = g (x ∩ ?lec) * g (x ∩ ?gec)
    unfolding forall_in_division[OF ‹d division_of_cbox a b›]
    using ‹k ∈ Basis›
  by (auto dest: Basis_imp)
have F (λl. g (l ∩ ?lec)) d * F (λl. g (l ∩ ?gec)) d = F g d
    using * by (simp add: distrib)
finally show ?thesis by auto
qed
qed
qed
qed

```

proposition *tagged_division*:

assumes *d tagged_division_of* (cbox a b)

shows $F (\lambda(_, l). g l) d = g (cbox a b)$

by (*metis asms box_empty_imp division_division_of_tagged_division over_tagged_division_lemma*)

end

locale *operative_real* = *comm_monoid_set* +

fixes $g :: \text{real set} \Rightarrow 'a$

assumes *neutral*: $b \leq a \implies g \{a..b\} = \mathbf{1}$

assumes *coalesce_less*: $a < c \implies c < b \implies g \{a..c\} * g \{c..b\} = g \{a..b\}$

begin

sublocale *operative* **where** $g = g$

rewrites *box* = (*greaterThanLessThan* :: *real* \Rightarrow $_$)

and *cbox* = (*atLeastAtMost* :: *real* \Rightarrow $_$)

and $\bigwedge x::\text{real}. x \in \text{Basis} \longleftrightarrow x = 1$

proof –

show *operative* $f z g$

proof

show $g (cbox a b) = \mathbf{1}$ **if** $box a b = \{\}$ **for** $a b$

using *that* **by** (*simp* *add*: *neutral*)

show $g (cbox a b) = g (cbox a b \cap \{x. x \cdot k \leq c\}) * g (cbox a b \cap \{x. c \leq x \cdot k\})$

if $k \in \text{Basis}$ **for** $a b c k$

proof –

from *that* **have** [*simp*]: $k = 1$

by *simp*

from *neutral* [*of* 0 1] *neutral* [*of* a a **for** a] *coalesce_less*

have [*simp*]: $g \{\} = \mathbf{1} \wedge a. g \{a\} = \mathbf{1}$

$\bigwedge a b c. a < c \implies c < b \implies g \{a..c\} * g \{c..b\} = g \{a..b\}$

by *auto*

```

    have  $g \{a..b\} = g \{a..min\ b\ c\} * g \{max\ a\ c..b\}$ 
      by (auto simp: min_def max_def le_less)
    then show  $g (cbox\ a\ b) = g (cbox\ a\ b \cap \{x.\ x \cdot k \leq c\}) * g (cbox\ a\ b \cap \{x.\ c \leq x \cdot k\})$ 
      by (simp add: atMost_def [symmetric] atLeast_def [symmetric])
  qed
qed
show  $box = (greaterThanLessThan :: real \Rightarrow \_)$ 
  and  $cbox = (atLeastAtMost :: real \Rightarrow \_)$ 
  and  $\bigwedge x::real.\ x \in Basis \longleftrightarrow x = 1$ 
  by (simp_all add: fun_eq_iff)
qed

```

```

lemma coalesce_less_eq:
   $g \{a..c\} * g \{c..b\} = g \{a..b\}$  if  $a \leq c \leq b$ 
  by (metis coalesce_less commute left_neutral less_eq_real_def neutral that)

```

end

```

lemma operative_realI:
  operative_real f z g if operative f z g
proof -
  interpret operative f z g
  using that .
  show ?thesis
  proof
    show  $g \{a..b\} = z$  if  $b \leq a$  for  $a\ b$ 
      using that box_empty_imp by simp
    show  $f (g \{a..c\}) (g \{c..b\}) = g \{a..b\}$  if  $a < c < b$  for  $a\ b\ c$ 
      using that Basis_imp [of 1 a b c]
      by (simp_all add: atMost_def [symmetric] atLeast_def [symmetric] max_def
min_def)
  qed
qed

```

7.13.8 Special case of additivity we need for the FTC

```

lemma additive_tagged_division_1:
  fixes  $f :: real \Rightarrow 'a::real\_normed\_vector$ 
  assumes  $a \leq b$ 
  and  $p$  tagged_division_of  $\{a..b\}$ 
  shows  $sum (\lambda(x,K). f(Sup\ K) - f(Inf\ K))\ p = f\ b - f\ a$ 
proof -
  let ?f = ( $\lambda K::real\ set.\ if\ K = \{\}\ then\ 0\ else\ f(interval\_upperbound\ K) - f(interval\_lowerbound\ K)$ )
  interpret operative_real plus 0 ?f
  rewrites comm_monoid_set.F (+) 0 = sum
  by standard[1] (auto simp add: sum_def)
  have p_td:  $p$  tagged_division_of  $cbox\ a\ b$ 

```

```

    using assms(2) box_real(2) by presburger
    have **: cbox a b ≠ {}
    using assms(1) by auto
    then have  $f\ b - f\ a = (\sum_{(x, l) \in p. \text{if } l = \{\}} \text{then } 0 \text{ else } f(\text{interval\_upperbound } l) - f(\text{interval\_lowerbound } l))$ 
    using assms(2) tagged_division by force
    then show ?thesis
    using assms by (auto intro!: sum.cong)
qed

```

7.13.9 Fine-ness of a partition w.r.t. a gauge

definition *fine* (*infixr fine* 46)
 where $d \text{ fine } s \longleftrightarrow (\forall (x, k) \in s. k \subseteq d\ x)$

lemma *fineI*:
 assumes $\bigwedge x\ k. (x, k) \in s \implies k \subseteq d\ x$
 shows $d \text{ fine } s$
 using *assms* **unfolding** *fine_def* by *auto*

lemma *fineD[dest]*:
 assumes $d \text{ fine } s$
 shows $\bigwedge x\ k. (x, k) \in s \implies k \subseteq d\ x$
 using *assms* **unfolding** *fine_def* by *auto*

lemma *fine_Int*: $(\lambda x. d1\ x \cap d2\ x) \text{ fine } p \longleftrightarrow d1 \text{ fine } p \wedge d2 \text{ fine } p$
unfolding *fine_def* by *auto*

lemma *fine_Inter*:
 $(\lambda x. \bigcap \{f\ d\ x \mid d. d \in s\}) \text{ fine } p \longleftrightarrow (\forall d \in s. (f\ d) \text{ fine } p)$
unfolding *fine_def* by *blast*

lemma *fine_Un*: $d \text{ fine } p1 \implies d \text{ fine } p2 \implies d \text{ fine } (p1 \cup p2)$
unfolding *fine_def* by *blast*

lemma *fine_Union*: $(\bigwedge p. p \in ps \implies d \text{ fine } p) \implies d \text{ fine } (\bigcup ps)$
unfolding *fine_def* by *auto*

lemma *fine_subset*: $p \subseteq q \implies d \text{ fine } q \implies d \text{ fine } p$
unfolding *fine_def* by *blast*

7.13.10 Some basic combining lemmas

lemma *tagged_division_Union_exists*:
 assumes *finite* *I*
 and $\forall i \in I. \exists p. p \text{ tagged_division_of } i \wedge d \text{ fine } p$
 and $\forall i1 \in I. \forall i2 \in I. i1 \neq i2 \longrightarrow \text{interior } i1 \cap \text{interior } i2 = \{\}$
 and $\bigcup I = i$
 obtains *p* where *p* *tagged_division_of* *i* and $d \text{ fine } p$
proof –

```

obtain pfm where pfm:
   $\bigwedge x. x \in I \implies \text{pfm } x \text{ tagged\_division\_of } x$ 
   $\bigwedge x. x \in I \implies d \text{ fine } \text{pfm } x$ 
  using assms by metis
show thesis
proof
  show  $\bigcup (\text{pfm } 'I) \text{ tagged\_division\_of } i$ 
    using assms pfm(1) tagged_division_Union by force
  show  $d \text{ fine } \bigcup (\text{pfm } 'I)$ 
    by (metis (mono_tags, lifting) fine_Union imageE pfm(2))
qed
qed

```

7.13.11 The set we're concerned with must be closed

```

lemma division_of_closed:
  fixes i :: 'n::euclidean_space set
  shows  $s \text{ division\_of } i \implies \text{closed } i$ 
  by blast

```

7.13.12 General bisection principle for intervals; might be useful elsewhere

```

lemma interval_bisection_step:
  fixes type :: 'a::euclidean_space
  assumes emp:  $P \ \{\}$ 
  and Un:  $\bigwedge S \ T. \llbracket P \ S; P \ T; \text{interior}(S) \cap \text{interior}(T) = \{\} \rrbracket \implies P \ (S \cup T)$ 
  and non:  $\neg P \ (\text{cbox } a \ (b::'a))$ 
  obtains c d where  $\neg P \ (\text{cbox } c \ d)$ 
  and  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq c \cdot i \wedge c \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i \wedge 2 * (d \cdot i - c \cdot i) \leq b \cdot i - a \cdot i$ 
proof -
  have  $\text{cbox } a \ b \neq \{\}$ 
  using emp non by metis
  then have ab:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i$ 
  by (force simp: mem_box)
  have UN_cases:  $\llbracket \text{finite } \mathcal{F};$ 
     $\bigwedge S. S \in \mathcal{F} \implies P \ S;$ 
     $\bigwedge S. S \in \mathcal{F} \implies \exists a \ b. S = \text{cbox } a \ b;$ 
     $\bigwedge S \ T. S \in \mathcal{F} \implies T \in \mathcal{F} \implies S \neq T \implies \text{interior } S \cap \text{interior } T = \{\} \rrbracket \implies$ 
   $P \ (\bigcup \mathcal{F})$  for  $\mathcal{F}$ 
  proof (induct  $\mathcal{F}$  rule: finite_induct)
  case empty show ?case
  using emp by auto
  next
  case (insert x f)
  then show ?case
  unfolding Union_insert by (metis Int_interior_Union_intervals Un_insert_iff open_interior)

```

```

qed
let ?ab =  $\lambda i. (a \cdot i + b \cdot i) / 2$ 
let ?A = {cbox c d | c d::'a.  $\forall i \in \text{Basis}. (c \cdot i = a \cdot i) \wedge (d \cdot i = ?ab \ i) \vee$ 
  (c \cdot i = ?ab \ i)  $\wedge (d \cdot i = b \cdot i)$ }
have P ( $\bigcup ?A$ )
  if  $\bigwedge c \ d. \forall i \in \text{Basis}. a \cdot i \leq c \cdot i \wedge c \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i \wedge 2 * (d \cdot i - c \cdot i) \leq b \cdot i$ 
  -  $a \cdot i \implies P (cbox \ c \ d)$ 
proof (rule UN_cases)
  let ?B = ( $\lambda S. cbox (\sum i \in \text{Basis}. (if \ i \in S \ \text{then} \ a \cdot i \ \text{else} \ ?ab \ i) *_R \ i :: 'a)$ 
    ( $\sum i \in \text{Basis}. (if \ i \in S \ \text{then} \ ?ab \ i \ \text{else} \ b \cdot i) *_R \ i))$  ' {s. s  $\subseteq$  Basis}
  have ?A  $\subseteq$  ?B
proof
  fix x
  assume x  $\in$  ?A
  then obtain c d
    where x: x = cbox c d
       $\bigwedge i. i \in \text{Basis} \implies c \cdot i = a \cdot i \wedge d \cdot i = ?ab \ i \vee c \cdot i = ?ab \ i \wedge d \cdot i =$ 
      b \cdot i
    by blast
  have c = ( $\sum i \in \text{Basis}. (if \ c \cdot i = a \cdot i \ \text{then} \ a \cdot i \ \text{else} \ ?ab \ i) *_R \ i$ )
    d = ( $\sum i \in \text{Basis}. (if \ c \cdot i = a \cdot i \ \text{then} \ ?ab \ i \ \text{else} \ b \cdot i) *_R \ i$ )
    using x(2) ab by (fastforce simp add: euclidean_eq_iff[where 'a='a])+
  then show x  $\in$  ?B
    unfolding x by (rule_tac x={i. i  $\in$  Basis  $\wedge$  c \cdot i = a \cdot i} in image_eqI) auto
qed
then show finite ?A
  by (rule finite_subset) auto
next
fix S
assume S  $\in$  ?A
then obtain c d
  where s: S = cbox c d
     $\bigwedge i. i \in \text{Basis} \implies c \cdot i = a \cdot i \wedge d \cdot i = ?ab \ i \vee c \cdot i = ?ab \ i \wedge d \cdot i = b \cdot i$ 
    by blast
show P S
  unfolding s using ab s(2) by (fastforce intro!: that)
show  $\exists a \ b. S = cbox \ a \ b$ 
  unfolding s by auto
fix T
assume T  $\in$  ?A
then obtain e f where t:
  T = cbox e f
   $\bigwedge i. i \in \text{Basis} \implies e \cdot i = a \cdot i \wedge f \cdot i = ?ab \ i \vee e \cdot i = ?ab \ i \wedge f \cdot i = b \cdot i$ 
  by blast
assume S  $\neq$  T
then have  $\neg (c = e \wedge d = f)$ 
  unfolding s t by auto
then obtain i where c \cdot i  $\neq$  e \cdot i  $\vee$  d \cdot i  $\neq$  f \cdot i and i': i  $\in$  Basis
  unfolding euclidean_eq_iff[where 'a='a] by auto

```



```

then have  $i$ :  $c \cdot i \neq e \cdot i \wedge d \cdot i \neq f \cdot i$ 
  using  $s(2)$   $t(2)$  by fastforce+
have  $*$ :  $\bigwedge s t. (\bigwedge a. a \in s \implies a \in t \implies \text{False}) \implies s \cap t = \{\}$ 
  by auto
show  $\text{interior } S \cap \text{interior } T = \{\}$ 
  unfolding  $s$   $t$  interior_cbox
proof (rule  $*$ )
  fix  $x$ 
  assume  $x \in \text{cbox } c \ d \ x \in \text{cbox } e \ f$ 
  then have  $c \cdot i < f \cdot i \wedge e \cdot i < d \cdot i$ 
    unfolding mem_box using  $i'$  by force+
  with  $i$   $i'$  show False
    using  $s(2)$   $t(2)$  by fastforce
qed
qed
also have  $\bigcup ?A = \text{cbox } a \ b$ 
proof (rule set_eqI,rule)
  fix  $x$ 
  assume  $x \in \bigcup ?A$ 
  then obtain  $c \ d$  where  $x$ :
     $x \in \text{cbox } c \ d$ 
     $\bigwedge i. i \in \text{Basis} \implies c \cdot i = a \cdot i \wedge d \cdot i = ?ab \ i \vee c \cdot i = ?ab \ i \wedge d \cdot i = b \cdot i$ 
    by blast
  then show  $x \in \text{cbox } a \ b$ 
    unfolding mem_box by force
next
  fix  $x$ 
  assume  $x$ :  $x \in \text{cbox } a \ b$ 
  then have  $\forall i \in \text{Basis}. \exists c \ d. (c = a \cdot i \wedge d = ?ab \ i \vee c = ?ab \ i \wedge d = b \cdot i) \wedge$ 
 $c \leq x \cdot i \wedge x \cdot i \leq d$ 
    (is  $\forall i \in \text{Basis}. \exists c \ d. ?P \ i \ c \ d$ )
    unfolding mem_box by (metis linear)
  then obtain  $\alpha \ \beta$  where  $\forall i \in \text{Basis}. (\alpha \cdot i = a \cdot i \wedge \beta \cdot i = ?ab \ i \vee$ 
 $\alpha \cdot i = ?ab \ i \wedge \beta \cdot i = b \cdot i) \wedge \alpha \cdot i \leq x \cdot i \wedge x \cdot i \leq \beta \cdot i$ 
    by (auto simp: choice_Basis_iff)
  then show  $x \in \bigcup ?A$ 
    by (force simp add: mem_box)
qed
finally show thesis
  by (metis (no_types, lifting) assms(3) that)
qed

lemma interval_bisection:
fixes  $\text{type} :: 'a::\text{euclidean\_space}$ 
assumes  $P \ \{\}$ 
  and  $Un$ :  $\bigwedge S \ T. [\![P \ S; P \ T; \text{interior}(S) \cap \text{interior}(T) = \{\}]\!] \implies P \ (S \cup T)$ 
  and  $\neg P \ (\text{cbox } a \ (b::'a))$ 
obtains  $x$  where  $x \in \text{cbox } a \ b$ 
  and  $\forall e > 0. \exists c \ d. x \in \text{cbox } c \ d \wedge \text{cbox } c \ d \subseteq \text{ball } x \ e \wedge \text{cbox } c \ d \subseteq \text{cbox } a \ b \wedge$ 

```

$\neg P (cbox\ c\ d)$

proof –

have $\forall x. \exists y. \neg P (cbox\ (fst\ x)\ (snd\ x)) \longrightarrow (\neg P (cbox\ (fst\ y)\ (snd\ y)) \wedge$
 $(\forall i \in Basis. fst\ x \cdot i \leq fst\ y \cdot i \wedge fst\ y \cdot i \leq snd\ y \cdot i \wedge snd\ y \cdot i \leq snd\ x \cdot i \wedge$
 $2 * (snd\ y \cdot i - fst\ y \cdot i) \leq snd\ x \cdot i - fst\ x \cdot i))$ (**is** $\forall x. ?P\ x$)

proof

show $?P\ x$ **for** x

proof (*cases* $P (cbox\ (fst\ x)\ (snd\ x))$)

case *True*

then show $?thesis$ **by** *auto*

next

case *False*

obtain $c\ d$ **where** $\neg P (cbox\ c\ d)$

$\bigwedge i. i \in Basis \implies$

$fst\ x \cdot i \leq c \cdot i \wedge$

$c \cdot i \leq d \cdot i \wedge$

$d \cdot i \leq snd\ x \cdot i \wedge$

$2 * (d \cdot i - c \cdot i) \leq snd\ x \cdot i - fst\ x \cdot i$

by (*blast intro: interval_bisection_step*[of P , $OF\ assms(1-2)$ *False*])

then show $?thesis$

by (*rule_tac* $x=(c,d)$ **in** *exI*) *auto*

qed

qed

then obtain f **where** f :

$\forall x.$

$\neg P (cbox\ (fst\ x)\ (snd\ x)) \longrightarrow$

$\neg P (cbox\ (fst\ (f\ x))\ (snd\ (f\ x))) \wedge$

$(\forall i \in Basis.$

$fst\ x \cdot i \leq fst\ (f\ x) \cdot i \wedge$

$fst\ (f\ x) \cdot i \leq snd\ (f\ x) \cdot i \wedge$

$snd\ (f\ x) \cdot i \leq snd\ x \cdot i \wedge$

$2 * (snd\ (f\ x) \cdot i - fst\ (f\ x) \cdot i) \leq snd\ x \cdot i - fst\ x \cdot i)$ **by** *metis*

define $AB\ A\ B$ **where** $ab_def: AB\ n = (f \rightsquigarrow n)\ (a,b)\ A\ n = fst\ (AB\ n)\ B\ n =$
 $snd\ (AB\ n)$ **for** n

have [*simp*]: $A\ 0 = a\ B\ 0 = b$

and *ABRAW*: $\bigwedge n. \neg P (cbox\ (A\ (Suc\ n))\ (B\ (Suc\ n))) \wedge$

$(\forall i \in Basis. A(n) \cdot i \leq A(Suc\ n) \cdot i \wedge A(Suc\ n) \cdot i \leq B(Suc\ n) \cdot i \wedge B(Suc$
 $n) \cdot i \leq B(n) \cdot i \wedge$
 $2 * (B(Suc\ n) \cdot i - A(Suc\ n) \cdot i) \leq B(n) \cdot i - A(n) \cdot i)$ (**is** $\bigwedge n. ?P\ n$)

proof –

show $A\ 0 = a\ B\ 0 = b$

unfolding ab_def **by** *auto*

note $S = ab_def\ funpow.simps\ o_def\ id_apply$

show $?P\ n$ **for** n

proof (*induct* n)

case 0

then show $?case$

unfolding S **using** $\langle \neg P (cbox\ a\ b) \rangle f$ **by** *auto*

next

```

    case (Suc n)
    then show ?case
      unfolding S
      by (force intro!: f[rule_format])
  qed
qed
then have AB:  $A(n) \cdot i \leq A(\text{Suc } n) \cdot i$   $A(\text{Suc } n) \cdot i \leq B(\text{Suc } n) \cdot i$ 
           $B(\text{Suc } n) \cdot i \leq B(n) \cdot i$   $2 * (B(\text{Suc } n) \cdot i - A(\text{Suc } n) \cdot i) \leq B(n) \cdot i -$ 
 $A(n) \cdot i$ 
      if  $i \in \text{Basis}$  for  $i \ n$ 
    using that by blast+
  have notPAB:  $\neg P(\text{cbox } (A(\text{Suc } n)) (B(\text{Suc } n)))$  for  $n$ 
    using ABRAW by blast
  have interv:  $\exists n. \forall x \in \text{cbox } (A \ n) (B \ n). \forall y \in \text{cbox } (A \ n) (B \ n). \text{dist } x \ y < e$ 
    if  $e: 0 < e$  for  $e$ 
  proof -
    obtain  $n$  where  $n: (\sum i \in \text{Basis}. b \cdot i - a \cdot i) / e < 2^{\wedge} n$ 
      using real_arch_pow[of 2 (sum ( $\lambda i. b \cdot i - a \cdot i$ ) Basis) / e] by auto
    show ?thesis
      proof (rule exI [where  $x=n$ ], clarify)
        fix  $x \ y$ 
        assume  $xy: x \in \text{cbox } (A \ n) (B \ n) \ y \in \text{cbox } (A \ n) (B \ n)$ 
        have  $\text{dist } x \ y \leq \text{sum } (\lambda i. |(x - y) \cdot i|)$  Basis
          unfolding dist_norm by (rule norm_le_l1)
        also have  $\dots \leq \text{sum } (\lambda i. B \ n \cdot i - A \ n \cdot i)$  Basis
          proof (rule sum_mono)
            fix  $i :: 'a$ 
            assume  $i \in \text{Basis}$ 
            with  $xy$  show  $|(x - y) \cdot i| \leq B \ n \cdot i - A \ n \cdot i$ 
              by (smt (verit, best) inner_diff_left mem_box(2))
          qed
        qed
      also have  $\dots \leq \text{sum } (\lambda i. b \cdot i - a \cdot i)$  Basis /  $2^{\wedge} n$ 
        unfolding sum_divide_distrib
      proof (rule sum_mono)
        show  $B \ n \cdot i - A \ n \cdot i \leq (b \cdot i - a \cdot i) / 2^{\wedge} n$  if  $i: i \in \text{Basis}$  for  $i$ 
          proof (induct  $n$ )
            case 0
            then show ?case
              unfolding AB by auto
          next
            case (Suc n)
            have  $B(\text{Suc } n) \cdot i - A(\text{Suc } n) \cdot i \leq (B \ n \cdot i - A \ n \cdot i) / 2$ 
              using AB(3) that AB(4)[of  $i \ n$ ] using  $i$  by auto
            also have  $\dots \leq (b \cdot i - a \cdot i) / 2^{\wedge} \text{Suc } n$ 
              using Suc by (auto simp add: field_simps)
            finally show ?case .
          qed
        qed
      qed
    also have  $\dots < e$ 

```

```

    using n using e by (auto simp add: field_simps)
    finally show  $\text{dist } x \ y < e$  .
qed
qed
have  $AB\text{subset}: \text{cbox } (A \ n) \ (B \ n) \subseteq \text{cbox } (A \ m) \ (B \ m)$  if  $m \leq n$  for  $m \ n$ 
  using that
proof (induction rule: inc_induct)
  case (step i) show ?case
    by (smt (verit, ccfv_SIG) ABRAW in_mono mem_box(2) step.IH subsetI)
qed simp
have  $\bigwedge n. \text{cbox } (A \ n) \ (B \ n) \neq \{\}$ 
  by (meson AB dual_order.trans interval_not_empty)
then obtain  $x0$  where  $x0: \bigwedge n. x0 \in \text{cbox } (A \ n) \ (B \ n)$ 
  using decreasing_closed_nest [OF closed_cbox] ABsubset interv by blast
show thesis
proof (intro that strip)
  show  $x0 \in \text{cbox } a \ b$ 
    using  $\langle A \ 0 = a \rangle \langle B \ 0 = b \rangle x0$  by blast
next
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  from interv[OF this] obtain  $n$ 
    where  $n: \forall x \in \text{cbox } (A \ n) \ (B \ n). \forall y \in \text{cbox } (A \ n) \ (B \ n). \text{dist } x \ y < e$  ..
  have  $\neg P (\text{cbox } (A \ n) \ (B \ n))$ 
  proof (cases  $0 < n$ )
    case True then show ?thesis
      by (metis Suc_pred' notPAB)
    case False then show ?thesis
      using  $\langle A \ 0 = a \rangle \langle B \ 0 = b \rangle \langle \neg P (\text{cbox } a \ b) \rangle$  by blast
  qed
  moreover have  $\text{cbox } (A \ n) \ (B \ n) \subseteq \text{ball } x0 \ e$ 
    using n using  $x0$ [of n] by auto
  moreover have  $\text{cbox } (A \ n) \ (B \ n) \subseteq \text{cbox } a \ b$ 
    using ABsubset  $\langle A \ 0 = a \rangle \langle B \ 0 = b \rangle$  by blast
  ultimately show  $\exists c \ d. x0 \in \text{cbox } c \ d \wedge \text{cbox } c \ d \subseteq \text{ball } x0 \ e \wedge \text{cbox } c \ d \subseteq$ 
 $\text{cbox } a \ b \wedge \neg P (\text{cbox } c \ d)$ 
    by (meson  $x0$ )
  qed
qed

```

7.13.13 Cousin's lemma

```

lemma fine_division_exists:
  fixes  $a \ b :: 'a::\text{euclidean\_space}$ 
  assumes gauge  $g$ 
  obtains  $p$  where  $p$  tagged_division_of (cbox  $a \ b$ )  $g$  fine  $p$ 
proof (cases  $\exists p. p$  tagged_division_of (cbox  $a \ b$ )  $\wedge g$  fine  $p$ )
  case True

```

```

then show ?thesis
  using that by auto
next
case False
assume  $\neg (\exists p. p \text{ tagged\_division\_of } (cbox\ a\ b) \wedge g \text{ fine } p)$ 
obtain  $x$  where  $x$ :
   $x \in (cbox\ a\ b)$ 
   $\bigwedge e. 0 < e \implies$ 
   $\exists c\ d.$ 
   $x \in cbox\ c\ d \wedge$ 
   $cbox\ c\ d \subseteq ball\ x\ e \wedge$ 
   $cbox\ c\ d \subseteq (cbox\ a\ b) \wedge$ 
   $\neg (\exists p. p \text{ tagged\_division\_of } cbox\ c\ d \wedge g \text{ fine } p)$ 
proof (rule interval_bisection[of  $\lambda s. \exists p. p \text{ tagged\_division\_of } s \wedge \_ p, OF \_$ 
   $\_ False$ ])
  show  $\exists p. p \text{ tagged\_division\_of } \{x\} \wedge g \text{ fine } p$ 
  by (simp add: fine_def)
qed (meson tagged_division_Un_fine_Un)+
obtain  $e$  where  $e: e > 0 \wedge ball\ x\ e \subseteq g\ x$ 
  by (meson assms gauge_def openE)
then obtain  $c\ d$  where  $c\_d: x \in cbox\ c\ d$ 
   $cbox\ c\ d \subseteq ball\ x\ e$ 
   $cbox\ c\ d \subseteq cbox\ a\ b$ 
   $\neg (\exists p. p \text{ tagged\_division\_of } cbox\ c\ d \wedge g \text{ fine } p)$ 
  by (meson  $x(2)$ )
have  $g \text{ fine } \{(x, cbox\ c\ d)\}$ 
  unfolding fine_def using  $e$  using  $c\_d(2)$  by auto
then show ?thesis
  using tagged_division_of_self[OF  $c\_d(1)$ ] using  $c\_d$  by simp
qed

lemma fine_division_exists_real:
  fixes  $a\ b :: real$ 
  assumes gauge  $g$ 
  obtains  $p$  where  $p \text{ tagged\_division\_of } \{a .. b\} \wedge g \text{ fine } p$ 
  by (metis assms box_real(2) fine_division_exists)

```

7.13.14 A technical lemma about "refinement" of division

```

lemma tagged_division_finer:
  fixes  $p :: ('a::euclidean\_space \times ('a::euclidean\_space\ set))\ set$ 
  assumes  $ptag: p \text{ tagged\_division\_of } (cbox\ a\ b)$ 
  and gauge  $d$ 
  obtains  $q$  where  $q \text{ tagged\_division\_of } (cbox\ a\ b)$ 
  and  $d \text{ fine } q$ 
  and  $\forall (x, K) \in p. K \subseteq d(x) \implies (x, K) \in q$ 
proof -
  have  $p: p \text{ finite } p \wedge p \text{ tagged\_partial\_division\_of } (cbox\ a\ b)$ 
  using  $ptag \text{ tagged\_division\_of\_def}$  by blast+

```

```

have ( $\exists q. q \text{ tagged\_division\_of } (\bigcup \{k. \exists x. (x,k) \in p\}) \wedge d \text{ fine } q \wedge (\forall (x,k) \in$ 
 $p. k \subseteq d(x) \longrightarrow (x,k) \in q)$ )
  if  $\text{finite } p$  using  $p \text{ tagged\_partial\_division\_of } (\text{cbox } a \ b) \text{ gauge } d$  for  $p$ 
  using  $\text{that}$ 
proof ( $\text{induct } p$ )
  case  $\text{empty}$ 
  show  $?case$ 
    by ( $\text{force simp add: fine\_def}$ )
next
  case ( $\text{insert } xk \ p$ )
  obtain  $x \ k$  where  $xk: xk = (x, k)$ 
    using  $\text{surj\_pair[of } xk]$  by  $\text{metis}$ 
  obtain  $q1$  where  $q1: q1 \text{ tagged\_division\_of } \bigcup \{k. \exists x. (x, k) \in p\}$ 
    and  $d \text{ fine } q1$ 
    and  $q1I: \bigwedge x \ k. \llbracket (x, k) \in p; k \subseteq d \ x \rrbracket \implies (x, k) \in q1$ 
    using  $\text{insert}$ 
  by ( $\text{metis (mono\_tags, lifting) case\_prodD subset\_insertI tagged\_partial\_division\_subset}$ )
  have  $*$ :  $\bigcup \{l. \exists y. (y,l) \in \text{insert } xk \ p\} = k \cup \bigcup \{l. \exists y. (y,l) \in p\}$ 
    unfolding  $xk$  by  $\text{auto}$ 
  note  $p = \text{tagged\_partial\_division\_ofD[OF insert(4)]}$ 
  obtain  $u \ v$  where  $uv: k = \text{cbox } u \ v$ 
    using  $p(4) \ xk$  by  $\text{blast}$ 
  have  $\{K. \exists x. (x, K) \in p\} \subseteq \text{snd } ' p$ 
    by  $\text{force}$ 
  then have  $\text{finite } \{K. \exists x. (x, K) \in p\}$ 
    using  $\text{finite\_surj insert.hyps(1)}$  by  $\text{blast}$ 
  then have  $\text{int: interior } (\text{cbox } u \ v) \cap \text{interior } (\bigcup \{k. \exists x. (x, k) \in p\}) = \{\}$ 
proof ( $\text{rule Int\_interior\_Union\_intervals}$ )
  show  $\text{open } (\text{interior } (\text{cbox } u \ v))$ 
    by  $\text{auto}$ 
  show  $\bigwedge T. T \in \{K. \exists x. (x, K) \in p\} \implies \exists a \ b. T = \text{cbox } a \ b$ 
    using  $p(4)$  by  $\text{auto}$ 
  show  $\bigwedge T. T \in \{K. \exists x. (x, K) \in p\} \implies \text{interior } (\text{cbox } u \ v) \cap \text{interior } T =$ 
 $\{\}$ 
    using  $\text{insert.hyps(2) } p(5) \ uv \ xk$  by  $\text{blast}$ 
qed
show  $?case$ 
proof ( $\text{cases } \text{cbox } u \ v \subseteq d \ x$ )
  case  $\text{True}$ 
  have  $\{(x, \text{cbox } u \ v)\} \text{ tagged\_division\_of } \text{cbox } u \ v$ 
    by ( $\text{simp add: } p(2) \ uv \ xk \text{ tagged\_division\_of\_self}$ )
  then have  $\{(x, \text{cbox } u \ v)\} \cup q1 \text{ tagged\_division\_of } \bigcup \{K. \exists x. (x, K) \in$ 
 $\text{insert } xk \ p\}$ 
    by ( $\text{smt (verit, best) } * \ \text{int } q1 \text{ tagged\_division\_Un } uv$ )
  moreover have  $d \text{ fine } (\{(x, \text{cbox } u \ v)\} \cup q1)$ 
    using  $\text{True } \langle d \text{ fine } q1 \rangle \text{ fine\_def}$  by  $\text{fastforce}$ 
  ultimately show  $?thesis$ 
    by ( $\text{metis (no\_types, lifting) case\_prodI2 insert\_iff insert\_is\_Un } q1I \ uv$ 
 $xk$ )

```

```

next
  case False
  obtain q2 where q2: q2 tagged_division_of cbox u v d fine q2
  using fine_division_exists[OF assms(2)] by blast
  show ?thesis
  proof (intro exI conjI)
    show q2  $\cup$  q1 tagged_division_of  $\cup$  {k.  $\exists x. (x, k) \in \text{insert } xk p$ }
    by (smt (verit, best) * int q1 q2(1) tagged_division_Un uv)
    show d fine q2  $\cup$  q1
    by (simp add: <d fine q1> fine_Un q2(2))
  qed (use False uv xk q1I in auto)
qed
qed
with p obtain q where q: q tagged_division_of  $\cup$  {k.  $\exists x. (x, k) \in p$ } d fine q
 $\forall (x, k) \in p. k \subseteq d x \longrightarrow (x, k) \in q$ 
  by (meson <gauge d>)
  with ptag that show ?thesis by auto
qed

```

Covering lemma

Some technical lemmas used in the approximation results that follow. Proof of the covering lemma is an obvious multidimensional generalization of Lemma 3, p65 of Swartz's "Introduction to Gauge Integrals".

proposition *covering_lemma*:

assumes $S \subseteq \text{cbox } a \ b \ \text{box } a \ b \neq \{\}$ *gauge* g

obtains \mathcal{D} **where**

countable $\mathcal{D} \cup \mathcal{D} \subseteq \text{cbox } a \ b$

$\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$

pairwise $(\lambda A \ B. \text{interior } A \cap \text{interior } B = \{\}) \ \mathcal{D}$

$\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq g \ x$

$\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^{\wedge} n$
 $S \subseteq \bigcup \mathcal{D}$

proof –

have *aibi*: $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i$ **and** *normab*: $0 < \text{norm}(b - a)$

using $\langle \text{box } a \ b \neq \{\} \rangle$ *box_eq_empty* *box_idem* **by** *fastforce*+

let $?K0 = \lambda(n, f::'a \Rightarrow \text{nat}).$

$\text{cbox } (\sum i \in \text{Basis}. (a \cdot i + (f \ i / 2^{\wedge} n) * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$

$(\sum i \in \text{Basis}. (a \cdot i + ((f \ i + 1) / 2^{\wedge} n) * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$

let $?D0 = ?K0 \ ' (SIGMA \ n:UNIV. Pi_E \ \text{Basis} \ (\lambda i::'a. \text{lessThan } (2^{\wedge} n)))$

obtain $\mathcal{D}0$ **where** *count*: *countable* $\mathcal{D}0$

and *sub*: $\bigcup \mathcal{D}0 \subseteq \text{cbox } a \ b$

and *int*: $\bigwedge K. K \in \mathcal{D}0 \implies (\text{interior } K \neq \{\}) \wedge (\exists c \ d. K = \text{cbox } c \ d)$

and *intdj*: $\bigwedge A \ B. \llbracket A \in \mathcal{D}0; B \in \mathcal{D}0 \rrbracket \implies A \subseteq B \vee B \subseteq A \vee \text{interior}$

$A \cap \text{interior } B = \{\}$

and *SK*: $\bigwedge x. x \in S \implies \exists K \in \mathcal{D}0. x \in K \wedge K \subseteq g \ x$

and *cbox*: $\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D}0 \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^{\wedge} n$

and *fin*: $\bigwedge K. K \in \mathcal{D}0 \implies \text{finite } \{L \in \mathcal{D}0. K \subseteq L\}$

```

proof
  show countable ?D0
    by (simp add: countable_PiE)
  next
    show  $\bigcup ?D0 \subseteq \text{cbox } a \ b$ 
      apply (simp add: UN_subset_iff)
      apply (intro conjI allI ballI subset_box_imp)
      apply (simp add: field_simps aibi mult_right_mono)
      apply (force simp: aibi scaling_mono nat_less_real_le dest: PiE_mem intro: mult_right_mono)
      done
    next
      show  $\bigwedge K. K \in ?D0 \implies \text{interior } K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
        using  $\langle \text{box } a \ b \neq \{\} \rangle$ 
        by (clarsimp simp: box_eq_empty) (fastforce simp add: field_split_simps dest: PiE_mem)
      next
        have realff:  $(\text{real } w) * 2^m < (\text{real } v) * 2^n \longleftrightarrow w * 2^m < v * 2^n$  for  $m \ n$ 
           $v \ w$ 
          using of_nat_less_iff less_imp_of_nat_less by fastforce
          have *:  $\forall v \ w. ?K0(m,v) \subseteq ?K0(n,w) \vee ?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
            for  $m \ n$  — The symmetry argument requires a single HOL formula
            proof (rule linorder_wlog [where  $a=m$  and  $b=n$ ], intro allI impI)
              fix  $v \ w \ m$  and  $n::\text{nat}$ 
                assume  $m \leq n$  — WLOG we can assume  $m \leq n$ , when the first disjunct becomes impossible
                have  $?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
                  apply (rule ccontr)
                  apply (simp add: subset_box_disjoint_interval)
                  apply (clarsimp simp add: aibi mult_le_cancel_right divide_le_cancel not_less not_le)
                  apply (drule_tac x=i in bspec, assumption)
                  using  $\langle m \leq n \rangle$  realff [of _ _ 1+_] realff [of 1+_ 1+_]
                  apply (auto simp: divide_simps add.commute not_le nat_le_iff_add realff)
                  apply (metis (no_types, opaque_lifting) power_add mult_Suc mult_less_cancel2 not_less_eq mult.assoc)
                  done
                then show  $?K0(m,v) \subseteq ?K0(n,w) \vee ?K0(n,w) \subseteq ?K0(m,v) \vee \text{interior}(?K0(m,v)) \cap \text{interior}(?K0(n,w)) = \{\}$ 
                  by meson
              qed auto
            show  $\bigwedge A \ B. [A \in ?D0; B \in ?D0] \implies A \subseteq B \vee B \subseteq A \vee \text{interior } A \cap \text{interior } B = \{\}$ 
              using * by fastforce
          next
            show  $\exists K \in ?D0. x \in K \wedge K \subseteq g \ x$  if  $x \in S$  for  $x$ 
            proof (simp only: bex_simps split_paired_Bex_Sigma)

```



```

show  $\exists n. \exists f \in \text{Basis} \rightarrow_E \{.. < 2^{\wedge} n\}. x \in ?K0(n,f) \wedge ?K0(n,f) \subseteq g x$ 
proof -
  obtain  $e$  where  $0 < e$ 
    and  $e: \bigwedge y. (\bigwedge i. i \in \text{Basis} \implies |x \cdot i - y \cdot i| \leq e) \implies y \in g x$ 
  proof -
    have  $x \in g x \text{ open } (g x)$ 
      using  $\langle \text{gauge } g \rangle$  by (auto simp: gauge_def)
    then obtain  $\varepsilon$  where  $0 < \varepsilon$  and  $\varepsilon: \text{ball } x \varepsilon \subseteq g x$ 
      using openE by blast
    have  $\text{norm } (x - y) < \varepsilon$ 
      if  $(\bigwedge i. i \in \text{Basis} \implies |x \cdot i - y \cdot i| \leq \varepsilon / (2 * \text{real DIM}('a)))$  for  $y$ 
    proof -
      have  $\text{norm } (x - y) \leq (\sum_{i \in \text{Basis}. |x \cdot i - y \cdot i|)$ 
        by (metis (no_types, lifting) inner_diff_left norm_le_l1 sum.cong)
      also have  $\dots \leq \text{DIM}('a) * (\varepsilon / (2 * \text{real DIM}('a)))$ 
        by (meson sum_bounded_above that)
      also have  $\dots = \varepsilon / 2$ 
        by (simp add: field_split_simps)
      finally show ?thesis
        using  $\langle 0 < \varepsilon \rangle$  by linarith
    qed
  then show ?thesis
    by (rule_tac  $e = \varepsilon / 2 / \text{DIM}('a)$  in that) (simp_all add:  $\langle 0 < \varepsilon \rangle$ )
dist_norm subsetD [OF  $\varepsilon$ ]
qed
have  $xab: x \in \text{cbox } a b$ 
  using  $\langle x \in S \rangle \langle S \subseteq \text{cbox } a b \rangle$  by blast
obtain  $n$  where  $n: \text{norm } (b - a) / 2^{\wedge} n < e$ 
  using real_arch_pow_inv [of  $e / \text{norm}(b - a) 1/2$ ] normab  $\langle 0 < e \rangle$ 
  by (auto simp: field_split_simps)
then have  $\text{norm } (b - a) < e * 2^{\wedge} n$ 
  by (auto simp: field_split_simps)
then have  $bai: b \cdot i - a \cdot i < e * 2^{\wedge} n$  if  $i \in \text{Basis}$  for  $i$ 
  by (smt (verit, del_insts) Basis_le_norm diff_add_cancel inner_simps(1)
that)
  have  $D: (a + n \leq x \wedge x \leq a + m) \implies (a + n \leq y \wedge y \leq a + m) \implies$ 
 $\text{abs}(x - y) \leq m - n$ 
    for  $a m n x$  and  $y::\text{real}$ 
  by auto
have  $\forall i \in \text{Basis}. \exists k < 2^{\wedge} n. (a \cdot i + \text{real } k * (b \cdot i - a \cdot i) / 2^{\wedge} n \leq x \cdot i \wedge$ 
 $x \cdot i \leq a \cdot i + (\text{real } k + 1) * (b \cdot i - a \cdot i) / 2^{\wedge} n)$ 
proof
  fix  $i::'a$  assume  $i \in \text{Basis}$ 
  consider  $x \cdot i = b \cdot i \mid x \cdot i < b \cdot i$ 
  using  $\langle i \in \text{Basis} \rangle \text{mem\_box}(2) xab$  by force
  then show  $\exists k < 2^{\wedge} n. (a \cdot i + \text{real } k * (b \cdot i - a \cdot i) / 2^{\wedge} n \leq x \cdot i \wedge$ 
 $x \cdot i \leq a \cdot i + (\text{real } k + 1) * (b \cdot i - a \cdot i) / 2^{\wedge} n)$ 
proof cases
  case 1 then show ?thesis

```

```

      by (rule_tac x = 2n - 1 in exI) (auto simp: algebra_simps
field_split_simps of_nat_diff ⟨i ∈ Basis⟩ aibi)
    next
      case 2
      then have abi_less: a · i < b · i
        using ⟨i ∈ Basis⟩ xab by (auto simp: mem_box)
      let ?k = nat [2n * (x · i - a · i) / (b · i - a · i)]
      show ?thesis
      proof (intro exI conjI)
        show ?k < 2n
          using aibi xab ⟨i ∈ Basis⟩
          by (force simp: nat_less_iff floor_less_iff field_split_simps 2
mem_box)
        next
          have a · i + real ?k * (b · i - a · i) / 2n ≤
            a · i + (2n * (x · i - a · i) / (b · i - a · i)) * (b · i - a · i) /
2n
          using aibi [OF ⟨i ∈ Basis⟩] xab 2
          apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
          apply (auto simp: ⟨i ∈ Basis⟩ mem_box field_split_simps)
          done
          also have ... = x · i
            using abi_less by (simp add: field_split_simps)
          finally show a · i + real ?k * (b · i - a · i) / 2n ≤ x · i .
        next
          have x · i ≤ a · i + (2n * (x · i - a · i) / (b · i - a · i)) * (b · i
- a · i) / 2n
            using abi_less by (simp add: field_split_simps)
          also have ... ≤ a · i + (real ?k + 1) * (b · i - a · i) / 2n
            using aibi [OF ⟨i ∈ Basis⟩] xab
            apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
            apply (auto simp: ⟨i ∈ Basis⟩ mem_box divide_simps)
            done
          finally show x · i ≤ a · i + (real ?k + 1) * (b · i - a · i) / 2n .
      qed
    qed
  qed
  then have ∃f∈Basis →E {..<2n}. x ∈ ?K0(n,f)
  apply (simp add: mem_box Bex_def)
  apply (clarify dest!: bchoice)
  apply (rule_tac x=restrict f Basis in exI, simp)
  done
  moreover have ∧f. x ∈ ?K0(n,f) ⇒ ?K0(n,f) ⊆ g x
  apply (clarsimp simp add: mem_box)
  apply (rule e)
  apply (drule bspec D, assumption)+
  apply (erule order_trans)

```

```

    apply (simp add: divide_simps)
    using bai apply (force simp add: algebra_simps)
    done
  ultimately show ?thesis by auto
qed
next
show  $\bigwedge u v. \text{cbox } u v \in ?D0 \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$ 
  by (force simp: eq_cbox box_eq_empty_field_simps dest!: aibi)
next
obtain  $j::'a$  where  $j \in \text{Basis}$ 
  using nonempty_Basis by blast
have finite  $\{L \in ?D0. ?K0(n,f) \subseteq L\}$  if  $f \in \text{Basis} \rightarrow_E \{.. < 2^n\}$  for  $n f$ 
proof (rule finite_subset)
  let  $?B = (\lambda(n, f)::'a \Rightarrow \text{nat}). \text{cbox } (\sum_{i \in \text{Basis}} (a \cdot i + (f i) / 2^n * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$ 
   $(\sum_{i \in \text{Basis}} (a \cdot i + ((f i) + 1) / 2^n * (b \cdot i - a \cdot i)) *_{\mathbb{R}} i)$ 
  ' (SIGMA  $m$ :atMost  $n. \text{Pi}_E \text{Basis } (\lambda i::'a. \text{lessThan } (2^m))$ )
  have  $?K0(m,g) \in ?B$  if  $g \in \text{Basis} \rightarrow_E \{.. < 2^m\}$   $?K0(n,f) \subseteq ?K0(m,g)$ 
for  $m g$ 
proof -
  have  $dd: w / m \leq v / n \wedge (v+1) / n \leq (w+1) / m$ 
     $\implies \text{inverse } n \leq \text{inverse } m$  for  $w m v n::\text{real}$ 
  by (auto simp: field_split_simps)
  have  $\text{bjaj}: b \cdot j - a \cdot j > 0$ 
    using  $\langle j \in \text{Basis} \rangle \langle \text{box } a b \neq \{\} \rangle \text{box\_eq\_empty}(1)$  by fastforce
  have  $((g j) / 2^m) * (b \cdot j - a \cdot j) \leq ((f j) / 2^n) * (b \cdot j - a \cdot j) \wedge$ 
     $((f j) + 1) / 2^n * (b \cdot j - a \cdot j) \leq (((g j) + 1) / 2^m) * (b \cdot j - a \cdot j)$ 
  using that  $\langle j \in \text{Basis} \rangle$  by (simp add: subset_box field_split_simps aibi)
  then have  $((g j) / 2^m) \leq ((f j) / 2^n) \wedge$ 
     $((\text{real}(f j) + 1) / 2^n) \leq ((\text{real}(g j) + 1) / 2^m)$ 
  by (metis bjaj mult.commute of_nat_1 of_nat_add mult_le_cancel_left_pos)
  then have  $\text{inverse } (2^n) \leq (\text{inverse } (2^m) :: \text{real})$ 
  by (rule dd)
  then have  $m \leq n$ 
  by auto
  show ?thesis
  by (rule imageI) (simp add:  $\langle m \leq n \rangle$  that)
qed
then show  $\{L \in ?D0. ?K0(n,f) \subseteq L\} \subseteq ?B$ 
  by auto
show finite ?B
  by (intro finite_imageI finite_SigmaI finite_atMost finite_lessThan finite_PiE finite_Basis)
qed
then show finite  $\{L \in ?D0. K \subseteq L\}$  if  $K \in ?D0$  for  $K$ 

```

```

    using that by auto
  qed
  let ?D1 = {K ∈ D0. ∃ x ∈ S ∩ K. K ⊆ g x}
  obtain D where count: countable D
    and sub: ⋃ D ⊆ cbox a b S ⊆ ⋃ D
    and int: ⋀ K. K ∈ D ⇒ (interior K ≠ {}) ∧ (∃ c d. K = cbox c d)
    and intdj: ⋀ A B. [A ∈ D; B ∈ D] ⇒ A ⊆ B ∨ B ⊆ A ∨ interior A
    ∩ interior B = {}
    and SK: ⋀ K. K ∈ D ⇒ ∃ x. x ∈ S ∩ K ∧ K ⊆ g x
    and cbox: ⋀ u v. cbox u v ∈ D ⇒ ∃ n. ∀ i ∈ Basis. v · i - u · i = (b ·
    i - a · i) / 2n
    and fin: ⋀ K. K ∈ D ⇒ finite {L. L ∈ D ∧ K ⊆ L}
  proof
    show countable ?D1 using count countable_subset
      by (simp add: count countable_subset)
    show ⋃ ?D1 ⊆ cbox a b
      using sub by blast
    show S ⊆ ⋃ ?D1
      using SK by (force simp:)
    show ⋀ K. K ∈ ?D1 ⇒ (interior K ≠ {}) ∧ (∃ c d. K = cbox c d)
      using int by blast
    show ⋀ A B. [A ∈ ?D1; B ∈ ?D1] ⇒ A ⊆ B ∨ B ⊆ A ∨ interior A ∩ interior
    B = {}
      using intdj by blast
    show ⋀ K. K ∈ ?D1 ⇒ ∃ x. x ∈ S ∩ K ∧ K ⊆ g x
      by auto
    show ⋀ u v. cbox u v ∈ ?D1 ⇒ ∃ n. ∀ i ∈ Basis. v · i - u · i = (b · i - a ·
    i) / 2n
      using cbox by blast
    show ⋀ K. K ∈ ?D1 ⇒ finite {L. L ∈ ?D1 ∧ K ⊆ L}
      using fin by simp (metis (mono_tags, lifting) Collect_mono rev_finite_subset)
  qed
  let ?D = {K ∈ D. ∀ K'. K' ∈ D ∧ K ≠ K' → ¬(K ⊆ K')}
  show ?thesis
  proof
    show countable ?D
      by (blast intro: countable_subset [OF _ count])
    show ⋃ ?D ⊆ cbox a b
      using sub by blast
    show S ⊆ ⋃ ?D
  proof clarsimp
    fix x
    assume x ∈ S
    then obtain X where x ∈ X X ∈ D using ⟨S ⊆ ⋃ D⟩ by blast
    let ?R = {(K,L). K ∈ D ∧ L ∈ D ∧ L ⊆ K}
    have irrR: irrefl ?R by (force simp: irrefl_def)
    have traR: trans ?R by (force simp: trans_def)
    have finR: ⋀ x. finite {y. (y, x) ∈ ?R}
    by simp (metis (mono_tags, lifting) fin ⟨X ∈ D⟩ finite_subset mem_Collect_eq

```

```

psubset_imp_subset subsetI)
  have  $\{X \in \mathcal{D}. x \in X\} \neq \{\}$ 
    using  $\langle X \in \mathcal{D} \rangle \langle x \in X \rangle$  by blast
  then obtain  $Y$  where  $Y \in \{X \in \mathcal{D}. x \in X\} \wedge Y'. (Y', Y) \in ?R \implies Y' \notin \{X \in \mathcal{D}. x \in X\}$ 
    by (rule wfE_min' [OF wf_finite_segments [OF irrR traR finR]]) blast
  then show  $\exists Y. Y \in \mathcal{D} \wedge (\forall K'. K' \in \mathcal{D} \wedge Y \neq K' \longrightarrow \neg Y \subseteq K') \wedge x \in Y$ 
    by blast
qed
show  $\bigwedge K. K \in ?\mathcal{D} \implies \text{interior } K \neq \{\} \wedge (\exists c d. K = \text{cbox } c d)$ 
  by (blast intro: dest: int)
show pairwise  $(\lambda A B. \text{interior } A \cap \text{interior } B = \{\}) ?\mathcal{D}$ 
  using intdj by (simp add: pairwise_def) metis
show  $\bigwedge K. K \in ?\mathcal{D} \implies \exists x \in S \cap K. K \subseteq g x$ 
  using SK by force
show  $\bigwedge u v. \text{cbox } u v \in ?\mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$ 
  using cbox by force
qed
qed

```

7.13.15 Division filter

Divisions over all gauges towards finer divisions.

definition *division_filter* :: $'a::\text{euclidean_space set} \Rightarrow ('a \times 'a \text{ set}) \text{ set filter}$
 where *division_filter* $s = (\text{INF } g \in \{g. \text{gauge } g\}. \text{principal } \{p. p \text{ tagged_division_of } s \wedge g \text{ fine } p\})$

proposition *eventually_division_filter*:

$(\forall_F p \text{ in } \text{division_filter } s. P p) \longleftrightarrow$
 $(\exists g. \text{gauge } g \wedge (\forall p. p \text{ tagged_division_of } s \wedge g \text{ fine } p \longrightarrow P p))$

unfolding *division_filter_def*

proof (subst *eventually_INF_base*; *clarsimp*)

fix $g1 g2 :: 'a \Rightarrow 'a \text{ set}$ **show** $\text{gauge } g1 \implies \text{gauge } g2 \implies \exists x. \text{gauge } x \wedge \{p. p \text{ tagged_division_of } s \wedge x \text{ fine } p\} \subseteq \{p. p \text{ tagged_division_of } s \wedge g1 \text{ fine } p\} \wedge$

$\{p. p \text{ tagged_division_of } s \wedge x \text{ fine } p\} \subseteq \{p. p \text{ tagged_division_of } s \wedge g2 \text{ fine } p\}$

by (intro *exI*[of $\lambda x. g1 x \cap g2 x$]) (auto simp: *fine_Int*)

qed (auto simp: *eventually_principal*)

lemma *division_filter_not_empty*: *division_filter* $(\text{cbox } a b) \neq \text{bot}$

unfolding *trivial_limit_def* *eventually_division_filter*

by (auto elim: *fine_division_exists*)

lemma *eventually_division_filter_tagged_division*:

eventually $(\lambda p. p \text{ tagged_division_of } s) (\text{division_filter } s)$

using *eventually_division_filter* by auto

2554

end

7.14 Henstock-Kurzweil Gauge Integration in Many Dimensions

theory *Henstock_Kurzweil_Integration*

imports

Lebesgue_Measure Tagged_Division

begin

lemma *norm_diff2*: $\llbracket y = y1 + y2; x = x1 + x2; e = e1 + e2; \text{norm}(y1 - x1) \leq e1; \text{norm}(y2 - x2) \leq e2 \rrbracket$
 $\implies \text{norm}(y-x) \leq e$
by (*smt (verit, ccfv_SIG) norm_diff_triangle_ineq*)

lemma *setcomp_dot1*: $\{z. P(z \cdot (i,0))\} = \{(x,y). P(x \cdot i)\}$
by *auto*

lemma *setcomp_dot2*: $\{z. P(z \cdot (0,i))\} = \{(x,y). P(y \cdot i)\}$
by *auto*

lemma *Sigma_Int_Paircomp1*: $(\text{Sigma } A B) \cap \{(x, y). P x\} = \text{Sigma } (A \cap \{x. P x\}) B$
by *blast*

lemma *Sigma_Int_Paircomp2*: $(\text{Sigma } A B) \cap \{(x, y). P y\} = \text{Sigma } A (\lambda z. B z \cap \{y. P y\})$
by *blast*

7.14.1 Content (length, area, volume...) of an interval

abbreviation *content* :: $'a::\text{euclidean_space}$ $\text{set} \Rightarrow \text{real}$
where $\text{content } s \equiv \text{measure } \text{l borel } s$

lemma *content_cbox_cases*:
 $\text{content } (\text{cbox } a b) = (\text{if } \forall i \in \text{Basis}. a \cdot i \leq b \cdot i \text{ then } \text{prod } (\lambda i. b \cdot i - a \cdot i) \text{ Basis else } 0)$
by (*simp add: measure_lborel_cbox_eq inner_diff*)

lemma *content_cbox*: $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i \implies \text{content } (\text{cbox } a b) = (\prod i \in \text{Basis}. b \cdot i - a \cdot i)$
unfolding *content_cbox_cases* **by** *simp*

lemma *content_cbox'*: $\text{cbox } a b \neq \{\} \implies \text{content } (\text{cbox } a b) = (\prod i \in \text{Basis}. b \cdot i - a \cdot i)$
by (*simp add: box_ne_empty inner_diff*)

lemma *content_cbox_if*: $\text{content } (\text{cbox } a b) = (\text{if } \text{cbox } a b = \{\} \text{ then } 0 \text{ else}$

$\prod_{i \in \text{Basis.}} b \cdot i - a \cdot i$
by (*simp add: content_cbox'*)

lemma *content_cbox_cart*:

$\text{cbox } a \ b \neq \{\} \implies \text{content}(\text{cbox } a \ b) = \text{prod } (\lambda i. b\$i - a\$i) \ \text{UNIV}$
by (*simp add: content_cbox_if Basis_vec_def cart_eq_inner_axis axis_eq_axis prod.UNION_disjoint*)

lemma *content_cbox_if_cart*:

$\text{content}(\text{cbox } a \ b) = (\text{if } \text{cbox } a \ b = \{\} \text{ then } 0 \text{ else } \text{prod } (\lambda i. b\$i - a\$i) \ \text{UNIV})$
by (*simp add: content_cbox_cart*)

lemma *content_division_of*:

assumes $K \in \mathcal{D} \ \mathcal{D} \ \text{division_of } S$
shows $\text{content } K = (\prod_{i \in \text{Basis.}} \text{interval_upperbound } K \cdot i - \text{interval_lowerbound } K \cdot i)$
proof –
obtain $a \ b$ **where** $K = \text{cbox } a \ b$
using *cbox_division_memE assms by metis*
then show *?thesis*
using *assms by (force simp: division_of_def content_cbox')*
qed

lemma *content_real*: $a \leq b \implies \text{content } \{a..b\} = b - a$
by *simp*

lemma *abs_eq_content*: $|y - x| = (\text{if } x \leq y \text{ then } \text{content } \{x..y\} \text{ else } \text{content } \{y..x\})$
by (*auto simp: content_real*)

lemma *content_singleton*: $\text{content } \{a\} = 0$
by *simp*

lemma *content_unit[iff]*: $\text{content } (\text{cbox } 0 \ (\text{One}::'a::\text{euclidean_space})) = 1$
by *simp*

lemma *content_pos_le [iff]*: $0 \leq \text{content } X$
by *simp*

corollary *content_nonneg [simp]*: $\neg \text{content } (\text{cbox } a \ b) < 0$
using *not_le by blast*

lemma *content_pos_lt*: $\forall i \in \text{Basis. } a \cdot i < b \cdot i \implies 0 < \text{content } (\text{cbox } a \ b)$
by (*auto simp: less_imp_le inner_diff box_eq_empty intro!: prod_pos*)

lemma *content_eq_0*: $\text{content } (\text{cbox } a \ b) = 0 \iff (\exists i \in \text{Basis. } b \cdot i \leq a \cdot i)$
by (*auto simp: content_cbox_cases not_le intro: less_imp_le antisym eq_refl*)

lemma *content_eq_0_interior*: $\text{content } (\text{cbox } a \ b) = 0 \iff \text{interior}(\text{cbox } a \ b) = \{\}$

unfolding *content_eq_0 interior_cbox box_eq_empty* **by** *auto*

lemma *content_pos_lt_eq*: $0 < \text{content } (\text{cbox } a \text{ } (b::'a::\text{euclidean_space})) \longleftrightarrow$
 $(\forall i \in \text{Basis}. a \cdot i < b \cdot i)$
by (*auto simp add: content_cbox_cases less_le prod_nonneg*)

lemma *content_empty* [*simp*]: $\text{content } \{\} = 0$
by *simp*

lemma *content_real_if* [*simp*]: $\text{content } \{a..b\} = (\text{if } a \leq b \text{ then } b - a \text{ else } 0)$
by (*simp add: content_real*)

lemma *content_subset*: $\text{cbox } a \ b \subseteq \text{cbox } c \ d \implies \text{content } (\text{cbox } a \ b) \leq \text{content } (\text{cbox } c \ d)$
unfolding *measure_def*
by (*intro enn2real_mono emeasure_mono*) (*auto simp: emeasure_lborel_cbox_eq*)

lemma *content_lt_nz*: $0 < \text{content } (\text{cbox } a \ b) \longleftrightarrow \text{content } (\text{cbox } a \ b) \neq 0$
by (*fact zero_less_measure_iff*)

lemma *content_Pair*: $\text{content } (\text{cbox } (a,c) \ (b,d)) = \text{content } (\text{cbox } a \ b) * \text{content } (\text{cbox } c \ d)$
unfolding *measure_lborel_cbox_eq Basis_prod_def*
apply (*subst prod.union_disjoint*)
apply (*auto simp: bex_Un ball_Un*)
apply (*subst (1 2) prod.reindex_nontrivial*)
apply *auto*
done

lemma *content_cbox_pair_eq0_D*:
 $\text{content } (\text{cbox } (a,c) \ (b,d)) = 0 \implies \text{content } (\text{cbox } a \ b) = 0 \vee \text{content } (\text{cbox } c \ d) = 0$
by (*simp add: content_Pair*)

lemma *content_cbox_plus*:
fixes $x :: 'a::\text{euclidean_space}$
shows $\text{content}(\text{cbox } x \ (x + h *_{\mathbb{R}} \text{One})) = (\text{if } h \geq 0 \text{ then } h \wedge \text{DIM}('a) \text{ else } 0)$
by (*simp add: algebra_simps content_cbox_if box_eq_empty*)

lemma *content_0_subset*: $\text{content}(\text{cbox } a \ b) = 0 \implies s \subseteq \text{cbox } a \ b \implies \text{content } s = 0$
using *emeasure_mono*[*of s cbox a b lborel*]
by (*auto simp: measure_def enn2real_eq_0_iff emeasure_lborel_cbox_eq*)

lemma *content_ball_pos*:
assumes $r > 0$
shows $\text{content } (\text{ball } c \ r) > 0$

proof —

from *rational_boxes*[*OF assms, of c*] **obtain** $a \ b$ **where** $ab: c \in \text{box } a \ b \ \text{box } a \ b$


```

 $\subseteq$  ball c r
  by auto
  then have  $0 < \text{content} (\text{box } a \ b)$ 
    by (subst measure_lborel_box_eq) (auto intro!: prod_pos simp: algebra_simps
    box_def)
  have  $\text{emeasure lborel} (\text{box } a \ b) \leq \text{emeasure lborel} (\text{ball } c \ r)$ 
    using ab by (intro emeasure_mono) auto
  then show ?thesis
    using  $\langle \text{content} (\text{box } a \ b) > 0 \rangle$ 
    by (smt (verit, best) Sigma_Algebra.measure_def emeasure_lborel_ball_finite
    enn2real_mono infinity_ennreal_def)
qed

```

```

lemma content_cball_pos:
  assumes  $r > 0$ 
  shows  $\text{content} (\text{cball } c \ r) > 0$ 
proof -
  from rational_boxes[OF assms, of c] obtain a b where ab:  $c \in \text{box } a \ b \ \text{box } a \ b$ 
 $\subseteq$  ball c r
  by auto
  then have  $0 < \text{content} (\text{box } a \ b)$ 
    by (subst measure_lborel_box_eq) (auto intro!: prod_pos simp: algebra_simps
    box_def)
  have  $\text{emeasure lborel} (\text{box } a \ b) \leq \text{emeasure lborel} (\text{cball } c \ r)$ 
    using ab by (intro emeasure_mono) auto
  also have  $\text{emeasure lborel} (\text{box } a \ b) = \text{ennreal} (\text{content} (\text{box } a \ b))$ 
    using emeasure_lborel_box_finite[of a b] by (intro emeasure_eq_ennreal_measure)
  auto
  also have  $\text{emeasure lborel} (\text{cball } c \ r) = \text{ennreal} (\text{content} (\text{cball } c \ r))$ 
    using emeasure_lborel_cball_finite[of c r] by (intro emeasure_eq_ennreal_measure)
  auto
  finally show ?thesis
    using  $\langle \text{content} (\text{box } a \ b) > 0 \rangle$  by simp
qed

```

```

lemma content_split:
  fixes a :: 'a::euclidean_space
  assumes  $k \in \text{Basis}$ 
  shows  $\text{content} (\text{cbox } a \ b) = \text{content}(\text{cbox } a \ b \cap \{x. x \cdot k \leq c\}) + \text{content}(\text{cbox } a \ b \cap \{x. x \cdot k \geq c\})$ 
  — Prove using measure theory
proof (cases  $\forall i \in \text{Basis}. a \cdot i \leq b \cdot i$ )
  case True
  have 1:  $\bigwedge X \ Y \ Z. (\prod i \in \text{Basis}. Z \ i \ (\text{if } i = k \ \text{then } X \ \text{else } Y \ i)) = Z \ k \ X \ * \ (\prod i \in \text{Basis} - \{k\}. Z \ i \ (Y \ i))$ 
    by (simp add: if_distrib prod.delta_remove assms)
  note simp = interval_split[OF assms] content_cbox_cases
  have 2:  $(\prod i \in \text{Basis}. b \cdot i - a \cdot i) = (\prod i \in \text{Basis} - \{k\}. b \cdot i - a \cdot i) * (b \cdot k - a \cdot k)$ 
    by (metis (no_types, lifting) assms finite_Basis mult.commute prod.remove)

```

2558

```

have  $\bigwedge x. \min (b \cdot k) c = \max (a \cdot k) c \implies$ 
   $x * (b \cdot k - a \cdot k) = x * (\max (a \cdot k) c - a \cdot k) + x * (b \cdot k - \max (a \cdot k) c)$ 
  by (auto simp add: field_simps)
moreover
have **:  $(\prod_{i \in \text{Basis}}. ((\sum_{i \in \text{Basis}}. (\text{if } i = k \text{ then } \min (b \cdot k) c \text{ else } b \cdot i) *_R i)$ 
 $\cdot i - a \cdot i)) =$ 
   $(\prod_{i \in \text{Basis}}. (\text{if } i = k \text{ then } \min (b \cdot k) c \text{ else } b \cdot i) - a \cdot i)$ 
   $(\prod_{i \in \text{Basis}}. b \cdot i - ((\sum_{i \in \text{Basis}}. (\text{if } i = k \text{ then } \max (a \cdot k) c \text{ else } a \cdot i) *_R i)$ 
 $\cdot i)) =$ 
   $(\prod_{i \in \text{Basis}}. b \cdot i - (\text{if } i = k \text{ then } \max (a \cdot k) c \text{ else } a \cdot i))$ 
  by (auto intro!: prod.cong)
have  $\neg a \cdot k \leq c \implies \neg c \leq b \cdot k \implies \text{False}$ 
  unfolding not_le using True assms by auto
ultimately show ?thesis
  using assms unfolding simpns ** 1[of  $\lambda i x. b \cdot i - x$ ] 1[of  $\lambda i x. x - a \cdot i$ ] 2
  by auto
next
case False
then have cbox a b = {}
  unfolding cbox_eq_empty by (auto simp: not_le)
then show ?thesis
  by (auto simp: not_le)
qed

```

```

lemma division_of_content_0:
  assumes content (cbox a b) = 0 d division_of (cbox a b)  $K \in d$ 
  shows content  $K = 0$ 
  unfolding forall_in_division[OF assms(2)]
  by (meson assms content_0_subset division_of_def)

```

```

lemma sum_content_null:
  assumes content (cbox a b) = 0
  and p tagged_division_of (cbox a b)
  shows  $(\sum_{(x,K) \in p. \text{content } K *_R f x) = (0::'a::\text{real\_normed\_vector})$ 
proof (intro sum.neutral strip)
  fix y
  assume y:  $y \in p$ 
  obtain x K where xk:  $y = (x, K)$ 
  using surj_pair[of y] by blast
  then obtain c d where k:  $K = \text{cbox } c \text{ d } K \subseteq \text{cbox } a \text{ b}$ 
  by (metis assms(2) tagged_division_ofD(3) tagged_division_ofD(4) y)
  have  $(\lambda(x',K'). \text{content } K' *_R f x') y = \text{content } K *_R f x$ 
  unfolding xk by auto
  also have ... = 0
  using assms(1) content_0_subset k(2) by auto
  finally show  $(\lambda(x, k). \text{content } k *_R f x) y = 0$  .
qed

```

global_interpretation sum_content: operative plus 0 content

```

rewrites comm_monoid_set.F plus 0 = sum
proof -
  interpret operative plus 0 content
  by standard (auto simp add: content_split [symmetric] content_eq_0_interior)
  show operative plus 0 content
  by standard
  show comm_monoid_set.F plus 0 = sum
  by (simp add: sum_def)
qed

lemma additive_content_division: d division_of (cbox a b)  $\implies$  sum content d =
content (cbox a b)
  by (fact sum_content.division)

lemma additive_content_tagged_division:
  d tagged_division_of (cbox a b)  $\implies$  sum ( $\lambda(x,l). \text{content } l$ ) d = content (cbox a
b)
  by (fact sum_content.tagged_division)

lemma subadditive_content_division:
  assumes  $\mathcal{D}$  division_of  $S$   $S \subseteq \text{cbox } a \ b$ 
  shows sum content  $\mathcal{D} \leq \text{content}(\text{cbox } a \ b)$ 
proof -
  have  $\mathcal{D}$  division_of  $\bigcup \mathcal{D}$   $\bigcup \mathcal{D} \subseteq \text{cbox } a \ b$ 
  using assms by auto
  then obtain  $\mathcal{D}'$  where  $\mathcal{D} \subseteq \mathcal{D}'$   $\mathcal{D}'$  division_of cbox a b
  using partial_division_extend_interval by metis
  then have sum content  $\mathcal{D} \leq \text{sum content } \mathcal{D}'$ 
  using sum_mono2 by blast
  also have ...  $\leq \text{content}(\text{cbox } a \ b)$ 
  by (simp add:  $\langle \mathcal{D}' \text{ division\_of } \text{cbox } a \ b \rangle$  additive_content_division less_eq_real_def)
  finally show ?thesis .
qed

lemma content_real_eq_0: content {a..b::real} = 0  $\iff$  a  $\geq$  b
  by simp

lemma property_empty_interval:  $\forall a \ b. \text{content } (\text{cbox } a \ b) = 0 \implies P (\text{cbox } a \ b)$ 
 $\implies P \{ \}$ 
  using content_empty unfolding empty_as_interval by auto

lemma interval_bounds_nz_content [simp]:
  assumes content (cbox a b)  $\neq$  0
  shows interval_upperbound (cbox a b) = b
  and interval_lowerbound (cbox a b) = a
  by (metis assms content_empty interval_bounds')+

```

7.14.2 Gauge integral

Case distinction to define it first on compact intervals first, then use a limit. This is only much later unified. In Fremlin: Measure Theory, Volume 4I this is generalized using residual sets.

definition *has_integral* :: ('n::euclidean_space \Rightarrow 'b::real_normed_vector) \Rightarrow 'b \Rightarrow 'n set \Rightarrow bool

(**infixr** *has'_integral* 46)

where (*f has_integral I*) *s* \longleftrightarrow

(if $\exists a b. s = \text{cbox } a b$

then $((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R f x) \longrightarrow I)$ (*division_filter s*)

else $(\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$

$(\exists z. ((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R (\text{if } x \in s \text{ then } f x \text{ else } 0)) \longrightarrow z)$

(*division_filter (cbox a b)*) \wedge

$\text{norm } (z - I) < e)$)

lemma *has_integral_cbox*:

(*f has_integral I*) (*cbox a b*) $\longleftrightarrow ((\lambda p. \sum_{(x,k) \in p. \text{content } k *_R f x) \longrightarrow I)$

(*division_filter (cbox a b)*)

by (*auto simp add: has_integral_def*)

lemma *has_integral*:

(*f has_integral y*) (*cbox a b*) \longleftrightarrow

$(\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$

$(\forall \mathcal{D}. \mathcal{D} \text{ tagged_division_of } (\text{cbox } a b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$

$\text{norm } (\text{sum } (\lambda(x,k). \text{content}(k) *_R f x) \mathcal{D} - y) < e)$)

by (*auto simp: dist_norm eventually_division_filter has_integral_def tendsto_iff*)

lemma *has_integral_real*:

(*f has_integral y*) {*a..b*::real} \longleftrightarrow

$(\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$

$(\forall \mathcal{D}. \mathcal{D} \text{ tagged_division_of } \{a..b\} \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$

$\text{norm } (\text{sum } (\lambda(x,k). \text{content}(k) *_R f x) \mathcal{D} - y) < e)$)

unfolding *box_real[symmetric]* **by** (*rule has_integral*)

lemma *has_integralD[dest]*:

assumes (*f has_integral y*) (*cbox a b*)

and $e > 0$

obtains γ

where *gauge* γ

and $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged_division_of } (\text{cbox } a b) \implies \gamma \text{ fine } \mathcal{D} \implies$

$\text{norm } ((\sum_{(x,k) \in \mathcal{D}. \text{content } k *_R f x) - y) < e$

using *assms unfolding has_integral by auto*

lemma *has_integral_alt*:

(*f has_integral y*) *i* \longleftrightarrow

(if $\exists a b. i = \text{cbox } a b$

then (*f has_integral y*) *i*

else $(\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$

($\exists z. ((\lambda x. \text{if } x \in i \text{ then } f x \text{ else } 0) \text{ has_integral } z) (\text{cbox } a \ b) \wedge \text{norm } (z - y) < e$))
by (subst has_integral_def) (auto simp add: has_integral_cbox)

lemma has_integral_altD:

assumes (f has_integral y) i
and $\neg (\exists a \ b. i = \text{cbox } a \ b)$
and $e > 0$
obtains B **where** $B > 0$
and $\forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow$
 $(\exists z. ((\lambda x. \text{if } x \in i \text{ then } f(x) \text{ else } 0) \text{ has_integral } z) (\text{cbox } a \ b) \wedge \text{norm}(z - y) < e)$
using assms has_integral_alt[of f y i] **by** auto

definition integrable_on (infixr integrable'_on 46)

where $f \text{ integrable_on } i \longleftrightarrow (\exists y. (f \text{ has_integral } y) i)$

definition integral i f = (SOME y. (f has_integral y) i $\vee \neg f \text{ integrable_on } i \wedge y=0$)

lemma integrable_integral[intro]: $f \text{ integrable_on } i \Longrightarrow (f \text{ has_integral } (\text{integral } i \ f)) \ i$

unfolding integrable_on_def integral_def **by** (metis (mono_tags, lifting))

lemma not_integrable_integral: $\neg f \text{ integrable_on } i \Longrightarrow \text{integral } i \ f = 0$

unfolding integrable_on_def integral_def **by** blast

lemma has_integral_integrable[dest]: $(f \text{ has_integral } i) \ s \Longrightarrow f \text{ integrable_on } s$

unfolding integrable_on_def **by** auto

lemma has_integral_integral: $f \text{ integrable_on } s \longleftrightarrow (f \text{ has_integral } (\text{integral } s \ f)) \ s$

by auto

7.14.3 Basic theorems about integrals

lemma has_integral_eq_rhs: $(f \text{ has_integral } j) \ S \Longrightarrow i = j \Longrightarrow (f \text{ has_integral } i) \ S$

by (rule forw_subst)

lemma has_integral_unique_cbox:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{real_normed_vector}$
shows $(f \text{ has_integral } k1) (\text{cbox } a \ b) \Longrightarrow (f \text{ has_integral } k2) (\text{cbox } a \ b) \Longrightarrow k1 = k2$

by (meson division_filter_not_empty has_integral_cbox tendsto_unique)

lemma has_integral_unique:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{real_normed_vector}$
assumes $(f \text{ has_integral } k1) \ i \ (f \text{ has_integral } k2) \ i$

shows $k1 = k2$
proof (rule ccontr)
let $?e = \text{norm } (k1 - k2)/2$
let $?F = (\lambda x. \text{if } x \in i \text{ then } f \ x \text{ else } 0)$
assume $k1 \neq k2$
then have $e: ?e > 0$
by auto
have nonbox: $\neg (\exists a \ b. i = \text{cbox } a \ b)$
using $\langle k1 \neq k2 \rangle$ assms has_integral_unique_cbox **by** blast
obtain $B1$ **where** $B1$:
 $0 < B1$
 $\bigwedge a \ b. \text{ball } 0 \ B1 \subseteq \text{cbox } a \ b \implies$
 $\exists z. (?F \text{ has_integral } z) (\text{cbox } a \ b) \wedge \text{norm } (z - k1) < \text{norm } (k1 - k2)/2$
by (rule has_integral_altD[OF assms(1) nonbox, OF e]) blast
obtain $B2$ **where** $B2$:
 $0 < B2$
 $\bigwedge a \ b. \text{ball } 0 \ B2 \subseteq \text{cbox } a \ b \implies$
 $\exists z. (?F \text{ has_integral } z) (\text{cbox } a \ b) \wedge \text{norm } (z - k2) < \text{norm } (k1 - k2)/2$
by (rule has_integral_altD[OF assms(2) nonbox, OF e]) blast
obtain $a \ b :: 'n$ **where** $ab: \text{ball } 0 \ B1 \subseteq \text{cbox } a \ b \ \text{ball } 0 \ B2 \subseteq \text{cbox } a \ b$
by (metis Un_subset_iff bounded_Un bounded_ball bounded_subset_cbox_symmetric)
obtain w **where** $w: (?F \text{ has_integral } w) (\text{cbox } a \ b) \ \text{norm } (w - k1) < \text{norm } (k1 - k2)/2$
using $B1(2)$ [OF $ab(1)$] **by** blast
obtain z **where** $z: (?F \text{ has_integral } z) (\text{cbox } a \ b) \ \text{norm } (z - k2) < \text{norm } (k1 - k2)/2$
using $B2(2)$ [OF $ab(2)$] **by** blast
have $z = w$
using has_integral_unique_cbox[OF $w(1)$ $z(1)$] **by** auto
then have $\text{norm } (k1 - k2) \leq \text{norm } (z - k2) + \text{norm } (w - k1)$
using norm_triangle_ineq4 [of $k1 - w$ $k2 - z$]
by (auto simp add: norm_minus_commute)
also have $\dots < \text{norm } (k1 - k2)/2 + \text{norm } (k1 - k2)/2$
by (metis add_strict_mono $z(2)$ $w(2)$)
finally show False **by** auto
qed

lemma integral_unique [intro]: $(f \text{ has_integral } y) \ k \implies \text{integral } k \ f = y$
unfolding integral_def
by (rule some_equality) (auto intro: has_integral_unique)

lemma has_integral_iff: $(f \text{ has_integral } i) \ S \longleftrightarrow (f \text{ integrable_on } S \wedge \text{integral } S \ f = i)$
by blast

lemma eq_integralD: $\text{integral } k \ f = y \implies (f \text{ has_integral } y) \ k \vee \neg f \text{ integrable_on } k \wedge y=0$
unfolding integral_def integrable_on_def
by (metis (mono_tags, lifting))

lemma *has_integral_const* [intro]:
fixes $a\ b :: 'a::\text{euclidean_space}$
shows $((\lambda x. c)\ \text{has_integral}\ (\text{content}\ (\text{cbox}\ a\ b)\ *_{\mathbb{R}}\ c))\ (\text{cbox}\ a\ b)$
using *eventually_division_filter_tagged_division*[of $\text{cbox}\ a\ b$]
additive_content_tagged_division[of $_ a\ b$]
by (*auto simp: has_integral_cbox_split_beta' scaleR_sum_left[symmetric]*
elim!: eventually_mono intro!: tendsto_cong[THEN iffD1, OF _ tendsto_const])

lemma *has_integral_const_real* [intro]:
fixes $a\ b :: \text{real}$
shows $((\lambda x. c)\ \text{has_integral}\ (\text{content}\ \{a..b\}\ *_{\mathbb{R}}\ c))\ \{a..b\}$
by (*metis box_real(2) has_integral_const*)

lemma *has_integral_integrable_integral*: $(f\ \text{has_integral}\ i)\ s \longleftrightarrow f\ \text{integrable_on}\ s \wedge \text{integral}\ s\ f = i$
by *blast*

lemma *integral_const* [simp]:
fixes $a\ b :: 'a::\text{euclidean_space}$
shows $\text{integral}\ (\text{cbox}\ a\ b)\ (\lambda x. c) = \text{content}\ (\text{cbox}\ a\ b)\ *_{\mathbb{R}}\ c$
by *blast*

lemma *integral_const_real* [simp]:
fixes $a\ b :: \text{real}$
shows $\text{integral}\ \{a..b\}\ (\lambda x. c) = \text{content}\ \{a..b\}\ *_{\mathbb{R}}\ c$
by *blast*

lemma *has_integral_is_0_cbox*:
fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{real_normed_vector}$
assumes $\bigwedge x. x \in \text{cbox}\ a\ b \implies f\ x = 0$
shows $(f\ \text{has_integral}\ 0)\ (\text{cbox}\ a\ b)$
unfolding *has_integral_cbox*
using *eventually_division_filter_tagged_division*[of $\text{cbox}\ a\ b$] *assms*
by (*subst tendsto_cong[where $g=\lambda_. 0$]*)
(auto elim!: eventually_mono intro!: sum.neutral simp: tag_in_interval)

lemma *has_integral_is_0*:
fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{real_normed_vector}$
assumes $\bigwedge x. x \in S \implies f\ x = 0$
shows $(f\ \text{has_integral}\ 0)\ S$
proof (*cases* $(\exists a\ b. S = \text{cbox}\ a\ b)$)
case *True* **with** *assms has_integral_is_0_cbox* **show** *?thesis*
by *blast*
next
case *False*
have $*$: $(\lambda x. \text{if}\ x \in S\ \text{then}\ f\ x\ \text{else}\ 0) = (\lambda x. 0)$
by (*auto simp add: assms*)

2564

```
show ?thesis
  using has_integral_is_0_cbox False
  by (subst has_integral_alt) (force simp add: *)
qed

lemma has_integral_0[simp]: (( $\lambda x::'n::euclidean\_space.$  0) has_integral 0) S
  by (rule has_integral_is_0) auto

lemma has_integral_0_eq[simp]: (( $\lambda x.$  0) has_integral i) S  $\longleftrightarrow$  i = 0
  using has_integral_unique[OF has_integral_0] by auto

lemma has_integral_linear_cbox:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes f: (f has_integral y) (cbox a b)
  and h: bounded_linear h
  shows ((h  $\circ$  f) has_integral (h y)) (cbox a b)
proof -
  interpret bounded_linear h using h .
  show ?thesis
    unfolding has_integral_cbox using tendsto [OF f [unfolded has_integral_cbox]]
    by (simp add: sum scaleR split_beta')
qed

lemma has_integral_linear:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes f: (f has_integral y) S
  and h: bounded_linear h
  shows ((h  $\circ$  f) has_integral (h y)) S
proof (cases ( $\exists a b. S = \text{cbox } a b$ ))
  case True with f h has_integral_linear_cbox show ?thesis
    by blast
next
  case False
  interpret h using h .
  from pos_bounded obtain B where B: 0 < B  $\wedge$   $\forall x. \text{norm } (h x) \leq \text{norm } x * B$ 
  by blast
  let ?S =  $\lambda f x. \text{if } x \in S \text{ then } f x \text{ else } 0$ 
  show ?thesis
  proof (subst has_integral_alt, clarsimp simp: False)
    fix e :: real
    assume e: e > 0
    have *: 0 < e/B using e B(1) by simp
    obtain M where M:
      M > 0
       $\bigwedge a b. \text{ball } 0 M \subseteq \text{cbox } a b \implies$ 
       $\exists z. (?S f \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - y) < e/B$ 
    using has_integral_altD[OF f False *] by blast
    show  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
      ( $\exists z. (?S(h \circ f) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - h y) < e$ )
```



```

proof (rule exI, intro allI conjI impI)
  show  $M > 0$  using  $M$  by metis
next
  fix  $a b :: 'n$ 
  assume  $sb$ :  $ball\ 0\ M \subseteq cbox\ a\ b$ 
  obtain  $z$  where  $z$ :  $(?S\ f\ has\_integral\ z)\ (cbox\ a\ b)\ norm\ (z - y) < e/B$ 
    using  $M(2)[OF\ sb]$  by blast
  have  $*$ :  $?S(h \circ f) = h \circ ?S\ f$ 
    using zero by auto
  show  $\exists z$ .  $(?S(h \circ f)\ has\_integral\ z)\ (cbox\ a\ b) \wedge norm\ (z - h\ y) < e$ 
proof (intro exI conjI)
  show  $(?S(h \circ f)\ has\_integral\ h\ z)\ (cbox\ a\ b)$ 
    by (simp add:  $*$  has\_integral\_linear\_cbox[OF  $z(1)\ h$ ])
  show  $norm\ (h\ z - h\ y) < e$ 
    by (metis  $B\ diff\ le\_less\_trans\ pos\_less\_divide\_eq\ z(2)$ )
qed
qed
qed
qed

lemma has\_integral\_scaleR\_left:
   $(f\ has\_integral\ y)\ S \implies ((\lambda x. f\ x *_{R}\ c)\ has\_integral\ (y *_{R}\ c))\ S$ 
  using has\_integral\_linear[OF \_ bounded\_linear\_scaleR\_left] by (simp add:
  comp\_def)

lemma integrable\_on\_scaleR\_left:
  assumes  $f$  integrable\_on  $A$ 
  shows  $(\lambda x. f\ x *_{R}\ y)$  integrable\_on  $A$ 
  using assms has\_integral\_scaleR\_left unfolding integrable\_on\_def by blast

lemma has\_integral\_mult\_left:
  fixes  $c :: \_ :: real\_normed\_algebra$ 
  shows  $(f\ has\_integral\ y)\ S \implies ((\lambda x. f\ x * c)\ has\_integral\ (y * c))\ S$ 
  using has\_integral\_linear[OF \_ bounded\_linear\_mult\_left] by (simp add: comp\_def)

lemma integrable\_on\_mult\_left:
  fixes  $c :: 'a :: real\_normed\_algebra$ 
  assumes  $f$  integrable\_on  $A$ 
  shows  $(\lambda x. f\ x * c)$  integrable\_on  $A$ 
  using assms has\_integral\_mult\_left by blast

lemma has\_integral\_divide:
  fixes  $c :: \_ :: real\_normed\_div\_algebra$ 
  shows  $(f\ has\_integral\ y)\ S \implies ((\lambda x. f\ x / c)\ has\_integral\ (y / c))\ S$ 
  unfolding divide\_inverse by (simp add: has\_integral\_mult\_left)

lemma integrable\_on\_divide:
  fixes  $c :: 'a :: real\_normed\_div\_algebra$ 
  assumes  $f$  integrable\_on  $A$ 

```

shows $(\lambda x. f x / c)$ *integrable_on* A
using *assms has_integral_divide* **by** *blast*

The case analysis eliminates the condition f *integrable_on* S at the cost of the type class constraint *division_ring*

corollary *integral_mult_left* [*simp*]:
fixes $c :: 'a :: \{real_normed_algebra, division_ring\}$
shows $integral\ S\ (\lambda x. f\ x * c) = integral\ S\ f * c$
proof (*cases f integrable_on S \vee c = 0*)
case *True* **then show** *?thesis*
by (*force intro: has_integral_mult_left*)
next
case *False* **then have** $\neg (\lambda x. f\ x * c)$ *integrable_on* S
using *has_integral_mult_left* [*of* $(\lambda x. f\ x * c)$ S *inverse* c]
by (*auto simp add: mult.assoc*)
with *False* **show** *?thesis* **by** (*simp add: not_integrable_integral*)
qed

corollary *integral_mult_right* [*simp*]:
fixes $c :: 'a :: \{real_normed_field\}$
shows $integral\ S\ (\lambda x. c * f\ x) = c * integral\ S\ f$
by (*simp add: mult.commute* [*of* c])

corollary *integral_divide* [*simp*]:
fixes $z :: 'a :: real_normed_field$
shows $integral\ S\ (\lambda x. f\ x / z) = integral\ S\ (\lambda x. f\ x) / z$
using *integral_mult_left* [*of* $S\ f$ *inverse* z]
by (*simp add: divide_inverse_commute*)

lemma *has_integral_mult_right*:
fixes $c :: 'a :: real_normed_algebra$
shows $(f\ has_integral\ y)\ A \implies ((\lambda x. c * f\ x)\ has_integral\ (c * y))\ A$
using *has_integral_linear*[*OF* $_ bounded_linear_mult_right$] **by** (*simp add: comp_def*)

lemma *integrable_on_mult_right*:
fixes $c :: 'a :: real_normed_algebra$
assumes f *integrable_on* A
shows $(\lambda x. c * f\ x)$ *integrable_on* A
using *assms has_integral_mult_right* **by** *blast*

lemma *integrable_on_mult_right_iff* [*simp*]:
fixes $c :: 'a :: real_normed_field$
assumes $c \neq 0$
shows $(\lambda x. c * f\ x)$ *integrable_on* $A \iff f$ *integrable_on* A
using *integrable_on_mult_right*[*of* $f\ A\ c$]
integrable_on_mult_right[*of* $\lambda x. c * f\ x\ A$ *inverse* c] *assms*
by (*auto simp: field_simps*)

lemma *integrable_on_mult_left_iff* [simp]:
fixes $c :: 'a :: \text{real_normed_field}$
assumes $c \neq 0$
shows $(\lambda x. f x * c) \text{ integrable_on } A \longleftrightarrow f \text{ integrable_on } A$
using *integrable_on_mult_right_iff*[OF *assms*, of $f A$] **by** (*simp add: mult.commute*)

lemma *integrable_on_div_iff* [simp]:
fixes $c :: 'a :: \text{real_normed_field}$
assumes $c \neq 0$
shows $(\lambda x. f x / c) \text{ integrable_on } A \longleftrightarrow f \text{ integrable_on } A$
using *integrable_on_mult_right_iff*[of *inverse c f A*] *assms* **by** (*simp add: field_simps*)

lemma *has_integral_cmul*: $(f \text{ has_integral } k) S \implies ((\lambda x. c *_R f x) \text{ has_integral } (c *_R k)) S$
unfolding *o_def*[*symmetric*]
by (*metis has_integral_linear bounded_linear_scaleR_right*)

lemma *has_integral_cmult_real*:
fixes $c :: \text{real}$
assumes $c \neq 0 \implies (f \text{ has_integral } x) A$
shows $((\lambda x. c * f x) \text{ has_integral } c * x) A$
by (*metis assms has_integral_is_0 has_integral_mult_right lambda_zero*)

lemma *has_integral_neg*: $(f \text{ has_integral } k) S \implies ((\lambda x. -(f x)) \text{ has_integral } -k) S$
by (*drule_tac c=-1 in has_integral_cmul*) *auto*

lemma *has_integral_neg_iff*: $((\lambda x. - f x) \text{ has_integral } k) S \longleftrightarrow (f \text{ has_integral } - k) S$
using *has_integral_neg*[of $f - k$] *has_integral_neg*[of $\lambda x. - f x k$] **by** *auto*

lemma *has_integral_add_cbox*:
fixes $f :: 'n :: \text{euclidean_space} \Rightarrow 'a :: \text{real_normed_vector}$
assumes $(f \text{ has_integral } k) (\text{cbox } a b)$ $(g \text{ has_integral } l) (\text{cbox } a b)$
shows $((\lambda x. f x + g x) \text{ has_integral } (k + l)) (\text{cbox } a b)$
using *assms*
unfolding *has_integral_cbox*
by (*simp add: split_beta' scaleR_add_right sum.distrib[abs_def] tendsto_add*)

lemma *has_integral_add*:
fixes $f :: 'n :: \text{euclidean_space} \Rightarrow 'a :: \text{real_normed_vector}$
assumes $f: (f \text{ has_integral } k) S$ **and** $g: (g \text{ has_integral } l) S$
shows $((\lambda x. f x + g x) \text{ has_integral } (k + l)) S$
proof (*cases* $\exists a b. S = \text{cbox } a b$)
case *True* **with** *has_integral_add_cbox* *assms* **show** *?thesis*
by *blast*
next
let $?S = \lambda f x. \text{if } x \in S \text{ then } f x \text{ else } 0$

```

case False
then show ?thesis
proof (subst has_integral_alt, clarsimp, goal_cases)
  case (1 e)
  then have e2:  $e/2 > 0$ 
  by auto
  obtain Bf where  $0 < Bf$ 
  and Bf:  $\bigwedge a b. \text{ball } 0 Bf \subseteq \text{cbox } a b \implies$ 
     $\exists z. (?S f \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - k) < e/2$ 
  using has_integral_altD[OF f False e2] by blast
  obtain Bg where  $0 < Bg$ 
  and Bg:  $\bigwedge a b. \text{ball } 0 Bg \subseteq (\text{cbox } a b) \implies$ 
     $\exists z. (?S g \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - l) < e/2$ 
  using has_integral_altD[OF g False e2] by blast
  show ?case
proof (rule_tac  $x = \max Bf Bg$  in exI, clarsimp simp add: max.strict_coboundedI1
<math>0 < Bf</math>›)
  fix a b
  assume ball_0 ( $\max Bf Bg$ )  $\subseteq$  cbox a ( $b::'n$ )
  then have fs:  $\text{ball } 0 Bf \subseteq \text{cbox } a (b::'n)$  and gs:  $\text{ball } 0 Bg \subseteq \text{cbox } a (b::'n)$ 
  by auto
  obtain w where w:  $(?S f \text{ has\_integral } w) (\text{cbox } a b) \text{norm } (w - k) < e/2$ 
  using Bf[OF fs] by blast
  obtain z where z:  $(?S g \text{ has\_integral } z) (\text{cbox } a b) \text{norm } (z - l) < e/2$ 
  using Bg[OF gs] by blast
  have *:  $\bigwedge x. (\text{if } x \in S \text{ then } f x + g x \text{ else } 0) = (?S f x) + (?S g x)$ 
  by auto
  show  $\exists z. (?S(\lambda x. f x + g x) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - (k +$ 
l))  $< e$ 
  proof (intro exI conjI)
    show  $(?S(\lambda x. f x + g x) \text{ has\_integral } (w + z)) (\text{cbox } a b)$ 
    by (simp add: has_integral_add_cbox[OF w(1) z(1), unfolded *[symmetric]])
    show  $\text{norm } (w + z - (k + l)) < e$ 
    by (metis dist_norm dist_triangle_add_half w(2) z(2))
  qed
qed
qed
qed

```

```

lemma has_integral_diff:
  ( $f \text{ has\_integral } k$ ) S  $\implies$  ( $g \text{ has\_integral } l$ ) S  $\implies$ 
  ( $(\lambda x. f x - g x) \text{ has\_integral } (k - l)$ ) S
  using has_integral_add[OF _ has_integral_neg, of f k S g l]
  by (auto simp: algebra_simps)

```

```

lemma integral_0 [simp]:
  integral S ( $\lambda x::'n::\text{euclidean\_space}. 0::'m::\text{real\_normed\_vector}$ ) = 0
  by auto

```

lemma *integral_add*: f integrable_on $S \implies g$ integrable_on $S \implies$
 $\text{integral } S (\lambda x. f x + g x) = \text{integral } S f + \text{integral } S g$
by (rule *integral_unique*) (metis *integrable_integral_has_integral_add*)

lemma *integral_cmul* [*simp*]: $\text{integral } S (\lambda x. c *_{\mathbb{R}} f x) = c *_{\mathbb{R}} \text{integral } S f$
proof (cases f integrable_on $S \vee c = 0$)
case *True* **with** *has_integral_cmul integrable_integral* **show** ?thesis
by *fastforce*
next
case *False* **then have** $\neg (\lambda x. c *_{\mathbb{R}} f x)$ integrable_on S
using *has_integral_cmul* [of $(\lambda x. c *_{\mathbb{R}} f x)$ S *inverse* c] **by** *auto*
with *False* **show** ?thesis **by** (*simp add: not_integrable_integral*)
qed

lemma *integral_mult*:
fixes $K :: \text{real}$
shows f integrable_on $X \implies K * \text{integral } X f = \text{integral } X (\lambda x. K * f x)$
by *simp*

lemma *integral_neg* [*simp*]: $\text{integral } S (\lambda x. - f x) = - \text{integral } S f$
by (metis *eq_integralD equation_minus_iff has_integral_iff has_integral_neg_iff neg_equal_0_iff_equal*)

lemma *integral_diff*: f integrable_on $S \implies g$ integrable_on $S \implies$
 $\text{integral } S (\lambda x. f x - g x) = \text{integral } S f - \text{integral } S g$
by (rule *integral_unique*) (metis *integrable_integral_has_integral_diff*)

lemma *integrable_0*: $(\lambda x. 0)$ integrable_on S
unfolding *integrable_on_def* **using** *has_integral_0* **by** *auto*

lemma *integrable_add*: f integrable_on $S \implies g$ integrable_on $S \implies (\lambda x. f x + g x)$ integrable_on S
unfolding *integrable_on_def* **by**(*auto intro: has_integral_add*)

lemma *integrable_cmul*: f integrable_on $S \implies (\lambda x. c *_{\mathbb{R}} f(x))$ integrable_on S
unfolding *integrable_on_def* **by**(*auto intro: has_integral_cmul*)

lemma *integrable_on_scaleR_iff* [*simp*]:
fixes $c :: \text{real}$
assumes $c \neq 0$
shows $(\lambda x. c *_{\mathbb{R}} f x)$ integrable_on $S \iff f$ integrable_on S
using *integrable_cmul*[of $\lambda x. c *_{\mathbb{R}} f x S 1 / c$] *integrable_cmul*[of $f S c$] $\langle c \neq 0 \rangle$
by *auto*

lemma *integrable_on_cmult_iff* [*simp*]:
fixes $c :: \text{real}$
assumes $c \neq 0$
shows $(\lambda x. c * f x)$ integrable_on $S \iff f$ integrable_on S
using *integrable_on_scaleR_iff* [of $c f$] *assms* **by** *simp*

lemma *integrable_on_cmult_left*:
assumes f *integrable_on* S
shows $(\lambda x. \text{of_real } c * f x)$ *integrable_on* S
using *integrable_cmul*[*of* S *of_real* c] *assms*
by (*simp add: scaleR_conv_of_real*)

lemma *integrable_neg*: f *integrable_on* $S \implies (\lambda x. -f(x))$ *integrable_on* S
unfolding *integrable_on_def* **by**(*auto intro: has_integral_neg*)

lemma *integrable_neg_iff*: $(\lambda x. -f(x))$ *integrable_on* $S \iff f$ *integrable_on* S
using *integrable_neg* **by** *fastforce*

lemma *integrable_diff*:
 f *integrable_on* $S \implies g$ *integrable_on* $S \implies (\lambda x. f x - g x)$ *integrable_on* S
unfolding *integrable_on_def* **by**(*auto intro: has_integral_diff*)

lemma *integrable_linear*:
 f *integrable_on* $S \implies$ *bounded_linear* $h \implies (h \circ f)$ *integrable_on* S
unfolding *integrable_on_def* **by**(*auto intro: has_integral_linear*)

lemma *integral_linear*:
 f *integrable_on* $S \implies$ *bounded_linear* $h \implies \text{integral } S (h \circ f) = h (\text{integral } S f)$
by (*meson has_integral_iff has_integral_linear*)

lemma *integrable_on_cnj_iff*:
 $(\lambda x. \text{cnj } (f x))$ *integrable_on* $A \iff f$ *integrable_on* A
using *integrable_linear*[*OF* *bounded_linear_cnj*, *of* f A]
integrable_linear[*OF* *bounded_linear_cnj*, *of* $\text{cnj} \circ f$ A]
by (*auto simp: o_def*)

lemma *integral_cnj*: $\text{cnj } (\text{integral } A f) = \text{integral } A (\lambda x. \text{cnj } (f x))$
by (*cases f integrable_on A*)
(simp_all add: integral_linear[OF bounded_linear_cnj, symmetric]
o_def integrable_on_cnj_iff not_integrable_integral)

lemma *has_integral_cnj*: $(\text{cnj} \circ f)$ *has_integral* $(\text{cnj } I)$ $A = (f)$ *has_integral* I A
unfolding *has_integral_iff comp_def*
by (*metis integral_cnj complex_cnj_cancel_iff integrable_on_cnj_iff*)

lemma *integral_component_eq*[*simp*]:
fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'm::\text{euclidean_space}$
assumes f *integrable_on* S
shows $\text{integral } S (\lambda x. f x \cdot k) = \text{integral } S f \cdot k$
unfolding *integral_linear*[*OF* *assms*(1) *bounded_linear_inner_left*, *unfolded* *o_def*]
..

lemma *has_integral_sum*:
assumes *finite* T

and $\bigwedge a. a \in T \implies ((f a) \text{ has_integral } (i a)) S$
shows $((\lambda x. \text{sum } (\lambda a. f a x) T) \text{ has_integral } (\text{sum } i T)) S$
using $\langle \text{finite } T \rangle \text{ subset_refl[of } T]$
by $(\text{induct rule: finite_subset_induct})$ $(\text{use assms in } \langle \text{auto simp: has_integral_add} \rangle)$

lemma *integral_sum*:

$\llbracket \text{finite } I; \bigwedge a. a \in I \implies f a \text{ integrable_on } S \rrbracket \implies$
 $\text{integral } S (\lambda x. \sum a \in I. f a x) = (\sum a \in I. \text{integral } S (f a))$
by $(\text{simp add: has_integral_sum integrable_integral integral_unique})$

lemma *integrable_sum*:

$\llbracket \text{finite } I; \bigwedge a. a \in I \implies f a \text{ integrable_on } S \rrbracket \implies (\lambda x. \sum a \in I. f a x) \text{ integrable_on } S$
unfolding *integrable_on_def* **using** *has_integral_sum[of I]* **by** *metis*

lemma *has_integral_eq*:

assumes $\bigwedge x. x \in s \implies f x = g x$
and $f: (f \text{ has_integral } k) s$
shows $(g \text{ has_integral } k) s$
using *has_integral_diff[OF f, of $\lambda x. f x - g x 0$]*
using *has_integral_is_0[of s $\lambda x. f x - g x$]*
using *assms*
by *auto*

lemma *integrable_eq*: $\llbracket f \text{ integrable_on } s; \bigwedge x. x \in s \implies f x = g x \rrbracket \implies g \text{ integrable_on } s$

unfolding *integrable_on_def*
using *has_integral_eq[of s f g]* *has_integral_eq* **by** *blast*

lemma *has_integral_cong*:

assumes $\bigwedge x. x \in s \implies f x = g x$
shows $(f \text{ has_integral } i) s = (g \text{ has_integral } i) s$
by $(\text{metis assms has_integral_eq})$

lemma *integrable_cong*:

assumes $\bigwedge x. x \in A \implies f x = g x$
shows $f \text{ integrable_on } A \longleftrightarrow g \text{ integrable_on } A$
using *has_integral_cong [OF assms]* **by** *fast*

lemma *integral_cong*:

assumes $\bigwedge x. x \in s \implies f x = g x$
shows $\text{integral } s f = \text{integral } s g$
unfolding *integral_def*

by $(\text{metis } (\text{full_types, opaque_lifting}) \text{ assms has_integral_cong integrable_eq})$

lemma *integrable_on_cmult_left_iff [simp]*:

assumes $c \neq 0$
shows $(\lambda x. \text{of_real } c * f x) \text{ integrable_on } s \longleftrightarrow f \text{ integrable_on } s$
 $(\text{is } ?lhs = ?rhs)$

proof

```

assume ?lhs
then have ( $\lambda x. \text{of\_real } (1 / c) * (\text{of\_real } c * f x)$ ) integrable_on s
  using integrable_cmul[of  $\lambda x. \text{of\_real } c * f x$  s 1 / of_real c]
  by (simp add: scaleR_conv_of_real)
then have ( $\lambda x. (\text{of\_real } (1 / c) * \text{of\_real } c * f x)$ ) integrable_on s
  by (simp add: algebra_simps)
with  $\langle c \neq 0 \rangle$  show ?rhs
  by (metis (no_types, lifting) integrable_eq mult.left_neutral nonzero_divide_eq_eq
of_real_1 of_real_mult)
qed (blast intro: integrable_on_cmult_left)

```

lemma *integrable_on_cmult_right*:

```

fixes  $f :: \_ \Rightarrow 'b :: \{\text{comm\_ring, real\_algebra\_1, real\_normed\_vector}\}$ 
assumes  $f \text{ integrable\_on } s$ 
shows ( $\lambda x. f x * \text{of\_real } c$ ) integrable_on s
using integrable_on_cmult_left [OF assms] by (simp add: mult.commute)

```

lemma *integrable_on_cmult_right_iff* [*simp*]:

```

fixes  $f :: \_ \Rightarrow 'b :: \{\text{comm\_ring, real\_algebra\_1, real\_normed\_vector}\}$ 
assumes  $c \neq 0$ 
shows ( $\lambda x. f x * \text{of\_real } c$ ) integrable_on s  $\longleftrightarrow f \text{ integrable\_on } s$ 
using integrable_on_cmult_left_iff [OF assms] by (simp add: mult.commute)

```

lemma *integrable_on_cdivide_iff* [*simp*]:

```

fixes  $f :: \_ \Rightarrow 'b :: \text{real\_normed\_field}$ 
assumes  $c \neq 0$ 
shows ( $\lambda x. f x / \text{of\_real } c$ ) integrable_on s  $\longleftrightarrow f \text{ integrable\_on } s$ 
by (simp add: divide_inverse assms flip: of_real_inverse)

```

lemma *has_integral_null* [*intro*]: $\text{content}(cbox\ a\ b) = 0 \implies (f \text{ has_integral } 0)$
(*cbox a b*)

```

unfolding has_integral_cbox
using eventually_division_filter_tagged_division[of cbox a b]
by (subst tendsto_cong[where g = \_. 0]) (auto elim: eventually_mono intro: sum_content_null)

```

lemma *has_integral_null_real* [*intro*]: $\text{content } \{a..b::\text{real}\} = 0 \implies (f \text{ has_integral } 0)$
 $\{a..b\}$

```

by (metis box_real(2) has_integral_null)

```

lemma *has_integral_null_eq*[*simp*]: $\text{content}(cbox\ a\ b) = 0 \implies (f \text{ has_integral } i)$
(*cbox a b*) $\longleftrightarrow i = 0$

```

by (auto simp add: has_integral_null dest!: integral_unique)

```

lemma *integral_null* [*simp*]: $\text{content}(cbox\ a\ b) = 0 \implies \text{integral}(cbox\ a\ b)\ f = 0$
by (*metis has_integral_null integral_unique*)

lemma *integrable_on_null* [*intro*]: $\text{content}(cbox\ a\ b) = 0 \implies f \text{ integrable_on}$


```

(cbox a b)
  by (simp add: has_integral_integrable)

lemma has_integral_empty[intro]: (f has_integral 0) {}
  by (meson ex_in_conv has_integral_is_0)

lemma has_integral_empty_eq[simp]: (f has_integral i) {}  $\longleftrightarrow$   $i = 0$ 
  by (auto simp add: has_integral_empty has_integral_unique)

lemma integrable_on_empty[intro]: f integrable_on {}
  unfolding integrable_on_def by auto

lemma integral_empty[simp]: integral {} f = 0
  by blast

lemma has_integral_refl[intro]:
  fixes a :: 'a::euclidean_space
  shows (f has_integral 0) (cbox a a)
    and (f has_integral 0) {a}
proof -
  show (f has_integral 0) (cbox a a)
    by (rule has_integral_null) simp
  then show (f has_integral 0) {a}
    by simp
qed

lemma integrable_on_refl[intro]: f integrable_on cbox a a
  unfolding integrable_on_def by auto

lemma integral_refl [simp]: integral (cbox a a) f = 0
  by auto

lemma integral_singleton [simp]: integral {a} f = 0
  by auto

lemma integral_blinfun_apply:
  assumes f integrable_on s
  shows integral s ( $\lambda x. \text{blinfun\_apply } h (f x)$ ) = blinfun_apply h (integral s f)
  by (subst integral_linear[symmetric, OF assms blinfun.bounded_linear_right])
  (simp add: o_def)

lemma blinfun_apply_integral:
  assumes f integrable_on s
  shows blinfun_apply (integral s f) x = integral s ( $\lambda y. \text{blinfun\_apply } (f y) x$ )
  by (metis (no_types, lifting) assms blinfun.prod_left_rep_eq integral_blinfun_apply
  integral_cong)

lemma has_integral_componentwise_iff:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space

```

shows $(f \text{ has_integral } y) A \longleftrightarrow (\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has_integral } (y \cdot b)) A)$

proof safe

fix $b :: 'b$ **assume** $(f \text{ has_integral } y) A$

from $\text{has_integral_linear}[OF \text{ this}(1) \text{ bounded_linear_inner_left, of } b]$

show $((\lambda x. f x \cdot b) \text{ has_integral } (y \cdot b)) A$ **by** (simp add: o_def)

next

assume $(\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has_integral } (y \cdot b)) A)$

hence $\forall b \in \text{Basis}. (((\lambda x. x *_R b) \circ (\lambda x. f x \cdot b)) \text{ has_integral } ((y \cdot b) *_R b)) A$

by $(\text{intro ballI has_integral_linear}) (\text{simp_all add: bounded_linear_scaleR_left})$

hence $((\lambda x. \sum b \in \text{Basis}. (f x \cdot b) *_R b) \text{ has_integral } (\sum b \in \text{Basis}. (y \cdot b) *_R b)) A$

by

$(\text{intro has_integral_sum}) (\text{simp_all add: o_def})$

thus $(f \text{ has_integral } y) A$ **by** $(\text{simp add: euclidean_representation})$

qed

lemma $\text{has_integral_componentwise}$:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$

shows $(\bigwedge b. b \in \text{Basis} \Longrightarrow ((\lambda x. f x \cdot b) \text{ has_integral } (y \cdot b)) A) \Longrightarrow (f \text{ has_integral } y) A$

by $(\text{subst has_integral_componentwise_iff}) \text{ blast}$

lemma $\text{integrable_componentwise_iff}$:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$

shows $f \text{ integrable_on } A \longleftrightarrow (\forall b \in \text{Basis}. (\lambda x. f x \cdot b) \text{ integrable_on } A)$

proof

assume $f \text{ integrable_on } A$

then obtain y **where** $(f \text{ has_integral } y) A$ **by** $(\text{auto simp: integrable_on_def})$

hence $(\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has_integral } (y \cdot b)) A)$

by $(\text{subst (asm) has_integral_componentwise_iff})$

thus $(\forall b \in \text{Basis}. (\lambda x. f x \cdot b) \text{ integrable_on } A)$ **by** $(\text{auto simp: integrable_on_def})$

next

assume $(\forall b \in \text{Basis}. (\lambda x. f x \cdot b) \text{ integrable_on } A)$

then obtain y **where** $\forall b \in \text{Basis}. ((\lambda x. f x \cdot b) \text{ has_integral } y \cdot b) A$

unfolding integrable_on_def **by** $(\text{subst (asm) bchoice_iff}) \text{ blast}$

hence $\forall b \in \text{Basis}. (((\lambda x. x *_R b) \circ (\lambda x. f x \cdot b)) \text{ has_integral } (y \cdot b) *_R b) A$

by $(\text{intro ballI has_integral_linear}) (\text{simp_all add: bounded_linear_scaleR_left})$

hence $((\lambda x. \sum b \in \text{Basis}. (f x \cdot b) *_R b) \text{ has_integral } (\sum b \in \text{Basis}. y \cdot b *_R b)) A$

by $(\text{intro has_integral_sum}) (\text{simp_all add: o_def})$

thus $f \text{ integrable_on } A$ **by** $(\text{auto simp: integrable_on_def o_def euclidean_representation})$

qed

lemma $\text{integrable_componentwise}$:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$

shows $(\bigwedge b. b \in \text{Basis} \Longrightarrow (\lambda x. f x \cdot b) \text{ integrable_on } A) \Longrightarrow f \text{ integrable_on } A$

by $(\text{subst integrable_componentwise_iff}) \text{ blast}$

lemma $\text{integral_componentwise}$:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$

assumes f integrable_on A
shows $\text{integral } A f = (\sum b \in \text{Basis}. \text{integral } A (\lambda x. (f x \cdot b) *_R b))$
proof –
from *assms* **have** $\text{integrable}: \forall b \in \text{Basis}. (\lambda x. x *_R b) \circ (\lambda x. (f x \cdot b)) \text{ integrable_on } A$
by (*subst (asm) integrable_componentwise_iff, intro integrable_linear ballI*)
(simp_all add: bounded_linear_scaleR_left)
have $\text{integral } A f = \text{integral } A (\lambda x. \sum b \in \text{Basis}. (f x \cdot b) *_R b)$
by (*simp add: euclidean_representation*)
also from *integrable* **have** $\dots = (\sum a \in \text{Basis}. \text{integral } A (\lambda x. (f x \cdot a) *_R a))$
by (*subst integral_sum*) (*simp_all add: o_def*)
finally show *?thesis* .
qed

lemma *integrable_component*:
 $f \text{ integrable_on } A \implies (\lambda x. f x \cdot (y :: 'b :: \text{euclidean_space})) \text{ integrable_on } A$
by (*drule integrable_linear[OF _ bounded_linear_inner_left[of y]]*) (*simp add: o_def*)

7.14.4 Cauchy-type criterion for integrability

proposition *integrable_Cauchy*:
fixes $f :: 'n :: \text{euclidean_space} \Rightarrow 'a :: \{\text{real_normed_vector}, \text{complete_space}\}$
shows $f \text{ integrable_on } \text{cbox } a b \iff$
 $(\forall e > 0. \exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall \mathcal{D}1 \ \mathcal{D}2. \mathcal{D}1 \text{ tagged_division_of } (\text{cbox } a b) \wedge \gamma \text{ fine } \mathcal{D}1 \wedge$
 $\mathcal{D}2 \text{ tagged_division_of } (\text{cbox } a b) \wedge \gamma \text{ fine } \mathcal{D}2 \implies$
 $\text{norm } ((\sum (x,K) \in \mathcal{D}1. \text{content } K *_R f x) - (\sum (x,K) \in \mathcal{D}2. \text{content } K *_R$
 $f x)) < e)$
(is $?I = (\forall e > 0. \exists \gamma. ?P e \gamma)$
proof (*intro iffI allI impI*)
assume $?I$
then obtain y
where $y: \bigwedge e. e > 0 \implies$
 $\exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged_division_of } \text{cbox } a b \wedge \gamma \text{ fine } \mathcal{D} \implies$
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - y) < e)$
by (*auto simp: integrable_on_def has_integral*)
show $\exists \gamma. ?P e \gamma$ **if** $e > 0$ **for** e
proof –
have $e/2 > 0$ **using** *that* **by** *auto*
with y **obtain** γ **where** *gauge* γ
and $\gamma: \bigwedge \mathcal{D}. \mathcal{D} \text{ tagged_division_of } \text{cbox } a b \wedge \gamma \text{ fine } \mathcal{D} \implies$
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - y) < e/2$
by *meson*
show *?thesis*
apply (*rule_tac x= γ in exI, clarsimp simp: ‹gauge γ ›*)
by (*blast intro!: γ dist_triangle_half_l[where $y=y$, unfolded dist_norm]*)
qed

next
assume $\forall e > 0. \exists \gamma. ?P e \gamma$
then have $\forall n :: \text{nat}. \exists \gamma. ?P (1 / (n + 1)) \gamma$
by *auto*
then obtain $\gamma :: \text{nat} \Rightarrow 'n \Rightarrow 'n$ **set where** γ :
 $\wedge m. \text{gauge } (\gamma m)$
 $\wedge m \mathcal{D}1 \mathcal{D}2. \llbracket \mathcal{D}1 \text{ tagged_division_of } \text{cbox } a b;$
 $\gamma m \text{ fine } \mathcal{D}1; \mathcal{D}2 \text{ tagged_division_of } \text{cbox } a b; \gamma m \text{ fine } \mathcal{D}2 \rrbracket$
 $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}1. \text{content } K *_R f x) - (\sum (x, K) \in \mathcal{D}2.$
 $\text{content } K *_R f x))$
 $< 1 / (m + 1)$
by *metis*
have $\text{gauge } (\lambda x. \bigcap \{\gamma i x \mid i. i \in \{0..n\}\})$ **for** n
using γ **by** (*intro gauge_Inter*) *auto*
then have $\forall n. \exists p. p \text{ tagged_division_of } (\text{cbox } a b) \wedge (\lambda x. \bigcap \{\gamma i x \mid i. i \in$
 $\{0..n\}\}) \text{ fine } p$
by (*meson fine_division_exists*)
then obtain p **where** $p: \wedge z. p z \text{ tagged_division_of } \text{cbox } a b$
 $\wedge z. (\lambda x. \bigcap \{\gamma i x \mid i. i \in \{0..z\}\}) \text{ fine } p z$
by *meson*
have $dp: \wedge i n. i \leq n \implies \gamma i \text{ fine } p n$
using p **unfolding** *fine_Inter*
using *atLeastAtMost_iff* **by** *blast*
have *Cauchy* $(\lambda n. \text{sum } (\lambda (x, K). \text{content } K *_R (f x)) (p n))$
proof (*rule CauchyI*)
fix $e :: \text{real}$
assume $0 < e$
then obtain N **where** $N \neq 0$ **and** $N: \text{inverse } (\text{real } N) < e$
using *real_arch_inverse[of e]* **by** *blast*
show $\exists M. \forall m \geq M. \forall n \geq M. \text{norm } ((\sum (x, K) \in p m. \text{content } K *_R f x) -$
 $(\sum (x, K) \in p n. \text{content } K *_R f x)) < e$
proof (*intro exI allI impI*)
fix $m n$
assume $mn: N \leq m N \leq n$
have $\text{norm } ((\sum (x, K) \in p m. \text{content } K *_R f x) - (\sum (x, K) \in p n. \text{content}$
 $K *_R f x)) < 1 / (\text{real } N + 1)$
by (*simp add: p(1) dp mn γ*)
also have $\dots < e$
using $N \langle N \neq 0 \rangle \langle 0 < e \rangle$ **by** (*auto simp: field_simps*)
finally show $\text{norm } ((\sum (x, K) \in p m. \text{content } K *_R f x) - (\sum (x, K) \in p n.$
 $\text{content } K *_R f x)) < e$.
qed
qed
then obtain y **where** $y: \exists no. \forall n \geq no. \text{norm } ((\sum (x, K) \in p n. \text{content } K *_R f$
 $x) - y) < r$ **if** $r > 0$ **for** r
by (*auto simp: convergent_eq_Cauchy[symmetric] dest: LIMSEQ_D*)
show $?l$
unfolding *integrable_on_def has_integral*
proof (*rule_tac x=y in exI, clarify*)

```

fix e :: real
assume e > 0
then have e2: e/2 > 0 by auto
then obtain N1::nat where N1: N1 ≠ 0 inverse (real N1) < e/2
  using real_arch_inverse by blast
obtain N2::nat where N2:  $\bigwedge n. n \geq N2 \implies \text{norm } ((\sum (x,K) \in p \text{ n. content } K *_R f x) - y) < e/2$ 
  using y[OF e2] by metis
show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
  ( $\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
   $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - y) < e$ )
proof (intro exI conjI allI impI)
  show gauge ( $\gamma (N1+N2)$ )
  using  $\gamma$  by auto
  show  $\text{norm } ((\sum (x,K) \in q. \text{content } K *_R f x) - y) < e$ 
  if  $q \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \gamma (N1+N2) \text{ fine } q$  for  $q$ 
  proof (rule norm_triangle_half_r)
    have  $\text{norm } ((\sum (x,K) \in p (N1+N2). \text{content } K *_R f x) - (\sum (x,K) \in q. \text{content } K *_R f x))$ 
      < 1 / (real (N1+N2) + 1)
    by (rule  $\gamma$ ; simp add: dp p that)
    also have ... < e/2
    using N1 < 0 < e> by (auto simp: field_simps intro: less_le_trans)
  finally show  $\text{norm } ((\sum (x,K) \in p (N1+N2). \text{content } K *_R f x) - (\sum (x,K) \in q. \text{content } K *_R f x)) < e/2$  .
  show  $\text{norm } ((\sum (x,K) \in p (N1+N2). \text{content } K *_R f x) - y) < e/2$ 
  using N2 le_add_same_cancel2 by blast
qed
qed
qed
qed

```

7.14.5 Additivity of integral on abutting intervals

lemma tagged_division_split_left_inj_content:

```

assumes  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of } S$ 
  and  $(x1, K1) \in \mathcal{D} (x2, K2) \in \mathcal{D} K1 \neq K2 K1 \cap \{x. x \cdot k \leq c\} = K2 \cap \{x. x \cdot k \leq c\} k \in \text{Basis}$ 
  shows  $\text{content } (K1 \cap \{x. x \cdot k \leq c\}) = 0$ 

```

proof –

```

  from tagged_division_ofD(4)[OF  $\mathcal{D} \langle (x1, K1) \in \mathcal{D} \rangle$ ] obtain a b where  $K1: K1 = \text{cbox } a \ b$ 

```

```
  by auto
```

```
  then have  $\text{interior } (K1 \cap \{x. x \cdot k \leq c\}) = \{\}$ 
```

```
  by (metis tagged_division_split_left_inj assms)
```

```
  then show ?thesis
```

```
  unfolding  $K1 \text{ interval\_split}[OF \langle k \in \text{Basis} \rangle]$  by (auto simp: content_eq_0_interior)
qed

```

lemma *tagged_division_split_right_inj_content*:
assumes \mathcal{D} : \mathcal{D} *tagged_division_of* S
and $(x1, K1) \in \mathcal{D}$ $(x2, K2) \in \mathcal{D}$ $K1 \neq K2$ $K1 \cap \{x. x \cdot k \geq c\} = K2 \cap \{x. x \cdot k \geq c\}$ $k \in \text{Basis}$
shows $\text{content}(K1 \cap \{x. x \cdot k \geq c\}) = 0$
proof –
from *tagged_division_of* \mathcal{D} (4) $[OF \ \mathcal{D} \ \langle(x1, K1) \in \mathcal{D}\rangle]$ **obtain** $a \ b$ **where** $K1$:
 $K1 = \text{cbox } a \ b$
by *auto*
then have $\text{interior}(K1 \cap \{x. c \leq x \cdot k\}) = \{\}$
by (*metis* *tagged_division_split_right_inj_assms*)
then show *?thesis*
unfolding $K1$ *interval_split* $[OF \ \langle k \in \text{Basis}\rangle]$
by (*auto simp: content_eq_0_interior*)
qed

proposition *has_integral_split*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{real_normed_vector}$
assumes fi : (f *has_integral* i) ($\text{cbox } a \ b \cap \{x. x \cdot k \leq c\}$)
and fj : (f *has_integral* j) ($\text{cbox } a \ b \cap \{x. x \cdot k \geq c\}$)
and k : $k \in \text{Basis}$
shows (f *has_integral* $(i + j)$) ($\text{cbox } a \ b$)
unfolding *has_integral*
proof *clarify*
fix $e::\text{real}$
assume $0 < e$
then have $e: e/2 > 0$
by *auto*
obtain $\gamma1$ **where** $\gamma1$: *gauge* $\gamma1$
and $\gamma1\text{norm}$:
 $\bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{ tagged_division_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}; \gamma1 \text{ fine } \mathcal{D} \rrbracket$
 $\implies \text{norm}((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - i) < e/2$
apply (*rule* *has_integralD* $[OF \ fi[\text{unfolded } \text{interval_split}[OF \ k]] \ e]$)
apply (*simp add: interval_split[symmetric] k*)
done
obtain $\gamma2$ **where** $\gamma2$: *gauge* $\gamma2$
and $\gamma2\text{norm}$:
 $\bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{ tagged_division_of } \text{cbox } a \ b \cap \{x. c \leq x \cdot k\}; \gamma2 \text{ fine } \mathcal{D} \rrbracket$
 $\implies \text{norm}((\sum (x, k) \in \mathcal{D}. \text{content } k *_R f x) - j) < e/2$
apply (*rule* *has_integralD* $[OF \ fj[\text{unfolded } \text{interval_split}[OF \ k]] \ e]$)
apply (*simp add: interval_split[symmetric] k*)
done
let $? \gamma = \lambda x. \text{if } x \cdot k = c \text{ then } (\gamma1 \ x \cap \gamma2 \ x) \text{ else } \text{ball } x \ |x \cdot k - c| \cap \gamma1 \ x \cap \gamma2 \ x$
have *gauge* $? \gamma$
using $\gamma1 \ \gamma2$ **unfolding** *gauge_def* **by** *auto*
then show $\exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall \mathcal{D}. \mathcal{D} \text{ tagged_division_of } \text{cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D} \implies$
 $\text{norm}((\sum (x, k) \in \mathcal{D}. \text{content } k *_R f x) - (i + j)) < e)$

```

proof (rule_tac x=? $\gamma$  in exI, safe)
  fix p
  assume p: p tagged_division_of (cbox a b) and ? $\gamma$  fine p
  have ab_eqp: cbox a b =  $\bigcup \{K. \exists x. (x, K) \in p\}$ 
    using p by blast
  have xk_le_c:  $x \cdot k \leq c$  if as:  $(x, K) \in p$  and K:  $K \cap \{x. x \cdot k \leq c\} \neq \{\}$  for x
K
  proof (rule ccontr)
    assume **:  $\neg x \cdot k \leq c$ 
    then have  $K \subseteq \text{ball } x \mid x \cdot k - c \mid$ 
      using <? $\gamma$  fine p> as by (fastforce simp: not_le algebra_simps)
    with K obtain y where  $y \in \text{ball } x \mid x \cdot k - c \mid y \cdot k \leq c$ 
      by blast
    then have  $|x \cdot k - y \cdot k| < |x \cdot k - c|$ 
      using Basis_le_norm[OF k, of x - y]
      by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
    with y show False
      using ** by (auto simp add: field_simps)
  qed
  have xk_ge_c:  $x \cdot k \geq c$  if as:  $(x, K) \in p$  and K:  $K \cap \{x. x \cdot k \geq c\} \neq \{\}$  for x
K
  proof (rule ccontr)
    assume **:  $\neg x \cdot k \geq c$ 
    then have  $K \subseteq \text{ball } x \mid x \cdot k - c \mid$ 
      using <? $\gamma$  fine p> as by (fastforce simp: not_le algebra_simps)
    with K obtain y where  $y \in \text{ball } x \mid x \cdot k - c \mid y \cdot k \geq c$ 
      by blast
    then have  $|x \cdot k - y \cdot k| < |x \cdot k - c|$ 
      using Basis_le_norm[OF k, of x - y]
      by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
    with y show False
      using ** by (auto simp add: field_simps)
  qed
  have fin_finite: finite  $\{(x, f K) \mid x K. (x, K) \in s \wedge P x K\}$ 
  if finite s for s and f :: 'a set  $\Rightarrow$  'a set and P :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  bool
  proof -
    from that have finite  $((\lambda(x, K). (x, f K)) \text{ ` } s)$ 
      by auto
    then show ?thesis
      by (rule rev_finite_subset) auto
  qed
  { fix  $\mathcal{G} :: 'a \text{ set} \Rightarrow 'a \text{ set}$ 
    fix  $i :: 'a \times 'a \text{ set}$ 
    assume  $i \in (\lambda(x, k). (x, \mathcal{G} k)) \text{ ` } p - \{(x, \mathcal{G} k) \mid x k. (x, k) \in p \wedge \mathcal{G} k \neq \{\}\}$ 
    then obtain x K where  $xk: i = (x, \mathcal{G} K) \ (x, K) \in p$ 
       $(x, \mathcal{G} K) \notin \{(x, \mathcal{G} K) \mid x K. (x, K) \in p \wedge \mathcal{G} K \neq \{\}\}$ 
      by auto
    have content  $(\mathcal{G} K) = 0$ 
      using xk using content_empty by auto
  }

```

```

then have  $(\lambda(x,K). \text{content } K *_R f x) i = 0$ 
unfolding xk split_conv by auto
} note [simp] = this
have finite p
using p by blast
let  $?M1 = \{(x, K \cap \{x. x \cdot k \leq c\}) \mid x K. (x,K) \in p \wedge K \cap \{x. x \cdot k \leq c\} \neq \{\}\}$ 
have  $\gamma 1\_fine: \gamma 1 \text{ fine } ?M1$ 
using  $\langle ?\gamma \text{ fine } p \rangle$  by (fastforce simp: fine_def split: if_split_asm)
have  $norm ((\sum_{(x,k) \in ?M1}. \text{content } k *_R f x) - i) < e/2$ 
proof (rule  $\gamma 1norm$  [OF tagged_division_ofI  $\gamma 1\_fine$ ])
show finite ?M1
by (rule fin_finite) (use p in blast)
show  $\bigcup \{k. \exists x. (x, k) \in ?M1\} = \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}$ 
by (auto simp: ab_eqp)

fix x L
assume  $xL: (x, L) \in ?M1$ 
then obtain  $x' L'$  where  $xL': x = x' L = L' \cap \{x. x \cdot k \leq c\}$ 
 $(x', L') \in p \cap \{x. x \cdot k \leq c\} \neq \{\}$ 

by blast
then obtain  $a' b'$  where  $ab': L' = \text{cbox } a' \ b'$ 
using p by blast
show  $x \in L \ L \subseteq \text{cbox } a \ b \cap \{x. x \cdot k \leq c\}$ 
using pk_le_c xL' by auto
show  $\exists a \ b. L = \text{cbox } a \ b$ 
using p xL' ab' by (auto simp add: interval_split[OF k,where c=c])

fix y R
assume  $yR: (y, R) \in ?M1$ 
then obtain  $y' R'$  where  $yR': y = y' R = R' \cap \{x. x \cdot k \leq c\}$ 
 $(y', R') \in p \cap \{x. x \cdot k \leq c\} \neq \{\}$ 

by blast
assume  $as: (x, L) \neq (y, R)$ 
show  $\text{interior } L \cap \text{interior } R = \{\}$ 
proof (cases  $L' = R' \longrightarrow x' = y'$ )
case False
have  $\text{interior } R' = \{\}$ 
by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5))
[OF p]  $xL'(\mathcal{I}) \ yR'(\mathcal{I})$ 
then show ?thesis
using  $yR'$  by simp
next
case True
then have  $L' \neq R'$ 
using as unfolding  $xL' \ yR'$  by auto
have  $\text{interior } L' \cap \text{interior } R' = \{\}$ 
by (metis (no_types) Pair_inject  $\langle L' \neq R' \rangle$  p tagged_division_ofD(5))
 $xL'(\mathcal{I}) \ yR'(\mathcal{I})$ 
then show ?thesis

```



```

    using xL'(2) yR'(2) by auto
  qed
  qed
  moreover
  let ?M2 = {(x,K ∩ {x. x·k ≥ c}) | x K. (x,K) ∈ p ∧ K ∩ {x. x·k ≥ c} ≠ {}}
  have γ2_fine: γ2 fine ?M2
    using ⟨?γ fine p⟩ by (fastforce simp: fine_def split: if_split_asm)
  have norm ((∑ (x, k) ∈ ?M2. content k *R f x) - j) < e/2
  proof (rule γ2norm [OF tagged_division_ofI γ2_fine])
    show finite ?M2
      by (rule fin_finite) (use p in blast)
    show ⋃ {k. ∃ x. (x, k) ∈ ?M2} = cbox a b ∩ {x. x·k ≥ c}
      by (auto simp: ab_eqp)

  fix x L
  assume xL: (x, L) ∈ ?M2
  then obtain x' L' where xL': x = x' L = L' ∩ {x. x · k ≥ c}
    (x', L') ∈ p L' ∩ {x. x · k ≥ c} ≠ {}

    by blast
  then obtain a' b' where ab': L' = cbox a' b'
    using p by blast
  show x ∈ L L ⊆ cbox a b ∩ {x. x · k ≥ c}
    using p xk_ge_c xL' by auto
  show ∃ a b. L = cbox a b
    using p xL' ab' by (auto simp add: interval_split[OF k,where c=c])

  fix y R
  assume yR: (y, R) ∈ ?M2
  then obtain y' R' where yR': y = y' R = R' ∩ {x. x · k ≥ c}
    (y', R') ∈ p R' ∩ {x. x · k ≥ c} ≠ {}

    by blast
  assume as: (x, L) ≠ (y, R)
  show interior L ∩ interior R = {}
  proof (cases L' = R' ⟶ x' = y')
    case False
      have interior R' = {}
        by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5))
    [OF p] xL'(3) yR'(3)
      then show ?thesis
        using yR' by simp
    next
    case True
      then have L' ≠ R'
        using as unfolding xL' yR' by auto
      have interior L' ∩ interior R' = {}
        by (metis (no_types) Pair_inject ⟨L' ≠ R'⟩ p tagged_division_ofD(5))
    xL'(3) yR'(3)
      then show ?thesis
        using xL'(2) yR'(2) by auto
  end

```

qed
qed
ultimately
have $norm \ ((\sum (x,K) \in ?M1. content \ K \ *_R \ f \ x) - i) + ((\sum (x,K) \in ?M2. content \ K \ *_R \ f \ x) - j) < e/2 + e/2$
using $norm_add_less$ **by** $blast$
moreover **have** $((\sum (x,K) \in ?M1. content \ K \ *_R \ f \ x) - i) + ((\sum (x,K) \in ?M2. content \ K \ *_R \ f \ x) - j) = (\sum (x,ka) \in p. content \ ka \ *_R \ f \ x) - (i + j)$
proof $-$
have $eq0: \bigwedge x \ y. x = (0::real) \implies x \ *_R \ (y::'b) = 0$
by $auto$
have $cont_eq: \bigwedge g. (\lambda(x,l). content \ l \ *_R \ f \ x) \circ (\lambda(x,l). (x,g \ l)) = (\lambda(x,l). content \ (g \ l) \ *_R \ f \ x)$
by $auto$
have $*$: $\bigwedge \mathcal{G} :: 'a \ set \implies 'a \ set.$
 $(\sum (x,K) \in \{(x, \mathcal{G} \ K) \mid x \ K. (x,K) \in p \wedge \mathcal{G} \ K \neq \{\}\}. content \ K \ *_R \ f \ x) =$
 $(\sum (x,K) \in (\lambda(x,K). (x, \mathcal{G} \ K)) \ 'p. content \ K \ *_R \ f \ x)$
by $(rule \ sum.mono_neutral_left) \ (auto \ simp: \ \langle finite \ p \rangle)$
have $((\sum (x, k) \in ?M1. content \ k \ *_R \ f \ x) - i) + ((\sum (x, k) \in ?M2. content \ k \ *_R \ f \ x) - j) =$
 $(\sum (x, k) \in ?M1. content \ k \ *_R \ f \ x) + (\sum (x, k) \in ?M2. content \ k \ *_R \ f \ x) - (i + j)$
by $auto$
moreover **have** $\dots = (\sum (x,K) \in p. content \ (K \cap \{x. x \cdot k \leq c\}) \ *_R \ f \ x) + (\sum (x,K) \in p. content \ (K \cap \{x. c \leq x \cdot k\}) \ *_R \ f \ x) - (i + j)$
unfolding $*$
apply $(subst \ (1 \ 2) \ sum.reindex_nontrivial)$
apply $(auto \ intro!: \ k \ p \ eq0 \ tagged_division_split_left_inj_content \ tagged_division_split_right_inj_content \ simp: \ cont_eq \ \langle finite \ p \rangle)$
done
moreover **have** $\bigwedge x. x \in p \implies (\lambda(a,B). content \ (B \cap \{a. a \cdot k \leq c\}) \ *_R \ f \ a) \ x +$
 $(\lambda(a,B). content \ (B \cap \{a. c \leq a \cdot k\}) \ *_R \ f \ a) \ x =$
 $(\lambda(a,B). content \ B \ *_R \ f \ a) \ x$
proof $clarify$
fix $a \ B$
assume $(a, B) \in p$
with p **obtain** $u \ v$ **where** $uv: B = cbox \ u \ v$ **by** $blast$
then **show** $content \ (B \cap \{x. x \cdot k \leq c\}) \ *_R \ f \ a + content \ (B \cap \{x. c \leq x \cdot k\}) \ *_R \ f \ a = content \ B \ *_R \ f \ a$
by $(auto \ simp: \ scaleR_left_distrib \ uv \ content_split[OF \ k,of \ u \ v \ c])$
qed
ultimately **show** $?thesis$
by $(auto \ simp: \ sum.distrib[symmetric])$
qed
ultimately **show** $norm \ ((\sum (x, k) \in p. content \ k \ *_R \ f \ x) - (i + j)) < e$

by auto
qed
qed

7.14.6 A sort of converse, integrability on subintervals

lemma *has_integral_separate_sides*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::real_normed_vector$
 assumes $f: (f \text{ has_integral } i) \text{ (cbox } a \text{ } b)$
 and $e > 0$
 and $k: k \in \text{Basis}$
 obtains d where gauge d
 $\forall p1 \ p2. p1 \text{ tagged_division_of } (cbox \ a \ b \cap \{x. x \cdot k \leq c\}) \wedge d \text{ fine } p1 \wedge$
 $p2 \text{ tagged_division_of } (cbox \ a \ b \cap \{x. x \cdot k \geq c\}) \wedge d \text{ fine } p2 \longrightarrow$
 $norm ((\text{sum } (\lambda(x,k). \text{content } k *_{\mathbb{R}} f \ x) \ p1 + \text{sum } (\lambda(x,k). \text{content } k *_{\mathbb{R}} f \ x)$
 $p2) - i) < e$
 proof -
 obtain γ where $d: \text{gauge } \gamma$
 $\wedge p. \llbracket p \text{ tagged_division_of } cbox \ a \ b; \gamma \text{ fine } p \rrbracket$
 $\implies norm ((\text{sum } (x, k) \in p. \text{content } k *_{\mathbb{R}} f \ x) - i) < e$
 using *has_integralD[OF f ‹e > 0›]* by metis
 { fix $p1 \ p2$
 assume $tdiv1: p1 \text{ tagged_division_of } (cbox \ a \ b) \cap \{x. x \cdot k \leq c\}$ and $\gamma \text{ fine}$
 $p1$
 note $p1 = \text{tagged_division_ofD}[OF \ \text{this}(1)]$
 assume $tdiv2: p2 \text{ tagged_division_of } (cbox \ a \ b) \cap \{x. c \leq x \cdot k\}$ and $\gamma \text{ fine}$
 $p2$
 note $p2 = \text{tagged_division_ofD}[OF \ \text{this}(1)]$
 note $\text{tagged_division_Un_interval}[OF \ tdiv1 \ tdiv2]$
 note $p12 = \text{tagged_division_ofD}[OF \ \text{this}] \ \text{this}$
 { fix $a \ b$
 assume $ab: (a, b) \in p1 \cap p2$
 have $(a, b) \in p1$
 using ab by auto
 obtain $u \ v$ where $uv: b = cbox \ u \ v$
 using $\langle (a, b) \in p1 \rangle \ p1(4)$ by moura
 have $b \subseteq \{x. x \cdot k = c\}$
 using $ab \ p1(3)[of \ a \ b] \ p2(3)[of \ a \ b]$ by fastforce
 moreover
 have $\text{interior } \{x::'a. x \cdot k = c\} = \{\}$
 proof (rule ccontr)
 assume $\neg ?thesis$
 then obtain x where $x: x \in \text{interior } \{x::'a. x \cdot k = c\}$
 by auto
 then obtain ε where $0 < \varepsilon$ and $\varepsilon: ball \ x \ \varepsilon \subseteq \{x. x \cdot k = c\}$
 using *mem_interior* by metis
 have $x: x \cdot k = c$
 using $x \text{ interior_subset}$ by fastforce
 have $*$: $\wedge i. i \in \text{Basis} \implies |(x - (x + (\varepsilon/2) *_{\mathbb{R}} k)) \cdot i| = (\text{if } i = k \text{ then } \varepsilon/2$

else 0)

using $\langle 0 < \varepsilon \rangle k$ **by** (auto simp: inner_simps inner_not_same_Basis)

have $(\sum_{i \in \text{Basis}} |(x - (x + (\varepsilon/2) *_{\mathbb{R}} k)) \cdot i|) =$
 $(\sum_{i \in \text{Basis}} (\text{if } i = k \text{ then } \varepsilon/2 \text{ else } 0))$

using * **by** (blast intro: sum.cong)

also have $\dots < \varepsilon$

by (subst sum.delta) (use $\langle 0 < \varepsilon \rangle$ **in** auto)

finally have $x + (\varepsilon/2) *_{\mathbb{R}} k \in \text{ball } x \ \varepsilon$

unfolding mem_ball dist_norm **by**(rule le_less_trans[OF norm_le_l1])

then have $x + (\varepsilon/2) *_{\mathbb{R}} k \in \{x. x \cdot k = c\}$

using ε **by** auto

then show False

using $\langle 0 < \varepsilon \rangle x k$ **by** (auto simp: inner_simps)

qed

ultimately have content b = 0

unfolding uv_content_eq_0_interior

using interior_mono **by** blast

then have content b $*_{\mathbb{R}}$ f a = 0

by auto

}

then have norm $((\sum_{(x, k) \in p1} \text{content } k *_{\mathbb{R}} f x) + (\sum_{(x, k) \in p2} \text{content } k$
 $*_{\mathbb{R}} f x) - i) =$
 $\text{norm } ((\sum_{(x, k) \in p1 \cup p2} \text{content } k *_{\mathbb{R}} f x) - i)$

by (subst sum.union_inter_neutral) (auto simp: p1 p2)

also have $\dots < e$

using d(2) p12 **by** (simp add: fine_Un k $\langle \gamma \text{ fine } p1 \rangle \langle \gamma \text{ fine } p2 \rangle$)

finally have norm $((\sum_{(x, k) \in p1} \text{content } k *_{\mathbb{R}} f x) + (\sum_{(x, k) \in p2} \text{content}$
 $k *_{\mathbb{R}} f x) - i) < e .$

}

then show ?thesis

using d(1) **that by** auto

qed

lemma integrable_split [intro]:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\{\text{real_normed_vector}, \text{complete_space}\}$

assumes $f: f \text{ integrable_on } \text{cbox } a \ b$

and $k: k \in \text{Basis}$

shows $f \text{ integrable_on } (\text{cbox } a \ b \cap \{x. x \cdot k \leq c\})$ (is ?thesis1)

and $f \text{ integrable_on } (\text{cbox } a \ b \cap \{x. x \cdot k \geq c\})$ (is ?thesis2)

proof –

obtain y **where** $y: (f \text{ has_integral } y) (\text{cbox } a \ b)$

using f **by** blast

define a' **where** $a' = (\sum_{i \in \text{Basis}} (\text{if } i = k \text{ then } \max (a \cdot k) \ c \ \text{else } a \cdot i) *_{\mathbb{R}} i)$

define b' **where** $b' = (\sum_{i \in \text{Basis}} (\text{if } i = k \text{ then } \min (b \cdot k) \ c \ \text{else } b \cdot i) *_{\mathbb{R}} i)$

have $\exists d. \text{gauge } d \wedge$

$(\forall p1 \ p2. p1 \text{ tagged_division_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\} \wedge d \text{ fine } p1 \wedge$

$p2 \text{ tagged_division_of } \text{cbox } a \ b \cap \{x. x \cdot k \leq c\} \wedge d \text{ fine } p2 \longrightarrow$

$\text{norm } ((\sum_{(x, K) \in p1} \text{content } K *_{\mathbb{R}} f x) - (\sum_{(x, K) \in p2} \text{content } K$

$*_{\mathbb{R}} f x)) < e)$

```

  if  $e > 0$  for  $e$ 
  proof -
    have  $e/2 > 0$  using that by auto
    with has_integral_separate_sides[OF  $y$  this  $k$ , of  $c$ ]
    obtain  $d$ 
      where gauge  $d$ 
      and  $d: \bigwedge p1\ p2. \llbracket p1\ \text{tagged\_division\_of}\ cbox\ a\ b\ \cap\ \{x.\ x \cdot k \leq c\};\ d\ \text{fine}\ p1;$ 
       $p2\ \text{tagged\_division\_of}\ cbox\ a\ b\ \cap\ \{x.\ c \leq x \cdot k\};\ d\ \text{fine}\ p2 \rrbracket$ 
       $\implies\ \text{norm}\ ((\sum (x,K) \in p1.\ \text{content}\ K\ *_R\ f\ x) + (\sum (x,K) \in p2.\ \text{content}\ K\ *_R\ f\ x) - y) < e/2$ 
      by metis
    show ?thesis
      proof (rule rule_tac  $x=d$  in exI, clarsimp simp add:  $\langle$ gauge  $d\rangle$ )
        fix  $p1\ p2$ 
        assume as:  $p1\ \text{tagged\_division\_of}\ (cbox\ a\ b) \cap \{x.\ x \cdot k \leq c\}\ d\ \text{fine}\ p1$ 
           $p2\ \text{tagged\_division\_of}\ (cbox\ a\ b) \cap \{x.\ x \cdot k \leq c\}\ d\ \text{fine}\ p2$ 
        show  $\text{norm}\ ((\sum (x, k) \in p1.\ \text{content}\ k\ *_R\ f\ x) - (\sum (x, k) \in p2.\ \text{content}\ k\ *_R\ f\ x)) < e$ 
          proof (rule rule_fine_division_exists[OF  $\langle$ gauge  $d\rangle$ , of  $a'\ b$ ])
            fix  $p$ 
            assume  $p\ \text{tagged\_division\_of}\ cbox\ a'\ b\ d\ \text{fine}\ p$ 
            then show ?thesis
              using as norm_triangle_half_l[OF  $d$ [of  $p1\ p$ ]  $d$ [of  $p2\ p$ ]]
              unfolding interval_split[OF  $k$ ] b'_def[symmetric] a'_def[symmetric]
              by (auto simp add: algebra_simps)
          qed
        qed
      qed
    with  $f$  show ?thesis1
      by (simp add: interval_split[OF  $k$ ] integrable_Cauchy)
    have  $\exists d.\ \text{gauge}\ d \wedge$ 
       $(\forall p1\ p2.\ p1\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x.\ x \cdot k \geq c\} \wedge d\ \text{fine}\ p1 \wedge$ 
         $p2\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x.\ x \cdot k \geq c\} \wedge d\ \text{fine}\ p2 \implies$ 
         $\text{norm}\ ((\sum (x,K) \in p1.\ \text{content}\ K\ *_R\ f\ x) - (\sum (x,K) \in p2.\ \text{content}\ K\ *_R\ f\ x)) < e)$ 
      if  $e > 0$  for  $e$ 
      proof -
        have  $e/2 > 0$  using that by auto
        with has_integral_separate_sides[OF  $y$  this  $k$ , of  $c$ ]
        obtain  $d$ 
          where gauge  $d$ 
          and  $d: \bigwedge p1\ p2. \llbracket p1\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x.\ x \cdot k \leq c\};\ d\ \text{fine}\ p1;$ 
           $p2\ \text{tagged\_division\_of}\ cbox\ a\ b \cap \{x.\ c \leq x \cdot k\};\ d\ \text{fine}\ p2 \rrbracket$ 
           $\implies\ \text{norm}\ ((\sum (x,K) \in p1.\ \text{content}\ K\ *_R\ f\ x) + (\sum (x,K) \in p2.\ \text{content}\ K\ *_R\ f\ x) - y) < e/2$ 
          by metis
        show ?thesis

```

```

proof (rule_tac x=d in exI, clarsimp simp add: ‹gauge d›)
  fix p1 p2
  assume as: p1 tagged_division_of (cbox a b) ∩ {x. x · k ≥ c} d fine p1
             p2 tagged_division_of (cbox a b) ∩ {x. x · k ≥ c} d fine p2
  show norm ((∑ (x, k) ∈ p1. content k *R f x) - (∑ (x, k) ∈ p2. content k *R
f x)) < e
  proof (rule fine_division_exists[OF ‹gauge d›, of a b])
    fix p
    assume p tagged_division_of cbox a b' d fine p
    then show ?thesis
      using as norm_triangle_half_l[OF d[of p p1] d[of p p2]]
      unfolding interval_split[OF k] b'_def[symmetric] a'_def[symmetric]
      by (auto simp add: algebra_simps)
    qed
  qed
qed
with f show ?thesis2
  by (simp add: interval_split[OF k] integrable_Cauchy)
qed

```

lemma operative_integralI:

```

fixes f :: 'a::euclidean_space ⇒ 'b::banach
shows operative (lift_option (+)) (Some 0)
  (λi. if f integrable_on i then Some (integral i f) else None)
proof -
  interpret comm_monoid lift_option plus Some (0::'b)
  by (rule comm_monoid_lift_option)
  (rule add.comm_monoid_axioms)
show ?thesis
proof
  fix a b c
  fix k :: 'a
  assume k: k ∈ Basis
  show (if f integrable_on cbox a b then Some (integral (cbox a b) f) else None)
=
  lift_option (+) (if f integrable_on cbox a b ∩ {x. x · k ≤ c} then Some
(integral (cbox a b ∩ {x. x · k ≤ c}) f) else None)
  (if f integrable_on cbox a b ∩ {x. c ≤ x · k} then Some (integral (cbox a b
∩ {x. c ≤ x · k}) f) else None)
  proof (cases f integrable_on cbox a b)
    case True
      with k show ?thesis
      by (auto simp: integrable_split intro: integral_unique [OF has_integral_split[OF
__ k]])
    next
      case False
      have ¬ (f integrable_on cbox a b ∩ {x. x · k ≤ c}) ∨ ¬ (f integrable_on cbox
a b ∩ {x. c ≤ x · k})
      proof (rule ccontr)

```

```

    assume  $\neg$  ?thesis
  then have  $f$  integrable_on cbox  $a$   $b$ 
    unfolding integrable_on_def
    apply (rule_tac  $x$ =integral (cbox  $a$   $b$   $\cap$   $\{x. x \cdot k \leq c\}$ )  $f$  + integral (cbox
 $a$   $b$   $\cap$   $\{x. x \cdot k \geq c\}$ )  $f$  in exI)
    apply (auto intro: has_integral_split[OF _ _  $k$ ])
    done
  then show False
    using False by auto
qed
then show ?thesis
  using False by auto
qed
next
  fix  $a$   $b$  :: 'a
  assume box  $a$   $b$  = {}
  then show (if  $f$  integrable_on cbox  $a$   $b$  then Some (integral (cbox  $a$   $b$ )  $f$ ) else
None) = Some 0
    using has_integral_null_eq
    by (auto simp: integrable_on_null content_eq_0_interior)
qed
qed

```

7.14.7 Bounds on the norm of Riemann sums and the integral itself

```

lemma dsum_bound:
  assumes  $p$ :  $p$  division_of (cbox  $a$   $b$ )
    and  $norm$   $c \leq e$ 
  shows  $norm$  (sum ( $\lambda l. content$   $l$  *R  $c$ )  $p$ )  $\leq e * content$ (cbox  $a$   $b$ )
proof -
  have sumeq:  $(\sum i \in p. |content$   $i|) = sum$  content  $p$ 
    by simp
  have  $e: 0 \leq e$ 
    using assms(2) norm_ge_zero order_trans by blast
  have  $norm$  (sum ( $\lambda l. content$   $l$  *R  $c$ )  $p$ )  $\leq (\sum i \in p. norm$  (content  $i$  *R  $c$ ))
    using norm_sum by blast
  also have ...  $\leq e * (\sum i \in p. |content$   $i|)$ 
    by (simp add: sum_distrib_left[symmetric] mult.commute assms(2) mult_right_mono
sum_nonneg)
  also have ...  $\leq e * content$  (cbox  $a$   $b$ )
    by (metis additive_content_division  $p$  eq_iff sumeq)
  finally show ?thesis .
qed

```

```

lemma rsum_bound:
  assumes  $p$ :  $p$  tagged_division_of (cbox  $a$   $b$ )
    and  $\forall x \in cbox$   $a$   $b. norm$  ( $f$   $x$ )  $\leq e$ 
  shows  $norm$  (sum ( $\lambda(x,k). content$   $k$  *R  $f$   $x$ )  $p$ )  $\leq e * content$  (cbox  $a$   $b$ )

```

proof (cases cbox a b = {})

 case True **show** ?thesis

 using p **unfolding** True tagged_division_of_trivial **by** auto

 next

 case False

 then have e: $e \geq 0$

 by (meson ex_in_conv assms(2) norm_ge_zero order_trans)

 have sum_le: $\text{sum } (\text{content} \circ \text{snd}) p \leq \text{content } (\text{cbox } a \ b)$

 unfolding additive_content_tagged_division[OF p, symmetric] split_def

 by (auto intro: eq_refl)

 have con: $\bigwedge xk. xk \in p \implies 0 \leq \text{content } (\text{snd } xk)$

 using tagged_division_ofD(4) [OF p] content_pos_le

 by force

 have norm (sum $(\lambda(x,k). \text{content } k *_R f x) p) \leq (\sum i \in p. \text{norm } (\text{case } i \text{ of } (x, k)$

 $\implies \text{content } k *_R f x)$)

 by (rule norm_sum)

 also have ... $\leq e * \text{content } (\text{cbox } a \ b)$

 proof -

 have $\bigwedge xk. xk \in p \implies \text{norm } (f (\text{fst } xk)) \leq e$

 using assms(2) p tag_in_interval **by** force

 moreover have $(\sum i \in p. |\text{content } (\text{snd } i)| * e) \leq e * \text{content } (\text{cbox } a \ b)$

 unfolding sum_distrib_right[symmetric]

 using con sum_le **by** (auto simp: mult.commute intro: mult_left_mono [OF

 - e])

 ultimately show ?thesis

 unfolding split_def norm_scaleR

 by (metis (no_types, lifting) mult_left_mono[OF _ abs_ge_zero] or-

 der_trans[OF sum_mono])

 qed

 finally show ?thesis .

 qed

lemma rsum_diff_bound:

assumes p tagged_division_of (cbox a b)

 and $\forall x \in \text{cbox } a \ b. \text{norm } (f x - g x) \leq e$

 shows $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f x) p - \text{sum } (\lambda(x,k). \text{content } k *_R g x)$

 $p) \leq$

 $e * \text{content } (\text{cbox } a \ b)$

 using order_trans[OF _ rsum_bound[OF assms]]

 by (simp add: split_def scaleR_diff_right sum_subtractf eq_refl)

lemma has_integral_bound:

fixes f :: 'a::euclidean_space \implies 'b::real_normed_vector

 assumes $0 \leq B$

 and f: (f has_integral i) (cbox a b)

 and $\bigwedge x. x \in \text{cbox } a \ b \implies \text{norm } (f x) \leq B$

 shows $\text{norm } i \leq B * \text{content } (\text{cbox } a \ b)$

proof (rule ccontr)

 assume \neg ?thesis


```

then have norm i - B * content (cbox a b) > 0
  by auto
with f[unfolded has_integral]
obtain  $\gamma$  where gauge  $\gamma$  and  $\gamma$ :
   $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma \text{ fine } p \rrbracket$ 
     $\implies \text{norm } ((\sum (x, K) \in p. \text{content } K *_{\mathbb{R}} f x) - i) < \text{norm } i - B * \text{content}$ 
(cbox a b)
  by metis
then obtain  $p$  where  $p$ :  $p \text{ tagged\_division\_of } \text{cbox } a \text{ } b$  and  $\gamma \text{ fine } p$ 
  using fine_division_exists by blast
have  $\bigwedge s B. \text{norm } s \leq B \implies \neg \text{norm } (s - i) < \text{norm } i - B$ 
  unfolding not_less
  by (metis diff_left_mono dist_commute dist_norm norm_triangle_ineq2 order_trans)
then show False
  using  $\gamma$  [OF  $p$   $\langle \gamma \text{ fine } p \rangle$ ] rsum_bound[OF  $p$ ] assms by metis
qed

```

corollary integrable_bound:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$ 
assumes  $0 \leq B$ 
  and  $f \text{ integrable\_on } (\text{cbox } a \text{ } b)$ 
  and  $\bigwedge x. x \in \text{cbox } a \text{ } b \implies \text{norm } (f x) \leq B$ 
shows  $\text{norm } (\text{integral } (\text{cbox } a \text{ } b) f) \leq B * \text{content } (\text{cbox } a \text{ } b)$ 
by (metis integrable_integral has_integral_bound assms)

```

7.14.8 Similar theorems about relationship among components

lemma rsum_component_le:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
assumes  $p$ :  $p \text{ tagged\_division\_of } (\text{cbox } a \text{ } b)$ 
  and  $\bigwedge x. x \in \text{cbox } a \text{ } b \implies (f x) \cdot i \leq (g x) \cdot i$ 
shows  $(\sum (x, K) \in p. \text{content } K *_{\mathbb{R}} f x) \cdot i \leq (\sum (x, K) \in p. \text{content } K *_{\mathbb{R}} g x)$ 
. i
unfolding inner_sum_left
proof (rule sum_mono, clarify)
  fix  $x K$ 
  assume  $ab$ :  $(x, K) \in p$ 
  with  $p$  obtain  $u v$  where  $K$ :  $K = \text{cbox } u \text{ } v$ 
  by blast
  then show  $(\text{content } K *_{\mathbb{R}} f x) \cdot i \leq (\text{content } K *_{\mathbb{R}} g x) \cdot i$ 
  by (metis  $ab$  assms inner_scaleR_left measure_nonneg mult_left_mono tag_in_interval)
qed

```

lemma has_integral_component_le:

```

fixes  $f g :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
assumes  $k$ :  $k \in \text{Basis}$ 
assumes  $(f \text{ has\_integral } i) S (g \text{ has\_integral } j) S$ 

```

```

    and f_le_g:  $\bigwedge x. x \in S \implies (f x) \cdot k \leq (g x) \cdot k$ 
  shows  $i \cdot k \leq j \cdot k$ 
proof -
  have ik_le_jk:  $i \cdot k \leq j \cdot k$ 
    if f_i: (f has_integral i) (cbox a b)
    and g_j: (g has_integral j) (cbox a b)
    and le:  $\forall x \in \text{cbox } a \text{ } b. (f x) \cdot k \leq (g x) \cdot k$ 
    for a b i and j :: 'b and f g :: 'a  $\implies$  'b
  proof (rule ccontr)
    assume  $\neg$  ?thesis
    then have *:  $0 < (i \cdot k - j \cdot k) / 3$ 
      by auto
    obtain  $\gamma 1$  where gauge  $\gamma 1$ 
      and  $\gamma 1$ :  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma 1 \text{ fine } p \rrbracket$ 
         $\implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - i) < (i \cdot k - j \cdot k) / 3$ 
      using f_i[unfolded has_integral, rule_format, OF *] by fastforce
    obtain  $\gamma 2$  where gauge  $\gamma 2$ 
      and  $\gamma 2$ :  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \text{ } b; \gamma 2 \text{ fine } p \rrbracket$ 
         $\implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R g x) - j) < (i \cdot k - j \cdot k) / 3$ 
      using g_j[unfolded has_integral, rule_format, OF *] by fastforce
    obtain p where p: p tagged_division_of cbox a b and  $\gamma 1$  fine p  $\gamma 2$  fine p
      using fine_division_exists[OF gauge_Int[OF  $\langle$  gauge  $\gamma 1 \rangle$   $\langle$  gauge  $\gamma 2 \rangle$ ], of a
    b] unfolding fine_Int
      by metis
    then have  $|((\sum (x, k) \in p. \text{content } k *_R f x) - i) \cdot k| < (i \cdot k - j \cdot k) / 3$ 
       $|((\sum (x, k) \in p. \text{content } k *_R g x) - j) \cdot k| < (i \cdot k - j \cdot k) / 3$ 
      using le_less_trans[OF Basis_le_norm[OF k]] k  $\gamma 1$   $\gamma 2$  by metis+
    then show False
      unfolding inner_simps
      using rsum_component_le[OF p] le
      by (fastforce simp add: abs_real_def split: if_split_asm)
  qed
  show ?thesis
proof (cases  $\exists a b. S = \text{cbox } a \text{ } b$ )
  case True
    with ik_le_jk assms show ?thesis
      by auto
  next
  case False
    show ?thesis
  proof (rule ccontr)
    assume  $\neg i \cdot k \leq j \cdot k$ 
    then have ij:  $(i \cdot k - j \cdot k) / 3 > 0$ 
      by auto
    obtain B1 where  $0 < B1$ 
      and B1:  $\bigwedge a b. \text{ball } 0 \text{ } B1 \subseteq \text{cbox } a \text{ } b \implies$ 
         $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a \text{ } b) \wedge$ 
         $\text{norm } (z - i) < (i \cdot k - j \cdot k) / 3$ 
      using has_integral_altD[OF _ False ij] assms by blast

```

```

obtain B2 where 0 < B2
  and B2:  $\bigwedge a b. \text{ball } 0 \ B2 \subseteq \text{cbox } a \ b \implies$ 
     $\exists z. ((\lambda x. \text{if } x \in S \text{ then } g \ x \ \text{else } 0) \text{ has\_integral } z) (\text{cbox } a \ b) \wedge$ 
     $\text{norm } (z - j) < (i \cdot k - j \cdot k) / 3$ 
  using has_integral_altD[OF False ij] assms by blast
have bounded (ball 0 B1  $\cup$  ball (0::'a) B2)
  unfolding bounded_Un by(rule conjI bounded_ball)+
from bounded_subset_cbox_symmetric[OF this]
obtain a b::'a where ab: ball 0 B1  $\subseteq$  cbox a b ball 0 B2  $\subseteq$  cbox a b
  by (meson Un_subset_iff)
then obtain w1 w2 where int_w1:  $((\lambda x. \text{if } x \in S \text{ then } f \ x \ \text{else } 0) \text{ has\_integral } w1)$  (cbox a b)
  and norm_w1:  $\text{norm } (w1 - i) < (i \cdot k - j \cdot k) / 3$ 
  and int_w2:  $((\lambda x. \text{if } x \in S \text{ then } g \ x \ \text{else } 0) \text{ has\_integral } w2)$ 
(cbox a b)
  and norm_w2:  $\text{norm } (w2 - j) < (i \cdot k - j \cdot k) / 3$ 
  using B1 B2 by blast
have *:  $\bigwedge w1 \ w2 \ j \ i::\text{real}. |w1 - i| < (i - j) / 3 \implies |w2 - j| < (i - j) / 3$ 
 $\implies w1 \leq w2 \implies \text{False}$ 
  by (simp add: abs_real_def split: if_split_asm)
have  $|w1 - i| \cdot k < (i \cdot k - j \cdot k) / 3$ 
   $|w2 - j| \cdot k < (i \cdot k - j \cdot k) / 3$ 
  using Basis_le_norm k le_less_trans norm_w1 norm_w2 by blast+
moreover
have  $w1 \cdot k \leq w2 \cdot k$ 
  using ik_le_jk int_w1 int_w2 f_le_g by auto
ultimately show False
  unfolding inner_simps by(rule *)
qed
qed
qed

```

```

lemma integral_component_le:
fixes g f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes k  $\in$  Basis
  and f integrable_on S g integrable_on S
  and  $\bigwedge x. x \in S \implies (f \ x) \cdot k \leq (g \ x) \cdot k$ 
shows  $(\text{integral } S \ f) \cdot k \leq (\text{integral } S \ g) \cdot k$ 
using has_integral_component_le assms by blast

```

```

lemma has_integral_component_nonneg:
fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes k  $\in$  Basis
  and (f has_integral i) S
  and  $\bigwedge x. x \in S \implies 0 \leq (f \ x) \cdot k$ 
shows  $0 \leq i \cdot k$ 
by (metis (no_types, lifting) assms euclidean_all_zero_iff has_integral_0 has_integral_component_le)

```

```

lemma integral_component_nonneg:

```

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes k  $\in$  Basis
  and  $\bigwedge x. x \in S \implies 0 \leq (f x) \cdot k$ 
shows  $0 \leq (\text{integral } S f) \cdot k$ 
by (smt (verit, ccfv_threshold) assms eq_integralD euclidean_all_zero_iff has_integral_component_no

```

lemma *has_integral_component_neg*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes k  $\in$  Basis
  and (f has_integral i) S
  and  $\bigwedge x. x \in S \implies (f x) \cdot k \leq 0$ 
shows  $i \cdot k \leq 0$ 
by (metis (no_types, lifting) assms has_integral_0 has_integral_component_le
inner_zero_left)

```

lemma *has_integral_component_lbound*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes (f has_integral i) (cbox a b)
  and  $\forall x \in \text{cbox } a \text{ b}. B \leq f(x) \cdot k$ 
  and k  $\in$  Basis
shows  $B * \text{content } (\text{cbox } a \text{ b}) \leq i \cdot k$ 
using has_integral_component_le[OF assms(3) has_integral_const assms(1), of
 $(\sum i \in \text{Basis}. B *_{\mathbb{R}} i) :: 'b$ ] assms(2-)
by (auto simp add: field_simps)

```

lemma *has_integral_component_ubound*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes (f has_integral i) (cbox a b)
  and  $\forall x \in \text{cbox } a \text{ b}. f x \cdot k \leq B$ 
  and k  $\in$  Basis
shows  $i \cdot k \leq B * \text{content } (\text{cbox } a \text{ b})$ 
using has_integral_component_le[OF assms(3,1) has_integral_const, of  $\sum i \in \text{Basis}.
B *_{\mathbb{R}} i$ ] assms(2-)
by (auto simp add: field_simps)

```

lemma *integral_component_lbound*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes f integrable_on cbox a b
  and  $\forall x \in \text{cbox } a \text{ b}. B \leq f(x) \cdot k$ 
  and k  $\in$  Basis
shows  $B * \text{content } (\text{cbox } a \text{ b}) \leq (\text{integral } (\text{cbox } a \text{ b}) f) \cdot k$ 
using assms has_integral_component_lbound by blast

```

lemma *integral_component_lbound_real*:

```

assumes f integrable_on {a .. b}
  and  $\forall x \in \{a .. b\}. B \leq f(x) \cdot k$ 
  and k  $\in$  Basis
shows  $B * \text{content } \{a .. b\} \leq (\text{integral } \{a .. b\} f) \cdot k$ 
using assms by (metis box_real(2) integral_component_lbound)

```

lemma *integral_component_ubound*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes f *integrable_on* $cbox\ a\ b$
and $\forall x \in cbox\ a\ b. f\ x \cdot k \leq B$
and $k \in Basis$
shows $(integral\ (cbox\ a\ b)\ f) \cdot k \leq B * content\ (cbox\ a\ b)$
using *assms* **has_integral_component_ubound** **by** *blast*

lemma *integral_component_ubound_real*:
fixes $f :: real \Rightarrow 'a::euclidean_space$
assumes f *integrable_on* $\{a..b\}$
and $\forall x \in \{a..b\}. f\ x \cdot k \leq B$
and $k \in Basis$
shows $(integral\ \{a..b\}\ f) \cdot k \leq B * content\ \{a..b\}$
using *assms* **by** (*metis* *box_real(2)* *integral_component_ubound*)

7.14.9 Uniform limit of integrable functions is integrable

lemma *real_arch_invD*:
 $0 < (e::real) \implies (\exists n::nat. n \neq 0 \wedge 0 < inverse\ (real\ n) \wedge inverse\ (real\ n) < e)$
by (*subst(asm)* *real_arch_inverse*)

lemma *integrable_uniform_limit*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$
assumes $\bigwedge e. e > 0 \implies \exists g. (\forall x \in cbox\ a\ b. norm\ (f\ x - g\ x) \leq e) \wedge g$ *integrable_on* $cbox\ a\ b$
shows f *integrable_on* $cbox\ a\ b$
proof (*cases* *content* $(cbox\ a\ b) > 0$)
case *False* **then show** *?thesis*
using *has_integral_null* **by** (*simp* *add: content_lt_nz integrable_on_def*)
next
case *True*
have $1 / (real\ n + 1) > 0$ **for** n
by *auto*
then have $\exists g. (\forall x \in cbox\ a\ b. norm\ (f\ x - g\ x) \leq 1 / (real\ n + 1)) \wedge g$ *integrable_on* $cbox\ a\ b$ **for** n
using *assms* **by** *blast*
then obtain g **where** $g_near_f: \bigwedge n\ x. x \in cbox\ a\ b \implies norm\ (f\ x - g\ n\ x) \leq 1 / (real\ n + 1)$
and $int_g: \bigwedge n. g\ n$ *integrable_on* $cbox\ a\ b$
by *metis*
then obtain h **where** $h: \bigwedge n. (g\ n$ *has_integral* $h\ n)$ $(cbox\ a\ b)$
unfolding *integrable_on_def* **by** *metis*
have *Cauchy* h
unfolding *Cauchy_def*
proof *clarify*
fix $e :: real$

```

assume e>0
then have e/4 / content (cbox a b) > 0
  using True by (auto simp: field_simps)
then obtain M where M ≠ 0 and M: 1 / (real M) < e/4 / content (cbox a
b)
  by (metis inverse_eq_divide real_arch_inverse)
show ∃ M. ∀ m ≥ M. ∀ n ≥ M. dist (h m) (h n) < e
proof (intro exI strip)
  fix m n
  assume m: M ≤ m and n: M ≤ n
  have e/4 > 0 using ⟨e > 0⟩ by auto
  then obtain gm gn where gauge gm gauge gn
    and gm: ⋀ D. D tagged_division_of cbox a b ∧ gm fine D
      ⇒ norm ((∑ (x, K) ∈ D. content K *R g m x) - h m) <
e/4
    and gn: ⋀ D. D tagged_division_of cbox a b ∧ gn fine D ⇒
      norm ((∑ (x, K) ∈ D. content K *R g n x) - h n) < e/4
  using h[unfolded has_integral] by meson
  then obtain D where D: D tagged_division_of cbox a b (λx. gm x ∩ gn x)
fine D
  by (metis (full_types) fine_division_exists gauge_Int)
  have triangle3: norm (i1 - i2) < e
  if no: norm (s2 - s1) ≤ e/2 norm (s1 - i1) < e/4 norm (s2 - i2) < e/4
for s1 s2 i1 and i2::'b
proof -
  have norm (i1 - i2) ≤ norm (i1 - s1) + norm (s1 - s2) + norm (s2 -
i2)
    using norm_triangle_ineq[of i1 - s1 s1 - i2]
    using norm_triangle_ineq[of s1 - s2 s2 - i2]
    by (auto simp: algebra_simps)
  also have ... < e
    using no by (auto simp: algebra_simps norm_minus_commute)
  finally show ?thesis .
qed
have finep: gm fine D gn fine D
  using fine_Int D by auto
have norm_le: norm (g n x - g m x) ≤ 2 / real M if x: x ∈ cbox a b for x
proof -
  have norm (f x - g n x) + norm (f x - g m x) ≤ 1 / (real n + 1) + 1 /
(real m + 1)
    using g_near_f[OF x, of n] g_near_f[OF x, of m] by simp
  also have ... ≤ 1 / (real M) + 1 / (real M)
    using ⟨M ≠ 0⟩ m n by (intro add_mono; force simp: field_split_simps)
  also have ... = 2 / real M
    by auto
  finally show norm (g n x - g m x) ≤ 2 / real M
    using norm_triangle_le[of g n x - f x f x - g m x 2 / real M]
    by (auto simp: algebra_simps simp add: norm_minus_commute)
qed

```

```

    have norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g n x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g m x$ ))  $\leq 2 / \text{real } M * \text{content } (\text{cbox } a b)$ 
      by (blast intro: norm_le_rsum_diff_bound[OF  $\mathcal{D}(1)$ , where  $e=2 / \text{real } M$ ])
    also have ...  $\leq e/2$ 
      using M True
      by (auto simp: field_simps)
    finally have le_e2: norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g n x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g m x$ ))  $\leq e/2$  .
    then show dist (h m) (h n)  $< e$ 
      unfolding dist_norm using gm gn  $\mathcal{D}$  finep by (auto intro!: triangle3)
  qed
qed
then obtain s where s:  $h \longrightarrow s$ 
  using convergent_eq_Cauchy[symmetric] by blast
show ?thesis
  unfolding integrable_on_def has_integral
proof (rule_tac x=s in exI, clarify)
  fix e::real
  assume e:  $0 < e$ 
  then have  $e/3 > 0$  by auto
  then obtain N1 where N1:  $\forall n \geq N1. \text{norm } (h n - s) < e/3$ 
    using LIMSEQ_D [OF s] by metis
  from e True have  $e/3 / \text{content } (\text{cbox } a b) > 0$ 
    by (auto simp: field_simps)
  then obtain N2 :: nat
    where N2  $\neq 0$  and N2:  $1 / (\text{real } N2) < e/3 / \text{content } (\text{cbox } a b)$ 
    by (metis inverse_eq_divide real_arch_inverse)
  obtain g' where gauge g'
    and g':  $\bigwedge \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a b \wedge g' \text{ fine } \mathcal{D} \implies$ 
      norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g (N1 + N2) x$ ) -  $h (N1 + N2)$ )  $< e/3$ 
    by (metis h has_integral  $\langle e/3 > 0 \rangle$ )
  have *: norm (sf - s)  $< e$ 
    if no: norm (sf - sg)  $\leq e/3$  norm(h - s)  $< e/3$  norm (sg - h)  $< e/3$  for
  sf sg h
  proof -
    have norm (sf - s)  $\leq \text{norm } (sf - sg) + \text{norm } (sg - h) + \text{norm } (h - s)$ 
      using norm_triangle_ineq[of sf - sg sg - s]
      using norm_triangle_ineq[of sg - h h - s]
      by (auto simp: algebra_simps)
    also have ...  $< e$ 
      using no by (auto simp: algebra_simps norm_minus_commute)
    finally show ?thesis .
  qed
{ fix  $\mathcal{D}$ 
  assume ptag:  $\mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a b)$  and g' fine  $\mathcal{D}$ 
  then have norm_less: norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g (N1 + N2) x$ ) -  $h (N1 + N2)$ )  $< e/3$ 
    using g' by blast

```

```

have content (cbox a b) < e/3 * (of_nat N2)
  using <N2 ≠ 0> N2 using True by (auto simp: field_split_simps)
moreover have e/3 * of_nat N2 ≤ e/3 * (of_nat (N1 + N2) + 1)
  using <e>0> by auto
ultimately have content (cbox a b) < e/3 * (of_nat (N1 + N2) + 1)
  by linarith
then have le_e3: 1 / (real (N1 + N2) + 1) * content (cbox a b) ≤ e/3
  unfolding inverse_eq_divide
  by (auto simp: field_simps)
have ne3: norm (h (N1 + N2) - s) < e/3
  using N1 by auto
have norm ((∑ (x,K) ∈ D. content K *_R f x) - (∑ (x,K) ∈ D. content K
*_R g (N1 + N2) x))
  ≤ 1 / (real (N1 + N2) + 1) * content (cbox a b)
  by (blast intro: g_near_f rsum_diff_bound[OF ptag])
then have norm ((∑ (x,K) ∈ D. content K *_R f x) - s) < e
  by (rule *[OF order_trans [OF _ le_e3] ne3 norm_less])
}
then show ∃ d. gauge d ∧
  (∀ D. D tagged_division_of cbox a b ∧ d fine D → norm ((∑ (x,K) ∈
D. content K *_R f x) - s) < e)
  by (blast intro: g' <gauge g'>)
qed
qed

```

lemmas integrable_uniform_limit_real = integrable_uniform_limit [where 'a=real, simplified]

7.14.10 Negligible sets

definition negligible (s :: 'a::euclidean_space set) ←→
 (∀ a b. ((indicator s :: 'a⇒real) has_integral 0) (cbox a b))

Negligibility of hyperplane

```

lemma content_doublesplit:
  fixes a :: 'a::euclidean_space
  assumes 0 < e
  and k: k ∈ Basis
  obtains d where 0 < d and content (cbox a b ∩ {x. |x·k - c| ≤ d}) < e
proof cases
  assume *: a · k ≤ c ∧ c ≤ b · k ∧ (∀ j ∈ Basis. a · j ≤ b · j)
  define a' where a' d = (∑ j ∈ Basis. (if j = k then max (a·j) (c - d) else a·j)
*_R j) for d
  define b' where b' d = (∑ j ∈ Basis. (if j = k then min (b·j) (c + d) else b·j)
*_R j) for d

  have ((λ d. ∏ j ∈ Basis. (b' d - a' d) · j) → (∏ j ∈ Basis. (b' 0 - a' 0) · j))
  (at_right 0)
  by (auto simp: b'_def a'_def intro!: tendsto_min tendsto_max tendsto_eq_intros)

```



```

also have ( $\prod_{j \in \text{Basis}} (b' \ 0 - a' \ 0) \cdot j = 0$ )
  using  $k \ *$ 
  by (intro prod_zero bexI[OF _ k])
  (auto simp: b'_def a'_def inner_diff inner_sum_left inner_not_same_Basis
intro!: sum.cong)
also have ( $(\lambda d. \prod_{j \in \text{Basis}} (b' \ d - a' \ d) \cdot j \longrightarrow 0)$  (at_right 0) =
  ( $(\lambda d. \text{content } (cbox \ a \ b \cap \{x. |x \cdot k - c| \leq d\}) \longrightarrow 0)$  (at_right 0))
proof (intro tendsto_cong eventually_at_rightI)
  fix  $d :: \text{real}$  assume  $d: d \in \{0 <..< 1\}$ 
  have  $cbox \ a \ b \cap \{x. |x \cdot k - c| \leq d\} = cbox \ (a' \ d) \ (b' \ d)$  for  $d$ 
  using  $*$   $d \ k$  by (auto simp add: cbox_def set_eq_iff Int_def ball_conj_distrib
abs_diff_le_iff a'_def b'_def)
  moreover have  $j \in \text{Basis} \implies a' \ d \cdot j \leq b' \ d \cdot j$  for  $j$ 
  using  $*$   $d \ k$  by (auto simp: a'_def b'_def)
  ultimately show ( $\prod_{j \in \text{Basis}} (b' \ d - a' \ d) \cdot j = \text{content } (cbox \ a \ b \cap \{x. |x \cdot k - c| \leq d\})$ )
    by simp
qed simp
finally have ( $(\lambda d. \text{content } (cbox \ a \ b \cap \{x. |x \cdot k - c| \leq d\}) \longrightarrow 0)$  (at_right 0)) .
  from order_tendstoD(2)[OF this <0 < e>]
  obtain  $d'$  where  $0 < d'$  and  $d': \bigwedge y. y > 0 \implies y < d' \implies \text{content } (cbox \ a \ b \cap \{x. |x \cdot k - c| \leq y\}) < e$ 
  by (subst (asm) eventually_at_right[of _ 1]) auto
  show ?thesis
  by (rule that[of d'/2], insert <0 < d'> d'[of d'/2], auto)
next
assume  $*$ :  $\neg (a \cdot k \leq c \wedge c \leq b \cdot k \wedge (\forall j \in \text{Basis}. a \cdot j \leq b \cdot j))$ 
then have  $(\exists j \in \text{Basis}. b \cdot j < a \cdot j) \vee (c < a \cdot k \vee b \cdot k < c)$ 
  by (auto simp: not_le)
show thesis
proof cases
  assume  $\exists j \in \text{Basis}. b \cdot j < a \cdot j$ 
  then have [simp]:  $cbox \ a \ b = \{\}$ 
  using box_ne_empty(1)[of a b] by auto
  show ?thesis
  by (rule that[of 1]) (simp_all add: <0 < e>)
next
assume  $\neg (\exists j \in \text{Basis}. b \cdot j < a \cdot j)$ 
with  $*$  have  $c < a \cdot k \vee b \cdot k < c$ 
  by auto
then show thesis
proof
assume  $c: c < a \cdot k$ 
moreover have  $x \in cbox \ a \ b \implies c \leq x \cdot k$  for  $x$ 
  using  $k \ c$  by (auto simp: cbox_def)
ultimately have  $cbox \ a \ b \cap \{x. |x \cdot k - c| \leq (a \cdot k - c)/2\} = \{\}$ 
  using  $k$  by (auto simp: cbox_def)
with  $<0 < e>$   $c$  that [of (a \cdot k - c)/2] show ?thesis

```

```

    by auto
  next
    assume  $c: b \cdot k < c$ 
    moreover have  $x \in \text{cbox } a \ b \implies x \cdot k \leq c$  for  $x$ 
      using  $k \ c$  by (auto simp: cbox_def)
    ultimately have  $\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq (c - b \cdot k)/2\} = \{\}$ 
      using  $k$  by (auto simp: cbox_def)
    with  $\langle 0 < e \rangle \ c$  that[of  $(c - b \cdot k)/2$ ] show ?thesis
      by auto
  qed
qed
qed

```

proposition *negligible_standard_hyperplane*[intro]:

```

  fixes  $k :: 'a::\text{euclidean\_space}$ 
  assumes  $k: k \in \text{Basis}$ 
  shows negligible  $\{x. x \cdot k = c\}$ 
  unfolding negligible_def has_integral
proof clarsimp
  fix  $a \ b$  and  $e::\text{real}$  assume  $e > 0$ 
  with  $k$  obtain  $d$  where  $0 < d$  and  $d: \text{content } (\text{cbox } a \ b \cap \{x. |x \cdot k - c| \leq d\}) < e$ 
    by (metis content_doublesplit)
  let ?i = indicator  $\{x::'a. x \cdot k = c\} :: 'a \Rightarrow \text{real}$ 
  show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
     $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D} \implies$ 
       $|\sum (x, K) \in \mathcal{D}. \text{content } K * ?i \ x| < e)$ 
  proof (intro exI, safe)
    show gauge  $(\lambda x. \text{ball } x \ d)$ 
      using  $\langle 0 < d \rangle$  by blast
  next
    fix  $\mathcal{D}$ 
    assume  $p: \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b) (\lambda x. \text{ball } x \ d) \text{ fine } \mathcal{D}$ 
    have content  $L = \text{content } (L \cap \{x. |x \cdot k - c| \leq d\})$ 
      if  $(x, L) \in \mathcal{D} \ ?i \ x \neq 0$  for  $x \ L$ 
    proof -
      have  $xk: x \cdot k = c$ 
        using that by (simp add: indicator_def split: if_split_asm)
      have  $L \subseteq \{x. |x \cdot k - c| \leq d\}$ 
    proof
      fix  $y$ 
      assume  $y: y \in L$ 
      have  $L \subseteq \text{ball } x \ d$ 
        using  $p(2)$  that(1) by auto
      then have norm  $(x - y) < d$ 
        by (simp add: dist_norm subset_iff  $y$ )
      then have  $|(x - y) \cdot k| < d$ 
        using  $k$  norm_bound_Basis_lt by blast
    end
  end
end

```

```

    then show  $y \in \{x. |x \cdot k - c| \leq d\}$ 
      unfolding inner_simps xk by auto
    qed
    then show content L = content (L  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d})
      by (metis inf.orderE)
    qed
    then have *:  $(\sum_{(x,K) \in \mathcal{D}} \text{content } K * ?i x) = (\sum_{(x,K) \in \mathcal{D}} \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * ?i x)$ 
      by (force simp add: split_paired_all intro!: sum.cong [OF refl])
    note p' = tagged_division_ofD[OF p(1)] and p'' = division_of_tagged_division[OF p(1)]
    have  $(\sum_{(x,K) \in \mathcal{D}} \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * \text{indicator } \{x. x \cdot k = c\} x) < e$ 
      proof -
        have  $(\sum_{(x,K) \in \mathcal{D}} \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}) * ?i x) \leq (\sum_{(x,K) \in \mathcal{D}} \text{content } (K \cap \{x. |x \cdot k - c| \leq d\}))$ 
          by (force simp add: indicator_def intro!: sum_mono)
        also have ... < e
          proof (subst sum.over_tagged_division_lemma[OF p(1)])
            fix u v :: 'a
            assume box u v = {}
            then have *: content (cbox u v) = 0
              unfolding content_eq_0_interior by simp
            have cbox u v  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d}  $\subseteq$  cbox u v
              by auto
            then have content (cbox u v  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d})  $\leq$  content (cbox u v)
              unfolding interval_doublesplit[OF k] by (rule content_subset)
            then show content (cbox u v  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d}) = 0
              unfolding * interval_doublesplit[OF k]
              by (blast intro: antisym)
          next
            have  $(\sum_{l \in \text{snd } \mathcal{D}} \text{content } (l \cap \{x. |x \cdot k - c| \leq d\})) = \text{sum content } ((\lambda l. l \cap \{x. |x \cdot k - c| \leq d\}) \{l \in \text{snd } \mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}\})$ 
              proof (subst (2) sum.reindex_nontrivial)
                fix x y assume  $x \in \{l \in \text{snd } \mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}\}$   $y \in \{l \in \text{snd } \mathcal{D}. l \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}\}$ 
                   $x \neq y$  and eq:  $x \cap \{x. |x \cdot k - c| \leq d\} = y \cap \{x. |x \cdot k - c| \leq d\}$ 
                  then obtain x' y' where  $(x', x) \in \mathcal{D}$   $x \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}$   $(y', y) \in \mathcal{D}$   $y \cap \{x. |x \cdot k - c| \leq d\} \neq \{\}$ 
                    by (auto)
                  from p'(5)[OF  $\langle(x', x) \in \mathcal{D}\rangle \langle(y', y) \in \mathcal{D}\rangle \langle x \neq y \rangle$ ] have interior (x  $\cap$  y) = {}
                    by auto
                  moreover have interior ((x  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d})  $\cap$  (y  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d}))  $\subseteq$  interior (x  $\cap$  y)
                    by (auto intro: interior_mono)
                  ultimately have interior (x  $\cap$  {x. |x  $\cdot$  k - c|  $\leq$  d}) = {}
                    by (auto simp: eq)
              qed
          qed
      qed

```

```

then show content (x ∩ {x. |x · k - c| ≤ d}) = 0
using p'(4)[OF (x', x) ∈ D] by (auto simp: interval_doublesplit[OF k]
content_eq_0_interior simp del: interior_Int)
qed (insert p'(1), auto intro!: sum_mono_neutral_right)
also have ... ≤ norm (∑ l ∈ (λl. l ∩ {x. |x · k - c| ≤ d}) {l ∈ snd 'D. l ∩
{x. |x · k - c| ≤ d} ≠ {}}). content l *R 1::real)
by simp
also have ... ≤ 1 * content (cbox a b ∩ {x. |x · k - c| ≤ d})
using division_doublesplit[OF p'' k, unfolded interval_doublesplit[OF k]]
unfolding interval_doublesplit[OF k] by (intro dsum_bound) auto
also have ... < e
using d by simp
finally show (∑ K ∈ snd 'D. content (K ∩ {x. |x · k - c| ≤ d})) < e .
qed
finally show (∑ (x, K) ∈ D. content (K ∩ {x. |x · k - c| ≤ d}) * ?i x) < e .
qed
then show |∑ (x, K) ∈ D. content K * ?i x| < e
unfolding * by (simp add: sum_nonneg_split: prod.split)
qed
qed

```

corollary negligible_standard_hyperplane_cart:

```

fixes k :: 'a::finite
shows negligible {x. x$k = (0::real)}
by (simp add: cart_eq_inner_axis negligible_standard_hyperplane)

```

Hence the main theorem about negligible sets

lemma has_integral_negligible_cbox:

```

fixes f :: 'b::euclidean_space ⇒ 'a::real_normed_vector
assumes negs: negligible S
and 0: ⋀x. x ∉ S ⇒ f x = 0
shows (f has_integral 0) (cbox a b)
unfolding has_integral

```

proof clarify

fix e::real

assume e > 0

then have nn_gt0: e/2 / ((real n+1) * (2 ^ n)) > 0 **for** n

by simp

then have ∃γ. gauge γ ∧

(∀D. D tagged_division_of cbox a b ∧ γ fine D ⟶

|∑ (x,K) ∈ D. content K *_R indicator S x|

< e/2 / ((real n + 1) * 2 ^ n) **for** n

using negs [unfolded negligible_def has_integral] **by** auto

then obtain γ **where**

gd: ⋀n. gauge (γ n)

and γ: ⋀n D. [D tagged_division_of cbox a b; γ n fine D]

⟹ |∑ (x,K) ∈ D. content K *_R indicator S x| < e/2 / ((real n + 1) * 2 ^ n)

```

by metis
show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
   $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
     $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - 0) < e)$ 
proof (intro exI, safe)
  show gauge  $(\lambda x. \gamma (\text{nat } \lfloor \text{norm } (f x) \rfloor)) x$ 
  using gd by (auto simp: gauge_def)

show norm  $((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - 0) < e$ 
  if  $\mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b) (\lambda x. \gamma (\text{nat } \lfloor \text{norm } (f x) \rfloor)) x \text{ fine } \mathcal{D}$  for  $\mathcal{D}$ 
proof (cases  $\mathcal{D} = \{\}$ )
  case True with  $\langle 0 < e \rangle$  show ?thesis by simp
next
  case False
  obtain  $N$  where  $\text{Max } ((\lambda(x, K). \text{norm } (f x)) \text{ ' } \mathcal{D}) \leq \text{real } N$ 
  using real_arch_simple by blast
  then have  $N: \bigwedge x. x \in (\lambda(x, K). \text{norm } (f x)) \text{ ' } \mathcal{D} \implies x \leq \text{real } N$ 
  by (meson Max_ge that(1) dual_order.trans finite_imageI tagged_division_of_finite)
  have  $\forall i. \exists q. q \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge (\gamma i) \text{ fine } q \wedge (\forall (x,K) \in \mathcal{D}.$ 
 $K \subseteq (\gamma i) x \longrightarrow (x, K) \in q)$ 
  by (auto intro: tagged_division_finer[OF that(1) gd])
  from choice[OF this]
  obtain  $q$  where  $q: \bigwedge n. q \ n \text{ tagged\_division\_of } \text{cbox } a \ b$ 
   $\bigwedge n. \gamma \ n \text{ fine } q \ n$ 
   $\bigwedge n \ x \ K. \llbracket (x, K) \in \mathcal{D}; K \subseteq \gamma \ n \ x \rrbracket \implies (x, K) \in q \ n$ 
  by fastforce
  have finite  $\mathcal{D}$ 
  using that(1) by blast
  then have sum_le_inc:  $\llbracket \text{finite } T; \bigwedge x \ y. (x,y) \in T \implies (0::\text{real}) \leq g(x,y);$ 
   $\bigwedge y. y \in \mathcal{D} \implies \exists x. (x,y) \in T \wedge f(y) \leq g(x,y) \rrbracket \implies \text{sum } f \ \mathcal{D} \leq$ 
 $\text{sum } g \ T$  for  $f \ g \ T$ 
  by (rule sum_le_included[of  $\mathcal{D} \ T \ g \ \text{snd } f$ ]; force)
  have norm  $(\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) \leq (\sum (x,K) \in \mathcal{D}. \text{norm } (\text{content}$ 
 $K *_R f x))$ 
  unfolding split_def by (rule norm_sum)
  also have  $\dots \leq (\sum (i, j) \in \text{Sigma } \{..N + 1\} q.$ 
   $(\text{real } i + 1) * (\text{case } j \text{ of } (x, K) \Rightarrow \text{content } K *_R \text{indicator } S$ 
 $x))$ 
proof (rule sum_le_inc, safe)
  show finite  $(\text{Sigma } \{..N+1\} q)$ 
  by (meson finite_SigmaI finite_atMost tagged_division_of_finite q(1))
next
  fix  $x \ K$ 
  assume  $xk: (x, K) \in \mathcal{D}$ 
  define  $n$  where  $n = \text{nat } \lfloor \text{norm } (f x) \rfloor$ 
  have  $*$ :  $\text{norm } (f x) \in (\lambda(x, K). \text{norm } (f x)) \text{ ' } \mathcal{D}$ 
  using  $xk$  by auto
  have  $nf x$ :  $\text{real } n \leq \text{norm } (f x) \ \text{norm } (f x) \leq \text{real } n + 1$ 
  unfolding  $n\_def$  by auto

```

```

then have  $n \in \{0..N + 1\}$ 
  using  $N[OF *]$  by auto
moreover have  $K \subseteq \gamma(\text{nat } [norm(f x)]) x$ 
  using  $that(2) xk$  by auto
moreover then have  $(x, K) \in q(\text{nat } [norm(f x)])$ 
  by  $(simp \text{ add: } q(3) xk)$ 
moreover then have  $(x, K) \in q n$ 
  using  $n\_def$  by blast
moreover
have  $norm(\text{content } K *_R f x) \leq (\text{real } n + 1) * (\text{content } K * \text{indicator } S x)$ 
proof (cases  $x \in S$ )
  case False
  then show ?thesis by  $(simp \text{ add: } 0)$ 
next
  case True
  have *:  $\text{content } K \geq 0$ 
    using  $tagged\_division\_ofD(4)[OF \text{ that}(1) xk]$  by auto
  moreover have  $\text{content } K * norm(f x) \leq \text{content } K * (\text{real } n + 1)$ 
    by  $(simp \text{ add: } mult\_left\_mono \text{ nfx}(2))$ 
  ultimately show ?thesis
    using  $nfx \text{ True}$  by  $(auto \text{ simp: } field\_simps)$ 
qed
ultimately show  $\exists y. (y, x, K) \in (\Sigma \{..N + 1\} q) \wedge norm(\text{content } K *_R f x) \leq$ 
   $(\text{real } y + 1) * (\text{content } K *_R \text{indicator } S x)$ 
  by force
qed auto
also have ... =  $(\sum_{i \leq N + 1} \sum_{j \in q i} (\text{real } i + 1) * (\text{case } j \text{ of } (x, K) \Rightarrow \text{content } K *_R \text{indicator } S x))$ 
  using  $q(1)$  by  $(intro \text{ sum\_Sigma\_product } [symmetric]) \text{ auto}$ 
also have ...  $\leq (\sum_{i \leq N + 1} (\text{real } i + 1) * |\sum_{(x,K) \in q i} \text{content } K *_R \text{indicator } S x|)$ 
  by  $(rule \text{ sum\_mono } (simp \text{ add: } sum\_distrib\_left [symmetric]))$ 
also have ...  $\leq (\sum_{i \leq N + 1} e/2/2^i)$ 
proof (rule  $sum\_mono$ )
  show  $(\text{real } i + 1) * |\sum_{(x,K) \in q i} \text{content } K *_R \text{indicator } S x| \leq e/2/2^i$ 
 $\wedge i$ 
    if  $i \in \{..N + 1\}$  for  $i$ 
    using  $\gamma[of \ q \ i \ i] \ q$  by  $(simp \text{ add: } divide\_simps \text{ mult.left\_commute})$ 
qed
also have ... =  $e/2 * (\sum_{i \leq N + 1} (1/2)^i)$ 
  unfolding  $sum\_distrib\_left$  by  $(metis \text{ divide\_inverse } \text{ inverse\_eq\_divide } \text{ power\_one\_over})$ 
also have ...  $< e/2 * 2$ 
proof (rule  $mult\_strict\_left\_mono$ )
  have  $sum(\text{power } (1/2)) \{..N + 1\} = sum(\text{power } (1/2::\text{real})) \{..<N + 2\}$ 
    using  $lessThan\_Suc\_atMost$  by auto
  also have ...  $< 2$ 

```

```

      by (auto simp: geometric_sum)
      finally show sum (power (1/2::real)) {..N + 1} < 2 .
    qed (use <0 < e> in auto)
    finally show ?thesis by auto
  qed
qed
qed

```

proposition *has_integral_negligible*:

```

  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes negs: negligible S
    and  $\bigwedge x. x \in (T - S) \implies f x = 0$ 
  shows (f has_integral 0) T
proof (cases  $\exists a b. T = \text{cbox } a b$ )
  case True
  then have (( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ ) has_integral 0) T
    using assms by (auto intro!: has_integral_negligible_cbox)
  then show ?thesis
    by (rule has_integral_eq [rotated]) auto
  next
  case False
  let ?f = ( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ )
  have (( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ ) has_integral 0) T
    apply (auto simp: False has_integral_alt [of ?f])
    apply (rule_tac x=1 in exI, auto)
    apply (rule_tac x=0 in exI, simp add: has_integral_negligible_cbox [OF negs]
  assms)
  done
  then show ?thesis
    by (rule_tac f=?f in has_integral_eq) auto
qed

```

lemma

```

  assumes negligible S
  shows integrable_negligible: f integrable_on S and integral_negligible: integral S
  f = 0
  using has_integral_negligible [OF assms]
  by (auto simp: has_integral_iff)

```

lemma *has_integral_spike*:

```

  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::real_normed_vector
  assumes negligible S
    and gf:  $\bigwedge x. x \in T - S \implies g x = f x$ 
    and fint: (f has_integral y) T
  shows (g has_integral y) T
proof -
  have *: (g has_integral y) (cbox a b)
    if (f has_integral y) (cbox a b)  $\forall x \in \text{cbox } a b - S. g x = f x$  for a b f and

```

```

g: 'b ⇒ 'a and y
proof -
  have ((λx. f x + (g x - f x)) has_integral (y + 0)) (cbox a b)
    using that by (intro has_integral_add has_integral_negligible) (auto intro!:
  ‹negligible S›)
  then show ?thesis
    by auto
qed
have §: ∧ a b z. [∧ x. x ∈ T ∧ x ∉ S ⇒ g x = f x;
  ((λx. if x ∈ T then f x else 0) has_integral z) (cbox a b)]
  ⇒ ((λx. if x ∈ T then g x else 0) has_integral z) (cbox a b)
  by (auto dest!: *[where f=λx. if x∈T then f x else 0 and g=λx. if x ∈ T
then g x else 0])
show ?thesis
  using fint gf
  apply (subst has_integral_alt)
  apply (subst (asm) has_integral_alt)
  apply (auto split: if_split_asm)
  apply (blast dest: *)
  using § by meson
qed

```

```

lemma has_integral_spike_eq:
  assumes negligible S and ∧ x. x ∈ T - S ⇒ g x = f x
  shows (f has_integral y) T ↔ (g has_integral y) T
  by (metis assms has_integral_spike)

```

```

lemma integrable_spike:
  assumes f integrable_on T negligible S ∧ x. x ∈ T - S ⇒ g x = f x
  shows g integrable_on T
  using assms unfolding integrable_on_def by (blast intro: has_integral_spike)

```

```

lemma integral_spike:
  assumes negligible S
  and ∧ x. x ∈ T - S ⇒ g x = f x
  shows integral T f = integral T g
  using has_integral_spike_eq[OF assms]
  by (auto simp: integral_def integrable_on_def)

```

7.14.11 Some other trivialities about negligible sets

```

lemma negligible_subset:
  assumes negligible s t ⊆ s
  shows negligible t
  unfolding negligible_def
  by (metis (no_types) Diff_iff assms contra_subsetD has_integral_negligible
indicator_simps(2))

```

```

lemma negligible_diff[intro?]:

```



```

assumes negligible s
shows negligible (s - t)
using assms by (meson Diff_subset negligible_subset)

```

```

lemma negligible_Int:
assumes negligible s  $\vee$  negligible t
shows negligible (s  $\cap$  t)
using assms negligible_subset by force

```

```

lemma negligible_Un:
assumes negligible S and T: negligible T
shows negligible (S  $\cup$  T)
proof -
  have (indicat_real (S  $\cup$  T) has_integral 0) (cbox a b)
    if S0: (indicat_real S has_integral 0) (cbox a b)
      and (indicat_real T has_integral 0) (cbox a b) for a b
  proof (subst has_integral_spike_eq[OF T])
    show indicat_real S x = indicat_real (S  $\cup$  T) x if x  $\in$  cbox a b - T for x
    using that by (simp add: indicator_def)
    show (indicat_real S has_integral 0) (cbox a b)
      by (simp add: S0)
  qed
with assms show ?thesis
unfolding negligible_def by blast
qed

```

```

lemma negligible_Un_eq[simp]: negligible (s  $\cup$  t)  $\longleftrightarrow$  negligible s  $\wedge$  negligible t
using negligible_Un negligible_subset by blast

```

```

lemma negligible_sing[intro]: negligible {a::'a::euclidean_space}
using negligible_standard_hyperplane[OF SOME_Basis, of a  $\cdot$  (SOME i. i  $\in$  Basis)] negligible_subset by blast

```

```

lemma negligible_insert[simp]: negligible (insert a s)  $\longleftrightarrow$  negligible s
by (metis insert_is_Un negligible_Un_eq negligible_sing)

```

```

lemma negligible_empty[iff]: negligible {}
using negligible_insert by blast

```

Useful in this form for backchaining

```

lemma empty_imp_negligible: S = {}  $\implies$  negligible S
by simp

```

```

lemma negligible_finite[intro]:
assumes finite s
shows negligible s
using assms by (induct s) auto

```

```

lemma negligible_Union[intro]:

```

assumes *finite* \mathcal{T}
and $\bigwedge t. t \in \mathcal{T} \implies \text{negligible } t$
shows $\text{negligible}(\bigcup \mathcal{T})$
using *assms* **by** *induct auto*

lemma *negligible*: $\text{negligible } S \longleftrightarrow (\forall T. (\text{indicat_real } S \text{ has_integral } 0) T)$

proof (*intro iffI allI*)

fix T

assume *negligible* S

then show (*indicator* S *has_integral* 0) T

by (*meson Diff_iff has_integral_negligible indicator_simps(2)*)

qed (*simp add: negligible_def*)

7.14.12 Finite case of the spike theorem is quite commonly needed

lemma *has_integral_spike_finite*:

assumes *finite* S

and $\bigwedge x. x \in T - S \implies g x = f x$

and (*f has_integral* y) T

shows (*g has_integral* y) T

using *assms* *has_integral_spike negligible_finite* **by** *blast*

lemma *has_integral_spike_finite_eq*:

assumes *finite* S

and $\bigwedge x. x \in T - S \implies g x = f x$

shows (*f has_integral* y) $T \longleftrightarrow$ (*g has_integral* y) T

by (*metis assms has_integral_spike_finite*)

lemma *integrable_spike_finite*:

assumes *finite* S

and $\bigwedge x. x \in T - S \implies g x = f x$

and *f integrable_on* T

shows *g integrable_on* T

using *assms* *has_integral_spike_finite* **by** *blast*

lemma *integrable_spike_finite_eq*:

assumes *finite* S

and $\bigwedge x. x \in T - S \implies f x = g x$

shows *f integrable_on* $T \longleftrightarrow$ *g integrable_on* T

by (*metis assms integrable_spike_finite*)

lemma *has_integral_bound_spike_finite*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{real_normed_vector}$

assumes $0 \leq B$ *finite* S

and f : (*f has_integral* i) (*cbox* a b)

and leB : $\bigwedge x. x \in \text{cbox } a \ b - S \implies \text{norm } (f x) \leq B$

shows $\text{norm } i \leq B * \text{content } (\text{cbox } a \ b)$

proof –

```

define g where  $g \equiv (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f\ x)$ 
then have  $\bigwedge x. x \in \text{cbox } a\ b - S \implies \text{norm } (g\ x) \leq B$ 
  using leB by simp
moreover have (g has_integral i) (cbox a b)
  using has_integral_spike_finite [OF  $\langle \text{finite } S \rangle$  _f]
  by (simp add: g_def)
ultimately show ?thesis
  by (simp add:  $\langle 0 \leq B \rangle$  g_def has_integral_bound)
qed

```

```

corollary has_integral_bound_real:
  fixes f :: real  $\Rightarrow$  'b::real_normed_vector
  assumes  $0 \leq B$  finite S
    and (f has_integral i) {a..b}
    and  $\bigwedge x. x \in \{a..b\} - S \implies \text{norm } (f\ x) \leq B$ 
  shows  $\text{norm } i \leq B * \text{content } \{a..b\}$ 
  by (metis assms box_real(2) has_integral_bound_spike_finite)

```

7.14.13 In particular, the boundary of an interval is negligible

```

lemma negligible_frontier_interval: negligible(cbox (a::'a::euclidean_space) b -
  box a b)
proof -
  let ?A =  $\bigcup ((\lambda k. \{x. x \cdot k = a \cdot k\} \cup \{x::'a. x \cdot k = b \cdot k\}) \text{ ` } \textit{Basis})$ 
  have negligible ?A
    by (force simp add: negligible_Union[OF finite_imageI])
  moreover have cbox a b - box a b  $\subseteq$  ?A
    by (force simp add: mem_box)
  ultimately show ?thesis
    by (rule negligible_subset)
qed

```

```

lemma has_integral_spike_interior:
  assumes f: (f has_integral y) (cbox a b) and gf:  $\bigwedge x. x \in \text{box } a\ b \implies g\ x = f\ x$ 
  shows (g has_integral y) (cbox a b)
  by (meson Diff_iff gf has_integral_spike[OF negligible_frontier_interval _f])

```

```

lemma has_integral_spike_interior_eq:
  assumes  $\bigwedge x. x \in \text{box } a\ b \implies g\ x = f\ x$ 
  shows (f has_integral y) (cbox a b)  $\longleftrightarrow$  (g has_integral y) (cbox a b)
  by (metis assms has_integral_spike_interior)

```

```

lemma integrable_spike_interior:
  assumes  $\bigwedge x. x \in \text{box } a\ b \implies g\ x = f\ x$ 
    and f integrable_on cbox a b
  shows g integrable_on cbox a b
  using assms has_integral_spike_interior_eq by blast

```

7.14.14 Integrability of continuous functions

```

lemma operative_approximableI:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes 0  $\leq$  e
  shows operative conj True ( $\lambda i. \exists g. (\forall x \in i. \text{norm } (f x - g (x::'b)) \leq e) \wedge g$ 
    integrable_on i)
proof -
  interpret comm_monoid conj True
  by standard auto
  show ?thesis
  proof (standard, safe)
    fix a b :: 'b
    show  $\exists g. (\forall x \in \text{cbox } a b. \text{norm } (f x - g x) \leq e) \wedge g$  integrable_on cbox a b
      if box a b = {} for a b
      using assms that
      by (metis content_eq_0_interior integrable_on_null_interior_cbox norm_zero
        right_minus_eq)
      {
        fix c g and k :: 'b
        assume fg:  $\forall x \in \text{cbox } a b. \text{norm } (f x - g x) \leq e$  and g: g integrable_on cbox
          a b
        assume k: k  $\in$  Basis
        show  $\exists g. (\forall x \in \text{cbox } a b \cap \{x. x \cdot k \leq c\}. \text{norm } (f x - g x) \leq e) \wedge g$ 
          integrable_on cbox a b  $\cap \{x. x \cdot k \leq c\}$ 
           $\exists g. (\forall x \in \text{cbox } a b \cap \{x. c \leq x \cdot k\}. \text{norm } (f x - g x) \leq e) \wedge g$  integrable_on
            cbox a b  $\cap \{x. c \leq x \cdot k\}$ 
          using fg g k by auto
        }
      show  $\exists g. (\forall x \in \text{cbox } a b. \text{norm } (f x - g x) \leq e) \wedge g$  integrable_on cbox a b
      if fg1:  $\forall x \in \text{cbox } a b \cap \{x. x \cdot k \leq c\}. \text{norm } (f x - g1 x) \leq e$ 
      and g1: g1 integrable_on cbox a b  $\cap \{x. x \cdot k \leq c\}$ 
      and fg2:  $\forall x \in \text{cbox } a b \cap \{x. c \leq x \cdot k\}. \text{norm } (f x - g2 x) \leq e$ 
      and g2: g2 integrable_on cbox a b  $\cap \{x. c \leq x \cdot k\}$ 
      and k: k  $\in$  Basis
      for c k g1 g2
      proof -
      let ?g =  $\lambda x. \text{if } x \cdot k = c \text{ then } f x \text{ else if } x \cdot k \leq c \text{ then } g1 x \text{ else } g2 x$ 
      show  $\exists g. (\forall x \in \text{cbox } a b. \text{norm } (f x - g x) \leq e) \wedge g$  integrable_on cbox a b
      proof (intro exI conjI ballI)
        show  $\text{norm } (f x - ?g x) \leq e$  if  $x \in \text{cbox } a b$  for x
          by (auto simp: that assms fg1 fg2)
        show ?g integrable_on cbox a b
        proof -
          have ?g integrable_on cbox a b  $\cap \{x. x \cdot k \leq c\}$  ?g integrable_on cbox a
            b  $\cap \{x. x \cdot k \geq c\}$ 
          by (rule integrable_spike[OF _ negligible_standard_hyperplane[of k c]],
            use k g1 g2 in auto)+
          with has_integral_split[OF _ _ k] show ?thesis
          unfolding integrable_on_def by blast
        qed
      qed
    qed
  qed

```

qed
 qed
 qed
 qed
 qed

lemma *comm_monoid_set_F_and*: *comm_monoid_set.F* (\wedge) *True* *f* *s* \longleftrightarrow (*finite* *s* \longrightarrow ($\forall x \in s. f\ x$))

proof –

interpret *bool*: *comm_monoid_set* $\langle (\wedge) \rangle$ *True* ..

show *?thesis*

by (*induction s rule: infinite_finite_induct*) *auto*

qed

lemma *approximable_on_division*:

fixes *f* :: '*b*::*euclidean_space* \Rightarrow '*a*::*banach*

assumes $0 \leq e$

and *d*: *d division_of* (*cbox a b*)

and *f*: $\forall i \in d. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ x) \leq e) \wedge g\ \text{integrable_on } i$

obtains *g* **where** $\forall x \in \text{cbox } a\ b. \text{norm } (f\ x - g\ x) \leq e$ *g integrable_on cbox a b*

proof –

interpret *operative conj True* $\lambda i. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ (x::'b)) \leq e) \wedge g\ \text{integrable_on } i$

using $\langle 0 \leq e \rangle$ **by** (*rule operative_approximableI*)

from *f local.division [OF d]* **that** **show** *thesis*

by *auto*

qed

lemma *integrable_continuous*:

fixes *f* :: '*b*::*euclidean_space* \Rightarrow '*a*::*banach*

assumes *continuous_on* (*cbox a b*) *f*

shows *f integrable_on cbox a b*

proof (*rule integrable_uniform_limit*)

fix *e* :: *real*

assume *e*: $e > 0$

then obtain *d* **where** $0 < d$ **and** *d*: $\forall x\ x'. \llbracket x \in \text{cbox } a\ b; x' \in \text{cbox } a\ b; \text{dist } x'\ x < d \rrbracket \Longrightarrow \text{dist } (f\ x')\ (f\ x) < e$

using *compact_uniformly_continuous[OF assms compact_cbox]* **unfolding** *uniformly_continuous_on_def* **by** *metis*

obtain *p* **where** *ptag*: *p tagged_division_of cbox a b* **and** *finep*: $(\lambda x. \text{ball } x\ d)$ *fine p*

using *fine_division_exists[OF gauge_ball[OF $\langle 0 < d \rangle$], of a b]* .

have $*$: $\forall i \in \text{snd } 'p. \exists g. (\forall x \in i. \text{norm } (f\ x - g\ x) \leq e) \wedge g\ \text{integrable_on } i$

proof (*safe, unfold snd_conv*)

fix *x l*

assume *as*: $(x, l) \in p$

obtain *a b* **where** *l*: $l = \text{cbox } a\ b$

using *as ptag* **by** *blast*

then have *x*: $x \in \text{cbox } a\ b$

```

    using as ptag by auto
    show  $\exists g. (\forall x \in l. \text{norm } (f x - g x) \leq e) \wedge g \text{ integrable\_on } l$ 
    proof (intro exI conjI strip)
      show  $(\lambda y. f x) \text{ integrable\_on } l$ 
      unfolding integrable_on_def l by blast
    next
      fix y
      assume y:  $y \in l$ 
      then have  $y \in \text{ball } x \ d$ 
      using as finep by fastforce
      then show  $\text{norm } (f y - f x) \leq e$ 
      using d x y as l
      by (metis dist_commute dist_norm less_imp_le mem_ball ptag subsetCE
tagged_division_ofD(3))
    qed
  qed
  from e have  $e \geq 0$ 
  by auto
  from approximable_on_division[OF this division_of_tagged_division[OF ptag]
*]
  show  $\exists g. (\forall x \in \text{cbox } a \ b. \text{norm } (f x - g x) \leq e) \wedge g \text{ integrable\_on } \text{cbox } a \ b$ 
  by metis
qed

```

lemma *integrable_continuous_interval*:
fixes $f :: 'b::\text{ordered_euclidean_space} \Rightarrow 'a::\text{banach}$
assumes *continuous_on* $\{a..b\} \ f$
shows $f \text{ integrable_on } \{a..b\}$
by (metis *assms integrable_continuous_interval_cbox*)

lemmas *integrable_continuous_real* = *integrable_continuous_interval*[**where** $'b = \text{real}$]

lemma *integrable_continuous_closed_segment*:
fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
assumes *continuous_on* (closed_segment a b) f
shows $f \text{ integrable_on } (\text{closed_segment } a \ b)$
by (metis *assms closed_segment_eq_real_ivl integrable_continuous_interval*)

7.14.15 Specialization of additivity to one dimension

7.14.16 A useful lemma allowing us to factor out the content size

lemma *has_integral_factor_content*:
 $(f \text{ has_integral } i) (\text{cbox } a \ b) \longleftrightarrow$
 $(\forall e > 0. \exists d. \text{gauge } d \wedge (\forall p. p \text{ tagged_division_of } (\text{cbox } a \ b) \wedge d \text{ fine } p \longrightarrow$
 $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f x) \ p - i) \leq e * \text{content } (\text{cbox } a \ b)))$
proof (cases *content* (cbox a b) = 0)
case True
have $\bigwedge e \ p. p \text{ tagged_division_of } \text{cbox } a \ b \implies \text{norm } ((\sum (x, k) \in p. \text{content } k *_R$

```

f x)) ≤ e * content (cbox a b)
  unfolding sum_content_null[OF True] True by force
  moreover have i = 0
  if  $\bigwedge e. e > 0 \implies \exists d. \text{gauge } d \wedge$ 
    ( $\forall p. p \text{ tagged\_division\_of } \text{cbox } a \ b \wedge$ 
       $d \text{ fine } p \longrightarrow$ 
       $\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - i) \leq e * \text{content } (\text{cbox } a \ b))$ )
  using that [of 1]
  by (force simp add: True sum_content_null[OF True] intro: fine_division_exists[of
_ a b])
  ultimately show ?thesis
  unfolding has_integral_null_eq[OF True]
  by (force simp add: )
next
case False
then have F:  $0 < \text{content } (\text{cbox } a \ b)$ 
  using zero_less_measure_iff by blast
let ?P =  $\lambda e \text{ opp}. \exists d. \text{gauge } d \wedge$ 
  ( $\forall p. p \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge d \text{ fine } p \longrightarrow \text{opp } (\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f x) - i)) e$ )
show ?thesis
proof (subst has_integral, safe)
  fix e :: real
  assume e:  $e > 0$ 
  show ?P (e * content (cbox a b)) (≤) if §[rule_format]:  $\forall \varepsilon > 0. ?P \ \varepsilon \ (<)$ 
  using § [of e * content (cbox a b)]
  by (meson F e less_imp_le mult_pos_pos)
  show ?P e (<) if §[rule_format]:  $\forall \varepsilon > 0. ?P (\varepsilon * \text{content } (\text{cbox } a \ b)) (\leq)$ 
  using § [of e/2 / content (cbox a b)]
  using F e by (force simp add: algebra_simps)
qed
qed

lemma has_integral_factor_content_real:
  (f has_integral i) {a..b::real}  $\longleftrightarrow$ 
  ( $\forall e > 0. \exists d. \text{gauge } d \wedge (\forall p. p \text{ tagged\_division\_of } \{a..b\} \wedge d \text{ fine } p \longrightarrow$ 
     $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f x) p - i) \leq e * \text{content } \{a..b\} )$ )
  unfolding box_real[symmetric]
  by (rule has_integral_factor_content)

```

7.14.17 Fundamental theorem of calculus

```

lemma interval_bounds_real:
  fixes q b :: real
  assumes a ≤ b
  shows Sup {a..b} = b
  and Inf {a..b} = a
  using assms by auto

```

theorem *fundamental_theorem_of_calculus*:
fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
assumes $a \leq b$
and $\text{vecd}: \bigwedge x. x \in \{a..b\} \implies (f \text{ has_vector_derivative } f' x)$ (at x within $\{a..b\}$)
shows $(f' \text{ has_integral } (f b - f a)) \{a..b\}$
unfolding $\text{has_integral_factor_content } \text{box_real}[\text{symmetric}]$
proof *safe*
fix $e :: \text{real}$
assume $e > 0$
then have $\forall x. \exists d > 0. x \in \{a..b\} \longrightarrow$
 $(\forall y \in \{a..b\}. \text{norm } (y-x) < d \longrightarrow \text{norm } (f y - f x - (y-x) *_R f' x) \leq e * \text{norm } (y-x))$
using $\text{vecd}[\text{unfolded } \text{has_vector_derivative_def } \text{has_derivative_within_alt}]$ **by** *blast*
then obtain d **where** $d: \bigwedge x. 0 < d x$
 $\bigwedge x y. \llbracket x \in \{a..b\}; y \in \{a..b\}; \text{norm } (y-x) < d x \rrbracket$
 $\implies \text{norm } (f y - f x - (y-x) *_R f' x) \leq e * \text{norm } (y-x)$
by *metis*
show $\exists d. \text{gauge } d \wedge (\forall p. p \text{ tagged_division_of } (\text{cbox } a b) \wedge d \text{ fine } p \longrightarrow$
 $\text{norm } ((\sum (x, k) \in p. \text{content } k *_R f' x) - (f b - f a)) \leq e * \text{content } (\text{cbox } a b))$
proof (*rule exI, safe*)
show $\text{gauge } (\lambda x. \text{ball } x (d x))$
using $d(1)$ *gauge_ball_dependent* **by** *blast*
next
fix p
assume $\text{ptag}: p \text{ tagged_division_of } \text{cbox } a b$ **and** $\text{finep}: (\lambda x. \text{ball } x (d x)) \text{ fine } p$
have $\text{ba}: b - a = (\sum (x, K) \in p. \text{Sup } K - \text{Inf } K)$ $f b - f a = (\sum (x, K) \in p. f(\text{Sup } K) - f(\text{Inf } K))$
using *additive_tagged_division_1* [**where** $f = \lambda x. x$] *additive_tagged_division_1* [**where** $f = f$]
 $\langle a \leq b \rangle$ *ptag* **by** *auto*
have $\text{norm } (\sum (x, K) \in p. (\text{content } K *_R f' x) - (f(\text{Sup } K) - f(\text{Inf } K)))$
 $\leq (\sum n \in p. e * (\text{case } n \text{ of } (x, k) \Rightarrow \text{Sup } k - \text{Inf } k))$
proof (*rule sum_norm_le, safe*)
fix $x K$
assume $(x, K) \in p$
then have $x \in K$ **and** $\text{kab}: K \subseteq \text{cbox } a b$
using *ptag* **by** *blast+*
then obtain $u v$ **where** $k: K = \text{cbox } u v$ **and** $x \in K$ **and** $\text{kab}: K \subseteq \text{cbox } a b$
using *ptag* $\langle (x, K) \in p \rangle$ **by** *auto*
have $u \leq v$
using $\langle x \in K \rangle$ *unfolding* k **by** *auto*
have $\text{ball}: \forall y \in K. y \in \text{ball } x (d x)$
using *finep* $\langle (x, K) \in p \rangle$ **by** *blast*
have $u \in K v \in K$
by (*simp_all* *add*: $\langle u \leq v \rangle k$)
have $\text{norm } ((v - u) *_R f' x - (f v - f u)) = \text{norm } (f u - f x - (u - x) *_R f' x - (f v - f x - (v - x) *_R f' x))$


```

    by (auto simp add: algebra_simps)
  also have ... ≤ norm (f u - f x - (u - x) *R f' x) + norm (f v - f x - (v
- x) *R f' x)
    by (rule norm_triangle_ineq4)
  finally have norm ((v - u) *R f' x - (f v - f u)) ≤
    norm (f u - f x - (u - x) *R f' x) + norm (f v - f x - (v - x) *R f' x) .
  also have ... ≤ e * norm (u - x) + e * norm (v - x)
  proof (rule add_mono)
    show norm (f u - f x - (u - x) *R f' x) ≤ e * norm (u - x)
    proof (rule d)
      show norm (u - x) < d x
        using ⟨u ∈ K⟩ ball by (auto simp add: dist_real_def)
    qed (use ⟨x ∈ K⟩ ⟨u ∈ K⟩ kab in auto)
    show norm (f v - f x - (v - x) *R f' x) ≤ e * norm (v - x)
    proof (rule d)
      show norm (v - x) < d x
        using ⟨v ∈ K⟩ ball by (auto simp add: dist_real_def)
    qed (use ⟨x ∈ K⟩ ⟨v ∈ K⟩ kab in auto)
  qed
  also have ... ≤ e * (Sup K - Inf K)
    using ⟨x ∈ K⟩ by (auto simp: k interval_bounds_real[OF ⟨u ≤ v⟩]
field_simps)
  finally show norm (content K *R f' x - (f (Sup K) - f (Inf K))) ≤ e *
(Sup K - Inf K)
    using ⟨u ≤ v⟩ by (simp add: k)
  qed
  with ⟨a ≤ b⟩ show norm ((∑ (x, K) ∈ p. content K *R f' x) - (f b - f a)) ≤
e * content (cbox a b)
    by (auto simp: ba split_def sum_subtractf [symmetric] sum_distrib_left)
  qed
qed

```

lemma *has_complex_derivative_imp_has_vector_derivative*:

```

  fixes f :: complex ⇒ complex
  assumes (f has_field_derivative f') (at (of_real a) within (cbox (of_real x)
(of_real y)))
  shows ((f o of_real) has_vector_derivative f') (at a within {x..y})
  using has_derivative_in_compose[of of_real of_real a {x..y} f (*) f'] assms
  by (simp add: scaleR_conv_of_real ac_simps has_vector_derivative_def has_field_derivative_def
o_def cbox_complex_of_real)

```

lemma *ident_has_integral*:

```

  fixes a::real
  assumes a ≤ b
  shows ((λx. x) has_integral (b2 - a2)/2) {a..b}
  proof -
    have ((λx. x) has_integral inverse 2 * b2 - inverse 2 * a2) {a..b}
      unfolding power2_eq_square
      by (rule fundamental_theorem_of_calculus [OF assms] derivative_eq_intros |

```

```

simp)+
  then show ?thesis
    by (simp add: field_simps)
qed

```

```

lemma integral_ident [simp]:
  fixes a::real
  assumes a ≤ b
  shows integral {a..b} (λx. x) = (if a ≤ b then (b2 - a2)/2 else 0)
  by (metis assms ident_has_integral integral_unique)

```

```

lemma ident_integrable_on:
  fixes a::real
  shows (λx. x) integrable_on {a..b}
  using continuous_on_id integrable_continuous_real by blast

```

```

lemma integral_sin [simp]:
  fixes a::real
  assumes a ≤ b shows integral {a..b} sin = cos a - cos b
proof -
  have (sin has_integral (- cos b - - cos a)) {a..b}
  proof (rule fundamental_theorem_of_calculus)
    show ((λa. - cos a) has_vector_derivative sin x) (at x within {a..b}) for x
    unfolding has_real_derivative_iff_has_vector_derivative [symmetric]
    by (rule derivative_eq_intros | force)+
  qed (use assms in auto)
  then show ?thesis
    by (simp add: integral_unique)
qed

```

```

lemma integral_cos [simp]:
  fixes a::real
  assumes a ≤ b shows integral {a..b} cos = sin b - sin a
proof -
  have (cos has_integral (sin b - sin a)) {a..b}
  proof (rule fundamental_theorem_of_calculus)
    show (sin has_vector_derivative cos x) (at x within {a..b}) for x
    unfolding has_real_derivative_iff_has_vector_derivative [symmetric]
    by (rule derivative_eq_intros | force)+
  qed (use assms in auto)
  then show ?thesis
    by (simp add: integral_unique)
qed

```

```

lemma integral_exp [simp]:
  fixes a::real
  assumes a ≤ b shows integral {a..b} exp = exp b - exp a
  by (meson DERIV_exp assms fundamental_theorem_of_calculus has_real_derivative_iff_has_vector_derivative_at_within integral_unique)

```

```

lemma has_integral_sin_nx: (( $\lambda x. \sin(\text{real\_of\_int } n * x)$ ) has_integral 0)  $\{-pi..pi\}$ 
proof (cases  $n = 0$ )
  case False
    have (( $\lambda x. \sin(n * x)$ ) has_integral ( $-\cos(n * pi)/n - -\cos(n * -pi)/n$ ))
 $\{-pi..pi\}$ 
    proof (rule fundamental_theorem_of_calculus)
      show (( $\lambda x. -\cos(n * x) / n$ ) has_vector_derivative  $\sin(n * a)$ ) (at a within
 $\{-pi..pi\}$ )
      if  $a \in \{-pi..pi\}$  for  $a :: \text{real}$ 
      using that False
      unfolding has_vector_derivative_def
      by (intro derivative_eq_intros | force)+
    qed auto
  then show ?thesis
    by simp
qed auto

```

```

lemma integral_sin_nx:
  integral  $\{-pi..pi\}$  ( $\lambda x. \sin(x * \text{real\_of\_int } n)$ ) = 0
  using has_integral_sin_nx [of n] by (force simp: mult.commute)

```

```

lemma has_integral_cos_nx:
  (( $\lambda x. \cos(\text{real\_of\_int } n * x)$ ) has_integral (if  $n = 0$  then  $2 * pi$  else 0))  $\{-pi..pi\}$ 
proof (cases  $n = 0$ )
  case True
    then show ?thesis
      using has_integral_const_real [of 1::real -pi pi] by auto
  next
    case False
      have (( $\lambda x. \cos(n * x)$ ) has_integral ( $\sin(n * pi)/n - \sin(n * -pi)/n$ ))
 $\{-pi..pi\}$ 
      proof (rule fundamental_theorem_of_calculus)
        show (( $\lambda x. \sin(n * x) / n$ ) has_vector_derivative  $\cos(n * x)$ ) (at x within
 $\{-pi..pi\}$ )
        if  $x \in \{-pi..pi\}$ 
        for  $x :: \text{real}$ 
        using that False
        unfolding has_vector_derivative_def
        by (intro derivative_eq_intros | force)+
      qed auto
    with False show ?thesis
      by (simp add: mult.commute)
qed

```

```

lemma integral_cos_nx:
  integral  $\{-pi..pi\}$  ( $\lambda x. \cos(x * \text{real\_of\_int } n)$ ) = (if  $n = 0$  then  $2 * pi$  else 0)
  using has_integral_cos_nx [of n] by (force simp: mult.commute)

```

7.14.18 Taylor series expansion

lemma *mtv_integral*:

fixes $f::'a::real_normed_vector \Rightarrow 'b::banach$

assumes f' [*derivative_intros*]:

$\bigwedge x. x \in S \implies (f \text{ has_derivative } f' \ x) \text{ (at } x \text{ within } S)$

assumes *line_in*: $\bigwedge t. t \in \{0..1\} \implies x + t *_R y \in S$

shows $f(x + y) - f x = \text{integral } \{0..1\} (\lambda t. f' (x + t *_R y) \ y)$

proof –

from *assms* **have** *subset*: $(\lambda x a. x + x a *_R y) \text{ ' } \{0..1\} \subseteq S$ **by** *auto*

note [*derivative_intros*] =

has_derivative_subset[*OF _ subset*]

has_derivative_in_compose[**where** $f = (\lambda x a. x + x a *_R y)$ **and** $g = f$]

note [*continuous_intros*] =

continuous_on_compose2[**where** $f = (\lambda x a. x + x a *_R y)$]

continuous_on_subset[*OF _ subset*]

have $\bigwedge t. t \in \{0..1\} \implies$

$((\lambda t. f(x + t *_R y)) \text{ has_vector_derivative } f' (x + t *_R y) \ y)$

$(\text{at } t \text{ within } \{0..1\})$

using *assms*

by (*auto simp*: *has_vector_derivative_def*

linear_cmul[*OF has_derivative_linear*[*OF f*], *symmetric*]

intro!: *derivative_eq_intros*)

from *fundamental_theorem_of_calculus*[*rule_format*, *OF _ this*]

show *?thesis*

by (*auto intro!*: *integral_unique*[*symmetric*])

qed

lemma (*in_bounded_bilinear*) *sum_prod_derivatives_has_vector_derivative*:

assumes $p > 0$

and $f0$: $Df \ 0 = f$

and Df : $\bigwedge m \ t. m < p \implies a \leq t \implies t \leq b \implies$

$(Df \ m \ \text{has_vector_derivative } Df \ (Suc \ m) \ t) \text{ (at } t \text{ within } \{a..b\})$

and $g0$: $Dg \ 0 = g$

and Dg : $\bigwedge m \ t. m < p \implies a \leq t \implies t \leq b \implies$

$(Dg \ m \ \text{has_vector_derivative } Dg \ (Suc \ m) \ t) \text{ (at } t \text{ within } \{a..b\})$

and *ivl*: $a \leq t \leq b$

shows $((\lambda t. \sum_{i < p} (-1)^{\hat{i}} *_R \text{prod } (Df \ i \ t) \ (Dg \ (p - Suc \ i) \ t))$

has_vector_derivative

$\text{prod } (f \ t) \ (Dg \ p \ t) - (-1)^{\hat{p}} *_R \text{prod } (Df \ p \ t) \ (g \ t))$

$(\text{at } t \text{ within } \{a..b\})$

using *assms*

proof *cases*

assume p : $p \neq 1$

define p' **where** $p' = p - 2$

from *assms* p **have** p' : $\{..<p\} = \{..Suc \ p'\}$ $p = Suc \ (Suc \ p')$

by (*auto simp*: p' _def)

have $*$: $\bigwedge i. i \leq p' \implies Suc \ (Suc \ p' - i) = (Suc \ (Suc \ p') - i)$

by *auto*

let $?f = \lambda i. (-1)^{\hat{i}} *_R (\text{prod } (Df \ i \ t) \ (Dg \ ((p - i)) \ t))$

```

have ( $\sum i < p. (-1) ^ i *_{\mathbb{R}} (\text{prod } (Df\ i\ t)\ (Dg\ (Suc\ (p - Suc\ i))\ t) +$ 
 $\text{prod } (Df\ (Suc\ i)\ t)\ (Dg\ (p - Suc\ i)\ t)) =$ 
 $(\sum i \le (Suc\ p').\ ?f\ i - ?f\ (Suc\ i))$ 
by (auto simp: algebra_simps p'(2) numeral_2_eq_2 * lessThan_Suc_atMost)
also note sum_telescope
finally
have ( $\sum i < p. (-1) ^ i *_{\mathbb{R}} (\text{prod } (Df\ i\ t)\ (Dg\ (Suc\ (p - Suc\ i))\ t) +$ 
 $\text{prod } (Df\ (Suc\ i)\ t)\ (Dg\ (p - Suc\ i)\ t))$ 
=  $\text{prod } (f\ t)\ (Dg\ p\ t) - (-1) ^ p *_{\mathbb{R}} \text{prod } (Df\ p\ t)\ (g\ t)$ 
unfolding p'[symmetric]
by (simp add: assms)
thus ?thesis
using assms
by (auto intro!: derivative_eq_intros has_vector_derivative)
qed (auto intro!: derivative_eq_intros has_vector_derivative)

```

lemma

```

fixes f::real=>'a::banach
assumes p>0
and f0: Df 0 = f
and Df:  $\bigwedge m\ t. m < p \implies a \le t \implies t \le b \implies$ 
 $(Df\ m\ \text{has\_vector\_derivative } Df\ (Suc\ m)\ t)\ (\text{at } t\ \text{within } \{a..b\})$ 
and ivl:  $a \le b$ 
defines i  $\equiv \lambda x. ((b - x) ^ (p - 1) / \text{fact } (p - 1)) *_{\mathbb{R}} Df\ p\ x$ 
shows Taylor_has_integral:
 $(i\ \text{has\_integral } f\ b - (\sum i < p. ((b - a) ^ i / \text{fact } i) *_{\mathbb{R}} Df\ i\ a))\ \{a..b\}$ 
and Taylor_integral:
 $f\ b = (\sum i < p. ((b - a) ^ i / \text{fact } i) *_{\mathbb{R}} Df\ i\ a) + \text{integral } \{a..b\}\ i$ 
and Taylor_integrable:
 $i\ \text{integrable\_on } \{a..b\}$ 
proof goal_cases
case 1
interpret bounded_bilinear scaleR::real=>'a=>'a
by (rule bounded_bilinear_scaleR)
define g where  $g\ s = (b - s) ^ (p - 1) / \text{fact } (p - 1)$  for s
define Dg where [abs_def]:
 $Dg\ n\ s = (\text{if } n < p\ \text{then } (-1) ^ n * (b - s) ^ (p - 1 - n) / \text{fact } (p - 1 - n)$ 
 $\text{else } 0)$  for n s
have g0:  $Dg\ 0 = g$ 
using <p > 0>
by (auto simp add: Dg_def field_split_simps g_def split: if_split_asm)
{
fix m
assume p > Suc m
hence  $p - Suc\ m = Suc\ (p - Suc\ (Suc\ m))$ 
by auto
hence  $\text{real } (p - Suc\ m) * \text{fact } (p - Suc\ (Suc\ m)) = \text{fact } (p - Suc\ m)$ 
by auto
} note fact_eq = this

```

```

have Dg:  $\bigwedge m t. m < p \implies a \leq t \implies t \leq b \implies$ 
  (Dg m has_vector_derivative Dg (Suc m) t) (at t within {a..b})
unfolding Dg_def
by (auto intro!: derivative_eq_intros simp: has_vector_derivative_def fact_eq
field_split_simps)
let ?sum =  $\lambda t. \sum i < p. (-1)^i *_{\mathbb{R}} Dg i t *_{\mathbb{R}} Df (p - Suc i) t$ 
from sum_prod_derivatives_has_vector_derivative[of _ Dg _ _ _ Df,
OF <p > 0> g0 Dg f0 Df]
have deriv:  $\bigwedge t. a \leq t \implies t \leq b \implies$ 
  (?sum has_vector_derivative
  g t *_{\mathbb{R}} Df p t - (-1)^p *_{\mathbb{R}} Dg p t *_{\mathbb{R}} f t) (at t within {a..b})
by auto
from fundamental_theorem_of_calculus[rule_format, OF <a ≤ b> deriv]
have (i has_integral ?sum b - ?sum a) {a..b}
  using atLeastatMost_empty'[simp del]
  by (simp add: i_def g_def Dg_def)
also
have one:  $(-1)^{p'} * (-1)^{p'} = (1::real) \{..<p\} \cap \{i. p = Suc i\} = \{p - 1\}$ 
  for p'
  using <p > 0> by (auto simp: power_mult_distrib)
then have ?sum b = f b
  using Suc_pred'[OF <p > 0>]
  by (simp add: diff_eq_eq Dg_def power_0_left le_Suc_eq if_distrib
if_distribR sum.If_cases f0)
also
have ?sum a =  $(\sum i < p. ((b-a)^i / fact i) *_{\mathbb{R}} Df i a)$ 
proof (rule sum.reindex_cong)
  have  $\bigwedge i. i < p \implies \exists j < p. i = p - Suc j$ 
  by (metis Suc_diff_Suc <p > 0> diff_Suc_less diff_diff_cancel less_or_eq_imp_le)
  then show  $\{..<p\} = (\lambda x. p - x - 1) \text{ ` } \{..<p\}$ 
  by force
qed (auto simp add: inj_on_def Dg_def one)
finally show c: ?case .
case 2 show ?case using c integral_unique
  by (metis (lifting) add commute diff_eq_eq integral_unique)
case 3 show ?case using c by force
qed

```

7.14.19 Only need trivial subintervals if the interval itself is trivial

```

proposition division_of_nontrivial:
  fixes  $\mathcal{D} :: 'a::euclidean\_space \text{ set set}$ 
  assumes sdiv:  $\mathcal{D} \text{ division\_of } (cbox a b)$ 
  and cont0:  $\text{content } (cbox a b) \neq 0$ 
  shows  $\{k. k \in \mathcal{D} \wedge \text{content } k \neq 0\} \text{ division\_of } (cbox a b)$ 
  using sdiv
proof (induction card  $\mathcal{D}$  arbitrary:  $\mathcal{D}$  rule: less_induct)
  case less

```

```

note  $\mathcal{D} = \text{division\_of } \mathcal{D} [OF \text{ less.prem}]$ 
{
  presume *:  $\{k \in \mathcal{D}. \text{content } k \neq 0\} \neq \mathcal{D} \implies ?case$ 
  then show ?case
    using less.prem by fastforce
}
assume noteq:  $\{k \in \mathcal{D}. \text{content } k \neq 0\} \neq \mathcal{D}$ 
then obtain  $K \ c \ d$  where  $K \in \mathcal{D}$  and  $\text{cont } K = 0$  and  $\text{keq}: K =$ 
 $c \text{ box } c \ d$ 
  using  $\mathcal{D}(4)$  by blast
then have  $\text{card } \mathcal{D} > 0$ 
  unfolding  $\text{card\_gt\_0\_iff}$  using less by auto
then have  $\text{card}: \text{card } (\mathcal{D} - \{K\}) < \text{card } \mathcal{D}$ 
  using less  $\langle K \in \mathcal{D} \rangle$  by (simp add:  $\mathcal{D}(1)$ )
have  $\text{closed}: \text{closed } (\bigcup (\mathcal{D} - \{K\}))$ 
  using less.prem by auto
have  $x \text{ islimpt } \bigcup (\mathcal{D} - \{K\})$  if  $x \in K$  for  $x$ 
  unfolding  $\text{islimpt\_approachable}$ 
proof (intro allI impI)
  fix  $e::\text{real}$ 
  assume  $e > 0$ 
  obtain  $i$  where  $i: c \cdot i = d \cdot i$   $i \in \text{Basis}$ 
    using  $\text{cont } \mathcal{D}(3)$  [OF  $\langle K \in \mathcal{D} \rangle$ ] unfolding  $\text{box\_ne\_empty}$   $\text{keq}$ 
    by (meson  $\text{content\_eq\_0}$   $\text{dual\_order.antisym}$ )
  then have  $x_i: x \cdot i = d \cdot i$ 
    using  $\langle x \in K \rangle$  unfolding  $\text{keq mem\_box}$  by (metis  $\text{antisym}$ )
  define  $y$  where  $y = (\sum_{j \in \text{Basis}. \text{if } j = i \text{ then if } c \cdot i \leq (a \cdot i + b \cdot i)/2 \text{ then } c \cdot i$ 
+
 $\text{min } e (b \cdot i - c \cdot i)/2 \text{ else } c \cdot i - \text{min } e (c \cdot i - a \cdot i)/2 \text{ else } x \cdot j) *_{\mathbb{R}} j$ 
  show  $\exists x' \in \bigcup (\mathcal{D} - \{K\}). x' \neq x \wedge \text{dist } x' \ x < e$ 
  proof (intro  $\text{be } I$   $\text{conj } I$ )
    have  $d \in c \text{ box } c \ d$ 
      using  $\mathcal{D}(3)$  [OF  $\langle K \in \mathcal{D} \rangle$ ] by (simp add:  $\text{box\_ne\_empty}(1)$   $\text{keq mem\_box}(2)$ )
    then have  $d \in c \text{ box } a \ b$ 
      using  $\mathcal{D}(2)$  [OF  $\langle K \in \mathcal{D} \rangle$ ] by (auto simp:  $\text{keq}$ )
    then have  $d_i: a \cdot i \leq d \cdot i \wedge d \cdot i \leq b \cdot i$ 
      using  $\langle i \in \text{Basis} \rangle$   $\text{mem\_box}(2)$  by blast
    then have  $x y_i: y \cdot i \neq x \cdot i$ 
      unfolding  $y\_def$   $i$   $x_i$  using  $\langle e > 0 \rangle$   $\text{cont } 0 \langle i \in \text{Basis} \rangle$ 
      by (auto simp:  $\text{content\_eq\_0}$   $\text{elim!}: \text{ball } E[\text{of } \_ \ i]$ )
    then show  $y \neq x$ 
      unfolding  $\text{euclidean\_eq\_iff}$  [where  $'a = 'a]$  using  $i$  by auto
    have  $\text{norm } (y - x) \leq (\sum_{b \in \text{Basis}. |(y - x) \cdot b|}$ 
      by (rule  $\text{norm\_le\_l1}$ )
    also have  $\dots = |(y - x) \cdot i| + (\sum_{b \in \text{Basis} - \{i\}. |(y - x) \cdot b|}$ 
      by (meson  $\text{finite\_Basis } i(2)$   $\text{sum.remove}$ )
    also have  $\dots < e + \text{sum } (\lambda i. 0) \ \text{Basis}$ 
      by (rule  $\text{add\_less\_le\_mono}$ )
    show  $|(y - x) \cdot i| < e$ 

```

```

    using di ⟨e > 0⟩ y_def i xi by (auto simp: inner_simps)
  show (∑ i∈Basis - {i}. |(y-x) · i|) ≤ (∑ i∈Basis. 0)
    unfolding y_def by (auto simp: inner_simps)
qed
finally have norm (y-x) < e + sum (λi. 0) Basis .
then show dist y x < e
  unfolding dist_norm by auto
have y ∉ K
  unfolding keq mem_box using i(1) i(2) xi xyi by fastforce
moreover have y ∈ ⋃ D
  using subsetD[OF D(2)[OF ⟨K ∈ D⟩] ⟨x ∈ K⟩] ⟨e > 0⟩ di i
  by (auto simp: D mem_box y_def field_simps elim!: ballE[of _ _ i])
ultimately show y ∈ ⋃ (D - {K}) by auto
qed
qed
then have K ⊆ ⋃ (D - {K})
  using closed closed_limpt by blast
then have ⋃ (D - {K}) = cbox a b
  unfolding D(6)[symmetric] by auto
then have D - {K} division_of cbox a b
  by (metis Diff_subset less.prems division_of_subset D(6))
then have {ka ∈ D - {K}. content ka ≠ 0} division_of (cbox a b)
  using card less.hyps by blast
moreover have {ka ∈ D - {K}. content ka ≠ 0} = {K ∈ D. content K ≠ 0}
  using contk by auto
ultimately show ?case by auto
qed

```

7.14.20 Integrability on subintervals

```

lemma operative_integrableI:
  fixes f :: 'b::euclidean_space ⇒ 'a::banach
  assumes 0 ≤ e
  shows operative conj True (λi. f integrable_on i)
proof -
  interpret comm_monoid conj True
  proof qed
  show ?thesis
  proof
    show ∧ a b. box a b = {} ⇒ (f integrable_on cbox a b) = True
      by (simp add: content_eq_0_interior integrable_on_null)
    show ∧ a b c k.
      k ∈ Basis ⇒
      (f integrable_on cbox a b) ↔
      (f integrable_on cbox a b ∩ {x. x · k ≤ c} ∧ f integrable_on cbox a b ∩
      {x. c ≤ x · k})
      unfolding integrable_on_def by (auto intro!: has_integral_split)
  qed
qed

```



```

lemma integrable_subinterval:
  fixes f :: 'b::euclidean_space  $\Rightarrow$  'a::banach
  assumes f: f integrable_on cbox a b
    and cd: cbox c d  $\subseteq$  cbox a b
  shows f integrable_on cbox c d
proof -
  interpret operative conj True  $\lambda i.$  f integrable_on i
    using order_refl by (rule operative_integrableI)
  show ?thesis
    by (metis cd division division_of_finite empty f partial_division_extend_1
remove)
qed

```

```

lemma integrable_subinterval_real:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes f integrable_on {a..b}
    and {c..d}  $\subseteq$  {a..b}
  shows f integrable_on {c..d}
  by (metis assms box_real(2) integrable_subinterval)

```

7.14.21 Combining adjacent intervals in 1 dimension

```

lemma has_integral_combine:
  fixes a b c :: real and j :: 'a::banach
  assumes a  $\leq$  c
    and c  $\leq$  b
    and ac: (f has_integral i) {a..c}
    and cb: (f has_integral j) {c..b}
  shows (f has_integral (i + j)) {a..b}
proof -
  interpret operative_real lift_option plus Some 0
     $\lambda i.$  if f integrable_on i then Some (integral i f) else None
    using operative_integralI by (rule operative_realI)
  from  $\langle a \leq c \rangle \langle c \leq b \rangle$  ac cb coalesce_less_eq
  have *: lift_option (+)
    (if f integrable_on {a..c} then Some (integral {a..c} f) else None)
    (if f integrable_on {c..b} then Some (integral {c..b} f) else None) =
    (if f integrable_on {a..b} then Some (integral {a..b} f) else None)
  by (auto simp: split: if_split_asm)
  then have f integrable_on cbox a b
    using ac cb by (auto split: if_split_asm)
  with * show ?thesis
    using ac cb by (auto simp add: integrable_on_def integral_unique split:
if_split_asm)
qed

```

```

lemma integral_combine:
  fixes f :: real  $\Rightarrow$  'a::banach

```

```

assumes  $a \leq c$ 
and  $c \leq b$ 
and  $ab: f \text{ integrable\_on } \{a..b\}$ 
shows  $\text{integral } \{a..c\} f + \text{integral } \{c..b\} f = \text{integral } \{a..b\} f$ 
proof –
  have  $(f \text{ has\_integral } \text{integral } \{a..c\} f) \{a..c\}$ 
    using  $ab \langle c \leq b \rangle \text{ integrable\_subinterval\_real}$  by fastforce
  moreover
  have  $(f \text{ has\_integral } \text{integral } \{c..b\} f) \{c..b\}$ 
    using  $ab \langle a \leq c \rangle \text{ integrable\_subinterval\_real}$  by fastforce
  ultimately show ?thesis
  by  $(\text{smt } (\text{verit, best}) \text{ assms has\_integral\_combine integral\_unique})$ 
qed

```

```

lemma integrable_combine:
fixes  $f :: \text{real} \Rightarrow 'a::\text{banach}$ 
assumes  $a \leq c$ 
and  $c \leq b$ 
and  $f \text{ integrable\_on } \{a..c\}$ 
and  $f \text{ integrable\_on } \{c..b\}$ 
shows  $f \text{ integrable\_on } \{a..b\}$ 
using assms has\_integral\_combine by blast

```

```

lemma integral_minus_sets:
fixes  $f::\text{real} \Rightarrow 'a::\text{banach}$ 
shows  $c \leq a \implies c \leq b \implies f \text{ integrable\_on } \{c .. \max a b\} \implies$ 
   $\text{integral } \{c .. a\} f - \text{integral } \{c .. b\} f =$ 
   $(\text{if } a \leq b \text{ then } - \text{integral } \{a .. b\} f \text{ else } \text{integral } \{b .. a\} f)$ 
using integral_combine[of  $c a b f$ ] integral_combine[of  $c b a f$ ]
by  $(\text{auto simp: algebra_simps max\_def})$ 

```

```

lemma integral_minus_sets':
fixes  $f::\text{real} \Rightarrow 'a::\text{banach}$ 
shows  $c \geq a \implies c \geq b \implies f \text{ integrable\_on } \{\min a b .. c\} \implies$ 
   $\text{integral } \{a .. c\} f - \text{integral } \{b .. c\} f =$ 
   $(\text{if } a \leq b \text{ then } \text{integral } \{a .. b\} f \text{ else } - \text{integral } \{b .. a\} f)$ 
using integral_combine[of  $b a c f$ ] integral_combine[of  $a b c f$ ]
by  $(\text{auto simp: algebra_simps min\_def})$ 

```

7.14.22 Reduce integrability to "local" integrability

```

lemma integrable_on_little_subintervals:
fixes  $f :: 'b::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $\forall x \in \text{cbox } a b. \exists d > 0. \forall u v. x \in \text{cbox } u v \wedge \text{cbox } u v \subseteq \text{ball } x d \wedge \text{cbox } u v \subseteq \text{cbox } a b \implies$ 
   $f \text{ integrable\_on } \text{cbox } u v$ 
shows  $f \text{ integrable\_on } \text{cbox } a b$ 
proof –
  interpret operative conj True  $\lambda i. f \text{ integrable\_on } i$ 

```

```

using order_refl by (rule operative_integrableI)
have  $\forall x. \exists d > 0. x \in \text{cbox } a \ b \longrightarrow (\forall u \ v. x \in \text{cbox } u \ v \wedge \text{cbox } u \ v \subseteq \text{ball } x \ d \wedge$ 
 $\text{cbox } u \ v \subseteq \text{cbox } a \ b \longrightarrow$ 
 $f \text{ integrable\_on } \text{cbox } u \ v)$ 
using assms by (metis zero_less_one)
then obtain d where d:  $\bigwedge x. 0 < d \ x$ 
 $\bigwedge x \ u \ v. [x \in \text{cbox } a \ b; x \in \text{cbox } u \ v; \text{cbox } u \ v \subseteq \text{ball } x \ (d \ x); \text{cbox } u \ v \subseteq \text{cbox}$ 
 $a \ b]$ 
 $\implies f \text{ integrable\_on } \text{cbox } u \ v$ 
by metis
obtain p where p: p tagged_division_of cbox a b ( $\lambda x. \text{ball } x \ (d \ x)$ ) fine p
using fine_division_exists[OF gauge_ball_dependent, of d a b] d(1) by blast
then have sndp: snd ' p division_of cbox a b
by (metis division_of_tagged_division)
have f integrable_on k if (x, k)  $\in$  p for x k
using tagged_division_ofD(2-4)[OF p(1) that] fineD[OF p(2) that] d[of x]
by auto
then show ?thesis
unfolding division [symmetric, OF sndp] comm_monoid_set_F_and
by auto
qed

```

7.14.23 Second FTC or existence of antiderivative

```

lemma integrable_const[intro]: ( $\lambda x. c$ ) integrable_on cbox a b
unfolding integrable_on_def by blast

```

```

lemma integral_has_vector_derivative_continuous_at:

```

```

fixes f :: real  $\Rightarrow$  'a::banach
assumes f: f integrable_on {a..b}
and x: x  $\in$  {a..b} - S
and finite S
and fx: continuous (at x within ({a..b} - S)) f
shows (( $\lambda u. \text{integral } \{a..u\} \ f$ ) has_vector_derivative f x) (at x within ({a..b} - S))
proof -
let ?I =  $\lambda a \ b. \text{integral } \{a..b\} \ f$ 
{ fix e::real
assume e > 0
obtain d where d>0 and d:  $\bigwedge x'. [x' \in \{a..b\} - S; |x' - x| < d] \implies \text{norm}(f$ 
 $x' - f x) \leq e$ 
using  $\langle e > 0 \rangle$  fx by (auto simp: continuous_within_eps_delta dist_norm
less_imp_le)
have norm (integral {a..y} f - integral {a..x} f - (y-x) *R f x)  $\leq e * |y -$ 
 $x|$  (is ?lhs  $\leq$  ?rhs)
if y: y  $\in$  {a..b} - S and yx: |y - x| < d for y
proof (cases y < x)
case False
have f integrable_on {a..y}

```

```

    using f y by (simp add: integrable_subinterval_real)
  then have Idiff: ?I a y - ?I a x = ?I x y
    using False x by (simp add: algebra_simps integral_combine)
  have fux_int: ((λu. f u - f x) has_integral integral {x..y} f - (y-x) *R f x)
{x..y}
  proof (rule has_integral_diff)
    show (f has_integral integral {x..y} f) {x..y}
    using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
[OF f]])
    show ((λu. f x) has_integral (y - x) *R f x) {x..y}
      using has_integral_const_real [of f x x y] False by simp
  qed
  have ?lhs ≤ e * content {x..y}
    using yx False d x y <e>0> assms
    by (intro has_integral_bound_real[where f=(λu. f u - f x)]) (auto simp:
Idiff fux_int)
  also have ... ≤ ?rhs
    using False by auto
  finally show ?thesis .
next
case True
have f integrable_on {a..x}
  using f x by (simp add: integrable_subinterval_real)
then have Idiff: ?I a x - ?I a y = ?I y x
  using True x y by (simp add: algebra_simps integral_combine)
have fux_int: ((λu. f u - f x) has_integral integral {y..x} f - (x - y) *R f
x) {y..x}
  proof (rule has_integral_diff)
    show (f has_integral integral {y..x} f) {y..x}
    using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
[OF f]])
    show ((λu. f x) has_integral (x - y) *R f x) {y..x}
      using has_integral_const_real [of f x y x] True by simp
  qed
  have norm (integral {a..x} f - integral {a..y} f - (x - y) *R f x) ≤ e *
content {y..x}
    using yx True d x y <e>0> assms
    by (intro has_integral_bound_real[where f=(λu. f u - f x)]) (auto simp:
Idiff fux_int)
  also have ... ≤ e * |y - x|
    using True by auto
  finally have norm (integral {a..x} f - integral {a..y} f - (x - y) *R f x) ≤
e * |y - x| .
  then show ?thesis
    by (simp add: algebra_simps norm_minus_commute)
  qed
  then have ∃ d>0. ∀ y∈{a..b} - S. |y - x| < d ⟶ norm (integral {a..y} f -
integral {a..x} f - (y-x) *R f x) ≤ e * |y - x|
    using <d>0> by blast

```

```

}
then show ?thesis
  by (simp add: has_vector_derivative_def has_derivative_within_alt bounded_linear_scaleR_left)
qed

```

```

lemma integral_has_vector_derivative:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes continuous_on {a..b} f
  and x  $\in$  {a..b}
  shows (( $\lambda$ u. integral {a..u} f) has_vector_derivative f(x)) (at x within {a..b})
  using assms integral_has_vector_derivative_continuous_at [OF integrable_continuous_real]
  by (fastforce simp: continuous_on_eq_continuous_within)

```

```

lemma integral_has_real_derivative:
  assumes continuous_on {a..b} g
  assumes t  $\in$  {a..b}
  shows (( $\lambda$ x. integral {a..x} g) has_real_derivative g t) (at t within {a..b})
  using integral_has_vector_derivative[of a b g t] assms
  by (auto simp: has_real_derivative_iff_has_vector_derivative)

```

```

lemma antiderivative_continuous:
  fixes q b :: real
  assumes continuous_on {a..b} f
  obtains g where  $\bigwedge$ x. x  $\in$  {a..b}  $\implies$  (g has_vector_derivative (f x :: 'a::banach))
  (at x within {a..b})
  using integral_has_vector_derivative[OF assms] by auto

```

7.14.24 Combined fundamental theorem of calculus

```

lemma antiderivative_integral_continuous:
  fixes f :: real  $\Rightarrow$  'a::banach
  assumes continuous_on {a..b} f
  obtains g where  $\forall$ u $\in$ {a..b}.  $\forall$ v  $\in$  {a..b}. u  $\leq$  v  $\implies$  (f has_integral (g v - g
u)) {u..v}
proof -
  obtain g
  where g:  $\bigwedge$ x. x  $\in$  {a..b}  $\implies$  (g has_vector_derivative f x) (at x within {a..b})

  using antiderivative_continuous[OF assms] by metis
  have (f has_integral g v - g u) {u..v} if u  $\in$  {a..b} v  $\in$  {a..b} u  $\leq$  v for u v
  proof -
    have  $\bigwedge$ x. x  $\in$  cbox u v  $\implies$  (g has_vector_derivative f x) (at x within cbox u v)
    by (metis atLeastAtMost_iff atLeastatMost_subset_iff box_real(2) g
has_vector_derivative_within_subset subsetCE that(1,2))
    then show ?thesis
    by (metis box_real(2) that(3) fundamental_theorem_of_calculus)
  qed
then show ?thesis

```

using *that* by *blast*
qed

7.14.25 General "twiddling" for interval-to-interval function image

lemma *has_integral_twiddle*:

assumes $0 < r$
 and *hg*: $\bigwedge x. h(g\ x) = x$
 and *gh*: $\bigwedge x. g(h\ x) = x$
 and *contg*: $\bigwedge x. \text{continuous (at } x) g$
 and *g*: $\bigwedge u\ v. \exists w\ z. g\ ' \text{cbox } u\ v = \text{cbox } w\ z$
 and *h*: $\bigwedge u\ v. \exists w\ z. h\ ' \text{cbox } u\ v = \text{cbox } w\ z$
 and *r*: $\bigwedge u\ v. \text{content}(g\ ' \text{cbox } u\ v) = r * \text{content (cbox } u\ v)$
 and *intfi*: $(f\ \text{has_integral } i) (\text{cbox } a\ b)$
 shows $((\lambda x. f(g\ x))\ \text{has_integral } (1 / r) *_R i) (h\ ' \text{cbox } a\ b)$
proof (*cases* *cbox* *a* *b* = $\{\}$)
 case *True*
 then show *?thesis*
 using *intfi* by *auto*
next
 case *False*
obtain *wz* **where** *wz*: $h\ ' \text{cbox } a\ b = \text{cbox } w\ z$
 using *h* by *blast*
have *inj*: *inj* *g* *inj* *h*
 using *hg* *gh* *injI* by *metis+*
from *h* **obtain** *ha* *hb* **where** *h_eq*: $h\ ' \text{cbox } a\ b = \text{cbox } ha\ hb$ by *blast*
have $\exists d. \text{gauge } d \wedge (\forall p. p\ \text{tagged_division_of } h\ ' \text{cbox } a\ b \wedge d\ \text{fine } p$
 $\longrightarrow \text{norm } ((\sum (x, k) \in p. \text{content } k * _R f(g\ x)) - (1 / r) * _R i) < e)$
 if $e > 0$ for *e*
proof –
 have $e * r > 0$ using *that* $\langle 0 < r \rangle$ by *simp*
 with *intfi*[*unfolded* *has_integral*]
obtain *d* **where** *gauge* *d*
 and *d*: $\bigwedge p. p\ \text{tagged_division_of } \text{cbox } a\ b \wedge d\ \text{fine } p$
 $\implies \text{norm } ((\sum (x, k) \in p. \text{content } k * _R f\ x) - i) < e * r$
 by *metis*
define *d'* **where** $d'\ x = g\ -\ ' d\ (g\ x)$ for *x*
show *?thesis*
proof (*rule_tac* *x=d'* in *exI*, *safe*)
 show *gauge* *d'*
 using $\langle \text{gauge } d \rangle$ *continuous_open_vimage*[*OF* *contg*] by (*auto* *simp*:
gauge_def *d'_def*)
next
fix *p*
assume *ptag*: *p* *tagged_division_of* $h\ ' \text{cbox } a\ b$ and *finp*: *d'* *fine* *p*
note *p* = *tagged_division_ofD*[*OF* *ptag*]
have *gab*: $g\ y \in \text{cbox } a\ b$ if $y \in K\ (x, K) \in p$ for *x* *y* *K*
 by (*metis* *hg* *inj*(2) *inj_image_mem_iff* *p*(3) *subsetCE* *that* *that*)

```

have gimp:  $(\lambda(x,K). (g x, g ' K)) ' p$  tagged_division_of (cbox a b)  $\wedge$ 
  d_fine  $(\lambda(x,k). (g x, g ' k)) ' p$ 
  unfolding tagged_division_of
proof safe
  show finite  $((\lambda(x,k). (g x, g ' k)) ' p)$ 
    using ptag by auto
  show d_fine  $(\lambda(x,k). (g x, g ' k)) ' p$ 
    using finep unfolding fine_def d'_def by auto
next
fix x K
assume xk:  $(x, K) \in p$ 
show  $g x \in g ' K$ 
  using p(2)[OF xk] by auto
show  $\exists u v. g ' K = \text{cbox } u v$ 
  using p(4)[OF xk] using assms(5-6) by auto
fix x' K' u
assume xk':  $(x', K') \in p$  and u:  $u \in \text{interior } (g ' K) \wedge u \in \text{interior } (g ' K')$ 
have  $\text{interior } K \cap \text{interior } K' \neq \{\}$ 
proof
  assume  $\text{interior } K \cap \text{interior } K' = \{\}$ 
  moreover have  $u \in g ' (\text{interior } K \cap \text{interior } K')$ 
    using interior_image_subset[OF <inj g> contg] u
    unfolding image_Int[OF inj(1)] by blast
  ultimately show False by blast
qed
then have same:  $(x, K) = (x', K')$ 
  using ptag xk' xk by blast
then show  $g x = g x'$ 
  by auto
show  $g u \in g ' K$  if  $u \in K$  for u
  using that same by auto
show  $g u \in g ' K$  if  $u \in K'$  for u
  using that same by auto
next
fix x
assume  $x \in \text{cbox } a b$ 
then have  $h x \in \bigcup \{k. \exists x. (x, k) \in p\}$ 
  using p(6) by auto
then obtain X y where  $h x \in X (y, X) \in p$  by blast
then show  $x \in \bigcup \{k. \exists x. (x, k) \in (\lambda(x,k). (g x, g ' k)) ' p\}$ 
  by clarsimp (metis (no_types, lifting) gh_image_eqI pair_imageI)
qed (use gab in auto)
have *:  $\text{inj\_on } (\lambda(x,k). (g x, g ' k)) p$ 
  using inj(1) unfolding inj_on_def by fastforce
have  $(\sum_{(x,K) \in (\lambda(y,L). (g y, g ' L)) ' p. \text{content } K *_R f x}$ 
  =  $(\sum_{u \in p. \text{case case } u \text{ of } (x,K) \Rightarrow (g x, g ' K) \text{ of } (y,L) \Rightarrow \text{content } L *_R$ 
f y)
  by (metis (mono_tags, lifting) * sum.reindex_cong)
also have ... =  $(\sum_{(x,K) \in p. r *_R \text{content } K *_R f (g x)}$ 

```

```

    using r by (auto intro!: * sum.cong simp: bij_betw_def dest!: p(4))
  finally
    have  $(\sum (x, K) \in \lambda(x, K). (g \ x, g \ 'K)) \ ' p. \text{content } K \ *_R \ f \ x) - i = r \ *_R$ 
 $(\sum (x, K) \in p. \text{content } K \ *_R \ f \ (g \ x)) - i$ 
    by (simp add: scaleR_right.sum_split_def)
    also have  $\dots = r \ *_R ((\sum (x, K) \in p. \text{content } K \ *_R \ f \ (g \ x)) - (1 / r) \ *_R \ i)$ 
    using  $\langle 0 < r \rangle$  by (auto simp: scaleR_diff_right)
    finally show  $\text{norm } ((\sum (x, K) \in p. \text{content } K \ *_R \ f \ (g \ x)) - (1 / r) \ *_R \ i) < e$ 
    using  $d[OF \ gimp] \langle 0 < r \rangle$  by auto
  qed
qed
then show ?thesis
  by (auto simp: h_eq has_integral)
qed

```

7.14.26 Special case of a basic affine transformation

lemma *AE_lborel_inner_neg*:

assumes $k: k \in \text{Basis}$

shows *AE x in lborel. $x \cdot k \neq c$*

proof –

interpret *finite_product_sigma_finite* $\lambda_. \text{lborel } \text{Basis}$

proof qed *simp*

have *emeasure lborel* $\{x \in \text{space } \text{lborel}. x \cdot k = c\}$

$= \text{emeasure } (\prod_M j::'a \in \text{Basis}. \text{lborel}) (\prod_E j \in \text{Basis}. \text{if } j = k \text{ then } \{c\} \text{ else}$

UNIV)

using *k*

by (auto simp add: *lborel_eq*[**where** $'a = 'a$] *emeasure_distr intro!*: *arg_cong2*[**where** $f = \text{emeasure}$])

(*auto simp: space_PiM PiE_iff extensional_def split: if_split_asm*)

also have $\dots = (\prod j \in \text{Basis}. \text{emeasure } \text{lborel} (\text{if } j = k \text{ then } \{c\} \text{ else } \text{UNIV}))$

by (*intro measure_times*) *auto*

also have $\dots = 0$

by (*intro prod_zero bexI*[*OF* $_ k$]) *auto*

finally show ?thesis

by (*subst AE_iff_measurable*[*OF* $_ refl$]) *auto*

qed

lemma *content_image_stretch_interval*:

fixes $m :: 'a::\text{euclidean_space} \Rightarrow \text{real}$

defines $s \ f \ x \equiv (\sum k::'a \in \text{Basis}. (f \ k * (x \cdot k)) \ *_R \ k)$

shows $\text{content } (s \ m \ ' \ \text{cbox } a \ b) = |\prod k \in \text{Basis}. m \ k| * \text{content } (\text{cbox } a \ b)$

proof *cases*

have *s*[*measurable*]: $s \ f \in \text{borel} \rightarrow_M \text{borel}$ **for** *f*

by (*auto simp: s_def*[*abs_def*])

assume $m: \forall k \in \text{Basis}. m \ k \neq 0$

then have *s_comp_s*: $s \ (\lambda k. 1 / m \ k) \circ s \ m = \text{id } s \ m \circ s \ (\lambda k. 1 / m \ k) = \text{id}$

by (*auto simp: s_def*[*abs_def*] *fun_eq_iff euclidean_representation*)

then have *inv* $(s \ (\lambda k. 1 / m \ k)) = s \ m \ \text{bij } (s \ (\lambda k. 1 / m \ k))$


```

  by (auto intro: inv_unique_comp o_bij)
  then have eq:  $s \ m \ \langle \text{cbox } a \ b = s \ (\lambda k. 1 / m \ k) \ - \ \langle \text{cbox } a \ b$ 
    using bij_vimage_eq_inv_image[OF  $\langle \text{bij } (s \ (\lambda k. 1 / m \ k)) \rangle$ , of cbox a b] by
  auto
  show ?thesis
    using m_unfolding eq measure_def
    by (subst lborel_affine_euclidean[where c=m and t=0])
      (simp_all add: emeasure_density measurable_sets_borel[OF s] abs_prod
  nn_integral_cmult
      s_def[symmetric] emeasure_distr vimage_comp s_comp_s
  enn2real_mult prod_nonneg)
  next
  assume  $\neg (\forall k \in \text{Basis}. m \ k \neq 0)$ 
  then obtain k where  $k: k \in \text{Basis} \ m \ k = 0$  by auto
  then have [simp]:  $(\prod k \in \text{Basis}. m \ k) = 0$ 
    by (intro prod_zero) auto
  have emeasure_lborel  $\{x \in \text{space } \text{lborel}. x \in s \ m \ \langle \text{cbox } a \ b\} = 0$ 
  proof (rule emeasure_eq_0_AE)
    show AE x in lborel.  $x \notin s \ m \ \langle \text{cbox } a \ b$ 
      using AE_lborel_inner_neq[OF  $\langle k \in \text{Basis} \rangle$ ]
    proof eventually_elim
      show  $x \cdot k \neq 0 \implies x \notin s \ m \ \langle \text{cbox } a \ b$  for x
        using k by (auto simp: s_def[abs_def] cbox_def)
    qed
  qed
  then show ?thesis
    by (simp add: measure_def)
  qed

```

```

lemma interval_image_affinity_interval:
   $\exists u \ v. (\lambda x. m \ *_R \ (x::'a::\text{euclidean\_space}) + c) \ \langle \text{cbox } a \ b = \text{cbox } u \ v$ 
  unfolding image_affinity_cbox
  by auto

```

```

lemma content_image_affinity_cbox:
   $\text{content}((\lambda x::'a::\text{euclidean\_space}. m \ *_R \ x + c) \ \langle \text{cbox } a \ b) =$ 
   $|m| \wedge \text{DIM}'a * \text{content}(\text{cbox } a \ b)$  (is ?l = ?r)
  proof (cases cbox a b = {})
    case True then show ?thesis by simp
  next
  case False
  show ?thesis
    proof (cases  $m \geq 0$ )
      case True
      with  $\langle \text{cbox } a \ b \neq \{\} \rangle$  have  $\text{cbox } (m \ *_R \ a + c) \ (m \ *_R \ b + c) \neq \{\}$ 
        by (simp add: box_ne_empty inner_left_distrib mult_left_mono)
      moreover from True have *:  $\bigwedge i. (m \ *_R \ b + c) \cdot i - (m \ *_R \ a + c) \cdot i = m$ 
         $*_R \ (b - a) \cdot i$ 
        by (simp add: inner_simps field_simps)
    qed
  qed

```

```

ultimately show ?thesis
  by (simp add: image_affinity_cbox True content_cbox' prod.distrib inner_diff_left)
next
  case False
  with ⟨cbox a b ≠ {}⟩ have cbox (m *R b + c) (m *R a + c) ≠ {}
    by (simp add: box_ne_empty inner_left_distrib mult_left_mono)
  moreover from False have *:  $\bigwedge i. (m *R a + c) \cdot i - (m *R b + c) \cdot i = (-m) *R (b - a) \cdot i$ 
    by (simp add: inner_simps field_simps)
  ultimately show ?thesis using False
    by (simp add: image_affinity_cbox content_cbox' prod.distrib[symmetric] inner_diff_left_flip: prod_constant)
qed
qed

```

```

lemma has_integral_affinity:
  fixes a :: 'a::euclidean_space
  assumes (f has_integral i) (cbox a b)
  and m ≠ 0
  shows (( $\lambda x. f(m *R x + c)$ ) has_integral (1 / (|m| ^ DIM('a))) *R i) (( $\lambda x. (1 / m) *R x + -((1 / m) *R c)$ ) ' cbox a b)
proof (rule has_integral_twiddle)
  show  $\exists w z. (\lambda x. (1 / m) *R x + -((1 / m) *R c)) ' cbox u v = cbox w z$ 
     $\exists w z. (\lambda x. m *R x + c) ' cbox u v = cbox w z$  for u v
  using interval_image_affinity_interval by blast+
  show content (( $\lambda x. m *R x + c$ ) ' cbox u v) = |m| ^ DIM('a) * content (cbox u v) for u v
  using content_image_affinity_cbox by blast
qed (use assms zero_less_power in ⟨auto simp: field_simps⟩)

```

```

lemma integrable_affinity:
  assumes f integrable_on cbox a b
  and m ≠ 0
  shows ( $\lambda x. f(m *R x + c)$ ) integrable_on (( $\lambda x. (1 / m) *R x + -((1 / m) *R c)$ ) ' cbox a b)
  using has_integral_affinity assms
  unfolding integrable_on_def by blast

```

```

lemmas has_integral_affinity01 = has_integral_affinity [of _ _ 0 1::real, simplified]

```

```

lemma integrable_on_affinity:
  assumes m ≠ 0 f integrable_on (cbox a b)
  shows ( $\lambda x. f(m *R x + c)$ ) integrable_on (( $\lambda x. (1 / m) *R x + -((1 / m) *R c)$ ) ' cbox a b)
proof -
  from assms obtain I where (f has_integral I) (cbox a b)
  by (auto simp: integrable_on_def)

```

from *has_integral_affinity*[*OF this assms(1), of c*] **show** *?thesis*
 by (*auto simp: integrable_on_def*)
qed

lemma *has_integral_cmul_iff*:
 assumes $c \neq 0$
 shows $((\lambda x. c *_{\mathbb{R}} f x) \text{ has_integral } (c *_{\mathbb{R}} I)) A \longleftrightarrow (f \text{ has_integral } I) A$
 using *assms has_integral_cmul*[*of f I A c*]
 has_integral_cmul[*of $\lambda x. c *_{\mathbb{R}} f x c *_{\mathbb{R}} I A$ inverse c*]
 by (*auto simp: field_simps*)

lemma *has_integral_cmul_iff'*:
 assumes $c \neq 0$
 shows $((\lambda x. c *_{\mathbb{R}} f x) \text{ has_integral } I) A \longleftrightarrow (f \text{ has_integral } I /_{\mathbb{R}} c) A$
 using *assms* **by** (*metis divideR_right has_integral_cmul_iff*)

lemma *has_integral_affinity'*:
 fixes $a :: 'a :: \text{euclidean_space}$
 assumes $(f \text{ has_integral } i) (cbox\ a\ b)$ **and** $m > 0$
 shows $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has_integral } (i /_{\mathbb{R}} m \wedge DIM('a)))$
 $(cbox\ ((a - c) /_{\mathbb{R}} m)\ ((b - c) /_{\mathbb{R}} m))$
proof (*cases cbox a b = {}*)
 case *True*
 hence $(cbox\ ((a - c) /_{\mathbb{R}} m)\ ((b - c) /_{\mathbb{R}} m)) = \{\}$
 using $\langle m > 0 \rangle$ **unfolding** *box_eq_empty* **by** (*auto simp: algebra_simps*)
 with *True* **and** *assms* **show** *?thesis* **by** *simp*
next
 case *False*
 have $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has_integral } (1 / |m| \wedge DIM('a)) *_{\mathbb{R}} i)$
 $((\lambda x. (1 / m) *_{\mathbb{R}} x + -((1 / m) *_{\mathbb{R}} c)) \text{ ' } cbox\ a\ b)$
 using *assms* **by** (*intro has_integral_affinity*) *auto*
 also have $((\lambda x. (1 / m) *_{\mathbb{R}} x + -((1 / m) *_{\mathbb{R}} c)) \text{ ' } cbox\ a\ b) =$
 $((\lambda x. -((1 / m) *_{\mathbb{R}} c) + x) \text{ ' } (\lambda x. (1 / m) *_{\mathbb{R}} x) \text{ ' } cbox\ a\ b)$
 by (*simp add: image_image algebra_simps*)
 also have $(\lambda x. (1 / m) *_{\mathbb{R}} x) \text{ ' } cbox\ a\ b = cbox\ ((1 / m) *_{\mathbb{R}} a)\ ((1 / m) *_{\mathbb{R}} b)$
 using $\langle m > 0 \rangle$ *False*
 by (*subst image_smult_cbox*) *simp_all*
 also have $(\lambda x. -((1 / m) *_{\mathbb{R}} c) + x) \text{ ' } \dots = cbox\ ((a - c) /_{\mathbb{R}} m)\ ((b - c) /_{\mathbb{R}} m)$
 by (*subst cbox_translation* [*symmetric*]) (*simp add: field_simps vector_add_divide_simps*)
 finally **show** *?thesis* **using** $\langle m > 0 \rangle$
 by (*simp add: field_simps*)
qed

lemma *has_integral_affinity_iff*:
 fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{real_normed_vector}$
 assumes $m > 0$
 shows $((\lambda x. f(m *_{\mathbb{R}} x + c)) \text{ has_integral } (I /_{\mathbb{R}} m \wedge DIM('a)))$
 $(cbox\ ((a - c) /_{\mathbb{R}} m)\ ((b - c) /_{\mathbb{R}} m)) \longleftrightarrow$

$(f \text{ has_integral } I) (cbox \ a \ b) \text{ (is } ?lhs = ?rhs)$

proof

assume $?lhs$

from $\text{has_integral_affinity}'[OF \text{ this, of } 1 / m - c /_R m]$ **and** $\langle m > 0 \rangle$

show $?rhs$ **by** $(\text{simp add: vector_add_divide_simps}) (\text{simp add: field_simps})$

next

assume $?rhs$

from $\text{has_integral_affinity}'[OF \text{ this, of } m \ c]$ **and** $\langle m > 0 \rangle$

show $?lhs$ **by** simp

qed

7.14.27 Special case of stretching coordinate axes separately

lemma $\text{has_integral_stretch}$:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{real_normed_vector}$

assumes $(f \text{ has_integral } i) (cbox \ a \ b)$

and $\forall k \in \text{Basis}. m \ k \neq 0$

shows $((\lambda x. f (\sum_{k \in \text{Basis}} (m \ k * (x \cdot k)) *_R k)) \text{ has_integral } ((1 / |\text{prod } m \ \text{Basis}|) *_R i)) ((\lambda x. (\sum_{k \in \text{Basis}} (1 / m \ k * (x \cdot k)) *_R k)) \text{ ` } cbox \ a \ b)$

apply $(\text{rule } \text{has_integral_twiddle}[\text{where } f=f])$

unfolding $\text{zero_less_abs_iff } \text{content_image_stretch_interval}$

unfolding $\text{image_stretch_interval_empty_as_interval } \text{euclidean_eq_iff}[\text{where } 'a='a]$

using assms

by auto

lemma $\text{has_integral_stretch_real}$:

fixes $f :: \text{real} \Rightarrow 'b::\text{real_normed_vector}$

assumes $(f \text{ has_integral } i) \{a..b\}$ **and** $m \neq 0$

shows $((\lambda x. f (m * x)) \text{ has_integral } (1 / |m|) *_R i) ((\lambda x. x / m) \text{ ` } \{a..b\})$

using $\text{has_integral_stretch } [of \ f \ i \ a \ b \ \lambda b. \ m]$ **assms** **by** simp

lemma $\text{integrable_stretch}$:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{real_normed_vector}$

assumes $f \text{ integrable_on } cbox \ a \ b$

and $\forall k \in \text{Basis}. m \ k \neq 0$

shows $(\lambda x::'a. f (\sum_{k \in \text{Basis}} (m \ k * (x \cdot k)) *_R k)) \text{ integrable_on } ((\lambda x. \sum_{k \in \text{Basis}} (1 / m \ k * (x \cdot k)) *_R k) \text{ ` } cbox \ a \ b)$

using assms **unfolding** integrable_on_def

by $(\text{force } \text{dest: } \text{has_integral_stretch})$

lemma vec_lambda_eq_sum :

$(\chi \ k. f \ k (x \ \$ \ k)) = (\sum_{k \in \text{Basis}} (f (\text{axis_index } k) (x \cdot k)) *_R k) \text{ (is } ?lhs = ?rhs)$

proof –

have $?lhs = (\chi \ k. f \ k (x \cdot \text{axis } k \ 1))$

by $(\text{simp add: } \text{cart_eq_inner_axis})$

also have $\dots = (\sum_{u \in \text{UNIV}. f \ u (x \cdot \text{axis } u \ 1)) *_R \text{axis } u \ 1)$

```

  by (simp add: vec_eq_iff axis_def if_distrib cong: if_cong)
  also have ... = ?rhs
  by (simp add: Basis_vec_def UNION_singleton_eq_range sum.reindex axis_eq_axis
inj_on_def)
  finally show ?thesis .
qed

```

lemma *has_integral_stretch_cart*:

```

  fixes m :: 'n::finite  $\Rightarrow$  real
  assumes f: (f has_integral i) (cbox a b) and m:  $\bigwedge k. m\ k \neq 0$ 
  shows (( $\lambda x. f(\chi\ k. m\ k * x\$k)$ ) has_integral i /R |prod m UNIV|)
        (( $\lambda x. \chi\ k. x\$k / m\ k$ ) ' (cbox a b))

```

proof –

```

  have *:  $\forall k:: \text{real}^n \in \text{Basis}. m\ (\text{axis\_index}\ k) \neq 0$ 
  using axis_index by (simp add: m)
  have eqp: ( $\prod k:: \text{real}^n \in \text{Basis}. m\ (\text{axis\_index}\ k)$ ) = prod m UNIV
  by (simp add: Basis_vec_def UNION_singleton_eq_range prod.reindex axis_eq_axis
inj_on_def)
  show ?thesis
  using has_integral_stretch [OF f *] vec_lambda_eq_sum [where f= $\lambda i\ x. m\ i$ 
* x] vec_lambda_eq_sum [where f= $\lambda i\ x. x / m\ i$ ]
  by (simp add: field_simps eqp)
qed

```

lemma *image_stretch_interval_cart*:

```

  fixes m :: 'n::finite  $\Rightarrow$  real
  shows ( $\lambda x. \chi\ k. m\ k * x\$k$ ) ' cbox a b =
        (if cbox a b = {} then {}
         else cbox ( $\chi\ k. \min\ (m\ k * a\$k)\ (m\ k * b\$k)$ ) ( $\chi\ k. \max\ (m\ k * a\$k)\ (m\ k * b\$k)$ ))

```

proof –

```

  have *: ( $\sum k \in \text{Basis}. \min\ (m\ (\text{axis\_index}\ k) * (a \cdot k))\ (m\ (\text{axis\_index}\ k) * (b \cdot k))$ ) *R k
        = ( $\chi\ k. \min\ (m\ k * a\ \$\ k)\ (m\ k * b\ \$\ k)$ )
        ( $\sum k \in \text{Basis}. \max\ (m\ (\text{axis\_index}\ k) * (a \cdot k))\ (m\ (\text{axis\_index}\ k) * (b \cdot k))$ ) *R k
        = ( $\chi\ k. \max\ (m\ k * a\ \$\ k)\ (m\ k * b\ \$\ k)$ )
  apply (simp_all add: Basis_vec_def cart_eq_inner_axis UNION_singleton_eq_range
sum.reindex axis_eq_axis inj_on_def)
  apply (simp_all add: vec_eq_iff axis_def if_distrib cong: if_cong)
  done
  show ?thesis
  by (simp add: vec_lambda_eq_sum [where f= $\lambda i\ x. m\ i * x$ ] image_stretch_interval
eq_cbox *)
qed

```

7.14.28 even more special cases

lemma *uminus_interval_vector*[simp]:

```

fixes  $a\ b :: 'a::euclidean\_space$ 
shows  $uminus\ 'cbox\ a\ b = cbox\ (-b)\ (-a)$ 
proof -
  have  $x \in uminus\ 'cbox\ a\ b$  if  $x \in cbox\ (-b)\ (-a)$  for  $x$ 
    by (smt (verit) add.inverse_inverse_image_iff inner_minus_left mem_box(2))
  that)
  then show ?thesis
    by (auto simp: mem_box)
qed

```

```

lemma has_integral_reflect_lemma[intro]:
  assumes ( $f$  has_integral  $i$ ) ( $cbox\ a\ b$ )
  shows  $((\lambda x. f(-x))$  has_integral  $i$ ) ( $cbox\ (-b)\ (-a)$ )
  using has_integral_affinity[OF assms, of -1 0]
  by auto

```

```

lemma has_integral_reflect_lemma_real[intro]:
  assumes ( $f$  has_integral  $i$ )  $\{a..b::real\}$ 
  shows  $((\lambda x. f(-x))$  has_integral  $i$ )  $\{-b .. -a\}$ 
  by (metis has_integral_reflect_lemma interval_cbox assms)

```

```

lemma has_integral_reflect[simp]:
   $((\lambda x. f(-x))$  has_integral  $i$ ) ( $cbox\ (-b)\ (-a)$ )  $\longleftrightarrow$  ( $f$  has_integral  $i$ ) ( $cbox\ a\ b$ )
  by (auto dest: has_integral_reflect_lemma)

```

```

lemma has_integral_reflect_real[simp]:
  fixes  $a\ b::real$ 
  shows  $((\lambda x. f(-x))$  has_integral  $i$ )  $\{-b..-a\}$   $\longleftrightarrow$  ( $f$  has_integral  $i$ )  $\{a..b\}$ 
  by (metis has_integral_reflect interval_cbox)

```

```

lemma integrable_reflect[simp]:  $(\lambda x. f(-x))$  integrable_on  $cbox\ (-b)\ (-a)$   $\longleftrightarrow$   $f$ 
integrable_on  $cbox\ a\ b$ 
  unfolding integrable_on_def by auto

```

```

lemma integrable_reflect_real[simp]:  $(\lambda x. f(-x))$  integrable_on  $\{-b .. -a\}$   $\longleftrightarrow$ 
 $f$  integrable_on  $\{a..b::real\}$ 
  unfolding box_real[symmetric]
  by (rule integrable_reflect)

```

```

lemma integral_reflect[simp]: integral ( $cbox\ (-b)\ (-a)$ )  $(\lambda x. f(-x)) =$  integral
( $cbox\ a\ b$ )  $f$ 
  unfolding integral_def by auto

```

```

lemma integral_reflect_real[simp]: integral  $\{-b .. -a\}$   $(\lambda x. f(-x)) =$  integral
 $\{a..b::real\}$   $f$ 
  unfolding box_real[symmetric]
  by (rule integral_reflect)

```

7.14.29 Stronger form of FCT; quite a tedious proof

lemma *split_minus[simp]*: $(\lambda(x, k). f\ x\ k)\ x - (\lambda(x, k). g\ x\ k)\ x = (\lambda(x, k). f\ x\ k - g\ x\ k)\ x$
by (*simp add: split_def*)

theorem *fundamental_theorem_of_calculus_interior*:

fixes $f :: \text{real} \Rightarrow 'a::\text{real_normed_vector}$

assumes $a \leq b$

and *contf*: *continuous_on* $\{a..b\}$ f

and *derf*: $\bigwedge x. x \in \{a <..< b\} \implies (f\ \text{has_vector_derivative}\ f'\ x)$ (*at* b)

shows $(f'\ \text{has_integral}\ (f\ b - f\ a))\ \{a..b\}$

proof (*cases* $a = b$)

case *True*

then have $*$: $\text{cbox}\ a\ b = \{b\}$ $f\ b - f\ a = 0$

by (*auto simp add: order_antisym*)

with *True* **show** *?thesis* **by** *auto*

next

case *False*

with $\langle a \leq b \rangle$ **have** *ab*: $a < b$ **by** *arith*

show *?thesis*

unfolding *has_integral_factor_content_real*

proof (*intro allI impI*)

fix $e :: \text{real}$

assume $e: e > 0$

then have *eba8*: $(e * (b - a)) / 8 > 0$

using *ab* **by** (*auto simp add: field_simps*)

note *derf_exp* = *derf*[*unfolded* *has_vector_derivative_def* *has_derivative_at_alt*,
THEN conjunct2, *rule_format*]

have *bounded*: $\bigwedge x. x \in \{a <..< b\} \implies \text{bounded_linear}\ (\lambda u. u *_{\mathbb{R}} f'\ x)$

by (*simp add: bounded_linear_scaleR_left*)

have $\forall x \in \text{box}\ a\ b. \exists d > 0. \forall y. \text{norm}\ (y - x) < d \implies \text{norm}\ (f\ y - f\ x - (y - x) *_{\mathbb{R}} f'\ x) \leq e/2 * \text{norm}\ (y - x)$

(**is** $\forall x \in \text{box}\ a\ b. ?Q\ x$) — The explicit quantifier is required by the following

step

proof

fix x **assume** $x \in \text{box}\ a\ b$

with e **show** *?Q* x

using *derf_exp* [*of* $x\ e/2$] **by** *auto*

qed

then obtain d **where** $d: \bigwedge x. 0 < d\ x$

$\bigwedge x\ y. \llbracket x \in \text{box}\ a\ b; \text{norm}\ (y - x) < d\ x \rrbracket \implies \text{norm}\ (f\ y - f\ x - (y - x) *_{\mathbb{R}} f'\ x) \leq e/2 * \text{norm}\ (y - x)$

unfolding *bgauge_existence_lemma* **by** *metis*

have *bounded* ($f\ ^\text{c}\ \text{cbox}\ a\ b$)

using *compact_cbox* *assms* **by** (*auto simp: compact_imp_bounded compact_continuous_image*)

then obtain B

where $0 < B$ **and** $B: \bigwedge x. x \in f\ ^\text{c}\ \text{cbox}\ a\ b \implies \text{norm}\ x \leq B$

unfolding *bounded_pos* **by** *metis*

```

obtain  $da$  where  $0 < da$ 
  and  $da: \bigwedge c. \llbracket a \leq c; \{a..c\} \subseteq \{a..b\}; \{a..c\} \subseteq \text{ball } a \ da \rrbracket$ 
     $\implies \text{norm} (\text{content } \{a..c\} *_R f' a - (f c - f a)) \leq (e * (b-a))$ 
/ 4
proof -
  have  $\text{continuous} (\text{at } a \ \text{within } \{a..b\}) f$ 
    using  $\text{contf\_continuous\_on\_eq\_continuous\_within}$  by  $\text{force}$ 
  with  $eba8$  obtain  $k$  where  $0 < k$ 
    and  $k: \bigwedge x. \llbracket x \in \{a..b\}; 0 < \text{norm} (x-a); \text{norm} (x-a) < k \rrbracket \implies \text{norm} (f x$ 
  -  $f a) < e * (b-a) / 8$ 
    unfolding  $\text{continuous\_within\_Lim\_within\_dist\_norm}$  by  $\text{metis}$ 
  obtain  $l$  where  $l: 0 < l \ \text{norm} (l *_R f' a) \leq e * (b-a) / 8$ 
  proof ( $\text{cases } f' a = 0$ )
    case  $\text{True}$  with  $ab\ e$  that show  $?thesis$  by  $\text{auto}$ 
  next
    case  $\text{False}$ 
    show  $?thesis$ 
    proof
      show  $\text{norm} ((e * (b-a) / 8 / \text{norm} (f' a)) *_R f' a) \leq e * (b-a) / 8$ 
         $0 < e * (b-a) / 8 / \text{norm} (f' a)$ 
        using  $\text{False } ab\ e$  by ( $\text{auto simp add: field\_simps}$ )
      qed
    qed
  have  $\text{norm} (\text{content } \{a..c\} *_R f' a - (f c - f a)) \leq e * (b-a) / 4$ 
    if  $a \leq c \ \{a..c\} \subseteq \{a..b\}$  and  $bmin: \{a..c\} \subseteq \text{ball } a \ (\text{min } k\ l)$  for  $c$ 
  proof -
    have  $\text{minkl}: |a - x| < \text{min } k\ l$  if  $x \in \{a..c\}$  for  $x$ 
      using  $bmin \ \text{dist\_real\_def}$  that by  $\text{auto}$ 
    then have  $\text{lel}: |c - a| \leq |l|$ 
      using  $\text{that}$  by  $\text{force}$ 
    have  $\text{norm} ((c - a) *_R f' a - (f c - f a)) \leq \text{norm} ((c - a) *_R f' a) +$ 
   $\text{norm} (f c - f a)$ 
      by ( $\text{rule norm\_triangle\_ineq4}$ )
    also have  $\dots \leq e * (b-a) / 8 + e * (b-a) / 8$ 
    proof ( $\text{rule add\_mono}$ )
      have  $\text{norm} ((c - a) *_R f' a) \leq \text{norm} (l *_R f' a)$ 
        by ( $\text{auto intro: mult\_right\_mono [OF lel]}$ )
      with  $l$  show  $\text{norm} ((c - a) *_R f' a) \leq e * (b-a) / 8$ 
        by  $\text{linarith}$ 
    next
      have  $\text{norm} (f c - f a) < e * (b-a) / 8$ 
      proof ( $\text{cases } a = c$ )
        case  $\text{True}$  then show  $?thesis$ 
          using  $eba8$  by  $\text{auto}$ 
        next
          case  $\text{False}$  show  $?thesis$ 
            by ( $\text{rule } k$ ) ( $\text{use minkl } \langle a \leq c \rangle$  that  $\text{False}$  in  $\text{auto}$ )
      qed
    then show  $\text{norm} (f c - f a) \leq e * (b-a) / 8$  by  $\text{simp}$ 

```



```

qed
finally show norm (content {a..c} *R f' a - (f c - f a)) ≤ e * (b-a) / 4
  unfolding content_real[OF ‹a ≤ c›] by auto
qed
then show ?thesis
  by (rule_tac da=min k l in that) (auto simp: l < 0 < k)
qed
obtain db where 0 < db
  and db: ∧c. [c ≤ b; {c..b} ⊆ {a..b}; {c..b} ⊆ ball b db]
    ⇒ norm (content {c..b} *R f' b - (f b - f c)) ≤ (e * (b-a))
/ 4
proof -
  have continuous (at b within {a..b}) f
    using contf continuous_on_eq_continuous_within by force
  with eba8 obtain k
    where 0 < k
      and k: ∧x. [x ∈ {a..b}; 0 < norm(x-b); norm(x-b) < k]
        ⇒ norm (f b - f x) < e * (b-a) / 8
  unfolding continuous_within Lim_within dist_norm norm_minus_commute
by metis
  obtain l where l: 0 < l norm (l *R f' b) ≤ (e * (b-a)) / 8
  proof (cases f' b = 0)
    case True thus ?thesis
      using ab e that by auto
  next
    case False show ?thesis
  proof
    show norm ((e * (b - a) / 8 / norm (f' b)) *R f' b) ≤ e * (b - a) / 8
      0 < e * (b - a) / 8 / norm (f' b)
      using False ab e by (auto simp add: field_simps)
  qed
qed
qed
have norm (content {c..b} *R f' b - (f b - f c)) ≤ e * (b-a) / 4
  if c ≤ b {c..b} ⊆ {a..b} and bmin: {c..b} ⊆ ball b (min k l) for c
proof -
  have minkl: |b - x| < min k l if x ∈ {c..b} for x
    using bmin dist_real_def that by auto
  then have lel: |b - c| ≤ |l|
    using that by force
  have norm ((b - c) *R f' b - (f b - f c)) ≤ norm ((b - c) *R f' b) +
norm (f b - f c)
  by (rule norm_triangle_ineq4)
  also have ... ≤ e * (b-a) / 8 + e * (b-a) / 8
  proof (rule add_mono)
    have norm ((b - c) *R f' b) ≤ norm (l *R f' b)
      by (auto intro: mult_right_mono [OF lel])
    also have ... ≤ e * (b-a) / 8
      by (rule l)
  finally show norm ((b - c) *R f' b) ≤ e * (b-a) / 8 .

```

```

next
  have norm (f b - f c) < e * (b-a) / 8
  proof (cases b = c)
    case True with eba8 show ?thesis
      by auto
    next
      case False show ?thesis
        by (rule k) (use minkl <c ≤ b> that False in auto)
  qed
  then show norm (f b - f c) ≤ e * (b-a) / 8 by simp
qed
finally show norm (content {c..b} *R f' b - (f b - f c)) ≤ e * (b-a) / 4
  unfolding content_real[OF <c ≤ b>] by auto
qed
then show ?thesis
  by (rule_tac db=min k l in that) (auto simp: l < 0 < k)
qed
let ?d = (λx. ball x (if x=a then da else if x=b then db else d x))
show ∃ d. gauge d ∧ (∀ p. p tagged_division_of {a..b} ∧ d fine p →
  norm ((∑ (x,K)∈p. content K *R f' x) - (f b - f a)) ≤ e * content
{a..b})
proof (rule exI, safe)
  show gauge ?d
    using ab <db > 0 <da > 0 d(1) by (auto intro: gauge_ball_dependent)
next
  fix p
  assume ptag: p tagged_division_of {a..b} and fine: ?d fine p
  let ?A = {t. fst t ∈ {a, b}}
  note p = tagged_division_ofD[OF ptag]
  have pA: p = (p ∩ ?A) ∪ (p - ?A) finite (p ∩ ?A) finite (p - ?A) (p ∩ ?A)
  ∩ (p - ?A) = {}
  using ptag fine by auto
  have le_xz: ∧ w x y z::real. y ≤ z/2 ⇒ w - x ≤ z/2 ⇒ w + y ≤ x + z
  by arith
  have non: False if xk: (x,K) ∈ p and x ≠ a x ≠ b
  and less: e * (Sup K - Inf K)/2 < norm (content K *R f' x - (f (Sup K)
- f (Inf K)))
  for x K
  proof -
    obtain u v where k: K = cbox u v
    using p(4) xk by blast
    then have u ≤ v and uv: {u, v} ⊆ cbox u v
    using p(2)[OF xk] by auto
    then have result: e * (v - u)/2 < norm ((v - u) *R f' x - (f v - f u))
    using less[unfolded k box_real interval_bounds_real content_real] by auto
    then have x ∈ box a b
    using p(2) p(3) <x ≠ a> <x ≠ b> xk by fastforce
  with d have *: ∧ y. norm (y-x) < d x
  ⇒ norm (f y - f x - (y-x) *R f' x) ≤ e/2 * norm (y-x)

```

```

  by metis
  have xd: norm (u - x) < d x norm (v - x) < d x
    using fineD[OF fine xk] ⟨x ≠ a⟩ ⟨x ≠ b⟩ uv
  by (auto simp add: k subset_eq dist_commute dist_real_def)
  have norm ((v - u) *R f' x - (f v - f u)) =
    norm ((f u - f x - (u - x) *R f' x) - (f v - f x - (v - x) *R f' x))
  by (rule arg_cong[where f=norm]) (auto simp: scaleR_left.diff)
  also have ... ≤ e/2 * norm (u - x) + e/2 * norm (v - x)
  by (metis norm_triangle_le_diff add_mono * xd)
  also have ... ≤ e/2 * norm (v - u)
  using p(2)[OF xk] by (auto simp add: field_simps k)
  also have ... < norm ((v - u) *R f' x - (f v - f u))
  using result by (simp add: ⟨u ≤ v⟩)
  finally have e * (v - u)/2 < e * (v - u)/2
  using uv by auto
  then show False by auto
qed
have norm (∑ (x, K) ∈ p - ?A. content K *R f' x - (f (Sup K) - f (Inf
K)))
  ≤ (∑ (x, K) ∈ p - ?A. norm (content K *R f' x - (f (Sup K) - f (Inf
K))))
  by (auto intro: sum_norm_le)
  also have ... ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k)/2)
  using non by (fastforce intro: sum_mono)
  finally have I: norm (∑ (x, k) ∈ p - ?A. content k *R f' x - (f (Sup k) - f
(Inf k)))
    ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k))/2
  by (simp add: sum_divide_distrib)
  have II: norm (∑ (x, k) ∈ p ∩ ?A. content k *R f' x - (f (Sup k) - f (Inf
k))) -
    (∑ n ∈ p ∩ ?A. e * (case n of (x, k) ⇒ Sup k - Inf k))
    ≤ (∑ n ∈ p - ?A. e * (case n of (x, k) ⇒ Sup k - Inf k))/2
proof -
  have ge0: 0 ≤ e * (Sup k - Inf k) if xkp: (x, k) ∈ p ∩ ?A for x k
  proof -
    obtain u v where uv: k = cbox u v
    by (meson Int_iff xkp p(4))
    with p that have cbox u v ≠ {}
    by blast
    then show 0 ≤ e * ((Sup k) - (Inf k))
    unfolding uv using e by (auto simp add: field_simps)
  qed
  let ?B = λx. {t ∈ p. fst t = x ∧ content (snd t) ≠ 0}
  let ?C = {t ∈ p. fst t ∈ {a, b} ∧ content (snd t) ≠ 0}
  have norm (∑ (x, k) ∈ p ∩ {t. fst t ∈ {a, b}}. content k *R f' x - (f (Sup
k) - f (Inf k))) ≤ e * (b-a)/2
  proof -
    have *: ∧S f e. sum f S = sum f (p ∩ ?C) ⇒ norm (sum f (p ∩ ?C))
    ≤ e ⇒ norm (sum f S) ≤ e

```

```

    by auto
  have 1: content K *_R (f' x) - (f ((Sup K)) - f ((Inf K))) = 0
    if (x,K) ∈ p ∩ {t. fst t ∈ {a, b}} - p ∩ ?C for x K
  proof -
    have xk: (x,K) ∈ p and k0: content K = 0
      using that by auto
    then obtain u v where uv: K = cbox u v u = v
      using xk k0 p by fastforce
    then show content K *_R (f' x) - (f ((Sup K)) - f ((Inf K))) = 0
      using xk unfolding uv by auto
  qed
  have 2: norm (∑ (x,K) ∈ p ∩ ?C. content K *_R f' x - (f (Sup K) - f (Inf
K))) ≤ e * (b-a)/2
  proof -
    have norm_le: norm (sum f S) ≤ e
      if ∧x y. [x ∈ S; y ∈ S] ⇒ x = y ∧x. x ∈ S ⇒ norm (f x) ≤ e e > 0
      for S f and e :: real
    proof (cases S = {})
      case True
        with that show ?thesis by auto
      next
        case False then obtain x where x ∈ S
          by auto
        then have S = {x}
          using that(1) by auto
        then show ?thesis
          using ⟨x ∈ S⟩ that(2) by auto
    qed
    have *: p ∩ ?C = ?B a ∪ ?B b
      by blast
    then have norm (∑ (x,K) ∈ p ∩ ?C. content K *_R f' x - (f (Sup K) -
f (Inf K))) =
      norm (∑ (x,K) ∈ ?B a ∪ ?B b. content K *_R f' x - (f (Sup
K) - f (Inf K)))
      by simp
    also have ... = norm ((∑ (x,K) ∈ ?B a. content K *_R f' x - (f (Sup
K) - f (Inf K))) +
      (∑ (x,K) ∈ ?B b. content K *_R f' x - (f (Sup K) - f
(Inf K))))
      using p(1) ab e by (subst sum.union_disjoint) auto
    also have ... ≤ e * (b - a) / 4 + e * (b - a) / 4
      proof (rule norm_triangle_le [OF add_mono])
        have pa: ∃ v. k = cbox a v ∧ a ≤ v if (a, k) ∈ p for k
          using p that by fastforce
        show norm (∑ (x,K) ∈ ?B a. content K *_R f' x - (f (Sup K) - f
(Inf K))) ≤ e * (b - a) / 4
          proof (intro norm_le; clarsimp)
            fix K K'
            assume K: (a, K) ∈ p (a, K') ∈ p and ne0: content K ≠ 0 content

```

$K' \neq 0$
 with pa obtain $v v'$ where $v: K = \text{cbox } a \ v \ a \leq v$ and $v': K' = \text{cbox } a \ v' \ a \leq v'$
 by *blast*
 let $?v = \min v \ v'$
 have $\text{box } a \ ?v \subseteq K \cap K'$
 unfolding $v \ v'$ by (*auto simp add: mem_box*)
 then have $\text{interior } (\text{box } a \ (\min v \ v')) \subseteq \text{interior } K \cap \text{interior } K'$
 using *interior_Int interior_mono* by *blast*
 moreover have $(a + ?v)/2 \in \text{box } a \ ?v$
 using *ne0 unfolding v v' content_eq_0 not_le*
 by (*auto simp add: mem_box*)
 ultimately have $(a + ?v)/2 \in \text{interior } K \cap \text{interior } K'$
 unfolding *interior_open[OF open_box]* by *auto*
 then show $K = K'$
 using *p(5)[OF K]* by *auto*
 next
 fix K
 assume $K: (a, K) \in p$ and *ne0: content K $\neq 0$*
 show *norm (content c *_R f' a - (f (Sup c) - f (Inf c))) * 4 $\leq e$ **
 (b-a)
 if $(a, c) \in p$ and *ne0: content c $\neq 0$* for c
 proof -
 obtain v where $v: c = \text{cbox } a \ v$ and $a \leq v$
 using *pa[OF <(a, c) $\in p$ >]* by *metis*
 then have $a \in \{a..v\} \ v \leq b$
 using *p(3)[OF <(a, c) $\in p$ >]* by *auto*
 moreover have $\{a..v\} \subseteq \text{ball } a \ da$
 using *fineD[OF <?d fine p> <(a, c) $\in p$ >]* by (*simp add: v split: if_split_asm*)
 ultimately show *?thesis*
 unfolding *v interval_bounds_real[OF <a $\leq v$ >]* *box_real*
 using *da <a $\leq v$ >* by *auto*
 qed
 qed (*use ab e in auto*)
 next
 have *pb: $\exists v. k = \text{cbox } v \ b \ \wedge \ b \geq v$ if $(b, k) \in p$ for k*
 using *p* that by *fastforce*
 show *norm ($\sum (x, K) \in ?B \ b. \text{content } K \ *_{\text{R}} \ f' \ x - (f (\text{Sup } K) - f (\text{Inf } K))$) $\leq e * (b - a) / 4$*
 proof (*intro norm_le; clarsimp*)
 fix $K \ K'$
 assume $K: (b, K) \in p$ $(b, K') \in p$ and *ne0: content K $\neq 0$ content*
 $K' \neq 0$
 with *pb* obtain $v v'$ where $v: K = \text{cbox } v \ b \ v \leq b$ and $v': K' = \text{cbox } v' \ b \ v' \leq b$
 by *blast*
 let $?v = \max v \ v'$
 have $\text{box } ?v \ b \subseteq K \cap K'$

```

      unfolding v v' by (auto simp: mem_box)
    then have interior (box (max v v') b)  $\subseteq$  interior K  $\cap$  interior K'
      using interior_Int interior_mono by blast
    moreover have ((b + ?v)/2)  $\in$  box ?v b
      using ne0 unfolding v v' content_eq_0 not_le by (auto simp:
mem_box)
    ultimately have ((b + ?v)/2)  $\in$  interior K  $\cap$  interior K'
      unfolding interior_open[OF open_box] by auto
    then show K = K'
      using p(5)[OF K] by auto
  next
  fix K
  assume K: (b, K)  $\in$  p and ne0: content K  $\neq$  0
  show norm (content c *_R f' b - (f (Sup c) - f (Inf c))) * 4  $\leq$  e *
(b-a)

    if (b, c)  $\in$  p and ne0: content c  $\neq$  0 for c
  proof -
  obtain v where v: c = cbox v b and v  $\leq$  b
    using  $\langle$ (b, c)  $\in$  p $\rangle$  pb by blast
  then have v  $\geq$  ab  $\in$  {v.. b}
    using p(3)[OF  $\langle$ (b, c)  $\in$  p $\rangle$ ] by auto
  moreover have {v..b}  $\subseteq$  ball b db
    using fineD[OF  $\langle$ ?d fine p $\rangle$   $\langle$ (b, c)  $\in$  p $\rangle$ ] box_real(2) v False by
force
  ultimately show ?thesis
    using db v by auto
  qed
  qed (use ab e in auto)
  qed
  also have ... = e * (b-a)/2
    by simp
  finally show norm ( $\sum$  (x,k) $\in$ p  $\cap$  ?C.
    content k *_R f' x - (f (Sup k) - f (Inf k)))  $\leq$  e * (b-a)/2 .
  qed
  show norm ( $\sum$  (x, k) $\in$ p  $\cap$  ?A. content k *_R f' x - (f ((Sup k)) - f ((Inf
k))))  $\leq$  e * (b-a)/2
    apply (rule * [OF sum_mono_neutral_right[OF pA(2)]])
    using 1 2 by (auto simp: split_paired_all)
  qed
  also have ... = ( $\sum$  n $\in$ p. e * (case n of (x, k)  $\Rightarrow$  Sup k - Inf k))/2
    unfolding sum_distrib_left[symmetric]
    by (subst additive_tagged_division_1[OF  $\langle$ a  $\leq$  b $\rangle$  ptag]) auto
  finally have norm_le: norm ( $\sum$  (x,K) $\in$ p  $\cap$  {t. fst t  $\in$  {a, b}}. content K
*_R f' x - (f (Sup K) - f (Inf K)))
     $\leq$  ( $\sum$  n $\in$ p. e * (case n of (x, K)  $\Rightarrow$  Sup K - Inf K))/2 .
  have le2:  $\bigwedge$  x s1 s2::real. 0  $\leq$  s1  $\implies$  x  $\leq$  (s1 + s2)/2  $\implies$  x - s1  $\leq$  s2/2
    by auto
  show ?thesis
    apply (rule le2 [OF sum_nonneg])

```

```

    using ge0 apply force
    by (metis (no_types, lifting) Diff_Diff_Int Diff_subset norm_le p(1)
sum.subset_diff)
  qed
  note * = additive_tagged_division_1[OF assms(1) ptag, symmetric]
  have norm ( $\sum (x,K) \in p \cap ?A \cup (p - ?A). \text{content } K *_R f' x - (f (Sup K) - f (Inf K))$ )
     $\leq e * (\sum (x,K) \in p \cap ?A \cup (p - ?A). Sup K - Inf K)$ 
    unfolding sum_distrib_left
    unfolding sum_union_disjoint[OF pA(2-)]
    using le_xz norm_triangle_le I II by blast
  then
  show norm ( $(\sum (x,K) \in p. \text{content } K *_R f' x) - (f b - f a)$ )  $\leq e * \text{content } \{a..b\}$ 
    by (simp only: content_real[OF ‹a ≤ b›] *[of λx. x] *[of f] sum_subtractf[symmetric]
split_minus pA(1) [symmetric])
  qed
  qed
  qed

```

7.14.30 Stronger form with finite number of exceptional points

lemma fundamental_theorem_of_calculus_interior_strong:

```

fixes f :: real ⇒ 'a::banach
assumes finite S
  and a ≤ b ∧ x. x ∈ {a <..} - S ⇒ (f has_vector_derivative f'(x)) (at x)
  and continuous_on {a .. b} f
shows (f' has_integral (f b - f a)) {a .. b}
using assms
proof (induction arbitrary: a b)
case empty
  then show ?case
    using fundamental_theorem_of_calculus_interior by force
next
case (insert x S)
  show ?case
  proof (cases x ∈ {a <..})
    case False then show ?thesis
      using insert by blast
  next
    case True then have a < x < b
      by auto
    have (f' has_integral f x - f a) {a..x} (f' has_integral f b - f x) {x..b}
      using ‹continuous_on {a..b} f› ‹a < x› ‹x < b› continuous_on_subset by
(force simp: intro!: insert)+
    then have (f' has_integral f x - f a + (f b - f x)) {a..b}
      using ‹a < x› ‹x < b› has_integral_combine less_imp_le by blast
    then show ?thesis
      by simp
  end
end

```

qed
qed

corollary *fundamental_theorem_of_calculus_strong*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
 assumes *finite S*
 and $a \leq b$
 and *vec*: $\bigwedge x. x \in \{a..b\} - S \implies (f \text{ has_vector_derivative } f'(x)) (at\ x)$
 and *continuous_on* $\{a..b\}$ f
 shows $(f' \text{ has_integral } (f\ b - f\ a)) \{a..b\}$
 by (*rule fundamental_theorem_of_calculus_interior_strong [OF ‹finite S›]*) (*force simp: assms*)+

proposition *indefinite_integral_continuous_left*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$
 assumes *intf*: $f \text{ integrable_on } \{a..b\}$ and $a < c \leq b$ $e > 0$
 obtains d where $d > 0$
 and $\forall t. c - d < t \wedge t \leq c \implies \text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) < e$

proof –

obtain w where $w > 0$ and $w: \bigwedge t. \llbracket c - w < t; t < c \rrbracket \implies \text{norm } (f\ c) * \text{norm}(c - t) < e/3$

proof (*cases* $f\ c = 0$)

case *False*

hence $e/3: 0 < e/3 / \text{norm } (f\ c)$ using $\langle e > 0 \rangle$ by *simp*

moreover have $\text{norm } (f\ c) * \text{norm } (c - t) < e/3$

if $t < c$ and $c - e/3 / \text{norm } (f\ c) < t$ for t

unfolding *real_norm_def*

by (*smt (verit) False divide_right_mono nonzero_mult_div_cancel_left norm_eq_zero norm_ge_zero that*)

ultimately show *?thesis*

using *that* by *auto*

qed (*use* $\langle e > 0 \rangle$ *in auto*)

let $?SUM = \lambda p. (\sum (x,K) \in p. \text{content } K *_{\mathbb{R}} f\ x)$

have $e/3: e/3 > 0$

using $\langle e > 0 \rangle$ by *auto*

have $f \text{ integrable_on } \{a..c\}$

using $\langle a < c \rangle \langle c \leq b \rangle$ by (*auto intro: integrable_subinterval_real [OF intf]*)

then obtain $d1$ where *gauge d1* and $d1$:

$\bigwedge p. \llbracket p \text{ tagged_division_of } \{a..c\}; d1 \text{ fine } p \rrbracket \implies \text{norm } (?SUM\ p - \text{integral } \{a..c\} f) < e/3$

using *integrable_integral_has_integral_real* $e/3$ by *metis*

define d where [*abs_def*]: $d\ x = \text{ball } x\ w \cap d1\ x$ for x

have *gauge d*

unfolding *d_def* using $\langle w > 0 \rangle \langle \text{gauge } d1 \rangle$ by *auto*

then obtain k where $0 < k$ and $k: \text{ball } c\ k \subseteq d\ c$

by (*meson gauge_def open_contains_ball*)


```

let ?d = min k (c - a)/2
show thesis
proof (intro that[of ?d] allI impI, safe)
  show ?d > 0
    using ‹0 < k› ‹a < c› by auto
next
fix t
assume t: c - ?d < t ≤ c
show norm (integral ({a..c}) f - integral ({a..t}) f) < e
proof (cases t < c)
  case False with ‹t ≤ c› show ?thesis
    by (simp add: ‹e > 0›)
next
case True
have f integrable_on {a..t}
  using ‹t < c› ‹c ≤ b› by (auto intro: integrable_subinterval_real[OF intf])
then obtain d2 where d2: gauge d2
  ∧ p. p tagged_division_of {a..t} ∧ d2 fine p ⇒ norm (?SUM p - integral
{a..t} f) < e/3
  using integrable_integral has_integral_real e3 by metis
define d3 where d3 x = (if x ≤ t then d1 x ∩ d2 x else d1 x) for x
have gauge d3
  using ‹gauge d1› ‹gauge d2› unfolding d3_def gauge_def by auto
then obtain p where ptag: p tagged_division_of {a..t} and pfine: d3 fine p
  by (metis box_real(2) fine_division_exists)
note p' = tagged_division_ofD[OF ptag]
have pt: (x,K) ∈ p ⇒ x ≤ t for x K
  by (meson atLeastAtMost_iff p'(2) p'(3) subsetCE)
with pfine have d2 fine p
  unfolding fine_def d3_def by fastforce
then have d2_fin: norm (?SUM p - integral {a..t} f) < e/3
  using d2(2) ptag by auto
have eqs: {a..c} ∩ {x. x ≤ t} = {a..t} {a..c} ∩ {x. x ≥ t} = {t..c}
  using t by (auto simp add: field_simps)
have p ∪ {(c, {t..c})} tagged_division_of {a..c}
proof (intro tagged_division_Un_interval_real)
  show {(c, {t..c})} tagged_division_of {a..c} ∩ {x. t ≤ x < 1}
    using ‹t ≤ c› by (auto simp: eqs tagged_division_of_self_real)
qed (auto simp: eqs ptag)
moreover
have d1 fine p ∪ {(c, {t..c})}
  unfolding fine_def
proof safe
  fix x K y
  assume (x,K) ∈ p and y ∈ K then show y ∈ d1 x
    by (metis Int_iff d3_def subsetD fineD pfine)
next
fix x assume x ∈ {t..c}
then have dist c x < k

```

```

    using t(1) by (auto simp add: field_simps dist_real_def)
    with k show x ∈ d1 c
      unfolding d_def by auto
  qed
  ultimately have d1_fin: norm (?SUM(p ∪ {(c, {t..c})}) - integral {a..c}
f) < e/3
    using d1 by metis
  have SUMEQ: ?SUM(p ∪ {(c, {t..c})}) = (c - t) *R f c + ?SUM p
  proof -
    have ?SUM(p ∪ {(c, {t..c})}) = (content{t..c} *R f c) + ?SUM p
    proof (subst sum.union_disjoint)
      show p ∩ {(c, {t..c})} = {}
        using ⟨t < c⟩ pt by force
    qed (use p'(1) in auto)
    also have ... = (c - t) *R f c + ?SUM p
      using ⟨t ≤ c⟩ by auto
    finally show ?thesis .
  qed
  have c - k < t
    using ⟨k > 0⟩ t(1) by (auto simp add: field_simps)
  moreover have k ≤ w
  proof (rule ccontr)
    assume ¬ k ≤ w
    then have c + (k + w) / 2 ∉ d c
      by (auto simp add: field_simps not_le not_less dist_real_def d_def)
    then have c + (k + w) / 2 ∉ ball c k
      using k by blast
    then show False
      using ⟨0 < w⟩ ⟨¬ k ≤ w⟩ dist_real_def by auto
  qed
  ultimately have cwt: c - w < t
    by (auto simp add: field_simps)
  have eq: integral {a..c} f - integral {a..t} f = -(((c - t) *R f c + ?SUM
p) -
    integral {a..c} f) + (?SUM p - integral {a..t} f) + (c - t) *R f c
    by auto
  have norm (integral {a..c} f - integral {a..t} f) < e/3 + e/3 + e/3
  unfolding eq
  proof (intro norm_triangle_lt add_strict_mono)
    show norm (- ((c - t) *R f c + ?SUM p - integral {a..c} f)) < e/3
      by (metis SUMEQ d1_fin norm_minus_cancel)
    show norm (?SUM p - integral {a..t} f) < e/3
      using d2_fin by blast
    show norm ((c - t) *R f c) < e/3
      using w cwt ⟨t < c⟩ by simp (simp add: field_simps)
  qed
  then show ?thesis by simp
  qed
  qed
  qed

```

qed

lemma *indefinite_integral_continuous_right*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$

assumes $f \text{ integrable_on } \{a..b\}$

and $a \leq c$

and $c < b$

and $e > 0$

obtains d where $0 < d$

and $\forall t. c \leq t \wedge t < c + d \longrightarrow \text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) <$

e

proof -

have $\text{intm}: (\lambda x. f (-x)) \text{ integrable_on } \{-b .. -a\} - b < -c - c \leq -a$

using *assms by auto*

from *indefinite_integral_continuous_left*[OF *intm* $\langle e > 0 \rangle$]

obtain d where $0 < d$

and $d: \bigwedge t. \llbracket -c - d < t; t \leq -c \rrbracket$

$\implies \text{norm } (\text{integral } \{-b..-c\} (\lambda x. f (-x)) - \text{integral } \{-b..t\} (\lambda x. f$

$(-x))) < e$

by *metis*

let $?d = \min d (b - c)$

show *?thesis*

proof (*intro that*[of $?d$] *allI impI*)

show $0 < ?d$

using $\langle 0 < d \rangle \langle c < b \rangle$ by *auto*

fix $t :: \text{real}$

assume $t: c \leq t \wedge t < c + ?d$

have $*$: $\text{integral } \{a..c\} f = \text{integral } \{a..b\} f - \text{integral } \{c..b\} f$

$\text{integral } \{a..t\} f = \text{integral } \{a..b\} f - \text{integral } \{t..b\} f$

using *assms t* by (*auto simp: algebra_simps integral_combine*)

have $(-c) - d < (-t) - t \leq -c$

using t by *auto*

from d [OF *this*] show $\text{norm } (\text{integral } \{a..c\} f - \text{integral } \{a..t\} f) < e$

by (*auto simp add: algebra_simps norm_minus_commute **)

qed

qed

lemma *indefinite_integral_continuous_1*:

fixes $f :: \text{real} \Rightarrow 'a::\text{banach}$

assumes $f \text{ integrable_on } \{a..b\}$

shows *continuous_on* $\{a..b\} (\lambda x. \text{integral } \{a..x\} f)$

proof -

have $\exists d > 0. \forall x' \in \{a..b\}. \text{dist } x' x < d \longrightarrow \text{dist } (\text{integral } \{a..x'\} f) (\text{integral } \{a..x\} f) < e$

if $x: x \in \{a..b\}$ and $e > 0$ for $x e :: \text{real}$

proof (*cases a = b*)

case *True*

with *that* show *?thesis* by *force*

next

```

    case False
    with x have  $a < b$  by force
    with x consider  $x = a \mid x = b \mid a < x < b$ 
      by force
    then show ?thesis
    proof cases
      case 1 then show ?thesis
      by (force simp: dist_norm_algebra_simps intro: indefinite_integral_continuous_right
        [OF assms _  $\langle a < b \rangle \langle e > 0 \rangle$ ])
      next
      case 2 then show ?thesis
      by (force simp: dist_norm_norm_minus_commute_algebra_simps intro:
        indefinite_integral_continuous_left [OF assms  $\langle a < b \rangle$  _  $\langle e > 0 \rangle$ ])
      next
      case 3
      obtain d1 where  $0 < d1$ 
        and d1:  $\bigwedge t. \llbracket x - d1 < t; t \leq x \rrbracket \implies \text{norm } (\text{integral } \{a..x\} f - \text{integral } \{a..t\} f) < e$ 
        using 3 by (auto intro: indefinite_integral_continuous_left [OF assms  $\langle a < x \rangle$  _  $\langle e > 0 \rangle$ ])
      obtain d2 where  $0 < d2$ 
        and d2:  $\bigwedge t. \llbracket x \leq t; t < x + d2 \rrbracket \implies \text{norm } (\text{integral } \{a..x\} f - \text{integral } \{a..t\} f) < e$ 
        using 3 by (auto intro: indefinite_integral_continuous_right [OF assms _  $\langle x < b \rangle \langle e > 0 \rangle$ ])
      show ?thesis
      proof (intro exI ballI conjI impI)
        show  $0 < \min d1 d2$ 
          using  $\langle 0 < d1 \rangle \langle 0 < d2 \rangle$  by simp
        show  $\text{dist } (\text{integral } \{a..y\} f) (\text{integral } \{a..x\} f) < e$ 
          if  $\text{dist } y x < \min d1 d2$  for y
            by (smt (verit) d1 d2 dist_norm dist_real_def norm_minus_commute
              that)
        qed
      qed
    then show ?thesis
      by (auto simp: continuous_on_iff)
    qed

lemma indefinite_integral_continuous_1':
  fixes f::real  $\Rightarrow$  'a::banach
  assumes f integrable_on  $\{a..b\}$ 
  shows continuous_on  $\{a..b\}$   $(\lambda x. \text{integral } \{x..b\} f)$ 
proof -
  have  $\text{integral } \{a..b\} f - \text{integral } \{a..x\} f = \text{integral } \{x..b\} f$  if  $x \in \{a..b\}$  for x
    using integral_combine[OF _ _ assms, of x] that
    by (auto simp: algebra_simps)
  with _ show ?thesis

```

```

  by (rule continuous_on_eq) (auto intro!: continuous_intros indefinite_integral_continuous_1
  assms)
qed

```

theorem *integral_has_vector_derivative'*:

```

  fixes f :: real  $\Rightarrow$  'b::banach
  assumes continuous_on {a..b} f
  and x  $\in$  {a..b}
  shows (( $\lambda$ x. integral {u..b} f) has_vector_derivative - f x) (at x within {a..b})
proof -
  have *: integral {x..b} f = integral {a .. b} f - integral {a .. x} f if a  $\leq$  x  $\leq$ 
  b for x
  using integral_combine[of a x b for x, OF that integrable_continuous_real[OF
  assms(1)]]
  by (simp add: algebra_simps)
  show ?thesis
  using <x  $\in$  _> *
  by (rule has_vector_derivative_transform)
  (auto intro!: derivative_eq_intros assms integral_has_vector_derivative)
qed

```

lemma *integral_has_real_derivative'*:

```

  assumes continuous_on {a..b} g
  assumes t  $\in$  {a..b}
  shows (( $\lambda$ x. integral {x..b} g) has_real_derivative -g t) (at t within {a..b})
  using integral_has_vector_derivative'[OF assms]
  by (auto simp: has_real_derivative_iff_has_vector_derivative)

```

7.14.31 This doesn't directly involve integration, but that gives an easy proof

lemma *has_derivative_zero_unique_strong_interval*:

```

  fixes f :: real  $\Rightarrow$  'a::banach
  assumes finite k
  and contf: continuous_on {a..b} f
  and f a = y
  and fder:  $\bigwedge$ x. x  $\in$  {a..b} - k  $\implies$  (f has_derivative ( $\lambda$ h. 0)) (at x within {a..b})
  and x: x  $\in$  {a..b}
  shows f x = y
proof -
  have a  $\leq$  b a  $\leq$  x
  using assms by auto
  have (( $\lambda$ x. 0::'a) has_integral f x - f a) {a..x}
proof (rule fundamental_theorem_of_calculus_interior_strong[OF <finite k> <a
 $\leq$  x>]; clarify?)
  have {a..x}  $\subseteq$  {a..b}
  using x by auto
  then show continuous_on {a..x} f
  by (rule continuous_on_subset[OF contf])

```

```

  show (f has_vector_derivative 0) (at z) if z: z ∈ {a<..

```

7.14.32 Generalize a bit to any convex set

```

lemma has_derivative_zero_unique_strong_convex:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes convex S finite K
  and contf: continuous_on S f
  and c ∈ S f c = y
  and derf: ∀x. x ∈ S - K ⇒ (f has_derivative (λh. 0)) (at x within S)
  and x ∈ S
  shows f x = y
proof (cases x = c)
  case True with ⟨f c = y⟩ show ?thesis
    by blast
  next
  case False
  let ?φ = λu. (1 - u) *R c + u *R x
  have contf': continuous_on {0 ..1} (f ∘ ?φ)
  proof (rule continuous_intros continuous_on_subset[OF contf])+
    show (λu. (1 - u) *R c + u *R x) ' {0..1} ⊆ S
      using ⟨convex S⟩ ⟨x ∈ S⟩ ⟨c ∈ S⟩ by (auto simp add: convex_alt algebra_simps)
  qed
  have t = u if ?φ t = ?φ u for t u
  proof -
    from that have (t - u) *R x = (t - u) *R c
      by (auto simp add: algebra_simps)
    then show ?thesis
      using ⟨x ≠ c⟩ by auto
  qed
  then have eq: (SOME t. ?φ t = ?φ u) = u for u
    by blast
  then have (?φ - ' K) ⊆ (λz. SOME t. ?φ t = z) ' K
    by (clarsimp simp: image_iff) (metis (no_types) eq)
  then have fin: finite (?φ - ' K)
    by (rule finite_surj[OF ⟨finite K⟩])

```

```

have derf': (( $\lambda u. f$  ( $? \varphi u$ )) has_derivative ( $\lambda h. 0$ )) (at  $t$  within  $\{0..1\}$ )
  if  $t \in \{0..1\} - \{t. ? \varphi t \in K\}$  for  $t$ 
proof -
  have df: ( $f$  has_derivative ( $\lambda h. 0$ )) (at ( $? \varphi t$ ) within  $? \varphi \text{ ` } \{0..1\}$ )
    using  $\langle \text{convex } S \rangle \langle x \in S \rangle \langle c \in S \rangle$  that
    by (auto simp add: convex_alt algebra_simps intro: has_derivative_subset
[OF derf])
  have ( $f \circ ? \varphi$  has_derivative ( $\lambda x. 0$ )  $\circ$  ( $\lambda z. (0 - z *_R c) + z *_R x$ )) (at  $t$  within
 $\{0..1\}$ )
    by (rule derivative_eq_intros df | simp)+
  then show ?thesis
    unfolding o_def .
qed
have ( $f \circ ? \varphi$ ) 1 =  $y$ 
  apply (rule has_derivative_zero_unique_strong_interval[OF fin contf'])
  unfolding o_def using  $\langle f c = y \rangle$  derf' by auto
then show ?thesis
  by auto
qed

```

Also to any open connected set with finite set of exceptions. Could generalize to locally convex set with limpt-free set of exceptions.

lemma has_derivative_zero_unique_strong_connected:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::banach$ 
assumes connected  $S$ 
  and open  $S$ 
  and finite  $K$ 
  and contf: continuous_on  $S f$ 
  and  $c \in S$ 
  and  $f c = y$ 
  and derf:  $\bigwedge x. x \in S - K \implies (f$  has_derivative ( $\lambda h. 0$ )) (at  $x$  within  $S$ )
  and  $x \in S$ 
shows  $f x = y$ 
proof -
  have  $\exists e > 0. \text{ball } x e \subseteq (S \cap f \text{ - } \{f x\})$  if  $x \in S$  for  $x$ 
  proof -
    obtain  $e$  where  $0 < e$  and  $e: \text{ball } x e \subseteq S$ 
      using  $\langle x \in S \rangle \langle \text{open } S \rangle$  open_contains_ball by blast
    have  $\text{ball } x e \subseteq \{u \in S. f u \in \{f x\}\}$ 
  proof safe
    fix  $y$ 
    assume  $y: y \in \text{ball } x e$ 
    then show  $y \in S$ 
      using  $e$  by auto
    show  $f y = f x$ 
  proof (rule has_derivative_zero_unique_strong_convex[OF convex_ball  $\langle$ finite  $K \rangle$ ])
    show continuous_on ( $\text{ball } x e$ )  $f$ 
      using contf continuous_on_subset  $e$  by blast

```

```

    show (f has_derivative (λh. 0)) (at u within ball x e)
      if u ∈ ball x e - K for u
      by (metis Diff_iff contra_subsetD derf e has_derivative_subset that)
    qed (use y e ⟨0 < e⟩ in auto)
  qed
  then show ∃ e>0. ball x e ⊆ (S ∩ f -' {f x})
    using ⟨0 < e⟩ by blast
  qed
  then have openin (top_of_set S) (S ∩ f -' {y})
    by (auto intro!: open_openin_trans[OF ⟨open S⟩] simp: open_contains_ball)
  moreover have closedin (top_of_set S) (S ∩ f -' {y})
    by (force intro!: continuous_closedin_preimage [OF contf])
  ultimately have (S ∩ f -' {y}) = {} ∨ (S ∩ f -' {y}) = S
    using ⟨connected S⟩ by (simp add: connected_clopen)
  then show ?thesis
    using ⟨x ∈ S⟩ ⟨f c = y⟩ ⟨c ∈ S⟩ by auto
  qed

lemma has_derivative_zero_connected_constant_on:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes connected S open S finite K continuous_on S f
    and ∀ x∈S-K. (f has_derivative (λh. 0)) (at x within S)
  shows f constant_on S
  by (smt (verit, best) assms constant_on_def has_derivative_zero_unique_strong_connected)

lemma DERIV_zero_connected_constant_on:
  fixes f :: 'a::{real_normed_field,euclidean_space} ⇒ 'a
  assumes *: connected S open S finite K continuous_on S f
    and 0: ∀ x∈S-K. DERIV f x :> 0
  shows f constant_on S
  using has_derivative_zero_connected_constant_on [OF *] 0
  by (metis has_derivative_at_withinI has_field_derivative_def lambda_zero)

lemma DERIV_zero_connected_constant:
  fixes f :: 'a::{real_normed_field,euclidean_space} ⇒ 'a
  assumes connected S and open S and finite K and continuous_on S f
    and ∀ x∈S-K. DERIV f x :> 0
  obtains c where ∧x. x ∈ S ⇒ f(x) = c
  by (metis DERIV_zero_connected_constant_on [OF assms] constant_on_def)

lemma has_field_derivative_0_imp_constant_on:
  fixes f :: 'a::{real_normed_field,euclidean_space} ⇒ 'a
  assumes ∧z. z ∈ S ⇒ (f has_field_derivative 0) (at z) and S: connected S
  open S
  shows f constant_on S
  using DERIV_zero_connected_constant_on [where K=Basis]
  by (metis DERIV_isCont Diff_iff assms continuous_at_imp_continuous_on
  eucl.finite_Basis)

```


7.14.33 Integrating characteristic function of an interval

```

lemma has_integral_restrict_open_subinterval:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::banach$ 
  assumes intf:  $(f \text{ has\_integral } i) (cbox\ c\ d)$ 
    and cb:  $cbox\ c\ d \subseteq cbox\ a\ b$ 
  shows  $((\lambda x. \text{if } x \in cbox\ c\ d \text{ then } f\ x \text{ else } 0) \text{ has\_integral } i) (cbox\ a\ b)$ 
proof (cases  $cbox\ c\ d = \{\}$ )
  case True
  then have  $cbox\ c\ d = \{\}$ 
    by (metis bot.extremum_uniqueI box_subset_cbox)
  then show ?thesis
    using True intf by auto
  next
  case False
  then obtain p where pdiv:  $p$  division_of  $cbox\ a\ b$  and inp:  $cbox\ c\ d \in p$ 
    using cb partial_division_extend_1 by blast
  define g where [abs_def]:  $g\ x = (\text{if } x \in cbox\ c\ d \text{ then } f\ x \text{ else } 0)$  for  $x$ 
  interpret operative lift_option_plus  $Some\ (0 :: 'b)$ 
     $\lambda i. \text{if } g \text{ integrable\_on } i \text{ then } Some\ (\text{integral } i\ g) \text{ else } None$ 
  by (fact operative_integralI)
  note operat = division [OF pdiv, symmetric]
  show ?thesis
  proof (cases  $(g \text{ has\_integral } i) (cbox\ a\ b)$ )
  case True then show ?thesis
    by (simp add: g_def)
  next
  case False
  have iterate:  $F (\lambda i. \text{if } g \text{ integrable\_on } i \text{ then } Some\ (\text{integral } i\ g) \text{ else } None) (p - \{cbox\ c\ d\}) = Some\ 0$ 
  proof (intro neutral_ballI)
  fix  $x$ 
  assume  $x: x \in p - \{cbox\ c\ d\}$ 
  then have  $x \in p$ 
    by auto
  then obtain  $u\ v$  where  $uv: x = cbox\ u\ v$ 
    using pdiv by blast
  have  $\text{interior } x \cap \text{interior } (cbox\ c\ d) = \{\}$ 
    using pdiv inp  $x$  by blast
  then have  $(g \text{ has\_integral } 0)\ x$ 
    unfolding uv using has_integral_spike_interior[where  $f = \lambda x. 0$ ]
  by (metis (no_types, lifting) disjoint_iff_not_equal g_def has_integral_0_eq_interior_cbox)
  then show  $(\text{if } g \text{ integrable\_on } x \text{ then } Some\ (\text{integral } x\ g) \text{ else } None) = Some\ 0$ 
    by auto
  qed
  interpret comm_monoid_set lift_option_plus  $Some\ (0 :: 'b)$ 
  by (intro comm_monoid_set.intro comm_monoid_lift_option add.comm_monoid_axioms)
  have intg:  $g \text{ integrable\_on } cbox\ c\ d$ 

```

```

    using integrable_spike_interior[where f=f]
    by (meson g_def has_integral_integrable intf)
  moreover have integral (cbox c d) g = i
    by (meson g_def has_integral_iff has_integral_spike_interior intf)
  ultimately have F ( $\lambda A. \text{if } g \text{ integrable\_on } A \text{ then Some (integral } A \text{ } g) \text{ else None}$ ) p = Some i
    by (metis (full_types, lifting) division_of_finite inp iterate pdiv remove_right_neutral)
  with False show ?thesis
    by (metis integrable_integral not_None_eq operat option.inject)
qed
qed

```

```

lemma has_integral_restrict_closed_subinterval:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes (f has_integral i) (cbox c d)
    and cbox c d  $\subseteq$  cbox a b
  shows (( $\lambda x. \text{if } x \in \text{cbox } c \text{ } d \text{ then } f \text{ } x \text{ else } 0$ ) has_integral i) (cbox a b)
proof -
  note has_integral_restrict_open_subinterval[OF assms]
  note * = has_integral_spike[OF negligible_frontier_interval _ this]
  show ?thesis
    by (rule *[of c d]) (use box_subset_cbox[of c d] in auto)
qed

```

```

lemma has_integral_restrict_closed_subintervals_eq:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes cbox c d  $\subseteq$  cbox a b
  shows (( $\lambda x. \text{if } x \in \text{cbox } c \text{ } d \text{ then } f \text{ } x \text{ else } 0$ ) has_integral i) (cbox a b)  $\longleftrightarrow$  (f
has_integral i) (cbox c d)
  (is ?l = ?r)
proof (cases cbox c d = {})
case False
let ?g =  $\lambda x. \text{if } x \in \text{cbox } c \text{ } d \text{ then } f \text{ } x \text{ else } 0$ 
show ?thesis
proof
  assume ?l
  then have ?g integrable_on cbox c d
    using assms has_integral_integrable_integrable_subinterval by blast
  then have f integrable_on cbox c d
    by (rule integrable_eq) auto
  moreover then have i = integral (cbox c d) f
    by (meson <(( $\lambda x. \text{if } x \in \text{cbox } c \text{ } d \text{ then } f \text{ } x \text{ else } 0$ ) has_integral i) (cbox a b)> assms has_integral_restrict_closed_subinterval has_integral_unique_integrable_integral)
  ultimately show ?r by auto
next
  assume ?r then show ?l

```

```

    by (rule has_integral_restrict_closed_subinterval[OF _ assms])
  qed
qed auto

```

Hence we can apply the limit process uniformly to all integrals.

lemma *has_integral'*:

```

  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  shows (f has_integral i) S  $\longleftrightarrow$ 
    ( $\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
      ( $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f(x) \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm}(z -$ 
        i) < e))
    (is ?l  $\longleftrightarrow$  ( $\forall e > 0. \exists r e$ ))
  proof (cases  $\exists a b. S = \text{cbox } a b$ )
  case False then show ?thesis
    by (simp add: has_integral_alt)
  next
  case True
  then obtain a b where S: S = cbox a b
    by blast
  obtain B where 0 < B and B:  $\bigwedge x. x \in \text{cbox } a b \implies \text{norm } x \leq B$ 
    using bounded_cbox[unfolded bounded_pos] by blast
  show ?thesis
  proof safe
    fix e :: real
    assume ?l and e > 0
    have (( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) has_integral i) (cbox c d)
      if ball 0 (B+1)  $\subseteq$  cbox c d for c d
      unfolding S using B that
      by (force intro: <?l>[unfolded S] has_integral_restrict_closed_subinterval)
    then show ?r e
      by (meson <0 < B> <0 < e> add_pos_pos le_less_trans zero_less_one
        norm_pths(2))
  next
  assume as:  $\forall e > 0. \exists r e$ 
  then obtain C
    where C:  $\bigwedge a b. \text{ball } 0 C \subseteq \text{cbox } a b \implies$ 
       $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b)$ 
    by (meson zero_less_one)
  define c :: 'n where c = ( $\sum_{i \in \text{Basis}} (- \max B C) *_{\mathbb{R}} i$ )
  define d :: 'n where d = ( $\sum_{i \in \text{Basis}} \max B C *_{\mathbb{R}} i$ )
  have c  $\cdot$  i  $\leq$  x  $\cdot$  i  $\wedge$  x  $\cdot$  i  $\leq$  d  $\cdot$  i if norm x  $\leq$  B i  $\in$  Basis for x i
    using that and Basis_le_norm[OF <i  $\in$  Basis>, of x]
    by (auto simp add: field_simps sum_negf c_def d_def)
  then have c_d: cbox a b  $\subseteq$  cbox c d
    by (meson B mem_box(2) subsetI)
  have c  $\cdot$  i  $\leq$  x  $\cdot$  i  $\wedge$  x  $\cdot$  i  $\leq$  d  $\cdot$  i
  if x: norm (0 - x) < C and i: i  $\in$  Basis for x i
    using Basis_le_norm[OF i, of x] x i by (auto simp: sum_negf c_def d_def)
  then have ball 0 C  $\subseteq$  cbox c d

```

```

    by (auto simp: mem_box dist_norm)
  with C obtain y where y: (f has_integral y) (cbox a b)
  using c_d has_integral_restrict_closed_subintervals_eq S by blast
  have y = i
  proof (rule ccontr)
    assume y ≠ i
    then have 0 < norm (y - i)
      by auto
    from as[rule_format, OF this]
    obtain C where C:  $\bigwedge a b. \text{ball } 0 C \subseteq \text{cbox } a b \implies$ 
       $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - i)$ 
      < norm (y - i)
    by auto
    define c :: 'n where c =  $(\sum_{i \in \text{Basis}} (- \max B C) *_{\mathbb{R}} i)$ 
    define d :: 'n where d =  $(\sum_{i \in \text{Basis}} \max B C *_{\mathbb{R}} i)$ 
    have c · i ≤ x · i ∧ x · i ≤ d · i
      if norm x ≤ B and i ∈ Basis for x i
      using that Basis_le_norm[of i x] by (auto simp add: field_simps sum_negf
c_def d_def)
    then have c_d: cbox a b ⊆ cbox c d
      by (simp add: B mem_box(2) subset_eq)
    have c · i ≤ x · i ∧ x · i ≤ d · i if norm (0 - x) < C and i ∈ Basis for x i
      using Basis_le_norm[of i x] that by (auto simp: sum_negf c_def d_def)
    then have ball 0 C ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    with C obtain z where z: (f has_integral z) (cbox a b) norm (z - i) < norm
(y - i)
    using has_integral_restrict_closed_subintervals_eq[OF c_d] S by blast
    moreover then have z = y
      by (blast intro: has_integral_unique[OF _ y])
    ultimately show False
      by auto
  qed
  then show ?l
    using y by (auto simp: S)
  qed
qed

```

```

lemma has_integral_le:
  fixes f :: 'n::euclidean_space ⇒ real
  assumes fg: (f has_integral i) S (g has_integral j) S
  and le:  $\bigwedge x. x \in S \implies f x \leq g x$ 
  shows i ≤ j
  using has_integral_component_le[OF _ fg, of 1] le by auto

```

```

lemma integral_le:
  fixes f :: 'n::euclidean_space ⇒ real
  assumes f integrable_on S
  and g integrable_on S

```

and $\bigwedge x. x \in S \implies f x \leq g x$
shows $\text{integral } S f \leq \text{integral } S g$
by (*meson assms has_integral_le integrable_integral*)

lemma *has_integral_nonneg*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow \text{real}$
assumes $(f \text{ has_integral } i) S$ **and** $\bigwedge x. x \in S \implies 0 \leq f x$
shows $0 \leq i$
using *assms has_integral_0 has_integral_le* **by** *blast*

lemma *integral_nonneg*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow \text{real}$
assumes $f: f \text{ integrable_on } S$ **and** $0: \bigwedge x. x \in S \implies 0 \leq f x$
shows $0 \leq \text{integral } S f$
by (*rule has_integral_nonneg[OF f[unfolded has_integral_integral] 0]*)

Hence a general restriction property.

lemma *has_integral_restrict [simp]*:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{banach}$
assumes $S \subseteq T$
shows $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has_integral } i) T \longleftrightarrow (f \text{ has_integral } i) S$
proof –
have $*$: $\bigwedge x. (\text{if } x \in T \text{ then if } x \in S \text{ then } f x \text{ else } 0 \text{ else } 0) = (\text{if } x \in S \text{ then } f x \text{ else } 0)$
using *assms* **by** *auto*
show *?thesis*
apply (*subst(2) has_integral'*)
apply (*subst has_integral'*)
apply (*simp add: **)
done

qed

corollary *has_integral_restrict_UNIV*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
shows $((\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ has_integral } i) \text{ UNIV} \longleftrightarrow (f \text{ has_integral } i) s$
by *auto*

lemma *has_integral_restrict_Int*:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{banach}$
shows $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has_integral } i) T \longleftrightarrow (f \text{ has_integral } i) (S \cap T)$
proof –
have $((\lambda x. \text{if } x \in T \text{ then if } x \in S \text{ then } f x \text{ else } 0 \text{ else } 0) \text{ has_integral } i) \text{ UNIV} = ((\lambda x. \text{if } x \in S \cap T \text{ then } f x \text{ else } 0) \text{ has_integral } i) \text{ UNIV}$
by (*rule has_integral_cong*) *auto*
then show *?thesis*
using *has_integral_restrict_UNIV* **by** *fastforce*
qed

lemma *integral_restrict_Int*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
shows $integral\ T\ (\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) = integral\ (S \cap T)\ f$
by (*metis* (*no_types*, *lifting*) *has_integral_cong has_integral_restrict_Int integrable_integral integral_unique not_integrable_integral*)

lemma *integrable_restrict_Int*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
shows $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ integrable_on } T \longleftrightarrow f \text{ integrable_on } (S \cap T)$
using *has_integral_restrict_Int* **by** *fastforce*

lemma *has_integral_on_superset*:

fixes $f :: 'n::euclidean_space \Rightarrow 'a::banach$
assumes (*f has_integral i*) S
and $\bigwedge x. x \notin S \implies f\ x = 0$
and $S \subseteq T$
shows (*f has_integral i*) T
by (*smt* (*verit*, *ccfv_SIG*) *assms has_integral_cong has_integral_restrict*)

lemma *integrable_on_superset*:

fixes $f :: 'n::euclidean_space \Rightarrow 'a::banach$
assumes $f \text{ integrable_on } S$ **and** $\bigwedge x. x \notin S \implies f\ x = 0$ **and** $S \subseteq t$
shows $f \text{ integrable_on } t$
by (*meson* *assms has_integral_on_superset integrable_integral integrable_on_def*)

lemma *integral_subset_negligible*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
assumes $S \subseteq T$ *negligible* $(T - S)$
shows $integral\ S\ f = integral\ T\ f$

proof –

have $integral\ T\ f = integral\ T\ (\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0)$
by (*rule* *integral_spike*[*of* $T - S$]) (*use* *assms* **in** *auto*)
also **have** $\dots = integral\ (S \cap T)\ f$
by (*subst* *integral_restrict_Int*) *auto*
also **have** $S \cap T = S$ **using** *assms* **by** *auto*
finally **show** *?thesis* **..**

qed

lemma *integral_restrict_UNIV*:

fixes $f :: 'n::euclidean_space \Rightarrow 'a::banach$
shows $integral\ UNIV\ (\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) = integral\ S\ f$
by (*simp* *add: integral_restrict_Int*)

lemma *integrable_restrict_UNIV*:

fixes $f :: 'n::euclidean_space \Rightarrow 'a::banach$
shows $(\lambda x. \text{if } x \in s \text{ then } f\ x \text{ else } 0) \text{ integrable_on } UNIV \longleftrightarrow f \text{ integrable_on } s$
unfolding *integrable_on_def*

by auto

lemma *has_integral_subset_component_le*:

fixes $f :: 'n::euclidean_space \Rightarrow 'm::euclidean_space$

assumes $k: k \in \text{Basis}$

and $S \subseteq T$ (f has_integral i) S (f has_integral j) $T \wedge x. x \in T \implies 0 \leq f(x) \cdot k$

shows $i \cdot k \leq j \cdot k$

proof –

have \S : $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has_integral } i) \text{ UNIV}$

$((\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ has_integral } j) \text{ UNIV}$

by (*simp_all* add: *assms*)

show ?thesis

using *assms* by (*force intro!*: *has_integral_component_le*[*OF k §*])

qed

7.14.34 Integrals on set differences

lemma *has_integral_setdiff*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$

assumes $S: (f \text{ has_integral } i) S$ and $T: (f \text{ has_integral } j) T$

and $neg: \text{negligible } (T - S)$

shows $(f \text{ has_integral } (i - j)) (S - T)$

proof –

show ?thesis

unfolding *has_integral_restrict_UNIV* [*symmetric, of f*]

proof (*rule has_integral_spike* [*OF neg*])

have *eq*: $(\lambda x. (\text{if } x \in S \text{ then } f x \text{ else } 0) - (\text{if } x \in T \text{ then } f x \text{ else } 0)) =$

$(\lambda x. \text{if } x \in T - S \text{ then } - f x \text{ else if } x \in S - T \text{ then } f x \text{ else } 0)$

by (*force simp add:*)

have $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has_integral } i) \text{ UNIV}$

$((\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ has_integral } j) \text{ UNIV}$

using $S T \text{ has_integral_restrict_UNIV}$ by *auto*

from *has_integral_diff* [*OF this*]

show $((\lambda x. \text{if } x \in T - S \text{ then } - f x \text{ else if } x \in S - T \text{ then } f x \text{ else } 0)$

$\text{has_integral } i - j) \text{ UNIV}$

by (*simp add:* *eq*)

qed *force*

qed

lemma *integral_setdiff*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$

assumes $f \text{ integrable_on } S$ $f \text{ integrable_on } T$ $\text{negligible}(T - S)$

shows $\text{integral } (S - T) f = \text{integral } S f - \text{integral } T f$

by (*rule integral_unique*) (*simp add:* *assms has_integral_setdiff integrable_integral*)

lemma *integrable_setdiff*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$

assumes $(f \text{ has_integral } i) S$ $(f \text{ has_integral } j) T$ $\text{negligible } (T - S)$

shows $f \text{ integrable_on } (S - T)$

2660

using *has_integral_setdiff* [*OF assms*]
by (*simp add: has_integral_iff*)

lemma *negligible_setdiff* [*simp*]: $T \subseteq S \implies \text{negligible } (T - S)$
by (*metis Diff_eq_empty_iff negligible_empty*)

lemma *negligible_on_intervals*: $\text{negligible } s \iff (\forall a b. \text{negligible}(s \cap \text{cbox } a b))$
(**is** ?l \iff ?r)

proof

assume *R*: ?r

show ?l

unfolding *negligible_def*

by (*metis Diff_iff Int_iff R has_integral_negligible indicator_simps(2)*)

qed (*simp add: negligible_Int*)

lemma *negligible_translation*:

assumes *negligible S*

shows *negligible ((+) c ' S)*

proof –

have *inj*: *inj* ((+) *c*)

by *simp*

show ?thesis

using *assms*

proof (*clarsimp simp: negligible_def*)

fix *a b*

assume $\forall x y. (\text{indicator } S \text{ has_integral } 0) (\text{cbox } x y)$

then have *: $(\text{indicator } S \text{ has_integral } 0) (\text{cbox } (a-c) (b-c))$

by (*meson Diff_iff assms has_integral_negligible indicator_simps(2)*)

have *eq*: $\text{indicator } ((+) c ' S) = (\lambda x. \text{indicator } S (x - c))$

by (*force simp add: indicator_def*)

show $(\text{indicator } ((+) c ' S) \text{ has_integral } 0) (\text{cbox } a b)$

using *has_integral_affinity* [*OF* *, *of* 1 -*c*]

cbox_translation [*of* *c -c+a -c+b*]

by (*simp add: eq*) (*simp add: ac_simps*)

qed

qed

lemma *negligible_translation_rev*:

assumes *negligible ((+) c ' S)*

shows *negligible S*

by (*metis negligible_translation* [*OF* *assms*, *of* -*c*] *translation_galois*)

lemma *negligible_atLeastAtMostI*: $b \leq a \implies \text{negligible } \{a..(b::\text{real})\}$

using *negligible_insert* **by** *fastforce*

lemma *has_integral_spike_set_eq*:

fixes *f* :: '*n*::euclidean_space \Rightarrow '*a*::banach

assumes *negligible* $\{x \in S - T. f x \neq 0\}$ *negligible* $\{x \in T - S. f x \neq 0\}$

shows $(f \text{ has_integral } y) S \iff (f \text{ has_integral } y) T$

proof –

have $((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has_integral } y) \text{ UNIV} =$
 $((\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ has_integral } y) \text{ UNIV}$
proof (rule has_integral_spike_eq)
show negligible $(\{x \in S - T. f x \neq 0\} \cup \{x \in T - S. f x \neq 0\})$
by (rule negligible_Un [OF assms])
qed auto
then show ?thesis
by (simp add: has_integral_restrict_UNIV)

qed

corollary integral_spike_set:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
assumes negligible $\{x \in S - T. f x \neq 0\}$ negligible $\{x \in T - S. f x \neq 0\}$
shows integral $S f = \text{integral } T f$
using has_integral_spike_set_eq [OF assms]
by (metis eq_integralD integral_unique)

lemma integrable_spike_set:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
assumes $f: f \text{ integrable_on } S$ **and** $\text{neg: negligible } \{x \in S - T. f x \neq 0\}$ negligible
 $\{x \in T - S. f x \neq 0\}$
shows $f \text{ integrable_on } T$
using has_integral_spike_set_eq [OF neg] f **by** blast

lemma integrable_spike_set_eq:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
assumes negligible $((S - T) \cup (T - S))$
shows $f \text{ integrable_on } S \longleftrightarrow f \text{ integrable_on } T$
by (blast intro: integrable_spike_set assms negligible_subset)

lemma integrable_on_insert_iff: $f \text{ integrable_on } (\text{insert } x X) \longleftrightarrow f \text{ integrable_on } X$

for $f :: _ \Rightarrow 'a::\text{banach}$
by (rule integrable_spike_set_eq) (auto simp: insert_Diff_if)

lemma has_integral_interior:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{banach}$
shows negligible(frontier S) $\implies (f \text{ has_integral } y) (\text{interior } S) \longleftrightarrow (f \text{ has_integral } y) S$
by (rule has_integral_spike_set_eq [OF empty_imp_negligible negligible_subset])
 (use interior_subset **in** ‹auto simp: frontier_def closure_def›)

lemma has_integral_closure:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{banach}$
shows negligible(frontier S) $\implies (f \text{ has_integral } y) (\text{closure } S) \longleftrightarrow (f \text{ has_integral } y) S$
by (rule has_integral_spike_set_eq [OF negligible_subset empty_imp_negligible])
 (auto simp: closure_Un_frontier)

lemma *has_integral_open_interval*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
shows $(f \text{ has_integral } y) (box\ a\ b) \longleftrightarrow (f \text{ has_integral } y) (cbox\ a\ b)$
unfolding *interior_cbox [symmetric]*
by $(metis\ frontier_cbox\ has_integral_interior\ negligible_frontier_interval)$

lemma *integrable_on_open_interval*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
shows $f \text{ integrable_on } box\ a\ b \longleftrightarrow f \text{ integrable_on } cbox\ a\ b$
by $(simp\ add:\ has_integral_open_interval\ integrable_on_def)$

lemma *integral_open_interval*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: banach$
shows $integral(box\ a\ b)\ f = integral(cbox\ a\ b)\ f$
by $(metis\ has_integral_integrable_integral\ has_integral_open_interval\ not_integrable_integral)$

lemma *integrable_on_open_interval_real*:

fixes $f :: real \Rightarrow 'b :: banach$
shows $f \text{ integrable_on } \{a <..<b\} \longleftrightarrow f \text{ integrable_on } \{a..b\}$
using *integrable_on_open_interval[of f a b]* **by** *simp*

lemma *integral_open_interval_real*:

$integral\ \{a..b\}\ (f :: real \Rightarrow 'a :: banach) = integral\ \{a <..<(b::real)\}\ f$
using *integral_open_interval[of a b f]* **by** *simp*

lemma *has_integral_Icc_iff_Ioo*:

fixes $f :: real \Rightarrow 'a :: banach$
shows $(f \text{ has_integral } I) \{a..b\} \longleftrightarrow (f \text{ has_integral } I) \{a <..<b\}$
by $(metis\ box_real(1)\ cbox_interval\ has_integral_open_interval)$

lemma *integrable_on_Icc_iff_Ioo*:

fixes $f :: real \Rightarrow 'a :: banach$
shows $f \text{ integrable_on } \{a..b\} \longleftrightarrow f \text{ integrable_on } \{a <..<b\}$
using *has_integral_Icc_iff_Ioo* **by** *blast*

7.14.35 More lemmas that are useful later

lemma *has_integral_subset_le*:

fixes $f :: 'n::euclidean_space \Rightarrow real$
assumes $s \subseteq t$
and $(f \text{ has_integral } i) s$
and $(f \text{ has_integral } j) t$
and $\forall x \in t. 0 \leq f x$
shows $i \leq j$
using *assms has_integral_subset_component_le[OF __ assms(1), of 1 f i j]*
by *auto*

lemma *integral_subset_component_le*:

```

fixes  $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$ 
assumes  $k \in \text{Basis}$ 
  and  $s \subseteq t$ 
  and  $f \text{ integrable\_on } s$ 
  and  $f \text{ integrable\_on } t$ 
  and  $\forall x \in t. 0 \leq f x \cdot k$ 
shows  $(\text{integral } s f) \cdot k \leq (\text{integral } t f) \cdot k$ 
by  $(\text{meson } \text{assms } \text{has\_integral\_subset\_component\_le } \text{integrable\_integral})$ 

```

```

lemma integral_subset_le:
fixes  $f :: 'n::euclidean\_space \Rightarrow \text{real}$ 
assumes  $s \subseteq t$ 
  and  $f \text{ integrable\_on } s$ 
  and  $f \text{ integrable\_on } t$ 
  and  $\forall x \in t. 0 \leq f x$ 
shows  $\text{integral } s f \leq \text{integral } t f$ 
using assms has\_integral\_subset\_le by blast

```

```

lemma has_integral_alt':
fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::\text{banach}$ 
shows  $(f \text{ has\_integral } i) s \longleftrightarrow$ 
   $(\forall a b. (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ integrable\_on } \text{cbox } a b) \wedge$ 
   $(\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
     $\text{norm } (\text{integral } (\text{cbox } a b) (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) - i) < e)$ 
  (is ?l = ?r)

```

```

proof
assume rhs: ?r
show ?l
proof  $(\text{subst } \text{has\_integral}' , \text{intro } \text{allI } \text{impI})$ 
  fix  $e::\text{real}$ 
  assume  $e > 0$ 
  from rhs  $[\text{THEN } \text{conjunct2}, \text{rule\_format}, \text{OF } \text{this}]$ 
  show  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
     $(\exists z. ((\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ has\_integral } z)$ 
       $(\text{cbox } a b) \wedge \text{norm } (z - i) < e)$ 
    by  $(\text{simp } \text{add: } \text{has\_integral\_iff } \text{rhs})$ 
  qed
next
  let  $?\Phi = \lambda e a b. \exists z. ((\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (\text{cbox } a b) \wedge$ 
     $\text{norm } (z - i) < e$ 
  assume ?l
  then have lhs:  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow ?\Phi e a b$  if  $e > 0$  for  $e$ 
    using that  $\text{has\_integral}'[\text{of } f]$  by auto
  let  $?f = \lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0$ 
  show ?r
  proof  $(\text{intro } \text{conjI } \text{allI } \text{impI})$ 
    fix  $a b :: 'n$ 
    from lhs  $[\text{OF } \text{zero\_less\_one}]$ 
    obtain  $B$  where  $0 < B$  and  $B: \bigwedge a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies ?\Phi 1 a b$ 

```

```

    by blast
  let ?a =  $\sum_{i \in \text{Basis}} \min(a \cdot i) (-B) *_R i :: 'n$ 
  let ?b =  $\sum_{i \in \text{Basis}} \max(b \cdot i) B *_R i :: 'n$ 
  show ?f integrable_on cbox a b
  proof (rule integrable_subinterval[of _ ?a ?b])
    have ?a · i ≤ x · i ∧ x · i ≤ ?b · i if norm (0 - x) < B i ∈ Basis for x i
      using Basis_le_norm[of i x] that by (auto simp add:field_simps)
    then have ball 0 B ⊆ cbox ?a ?b
      by (auto simp: mem_box dist_norm)
    then show ?f integrable_on cbox ?a ?b
      unfolding integrable_on_def using B by blast
    show cbox a b ⊆ cbox ?a ?b
      by (force simp: mem_box)
  qed
  fix e :: real
  assume e > 0
  with lhs show ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b →
    norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) - i) < e
    by (metis (no_types, lifting) has_integral_integrable_integral)
  qed
  qed

```

7.14.36 Continuity of the integral (for a 1-dimensional interval)

lemma *integrable_alt*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$

shows f integrable_on $s \iff$

$(\forall a b. (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) \text{ integrable_on cbox } a b) \wedge$

$(\forall e > 0. \exists B > 0. \forall a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d \implies$

$\text{norm} (\text{integral} (\text{cbox } a b) (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0) -$

$\text{integral} (\text{cbox } c d) (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0)) < e)$

(is ?l = ?r)

proof

let ?F = $\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0$

assume ?l

then obtain y where $\text{int}F: \bigwedge a b. ?F \text{ integrable_on cbox } a b$

and $y: \bigwedge e. 0 < e \implies$

$\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies \text{norm} (\text{integral} (\text{cbox } a b) ?F - y) < e$

unfolding integrable_on_def has_integral_alt'[of f] by auto

show ?r

proof (intro conjI allI impI intF)

fix $e::\text{real}$

assume $e > 0$

then have $e/2 > 0$

by auto

obtain B where $0 < B$

and $B: \bigwedge a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies \text{norm} (\text{integral} (\text{cbox } a b) ?F - y) <$

```

e/2
  using ‹0 < e/2› y by blast
  show  $\exists B > 0. \forall a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d \longrightarrow$ 
     $\text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
  proof (intro conjI exI impI allI, rule ‹0 < B›)
    fix a b c d :: 'n
    assume sub:  $\text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d$ 
    show  $\text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
      using sub by (auto intro: norm_triangle_half_l dest: B)
  qed
qed
next
let ?F =  $\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0$ 
assume rhs: ?r
let ?cube =  $\lambda n. \text{cbox } (\sum i \in \text{Basis}. - \text{real } n *_{\mathbb{R}} i :: 'n) (\sum i \in \text{Basis}. \text{real } n *_{\mathbb{R}} i)$ 
have Cauchy ( $\lambda n. \text{integral } (?cube n) ?F$ )
  unfolding Cauchy_def
proof (intro allI impI)
  fix e :: real
  assume e > 0
  with rhs obtain B where 0 < B
  and B:  $\bigwedge a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c d$ 
     $\implies \text{norm } (\text{integral } (\text{cbox } a b) ?F - \text{integral } (\text{cbox } c d) ?F) < e$ 
  by blast
  obtain N where N:  $B \leq \text{real } N$ 
  using real_arch_simple by blast
  have  $\text{ball } 0 B \subseteq ?cube n$  if n:  $n \geq N$  for n
  proof -
    have  $\text{sum } ((*_{\mathbb{R}}) (- \text{real } n)) \text{Basis} \cdot i \leq x \cdot i \wedge$ 
       $x \cdot i \leq \text{sum } ((*_{\mathbb{R}}) (\text{real } n)) \text{Basis} \cdot i$ 
    if  $\text{norm } x < B$  i  $\in \text{Basis}$  for x i :: 'n
      using Basis_le_norm[of i x] n N that by (auto simp add: field_simps
sum_negf)
    then show ?thesis
      by (auto simp: mem_box dist_norm)
  qed
  then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (\text{integral } (?cube m) ?F) (\text{integral } (?cube$ 
n) ?F) < e
    by (fastforce simp add: dist_norm intro!: B)
  qed
  then obtain i where i:  $(\lambda n. \text{integral } (?cube n) ?F) \longrightarrow i$ 
  using convergent_eq_Cauchy by blast
  have  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow \text{norm } (\text{integral } (\text{cbox } a b) ?F - i)$ 
< e
    if e > 0 for e
  proof -
    have *:  $e/2 > 0$  using that by auto
    then obtain N where N:  $\bigwedge n. N \leq n \implies \text{norm } (i - \text{integral } (?cube n) ?F)$ 
< e/2

```

```

using i[THEN LIMSEQ_D, simplified norm_minus_commute] by meson
obtain B where 0 < B
and B:  $\bigwedge a b c d. \llbracket \text{ball } 0 B \subseteq \text{cbox } a b; \text{ball } 0 B \subseteq \text{cbox } c d \rrbracket \implies$ 
      norm (integral (cbox a b) ?F - integral (cbox c d) ?F) < e/2
using rhs * by meson
let ?B = max (real N) B
show ?thesis
proof (intro exI conjI allI impI)
  show 0 < ?B
    using <B > 0> by auto
  fix a b :: 'n
  assume ball 0 ?B  $\subseteq$  cbox a b
  moreover obtain n where n: max (real N) B  $\leq$  real n
    using real_arch_simple by blast
  moreover have ball 0 B  $\subseteq$  ?cube n
  proof
    fix x :: 'n
    assume x: x  $\in$  ball 0 B
    have  $\llbracket \text{norm } (0 - x) < B; i \in \text{Basis} \rrbracket$ 
       $\implies \text{sum } ((*_R) (-n)) \text{Basis} \cdot i \leq x \cdot i \wedge x \cdot i \leq \text{sum } ((*_R) n) \text{Basis} \cdot i$ 
  for i
    using Basis_le_norm[of i x] n by (auto simp add: field_simps sum_negf)
  then show x  $\in$  ?cube n
    using x by (auto simp: mem_box dist_norm)
  qed
  ultimately show norm (integral (cbox a b) ?F - i) < e
    using norm_triangle_half_l [OF B N] by force
  qed
  qed
  then show ?l unfolding integrable_on_def has_integral_alt'[of f]
    using rhs by blast
  qed

```

lemma integrable_altD:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
assumes f integrable_on s
shows  $\bigwedge a b. (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0)$  integrable_on cbox a b
and  $\bigwedge e. e > 0 \implies \exists B > 0. \forall a b c d. \text{ball } 0 B \subseteq \text{cbox } a b \wedge \text{ball } 0 B \subseteq \text{cbox } c$ 
 $d \implies$ 
  norm (integral (cbox a b)  $(\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0)$  - integral (cbox c d)
 $(\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0)) < e$ 
using assms[unfolded integrable_alt[of f]] by auto

```

lemma integrable_alt_subset:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
shows
  f integrable_on S  $\longleftrightarrow$ 
   $(\forall a b. (\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0)$  integrable_on cbox a b)  $\wedge$ 
   $(\forall e > 0. \exists B > 0. \forall a b c d.$ 

```

```

      ball 0 B ⊆ cbox a b ∧ cbox a b ⊆ cbox c d
      → norm(integral (cbox a b) (λx. if x ∈ S then f x else 0) -
              integral (cbox c d) (λx. if x ∈ S then f x else 0)) < e)
    (is _ = ?rhs)
  proof -
    let ?g = λx. if x ∈ S then f x else 0
    have f integrable_on S ↔
      (∀ a b. ?g integrable_on cbox a b) ∧
      (∀ e>0. ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d →
        norm (integral (cbox a b) ?g - integral (cbox c d) ?g) < e)
    by (rule integrable_alt)
    also have ... = ?rhs
    proof -
      { fix e :: real
        assume e: ∧e. e>0 ⇒ ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ cbox a b ⊆
        cbox c d →
          norm (integral (cbox a b) ?g - integral (cbox c d) ?g)
        < e
          and e > 0
          obtain B where B > 0
          and B: ∧a b c d. [ball 0 B ⊆ cbox a b; cbox a b ⊆ cbox c d] ⇒
            norm (integral (cbox a b) ?g - integral (cbox c d) ?g) < e/2
          using ‹e > 0› e [of e/2] by force
          have ∃ B>0. ∀ a b c d.
            ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d →
            norm (integral (cbox a b) ?g - integral (cbox c d) ?g) < e
          proof (intro exI allI conjI impI)
            fix a b c d :: 'a
            let ?α = ∑ i∈Basis. max (a · i) (c · i) *R i
            let ?β = ∑ i∈Basis. min (b · i) (d · i) *R i
            show norm (integral (cbox a b) ?g - integral (cbox c d) ?g) < e
              if ball: ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d
            proof -
              have B': norm (integral (cbox a b ∩ cbox c d) ?g - integral (cbox x y) ?g)
              < e/2
                if cbox a b ∩ cbox c d ⊆ cbox x y for x y
              using B [of ?α ?β x y] ball that by (simp add: Int_interval [symmetric])
              show ?thesis
                using B' [of a b] B' [of c d] norm_triangle_half_r by blast
            qed
          qed (use ‹B > 0› in auto)}
      then show ?thesis
        by force
    qed
  finally show ?thesis .
qed

```

lemma integrable_on_subcbox:

fixes f :: 'n::euclidean_space ⇒ 'a::banach

assumes *intf*: f integrable_on S
and *sub*: $\text{cbox } a \ b \subseteq S$
shows f integrable_on $\text{cbox } a \ b$
proof –
have $(\lambda x. \text{if } x \in S \text{ then } f \ x \ \text{else } 0)$ integrable_on $\text{cbox } a \ b$
by (*simp add: intf integrable_altD(1)*)
then show ?thesis
by (*metis (mono_tags) sub integrable_restrict_Int le_inf_iff order_refl subset_antisym*)
qed

7.14.37 A straddling criterion for integrability

lemma *integrable_straddle_interval*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow \text{real}$
assumes $\bigwedge e. e > 0 \implies \exists g \ h \ i \ j. (g \ \text{has_integral } i) (\text{cbox } a \ b) \wedge (h \ \text{has_integral } j) (\text{cbox } a \ b) \wedge$

$$|i - j| < e \wedge (\forall x \in \text{cbox } a \ b. (g \ x) \leq f \ x \wedge f \ x \leq h \ x)$$

shows f integrable_on $\text{cbox } a \ b$

proof –

have $\exists d. \text{gauge } d \wedge$

$$(\forall p1 \ p2. p1 \ \text{tagged_division_of } \text{cbox } a \ b \wedge d \ \text{fine } p1 \wedge$$

$$p2 \ \text{tagged_division_of } \text{cbox } a \ b \wedge d \ \text{fine } p2 \implies$$

$$|(\sum (x,K) \in p1. \text{content } K \ *_R \ f \ x) - (\sum (x,K) \in p2. \text{content } K \ *_R$$

$$f \ x)| < e)$$

if $e > 0$ **for** e

proof –

have $e/3 > 0$

using *that by auto*

then obtain $g \ h \ i \ j$ **where** $ij: |i - j| < e/3$

and $(g \ \text{has_integral } i) (\text{cbox } a \ b)$

and $(h \ \text{has_integral } j) (\text{cbox } a \ b)$

and $fgh: \bigwedge x. x \in \text{cbox } a \ b \implies g \ x \leq f \ x \wedge f \ x \leq h \ x$

using *assms real_norm_def by metis*

then obtain $d1 \ d2$ **where** *gauge* $d1$ *gauge* $d2$

and $d1: \bigwedge p. \llbracket p \ \text{tagged_division_of } \text{cbox } a \ b; d1 \ \text{fine } p \rrbracket \implies$

$$|(\sum (x,K) \in p. \text{content } K \ *_R \ g \ x) - i| < e/3$$

and $d2: \bigwedge p. \llbracket p \ \text{tagged_division_of } \text{cbox } a \ b; d2 \ \text{fine } p \rrbracket \implies$

$$|(\sum (x,K) \in p. \text{content } K \ *_R \ h \ x) - j| < e/3$$

by (*metis e has_integral real_norm_def*)

have $|(\sum (x,K) \in p1. \text{content } K \ *_R \ f \ x) - (\sum (x,K) \in p2. \text{content } K \ *_R \ f \ x)| < e$

if $p1: p1 \ \text{tagged_division_of } \text{cbox } a \ b$ **and** $11: d1 \ \text{fine } p1$ **and** $21: d2 \ \text{fine } p1$

and $p2: p2 \ \text{tagged_division_of } \text{cbox } a \ b$ **and** $12: d1 \ \text{fine } p2$ **and** $22: d2 \ \text{fine } p2$

for $p1 \ p2$

proof –

have $*$: $\bigwedge g1 \ g2 \ h1 \ h2 \ f1 \ f2.$

$$\llbracket |g2 - i| < e/3; |g1 - i| < e/3; |h2 - j| < e/3; |h1 - j| < e/3; \\ g1 - h2 \leq f1 - f2; f1 - f2 \leq h1 - g2 \rrbracket$$


```

       $\implies |f1 - f2| < e$ 
    using  $\langle e > 0 \rangle$  ij by arith
  have 0:  $(\sum (x, k) \in p1. \text{content } k *_R f x) - (\sum (x, k) \in p1. \text{content } k *_R g x)$ 
 $\geq 0$ 
       $0 \leq (\sum (x, k) \in p2. \text{content } k *_R h x) - (\sum (x, k) \in p2. \text{content } k *_R f x)$ 
       $(\sum (x, k) \in p2. \text{content } k *_R f x) - (\sum (x, k) \in p2. \text{content } k *_R g x) \geq 0$ 
       $0 \leq (\sum (x, k) \in p1. \text{content } k *_R h x) - (\sum (x, k) \in p1. \text{content } k *_R f x)$ 
    unfolding sum_subtractf[symmetric]
      apply (auto intro!: sum_nonneg)
      apply (meson fgh measure_nonneg mult_left_mono tag_in_interval that sum_nonneg)
    done
  show ?thesis
  proof (rule *)
    show  $|(\sum (x, K) \in p2. \text{content } K *_R g x) - i| < e/3$ 
      by (rule d1[OF p2 12])
    show  $|(\sum (x, K) \in p1. \text{content } K *_R g x) - i| < e/3$ 
      by (rule d1[OF p1 11])
    show  $|(\sum (x, K) \in p2. \text{content } K *_R h x) - j| < e/3$ 
      by (rule d2[OF p2 22])
    show  $|(\sum (x, K) \in p1. \text{content } K *_R h x) - j| < e/3$ 
      by (rule d2[OF p1 21])
  qed (use 0 in auto)
  qed
  then show ?thesis
  by (rule_tac  $x = \lambda x. d1 x \cap d2 x$  in exI)
    (auto simp: fine_Int intro: gauge_d1 gauge_d2 d1 d2)
  qed
  then show ?thesis
  by (simp add: integrable_Cauchy)
  qed

```

lemma *integrable_straddle*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow \text{real}$

assumes $\bigwedge e. e > 0 \implies \exists g h i j. (g \text{ has_integral } i) s \wedge (h \text{ has_integral } j) s \wedge$
 $|i - j| < e \wedge (\forall x \in s. g x \leq f x \wedge f x \leq h x)$

shows $f \text{ integrable_on } s$

proof -

let $?fs = (\lambda x. \text{if } x \in s \text{ then } f x \text{ else } 0)$

have $?fs \text{ integrable_on } \text{cbox } a \ b$ for $a \ b$

proof (rule *integrable_straddle_interval*)

fix $e::\text{real}$

assume $e > 0$

then have $*: e/4 > 0$

by *auto*

with *assms* obtain $g \ h \ i \ j$ where $g: (g \text{ has_integral } i) s$ and $h: (h \text{ has_integral } j) s$

and *ij*: $|i - j| < e/4$

and *fgh*: $\bigwedge x. x \in s \implies g x \leq f x \wedge f x \leq h x$

```

    by metis
    let ?gs = (λx. if x ∈ s then g x else 0)
    let ?hs = (λx. if x ∈ s then h x else 0)
    obtain Bg where Bg: ∧a b. ball 0 Bg ⊆ cbox a b ⇒ |integral (cbox a b) ?gs
    - i| < e/4
      and int_g: ∧a b. ?gs integrable_on cbox a b
      using g * unfolding has_integral_alt' real_norm_def by meson
    obtain Bh where
      Bh: ∧a b. ball 0 Bh ⊆ cbox a b ⇒ |integral (cbox a b) ?hs - j| < e/4
      and int_h: ∧a b. ?hs integrable_on cbox a b
      using h * unfolding has_integral_alt' real_norm_def by meson
    define c where c = (∑ i∈Basis. min (a·i) (- (max Bg Bh)) *R i)
    define d where d = (∑ i∈Basis. max (b·i) (max Bg Bh) *R i)
    have [norm (0 - x) < Bg; i ∈ Basis] ⇒ c · i ≤ x · i ∧ x · i ≤ d · i for x i
      using Basis_le_norm[of i x] unfolding c_def d_def by auto
    then have ballBg: ball 0 Bg ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    have [norm (0 - x) < Bh; i ∈ Basis] ⇒ c · i ≤ x · i ∧ x · i ≤ d · i for x i
      using Basis_le_norm[of i x] unfolding c_def d_def by auto
    then have ballBh: ball 0 Bh ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    have ab_cd: cbox a b ⊆ cbox c d
      by (auto simp: c_def d_def subset_box_imp)
    have **: ∧ch cg ag ah::real. [|ah - ag| ≤ |ch - cg|; |cg - i| < e/4; |ch - j|
    < e/4]
      ⇒ |ag - ah| < e
      using ij by arith
    show ∃g h i j. (g has_integral i) (cbox a b) ∧ (h has_integral j) (cbox a b) ∧
    |i - j| < e ∧
      (∀x∈cbox a b. g x ≤ (if x ∈ s then f x else 0) ∧
        (if x ∈ s then f x else 0) ≤ h x)
    proof (intro exI ballI conjI)
      have eq: ∧x f g. (if x ∈ s then f x else 0) - (if x ∈ s then g x else 0) =
        (if x ∈ s then f x - g x else (0::real))
      by auto
      have int_hg: (λx. if x ∈ s then h x - g x else 0) integrable_on cbox a b
        (λx. if x ∈ s then h x - g x else 0) integrable_on cbox c d
      by (metis (no_types) integrable_diff g h has_integral_integrable inte-
      grable_altD(1))+
      show (?gs has_integral integral (cbox a b) ?gs) (cbox a b)
        (?hs has_integral integral (cbox a b) ?hs) (cbox a b)
      by (intro integrable_integral int_g int_h)+
      then have integral (cbox a b) ?gs ≤ integral (cbox a b) ?hs
      using fgh by (force intro: has_integral_le)
      then have 0 ≤ integral (cbox a b) ?hs - integral (cbox a b) ?gs
      by simp
      then have |integral (cbox a b) ?hs - integral (cbox a b) ?gs|
        ≤ |integral (cbox c d) ?hs - integral (cbox c d) ?gs|
      apply (simp add: integral_diff [symmetric] int_g int_h)

```

```

apply (subst abs_of_nonneg[OF integral_nonneg[OF integrable_diff, OF
int_h int_g]])
using fgh apply (force simp: eq intro!: integral_subset_le [OF ab_cd
int_hg])+
done
then show |integral (cbox a b) ?gs - integral (cbox a b) ?hs| < e
using ** Bg ballBg Bh ballBh by blast
show  $\bigwedge x. x \in \text{cbox } a \text{ } b \implies ?gs \ x \leq ?fs \ x \wedge x. x \in \text{cbox } a \text{ } b \implies ?fs \ x \leq ?hs \ x$ 
using fgh by auto
qed
qed
then have int_f: ?fs integrable_on cbox a b for a b
by simp
have  $\exists B > 0. \forall a \ b \ c \ d.$ 

$$\text{ball } 0 \ B \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ B \subseteq \text{cbox } c \ d \longrightarrow$$


$$\text{abs } (\text{integral } (\text{cbox } a \ b) \ ?fs - \text{integral } (\text{cbox } c \ d) \ ?fs) < e$$

if  $0 < e$  for e
proof -
have *:  $e/3 > 0$ 
using that by auto
with assms obtain g h i j where g: (g has_integral i) s and h: (h has_integral
j) s
and ij:  $|i - j| < e/3$ 
and fgh:  $\bigwedge x. x \in s \implies g \ x \leq f \ x \wedge f \ x \leq h \ x$ 
by metis
let ?gs = ( $\lambda x. \text{if } x \in s \text{ then } g \ x \text{ else } 0$ )
let ?hs = ( $\lambda x. \text{if } x \in s \text{ then } h \ x \text{ else } 0$ )
obtain Bg where Bg > 0
and Bg:  $\bigwedge a \ b. \text{ball } 0 \ Bg \subseteq \text{cbox } a \ b \implies |\text{integral } (\text{cbox } a \ b) \ ?gs - i| <$ 
 $e/3$ 
and int_g:  $\bigwedge a \ b. ?gs \ \text{integrable\_on } \text{cbox } a \ b$ 
using g * unfolding has_integral_alt' real_norm_def by meson
obtain Bh where Bh > 0
and Bh:  $\bigwedge a \ b. \text{ball } 0 \ Bh \subseteq \text{cbox } a \ b \implies |\text{integral } (\text{cbox } a \ b) \ ?hs - j|$ 
 $< e/3$ 
and int_h:  $\bigwedge a \ b. ?hs \ \text{integrable\_on } \text{cbox } a \ b$ 
using h * unfolding has_integral_alt' real_norm_def by meson
{ fix a b c d :: 'n
assume as:  $\text{ball } 0 \ (\max \ Bg \ Bh) \subseteq \text{cbox } a \ b \wedge \text{ball } 0 \ (\max \ Bg \ Bh) \subseteq \text{cbox } c \ d$ 
have **:  $\text{ball } 0 \ Bg \subseteq \text{ball } (0::'n) \ (\max \ Bg \ Bh) \wedge \text{ball } 0 \ Bh \subseteq \text{ball } (0::'n) \ (\max$ 
Bg Bh)
by auto
have *:  $\bigwedge ga \ gc \ ha \ hc \ fa \ fc. [|ga - i| < e/3; |gc - i| < e/3; |ha - j| < e/3;$ 
 $|hc - j| < e/3; ga \leq fa; fa \leq ha; gc \leq fc; fc \leq hc] \implies$ 
 $|fa - fc| < e$ 
using ij by arith
have abs (integral (cbox a b) ( $\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0$ ) - integral (cbox c
d)
( $\lambda x. \text{if } x \in s \text{ then } f \ x \text{ else } 0$ )) < e

```

```

proof (rule *)
  show |integral (cbox a b) ?gs - i| < e/3
    using ** Bg as by blast
  show |integral (cbox c d) ?gs - i| < e/3
    using ** Bg as by blast
  show |integral (cbox a b) ?hs - j| < e/3
    using ** Bh as by blast
  show |integral (cbox c d) ?hs - j| < e/3
    using ** Bh as by blast
  qed (use int_f int_g int_h fgh in <simp_all add: integral_le>)
}
then show ?thesis
  apply (rule_tac x=max Bg Bh in exI)
  using <Bg > 0> by auto
qed
then show ?thesis
  unfolding integrable_alt[of f] real_norm_def by (blast intro: int_f)
qed

```

7.14.38 Adding integrals over several sets

```

lemma has_integral_Un:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f: (f has_integral i) S (f has_integral j) T
  and neg: negligible (S  $\cap$  T)
  shows (f has_integral (i + j)) (S  $\cup$  T)
  unfolding has_integral_restrict_UNIV[symmetric, of f]
proof (rule has_integral_spike[OF neg])
  let ?f =  $\lambda x.$  (if x  $\in$  S then f x else 0) + (if x  $\in$  T then f x else 0)
  show (?f has_integral i + j) UNIV
  by (simp add: f has_integral_add)
qed auto

lemma integral_Un [simp]:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes f integrable_on S f integrable_on T negligible (S  $\cap$  T)
  shows integral (S  $\cup$  T) f = integral S f + integral T f
  by (simp add: has_integral_Un assms integrable_integral integral_unique)

```

```

lemma integrable_Un:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes negligible (A  $\cap$  B) f integrable_on A f integrable_on B
  shows f integrable_on (A  $\cup$  B)
proof -
  from assms obtain y z where (f has_integral y) A (f has_integral z) B
  by (auto simp: integrable_on_def)
  from has_integral_Un[OF this assms(1)] show ?thesis by (auto simp: integrable_on_def)
qed

```

```

lemma integrable_Un':
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::banach
  assumes f integrable_on A f integrable_on B negligible (A  $\cap$  B) C = A  $\cup$  B
  shows f integrable_on C
  using integrable_Un[of A B f] assms by simp

lemma has_integral_UN:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  assumes finite I
  and int:  $\bigwedge i. i \in I \implies (f \text{ has\_integral } (g \ i)) (\mathcal{T} \ i)$ 
  and neg: pairwise  $(\lambda i \ i'. \text{negligible } (\mathcal{T} \ i \cap \mathcal{T} \ i')) I$ 
  shows (f has\_integral (sum g I))  $(\bigcup i \in I. \mathcal{T} \ i)$ 
proof -
  let ?U =  $((\lambda(a,b). \mathcal{T} \ a \cap \mathcal{T} \ b) \text{ ` } \{(a,b). a \in I \wedge b \in I - \{a\}\})$ 
  have  $((\lambda x. \text{if } x \in (\bigcup i \in I. \mathcal{T} \ i) \text{ then } f \ x \text{ else } 0) \text{ has\_integral sum g I}) \text{ UNIV}$ 
  proof (rule has\_integral_spike)
    show negligible  $(\bigcup ?U)$ 
  proof (rule negligible_Union)
    have finite (I  $\times$  I)
    by (simp add: <finite I>)
    moreover have  $\{(a,b). a \in I \wedge b \in I - \{a\}\} \subseteq I \times I$ 
    by auto
    ultimately show finite ?U
    by (simp add: finite_subset)
    show  $\bigwedge t. t \in ?U \implies \text{negligible } t$ 
    using neg unfolding pairwise_def by auto
  qed
  next
  show  $(\text{if } x \in (\bigcup i \in I. \mathcal{T} \ i) \text{ then } f \ x \text{ else } 0) = (\sum i \in I. \text{if } x \in \mathcal{T} \ i \text{ then } f \ x \text{ else } 0)$ 
  if  $x \in \text{UNIV} - (\bigcup ?U)$  for x
  proof clarsimp
    fix i assume i:  $i \in I \ x \in \mathcal{T} \ i$ 
    then have  $\forall j \in I. x \in \mathcal{T} \ j \longleftrightarrow j = i$ 
    using that by blast
    with i show  $f \ x = (\sum i \in I. \text{if } x \in \mathcal{T} \ i \text{ then } f \ x \text{ else } 0)$ 
    by (simp add: sum.delta[OF <finite I>])
  qed
  next
  show  $((\lambda x. (\sum i \in I. \text{if } x \in \mathcal{T} \ i \text{ then } f \ x \text{ else } 0)) \text{ has\_integral sum g I}) \text{ UNIV}$ 
  using int by (simp add: has\_integral_restrict_UNIV has\_integral_sum [OF
<finite I>])
  qed
  then show ?thesis
  using has\_integral_restrict_UNIV by blast
qed

```

```

lemma has_integral_Union:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach

```

```

assumes finite  $\mathcal{T}$ 
and  $\bigwedge S. S \in \mathcal{T} \implies (f \text{ has\_integral } (i \ S)) \ S$ 
and pairwise  $(\lambda S \ S'. \text{negligible } (S \cap S')) \ \mathcal{T}$ 
shows  $(f \text{ has\_integral } (\text{sum } i \ \mathcal{T})) \ (\bigcup \mathcal{T})$ 
proof –
have  $(f \text{ has\_integral } (\text{sum } i \ \mathcal{T})) \ (\bigcup_{S \in \mathcal{T}} S)$ 
by (intro has_integral_UN assms)
then show ?thesis
by force
qed

```

In particular adding integrals over a division, maybe not of an interval.

```

lemma has_integral_combine_division:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $\mathcal{D}$  division_of  $S$ 
and  $\bigwedge k. k \in \mathcal{D} \implies (f \text{ has\_integral } (i \ k)) \ k$ 
shows  $(f \text{ has\_integral } (\text{sum } i \ \mathcal{D})) \ S$ 
proof –
note  $\mathcal{D} = \text{division\_of} D[\text{OF assms}(1)]$ 
have neg: negligible  $(S \cap s')$  if  $S \in \mathcal{D} \ s' \in \mathcal{D} \ S \neq s'$  for  $S \ s'$ 
proof –
obtain  $a \ c \ b \ \mathcal{D}$  where obt:  $S = \text{cbox } a \ b \ s' = \text{cbox } c \ \mathcal{D}$ 
by (meson  $\langle S \in \mathcal{D} \ \langle s' \in \mathcal{D} \rangle \ \mathcal{D}(4)$ )
from  $\mathcal{D}(5)[\text{OF that}]$  show ?thesis
unfolding obt interior_cbox
by (metis (no_types, lifting) Diff_empty Int_interval box_Int_box negligible_frontier_interval)
qed
show ?thesis
unfolding  $\mathcal{D}(6)[\text{symmetric}]$ 
by (auto intro:  $\mathcal{D}$  neg assms has_integral_Union pairwiseI)
qed

```

```

lemma integral_combine_division_bottomup:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $\mathcal{D}$  division_of  $S \ \bigwedge k. k \in \mathcal{D} \implies f \text{ integrable\_on } k$ 
shows  $\text{integral } S \ f = \text{sum } (\lambda i. \text{integral } i \ f) \ \mathcal{D}$ 
by (meson assms integral_unique has_integral_combine_division has_integral_integrable_integral)

```

```

lemma has_integral_combine_division_topdown:
fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes  $f: f \text{ integrable\_on } S$ 
and  $\mathcal{D}: \mathcal{D}$  division_of  $K$ 
and  $K \subseteq S$ 
shows  $(f \text{ has\_integral } (\text{sum } (\lambda i. \text{integral } i \ f) \ \mathcal{D})) \ K$ 
proof –
have  $f \text{ integrable\_on } L$  if  $L \in \mathcal{D}$  for  $L$ 
by (smt (verit, best) assms cbox_division_memE f integrable_on_subcbox subset_trans that)

```

```

then show ?thesis
  by (meson  $\mathcal{D}$  has_integral_combine_division has_integral_integrable_integral)
qed

```

```

lemma integral_combine_division_topdown:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::banach$ 
  assumes  $f$  integrable_on  $S$ 
  and  $\mathcal{D}$  division_of  $S$ 
  shows  $\text{integral } S f = \text{sum } (\lambda i. \text{integral } i f) \mathcal{D}$ 
  using assms has_integral_combine_division_topdown by blast

```

```

lemma integrable_combine_division:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::banach$ 
  assumes  $\mathcal{D}: \mathcal{D}$  division_of  $S$ 
  and  $f: \bigwedge i. i \in \mathcal{D} \implies f$  integrable_on  $i$ 
  shows  $f$  integrable_on  $S$ 
  using  $f$  unfolding integrable_on_def by (metis has_integral_combine_division[OF  $\mathcal{D}$ ])

```

```

lemma integrable_on_subdivision:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::banach$ 
  assumes  $\mathcal{D}: \mathcal{D}$  division_of  $i$ 
  and  $f: f$  integrable_on  $S$ 
  and  $i \subseteq S$ 
  shows  $f$  integrable_on  $i$ 
proof –
  have  $f$  integrable_on  $i$  if  $i \in \mathcal{D}$  for  $i$ 
    by (smt (verit, best) assms cbox_division_memE  $f$  integrable_on_subcbox_order_trans that)
  then show ?thesis
    using  $\mathcal{D}$  integrable_combine_division by blast
qed

```

7.14.39 Also tagged divisions

```

lemma has_integral_combine_tagged_division:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::banach$ 
  assumes  $p$  tagged_division_of  $S$ 
  and  $\bigwedge x k. (x,k) \in p \implies (f \text{ has\_integral } (i k)) k$ 
  shows  $(f \text{ has\_integral } (\sum (x,k) \in p. i k)) S$ 
proof –
  have *:  $(f \text{ has\_integral } (\sum k \in \text{snd}'p. \text{integral } k f)) S$ 
    by (smt (verit, del_insts) assms division_of_tagged_division has_integral_combine_division
has_integral_iff_imageE prod.collapse)
  also have  $(\sum k \in \text{snd}'p. \text{integral } k f) = (\sum (x, k) \in p. \text{integral } k f)$ 
    by (intro sum.over_tagged_division_lemma[OF assms(1), symmetric] integrable_null)
    (simp add: content_eq_0_interior)
  finally show ?thesis

```

using *assms* by (auto simp add: has_integral_iff intro!: sum.cong)
qed

lemma *integral_combine_tagged_division_bottomup*:
 fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
 assumes $p: p \text{ tagged_division_of } (cbox\ a\ b)$
 and $f: \bigwedge x\ k. (x,k) \in p \implies f \text{ integrable_on } k$
 shows $\text{integral } (cbox\ a\ b)\ f = \text{sum } (\lambda(x,k). \text{integral } k\ f)\ p$
 by (simp add: has_integral_combine_tagged_division[OF p] integral_unique f
 integrable_integral)

lemma *has_integral_combine_tagged_division_topdown*:
 fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
 assumes $f: f \text{ integrable_on } cbox\ a\ b$
 and $p: p \text{ tagged_division_of } (cbox\ a\ b)$
 shows $(f \text{ has_integral } (\text{sum } (\lambda(x,K). \text{integral } K\ f)\ p)) (cbox\ a\ b)$
proof –
 have $(f \text{ has_integral } \text{integral } K\ f)\ K$ if $(x,K) \in p$ for $x\ K$
 by (metis *assms* integrable_integral integrable_on_subcbox tagged_division_ofD(3,4)
 that)
 then show ?thesis
 by (simp add: has_integral_combine_tagged_division p)
 qed

lemma *integral_combine_tagged_division_topdown*:
 fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
 assumes $f \text{ integrable_on } cbox\ a\ b$
 and $p \text{ tagged_division_of } (cbox\ a\ b)$
 shows $\text{integral } (cbox\ a\ b)\ f = \text{sum } (\lambda(x,k). \text{integral } k\ f)\ p$
 using *assms* by (auto intro: integral_unique [OF has_integral_combine_tagged_division_topdown])

7.14.40 Henstock's lemma

lemma *Henstock_lemma_part1*:
 fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'a::\text{banach}$
 assumes *intf*: $f \text{ integrable_on } cbox\ a\ b$
 and $e > 0$
 and *gauge* d
 and *less_e*: $\bigwedge p. [p \text{ tagged_division_of } (cbox\ a\ b); d \text{ fine } p] \implies$
 $\text{norm } (\text{sum } (\lambda(x,K). \text{content } K *_{\mathbb{R}} f\ x)\ p - \text{integral}(cbox\ a\ b)\ f)$
 $< e$
 and $p: p \text{ tagged_partial_division_of } (cbox\ a\ b)\ d \text{ fine } p$
 shows $\text{norm } (\text{sum } (\lambda(x,K). \text{content } K *_{\mathbb{R}} f\ x) - \text{integral } K\ f)\ p \leq e$ (is ?lhs \leq
 e)
proof (rule *field_le_epsilon*)
 fix $k :: \text{real}$
 assume $k > 0$
 let ?SUM = $\lambda p. (\sum (x,K) \in p. \text{content } K *_{\mathbb{R}} f\ x)$
 note $p' = \text{tagged_partial_division_ofD}[OF\ p(1)]$


```

have  $\bigcup (snd \text{ ` } p) \subseteq cbox \ a \ b$ 
  using  $p'(3)$  by fastforce
then obtain  $q$  where  $q: snd \text{ ` } p \subseteq q$  and  $qdiv: q \text{ division\_of } cbox \ a \ b$ 
  by (meson  $p(1)$  partial_division_extend_interval partial_division_of_tagged_division)
note  $q' = division\_ofD[OF \ qdiv]$ 
define  $r$  where  $r = q - snd \text{ ` } p$ 
have  $snd \text{ ` } p \cap r = \{\}$ 
  unfolding  $r\_def$  by auto
have finite  $r$ 
  using  $q'$  unfolding  $r\_def$  by auto
have  $\exists p. p \text{ tagged\_division\_of } i \wedge d \text{ fine } p \wedge$ 
  norm ( $?SUM \ p - integral \ i \ f$ )  $< k / (real (card \ r) + 1)$ 
  if  $i \in r$  for  $i$ 
proof -
  have  $gt0: k / (real (card \ r) + 1) > 0$  using  $\langle k > 0 \rangle$  by simp
  have  $i: i \in q$ 
    using that unfolding  $r\_def$  by auto
  then obtain  $u \ v$  where  $uv: i = cbox \ u \ v$ 
    using  $q'(4)$  by blast
  then have  $cbox \ u \ v \subseteq cbox \ a \ b$ 
    using  $i \ q'(2)$  by auto
  then have  $f \text{ integrable\_on } cbox \ u \ v$ 
    by (rule integrable_subinterval[OF  $intf$ ])
  with integrable_integral[OF this, unfolded has_integral[of  $f$ ]]
  obtain  $dd$  where gauge  $dd$  and  $dd:$ 
     $\bigwedge \mathcal{D}. [\mathcal{D} \text{ tagged\_division\_of } cbox \ u \ v; dd \text{ fine } \mathcal{D}] \implies$ 
    norm ( $?SUM \ \mathcal{D} - integral \ (cbox \ u \ v) \ f$ )  $< k / (real (card \ r) + 1)$ 
    using  $gt0$  by auto
  with gauge_Int[OF  $\langle gauge \ d \rangle \ \langle gauge \ dd \rangle$ ]
  obtain  $qq$  where  $qq: qq \text{ tagged\_division\_of } cbox \ u \ v \ (\lambda x. d \ x \cap dd \ x) \text{ fine } qq$ 
    using fine_division_exists by blast
  with  $dd$ [of  $qq$ ] show ?thesis
    by (auto simp: fine_Int  $uv$ )
qed
then obtain  $qq$  where  $qq: \bigwedge i. i \in r \implies qq \ i \text{ tagged\_division\_of } i \wedge$ 
   $d \text{ fine } qq \ i \wedge norm \ (?SUM \ (qq \ i) - integral \ i \ f) < k / (real (card \ r) + 1)$ 
  by metis

let  $?p = p \cup \bigcup (qq \text{ ` } r)$ 
have norm ( $?SUM \ ?p - integral \ (cbox \ a \ b) \ f$ )  $< e$ 
proof (rule less_e)
  show  $d \text{ fine } ?p$ 
    by (metis (mono_tags, opaque_lifting)  $qq \text{ fine\_Un } \text{fine\_Union } imageE \ p(2)$ )
  note  $ptag = tagged\_partial\_division\_of\_Union\_self[OF \ p(1)]$ 
  have  $p \cup \bigcup (qq \text{ ` } r) \text{ tagged\_division\_of } \bigcup (snd \text{ ` } p) \cup \bigcup r$ 
  proof (rule tagged_division_Un[OF  $ptag \text{ tagged\_division\_Union } [OF \ \langle finite \ r \rangle]$ ])
    show  $\bigwedge i. i \in r \implies qq \ i \text{ tagged\_division\_of } i$ 
      using  $qq$  by auto
  end
end

```

```

show  $\bigwedge i1\ i2. \llbracket i1 \in r; i2 \in r; i1 \neq i2 \rrbracket \implies interior\ i1 \cap interior\ i2 = \{\}$ 
  by (simp add: q'(5) r_def)
show  $interior\ (\bigcup (snd\ 'p)) \cap interior\ (\bigcup r) = \{\}$ 
proof (rule Int_interior_Union_intervals [OF <finite r>])
  show open (interior ( $\bigcup (snd\ 'p)$ ))
    by blast
  show  $\bigwedge T. T \in r \implies \exists a\ b. T = cbox\ a\ b$ 
    by (simp add: q'(4) r_def)
  have  $interior\ T \cap interior\ (\bigcup (snd\ 'p)) = \{\}$  if  $T \in r$  for  $T$ 
  proof (rule Int_interior_Union_intervals)
    show  $\bigwedge U. U \in snd\ 'p \implies \exists a\ b. U = cbox\ a\ b$ 
      using q q'(4) by blast
    show  $\bigwedge U. U \in snd\ 'p \implies interior\ T \cap interior\ U = \{\}$ 
      by (metis DiffE q q'(5) r_def subsetD that)
  qed (use p' in auto)
  then show  $\bigwedge T. T \in r \implies interior\ (\bigcup (snd\ 'p)) \cap interior\ T = \{\}$ 
    by (metis Int_commute)
  qed
  qed
  moreover have  $\bigcup (snd\ 'p) \cup \bigcup r = cbox\ a\ b$  and  $\{qq\ i\ | i. i \in r\} = qq\ 'r$ 
    using qdiv q unfolding Union_Un_distrib[symmetric] r_def by auto
  ultimately show ?p tagged_division_of (cbox a b)
    by fastforce
  qed
  then have  $norm\ (?SUM\ p + (?SUM\ (\bigcup (qq\ 'r))) - integral\ (cbox\ a\ b)\ f) < e$ 
  proof (subst sum.union_inter_neutral[symmetric, OF <finite p>], safe)
    show content  $L *_R f\ x = 0$  if  $(x, L) \in p$   $(x, L) \in qq\ K$   $K \in r$  for  $x\ K\ L$ 
    proof -
      obtain  $u\ v$  where  $uv: L = cbox\ u\ v$ 
        using <(x,L) ∈ p> p'(4) by blast
      have  $L \subseteq K$ 
        using qq[OF that(3)] tagged_division_ofD(3) <(x,L) ∈ qq K> by metis
      have  $L \in snd\ 'p$ 
        using <(x,L) ∈ p> image_iff by fastforce
      then have  $L \in q\ K \in q\ L \neq K$ 
        using that q(1) unfolding r_def by auto
      with q'(5) show content  $L *_R f\ x = 0$ 
        by (metis <L ⊆ K> content_eq_0_interior inf.orderE interior_Int scaleR_eq_0_iff
          uv)
    qed
  qed
  show finite ( $\bigcup (qq\ 'r)$ )
    by (meson finite_UN qq <finite r> tagged_division_of_finite)
  qed
  moreover have content  $M *_R f\ x = 0$ 
    if  $x: (x, M) \in qq\ K$   $(x, M) \in qq\ L$  and  $KL: qq\ K \neq qq\ L$  and  $r: K \in r\ L \in r$ 
    for  $x\ M\ K\ L$ 
  proof -
    note  $kl = tagged\_division\_ofD(3,4)[OF\ qq[THEN\ conjunct1]]$ 
    obtain  $u\ v$  where  $uv: M = cbox\ u\ v$ 

```

```

    using ⟨(x, M) ∈ qq L⟩ ⟨L ∈ r⟩ kl(2) by blast
  have empty: interior (K ∩ L) = {}
    by (metis DiffD1 interior_Int q'(5) r_def KL r)
  with that kl show content M *R f x = 0
    by (metis content_eq_0_interior dual_order.refl inf.orderE inf_mono interior_mono
        scaleR_eq_0_iff subset_empty uv x)
qed
ultimately have norm (?SUM p + sum ?SUM (qq ' r) - integral (cbox a b) f)
< e
  apply (subst (asm) sum.Union_comp)
  using qq by (force simp: split_paired_all)+
  moreover have content M *R f x = 0
    if K ∈ r L ∈ r K ≠ L qq K = qq L (x, M) ∈ qq K for K L x M
    using tagged_division_ofD(6) qq that by (metis (no_types, lifting))
  ultimately have less_e: norm (?SUM p + sum (?SUM ∘ qq) r - integral (cbox
a b) f) < e
  proof (subst (asm) sum.reindex_nontrivial [OF ⟨finite r⟩])
    qed (auto simp: split_paired_all sum.neutral)
  have norm_le: norm (cp - ip) ≤ e + k
    if norm ((cp + cr) - i) < e norm (cr - ir) < k ip + ir = i
    for ir ip i cr cp::'a
    using norm_triangle_le[of cp + cr - i - (cr - ir)] that
    unfolding that(3)[symmetric] norm_minus_cancel
    by (auto simp add: algebra_simps)

  have ?lhs = norm (?SUM p - (∑ (x, k) ∈ p. integral k f))
    unfolding split_def sum_subtractf ..
  also have ... ≤ e + k
  proof (rule norm_le[OF less_e])
    have lessk: k * real (card r) / (1 + real (card r)) < k
      using ⟨k > 0⟩ by (auto simp add: field_simps)
    have norm (sum (?SUM ∘ qq) r - (∑ k ∈ r. integral k f)) ≤ (∑ x ∈ r. k / (real
(card r) + 1))
      unfolding sum_subtractf[symmetric] by (force dest: qq intro!: sum_norm_le)
    also have ... < k
      by (simp add: lessk add.commute mult.commute)
    finally show norm (sum (?SUM ∘ qq) r - (∑ k ∈ r. integral k f)) < k .
  next
  from q(1) have [simp]: snd ' p ∪ q = q by auto
  have integral l f = 0
    if inp: (x, l) ∈ p (y, m) ∈ p and ne: (x, l) ≠ (y, m) and l = m for x l y m
  proof -
    obtain u v where uv: l = cbox u v
      using inp p'(4) by blast
    then show ?thesis
      using uv that p
      by (metis content_eq_0_interior dual_order.refl inf.orderE integral_null
ne tagged_partial_division_ofD(5))

```

qed
then have $(\sum (x, K) \in p. \text{integral } K f) = (\sum K \in \text{snd } \langle p, \text{integral } K f \rangle)$
apply $(\text{subst } \text{sum.reindex_nontrivial } [OF \langle \text{finite } p \rangle])$
unfolding $\text{split_paired_all_split_def}$ **by** auto
then show $(\sum (x, k) \in p. \text{integral } k f) + (\sum k \in r. \text{integral } k f) = \text{integral } (\text{cbox } a \ b) f$
unfolding $\text{integral_combine_division_topdown}$ $[OF \text{intf } qdiv]$ r_def
using $q'(1)$ $p'(1)$ $\text{sum.union_disjoint}$ $[of \text{snd } \langle p \ q \rangle - \text{snd } \langle p, \text{symmetric} \rangle]$
by simp
qed
finally show $?lhs \leq e + k$.
qed

lemma *Henstock_lemma_part2*:

fixes $f :: 'm::\text{euclidean_space} \Rightarrow 'n::\text{euclidean_space}$
assumes $\text{fed}: f \text{integrable_on } \text{cbox } a \ b \ e > 0 \ \text{gauge } d$
and $\text{less_e}: \bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{tagged_division_of } (\text{cbox } a \ b); \ d \ \text{fine } \mathcal{D} \rrbracket \implies$
 $\text{norm } (\text{sum } (\lambda(x,k). \text{content } k *_R f \ x) \ \mathcal{D} - \text{integral } (\text{cbox } a \ b) f)$
 $< e$
and $\text{tag}: p \ \text{tagged_partial_division_of } (\text{cbox } a \ b)$
and $d \ \text{fine } p$
shows $\text{sum } (\lambda(x,k). \text{norm } (\text{content } k *_R f \ x - \text{integral } k f)) \ p \leq 2 * \text{real } (\text{DIM } ('n)) * e$
proof –
have $\text{finite } p$
using $\text{tag } \text{tagged_partial_division_of } \mathcal{D}$ **by** blast
then show $?thesis$
unfolding split_def
proof $(\text{rule } \text{sum_norm_allsubsets_bound})$
fix Q
assume $Q: Q \subseteq p$
then have $\text{fine}: d \ \text{fine } Q$
by $(\text{simp } \text{add: } \langle d \ \text{fine } p \rangle \ \text{fine_subset})$
show $\text{norm } (\sum x \in Q. \text{content } (\text{snd } x) *_R f \ (\text{fst } x) - \text{integral } (\text{snd } x) f) \leq e$
apply $(\text{rule } \text{Henstock_lemma_part1} [OF \ \text{fed } \text{less_e}, \ \text{unfolded } \text{split_def}])$
using $Q \ \text{tag } \text{tagged_partial_division_subset}$ **by** $(\text{force } \text{simp } \text{add: } \text{fine})+$
qed
qed

lemma *Henstock_lemma*:

fixes $f :: 'm::\text{euclidean_space} \Rightarrow 'n::\text{euclidean_space}$
assumes $\text{intf}: f \text{integrable_on } \text{cbox } a \ b$
and $e > 0$
obtains γ **where** $\text{gauge } \gamma$
and $\bigwedge p. \llbracket p \ \text{tagged_partial_division_of } (\text{cbox } a \ b); \ \gamma \ \text{fine } p \rrbracket \implies$
 $\text{sum } (\lambda(x,k). \text{norm}(\text{content } k *_R f \ x - \text{integral } k f)) \ p < e$
proof –
have $*$: $e / (2 * (\text{real } \text{DIM } ('n) + 1)) > 0$ **using** $\langle e > 0 \rangle$ **by** simp
with $\text{integrable_integral} [OF \ \text{intf}, \ \text{unfolded } \text{has_integral}]$

```

obtain  $\gamma$  where gauge  $\gamma$ 
  and  $\gamma: \bigwedge \mathcal{D}. [\mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; \ \gamma \text{ fine } \mathcal{D}] \implies$ 
     $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x) - \text{integral } (\text{cbox } a \ b) \ f)$ 
     $< e / (2 * (\text{real } \text{DIM}('n) + 1))$ 
  by metis
show thesis
proof (rule that [OF  $\langle \text{gauge } \gamma \rangle$ ])
  fix  $p$ 
  assume  $p: p \text{ tagged\_partial\_division\_of } \text{cbox } a \ b \ \gamma \text{ fine } p$ 
  have  $(\sum (x,K) \in p. \text{norm } (\text{content } K *_{\mathbb{R}} f \ x - \text{integral } K \ f))$ 
     $\leq 2 * \text{real } \text{DIM}('n) * (e / (2 * (\text{real } \text{DIM}('n) + 1)))$ 
  using Henstock_lemma_part2[OF  $\text{intf} * \langle \text{gauge } \gamma \rangle \ \gamma \ p$ ] by metis
  also have ...  $< e$ 
  using  $\langle e > 0 \rangle$  by (auto simp add: field_simps)
  finally
  show  $(\sum (x,K) \in p. \text{norm } (\text{content } K *_{\mathbb{R}} f \ x - \text{integral } K \ f)) < e .$ 
qed
qed

```

7.14.41 Monotone convergence (bounded interval first)

lemma bounded_increasing_convergent:

```

fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
shows  $[\text{bounded } (\text{range } f); \ \bigwedge n. f \ n \leq f \ (\text{Suc } n)] \implies \exists l. f \ \longrightarrow \ l$ 
using Bseq_mono_convergent[of  $f$ ] incseq_Suc_iff[of  $f$ ]
by (auto simp: image_def Bseq_eq_bounded convergent_def incseq_def)

```

lemma monotone_convergence_interval:

```

fixes  $f :: \text{nat} \Rightarrow 'n::\text{euclidean\_space} \Rightarrow \text{real}$ 
assumes  $\text{intf}: \bigwedge k. (f \ k) \text{ integrable\_on } \text{cbox } a \ b$ 
  and  $\text{le}: \bigwedge k \ x. x \in \text{cbox } a \ b \implies (f \ k \ x) \leq f \ (\text{Suc } k) \ x$ 
  and  $\text{fg}: \bigwedge x. x \in \text{cbox } a \ b \implies ((\lambda k. f \ k \ x) \longrightarrow g \ x) \text{ sequentially}$ 
  and  $\text{bou}: \text{bounded } (\text{range } (\lambda k. \text{integral } (\text{cbox } a \ b) \ (f \ k)))$ 
shows  $g \text{ integrable\_on } \text{cbox } a \ b \wedge ((\lambda k. \text{integral } (\text{cbox } a \ b) \ (f \ k)) \longrightarrow \text{integral } (\text{cbox } a \ b) \ g) \text{ sequentially}$ 
proof (cases  $\text{content } (\text{cbox } a \ b) = 0$ )
  case True then show ?thesis
    by auto
next
  case False
  have  $\text{fg1}: (f \ k \ x) \leq (g \ x) \text{ if } x: x \in \text{cbox } a \ b \text{ for } x \ k$ 
  proof -
    have  $\forall_F j \text{ in sequentially. } f \ k \ x \leq f \ j \ x$ 
  proof (rule eventually_sequentiallyI [of  $k$ ])
    show  $\bigwedge j. k \leq j \implies f \ k \ x \leq f \ j \ x$ 
    using  $\text{le } x$  by (force intro: transitive_stepwise_le)
  qed
  then show  $f \ k \ x \leq g \ x$ 
  using tendsto_lowerbound [OF  $\text{fg}$ ]  $x \text{ trivial\_limit\_sequentially}$  by blast

```

```

qed
have int_inc:  $\bigwedge n. \text{integral } (\text{cbox } a \ b) \ (f \ n) \leq \text{integral } (\text{cbox } a \ b) \ (f \ (\text{Suc } n))$ 
  by (metis integral_le_intf_le)
then obtain i where i:  $(\lambda k. \text{integral } (\text{cbox } a \ b) \ (f \ k)) \longrightarrow i$ 
  using bounded_increasing_convergent bou by blast
have  $\bigwedge k. \forall_F x \text{ in sequentially. } \text{integral } (\text{cbox } a \ b) \ (f \ k) \leq \text{integral } (\text{cbox } a \ b) \ (f \ x)$ 
  unfolding eventually_sequentially
  by (force intro: transitive_stepwise_le int_inc)
then have i':  $\bigwedge k. (\text{integral } (\text{cbox } a \ b) \ (f \ k)) \leq i$ 
  using tendsto_le [OF trivial_limit_sequentially i] by blast
have (g has_integral i) (cbox a b)
  unfolding has_integral_real_norm_def
proof clarify
  fix e::real
  assume e:  $e > 0$ 
  have  $\bigwedge k. (\exists \gamma. \text{gauge } \gamma \wedge (\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b) \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
 $\text{abs } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ k \ x) - \text{integral } (\text{cbox } a \ b) \ (f \ k)) < e/2 \wedge (k + 2)))$ 
  using intf_e by (auto simp: has_integral_integral has_integral)
  then obtain c where c:  $\bigwedge x. \text{gauge } (c \ x)$ 
 $\bigwedge x \ \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; \ c \ x \ \text{fine } \mathcal{D} \rrbracket \Longrightarrow$ 
 $\text{abs } ((\sum (u,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x \ u) - \text{integral } (\text{cbox } a \ b) \ (f \ x)) < e/2 \wedge (x + 2)$ 
  by metis

  have  $\exists r. \forall k \geq r. 0 \leq i - (\text{integral } (\text{cbox } a \ b) \ (f \ k)) \wedge i - (\text{integral } (\text{cbox } a \ b) \ (f \ k)) < e/4$ 
  proof -
  have  $e/4 > 0$ 
  using e by auto
  show ?thesis
  using LIMSEQ_D [OF i <e/4 > 0] i' by auto
qed
then obtain r where r:  $\bigwedge k. r \leq k \Longrightarrow 0 \leq i - \text{integral } (\text{cbox } a \ b) \ (f \ k)$ 
 $\bigwedge k. r \leq k \Longrightarrow i - \text{integral } (\text{cbox } a \ b) \ (f \ k) < e/4$ 
  by metis
have  $\exists n \geq r. \forall k \geq n. 0 \leq (g \ x) - (f \ k \ x) \wedge (g \ x) - (f \ k \ x) < e/(4 * \text{content}(\text{cbox } a \ b))$ 
  if  $x \in \text{cbox } a \ b$  for x
  proof -
  have  $e/(4 * \text{content } (\text{cbox } a \ b)) > 0$ 
  by (simp add: False_content_lt_nz e)
  with fg that LIMSEQ_D
  obtain N where  $\forall n \geq N. \text{norm } (f \ n \ x - g \ x) < e/(4 * \text{content } (\text{cbox } a \ b))$ 
  by metis
  then show  $\exists n \geq r. \forall k \geq n. 0 \leq g \ x - f \ k \ x \wedge g \ x - f \ k \ x < e/(4 * \text{content } (\text{cbox } a \ b))$ 

```

```

    apply (rule_tac x=N + r in exI)
    using fg1[OF that] by (auto simp add: field_simps)
  qed
  then obtain m where r_le_m:  $\bigwedge x. x \in \text{cbox } a \ b \implies r \leq m \ x$ 
    and m:  $\bigwedge x \ k. \llbracket x \in \text{cbox } a \ b; m \ x \leq k \rrbracket$ 
       $\implies 0 \leq g \ x - f \ k \ x \wedge g \ x - f \ k \ x < e/(4 * \text{content } (\text{cbox } a \ b))$ 
    by metis
  define d where d x = c (m x) x for x
  show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
     $(\forall \mathcal{D}. \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b \wedge$ 
       $\gamma \text{ fine } \mathcal{D} \longrightarrow \text{abs } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x) - i) < e)$ 
  proof (rule exI, safe)
    show gauge d
      using c(1) unfolding gauge_def d_def by auto
  next
    fix  $\mathcal{D}$ 
    assume ptag:  $\mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b)$  and d fine  $\mathcal{D}$ 
    note p'= $\text{tagged\_division\_of } \mathcal{D}$ [OF ptag]
    obtain s where s:  $\bigwedge x. x \in \mathcal{D} \implies m (\text{fst } x) \leq s$ 
    by (metis finite_imageI finite_nat_set_iff_bounded_le p'(1) rev_image_eqI)
    have *:  $|a - d| < e$  if  $|a - b| \leq e/4 \ |b - c| < e/2 \ |c - d| < e/4$  for  $a \ b \ c \ d$ 
      using that norm_triangle_lt[of a - b b - c 3* e/4]
        norm_triangle_lt[of a - b + (b - c) c - d e]
      by (auto simp add: algebra_simps)
    show  $|(\sum (x, k) \in \mathcal{D}. \text{content } k *_{\mathbb{R}} g \ x) - i| < e$ 
    proof (rule *)
      have  $|(\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x) - (\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f (m$ 
 $x) \ x)|$ 
         $\leq (\sum i \in \mathcal{D}. |(case \ i \ \text{of } (x, K) \Rightarrow \text{content } K *_{\mathbb{R}} g \ x) - (case \ i \ \text{of } (x, K)$ 
 $\Rightarrow \text{content } K *_{\mathbb{R}} f (m \ x) \ x)|)$ 
      by (metis (mono_tags) sum_subtractf sum_abs)
      also have ...  $\leq (\sum (x, k) \in \mathcal{D}. \text{content } k * (e/(4 * \text{content } (\text{cbox } a \ b))))$ 
    proof (rule sum_mono, simp add: split_paired_all)
      fix  $x \ K$ 
      assume xk:  $(x, K) \in \mathcal{D}$ 
      with ptag have x:  $x \in \text{cbox } a \ b$ 
      by blast
      then have abs (content K * (g x - f (m x) x))  $\leq \text{content } K * (e/(4 * \text{content } (\text{cbox } a \ b)))$ 
      by (metis m[OF x] mult_nonneg_nonneg abs_of_nonneg less_eq_real_def measure_nonneg_mult_left_mono_order_refl)
      then show  $|\text{content } K * g \ x - \text{content } K * f (m \ x) \ x| \leq \text{content } K * e/(4 * \text{content } (\text{cbox } a \ b))$ 
      by (simp add: algebra_simps)
    qed
    also have ... =  $(e/(4 * \text{content } (\text{cbox } a \ b))) * (\sum (x, k) \in \mathcal{D}. \text{content } k)$ 
    by (simp add: sum_distrib_left sum_divide_distrib split_def mult commute)
    also have ...  $\leq e/4$ 
    by (metis False additive_content_tagged_division [OF ptag] nonzero_mult_divide_mult_cancel_right

```

```

order_refl times_divide_eq_left)
  finally show |( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R g x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R f (m x) x$ )|  $\leq e/4$  .

  next
  have norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R f (m x) x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{integral } K (f (m x))$ ))
     $\leq$  norm ( $\sum j = 0..s. \sum (x,K) \in \{xk \in \mathcal{D}. m (fst xk) = j\}. \text{content } K *_R f (m x) x - \text{integral } K (f (m x))$ )
    apply (subst sum.group)
    using s by (auto simp: sum_subtractf split_def p'(1))
  also have ...  $< e/2$ 
  proof -
    have norm ( $\sum j = 0..s. \sum (x, k) \in \{xk \in \mathcal{D}. m (fst xk) = j\}. \text{content } k *_R f (m x) x - \text{integral } k (f (m x))$ )
       $\leq$  ( $\sum i = 0..s. e/2 ^ (i + 2)$ )
    proof (rule sum_norm_le)
      fix t
      assume t  $\in \{0..s\}$ 
      have norm ( $\sum (x,k) \in \{xk \in \mathcal{D}. m (fst xk) = t\}. \text{content } k *_R f (m x) x - \text{integral } k (f (m x))$ ) =
        norm ( $\sum (x,k) \in \{xk \in \mathcal{D}. m (fst xk) = t\}. \text{content } k *_R f t x - \text{integral } k (f t)$ )
      by (force intro!: sum.cong arg_cong[where f=norm])
      also have ...  $\leq e/2 ^ (t + 2)$ 
      proof (rule Henstock_lemma_part1 [OF intf])
        show  $\{xk \in \mathcal{D}. m (fst xk) = t\}$  tagged_partial_division_of cbox a b
        proof (rule tagged_partial_division_subset[of  $\mathcal{D}$ ])
          show  $\mathcal{D}$  tagged_partial_division_of cbox a b
          using ptag tagged_division_of_def by blast
        qed auto
        show c t fine  $\{xk \in \mathcal{D}. m (fst xk) = t\}$ 
        using  $\langle d \text{ fine } \mathcal{D} \rangle$  by (auto simp: fine_def d_def)
      qed (use c e in auto)
      finally show norm ( $\sum (x,K) \in \{xk \in \mathcal{D}. m (fst xk) = t\}. \text{content } K *_R f (m x) x - \text{integral } K (f (m x))$ )  $\leq e/2 ^ (t + 2)$  .
    qed
  also have ... = ( $e/2/2$ ) * ( $\sum i = 0..s. (1/2) ^ i$ )
    by (simp add: sum_distrib_left field_simps)
  also have ...  $< e/2$ 
    by (simp add: sum_gp_mult_strict_left_mono[OF _ e])
  finally show norm ( $\sum j = 0..s. \sum (x, k) \in \{xk \in \mathcal{D}. m (fst xk) = j\}. \text{content } k *_R f (m x) x - \text{integral } k (f (m x))$ )  $< e/2$  .
  qed
  finally show |( $\sum (x,K) \in \mathcal{D}. \text{content } K *_R f (m x) x$ ) - ( $\sum (x,K) \in \mathcal{D}. \text{integral } K (f (m x))$ )|  $< e/2$ 
    by simp
  next

```



```

have comb: integral (cbox a b) (f y) = ( $\sum (x, k) \in \mathcal{D}. \text{integral } k (f y)$ ) for y
  using integral_combine_tagged_division_topdown[OF intf_ptag] by metis
have f_le:  $\bigwedge y m n. \llbracket y \in \text{cbox } a b; n \geq m \rrbracket \implies f m y \leq f n y$ 
  using le by (auto intro: transitive_stepwise_le)
have ( $\sum (x, k) \in \mathcal{D}. \text{integral } k (f r)$ )  $\leq$  ( $\sum (x, K) \in \mathcal{D}. \text{integral } K (f (m x))$ )
proof (rule sum_mono, simp add: split_paired_all)
  fix x K
  assume xK:  $(x, K) \in \mathcal{D}$ 
  show integral K (f r)  $\leq$  integral K (f (m x))
  proof (rule integral_le)
    show f r integrable_on K
      by (metis integrable_on_subcbox intf p'(3) p'(4) xK)
    show f (m x) integrable_on K
      by (metis elementary_interval integrable_on_subdivision intf p'(3)
p'(4) xK)
    show f r y  $\leq$  f (m x) y if y  $\in$  K for y
      using that r_le_m[of x] p'(2-3)[OF xK] f_le by auto
  qed
qed
moreover have ( $\sum (x, K) \in \mathcal{D}. \text{integral } K (f (m x))$ )  $\leq$  ( $\sum (x, k) \in \mathcal{D}. \text{integral } k (f s)$ )
proof (rule sum_mono, simp add: split_paired_all)
  fix x K
  assume xK:  $(x, K) \in \mathcal{D}$ 
  show integral K (f (m x))  $\leq$  integral K (f s)
  proof (rule integral_le)
    show f (m x) integrable_on K
      by (metis elementary_interval integrable_on_subdivision intf p'(3)
p'(4) xK)
    show f s integrable_on K
      by (metis integrable_on_subcbox intf p'(3) p'(4) xK)
    show f (m x) y  $\leq$  f s y if y  $\in$  K for y
      using that s xK f_le p'(3) by fastforce
  qed
qed
moreover have  $0 \leq i - \text{integral } (\text{cbox } a b) (f r) i - \text{integral } (\text{cbox } a b) (f r) < e/4$ 
  using r by auto
  ultimately show  $|\sum (x, K) \in \mathcal{D}. \text{integral } K (f (m x)) - i| < e/4$ 
    using comb i'[of s] by auto
  qed
qed
with i integral_unique show ?thesis
  by blast
qed

lemma monotone_convergence_increasing:
  fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  real

```

```

assumes int_f:  $\bigwedge k. (f\ k)\ \text{integrable\_on}\ S$ 
and  $\bigwedge k\ x. x \in S \implies (f\ k\ x) \leq (f\ (\text{Suc}\ k)\ x)$ 
and fg:  $\bigwedge x. x \in S \implies ((\lambda k. f\ k\ x) \longrightarrow g\ x)\ \text{sequentially}$ 
and bou: bounded (range ( $\lambda k. \text{integral}\ S\ (f\ k)$ ))
shows g integrable_on S  $\wedge ((\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow \text{integral}\ S\ g)\ \text{sequentially}$ 
proof -
  have lem: g integrable_on S  $\wedge ((\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow \text{integral}\ S\ g)\ \text{sequentially}$ 
  if f0:  $\bigwedge k\ x. x \in S \implies 0 \leq f\ k\ x$ 
  and int_f:  $\bigwedge k. (f\ k)\ \text{integrable\_on}\ S$ 
  and le:  $\bigwedge k\ x. x \in S \implies f\ k\ x \leq f\ (\text{Suc}\ k)\ x$ 
  and lim:  $\bigwedge x. x \in S \implies ((\lambda k. f\ k\ x) \longrightarrow g\ x)\ \text{sequentially}$ 
  and bou: bounded (range ( $\lambda k. \text{integral}\ S\ (f\ k)$ ))
  for f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  real and g S
proof -
  have fg:  $(f\ k\ x) \leq (g\ x)$  if x  $\in S$  for x k
  proof -
    have  $\bigwedge xa. k \leq xa \implies f\ k\ x \leq f\ xa\ x$ 
    using le by (force intro: transitive_stepwise_le that)
    then show ?thesis
    using tendsto_lowerbound [OF lim [OF that]] eventually_sequentiallyI by
force
  qed
  obtain i where i:  $(\lambda k. \text{integral}\ S\ (f\ k)) \longrightarrow i$ 
  using bounded_increasing_convergent [OF bou] le int_f integral_le by blast
  have i':  $(\text{integral}\ S\ (f\ k)) \leq i$  for k
  proof -
    have  $\bigwedge k. \bigwedge x. x \in S \implies \forall n \geq k. f\ k\ x \leq f\ n\ x$ 
    using le by (force intro: transitive_stepwise_le)
    then show ?thesis
    using tendsto_lowerbound [OF i eventually_sequentiallyI trivial_limit_sequentially]
    by (meson int_f integral_le)
  qed
  let ?f =  $(\lambda k\ x. \text{if } x \in S \text{ then } f\ k\ x \text{ else } 0)$ 
  let ?g =  $(\lambda x. \text{if } x \in S \text{ then } g\ x \text{ else } 0)$ 
  have int: ?f k integrable_on cbox a b for a b k
  by (simp add: int_f integrable_altD(1))
  have int':  $\bigwedge k\ a\ b. f\ k\ \text{integrable\_on}\ \text{cbox}\ a\ b \cap S$ 
  using int by (simp add: Int_commute integrable_restrict_Int)
  have g: ?g integrable_on cbox a b  $\wedge$ 
     $(\lambda k. \text{integral}\ (\text{cbox}\ a\ b)\ (?f\ k)) \longrightarrow \text{integral}\ (\text{cbox}\ a\ b)\ ?g$  for a b
  proof (rule monotone_convergence_interval)
    have norm  $(\text{integral}\ (\text{cbox}\ a\ b)\ (?f\ k)) \leq \text{norm}\ (\text{integral}\ S\ (f\ k))$  for k
    proof -
      have  $0 \leq \text{integral}\ (\text{cbox}\ a\ b)\ (?f\ k)$ 
      by (metis (no_types) integral_nonneg Int_iff f0 inf_commute integral_restrict_Int int')
      moreover have  $0 \leq \text{integral}\ S\ (f\ k)$ 
      by (simp add: integral_nonneg f0 int_f)
    
```

```

moreover have  $\text{integral } (S \cap \text{cbox } a \ b) \ (f \ k) \leq \text{integral } S \ (f \ k)$ 
  by (metis f0 inf_commute int' int_f integral_subset_le le_inf_iff order_refl)
ultimately show ?thesis
  by (simp add: integral_restrict_Int)
qed
moreover obtain  $B$  where  $\bigwedge x. x \in \text{range } (\lambda k. \text{integral } S \ (f \ k)) \implies \text{norm } x \leq B$ 
  using bow unfolding bounded_iff by blast
ultimately show  $\text{bounded } (\text{range } (\lambda k. \text{integral } (\text{cbox } a \ b) \ (?f \ k)))$ 
  unfolding bounded_iff by (blast intro: order_trans)
qed (use int le lim in auto)
moreover have  $\exists B > 0. \forall a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \longrightarrow \text{norm } (\text{integral } (\text{cbox } a \ b) \ ?g - i) < e$ 
  if  $0 < e$  for  $e$ 
proof -
  have  $e/4 > 0$ 
    using that by auto
  with LIMSEQ_D [OF  $i$ ] obtain  $N$  where  $N: \bigwedge n. n \geq N \implies \text{norm } (\text{integral } S \ (f \ n) - i) < e/4$ 
    by metis
  with int_f[of  $N$ , unfolded has_integral_integral has_integral_alt'[of  $f \ N$ ]]
  obtain  $B$  where  $0 < B$  and  $B$ :
     $\bigwedge a \ b. \text{ball } 0 \ B \subseteq \text{cbox } a \ b \implies \text{norm } (\text{integral } (\text{cbox } a \ b) \ (?f \ N) - \text{integral } S \ (f \ N)) < e/4$ 
    by (meson  $\langle 0 < e/4 \rangle$ )
  have  $\text{norm } (\text{integral } (\text{cbox } a \ b) \ ?g - i) < e$  if  $ab: \text{ball } 0 \ B \subseteq \text{cbox } a \ b$  for  $a \ b$ 
proof -
    obtain  $M$  where  $M: \bigwedge n. n \geq M \implies \text{abs } (\text{integral } (\text{cbox } a \ b) \ (?f \ n) - \text{integral } (\text{cbox } a \ b) \ ?g) < e/2$ 
      using  $\langle e > 0 \rangle$   $g$  by (fastforce simp add: dest!: LIMSEQ_D [where  $r = e/2$ ])
    have  $*$ :  $\bigwedge \alpha \ \beta \ g. [\![\alpha - i] < e/2; |\beta - g| < e/2; \alpha \leq \beta; \beta \leq i]\!] \implies |g - i| < e$ 
      unfolding real_inner_1_right by arith
    show  $\text{norm } (\text{integral } (\text{cbox } a \ b) \ ?g - i) < e$ 
      unfolding real_norm_def
    proof (rule  $*$ )
      show  $|\text{integral } (\text{cbox } a \ b) \ (?f \ N) - i| < e/2$ 
        proof (rule abs_triangle_half_l)
          show  $|\text{integral } (\text{cbox } a \ b) \ (?f \ N) - \text{integral } S \ (f \ N)| < e/2/2$ 
            using  $B$ [OF  $ab$ ] by simp
          show  $\text{abs } (i - \text{integral } S \ (f \ N)) < e/2/2$ 
            using  $N$  by (simp add: abs_minus_commute)
        qed
      show  $|\text{integral } (\text{cbox } a \ b) \ (?f \ (M + N)) - \text{integral } (\text{cbox } a \ b) \ ?g| < e/2$ 
        by (metis le_add1  $M$ [of  $M + N$ ])
      show  $\text{integral } (\text{cbox } a \ b) \ (?f \ N) \leq \text{integral } (\text{cbox } a \ b) \ (?f \ (M + N))$ 
        proof (intro ballI integral_le[OF int int])

```

```

fix x assume x ∈ cbox a b
have (f m x) ≤ (f n x) if x ∈ S n ≥ m for m n
proof (rule transitive_stepwise_le [OF ⟨n ≥ m⟩ order_refl])
  show  $\bigwedge u y z. \llbracket f u x \leq f y x; f y x \leq f z x \rrbracket \implies f u x \leq f z x$ 
    using dual_order.trans by blast
qed (simp add: le ⟨x ∈ S⟩)
then show (?f N)x ≤ (?f (M+N))x
  by auto
qed
have integral (cbox a b ∩ S) (f (M + N)) ≤ integral S (f (M + N))
  by (metis Int_lower1 f0 inf_commute int' int_f integral_subset_le)
then have integral (cbox a b) (?f (M + N)) ≤ integral S (f (M + N))
  by (metis (no_types) inf_commute integral_restrict_Int)
also have ... ≤ i
  using i'[of M + N] by auto
finally show integral (cbox a b) (?f (M + N)) ≤ i .
qed
qed
then show ?thesis
  using ⟨0 < B⟩ by blast
qed
ultimately have (g has_integral i) S
  unfolding has_integral_alt' by auto
then show ?thesis
  using has_integral_integrable_integral i integral_unique by metis
qed

have sub:  $\bigwedge k. \text{integral } S (\lambda x. f k x - f 0 x) = \text{integral } S (f k) - \text{integral } S (f 0)$ 
  by (simp add: integral_diff int_f)
have *:  $\bigwedge x m n. x \in S \implies n \geq m \implies f m x \leq f n x$ 
  using assms(2) by (force intro: transitive_stepwise_le)
have gf:  $(\lambda x. g x - f 0 x) \text{integrable\_on } S \wedge ((\lambda k. \text{integral } S (\lambda x. f (Suc k) x - f 0 x)) \longrightarrow \text{integral } S (\lambda x. g x - f 0 x)) \longrightarrow$ 
   $\text{integral } S (\lambda x. g x - f 0 x)$  sequentially
proof (rule lem)
  show  $\bigwedge k. (\lambda x. f (Suc k) x - f 0 x) \text{integrable\_on } S$ 
    by (simp add: integrable_diff int_f)
  show  $(\lambda k. f (Suc k) x - f 0 x) \longrightarrow g x - f 0 x$  if x ∈ S for x
  proof -
    have  $(\lambda n. f (Suc n) x) \longrightarrow g x$ 
      using LIMSEQ_ignore_initial_segment[OF fg[OF ⟨x ∈ S⟩], of 1] by simp
    then show ?thesis
      by (simp add: tendsto_diff)
  qed
show bounded (range  $(\lambda k. \text{integral } S (\lambda x. f (Suc k) x - f 0 x))$ )
proof -
  obtain B where B:  $\bigwedge k. \text{norm } (\text{integral } S (f k)) \leq B$ 
    using bou by (auto simp: bounded_iff)
  then have norm (integral S  $(\lambda x. f (Suc k) x - f 0 x)$ )

```

```

      ≤ B + norm (integral S (f 0)) for k
    unfolding sub by (meson add_le_cancel_right norm_triangle_le_diff)
  then show ?thesis
    unfolding bounded_iff by blast
  qed
  qed (use * in auto)
  then have (λx. integral S (λxa. f (Suc x) xa - f 0 xa) + integral S (f 0))
    ———> integral S (λx. g x - f 0 x) + integral S (f 0)
    by (auto simp add: tendsto_add)
  moreover have (λx. g x - f 0 x + f 0 x) integrable_on S
    using gf_integrable_add_int_f [of 0] by metis
  ultimately show ?thesis
    by (simp add: integral_diff_int_f LIMSEQ_imp_Suc sub)
  qed

```

lemma *has_integral_monotone_convergence_increasing*:

```

  fixes f :: nat ⇒ 'a::euclidean_space ⇒ real
  assumes f: ∧k. (f k has_integral x k) s
  assumes l: ∧k x. x ∈ s ⇒ f k x ≤ f (Suc k) x
  assumes r: ∧x. x ∈ s ⇒ (λk. f k x) ———> g x
  assumes x ———> x'
  shows (g has_integral x') s
  proof -
    have x_eq: x = (λi. integral s (f i))
      by (simp add: integral_unique[OF f])
    then have x: range(λk. integral s (f k)) = range x
      by auto
    have *: g integrable_on s ∧ (λk. integral s (f k)) ———> integral s g
    proof (intro monotone_convergence_increasing allI ballI assms)
      show bounded (range(λk. integral s (f k)))
        using x convergent_imp_bounded assms by metis
    qed (use f in auto)
    then have integral s g = x'
      by (intro LIMSEQ_unique[OF _ ⟨x ———> x'⟩]) (simp add: x_eq)
    with * show ?thesis
      by (simp add: has_integral_integral)
  qed

```

lemma *monotone_convergence_decreasing*:

```

  fixes f :: nat ⇒ 'n::euclidean_space ⇒ real
  assumes intf: ∧k. (f k) integrable_on S
  and le: ∧k x. x ∈ S ⇒ f (Suc k) x ≤ f k x
  and fg: ∧x. x ∈ S ⇒ ((λk. f k x) ———> g x) sequentially
  and bou: bounded (range(λk. integral S (f k)))
  shows g integrable_on S ∧ (λk. integral S (f k)) ———> integral S g
  proof -
    have *: range(λk. integral S (λx. - f k x)) = (*R) (- 1) ‘ (range(λk. integral S
      (f k)))
      by force

```

```

have  $(\lambda x. - g x)$  integrable_on  $S \wedge (\lambda k. \text{integral } S (\lambda x. - f k x)) \longrightarrow \text{integral } S (\lambda x. - g x)$ 
proof (rule monotone_convergence_increasing)
  show  $\bigwedge k. (\lambda x. - f k x)$  integrable_on  $S$ 
    by (blast intro: integrable_neg intf)
  show  $\bigwedge k x. x \in S \implies - f k x \leq - f (Suc k) x$ 
    by (simp add: le)
  show  $\bigwedge x. x \in S \implies (\lambda k. - f k x) \longrightarrow - g x$ 
    by (simp add: fg tendsto_minus)
  show bounded (range( $\lambda k. \text{integral } S (\lambda x. - f k x)$ ))
    using * bou bounded_scaling by auto
qed
then show ?thesis
  by (force dest: integrable_neg tendsto_minus)
qed

```

lemma *integral_norm_bound_integral*:

```

fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'a::\text{banach}$ 
assumes int_f:  $f$  integrable_on  $S$ 
  and int_g:  $g$  integrable_on  $S$ 
  and le_g:  $\bigwedge x. x \in S \implies \text{norm } (f x) \leq g x$ 
shows  $\text{norm } (\text{integral } S f) \leq \text{integral } S g$ 
proof -
  have norm:  $\text{norm } \eta \leq y + e$ 
    if  $\text{norm } \zeta \leq x$  and  $|x - y| < e/2$  and  $\text{norm } (\zeta - \eta) < e/2$ 
    for  $e x y$  and  $\zeta \eta :: 'a$ 
  proof -
    have  $\text{norm } (\eta - \zeta) < e/2$ 
      by (metis norm_minus_commute that(3))
    moreover have  $x \leq y + e/2$ 
      using that(2) by linarith
    ultimately show ?thesis
      using that(1) le_less_trans[OF norm_triangle_sub[of  $\eta$   $\zeta$ ]] by (auto simp: less_imp_le)
  qed
have lem:  $\text{norm } (\text{integral } (\text{cbox } a b) f) \leq \text{integral } (\text{cbox } a b) g$ 
  if  $f$ :  $f$  integrable_on  $\text{cbox } a b$ 
  and  $g$ :  $g$  integrable_on  $\text{cbox } a b$ 
  and nle:  $\bigwedge x. x \in \text{cbox } a b \implies \text{norm } (f x) \leq g x$ 
  for  $f :: 'n \Rightarrow 'a$  and  $g a b$ 
proof (rule field_le_epsilon)
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  then have  $e: e/2 > 0$ 
    by auto
  with integrable_integral[OF f,unfolded has_integral[of f]]
  obtain  $\gamma$  where gauge  $\gamma$ 
     $\bigwedge \mathcal{D}. \mathcal{D}$  tagged_division_of  $\text{cbox } a b \wedge \gamma$  fine  $\mathcal{D}$ 
     $\implies \text{norm } ((\sum (x, k) \in \mathcal{D}. \text{content } k *_R f x) - \text{integral } (\text{cbox } a b) f) < e/2$ 

```

```

    by meson
  moreover
  from integrable_integral[OF g,unfolded has_integral[of g]] e
  obtain  $\delta$  where  $\delta$ : gauge  $\delta$ 
     $\wedge \mathcal{D}. \mathcal{D}$  tagged_division_of_cbox a b  $\wedge \delta$  fine  $\mathcal{D}$ 
     $\implies \text{norm} ((\sum (x, k) \in \mathcal{D}. \text{content } k *_R g x) - \text{integral } (\text{cbox } a b) g) < e/2$ 
  by meson
  ultimately have gauge  $(\lambda x. \gamma x \cap \delta x)$ 
    using gauge_Int by blast
  with fine_division_exists obtain  $\mathcal{D}$ 
    where  $p: \mathcal{D}$  tagged_division_of_cbox a b  $(\lambda x. \gamma x \cap \delta x)$  fine  $\mathcal{D}$ 
  by metis
  have  $\gamma$  fine  $\mathcal{D}$   $\delta$  fine  $\mathcal{D}$ 
    using fine_Int p(2) by blast+
  show norm (integral (cbox a b) f)  $\leq$  integral (cbox a b) g + e
  proof (rule norm)
    have norm (content K *_R f x)  $\leq$  content K *_R g x if  $(x, K) \in \mathcal{D}$  for  $x K$ 
    proof-
      have  $K: x \in K K \subseteq \text{cbox } a b$ 
        using  $\langle (x, K) \in \mathcal{D} \rangle$  p(1) by blast+
      obtain u v where  $K = \text{cbox } u v$ 
        using  $\langle (x, K) \in \mathcal{D} \rangle$  p(1) by blast
      moreover have content K * norm (f x)  $\leq$  content K * g x
        by (meson K(1) K(2) content_pos_le mult_left_mono nle_subsetD)
      then show ?thesis
        by simp
    qed
  qed
  then show norm  $(\sum (x, k) \in \mathcal{D}. \text{content } k *_R f x) \leq (\sum (x, k) \in \mathcal{D}. \text{content } k *_R g x)$ 
    by (simp add: sum_norm_le split_def)
  show norm  $((\sum (x, k) \in \mathcal{D}. \text{content } k *_R f x) - \text{integral } (\text{cbox } a b) f) < e/2$ 
    using  $\langle \gamma \text{ fine } \mathcal{D} \rangle \gamma$  p(1) by simp
  show  $|(\sum (x, k) \in \mathcal{D}. \text{content } k *_R g x) - \text{integral } (\text{cbox } a b) g| < e/2$ 
    using  $\langle \delta \text{ fine } \mathcal{D} \rangle \delta$  p(1) by simp
  qed
  qed
  show ?thesis
  proof (rule field_le_epsilon)
    fix e :: real
    assume e > 0
    then have e: e/2 > 0
      by auto
    let ?f =  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0)$ 
    let ?g =  $(\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0)$ 
    have f: ?f integrable_on cbox a b and g: ?g integrable_on cbox a b for a b
      using int_f int_g integrable_altD by auto
    obtain Bf where 0 < Bf
      and Bf:  $\wedge a b. \text{ball } 0 Bf \subseteq \text{cbox } a b \implies$ 
       $\exists z. (?f \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - \text{integral } S f) < e/2$ 

```

```

    using integrable_integral [OF int_f,unfolded has_integral[of f]] e that by
blast
  obtain Bg where 0 < Bg
  and Bg:  $\bigwedge a b. \text{ball } 0 Bg \subseteq \text{cbox } a b \implies$ 
     $\exists z. (?g \text{ has\_integral } z) (\text{cbox } a b) \wedge \text{norm } (z - \text{integral } S g) < e/2$ 
  using integrable_integral [OF int_g,unfolded has_integral[of g]] e that by
blast
  obtain a b::'n where ab:  $\text{ball } 0 Bf \cup \text{ball } 0 Bg \subseteq \text{cbox } a b$ 
  using ball_max_Un by (metis bounded_ball bounded_subset_cbox_symmetric)
  have  $\text{ball } 0 Bf \subseteq \text{cbox } a b$ 
  using ab by auto
  with Bf obtain z where int_fz:  $(?f \text{ has\_integral } z) (\text{cbox } a b)$  and z:  $\text{norm } (z - \text{integral } S f) < e/2$ 
  by meson
  have  $\text{ball } 0 Bg \subseteq \text{cbox } a b$ 
  using ab by auto
  with Bg obtain w where int_gw:  $(?g \text{ has\_integral } w) (\text{cbox } a b)$  and w:  $\text{norm } (w - \text{integral } S g) < e/2$ 
  by meson
  show  $\text{norm } (\text{integral } S f) \leq \text{integral } S g + e$ 
  proof (rule norm)
    show  $\text{norm } (\text{integral } (\text{cbox } a b) ?f) \leq \text{integral } (\text{cbox } a b) ?g$ 
    by (simp add: le_g lem[OF f g, of a b])
    show  $|\text{integral } (\text{cbox } a b) ?g - \text{integral } S g| < e/2$ 
    using int_gw integral_unique w by auto
    show  $\text{norm } (\text{integral } (\text{cbox } a b) ?f - \text{integral } S f) < e/2$ 
    using int_fz integral_unique z by blast
  qed
qed
qed

```

lemma *continuous_on_imp_absolutely_integrable_on*:

```

  fixes f::real  $\Rightarrow$  'a::banach
  shows continuous_on {a..b} f  $\implies$ 
     $\text{norm } (\text{integral } \{a..b\} f) \leq \text{integral } \{a..b\} (\lambda x. \text{norm } (f x))$ 
  by (intro integral_norm_bound_integral_integrable_continuous_real_continuous_on_norm)
  auto

```

lemma *integral_bound*:

```

  fixes f::real  $\Rightarrow$  'a::banach
  assumes  $a \leq b$ 
  assumes continuous_on {a .. b} f
  assumes  $\bigwedge t. t \in \{a .. b\} \implies \text{norm } (f t) \leq B$ 
  shows  $\text{norm } (\text{integral } \{a .. b\} f) \leq B * (b - a)$ 
  proof -
    note continuous_on_imp_absolutely_integrable_on[OF assms(2)]
    also have  $\text{integral } \{a..b\} (\lambda x. \text{norm } (f x)) \leq \text{integral } \{a..b\} (\lambda_. B)$ 
    by (rule integral_le)
    (auto intro!: integrable_continuous_real_continuous_intros assms)
  qed

```



```

    also have ... = B * (b - a) using assms by simp
    finally show ?thesis .
qed

lemma integral_norm_bound_integral_component:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  fixes g :: 'n  $\Rightarrow$  'b::euclidean_space
  assumes f: f integrable_on S and g: g integrable_on S
    and fg:  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq (g x) \cdot k$ 
  shows norm (integral S f)  $\leq$  (integral S g)  $\cdot k$ 
proof -
  have norm (integral S f)  $\leq$  integral S (( $\lambda x. x \cdot k$ )  $\circ$  g)
    using integral_norm_bound_integral[OF f integrable_linear[OF g]]
    by (simp add: bounded_linear_inner_left fg)
  then show ?thesis
    unfolding o_def integral_component_eq[OF g] .
qed

lemma has_integral_norm_bound_integral_component:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'a::banach
  fixes g :: 'n  $\Rightarrow$  'b::euclidean_space
  assumes (f has_integral i) S and (g has_integral j) S
    and  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq (g x) \cdot k$ 
  shows norm i  $\leq$  j  $\cdot k$ 
by (metis assms has_integral_integrable integral_norm_bound_integral_component
integral_unique)

lemma uniformly_convergent_improper_integral:
  fixes f :: 'b  $\Rightarrow$  real  $\Rightarrow$  'a :: {banach}
  assumes deriv:  $\bigwedge x. x \geq a \implies (G \text{ has\_field\_derivative } g x)$  (at x within {a..})
  assumes integrable:  $\bigwedge a' b x. x \in A \implies a' \geq a \implies b \geq a' \implies f x \text{ integrable\_on } \{a'..b\}$ 
  assumes G: convergent G
  assumes le:  $\bigwedge y x. y \in A \implies x \geq a \implies \text{norm}(f y x) \leq g x$ 
  shows uniformly_convergent_on A ( $\lambda b x. \text{integral } \{a..b\} (f x)$ )
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI', goal_cases)
  case (1  $\varepsilon$ )
  from G have Cauchy G
    by (auto intro!: convergent_Cauchy)
  with 1 obtain M where M: dist (G (real m)) (G (real n))  $< \varepsilon$  if  $m \geq M$   $n \geq M$  for m n
  by (force simp: Cauchy_def)
  define M' where M' = max (nat  $\lceil a \rceil$ ) M

  show ?case
proof (rule exI[of _ M'], safe, goal_cases)
  case (1 x m n)
  have M': M'  $\geq a$  M'  $\geq M$  unfolding M'_def by linarith+

```

```

have int_g: (g has_integral (G (real n) - G (real m))) {real m..real n}
  using 1 M' by (intro fundamental_theorem_of_calculus)
                (auto simp: has_real_derivative_iff_has_vector_derivative
[symmetric]
                intro!: DERIV_subset[OF deriv])
have int_f: f x integrable_on {a'..real n} if a' ≥ a for a'
  using that 1 by (cases a' ≤ real n) (auto intro: integrable)

have dist (integral {a..real m} (f x)) (integral {a..real n} (f x)) =
  norm (integral {a..real n} (f x) - integral {a..real m} (f x))
  by (simp add: dist_norm norm_minus_commute)
also have integral {a..real m} (f x) + integral {real m..real n} (f x) =
  integral {a..real n} (f x)
  using M' and 1 by (intro integral_combine int_f) auto
hence integral {a..real n} (f x) - integral {a..real m} (f x) =
  integral {real m..real n} (f x)
  by (simp add: algebra_simps)
also have norm ... ≤ integral {real m..real n} g
  using le 1 M' int_f int_g by (intro integral_norm_bound_integral) auto
also from int_g have integral {real m..real n} g = G (real n) - G (real m)
  by (simp add: has_integral_iff)
also have ... ≤ dist (G m) (G n)
  by (simp add: dist_norm)
also from 1 and M' have ... < ε
  by (intro M) auto
finally show ?case .
qed
qed

```

```

lemma uniformly_convergent_improper_integral':
  fixes f :: 'b ⇒ real ⇒ 'a :: {banach, real_normed_algebra}
  assumes deriv: ∧x. x ≥ a ⇒ (G has_field_derivative g x) (at x within {a..})
  assumes integrable: ∧a' b x. x ∈ A ⇒ a' ≥ a ⇒ b ≥ a' ⇒ f x integrable_on
{a'..b}
  assumes G: convergent G
  assumes le: eventually (λx. ∀y∈A. norm (f y x) ≤ g x) at_top
  shows uniformly_convergent_on A (λb x. integral {a..b} (f x))
proof -
  from le obtain a'' where le: ∧y x. y ∈ A ⇒ x ≥ a'' ⇒ norm (f y x) ≤ g x
  by (auto simp: eventually_at_top_linorder)
  define a' where a' = max a a''

```

```

have uniformly_convergent_on A (λb x. integral {a'..real b} (f x))
proof (rule uniformly_convergent_improper_integral)
  fix t assume t: t ≥ a'
  hence (G has_field_derivative g t) (at t within {a..})
  by (intro deriv) (auto simp: a'_def)
  moreover have {a'..} ⊆ {a..} unfolding a'_def by auto
  ultimately show (G has_field_derivative g t) (at t within {a'..})

```

```

    by (rule DERIV_subset)
  qed (insert le, auto simp: a'_def intro: integrable G)
  hence uniformly_convergent_on A ( $\lambda b x. \text{integral } \{a..a'\} (f x) + \text{integral } \{a'..real\} (f x)$ )
  (is ?P) by (intro uniformly_convergent_add) auto
  also have eventually ( $\lambda x. \forall y \in A. \text{integral } \{a..a'\} (f y) + \text{integral } \{a'..x\} (f y) =$ 
     $\text{integral } \{a..x\} (f y)$ ) sequentially
  by (intro eventually_mono [OF eventually_ge_at_top[of nat [a']]]) ballI integrals_combine)
  (auto simp: a'_def intro: integrable)
  hence ?P  $\longleftrightarrow$  ?thesis
  by (intro uniformly_convergent_cong) simp_all
  finally show ?thesis .
qed

```

7.14.42 differentiation under the integral sign

lemma *integral_continuous_on_param*:

```

fixes f::'a::topological_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::banach
assumes cont_fx: continuous_on (U  $\times$  cbox a b) ( $\lambda(x, t). f x t$ )
shows continuous_on U ( $\lambda x. \text{integral } (\text{cbox } a \text{ } b) (f x)$ )

```

proof cases

```

assume content (cbox a b)  $\neq$  0
then have ne: cbox a b  $\neq$  {} by auto

```

```

note [continuous_intros] =
  continuous_on_compose2[OF cont_fx, where f= $\lambda y.$  Pair x y for x,
    unfolded split_beta fst_conv snd_conv]

```

show ?thesis

```

unfolding continuous_on_def

```

```

proof (intro strip tendstoI)

```

```

fix e'::real and x

```

```

assume e' > 0

```

```

define e where e = e' / (content (cbox a b) + 1)

```

```

have e > 0 using ⟨e' > 0⟩ by (auto simp: e_def intro!: divide_pos_pos
add_nonneg_pos)

```

```

assume x  $\in$  U

```

```

from continuous_on_prod_compactE[OF cont_fx compact_cbox ⟨x  $\in$  U⟩ ⟨0 < e⟩]

```

```

obtain X0 where X0: x  $\in$  X0 open X0

```

```

and fx_bound:  $\bigwedge y t. y \in X0 \cap U \implies t \in \text{cbox } a \text{ } b \implies \text{norm } (f y t - f x t) \leq e$ 

```

```

unfolding split_beta fst_conv snd_conv dist_norm

```

```

by metis

```

```

have  $\forall_F y$  in at x within U. y  $\in$  X0  $\cap$  U

```

```

using X0(1) X0(2) eventually_at_topological by auto

```

```

then show  $\forall_F y$  in at x within U. dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) < e'

```

```

proof eventually_elim
  case (elim y)
  have dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) =
    norm (integral (cbox a b) ( $\lambda t. f y t - f x t$ ))
  using elim  $\langle x \in U \rangle$ 
  unfolding dist_norm
  by (subst integral_diff)
    (auto intro!: integrable_continuous continuous_intros)
  also have ...  $\leq e * \text{content } (cbox a b)$ 
  using elim  $\langle x \in U \rangle$ 
  by (intro integrable_bound)
    (auto intro!: fx_bound  $\langle x \in U \rangle$  less_imp_le[OF  $\langle 0 < e \rangle$ ]
      integrable_continuous continuous_intros)
  also have ...  $< e'$ 
  using  $\langle 0 < e' \rangle \langle e > 0 \rangle$ 
  by (auto simp: e_def field_split_simps)
  finally show dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x))  $< e'$ .
qed
qed
qed (auto intro!: continuous_on_const)

```

lemma leibniz_rule:

```

fixes f::'a::banach  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::banach
assumes fx:  $\bigwedge x t. x \in U \implies t \in cbox a b \implies$ 
  ( $\lambda x. f x t$ ) has_derivative blinfun_apply (fx x t) (at x within U)
assumes integrable_f2:  $\bigwedge x. x \in U \implies f x$  integrable_on cbox a b
assumes cont_fx: continuous_on (U  $\times$  (cbox a b)) ( $\lambda(x, t). f x t$ )
assumes [intro]:  $x0 \in U$ 
assumes convex U
shows
  ( $\lambda x. \text{integral } (cbox a b) (f x)$ ) has_derivative integral (cbox a b) (fx x0) (at x0
  within U)
  (is (?F has_derivative ?dF) _)
proof cases
  assume content (cbox a b)  $\neq 0$ 
  then have ne: cbox a b  $\neq \{\}$  by auto
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f= $\lambda y. \text{Pair } x y$  for x,
      unfolded split_beta fst_conv snd_conv]
  show ?thesis
proof (intro has_derivativeI bounded_linear_scaleR_left tendstoI, fold norm_conv_dist)
  have cont_f1:  $\bigwedge t. t \in cbox a b \implies$  continuous_on U ( $\lambda x. f x t$ )
  by (auto simp: continuous_on_eq continuous_within intro!: has_derivative_continuous
    fx)
  note [continuous_intros] = continuous_on_compose2[OF cont_f1]
  fix e'::real
  assume e'  $> 0$ 
  define e where e = e' / (content (cbox a b) + 1)
  have e  $> 0$  using  $\langle e' > 0 \rangle$  by (auto simp: e_def intro!: divide_pos_pos)

```

```

add_nonneg_pos)
  from continuous_on_prod_compactE[OF cont_fx compact_cbox ‹x0 ∈ U› ‹e
> 0›]
  obtain X0 where X0: x0 ∈ X0 open X0
  and fx_bound:  $\bigwedge x t. x \in X0 \cap U \implies t \in \text{cbox } a \ b \implies \text{norm } (fx \ x \ t - fx \ x0 \ t) \leq e$ 
  unfolding split_beta fst_conv snd_conv
  by (metis dist_norm)

note eventually_closed_segment[OF ‹open X0› ‹x0 ∈ X0›, of U]
moreover
have  $\forall_F x$  in at x0 within U.  $x \in X0$ 
  using ‹open X0› ‹x0 ∈ X0› eventually_at_topological by blast
moreover have  $\forall_F x$  in at x0 within U.  $x \neq x0$ 
  by (auto simp: eventually_at_filter)
moreover have  $\forall_F x$  in at x0 within U.  $x \in U$ 
  by (auto simp: eventually_at_filter)
ultimately
show  $\forall_F x$  in at x0 within U.  $\text{norm } ((?F \ x - ?F \ x0 - ?dF \ (x - x0)) /_R \ \text{norm } (x - x0)) < e'$ 
proof eventually_elim
  case (elim x)
  from elim have  $0 < \text{norm } (x - x0)$  by simp
  have closed_segment x0 x  $\subseteq U$ 
  by (simp add: assms closed_segment_subset elim(4))
  from elim have [intro]:  $x \in U$  by auto
  have  $?F \ x - ?F \ x0 - ?dF \ (x - x0) =$ 
    integral (cbox a b) ( $\lambda y. f \ x \ y - f \ x0 \ y - fx \ x0 \ y \ (x - x0)$ )
    (is _ = ?id)
  using ‹x ≠ x0›
  by (subst blinfun_apply_integral integral_diff,
    auto intro!: integrable_diff integrable_f2 continuous_intros
    intro: integrable_continuous)+
also
{
  fix t assume t:  $t \in (\text{cbox } a \ b)$ 
  then have deriv:
    (( $\lambda x. f \ x \ t$ ) has_derivative (fx y t)) (at y within X0 ∩ U)
  if  $y \in X0 \cap U$  for y
  using fx_has_derivative_subset that by fastforce
  have seg:  $\bigwedge t. t \in \{0..1\} \implies x0 + t *_R (x - x0) \in X0 \cap U$ 
  using ‹closed_segment x0 x  $\subseteq U$ ›
    ‹closed_segment x0 x  $\subseteq X0$ ›
  by (force simp: closed_segment_def algebra_simps)
  have  $\bigwedge x. x \in X0 \cap U \implies \text{onorm } (\text{blinfun\_apply } (fx \ x \ t) - (fx \ x0 \ t)) \leq e$ 
  using fx_bound t
  by (auto simp add: norm_blinfun_def fun_diff_def blinfun.bilinear_simps[symmetric])
  from differentiable_bound_linearization[OF seg deriv this] X0
  have  $\text{norm } (f \ x \ t - f \ x0 \ t - fx \ x0 \ t \ (x - x0)) \leq e * \text{norm } (x - x0)$ 

```

```

    by (auto simp add: ac_simps)
  }
  then have norm ?id ≤ integral (cbox a b) (λ_. e * norm (x - x0))
    by (intro integral_norm_bound_integral)
      (auto intro!: continuous_intros integrable_diff integrable_f2
        intro: integrable_continuous)
  also have ... = content (cbox a b) * e * norm (x - x0)
    by simp
  also have ... < e' * norm (x - x0)
  proof (intro mult_strict_right_mono[OF _ <0 < norm (x - x0)])
    show content (cbox a b) * e < e'
      using <e' > 0 by (simp add: divide_simps e_def not_less)
  qed
  finally have norm (?F x - ?F x0 - ?dF (x - x0)) < e' * norm (x - x0) .
  then show ?case
    by (auto simp: divide_simps)
  qed
qed (rule blinfun.bounded_linear_right)
qed (auto intro!: derivative_eq_intros simp: blinfun.bilinear_simps)

lemma has_vector_derivative_eq_has_derivative_blinfun:
  (f has_vector_derivative f') (at x within U) ↔
  (f has_derivative blinfun_scaleR_left f') (at x within U)
  by (simp add: has_vector_derivative_def)

lemma leibniz_rule_vector_derivative:
  fixes f::real ⇒ 'b::euclidean_space ⇒ 'c::banach
  assumes fx: ∧x t. x ∈ U ⇒ t ∈ cbox a b ⇒
    ((λx. f x t) has_vector_derivative (fx x t)) (at x within U)
  assumes integrable_f2: ∧x. x ∈ U ⇒ (f x) integrable_on cbox a b
  assumes cont_fx: continuous_on (U × cbox a b) (λ(x, t). fx x t)
  assumes U: x0 ∈ U convex U
  shows ((λx. integral (cbox a b) (f x)) has_vector_derivative integral (cbox a b)
    (fx x0))
    (at x0 within U)
  proof -
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f=λy. Pair x y for x,
      unfolded split_beta fst_conv snd_conv]
  show ?thesis
    unfolding has_vector_derivative_eq_has_derivative_blinfun
  proof (rule has_derivative_eq_rhs [OF leibniz_rule[OF integrable_f2 _ U]])
    show continuous_on (U × cbox a b) (λ(x, t). blinfun_scaleR_left (fx x t))
      using cont_fx by (auto simp: split_beta intro!: continuous_intros)
    show blinfun_apply (integral (cbox a b) (λt. blinfun_scaleR_left (fx x0 t))) =
      blinfun_apply (blinfun_scaleR_left (integral (cbox a b) (fx x0)))
    by (subst integral_linear[symmetric])
      (auto simp: has_vector_derivative_def o_def
        intro!: integrable_continuous U continuous_intros bounded_linear_intros)
  qed
  qed

```

qed (use fx in ⟨auto simp: has_vector_derivative_def⟩)
qed

lemma *has_field_derivative_eq_has_derivative_blinfun*:
 (f has_field_derivative f') (at x within U) \longleftrightarrow (f has_derivative blinfun_mult_right f') (at x within U)
by (simp add: has_field_derivative_def)

lemma *leibniz_rule_field_derivative*:
fixes f::'a::{real_normed_field, banach} \Rightarrow 'b::euclidean_space \Rightarrow 'a
assumes fx: $\bigwedge x t. x \in U \implies t \in \text{cbox } a \text{ } b \implies ((\lambda x. f x t) \text{ has_field_derivative } fx x t)$ (at x within U)
assumes integrable_f2: $\bigwedge x. x \in U \implies (f x) \text{ integrable_on } \text{cbox } a \text{ } b$
assumes cont_fx: continuous_on (U \times (cbox a b)) ($\lambda(x, t). fx x t$)
assumes U: $x0 \in U$ convex U
shows ($\lambda x. \text{integral } (\text{cbox } a \text{ } b) (f x)$) has_field_derivative integral (cbox a b) (fx x0)) (at x0 within U)

proof –
note [continuous_intros] =
 continuous_on_compose2[OF cont_fx, **where** f= $\lambda y. \text{Pair } x \text{ } y$ **for** x,
 unfolded split_beta fst_conv snd_conv]
have *: blinfun_mult_right (integral (cbox a b) (fx x0)) =
 integral (cbox a b) ($\lambda t. \text{blinfun_mult_right } (fx \text{ } x0 \text{ } t)$)
by (subst integral_linear[symmetric])
 (auto simp: has_vector_derivative_def o_def
 intro!: integrable_continuous U continuous_intros bounded_linear_intros)
show ?thesis
unfolding has_field_derivative_eq_has_derivative_blinfun
proof (rule has_derivative_eq_rhs [OF leibniz_rule[OF integrable_f2 U,
where fx= $\lambda x t. \text{blinfun_mult_right } (fx \text{ } x \text{ } t)$])
show continuous_on (U \times cbox a b) ($\lambda(x, t). \text{blinfun_mult_right } (fx \text{ } x \text{ } t)$)
using cont_fx **by** (auto simp: split_beta intro!: continuous_intros)
show blinfun_apply (integral (cbox a b) ($\lambda t. \text{blinfun_mult_right } (fx \text{ } x0 \text{ } t)$)) =
 blinfun_apply (blinfun_mult_right (integral (cbox a b) (fx x0)))
by (subst integral_linear[symmetric])
 (auto simp: has_vector_derivative_def o_def
 intro!: integrable_continuous U continuous_intros bounded_linear_intros)
qed (use fx in ⟨auto simp: has_field_derivative_def⟩)
qed

lemma *leibniz_rule_field_differentiable*:
fixes f::'a::{real_normed_field, banach} \Rightarrow 'b::euclidean_space \Rightarrow 'a
assumes $\bigwedge x t. x \in U \implies t \in \text{cbox } a \text{ } b \implies ((\lambda x. f x t) \text{ has_field_derivative } fx \text{ } x \text{ } t)$ (at x within U)
assumes $\bigwedge x. x \in U \implies (f x) \text{ integrable_on } \text{cbox } a \text{ } b$
assumes continuous_on (U \times (cbox a b)) ($\lambda(x, t). fx \text{ } x \text{ } t$)
assumes $x0 \in U$ convex U
shows ($\lambda x. \text{integral } (\text{cbox } a \text{ } b) (f x)$) field_differentiable at x0 within U
using leibniz_rule_field_derivative[OF assms] **by** (auto simp: field_differentiable_def)

7.14.43 Exchange uniform limit and integral

```

lemma uniform_limit_integral_cbox:
  fixes  $f::'a \Rightarrow 'b::\text{euclidean\_space} \Rightarrow 'c::\text{banach}$ 
  assumes  $u$ : uniform_limit (cbox a b) f g F
  assumes  $c$ :  $\bigwedge n$ . continuous_on (cbox a b) (f n)
  assumes [simp]:  $F \neq \text{bot}$ 
  obtains  $I J$  where
     $\bigwedge n$ . (f n has_integral I n) (cbox a b)
    (g has_integral J) (cbox a b)
    (I  $\longrightarrow$  J) F
proof -
  have  $fi$ [simp]: f n integrable_on (cbox a b) for n
    by (auto intro!: integrable_continuous assms)
  then obtain I where  $I$ :  $\bigwedge n$ . (f n has_integral I n) (cbox a b)
    unfolding integrable_on_def by metis

  moreover
  have  $gi$ [simp]: g integrable_on (cbox a b)
    by (auto intro!: integrable_continuous uniform_limit_theorem[OF _ u] eventuallyI c)
  then obtain J where  $J$ : (g has_integral J) (cbox a b)
    by blast

  moreover
  have (I  $\longrightarrow$  J) F
proof cases
  assume content (cbox a b) = 0
  hence  $I = (\lambda \_ . 0)$   $J = 0$ 
    by (auto intro!: has_integral_unique I J)
  thus ?thesis by simp
next
  assume content_nonzero: content (cbox a b)  $\neq 0$ 
  show ?thesis
proof (rule tendstoI)
  fix  $e::\text{real}$ 
  assume  $e > 0$ 
  define  $e'$  where  $e' = e/2$ 
  with  $\langle e > 0 \rangle$  have  $e' > 0$  by simp
  then have  $\forall_F n$  in F.  $\forall x \in \text{cbox } a \text{ b. } \text{norm } (f \ n \ x - g \ x) < e' / \text{content } (\text{cbox } a \text{ b})$ 
    using  $u$  content_nonzero by (auto simp: uniform_limit_iff dist_norm zero_less_measure_iff)
  then show  $\forall_F n$  in F.  $\text{dist } (I \ n) \ J < e$ 
proof eventually_elim
  case (elim n)
  have  $I \ n = \text{integral } (\text{cbox } a \text{ b}) \ (f \ n)$ 
     $J = \text{integral } (\text{cbox } a \text{ b}) \ g$ 
    using  $I$ [of n]  $J$  by (simp_all add: integral_unique)
  then have  $\text{dist } (I \ n) \ J = \text{norm } (\text{integral } (\text{cbox } a \text{ b}) \ (\lambda x. f \ n \ x - g \ x))$ 

```



```

    by (simp add: integral_diff_dist_norm)
  also have ... ≤ integral (cbox a b) (λx. (e' / content (cbox a b)))
    using elim
    by (intro integral_norm_bound_integral) (auto intro!: integrable_diff)
  also have ... < e
    using ⟨0 < e⟩ by (simp add: e'_def)
  finally show ?case .
qed
qed
qed
ultimately show ?thesis ..
qed

```

```

lemma uniform_limit_integral:
  fixes f::'a ⇒ 'b::ordered_euclidean_space ⇒ 'c::banach
  assumes u: uniform_limit {a..b} f g F
  assumes c: ∧n. continuous_on {a..b} (f n)
  assumes [simp]: F ≠ bot
  obtains I J where
    ∧n. (f n has_integral I n) {a..b}
    (g has_integral J) {a..b}
    (I ⟶ J) F
  by (metis interval_cbox assms uniform_limit_integral_cbox)

```

7.14.44 Integration by parts

```

lemma integration_by_parts_interior_strong:
  fixes prod :: _ ⇒ _ ⇒ 'b :: banach
  assumes bilinear: bounded_bilinear (prod)
  assumes s: finite s and le: a ≤ b
  assumes cont [continuous_intros]: continuous_on {a..b} f continuous_on {a..b}
  g
  assumes deriv: ∧x. x∈{a<..} - s ⟹ (f has_vector_derivative f' x) (at x)
    ∧x. x∈{a<..} - s ⟹ (g has_vector_derivative g' x) (at x)
  assumes int: ((λx. prod (f x) (g' x)) has_integral
    (prod (f b) (g b) - prod (f a) (g a) - y)) {a..b}
  shows ((λx. prod (f' x) (g x)) has_integral y) {a..b}
proof -
  interpret bounded_bilinear prod by fact
  have ((λx. prod (f x) (g' x) + prod (f' x) (g x)) has_integral
    (prod (f b) (g b) - prod (f a) (g a))) {a..b}
    using deriv by (intro fundamental_theorem_of_calculus_interior_strong[OF
  s le])
    (auto intro!: continuous_intros continuous_on has_vector_derivative)
  from has_integral_diff[OF this int] show ?thesis by (simp add: algebra_simps)
qed

```

```

lemma integration_by_parts_interior:
  fixes prod :: _ ⇒ _ ⇒ 'b :: banach

```

assumes *bounded_bilinear* (prod) $a \leq b$
continuous_on {a..b} *f* *continuous_on* {a..b} *g*
assumes $\bigwedge x. x \in \{a < .. < b\} \implies (f \text{ has_vector_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a < .. < b\} \implies (g \text{ has_vector_derivative } g' x) (at x)$
assumes $((\lambda x. \text{prod } (f x) (g' x)) \text{ has_integral } (\text{prod } (f b) (g b) - \text{prod } (f a) (g a) - y)) \{a..b\}$
shows $((\lambda x. \text{prod } (f' x) (g x)) \text{ has_integral } y) \{a..b\}$
by (*rule integration_by_parts_interior_strong*[of _ {} _ _ *f g f' g'*]) (*use assms in simp_all*)

lemma *integration_by_parts*:

fixes *prod* :: $_ \Rightarrow _ \Rightarrow 'b :: \text{banach}$
assumes *bounded_bilinear* (prod) $a \leq b$
continuous_on {a..b} *f* *continuous_on* {a..b} *g*
assumes $\bigwedge x. x \in \{a..b\} \implies (f \text{ has_vector_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a..b\} \implies (g \text{ has_vector_derivative } g' x) (at x)$
assumes $((\lambda x. \text{prod } (f x) (g' x)) \text{ has_integral } (\text{prod } (f b) (g b) - \text{prod } (f a) (g a) - y)) \{a..b\}$
shows $((\lambda x. \text{prod } (f' x) (g x)) \text{ has_integral } y) \{a..b\}$
by (*rule integration_by_parts_interior*[of _ _ _ *f g f' g'*]) (*use assms in simp_all*)

lemma *integrable_by_parts_interior_strong*:

fixes *prod* :: $_ \Rightarrow _ \Rightarrow 'b :: \text{banach}$
assumes *bilinear*: *bounded_bilinear* (prod)
assumes *s*: *finite s* **and** *le*: $a \leq b$
assumes *cont* [*continuous_intros*]: *continuous_on* {a..b} *f* *continuous_on* {a..b} *g*
assumes *deriv*: $\bigwedge x. x \in \{a < .. < b\} - s \implies (f \text{ has_vector_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a < .. < b\} - s \implies (g \text{ has_vector_derivative } g' x) (at x)$
assumes *int*: $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable_on } \{a..b\}$
shows $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable_on } \{a..b\}$

proof -

from *int* **obtain** *I* **where** $((\lambda x. \text{prod } (f x) (g' x)) \text{ has_integral } I) \{a..b\}$
unfolding *integrable_on_def* **by** *blast*
hence $((\lambda x. \text{prod } (f x) (g' x)) \text{ has_integral } (\text{prod } (f b) (g b) - \text{prod } (f a) (g a) - (\text{prod } (f b) (g b) - \text{prod } (f a) (g a) - I))) \{a..b\}$ **by** *simp*
from *integration_by_parts_interior_strong*[OF *assms*(1-7) *this*]
show *?thesis* **unfolding** *integrable_on_def* **by** *blast*

qed

lemma *integrable_by_parts_interior*:

fixes *prod* :: $_ \Rightarrow _ \Rightarrow 'b :: \text{banach}$
assumes *bounded_bilinear* (prod) $a \leq b$
continuous_on {a..b} *f* *continuous_on* {a..b} *g*
assumes $\bigwedge x. x \in \{a < .. < b\} \implies (f \text{ has_vector_derivative } f' x) (at x)$
 $\bigwedge x. x \in \{a < .. < b\} \implies (g \text{ has_vector_derivative } g' x) (at x)$
assumes $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable_on } \{a..b\}$
shows $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable_on } \{a..b\}$

by (rule integrable_by_parts_interior_strong[of _ { } _ _ f g f' g']) (use assms in simp_all)

lemma integrable_by_parts:

fixes prod :: $_ \Rightarrow _ \Rightarrow _$ 'b :: banach
 assumes bounded_bilinear (prod) $a \leq b$
 continuous_on {a..b} f continuous_on {a..b} g
 assumes $\bigwedge x. x \in \{a..b\} \implies (f \text{ has_vector_derivative } f' x) \text{ (at } x)$
 $\bigwedge x. x \in \{a..b\} \implies (g \text{ has_vector_derivative } g' x) \text{ (at } x)$
 assumes $(\lambda x. \text{prod } (f x) (g' x)) \text{ integrable_on } \{a..b\}$
 shows $(\lambda x. \text{prod } (f' x) (g x)) \text{ integrable_on } \{a..b\}$
 by (rule integrable_by_parts_interior_strong[of _ { } _ _ f g f' g']) (use assms in simp_all)

7.14.45 Integration by substitution

lemma has_integral_substitution_general:

fixes f :: real \Rightarrow 'a::euclidean_space and g :: real \Rightarrow real
 assumes s: finite s and le: $a \leq b$
 and subset: $g \text{ ' } \{a..b\} \subseteq \{c..d\}$
 and f [continuous_intros]: continuous_on {c..d} f
 and g [continuous_intros]: continuous_on {a..b} g
 and deriv [derivative_intros]:
 $\bigwedge x. x \in \{a..b\} - s \implies (g \text{ has_field_derivative } g' x) \text{ (at } x \text{ within } \{a..b\})$
 shows $((\lambda x. g' x *_R f (g x)) \text{ has_integral } (\text{integral } \{g a..g b\} f - \text{integral } \{g b..g a\} f)) \{a..b\}$

proof -

let ?F = $\lambda x. \text{integral } \{c..g x\} f$
 have cont_int: continuous_on {a..b} ?F
 by (rule continuous_on_compose2[OF _ g subset] indefinite_integral_continuous_1 f integrable_continuous_real)+
 have deriv: $((\lambda x. \text{integral } \{c..x\} f) \circ g) \text{ has_vector_derivative } g' x *_R f (g x)$
 (at x within {a..b}) if $x \in \{a..b\} - s$ for x
 proof (rule has_vector_derivative_eq_rhs [OF vector_diff_chain_within refl])
 show $(g \text{ has_vector_derivative } g' x) \text{ (at } x \text{ within } \{a..b\})$
 using deriv has_real_derivative_iff_has_vector_derivative that by blast
 show $((\lambda x. \text{integral } \{c..x\} f) \text{ has_vector_derivative } f (g x))$
 (at (g x) within g ' {a..b})
 using that le subset
 by (blast intro: has_vector_derivative_within_subset integral_has_vector_derivative f)

qed

have deriv: $(?F \text{ has_vector_derivative } g' x *_R f (g x))$
 (at x) if $x \in \{a..b\} - (s \cup \{a,b\})$ for x
 using deriv[of x] that by (simp add: at_within_Icc_at_o_def)
 have $((\lambda x. g' x *_R f (g x)) \text{ has_integral } (?F b - ?F a)) \{a..b\}$
 using le cont_int s deriv cont_int
 by (intro fundamental_theorem_of_calculus_interior_strong[of s \cup {a,b}])
 simp_all

also
from *subset* **have** $g \ x \in \{c..d\}$ **if** $x \in \{a..b\}$ **for** x **using** *that* **by** *blast*
from *this[of a]* *this[of b]* **le** **have** $cd: c \leq g \ a \ g \ b \leq d \ c \leq g \ b \ g \ a \leq d$ **by** *auto*
have $\text{integral } \{c..g \ b\} \ f - \text{integral } \{c..g \ a\} \ f = \text{integral } \{g \ a..g \ b\} \ f - \text{integral } \{g \ b..g \ a\} \ f$
proof *cases*
assume $g \ a \leq g \ b$
note $le = le \ \text{this}$
from *cd* **have** $\text{integral } \{c..g \ a\} \ f + \text{integral } \{g \ a..g \ b\} \ f = \text{integral } \{c..g \ b\} \ f$
by (*intro integral_combine integrable_continuous_real continuous_on_subset[OF f] le simp_all*)
with *le* **show** *?thesis*
by (*cases g a = g b (simp_all add: algebra_simps)*)
next
assume *less: $\neg g \ a \leq g \ b$*
then **have** $g \ a \geq g \ b$ **by** *simp*
note $le = le \ \text{this}$
from *cd* **have** $\text{integral } \{c..g \ b\} \ f + \text{integral } \{g \ b..g \ a\} \ f = \text{integral } \{c..g \ a\} \ f$
by (*intro integral_combine integrable_continuous_real continuous_on_subset[OF f] le simp_all*)
with *less* **show** *?thesis*
by (*simp_all add: algebra_simps*)
qed
finally **show** *?thesis* .
qed

lemma *has_integral_substitution_strong*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$ **and** $g :: \text{real} \Rightarrow \text{real}$
assumes *s: finite s* **and** *le: $a \leq b \ g \ a \leq g \ b$*
and *subset: $g \ ' \{a..b\} \subseteq \{c..d\}$*
and f [*continuous_intros*]: *continuous_on $\{c..d\}$ f*
and g [*continuous_intros*]: *continuous_on $\{a..b\}$ g*
and *deriv [derivative_intros]:*
 $\bigwedge x. x \in \{a..b\} - s \implies (g \ \text{has_field_derivative } g' \ x) \ (\text{at } x \ \text{within } \{a..b\})$
shows $((\lambda x. g' \ x \ *_{\mathbb{R}} \ f \ (g \ x)) \ \text{has_integral} \ (\text{integral } \{g \ a..g \ b\} \ f)) \ \{a..b\}$
using *has_integral_substitution_general[OF s le(1) subset f g deriv] le(2)*
by (*cases g a = g b*) *auto*

lemma *has_integral_substitution*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$ **and** $g :: \text{real} \Rightarrow \text{real}$
assumes $a \leq b \ g \ a \leq g \ b \ g \ ' \{a..b\} \subseteq \{c..d\}$
and *continuous_on $\{c..d\}$ f*
and $\bigwedge x. x \in \{a..b\} \implies (g \ \text{has_field_derivative } g' \ x) \ (\text{at } x \ \text{within } \{a..b\})$
shows $((\lambda x. g' \ x \ *_{\mathbb{R}} \ f \ (g \ x)) \ \text{has_integral} \ (\text{integral } \{g \ a..g \ b\} \ f)) \ \{a..b\}$
by (*intro has_integral_substitution_strong[of $\{ \}$ a b g c d] assms*)
(auto intro: DERIV_continuous_on assms)

lemma *integral_shift*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$

```

  assumes cont: continuous_on {a + c..b + c} f
  shows integral {a..b} (f ∘ (λx. x + c)) = integral {a + c..b + c} f
proof (cases a ≤ b)
  case True
  have ((λx. 1 *R f (x + c)) has_integral integral {a+c..b+c} f) {a..b}
  using True cont
  by (intro has_integral_substitution[where c = a + c and d = b + c])
    (auto intro!: derivative_eq_intros)
  thus ?thesis by (simp add: has_integral_iff o_def)
qed auto

```

7.14.46 Compute a double integral using iterated integrals and switching the order of integration

```

lemma continuous_on_imp_integrable_on_Pair1:
  fixes f :: _ ⇒ 'b::banach
  assumes con: continuous_on (cbox (a,c) (b,d)) f and x: x ∈ cbox a b
  shows (λy. f (x, y)) integrable_on (cbox c d)
proof -
  have f ∘ (λy. (x, y)) integrable_on (cbox c d)
  proof (intro integrable_continuous continuous_on_compose [OF _ continuous_on_subset [OF con]])
    show continuous_on (cbox c d) (Pair x)
    by (simp add: continuous_on_Pair)
    show Pair x ' cbox c d ⊆ cbox (a,c) (b,d)
    using x by blast
  qed
  then show ?thesis
  by (simp add: o_def)
qed

```

```

lemma integral_integrable_2dim:
  fixes f :: ('a::euclidean_space * 'b::euclidean_space) ⇒ 'c::banach
  assumes continuous_on (cbox (a,c) (b,d)) f
  shows (λx. integral (cbox c d) (λy. f (x,y))) integrable_on cbox a b
proof (cases content(cbox c d) = 0)
  case True
  then show ?thesis
  by (simp add: True integrable_const)
next
  case False
  have uc: uniformly_continuous_on (cbox (a,c) (b,d)) f
  by (simp add: assms compact_cbox compact_uniformly_continuous)
  { fix x::'a and e::real
    assume x: x ∈ cbox a b and e: 0 < e
    then have e2_gt: 0 < e/2 / content (cbox c d) and e2_less: e/2 / content
(cbox c d) * content (cbox c d) < e
    by (auto simp: False content_lt_nz e)
    then obtain dd

```

```

where dd:  $\bigwedge x x'. \llbracket x \in \text{cbox } (a, c) (b, d); x' \in \text{cbox } (a, c) (b, d); \text{norm } (x' - x) < dd \rrbracket$ 
       $\implies \text{norm } (f x' - f x) \leq e / (2 * \text{content } (\text{cbox } c d)) \quad dd > 0$ 
using uc [unfolded_uniformly_continuous_on_def, THEN spec, of e/(2 * content (cbox c d))]
by (auto simp: dist_norm intro: less_imp_le)
have  $\exists \text{delta} > 0. \forall x' \in \text{cbox } a b. \text{norm } (x' - x) < \text{delta} \longrightarrow \text{norm } (\text{integral } (\text{cbox } c d) (\lambda u. f (x', u) - f (x, u))) < e$ 
using dd e2_gt assms x
apply (rule_tac x=dd in exI)
apply clarify
apply (rule le_less_trans [OF integrable_bound e2_less])
apply (auto intro: integrable_diff continuous_on_imp_integrable_on_Pair1)
done
} note * = this
show ?thesis
proof (rule integrable_continuous)
show continuous_on (cbox a b) ( $\lambda x. \text{integral } (\text{cbox } c d) (\lambda y. f (x, y))$ )
by (simp add: * continuous_on_iff_dist_norm_integrable_diff [symmetric]
continuous_on_imp_integrable_on_Pair1 [OF assms])
qed
qed

```

lemma *integral_split*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::{real_normed_vector, complete_space}
assumes f: f integrable_on (cbox a b)
and k: k  $\in$  Basis
shows  $\text{integral } (\text{cbox } a b) f =$ 
       $\text{integral } (\text{cbox } a b \cap \{x. x \cdot k \leq c\}) f +$ 
       $\text{integral } (\text{cbox } a b \cap \{x. x \cdot k \geq c\}) f$ 
using k f
by (auto simp: has_integral_integral intro: integral_unique [OF has_integral_split])

```

lemma *integral_swap_operativeI*:

```

fixes f :: ('a::euclidean_space * 'b::euclidean_space)  $\Rightarrow$  'c::banach
assumes f: continuous_on s f and e:  $0 < e$ 
shows operative conj True
      ( $\lambda k. \forall a b c d. \text{cbox } (a, c) (b, d) \subseteq k \wedge \text{cbox } (a, c) (b, d) \subseteq s$ 
       $\longrightarrow \text{norm}(\text{integral } (\text{cbox } (a, c) (b, d)) f -$ 
       $\text{integral } (\text{cbox } a b) (\lambda x. \text{integral } (\text{cbox } c d) (\lambda y. f((x, y))))$ 
       $\leq e * \text{content } (\text{cbox } (a, c) (b, d)))$ )

```

proof (standard, auto)

```

fix a::'a and c::'b and b::'a and d::'b and u::'a and v::'a and w::'b and z::'b
assume *:  $\text{box } (a, c) (b, d) = \{\}$ 
and cb1:  $\text{cbox } (u, w) (v, z) \subseteq \text{cbox } (a, c) (b, d)$ 
and cb2:  $\text{cbox } (u, w) (v, z) \subseteq s$ 
then have c0:  $\text{content } (\text{cbox } (a, c) (b, d)) = 0$ 
using * unfolding content_eq_0_interior by simp

```

```

have c0': content (cbox (u, w) (v, z)) = 0
  by (fact content_0_subset [OF c0 cb1])
show norm (integral (cbox (u,w) (v,z)) f - integral (cbox u v) (λx. integral (cbox
w z) (λy. f (x, y))))
  ≤ e * content (cbox (u,w) (v,z))
  using content_cbox_pair_eq0_D [OF c0']
  by (force simp add: c0')
next
fix a::'a and c::'b and b::'a and d::'b
and M::real and i::'a and j::'b
and u::'a and v::'a and w::'b and z::'b
assume ij: (i,j) ∈ Basis
  and n1: ∀ a' b' c' d'.
    cbox (a',c') (b',d') ⊆ cbox (a,c) (b,d) ∧
    cbox (a',c') (b',d') ⊆ {x. x · (i,j) ≤ M} ∧ cbox (a',c') (b',d') ⊆ s →
    norm (integral (cbox (a',c') (b',d')) f - integral (cbox a' b') (λx.
integral (cbox c' d') (λy. f (x,y))))
    ≤ e * content (cbox (a',c') (b',d'))
  and n2: ∀ a' b' c' d'.
    cbox (a',c') (b',d') ⊆ cbox (a,c) (b,d) ∧
    cbox (a',c') (b',d') ⊆ {x. M ≤ x · (i,j)} ∧ cbox (a',c') (b',d') ⊆ s →
    norm (integral (cbox (a',c') (b',d')) f - integral (cbox a' b') (λx.
integral (cbox c' d') (λy. f (x,y))))
    ≤ e * content (cbox (a',c') (b',d'))
  and subs: cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d) cbox (u,w) (v,z) ⊆ s
have fcont: continuous_on (cbox (u, w) (v, z)) f
  using asms(1) continuous_on_subset subs(2) by blast
then have fint: f integrable_on cbox (u, w) (v, z)
  by (metis integrable_continuous)
consider i ∈ Basis j=0 | j ∈ Basis i=0 using ij
  by (auto simp: Euclidean_Space.Basis_prod_def)
then show norm (integral (cbox (u,w) (v,z)) f - integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
  ≤ e * content (cbox (u,w) (v,z)) (is ?normle)
proof cases
case 1
  then have e: e * content (cbox (u, w) (v, z)) =
    e * (content (cbox u v ∩ {x. x · i ≤ M}) * content (cbox w z)) +
    e * (content (cbox u v ∩ {x. M ≤ x · i}) * content (cbox w z))
  by (simp add: content_split [where c=M] content_Pair algebra_simps)
  have *: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =
    integral (cbox u v ∩ {x. x · i ≤ M}) (λx. integral (cbox w z) (λy. f
(x, y))) +
    integral (cbox u v ∩ {x. M ≤ x · i}) (λx. integral (cbox w z) (λy. f
(x, y)))
  using 1 f subs integral_integrable_2dim continuous_on_subset
  by (blast intro: integral_split)
show ?normle
  apply (rule norm_diff2 [OF integral_split [where c=M, OF fint ij] * e])

```

```

using 1 subs
apply (simp_all add: cbox_Pair_eq setcomp_dot1 [of  $\lambda u. M \leq u$ ] setcomp_dot1
[of  $\lambda u. u \leq M$ ] Sigma_Int_Paircomp1)
  apply (simp_all add: interval_split ij_flip: cbox_Pair_eq content_Pair)
  apply (force simp flip: interval_split intro!: n1 [rule_format])
  apply (force simp flip: interval_split intro!: n2 [rule_format])
done
next
case 2
then have e:  $e * \text{content } (cbox \ (u, w) \ (v, z)) =$ 
   $e * (\text{content } (cbox \ u \ v) * \text{content } (cbox \ w \ z \cap \{x. x \cdot j \leq M\})) +$ 
   $e * (\text{content } (cbox \ u \ v) * \text{content } (cbox \ w \ z \cap \{x. M \leq x \cdot j\}))$ 
  by (simp add: content_split [where  $c=M$ ] content_Pair algebra_simps)
  have ( $\lambda x. \text{integral } (cbox \ w \ z \cap \{x. x \cdot j \leq M\}) \ (\lambda y. f \ (x, y))$ ) integrable_on
cbox u v
  ( $\lambda x. \text{integral } (cbox \ w \ z \cap \{x. M \leq x \cdot j\}) \ (\lambda y. f \ (x, y))$ ) integrable_on cbox
u v
  using 2 subs
  apply (simp_all add: interval_split)
  apply (rule integral_integrable_2dim [OF continuous_on_subset [OF f]]);
auto simp: cbox_Pair_eq interval_split [symmetric] +
done
with 2 have *:  $\text{integral } (cbox \ u \ v) \ (\lambda x. \text{integral } (cbox \ w \ z) \ (\lambda y. f \ (x, y))) =$ 
 $\text{integral } (cbox \ u \ v) \ (\lambda x. \text{integral } (cbox \ w \ z \cap \{x. x \cdot j \leq M\}) \ (\lambda y. f$ 
( $x, y$ ))) +
 $\text{integral } (cbox \ u \ v) \ (\lambda x. \text{integral } (cbox \ w \ z \cap \{x. M \leq x \cdot j\}) \ (\lambda y. f$ 
( $x, y$ )))
  by (simp add: integral_add [symmetric] integral_split [symmetric]
continuous_on_imp_integrable_on_Pair1 [OF fcont] cong:
integral_cong)
show ?normle
  apply (rule norm_diff2 [OF integral_split [where  $c=M$ , OF fint ij] * e])
  using 2 subs
  apply (simp_all add: cbox_Pair_eq setcomp_dot2 [of  $\lambda u. M \leq u$ ] setcomp_dot2
[of  $\lambda u. u \leq M$ ] Sigma_Int_Paircomp2)
  apply (simp_all add: interval_split ij_flip: cbox_Pair_eq content_Pair)
  apply (force simp flip: interval_split intro!: n1 [rule_format])
  apply (force simp flip: interval_split intro!: n2 [rule_format])
done
qed
qed

```

lemma *integral_Pair_const*:

```

integral (cbox (a,c) (b,d)) ( $\lambda x. k$ ) =
integral (cbox a b) ( $\lambda x. \text{integral } (cbox \ c \ d) \ (\lambda y. k)$ )
by (simp add: content_Pair)

```

lemma *integral_prod_continuous*:

```

fixes f :: ('a::euclidean_space * 'b::euclidean_space)  $\Rightarrow$  'c::banach

```



```

assumes continuous_on (cbox (a, c) (b, d)) f (is continuous_on ?CBOX f)
shows integral (cbox (a, c) (b, d)) f = integral (cbox a b) ( $\lambda x.$  integral (cbox c
d) ( $\lambda y.$  f (x, y)))
proof (cases content ?CBOX = 0)
  case True
    then show ?thesis
      by (auto simp: content_Pair)
  next
    case False
      then have cbp: content ?CBOX > 0
        using content_lt_nz by blast
      have norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x.$  integral (cbox c d) ( $\lambda y.$ 
f (x,y)))) = 0
      proof (rule dense_eq0_I, simp)
        fix e :: real
        assume 0 < e
        with ⟨content ?CBOX > 0⟩ have 0 < e / content ?CBOX
          by simp
        have f_int_acbd: f integrable_on ?CBOX
          by (rule integrable_continuous [OF assms])
        { fix p
          assume p: p division_of ?CBOX
          then have finite p
            by blast
          define e' where e' = e / content ?CBOX
          with ⟨0 < e⟩ ⟨0 < e / content ?CBOX⟩
          have 0 < e'
            by simp
          interpret operative conj True
             $\lambda k. \forall a' b' c' d'.
              \text{cbox } (a', c') (b', d') \subseteq k \wedge \text{cbox } (a', c') (b', d') \subseteq ?CBOX
              \longrightarrow \text{norm } (\text{integral } (\text{cbox } (a', c') (b', d')) f -
                \text{integral } (\text{cbox } a' b') (\lambda x. \text{integral } (\text{cbox } c' d') (\lambda y. f ((x, y))))))
                \leq e' * \text{content } (\text{cbox } (a', c') (b', d'))$ 
          using assms ⟨0 < e'⟩ by (rule integral_swap_operativeI)
          have norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x.$  integral (cbox c d)
( $\lambda y.$  f (x, y))))
             $\leq e' * \text{content } ?CBOX$ 
          if  $\bigwedge t u v w z. t \in p \implies \text{cbox } (u, w) (v, z) \subseteq t \implies \text{cbox } (u, w) (v, z) \subseteq
?CBOX$ 
             $\implies \text{norm } (\text{integral } (\text{cbox } (u, w) (v, z)) f -
              \text{integral } (\text{cbox } u v) (\lambda x. \text{integral } (\text{cbox } w z) (\lambda y. f (x, y))))
              \leq e' * \text{content } (\text{cbox } (u, w) (v, z))$ 
          using that division [of p (a, c) (b, d)] p ⟨finite p⟩ by (auto simp add:
comm_monoid_set_F_and)
          with False have norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x.$  integral
(cbox c d) ( $\lambda y.$  f (x, y))))
             $\leq e$ 
          if  $\bigwedge t u v w z. t \in p \implies \text{cbox } (u, w) (v, z) \subseteq t \implies \text{cbox } (u, w) (v, z) \subseteq$ 

```

```

?CBOX
  ⇒ norm (integral (cbox (u, w) (v, z)) f -
            integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))))
    ≤ e * content (cbox (u, w) (v, z)) / content ?CBOX
  using that by (simp add: e'_def)
} note op_acbd = this
{ fix k::real and D and u::'a and v w and z::'b and t1 t2 l
  assume k: 0 < k
    and nf: ∧x y u v.
      [[x ∈ cbox a b; y ∈ cbox c d; u ∈ cbox a b; v ∈ cbox c d; norm (u-x,
v-y) < k]]
      ⇒ norm (f(u,v) - f(x,y)) < e/(2 * (content ?CBOX))
    and p_acbd: D tagged_division_of cbox (a,c) (b,d)
    and fine: (λx. ball x k) fine D ((t1,t2), l) ∈ D
    and uwvz_sub: cbox (u,w) (v,z) ⊆ l cbox (u,w) (v,z) ⊆ cbox (a,c) (b,d)
  have t: t1 ∈ cbox a b t2 ∈ cbox c d
    by (meson fine p_acbd cbox_Pair_iff_tag_in_interval)+
  have ls: l ⊆ ball (t1,t2) k
    using fine by (simp add: fine_def Ball_def)
{ fix x1 x2
  assume xuvwz: x1 ∈ cbox u v x2 ∈ cbox w z
  then have x: x1 ∈ cbox a b x2 ∈ cbox c d
    using uwvz_sub by auto
  have norm (x1 - t1, x2 - t2) = norm (t1 - x1, t2 - x2)
    by (simp add: norm_Pair norm_minus_commute)
  also have norm (t1 - x1, t2 - x2) < k
    using xuvwz ls uwvz_sub unfolding ball_def
    by (force simp add: cbox_Pair_eq dist_norm)
  finally have norm (f (x1,x2) - f (t1,t2)) ≤ e/content ?CBOX/2
    using nf [OF t x] by simp
} note nf' = this
have f_int_uwvz: f integrable_on cbox (u,w) (v,z)
  using f_int_acbd uwvz_sub integrable_on_subcbox by blast
have f_int_uv: ∧x. x ∈ cbox u v ⇒ (λy. f (x,y)) integrable_on cbox w z
  using assms continuous_on_subset uwvz_sub
  by (blast intro: continuous_on_imp_integrable_on_Pair1)
have 1: norm (integral (cbox (u,w) (v,z)) f - integral (cbox (u,w) (v,z)) (λx.
f (t1,t2)))
  ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
  using cbp ⟨0 < e/content ?CBOX⟩ nf'
  apply (simp only: integral_diff [symmetric] f_int_uwvz integrable_const)
  apply (auto simp: integrable_diff f_int_uwvz integrable_const intro: or-
der_trans [OF integrable_bound [of e/content ?CBOX/2]])
done
  have int_integrable: (λx. integral (cbox w z) (λy. f (x, y))) integrable_on
cbox u v
  using assms integral_integrable_2dim continuous_on_subset uwvz_sub(2)
by blast
  have normint_wz:

```

```

     $\bigwedge x. x \in \text{cbox } u \ v \implies$ 
      norm (integral (cbox w z) ( $\lambda y. f(x, y)$ ) - integral (cbox w z) ( $\lambda y. f$ 
(t1, t2)))
         $\leq e * \text{content } (\text{cbox } w \ z) / \text{content } (\text{cbox } (a, c) \ (b, d)) / 2$ 
      using cbp <0 < e/content ?CBOX> nf'
      apply (simp only: integral_diff [symmetric] f_int_uv integrable_const)
      apply (auto simp: integrable_diff f_int_uv integrable_const intro: or-
der_trans [OF integrable_bound [of e/content ?CBOX/2]])
      done
      have norm (integral (cbox u v)
        ( $\lambda x. \text{integral } (\text{cbox } w \ z) \ (\lambda y. f(x, y)) - \text{integral } (\text{cbox } w \ z) \ (\lambda y. f$ 
(t1, t2))))
         $\leq e * \text{content } (\text{cbox } w \ z) / \text{content } ?CBOX / 2 * \text{content } (\text{cbox } u \ v)$ 
      using cbp <0 < e/content ?CBOX>
      apply (intro integrable_bound [OF __ normint_wz])
      apply (auto simp: field_split_simps integrable_diff int_integrable inte-
grable_const)
      done
      also have ...  $\leq e * \text{content } (\text{cbox } (u, w) \ (v, z)) / \text{content } ?CBOX / 2$ 
      by (simp add: content_Pair field_split_simps)
      finally have 2: norm (integral (cbox u v) ( $\lambda x. \text{integral } (\text{cbox } w \ z) \ (\lambda y. f$ 
(x, y))) -
        integral (cbox u v) ( $\lambda x. \text{integral } (\text{cbox } w \ z) \ (\lambda y. f(t1, t2))$ ))
         $\leq e * \text{content } (\text{cbox } (u, w) \ (v, z)) / \text{content } ?CBOX / 2$ 
      by (simp only: integral_diff [symmetric] int_integrable integrable_const)
      have norm_xx:  $\llbracket x' = y'; \text{norm}(x - x') \leq e/2; \text{norm}(y - y') \leq e/2 \rrbracket \implies$ 
norm(x - y)  $\leq e$  for  $x::'c$  and  $y \ x' \ y' \ e$ 
      using norm_triangle_mono [of x-y' e/2 y'-y e/2] field_sum_of_halves
      by (simp add: norm_minus_commute)
      have norm (integral (cbox (u, w) (v, z)) f - integral (cbox u v) ( $\lambda x. \text{integral}$ 
(cbox w z) ( $\lambda y. f(x, y)$ )))
         $\leq e * \text{content } (\text{cbox } (u, w) \ (v, z)) / \text{content } ?CBOX$ 
      by (rule norm_xx [OF integral_Pair_const 1 2])
    } note * = this
    have norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x. \text{integral } (\text{cbox } c \ d)$ 
( $\lambda y. f(x, y)$ )))  $\leq e$ 
      if  $\forall x \in ?CBOX. \forall x' \in ?CBOX. \text{norm}(x' - x) < k \implies \text{norm}(f \ x' - f \ x) < e$ 
/(2 * content (?CBOX)) 0 < k for k
    proof -
      obtain p where ptag: p tagged_division_of cbox (a, c) (b, d)
        and fine: ( $\lambda x. \text{ball } x \ k$ ) fine p
      using fine_division_exists <0 < k> by blast
      show ?thesis
        using that fine ptag <0 < k>
        by (auto simp: * intro: op_acbd [OF division_of_tagged_division [OF ptag]])
    qed
    then show norm (integral ?CBOX f - integral (cbox a b) ( $\lambda x. \text{integral } (\text{cbox}$ 
c d) ( $\lambda y. f(x, y)$ )))  $\leq e$ 
      using compact_uniformly_continuous [OF assms compact_cbox]

```

```

    apply (simp add: uniformly_continuous_on_def dist_norm)
    apply (drule_tac x=e/2 / content?CBOX in spec)
    using cbp ⟨0 < e⟩ by (auto simp: zero_less_mult_iff)
  qed
  then show ?thesis
    by simp
  qed

lemma integral_swap_2dim:
  fixes f :: ['a::euclidean_space, 'b::euclidean_space] ⇒ 'c::banach
  assumes continuous_on (cbox (a,c) (b,d)) (λ(x,y). f x y)
  shows integral (cbox (a, c) (b, d)) (λ(x, y). f x y) = integral (cbox (c, a) (d,
  b)) (λ(x, y). f y x)
  proof -
    have ((λ(x, y). f x y) has_integral integral (cbox (c, a) (d, b)) (λ(x, y). f y x))
      (prod.swap ' (cbox (c, a) (d, b)))
    proof (rule has_integral_twiddle [of 1 prod.swap prod.swap λ(x,y). f y x integral
      (cbox (c, a) (d, b)) (λ(x, y). f y x), simplified])
      show ∧u v. content (prod.swap ' cbox u v) = content (cbox u v)
        by (metis content_Pair mult.commute old.prod.exhaust swap_cbox_Pair)
      show ((λ(x, y). f y x) has_integral integral (cbox (c, a) (d, b)) (λ(x, y). f y x))
        (cbox (c, a) (d, b))
        by (simp add: assms integrable_continuous integrable_integral swap_continuous)
    qed (use isCont_swap in ⟨fastforce+⟩)
  then show ?thesis
    by force
  qed

```

```

theorem integral_swap_continuous:
  fixes f :: ['a::euclidean_space, 'b::euclidean_space] ⇒ 'c::banach
  assumes continuous_on (cbox (a,c) (b,d)) (λ(x,y). f x y)
  shows integral (cbox a b) (λx. integral (cbox c d) (f x)) =
    integral (cbox c d) (λy. integral (cbox a b) (λx. f x y))
  proof -
    have integral (cbox a b) (λx. integral (cbox c d) (f x)) = integral (cbox (a, c) (b,
    d)) (λ(x, y). f x y)
    using integral_prod_continuous [OF assms] by auto
    also have ... = integral (cbox (c, a) (d, b)) (λ(x, y). f y x)
    by (rule integral_swap_2dim [OF assms])
    also have ... = integral (cbox c d) (λy. integral (cbox a b) (λx. f x y))
    using integral_prod_continuous [OF swap_continuous] assms
    by auto
    finally show ?thesis .
  qed

```

7.14.47 Definite integrals for exponential and power function

```

lemma has_integral_exp_minus_to_infinity:
  assumes a: a > 0

```

```

shows (( $\lambda x::\text{real}. \exp(-a*x)$ ) has_integral  $\exp(-a*c)/a$ ) {c..}
proof -
  define f where  $f = (\lambda k\ x. \text{if } x \in \{c.. \text{real } k\} \text{ then } \exp(-a*x) \text{ else } 0)$ 
  {
    fix k :: nat assume k: of_nat k  $\geq c$ 
    from k a
    have (( $\lambda x. \exp(-a*x)$ ) has_integral ( $-\exp(-a*\text{real } k)/a - (-\exp(-a*c)/a)$ )
  {c..real k}
    by (intro fundamental_theorem_of_calculus)
      (auto intro!: derivative_eq_intros
        simp: has_real_derivative_iff_has_vector_derivative [symmetric])
    hence (f k has_integral ( $\exp(-a*c)/a - \exp(-a*\text{real } k)/a$ ) {c..} unfolding
f_def
    by (subst has_integral_restrict) simp_all
  } note has_integral_f = this

  have [simp]:  $f\ k = (\lambda_. 0)$  if of_nat k < c for k using that by (auto simp:
fun_eq_iff f_def)
  have integral_f: integral {c..} (f k) =
    (if real k  $\geq c$  then  $\exp(-a*c)/a - \exp(-a*\text{real } k)/a$  else 0)
    for k using integral_unique[OF has_integral_f[of k]] by simp

  have A: ( $\lambda x. \exp(-a*x)$ ) integrable_on {c..}  $\wedge$ 
    ( $\lambda k. \text{integral } \{c..\} (f\ k) \longrightarrow \text{integral } \{c..\} (\lambda x. \exp(-a*x))$ )
  proof (intro monotone_convergence_increasing_allI ballI)
    fix k :: nat
    have ( $\lambda x. \exp(-a*x)$ ) integrable_on {c..of_real k}
    unfolding f_def by (auto intro!: continuous_intros integrable_continuous_real)
    hence (f k) integrable_on {c..of_real k}
    by (rule integrable_eq) (simp add: f_def)
    then show f k integrable_on {c..}
    by (rule integrable_on_superset) (auto simp: f_def)
  next
    fix x assume x:  $x \in \{c..\}$ 
    have sequentially  $\leq$  principal {nat  $\lceil x \rceil ..$ } unfolding at_top_def by (simp add:
Inf_lower)
    also have {nat  $\lceil x \rceil ..$ }  $\subseteq$  {k. x  $\leq$  real k} by auto
    also have inf (principal ...) (principal {k.  $\neg x \leq$  real k}) =
      principal ({k. x  $\leq$  real k}  $\cap$  {k.  $\neg x \leq$  real k})
    by simp
    also have {k. x  $\leq$  real k}  $\cap$  {k.  $\neg x \leq$  real k} = {} by blast
    finally have inf sequentially (principal {k.  $\neg x \leq$  real k}) = bot
    by (simp add: inf.coboundedI1 bot_unique)
    with x show ( $\lambda k. f\ k\ x$ )  $\longrightarrow$   $\exp(-a*x)$  unfolding f_def
    by (intro filterlim_I1) auto
  next
    have  $|\text{integral } \{c..\} (f\ k)| \leq \exp(-a*c)/a$  for k
    proof (cases  $c > \text{of\_nat } k$ )
      case False

```

hence $abs (integral \{c..\} (f k)) = abs (exp (- (a * c)) / a - exp (- (a * real k)) / a)$
by (*simp add: integral_f*)
also have $abs (exp (- (a * c)) / a - exp (- (a * real k)) / a) =$
 $exp (- (a * c)) / a - exp (- (a * real k)) / a$
using *False a by (intro abs_of_nonneg) (simp_all add: field_simps)*
also have $\dots \leq exp (- a * c) / a$ **using** *a by simp*
finally show *?thesis .*
qed (*insert a, simp_all add: integral_f*)
thus *bounded (range($\lambda k. integral \{c..\} (f k)$))*
by (*intro boundedI[of _ exp (-a*c)/a] auto*)
qed (*auto simp: f_def*)
have ($\lambda k. exp (-a*c)/a - exp (-a * of_nat k)/a \longrightarrow exp (-a*c)/a - 0/a$)
by (*intro tendsto_intros filterlim_compose[OF exp_at_bot]*
filterlim_tendsto_neg_mult_at_bot[OF tendsto_const] filterlim_real_sequentially)
(use a in simp_all)
moreover
from *eventually_gt_at_top[of nat [c]]* **have** *eventually ($\lambda k. of_nat k > c$)*
sequentially
by *eventually_elim linarith*
hence *eventually ($\lambda k. exp (-a*c)/a - exp (-a * of_nat k)/a = integral \{c..\}$*
(f k)) sequentially
by *eventually_elim (simp add: integral_f)*
ultimately have ($\lambda k. integral \{c..\} (f k) \longrightarrow exp (-a*c)/a - 0/a$)
by (*rule Lim_transform_eventually*)
from *LIMSEQ_unique[OF conjunct2[OF A] this]*
have $integral \{c..\} (\lambda x. exp (-a*x)) = exp (-a*c)/a$ **by** *simp*
with *conjunct1[OF A]* **show** *?thesis*
by (*simp add: has_integral_integral*)
qed

lemma integrable_on_exp_minus_to_infinity: $a > 0 \implies (\lambda x. exp (-a*x) :: real)$
integrable_on {c..}
using *has_integral_exp_minus_to_infinity[of a c]* **unfolding** *integrable_on_def*
by *blast*

lemma has_integral_pour_from_0:
assumes $a > (-1::real)$ **and** $c \geq 0$
shows $((\lambda x. x \text{ pour } a) \text{ has_integral } (c \text{ pour } (a+1) / (a+1))) \{0..c\}$
proof (*cases c = 0*)
case *False*
define *f* **where** $f = (\lambda k x. \text{if } x \in \{\text{inverse } (of_nat (Suc k))..c\} \text{ then } x \text{ pour } a$
else 0)
define *F* **where** $F = (\lambda k. \text{if } \text{inverse } (of_nat (Suc k)) \leq c \text{ then}$
 $c \text{ pour } (a+1)/(a+1) - \text{inverse } (real (Suc k)) \text{ pour}$
 $(a+1)/(a+1) \text{ else } 0)$
have *has_integral_f: (f k has_integral F k) {0..c}* **for** $k::nat$
proof (*cases inverse (of_nat (Suc k)) ≤ c*)
case *True*

```

have  $x: x > 0$  if  $x \geq \text{inverse } (1 + \text{real } k)$  for  $x$ 
  by (smt (verit) that inverse_Suc of_nat_Suc)
hence  $((\lambda x. x \text{ powr } a) \text{ has\_integral } c \text{ powr } (a + 1) / (a + 1) -$ 
   $\text{inverse } (\text{real } (\text{Suc } k)) \text{ powr } (a + 1) / (a + 1)) \{ \text{inverse } (\text{real } (\text{Suc}$ 
 $k))..c \}$ 
  using True  $a$  by (intro fundamental_theorem_of_calculus)
  (auto intro!: derivative_eq_intros continuous_on_powr' continuous_on_const
  simp: has_real_derivative_iff_has_vector_derivative [symmetric])
with True show ?thesis unfolding f_def F_def by (subst has_integral_restrict)
simp_all
next
  case False
  thus ?thesis unfolding f_def F_def
    by (subst has_integral_restrict) auto
qed
then have integral_f:  $\text{integral } \{0..c\} (f k) = F k$  for  $k$ 
  by blast

have A:  $(\lambda x. x \text{ powr } a) \text{ integrable\_on } \{0..c\} \wedge$ 
   $(\lambda k. \text{integral } \{0..c\} (f k)) \longrightarrow \text{integral } \{0..c\} (\lambda x. x \text{ powr } a)$ 
proof (intro monotone_convergence_increasing ballI allI)
  fix  $k$  from has_integral_f[of  $k$ ] show  $f k$  integrable_on  $\{0..c\}$ 
    by (auto simp: integrable_on_def)
next
  fix  $k :: \text{nat}$  and  $x :: \text{real}$ 
  {
    assume  $x: \text{inverse } (\text{real } (\text{Suc } k)) \leq x$ 
    then have  $\text{inverse } (\text{real } (\text{Suc } (\text{Suc } k))) \leq x$ 
      using dual_order.trans by fastforce
  }
thus  $f k x \leq f (\text{Suc } k) x$  by (auto simp: f_def simp del: of_nat_Suc)
next
  fix  $x$  assume  $x: x \in \{0..c\}$ 
  show  $(\lambda k. f k x) \longrightarrow x \text{ powr } a$ 
  proof (cases  $x = 0$ )
    case False
    with  $x$  have  $x > 0$  by simp
    from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
      have eventually  $(\lambda k. x \text{ powr } a = f k x)$  sequentially
      by eventually_elim (insert  $x$ , simp add: f_def)
    moreover have  $(\lambda_. x \text{ powr } a) \longrightarrow x \text{ powr } a$  by simp
    ultimately show ?thesis by (blast intro: Lim_transform_eventually)
  qed (simp_all add: f_def)
next
  {
    fix  $k$ 
    from  $a$  have  $F k \leq c \text{ powr } (a + 1) / (a + 1)$ 
      by (auto simp add: F_def divide_simps)
    also from  $a$  have  $F k \geq 0$ 

```

by (auto simp: F_def divide_simps simp del: of_nat_Suc intro!: powr_mono2)
 hence $F k = \text{abs } (F k)$ by simp
 finally have $\text{abs } (F k) \leq c \text{ powr } (a + 1) / (a + 1)$.
 }
 thus bounded (range($\lambda k. \text{integral } \{0..c\} (f k)$))
 by (intro boundedI[of _ c powr (a+1) / (a+1)]) (auto simp: integral_f)
 qed

from False c have $c > 0$ by simp
 from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
 have eventually ($\lambda k. c \text{ powr } (a + 1) / (a + 1) - \text{inverse } (\text{real } (\text{Suc } k)) \text{ powr } (a+1) / (a+1) =$
 $\text{integral } \{0..c\} (f k)$ sequentially
 by eventually_elim (simp add: integral_f F_def)
 moreover have ($\lambda k. c \text{ powr } (a + 1) / (a + 1) - \text{inverse } (\text{real } (\text{Suc } k)) \text{ powr } (a + 1) / (a + 1)$)
 $\longrightarrow c \text{ powr } (a + 1) / (a + 1) - 0 \text{ powr } (a + 1) / (a + 1)$
 using a by (intro tendsto_intros LIMSEQ_inverse_real_of_nat) auto
 hence ($\lambda k. c \text{ powr } (a + 1) / (a + 1) - \text{inverse } (\text{real } (\text{Suc } k)) \text{ powr } (a + 1) / (a + 1)$)
 $\longrightarrow c \text{ powr } (a + 1) / (a + 1)$ by simp
 ultimately have ($\lambda k. \text{integral } \{0..c\} (f k)$) $\longrightarrow c \text{ powr } (a+1) / (a+1)$
 by (blast intro: Lim_transform_eventually)
 with A have $\text{integral } \{0..c\} (\lambda x. x \text{ powr } a) = c \text{ powr } (a+1) / (a+1)$
 by (blast intro: LIMSEQ_unique)
 with A show ?thesis by (simp add: has_integral_integral)
 qed (simp_all add: has_integral_refl)

lemma integrable_on_powr_from_0:
 assumes $a: a > (-1::\text{real})$ and $c: c \geq 0$
 shows ($\lambda x. x \text{ powr } a$) integrable_on $\{0..c\}$
 using has_integral_powr_from_0[OF assms] unfolding integrable_on_def by
 blast

lemma has_integral_powr_to_inf:
 fixes $a e :: \text{real}$
 assumes $e < -1$ $a > 0$
 shows ($\lambda x. x \text{ powr } e$) has_integral $-(a \text{ powr } (e + 1)) / (e + 1)$ $\{a.. \}$
proof -
 define $f :: \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}$ where $f = (\lambda n x. \text{if } x \in \{a..n\} \text{ then } x \text{ powr } e \text{ else } 0)$
 define F where $F = (\lambda x. x \text{ powr } (e + 1) / (e + 1))$

have has_integral_f: ($f n$ has_integral $(F n - F a)$) $\{a.. \}$
 if $n: n \geq a$ for $n :: \text{nat}$
proof -
 from n assms have ($\lambda x. x \text{ powr } e$) has_integral $(F n - F a)$ $\{a..n\}$
 by (intro fundamental_theorem_of_calculus) (auto intro!: derivative_eq_intros
 simp: has_real_derivative_iff_has_vector_derivative [symmetric] F_def)


```

    hence (f n has_integral (F n - F a)) {a..n}
      by (rule has_integral_eq [rotated]) (simp add: f_def)
    thus (f n has_integral (F n - F a)) {a..}
      by (rule has_integral_on_superset) (auto simp: f_def)
  qed
  have integral_f: integral {a..} (f n) = (if n ≥ a then F n - F a else 0) for n ::
nat
  proof (cases n ≥ a)
    case True
      with has_integral_f[OF this] show ?thesis by (simp add: integral_unique)
    next
      case False
        have (f n has_integral 0) {a} by (rule has_integral_refl)
        hence (f n has_integral 0) {a..}
          using False f_def by force
        with False show ?thesis by (simp add: integral_unique)
  qed

  have *: (λx. x powr e) integrable_on {a..} ∧
    (λn. integral {a..} (f n)) ⟶ integral {a..} (λx. x powr e)
  proof (intro monotone_convergence_increasing_allI ballI)
    fix n :: nat
    from assms have (λx. x powr e) integrable_on {a..n}
      by (auto intro!: integrable_continuous_real continuous_intros)
    hence f n integrable_on {a..n}
      by (rule integrable_eq) (auto simp: f_def)
    thus f n integrable_on {a..}
      by (rule integrable_on_superset) (auto simp: f_def)
  next
    fix n :: nat and x :: real
    show f n x ≤ f (Suc n) x by (auto simp: f_def)
  next
    fix x :: real assume x: x ∈ {a..}
    from filterlim_real_sequentially
      have eventually (λn. real n ≥ x) at_top
        by (simp add: filterlim_at_top)
    with x have eventually (λn. f n x = x powr e) at_top
      by (auto elim!: eventually_mono simp: f_def)
    thus (λn. f n x) ⟶ x powr e by (simp add: tendsto_eventually)
  next
    have norm (integral {a..} (f n)) ≤ -F a for n :: nat
  proof (cases n ≥ a)
    case True
      with assms have a powr (e + 1) ≥ n powr (e + 1)
        by (intro powr_mono2') simp_all
      with assms show ?thesis by (auto simp: divide_simps F_def integral_f)
  qed (use assms in ⟨simp add: integral_f F_def field_split_simps⟩)
  thus bounded (range(λk. integral {a..} (f k)))
    unfolding bounded_iff by (intro exI[of _ -F a]) auto

```

qed

from *filterlim_real_sequentially*
 have *eventually* $(\lambda n. \text{real } n \geq a)$ *at_top*
 by (*simp add: filterlim_at_top*)
 hence *eventually* $(\lambda n. F\ n - F\ a = \text{integral } \{a..\} (f\ n))$ *at_top*
 by *eventually_elim* (*simp add: integral_f*)
 moreover have $(\lambda n. F\ n - F\ a) \longrightarrow 0 / (e + 1) - F\ a$ **unfolding** *F_def*
 by (*insert assms, (rule tendsto_intros filterlim_compose[OF tendsto_neg_powr]*
filterlim_ident filterlim_real_sequentially | simp)+)
 hence $(\lambda n. F\ n - F\ a) \longrightarrow -F\ a$ **by** *simp*
 ultimately have $(\lambda n. \text{integral } \{a..\} (f\ n)) \longrightarrow -F\ a$ **by** (*blast intro: Lim_transform_eventually*)
 then have $\text{integral } \{a..\} (\lambda x. x\ \text{powr } e) = -F\ a$
 using * *LIMSEQ_unique* **by** *blast*
 with * **show** *?thesis*
 by (*simp add: has_integral_integral F_def*)

qed

lemma *has_integral_inverse_power_to_inf*:

fixes *a* :: *real* and *n* :: *nat*

assumes $n > 1$ $a > 0$

shows $((\lambda x. 1 / x^n) \text{ has_integral } 1 / (\text{real } (n - 1) * a^{(n - 1)})) \{a..\}$

proof -

from *assms* have *real_of_int* $(-int\ n) < -1$ **by** *simp*

note *has_integral_powr_to_inf*[*OF this* $\langle a > 0 \rangle$]

also have $-(a\ \text{powr } (\text{real_of_int } (-int\ n) + 1)) / (\text{real_of_int } (-int\ n) + 1) =$

$$1 / (\text{real } (n - 1) * a\ \text{powr } (\text{real } (n - 1))) \text{ using } \textit{assms}$$

by (*simp add: field_split_simps powr_add [symmetric] of_nat_diff*)

also from *assms* have $a\ \text{powr } (\text{real } (n - 1)) = a^{(n - 1)}$

by (*intro powr_realpow*)

finally show *?thesis*

by (*rule has_integral_eq [rotated]*)

(*insert assms, simp_all add: powr_minus powr_realpow field_split_simps*)

qed

Adaption to ordered Euclidean spaces and the Cartesian Euclidean space

lemma *integral_component_eq_cart*[*simp*]:

fixes *f* :: '*n*::*euclidean_space* \Rightarrow *real*^{*m*}

assumes *f* *integrable_on* *s*

shows *integral* *s* $(\lambda x. f\ x\ \$\ k) = \text{integral } s\ f\ \$\ k$

using *integral_linear*[*OF assms*(1) *bounded_linear_vec_nth,unfolding o_def*].

lemma *content_closed_interval*:

fixes *a* :: '*a*::*ordered_euclidean_space*

assumes $a \leq b$

shows *content* $\{a..b\} = (\prod_{i \in \text{Basis.}} b \cdot i - a \cdot i)$

using *content_cbox*[of *a b*] *assms* **by** (*simp add: cbox_interval eucl_le*[**where** 'a='a])

lemma *integrable_const_ivl*[*intro*]:
fixes *a*::'a::ordered_euclidean_space
shows ($\lambda x. c$) *integrable_on* {*a..b*}
unfolding *cbox_interval*[*symmetric*] **by** (*rule integrable_const*)

lemma *integrable_on_subinterval*:
fixes *f* :: 'n::ordered_euclidean_space \Rightarrow 'a::banach
assumes *f integrable_on S* {*a..b*} $\subseteq S$
shows *f integrable_on* {*a..b*}
using *integrable_on_subcbox*[of *f S a b*] *assms* **by** (*simp add: cbox_interval*)

end

Chapter 8

Kronecker's Theorem with Applications

```
theory Kronecker_Approximation_Theorem
```

```
imports Complex_Transcendental Henstock_Kurzweil_Integration
        HOL-Real_Asymp.Real_Asymp
```

```
begin
```

8.1 Dirichlet's Approximation Theorem

Simultaneous version. From Hardy and Wright, An Introduction to the Theory of Numbers, page 170.

```
theorem Dirichlet_approx_simult:
```

```
  fixes  $\vartheta :: nat \Rightarrow real$  and  $N n :: nat$ 
```

```
  assumes  $N > 0$ 
```

```
  obtains  $q p$  where  $0 < q \leq \text{int } (N^n)$ 
```

```
    and  $\bigwedge i. i < n \implies |\text{of\_int } q * \vartheta i - \text{of\_int } (p i)| < 1/N$ 
```

```
proof -
```

```
  have  $\text{lessN}: \lfloor x * \text{real } N \rfloor < N$  if  $0 \leq x < 1$  for  $x$ 
```

```
  proof -
```

```
    have  $\lfloor x * \text{real } N \rfloor < N$ 
```

```
      using that by (simp add: assms floor_less_iff)
```

```
    with assms show ?thesis by linarith
```

```
qed
```

```
define interv where  $\text{interv} \equiv \lambda k. \{ \text{real } k/N ..< \text{Suc } k/N \}$ 
```

```
define fracs where  $\text{fracs} \equiv \lambda k. \text{map } (\lambda i. \text{frac } (\text{real } k * \vartheta i)) [0..<n]$ 
```

```
define X where  $X \equiv \text{fracs } ' \{ ..N^n \}$ 
```

```
define Y where  $Y \equiv \text{set } (\text{List.n\_lists } n (\text{map } \text{interv } [0..<N]))$ 
```

```
have interv_iff:  $\text{interv } k = \text{interv } k' \iff k=k'$  for  $k k'$ 
```

```
  using assms by (auto simp: interv_def Ico_eq_Ico divide_strict_right_mono)
```

```
have in_interv:  $x \in \text{interv } ( \lfloor x * \text{real } N \rfloor )$  if  $x \geq 0$  for  $x$ 
```

```
  using that assms by (simp add: interv_def divide_simps) linarith
```

```

have False
  if non:  $\forall a b. b \leq N \hat{n} \longrightarrow a < b \longrightarrow \neg(\forall i < n. |\text{frac}(\text{real } b * \vartheta i) - \text{frac}(\text{real } a * \vartheta i)| < 1/N)$ 
proof -
  have inj_on ( $\lambda k. \text{map}(\lambda i. \text{frac}(k * \vartheta i)) [0..<n]$ )  $\{..N \hat{n}\}$ 
    using that assms by (intro linorder_inj_onI) fastforce+
  then have caX:  $\text{card } X = \text{Suc}(N \hat{n})$ 
    by (simp add: X_def fracs_def card_image)
  have caY:  $\text{card } Y \leq N \hat{n}$ 
    by (metis Y_def card_length diff_zero length_map length_n_lists length_upt)
  define f where  $f \equiv \lambda l. \text{map}(\lambda x. \text{inter}(\text{nat} \lfloor x * \text{real } N \rfloor)) l$ 
  have f'  $X \subseteq Y$ 
    by (auto simp: f_def Y_def X_def fracs_def o_def set_n_lists frac_lt_1 lessN)
  then have  $\neg \text{inj\_on } f X$ 
    using Y_def caX caY card_inj_on_le not_less_eq_eq by fastforce
  then obtain  $x x'$  where  $x \neq x' \ x \in X \ x' \in X$  and  $\text{eq}: f x = f x'$ 
    by (auto simp: inj_on_def)
  then obtain  $c c'$  where  $c: c \neq c' \ c \leq N \hat{n} \ c' \leq N \hat{n}$ 
    and  $\text{xeq}: x = \text{fracs } c$  and  $\text{xeq}': x' = \text{fracs } c'$ 
    unfolding X_def by (metis atMost_iff image_iff)
  have [simp]:  $\text{length } x' = n$ 
    by (auto simp: xeq' fracs_def)
  have  $\text{ge0}: x^!i \geq 0$  if  $i < n$  for  $i$ 
    using that  $\langle x' \in X \rangle$  by (auto simp: X_def fracs_def)
  have  $f x \in Y$ 
    using  $\langle f' X \subseteq Y \rangle \langle x \in X \rangle$  by blast
  define k where  $k \equiv \text{map}(\lambda r. \text{nat} \lfloor r * \text{real } N \rfloor) x$ 
  have  $|\text{frac}(\text{real } c * \vartheta i) - \text{frac}(\text{real } c' * \vartheta i)| < 1/N$  if  $i < n$  for  $i$ 
  proof -
    have  $k: x^!i \in \text{inter}(k^!i)$ 
      using  $\langle i < n \rangle$  assms by (auto simp: divide_simps k_def inter_def xeq fracs_def) linarith
    then have  $k': x^!i \in \text{inter}(k^!i)$ 
      using  $\langle i < n \rangle$  eq assms  $\text{ge0}[OF \langle i < n \rangle]$ 
      by (auto simp: k_def f_def divide_simps map_equality_iff in_inter inter_iff)
    with assms k show ?thesis
      using  $\langle i < n \rangle$  by (auto simp add: xeq xeq' fracs_def inter_def add_divide_distrib)
  qed
  then show False
    by (metis c non nat_neq_iff abs_minus_commute)
  qed
  then obtain  $a b$  where  $a < b \ b \leq N \hat{n}$  and  $*$ :  $\bigwedge i. i < n \implies |\text{frac}(\text{real } b * \vartheta i) - \text{frac}(\text{real } a * \vartheta i)| < 1/N$ 
    by blast
  let ?k =  $b - a$ 
  let ?h =  $\lambda i. \lfloor b * \vartheta i \rfloor - \lfloor a * \vartheta i \rfloor$ 
  show ?thesis

```

```

proof
  fix  $i$ 
  assume  $i < n$ 
  have  $\text{frac } (b * \vartheta i) - \text{frac } (a * \vartheta i) = ?k * \vartheta i - ?h i$ 
    using  $\langle a < b \rangle$  by (simp add: frac_def left_diff_distrib' of_nat_diff)
  then show  $|\text{of\_int } ?k * \vartheta i - ?h i| < 1/N$ 
    by (metis *  $\langle i < n \rangle$  of_int_of_nat_eq)
  qed (use  $\langle a < b \rangle$   $\langle b \leq N \hat{n} \rangle$  in auto)
qed

```

Theorem 7.1

corollary *Dirichlet_approx*:

```

fixes  $\vartheta:: \text{real}$  and  $N:: \text{nat}$ 
assumes  $N > 0$ 
obtains  $h k$  where  $0 < k \leq \text{int } N$   $|\text{of\_int } k * \vartheta - \text{of\_int } h| < 1/N$ 
by (rule Dirichlet_approx_simult [OF assms, where  $n=1$  and  $\vartheta=\lambda_. \vartheta$ ]) auto

```

Theorem 7.2

corollary *Dirichlet_approx_coprime*:

```

fixes  $\vartheta:: \text{real}$  and  $N:: \text{nat}$ 
assumes  $N > 0$ 
obtains  $h k$  where coprime  $h k$   $0 < k \leq \text{int } N$   $|\text{of\_int } k * \vartheta - \text{of\_int } h| < 1/N$ 
proof -
  obtain  $h' k'$  where  $0 < k' \leq \text{int } N$  and  $*$ :  $|\text{of\_int } k' * \vartheta - \text{of\_int } h'| < 1/N$ 

```

```

  by (meson Dirichlet_approx assms)
  let  $?d = \text{gcd } h' k'$ 
  show thesis
  proof (cases ?d = 1)
    case True
      with  $k' * \text{that}$  show ?thesis by auto
    next
      case False
      then have  $1: ?d > 1$ 
        by (smt (verit, ccfv_threshold)  $\langle k' > 0 \rangle$  gcd_pos_int)
      let  $?k = k' \text{ div } ?d$ 
      let  $?h = h' \text{ div } ?d$ 
      show ?thesis
      proof
        show coprime ( $?h::\text{int}$ )  $?k$ 
          by (metis 1 div_gcd_coprime gcd_eq_0_iff_not_one_less_zero)
        show  $k0: 0 < ?k$ 
          using  $\langle k' > 0 \rangle$  pos_imp_zdiv_pos_iff by force
        show  $?k \leq \text{int } N$ 
          using  $k' 1 \text{ int\_div\_less\_self}$  by fastforce
        have  $|\vartheta - \text{of\_int } ?h / \text{of\_int } ?k| = |\vartheta - \text{of\_int } h' / \text{of\_int } k'|$ 
          by (simp add: real_of_int_div)
        also have  $\dots < 1 / \text{of\_int } (N * k')$ 

```

```

    using k' * by (simp add: field_simps)
  also have ... < 1 / of_int (N * ?k)
    using assms ‹k'>0› k0 1
    by (simp add: frac_less2 int_div_less_self)
  finally show |of_int ?k * ϑ - of_int ?h| < 1 / real N
    using k0 k' by (simp add: field_simps)
qed
qed
qed

```

definition *approx_set* :: *real* ⇒ (*int* × *int*) *set*
where *approx_set* ϑ ≡
 {(*h*, *k*) | *h* *k*::*int*. *k* > 0 ∧ *coprime* *h* *k* ∧ |ϑ - of_int *h* / of_int *k*| < 1/*k*²}
for ϑ::*real*

Theorem 7.3

```

lemma approx_set_nonempty: approx_set ϑ ≠ {}
proof -
  have |ϑ - of_int [ϑ] / of_int 1| < 1 / (of_int 1)2
    by simp linarith
  then have ([ϑ], 1) ∈ approx_set ϑ
    by (auto simp: approx_set_def)
  then show ?thesis
    by blast
qed

```

Theorem 7.4(c)

```

theorem infinite_approx_set:
  assumes infinite (approx_set ϑ)
  shows ∃ h k. (h,k) ∈ approx_set ϑ ∧ k > K
proof (rule ccontr, simp add: not_less)
  assume Kb [rule_format]: ∀ h k. (h, k) ∈ approx_set ϑ → k ≤ K
  define H where H ≡ 1 + |K * ϑ|
  have k0: k > 0 if (h,k) ∈ approx_set ϑ for h k
    using approx_set_def that by blast
  have Hb: of_int |h| < H if (h,k) ∈ approx_set ϑ for h k
proof -
  have *: |k * ϑ - h| < 1
  proof -
  have |k * ϑ - h| < 1 / k
    using that by (auto simp: field_simps approx_set_def eval_nat_numeral)
  also have ... ≤ 1
    using divide_le_eq_1 by fastforce
  finally show ?thesis .
qed
have k > 0
  using approx_set_def that by blast
have of_int |h| ≤ |k * ϑ - h| + |k * ϑ|
  by force

```



```

    also have ... < 1 + |k *  $\vartheta$ |
      using * that by simp
    also have ...  $\leq H$ 
      using Kb [OF that] <k>0 by (auto simp add: H_def abs_if mult_right_mono
mult_less_0_iff)
    finally show ?thesis .
  qed
  have approx_set  $\vartheta \subseteq \{-(\text{ceiling } H).. \text{ceiling } H\} \times \{0..K\}$ 
    using Hb Kb k0
  apply (simp add: subset_iff)
  by (smt (verit, best) ceiling_add_of_int less_ceiling_iff)
  then have finite (approx_set  $\vartheta$ )
    by (meson finite_SigmaI finite_atLeastAtMost_int finite_subset)
  then show False
    using assms by blast
  qed

```

Theorem 7.4(b,d)

```

theorem rational_iff_finite_approx_set:
  shows  $\vartheta \in \mathbb{Q} \longleftrightarrow \text{finite } (\text{approx\_set } \vartheta)$ 
proof
  assume  $\vartheta \in \mathbb{Q}$ 
  then obtain a b where eq:  $\vartheta = \text{of\_int } a / \text{of\_int } b$  and  $b > 0$  and coprime a b
    by (meson Rats_cases')
  then have ab:  $(a,b) \in \text{approx\_set } \vartheta$ 
    by (auto simp: approx_set_def)
  show finite (approx_set  $\vartheta$ )
  proof (rule ccontr)
    assume infinite (approx_set  $\vartheta$ )
    then obtain h k where  $(h,k) \in \text{approx\_set } \vartheta$   $k > b$   $k > 0$ 
      using infinite_approx_set by (smt (verit, best))
    then have coprime h k and hk:  $|a/b - h/k| < 1 / (\text{of\_int } k)^2$ 
      by (auto simp: approx_set_def eq)
    have False if  $a * k = h * b$ 
  proof -
    have b dvd k
      using that <coprime a b>
      by (metis coprime_commute coprime_dvd_mult_right_iff dvd_triv_right)
    then obtain d where  $k = b * d$ 
      by (metis dvd_def)
    then have  $h = a * d$ 
      using <0 < b> that by force
    then show False
      using <0 < b> <b < k> <coprime h k> <k = b * d> by auto
  qed
  then have 0:  $0 < |a*k - b*h|$ 
    by auto
  have  $|a*k - b*h| < b/k$ 
    using <k>0 <b>0 hk by (simp add: field_simps eval_nat_numeral)

```

```

    also have ... < 1
      by (simp add: <0 < k> <b < k>)
    finally show False
      using 0 by linarith
  qed
next
assume fin: finite (approx_set  $\vartheta$ )
show  $\vartheta \in \mathbb{Q}$ 
proof (rule ccontr)
  assume irr:  $\vartheta \notin \mathbb{Q}$ 
  define A where A  $\equiv ((\lambda(h,k). |\vartheta - h/k|) \text{ ` } \text{approx\_set } \vartheta)$ 
  let ? $\alpha$  = Min A
  have 0  $\notin$  A
    using irr by (auto simp: A_def approx_set_def)
  moreover have  $\forall x \in A. x \geq 0$  and A: finite A A  $\neq \{\}$ 
    using approx_set_nonempty by (auto simp: A_def fin)
  ultimately have  $\alpha$ : ? $\alpha$  > 0
    using Min_in by force
  then obtain N where N: real N > 1 / ? $\alpha$ 
    using reals_Archimedean2 by blast
  have 0 < 1 / ? $\alpha$ 
    using  $\alpha$  by auto
  also have ... < real N
    by (fact N)
  finally have N > 0
    by simp
  from N have 1/N < ? $\alpha$ 
    by (simp add:  $\alpha$  divide_less_eq mult.commute)
  then obtain h k where coprime h k 0 < k k  $\leq$  int N |of_int k *  $\vartheta$  - of_int
h| < 1/N
    by (metis Dirichlet_approx_coprime  $\alpha$  N divide_less_0_1_iff less_le not_less_iff_gr_or_eq
of_nat_0_le_iff of_nat_le_iff of_nat_0)
  then have  $\S$ :  $|\vartheta - h/k| < 1 / (k*N)$ 
    by (simp add: field_simps)
  also have ...  $\leq 1/k^2$ 
    using <k  $\leq$  int N> by (simp add: eval_nat_numeral divide_simps)
  finally have hk_in: (h,k)  $\in$  approx_set  $\vartheta$ 
    using <0 < k> <coprime h k> by (auto simp: approx_set_def)
  then have  $|\vartheta - h/k| \in A$ 
    by (auto simp: A_def)
  moreover have 1 / real_of_int (k * int N) < ? $\alpha$ 
  proof -
    have 1 / real_of_int (k * int N) = 1 / real N / of_int k
      by simp
    also have ... < ? $\alpha$  / of_int k
      using <k > 0>  $\alpha$  <N > 0> N by (intro divide_strict_right_mono) (auto
simp: field_simps)
    also have ...  $\leq$  ? $\alpha$  / 1
      using  $\alpha$  <k > 0> by (intro divide_left_mono) auto

```

```

    finally show ?thesis
      by simp
  qed
  ultimately show False using A § by auto
  qed
  qed

```

No formalisation of Liouville's Approximation Theorem because this is already in the AFP as Liouville_Numbers. Apostol's Theorem 7.5 should be exactly the theorem liouville_irrational_algebraic. There is a minor discrepancy in the definition of "Liouville number" between Apostol and Eberl: he requires the denominator to be positive, while Eberl require it to exceed 1.

8.2 Kronecker's Approximation Theorem: the One-dimensional Case

```

lemma frac_int_mult:
  assumes  $m > 0$  and  $le: 1 - \text{frac } r \leq 1/m$ 
  shows  $1 - \text{frac } (\text{of\_int } m * r) = m * (1 - \text{frac } r)$ 
  proof -
    have  $\text{frac } (\text{of\_int } m * r) = 1 - m * (1 - \text{frac } r)$ 
    proof (subst frac_unique_iff, intro conjI)
      show  $\text{of\_int } m * r - (1 - \text{of\_int } m * (1 - \text{frac } r)) \in \mathbb{Z}$ 
        by (simp add: algebra_simps frac_def)
    qed (use assms in <auto simp: divide_simps mult_ac frac_lt_1>)
    then show ?thesis
      by simp
  qed

```

Concrete statement of Theorem 7.7, and the real proof

```

theorem Kronecker_approx_1_explicit:
  fixes  $\vartheta :: \text{real}$ 
  assumes  $\vartheta \notin \mathbb{Q}$  and  $\alpha: 0 \leq \alpha \leq 1$  and  $\varepsilon > 0$ 
  obtains  $k$  where  $k > 0 \mid \text{frac}(\text{real } k * \vartheta) - \alpha < \varepsilon$ 
  proof -
    obtain  $N :: \text{nat}$  where  $1/N < \varepsilon \wedge N > 0$ 
      by (metis <math>\varepsilon > 0</math> gr_zeroI inverse_eq_divide real_arch_inverse)
    then obtain  $h k$  where  $0 < k$  and  $hk: \mid \text{of\_int } k * \vartheta - \text{of\_int } h \mid < \varepsilon$ 
      using Dirichlet_approx that by (metis less_trans)
    have irrat:  $\text{of\_int } n * \vartheta \in \mathbb{Q} \implies n = 0$  for  $n$ 
      by (metis Rats_divide Rats_of_int assms(1) nonzero_mult_div_cancel_left
        of_int_0_eq_iff)
    then consider  $\text{of\_int } k * \vartheta < \text{of\_int } h \mid \text{of\_int } k * \vartheta > \text{of\_int } h$ 
      by (metis Rats_of_int <math>0 < k</math> less_irrefl linorder_neqE_linordered_idom)
    then show thesis
      proof cases

```

```

case 1
define  $\xi$  where  $\xi \equiv 1 - \text{frac} (\text{of\_int } k * \vartheta)$ 
have pos:  $\xi > 0$ 
  by (simp add:  $\xi\_def \text{frac\_lt\_1}$ )
define  $N$  where  $N \equiv \lfloor 1/\xi \rfloor$ 
have  $N > 0$ 
  by (simp add:  $N\_def \xi\_def \text{frac\_lt\_1}$ )
have False if  $1/\xi \in \mathbb{Z}$ 
proof -
  from that of_int_ceiling
  obtain  $r$  where  $r: \text{of\_int } r = 1/\xi$  by blast
  then obtain  $s$  where  $s: \text{of\_int } k * \vartheta = \text{of\_int } s + 1 - 1/r$ 
    by (simp add:  $\xi\_def \text{frac\_def}$ )
  from  $r$  pos  $s \langle k > 0 \rangle$  have  $\vartheta = (\text{of\_int } s + 1 - 1/r) / k$ 
    by (auto simp: field_simps)
  with assms show False
    by simp
qed
then have  $N0: N < 1/\xi$ 
  unfolding  $N\_def$  by (metis Ints_of_int floor_correct less_le)
then have  $N2: 1/(N+1) < \xi$ 
  unfolding  $N\_def$  by (smt (verit) divide_less_0_iff divide_less_eq floor_correct
mult.commute pos)
have  $\xi * (N+1) > 1$ 
  by (smt (verit)  $N2 \langle 0 < N \rangle \text{of\_int\_1\_less\_iff\_pos\_divide\_less\_eq}$ )
then have  $ex: \exists m. \text{int } m \leq N+1 \wedge 1-\alpha < m * \xi$ 
  by (smt (verit, best)  $\langle 0 < N \rangle \langle 0 \leq \alpha \rangle \text{floor\_of\_int\_floor\_of\_nat mult.commute}$ 
of_nat_nat)
define  $m$  where  $m \equiv \text{LEAST } m. \text{int } m \leq N+1 \wedge 1-\alpha < m * \xi$ 
have  $m: \text{int } m \leq N+1 \wedge 1-\alpha < m * \xi$ 
  using LeastI_ex [OF ex] unfolding  $m\_def$  by blast
have  $m > 0$ 
  using  $m \text{ gr0I } \langle \alpha \leq 1 \rangle$  by force
have  $k\vartheta: \xi < \varepsilon$ 
  using  $hk \ 1$  by (smt (verit, best) floor_eq_iff frac_def  $\xi\_def$ )
show thesis
proof (cases  $m=1$ )
case True
  then have  $|\text{frac} (\text{real} (\text{nat } k) * \vartheta) - \alpha| < \varepsilon$ 
    using  $m \langle \alpha \leq 1 \rangle \langle 0 < k \rangle \xi\_def k\vartheta$  by force
  with  $\langle 0 < k \rangle \text{zero\_less\_nat\_eq}$  that show thesis by blast
case False
  with  $\langle 0 < m \rangle$  have  $m > 1$  by linarith
  have  $\xi < 1 / N$ 
  by (metis  $N0 \langle 0 < N \rangle \text{mult\_of\_int\_commute of\_int\_pos pos\_less\_divide\_eq}$ )
  also have  $\dots \leq 1 / (\text{real } m - 1)$ 
    using  $\langle m > 1 \rangle m$  by (simp add: divide_simps)
  finally have  $\xi < 1 / (\text{real } m - 1)$  .

```

```

then have  $m1\_eq$ :  $(int\ m - 1) * \xi = 1 - frac\ (of\_int\ ((int\ m - 1) * k) * \vartheta)$ 
using  $frac\_int\_mult$  [ $of\ (int\ m - 1)\ k * \vartheta$ ]  $\langle 1 < m \rangle$ 
by ( $simp\ add$ :  $\xi\_def\ mult.assoc$ )
then
have  $m1\_eq'$ :  $frac\ (of\_int\ ((int\ m - 1) * k) * \vartheta) = 1 - (int\ m - 1) * \xi$ 
by  $simp$ 
moreover have  $(m - Suc\ 0) * \xi \leq 1 - \alpha$ 
using  $Least\_le$  [where  $k=m-Suc\ 0$ ]  $m$ 
by ( $smt\ (verit, best)\ Suc\_n\_not\_le\_n\ Suc\_pred\ \langle 0 < m \rangle\ m\_def\ of\_nat\_le\_iff$ )
ultimately have  $le\ \alpha$ :  $\alpha \leq frac\ (of\_int\ ((int\ m - 1) * k) * \vartheta)$ 
by ( $simp\ add$ :  $Suc\_leI\ \langle 0 < m \rangle\ of\_nat\_diff$ )
moreover have  $m * \xi + frac\ (of\_int\ ((int\ m - 1) * k) * \vartheta) = \xi + 1$ 
by ( $subst\ m1\_eq'$ ) ( $simp\ add$ :  $\xi\_def\ algebra\_simps$ )
ultimately have  $|frac\ ((int\ (m - 1) * k) * \vartheta) - \alpha| < \varepsilon$ 
by ( $smt\ (verit, best)\ One\_nat\_def\ Suc\_leI\ \langle 0 < m \rangle\ int\_ops(2)\ k\ \vartheta\ m$ 
 $of\_nat\_diff$ )
with that show thesis
by ( $metis\ \langle 0 < k \rangle\ \langle 1 < m \rangle\ mult\_pos\_pos\ of\_int\_of\_nat\_eq\ of\_nat\_mult$ 
 $pos\_int\_cases\ zero\_less\_diff$ )
qed
next
case 2
define  $\xi$  where  $\xi \equiv frac\ (of\_int\ k * \vartheta)$ 
have  $recip\_frac$ : False if  $1/\xi \in \mathbb{Z}$ 
proof -
have  $frac\ (of\_int\ k * \vartheta) \in \mathbb{Q}$ 
using that unfolding  $\xi\_def$ 
by ( $metis\ Ints\_cases\ Rats\_1\ Rats\_divide\ Rats\_of\_int\ div\_by\_1\ di$ 
 $vide\_divide\_eq\_right\ mult\_cancel\_right2$ )
then show False
using  $\langle 0 < k \rangle\ frac\_in\_Rats\_iff\ irrat$  by  $blast$ 
qed
have  $pos$ :  $\xi > 0$ 
by ( $metis\ \xi\_def\ Ints\_0\ division\_ring\_divide\_zero\ frac\_unique\_iff\ less\_le$ 
 $recip\_frac$ )
define  $N$  where  $N \equiv \lfloor 1 / \xi \rfloor$ 
have  $N > 0$ 
unfolding  $N\_def$ 
by ( $smt\ (verit)\ \xi\_def\ divide\_less\_eq\_1\_pos\ floor\_less\_one\ frac\_lt\_1\ pos$ )
have  $N0$ :  $N < 1 / \xi$ 
unfolding  $N\_def$  by ( $metis\ Ints\_of\_int\ floor\_eq\_iff\ less\_le\ recip\_frac$ )
then have  $N2$ :  $1/(N+1) < \xi$ 
unfolding  $N\_def$ 
by ( $smt\ (verit, best)\ divide\_less\_0\_iff\ divide\_less\_eq\ floor\_correct\ mult.commute$ 
 $pos$ )
have  $\xi * (N+1) > 1$ 
by ( $smt\ (verit)\ N2\ \langle 0 < N \rangle\ of\_int\_1\_less\_iff\ pos\_divide\_less\_eq$ )
then have  $ex$ :  $\exists m. int\ m \leq N+1 \wedge \alpha < m * \xi$ 

```

```

    by (smt (verit, best) mult.commute ⟨ $\alpha \leq 1$ ⟩ ⟨ $0 < N$ ⟩ of_int_of_nat_eq
pos_int_cases)
  define m where m  $\equiv$  LEAST m. int m  $\leq$  N+1  $\wedge$   $\alpha <$  m *  $\xi$ 
  have m: int m  $\leq$  N+1  $\wedge$   $\alpha <$  m *  $\xi$ 
    using LeastI_ex [OF ex] unfolding m_def by blast
  have m > 0
    using ⟨ $0 \leq \alpha$ ⟩ m gr0I by force
  have k $\vartheta$ :  $\xi <$   $\varepsilon$ 
    using hk 2 unfolding  $\xi$ _def by (smt (verit, best) floor_eq_iff frac_def)
  have mk_eq: frac (of_int (m*k) *  $\vartheta$ ) = m * frac (of_int k *  $\vartheta$ )
    if m>0 frac (of_int k *  $\vartheta$ ) < 1/m for m k
  proof (subst frac_unique_iff , intro conjI)
    show real_of_int (m * k) *  $\vartheta$  - real_of_int m * frac (real_of_int k *  $\vartheta$ )  $\in$ 
 $\mathbb{Z}$ 
    by (simp add: algebra_simps frac_def)
  qed (use that in ⟨auto simp: divide_simps mult_ac⟩)
  show thesis
  proof (cases m=1)
    case True
      then have |frac (real (nat k) *  $\vartheta$ ) -  $\alpha$ | <  $\varepsilon$ 
        using m  $\alpha$  ⟨ $0 < k$ ⟩  $\xi$ _def k $\vartheta$  by force
      with ⟨ $0 < k$ ⟩ zero_less_nat_eq that show ?thesis by blast
    next
      case False
        with ⟨ $0 < m$ ⟩ have m>1 by linarith
        with ⟨ $0 < k$ ⟩ have mk_pos:(m - Suc 0) * nat k > 0 by force
        have real_of_int (int m - 1) < 1 / frac (real_of_int k *  $\vartheta$ )
          using N0  $\xi$ _def m by force
        then
          have m1_eq: (int m - 1) *  $\xi$  = frac (((int m - 1) * k) *  $\vartheta$ )
            using m mk_eq [of int m-1 k] pos ⟨m>1 by (simp add: divide_simps
mult_ac  $\xi$ _def)
          moreover have (m - Suc 0) *  $\xi \leq \alpha$ 
            using Least_le [where k=m-Suc 0] m
          by (smt (verit, best) Suc_n_not_le_n Suc_pred ⟨ $0 < m$ ⟩ m_def of_nat_le_iff)
          ultimately have le $\alpha$ : frac (of_int ((int m - 1) * k) *  $\vartheta$ )  $\leq \alpha$ 
            by (simp add: Suc_leI ⟨ $0 < m$ ⟩ of_nat_diff)
          moreover have (m *  $\xi$  - frac (of_int ((int m - 1) * k) *  $\vartheta$ )) <  $\varepsilon$ 
            by (metis m1_eq add_diff_cancel_left' diff_add_cancel k $\vartheta$  left_diff_distrib'

          mult_cancel_right2 of_int_1 of_int_diff of_int_of_nat_eq)
          ultimately have |frac (real ((m - 1) * nat k) *  $\vartheta$ ) -  $\alpha$ | <  $\varepsilon$ 
            using ⟨ $0 < k$ ⟩ ⟨ $0 < m$ ⟩ by simp (smt (verit, best) One_nat_def Suc_leI
m of_nat_1 of_nat_diff)
          with ⟨m > 0⟩ that show thesis
            using mk_pos One_nat_def by presburger
        qed
      qed
    qed
  
```

Theorem 7.7 expressed more abstractly using *closure*

corollary *Kronecker_approx_1*:

```

fixes  $\vartheta :: \text{real}$ 
assumes  $\vartheta \notin \mathbb{Q}$ 
shows  $\text{closure} (\text{range} (\lambda n. \text{frac} (\text{real } n * \vartheta))) = \{0..1\}$  (is  $?C = \_$ )
proof -
have  $\exists k > 0. |\text{frac}(\text{real } k * \vartheta) - \alpha| < \varepsilon$  if  $0 \leq \alpha \leq 1$   $\varepsilon > 0$  for  $\alpha \varepsilon$ 
  by (meson Kronecker_approx_1_explicit assms that)
then have  $x \in ?C$  if  $0 \leq x \leq 1$  for  $x :: \text{real}$ 
  using that by (auto simp add: closure_approachable dist_real_def)
moreover
have  $(\text{range} (\lambda n. \text{frac} (\text{real } n * \vartheta))) \subseteq \{0..1\}$ 
  by (smt (verit) atLeastAtMost_iff frac_unique_iff image_subset_iff)
then have  $?C \subseteq \{0..1\}$ 
  by (simp add: closure_minimal)
ultimately show ?thesis by auto
qed

```

The next theorem removes the restriction $0 \leq \alpha \leq 1$.

Theorem 7.8

corollary *sequence_of_fractional_parts_is_dense*:

```

fixes  $\vartheta :: \text{real}$ 
assumes  $\vartheta \notin \mathbb{Q}$   $\varepsilon > 0$ 
obtains  $h k$  where  $k > 0$   $|\text{of\_int } k * \vartheta - \text{of\_int } h - \alpha| < \varepsilon$ 
proof -
obtain  $k$  where  $k > 0$   $|\text{frac}(\text{real } k * \vartheta) - \text{frac } \alpha| < \varepsilon$ 
  by (metis Kronecker_approx_1_explicit assms frac_ge_0 frac_lt_1 less_le_not_le)
then have  $|\text{real\_of\_int } k * \vartheta - \text{real\_of\_int } (\lfloor k * \vartheta \rfloor - \lfloor \alpha \rfloor) - \alpha| < \varepsilon$ 
  by (auto simp: frac_def)
then show thesis
  by (meson <0 < k> of_nat_0_less_iff that)
qed

```

8.3 Extension of Kronecker's Theorem to Simultaneous Approximation

8.3.1 Towards Lemma 1

lemma *integral_exp*:

```

assumes  $T \geq 0$   $a \neq 0$ 
shows  $\text{integral} \{0..T\} (\lambda t. \text{exp}(a * \text{complex\_of\_real } t)) = (\text{exp}(a * \text{of\_real } T) - 1) / a$ 
proof -
have  $\bigwedge t. t \in \{0..T\} \implies ((\lambda x. \text{exp}(a * x) / a)$  has_vector_derivative  $\text{exp}(a * t))$ 
  (at  $t$  within  $\{0..T\}$ )
  using assms
  by (intro derivative_eq_intros has_complex_derivative_imp_has_vector_derivative [unfolded o_def] | simp)+

```

then have $((\lambda t. \exp(a * \text{of_real } t)) \text{ has_integral } \exp(a * \text{complex_of_real } T)/a - \exp(a * \text{of_real } 0)/a) \{0..T\}$
by $(\text{meson fundamental_theorem_of_calculus } \langle T \geq 0 \rangle)$
then show *?thesis*
by $(\text{simp add: diff_divide_distrib integral_unique})$
qed

lemma *Kronecker_simult_aux1*:

fixes $\eta:: \text{nat} \Rightarrow \text{real}$ **and** $c:: \text{nat} \Rightarrow \text{complex}$ **and** $N:: \text{nat}$
assumes $\text{inj: inj_on } \eta \{..N\}$ **and** $k \leq N$
defines $f \equiv \lambda t. \sum r \leq N. c \ r * \exp(i * \text{of_real } t * \eta \ r)$
shows $((\lambda T. (1/T) * \text{integral } \{0..T\} (\lambda t. f \ t * \exp(-i * \text{of_real } t * \eta \ k))) \longrightarrow c \ k) \text{ at_top}$
proof –
define F **where** $F \equiv \lambda k \ t. f \ t * \exp(-i * \text{of_real } t * \eta \ k)$
have $f: F \ k = (\lambda t. \sum r \leq N. c \ r * \exp(i * (\eta \ r - \eta \ k) * \text{of_real } t))$
by $(\text{simp add: F_def f_def sum_distrib_left field_simps exp_diff exp_minus})$
have $*$: $\text{integral } \{0..T\} (F \ k)$
 $= c \ k * T + (\sum r \in \{..N\} - \{k\}. c \ r * \text{integral } \{0..T\} (\lambda t. \exp(i * (\eta \ r - \eta \ k) * \text{of_real } t)))$
if $T > 0$ **for** T
using $\langle k \leq N \rangle$ **that**
by $(\text{simp add: f integral_sum integrable_continuous_interval continuous_intros Groups_Big.sum_diff scaleR_conv_of_real})$

have $**$: $(1/T) * \text{integral } \{0..T\} (F \ k)$
 $= c \ k + (\sum r \in \{..N\} - \{k\}. c \ r * (\exp(i * (\eta \ r - \eta \ k) * \text{of_real } T) - 1) / (i * (\eta \ r - \eta \ k) * \text{of_real } T))$
if $T > 0$ **for** T
proof –
have I : $\text{integral } \{0..T\} (\lambda t. \exp(i * (\text{complex_of_real } t * \eta \ r) - i * (\text{complex_of_real } t * \eta \ k)))$
 $= (\exp(i * (\eta \ r - \eta \ k) * T) - 1) / (i * (\eta \ r - \eta \ k))$
if $r \leq N$ $r \neq k$ **for** r
using $\text{that } \langle k \leq N \rangle \text{ inj } \langle T > 0 \rangle \text{ integral_exp [of } T \ i * (\eta \ r - \eta \ k)]$
by $(\text{simp add: inj_on_eq_iff algebra_simps})$
show *?thesis*
using $\text{that by (subst *) (auto simp add: algebra_simps sum_divide_distrib } I)$
qed
have $((\lambda T. c \ r * (\exp(i * (\eta \ r - \eta \ k) * \text{of_real } T) - 1) / (i * (\eta \ r - \eta \ k) * \text{of_real } T)) \longrightarrow 0) \text{ at_top}$
for r
proof –
have $((\lambda x. (\cos((\eta \ r - \eta \ k) * x) - 1) / x) \longrightarrow 0) \text{ at_top}$
 $((\lambda x. \sin((\eta \ r - \eta \ k) * x) / x) \longrightarrow 0) \text{ at_top}$
by *real_asympt+*
hence $((\lambda T. (\exp(i * (\eta \ r - \eta \ k) * \text{of_real } T) - 1) / \text{of_real } T) \longrightarrow 0) \text{ at_top}$


```

    by (simp add: tendsto_complex_iff_Re_exp_Im_exp)
  from tendsto_mult[OF this tendsto_const[of c r / (i * (η r - η k))]] show
?thesis
    by (simp add: field_simps)
  qed
  then have ((λT. c k + (∑ r ∈ {..N} - {k}. c r * (exp(i * (η r - η k) * of_real
T) - 1) /
    (i * (η r - η k) * of_real T))) → c k + 0) at_top
    by (intro tendsto_add tendsto_null_sum) auto
  also have ?this ↔ ?thesis
  proof (rule filterlim_cong)
    show ∀ F x in at_top. c k + (∑ r ∈ {..N} - {k}. c r * (exp (i * of_real (η r -
η k) * of_real x) - 1) /
      (i * of_real (η r - η k) * of_real x)) =
      1 / of_real x * integral {0..x} (λt. f t * exp (- i * of_real t * of_real (η
k)))
    using eventually_gt_at_top[of 0]
  proof eventually_elim
    case (elim T)
    show ?case
      using **[of T] elim by (simp add: F_def)
  qed
  qed auto
  finally show ?thesis .
qed

```

the version of Lemma 1 that we actually need

lemma *Kronecker_simult_aux1_strict*:

```

  fixes η:: nat ⇒ real and c:: nat ⇒ complex and N::nat
  assumes η: inj_on η {..<N} 0 ∉ η ' {..<N} and k < N
  defines f ≡ λt. 1 + (∑ r<N. c r * exp(i * of_real t * η r))
  shows ((λT. (1/T) * integral {0..T} (λt. f t * exp(-i * of_real t * η k))) →
c k) at_top
  proof -
    define c' where c' ≡ case_nat 1 c
    define η' where η' ≡ case_nat 0 η
    define f' where f' ≡ λt. (∑ r≤N. c' r * exp(i * of_real t * η' r))
    have inj_on η' {..N}
      using η by (auto simp: η'_def inj_on_def split: nat.split_asm)
    moreover have Suc k ≤ N
      using ⟨k < N⟩ by auto
    ultimately have ((λT. 1 / T * integral {0..T} (λt. (∑ r≤N. c' r * exp (i *
of_real t * η' r)) *
      exp (- i * t * η' (Suc k)))) → c' (Suc k))
      at_top
    by (intro Kronecker_simult_aux1)
    moreover have (∑ r≤N. c' r * exp (i * of_real t * η' r)) = 1 + (∑ r<N. c r
* exp (i * of_real t * η r)) for t
      by (simp add: c'_def η'_def sum.atMost_shift)
  qed

```

ultimately show *?thesis*
 by (simp add: f_def c'_def η'_def)
 qed

8.3.2 Towards Lemma 2

lemma *cos_monotone_aux*: $\llbracket 2 * \pi * \text{of_int } i + x \rrbracket \leq y; y \leq \pi i \rrbracket \implies \cos y \leq \cos x$
 by (metis add.commute abs_ge_zero cos_abs_real cos_monotone_0_pi_le sin_cos_eq_iff)

lemma *Figure7_1*:
 assumes $0 \leq \varepsilon \leq |x| \leq \pi$
 shows $\text{cmod } (1 + \exp(i * x)) \leq \text{cmod } (1 + \exp(i * \varepsilon))$
 proof -
 have *norm_eq*: $\text{sqrt } (2 * (1 + \cos t)) = \text{cmod } (1 + \text{cis } t)$ for t
 by (simp add: norm_complex_def power2_eq_square algebra_simps)
 have $\cos |x| \leq \cos \varepsilon$
 by (rule cos_monotone_0_pi_le) (use assms in auto)
 hence $\text{sqrt } (2 * (1 + \cos |x|)) \leq \text{sqrt } (2 * (1 + \cos \varepsilon))$
 by simp
 also have $\cos |x| = \cos x$
 by simp
 finally show *?thesis*
 unfolding *norm_eq* by (simp add: cis_conv_exp)
 qed

lemma *Kronecker_simult_aux2*:
 fixes $\alpha:: \text{nat} \Rightarrow \text{real}$ and $\vartheta:: \text{nat} \Rightarrow \text{real}$ and $n:: \text{nat}$
 defines $F \equiv \lambda t. 1 + (\sum r < n. \exp(i * \text{of_real } (2 * \pi * (t * \vartheta r - \alpha r))))$
 defines $L \equiv \text{Sup } (\text{range } (\text{norm} \circ F))$
 shows $(\forall \varepsilon > 0. \exists t h. \forall r < n. |t * \vartheta r - \alpha r - \text{of_int } (h r)| < \varepsilon) \iff L = \text{Suc } n$ (is *?lhs = _*)
 proof
 assume *?lhs*
 then have $\bigwedge k. \exists t h. \forall r < n. |t * \vartheta r - \alpha r - \text{of_int } (h r)| < 1 / (2 * \pi * \text{Suc } k)$
 by simp
 then obtain $t h$ where $th: \bigwedge k r. r < n \implies |t k * \vartheta r - \alpha r - \text{of_int } (h k r)| < 1 / (2 * \pi * \text{Suc } k)$
 by metis
 have *Fle*: $\bigwedge x. \text{cmod } (F x) \leq \text{real } (\text{Suc } n)$
 by (force simp: F_def intro: order_trans [OF norm_triangle_ineq] order_trans [OF norm_sum])
 then have *boundedF*: $\text{bounded } (\text{range } F)$
 by (metis bounded_iff rangeE)
 have *leL*: $1 + n * \cos(1 / \text{Suc } k) \leq L$ for k
 proof -
 have $*$: $\cos(1 / \text{Suc } k) \leq \cos(2 * \pi * (t k * \vartheta r - \alpha r))$ if $r < n$ for r
 proof (rule cos_monotone_aux)

```

    have  $|2 * pi * (- h k r) + 2 * pi * (t k * \vartheta r - \alpha r)| \leq |t k * \vartheta r - \alpha r - h k r| * 2 * pi$ 
      by (simp add: algebra_simps abs_mult_pos abs_mult_pos')
    also have  $\dots \leq 1 / \text{real } (Suc k)$ 
      using th [OF that, of k] by (simp add: divide_simps)
    finally show  $|2 * pi * \text{real\_of\_int } (- h k r) + 2 * pi * (t k * \vartheta r - \alpha r)| \leq 1 / \text{real } (Suc k)$  .
    have  $1 / \text{real } (Suc k) \leq 1$ 
      by auto
    then show  $1 / \text{real } (Suc k) \leq pi$ 
      using pi_ge_two by linarith
  qed
  have  $1 + n * \cos(1 / Suc k) = 1 + (\sum r < n. \cos(1 / Suc k))$ 
    by simp
  also have  $\dots \leq 1 + (\sum r < n. \cos(2 * pi * (t k * \vartheta r - \alpha r)))$ 
    using * by (intro add_mono sum_mono) auto
  also have  $\dots \leq \text{Re } (F(t k))$ 
    by (simp add: F_def Re_exp)
  also have  $\dots \leq \text{norm } (F(t k))$ 
    by (simp add: complex_Re_le_cmod)
  also have  $\dots \leq L$ 
    by (simp add: L_def boundedF bounded_norm_le_SUP_norm)
  finally show ?thesis .
qed
show  $L = Suc n$ 
proof (rule antisym)
  show  $L \leq Suc n$ 
    using Fle by (auto simp add: L_def intro: cSup_least)
  have  $((\lambda k. 1 + \text{real } n * \cos(1 / \text{real } (Suc k))) \longrightarrow 1 + \text{real } n)$  at_top
    by real_asymp
  with LIMSEQ_le_const2 leL show  $Suc n \leq L$ 
    by fastforce
qed
next
assume L:  $L = Suc n$ 
show ?lhs
proof (rule ccontr)
  assume  $\neg ?lhs$ 
  then obtain  $e0$  where  $e0 > 0$  and  $e0: \bigwedge t h. \exists k < n. |t * \vartheta k - \alpha k - \text{of\_int } (h k)| \geq e0$ 
    by (force simp: not_less)
  define  $\varepsilon$  where  $\varepsilon \equiv \min e0 (pi/4)$ 
  have  $\varepsilon: \bigwedge t h. \exists k < n. |t * \vartheta k - \alpha k - \text{of\_int } (h k)| \geq \varepsilon$ 
    unfolding  $\varepsilon\_def$  using  $e0$  min.coboundedI1 by blast
  have  $\varepsilon\_bounds: \varepsilon > 0 \ \varepsilon < pi \ \varepsilon \leq pi/4$ 
    using  $\langle e0 > 0 \rangle$  by (auto simp:  $\varepsilon\_def$  min_def)
  with  $\text{sin\_gt\_zero}$  have  $\text{sin } \varepsilon > 0$ 
    by blast
  then have  $\neg \text{collinear}\{0, 1, \exp(i * \varepsilon)\}$ 

```

```

using collinear_iff_Reals cis.sel cis_conv_exp complex_is_Real_iff by force
then have norm (1 + exp (i * ε)) < 2
  using norm_triangle_eq_imp_collinear
by (smt (verit) linorder_not_le norm_exp_i_times_norm_one norm_triangle_lt)
then obtain δ where δ > 0 and δ: norm (1 + exp (i * ε)) = 2 - δ
  by (smt (verit, best))
have norm (F t) ≤ n+1-δ for t
proof -
  define h where h ≡ λr. round (t * ∅ r - α r)
  define X where X ≡ λr. t * ∅ r - α r - h r
  have exp (i * (of_int j * (of_real pi * 2))) = 1 for j
    by (metis add_0 exp_plus_2pin exp_zero)
  then have exp_X: exp (i * (2 * of_real pi * of_real (X r)))
    = exp (i * of_real (2 * pi * (t * ∅ r - α r))) for r
    by (simp add: X_def right_diff_distrib exp_add exp_diff mult.commute)
  have norm_pr_12: X r ∈ {-1/2..<1/2} for r
    apply (simp add: X_def h_def)
  by (smt (verit, best) abs_of_nonneg half_bounded_equal of_int_round_abs_le
of_int_round_gt)
  obtain k where k < n and 1: |X k| ≥ ε
    using X_def ε [of t h] by auto
  have 2: 2*pi ≥ 1
    using pi_ge_two by auto
  have |2 * pi * X k| ≥ ε
    using mult_mono [OF 2 1] pi_ge_zero by linarith
  moreover
  have |2 * pi * X k| ≤ pi
    using norm_pr_12 [of k] apply (simp add: abs_if_field_simps)
    by (smt (verit, best) divide_le_eq_1_pos divide_minus_left m2pi_less_pi
nonzero_mult_div_cancel_left)
  ultimately
  have *: norm (1 + exp (i * of_real (2 * pi * X k))) ≤ norm (1 + exp (i *
ε))
    using Figure7_1[of ε 2 * pi * X k] ε_bounds by linarith
  have norm (F t) = norm (1 + (∑ r < n. exp(i * of_real (2 * pi * (X r)))))
    by (auto simp: F_def exp_X)
  also have ... ≤ norm (1 + exp(i * of_real (2 * pi * X k))) + (∑ r ∈
{..<n}-{k}. exp(i * of_real (2 * pi * X r)))
    by (simp add: comm_monoid_add_class.sum.remove [where x=k] ⟨k < n⟩
algebra_simps)
  also have ... ≤ norm (1 + exp(i * of_real (2 * pi * X k))) + (∑ r ∈
{..<n}-{k}. norm (exp(i * of_real (2 * pi * X r))))
    by (intro norm_triangle_mono sum_norm_le order_refl)
  also have ... ≤ (2 - δ) + (n - 1)
    using ⟨k < n⟩ δ
    by simp (metis * mult_2 of_real_add of_real_mult)
  also have ... = n + 1 - δ
    using ⟨k < n⟩ by simp
  finally show ?thesis .

```

```

qed
then have  $L \leq 1 + \text{real } n - \delta$ 
  by (auto simp: L_def intro: cSup_least)
with  $L < \delta > 0$  show False
  by linarith
qed
qed

```

8.3.3 Towards lemma 3

The text doesn't say so, but the generated polynomial needs to be "clean": all coefficients nonzero, and with no exponent string mentioned more than once. And no constant terms (with all exponents zero).

Some tools for combining integer-indexed sequences or splitting them into parts

```

lemma less_sum_nat_iff:
  fixes  $b::\text{nat}$  and  $k::\text{nat}$ 
  shows  $b < (\sum_{i < k}. a\ i) \longleftrightarrow (\exists j < k. (\sum_{i < j}. a\ i) \leq b \wedge b < (\sum_{i < j}. a\ i) + a\ j)$ 
proof (induction k arbitrary: b)
  case (Suc k)
  then show ?case
    by (simp add: less_Suc_eq) (metis not_less trans_less_add1)
qed auto

```

```

lemma less_sum_nat_iff_uniq:
  fixes  $b::\text{nat}$  and  $k::\text{nat}$ 
  shows  $b < (\sum_{i < k}. a\ i) \longleftrightarrow (\exists! j. j < k \wedge (\sum_{i < j}. a\ i) \leq b \wedge b < (\sum_{i < j}. a\ i) + a\ j)$ 
  unfolding less_sum_nat_iff by (meson leD less_sum_nat_iff nat_neq_iff)

```

```

definition part :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat
  where part a k b  $\equiv$  (THE j. j < k  $\wedge$  ( $\sum_{i < j}. a\ i$ )  $\leq$  b  $\wedge$  b < ( $\sum_{i < j}. a\ i$ ) + a j)

```

```

lemma part:
   $b < (\sum_{i < k}. a\ i) \longleftrightarrow (\text{let } j = \text{part } a\ k\ b \text{ in } j < k \wedge (\sum_{i < j}. a\ i) \leq b \wedge b < (\sum_{i < j}. a\ i) + a\ j)$ 
  (is ?lhs = ?rhs)

```

```

proof
  assume ?lhs
  then show ?rhs
    unfolding less_sum_nat_iff_uniq part_def by (metis (no_types, lifting) theI')
qed (auto simp: less_sum_nat_iff Let_def)

```

```

lemma part_Suc: part a (Suc k) b = (if sum a {.. $k$ }  $\leq$  b  $\wedge$  b < sum a {.. $k$ } + a k then k else part a k b)
  using leD

```

by (fastforce simp: part_def less_Suc_eq less_sum_nat_iff conj_disj_distribR cong: conj_cong)

The polynomial expansions used in Lemma 3

definition *gpoly* :: [nat, nat \Rightarrow complex, nat, nat \Rightarrow nat, [nat,nat] \Rightarrow nat] \Rightarrow complex
 where *gpoly* n x N a r \equiv (\sum k<N. a k * (\prod i<n. x i ^ r k i))

lemma *gpoly_cong*:

assumes $\bigwedge k. k < N \implies a' k = a k \wedge \bigwedge k i. \llbracket k < N; i < n \rrbracket \implies r' k i = r k i$
 shows *gpoly* n x N a r = *gpoly* n x N a' r'
 using *assms* by (auto simp: *gpoly_def*)

lemma *gpoly_inc*: *gpoly* n x N a r = *gpoly* (Suc n) x N a ($\lambda k. (r k)(n:=0)$)
 by (simp add: *gpoly_def* algebra_simps sum_distrib_left)

lemma *monom_times_gpoly*: *gpoly* n x N a r * x n ^ q = *gpoly* (Suc n) x N a
 ($\lambda k. (r k)(n:=q)$)
 by (simp add: *gpoly_def* algebra_simps sum_distrib_left)

lemma *const_times_gpoly*: e * *gpoly* n x N a r = *gpoly* n x N ((*e o a) r)
 by (simp add: *gpoly_def* sum_distrib_left mult.assoc)

lemma *one_plus_gpoly*: 1 + *gpoly* n x N a r = *gpoly* n x (Suc N) (a(N:=1))
 (r(N:=($\lambda _.$ 0)))
 by (simp add: *gpoly_def*)

lemma *gpoly_add*:

gpoly n x N a r + *gpoly* n x N' a' r'
 = *gpoly* n x (N+N') ($\lambda k. \text{if } k < N \text{ then } a k \text{ else } a' (k-N)$) ($\lambda k. \text{if } k < N \text{ then } r k$
 else $r' (k-N)$)

proof –

have $\{..<N+N'\} = \{..<N\} \cup \{N..<N+N'\}$ $\{..<N\} \cap \{N..<N+N'\} = \{\}$
 by auto

then show ?thesis

by (simp add: *gpoly_def* sum.union_disjoint sum.atLeastLessThan_shift_0[of
 _ N] atLeast0LessThan)

qed

lemma *gpoly_sum*:

fixes n Nf af rf p

defines N \equiv sum Nf $\{..<p\}$

defines a $\equiv \lambda k. \text{let } q = (\text{part Nf p } k) \text{ in } af q (k - \text{sum Nf } \{..<q\})$

defines r $\equiv \lambda k. \text{let } q = (\text{part Nf p } k) \text{ in } rf q (k - \text{sum Nf } \{..<q\})$

shows (\sum q<p. *gpoly* n x (Nf q) (af q) (rf q)) = *gpoly* n x N a r

unfolding N_def a_def r_def

proof (induction p)

case 0

then show ?case

by (simp add: *gpoly_def*)

```

next
  case (Suc p)
  then show ?case
    by (auto simp: gpoly_add Let_def part_Suc intro: gpoly_cong)
qed

```

For excluding constant terms: with all exponents zero

```

definition nontriv_exponents :: [nat, nat, [nat,nat]⇒nat] ⇒ bool
  where nontriv_exponents n N r ≡ ∀ k<N. ∃ i<n. r k i ≠ 0

```

```

lemma nontriv_exponents_add:
  [[nontriv_exponents n M r; nontriv_exponents (Suc n) N r]] ⇒
  nontriv_exponents (Suc n) (M + N) (λk. if k<M then r k else r' (k-M))
  by (force simp add: nontriv_exponents_def less_Suc_eq)

```

```

lemma nontriv_exponents_sum:
  assumes ∧q. q < p ⇒ nontriv_exponents n (N q) (r q)
  shows nontriv_exponents n (sum N {..

```

```

definition uniq_exponents :: [nat, nat, [nat,nat]⇒nat] ⇒ bool
  where uniq_exponents n N r ≡ ∀ k<N. ∀ k'<k. ∃ i<n. r k i ≠ r k' i

```

```

lemma uniq_exponents_inj: uniq_exponents n N r ⇒ inj_on r {..

```

All coefficients must be nonzero

```

definition nonzero_coeffs :: [nat, nat⇒nat] ⇒ bool
  where nonzero_coeffs N a ≡ ∀ k<N. a k ≠ 0

```

Any polynomial expansion can be cleaned up, removing zero coeffs, etc.

```

lemma gpoly_uniq_exponents:
  obtains N' a' r'
  where ∧x. gpoly n x N a r = gpoly n x N' a' r'
    uniq_exponents n N' r' nonzero_coeffs N' a' N' ≤ N
    nontriv_exponents n N r ⇒ nontriv_exponents n N' r'
proof (cases ∀ k<N. a k = 0)
  case True
  show thesis
  proof
    show gpoly n x N a r = gpoly n x 0 a r for x
      by (auto simp: gpoly_def True)
  qed (auto simp: uniq_exponents_def nonzero_coeffs_def nontriv_exponents_def)
next
  case False
  define cut where cut f i ≡ if i<n then f i else (0::nat) for f i
  define R where R ≡ cut ' r ' ({..define N' where N' ≡ card R

```

```

define  $r'$  where  $r' \equiv \text{from\_nat\_into } R$ 
define  $a'$  where  $a' \equiv \lambda k'. \sum_{k < N}. \text{if } r' k' = \text{cut } (r k) \text{ then } a k \text{ else } 0$ 
have finite  $R$ 
  using  $R\_def$  by blast
then have  $\text{bij: } \text{bij\_betw } r' \{..<N'\} R$ 
  using  $\text{bij\_betw\_from\_nat\_into\_finite } N'\_def r'\_def$  by blast
have  $1: \text{card } \{i. i < N' \wedge r' i = \text{cut } (r k)\} = \text{Suc } 0$ 
  if  $k < N$  and  $k > 0$  for  $k$ 
proof –
  have  $\text{card } \{i. i < N' \wedge r' i = \text{cut } (r k)\} \leq \text{Suc } 0$ 
  using  $\text{bij}$  by (simp add: card_le_Suc0_iff_eq bij_betw_iff_bijections Ball_def)
metis
  moreover have  $\text{card } \{i. i < N' \wedge r' i = \text{cut } (r k)\} > 0$ 
  using  $\text{bij}$  that by (auto simp: card_gt_0_iff bij_betw_iff_bijections R_def)
  ultimately show  $\text{card } \{i. i < N' \wedge r' i = \text{cut } (r k)\} = \text{Suc } 0$ 
  using that by auto
qed
show thesis
proof
  have  $\exists i < n. r' k i \neq r' k' i$  if  $k < N'$  and  $k' < k$  for  $k k'$ 
  proof –
  have  $k' < N'$ 
    using order.strict_trans that by blast
  then have  $r' k \neq r' k'$ 
    by (metis  $\text{bij}$   $\text{bij\_betw\_iff\_bijections lessThan\_iff nat\_neq\_iff that}$ )
  moreover obtain  $sk sk'$  where  $r' k = \text{cut } sk$   $r' k' = \text{cut } sk'$ 
    by (metis  $R\_def$   $\langle k < N' \rangle \langle k' < N' \rangle \text{bij}$   $\text{bij\_betwE image\_iff lessThan\_iff}$ )
  ultimately show ?thesis
    using local.cut_def by force
qed
then show uniq_exponents  $n N' r'$ 
  by (auto simp: uniq_exponents_def R_def)
have  $R \subseteq (\text{cut } \circ r) \{..<N\}$ 
  by (auto simp: R_def)
then show  $N' \leq N$ 
  unfolding  $N'\_def$  by (metis  $\text{card\_lessThan finite\_lessThan surj\_card\_le}$ )
show  $\text{gpoly } n x N a r = \text{gpoly } n x N' a' r'$  for  $x$ 
proof –
  have  $a k * (\prod_{i < n}. x i \wedge r k i)$ 
     $= (\sum_{i < N'}. (\text{if } r' i = \text{cut } (r k) \text{ then } \text{of\_nat } (a k) \text{ else } \text{of\_nat } 0) * (\prod_{j < n.}$ 
 $x j \wedge r' i j))$ 
  if  $k < N$  for  $k$ 
  using that
  by (cases  $k = 0$ )
  (simp_all add: if_distrib [of  $\lambda v. v * \_$ ] 1 cut_def flip: sum.inter_filter
cong: if_cong)
  then show ?thesis
    by (simp add: gpoly_def a'_def sum_distrib_right sum.swap [where
 $A = \{..<N'\}$  if_distrib [of of_nat])

```



```

qed
show nontriv_exponents n N' r' if ne: nontriv_exponents n N r
proof -
  have  $\exists i < n. 0 < r' k' i$  if  $k' < N'$  for  $k'$ 
  proof -
    have  $r' k' \in R$ 
    using bij bij_betwE that by blast
    then obtain  $k$  where  $k < N$  and  $k: r' k' = \text{cut } (r k)$ 
    using R_def by blast
    with ne obtain  $i$  where  $i < n$   $r k i > 0$ 
    by (auto simp: nontriv_exponents_def)
    then show ?thesis
    using k local.cut_def by auto
  qed
  then show ?thesis
  by (simp add: nontriv_exponents_def)
qed
have  $0 < a' k'$  if  $k' < N'$  for  $k'$ 
proof -
  have  $r' k' \in R$ 
  using bij bij_betwE that by blast
  then obtain  $k$  where  $k < N$   $a k > 0$   $r' k' = \text{cut } (r k)$ 
  using R_def by blast
  then have False if  $a' k' = 0$ 
  using that by (force simp add: a'_def Ball_def )
  then show ?thesis
  by blast
qed
then show nonzero_coeffs N' a'
by (auto simp: nonzero_coeffs_def)
qed
qed

lemma Kronecker_simult_aux3:
 $\exists N a r. (\forall x. (1 + (\sum i < n. x i))^p = 1 + \text{gpoly } n x N a r) \wedge \text{Suc } N \leq (p+1)^n$ 
 $\wedge \text{nontriv\_exponents } n N r$ 
proof (induction n arbitrary: p)
  case 0
  then show ?case
  by (auto simp: gpoly_def nontriv_exponents_def nonzero_coeffs_def)
next
  case (Suc n)
  then obtain Nf af rf
  where feq:  $\bigwedge q x. (1 + (\sum i < n. x i))^q = 1 + \text{gpoly } n x (Nf q) (af q) (rf q)$ 
  and Nle:  $\bigwedge q. \text{Suc } (Nf q) \leq (q+1)^n$ 
  and nontriv:  $\bigwedge q. \text{nontriv\_exponents } n (Nf q) (rf q)$ 
  by metis
  define N where  $N \equiv Nf p + (\sum q < p. \text{Suc } (Nf q))$ 

```

```

define a where a  $\equiv \lambda k$ . if k < Nf p then af p k
      else let q = part ( $\lambda t$ . Suc (Nf t)) p (k - Nf p)
      in (p choose q) *
      (if k - Nf p - ( $\sum t < q$ . Suc (Nf t)) = Nf q then Suc 0
      else af q (k - Nf p - ( $\sum t < q$ . Suc (Nf t))))
define r where r  $\equiv \lambda k$ . if k < Nf p then (rf p k)(n := 0)
      else let q = part ( $\lambda t$ . Suc (Nf t)) p (k - Nf p)
      in (if k - Nf p - ( $\sum t < q$ . Suc (Nf t)) = Nf q
      then  $\lambda$ _. 0
      else rf q (k - Nf p - ( $\sum t < q$ . Suc (Nf t))))
then  $\lambda$ _. 0
(n := p - q)
have peq: {..p} = insert p {..<p}
by auto
have nontriv_exponents n (Nf p) ( $\lambda i$ . (rf p i)(n := 0))
   $\wedge q$ . q < p  $\implies$  nontriv_exponents (Suc n) (Suc (Nf q)) ( $\lambda k$ . (if k = Nf q then
 $\lambda$ _. 0 else rf q k) (n := p - q))
using nontriv by (fastforce simp: nontriv_exponents_def)+
then have nontriv_exponents (Suc n) N r
unfolding N_def r_def by (intro nontriv_exponents_add nontriv_exponents_sum)
moreover
{ fix x :: nat  $\implies$  complex
  have (1 + ( $\sum i < \text{Suc } n$ . x i)) ^ p = (1 + ( $\sum i < n$ . x i) + x n) ^ p
    by (simp add: add_ac)
  also have ... = ( $\sum q \leq p$ . (p choose q) * (1 + ( $\sum i < n$ . x i)) ^ q * x n ^ (p - q))
    by (simp add: binomial_ring)
  also have ... = ( $\sum q \leq p$ . (p choose q) * (1 + gpoly n x (Nf q) (af q) (rf q)) *
x n ^ (p - q))
    by (simp add: feq)
  also have ... = 1 + (gpoly n x (Nf p) (af p) (rf p) + ( $\sum q < p$ . (p choose q) *
(1 + gpoly n x (Nf q) (af q) (rf q)) * x n ^ (p - q)))
    by (simp add: algebra_simps sum.distrib peq)
  also have ... = 1 + gpoly (Suc n) x N a r
    apply (subst one_plus_gpoly)
    apply (simp add: const_times_gpoly monom_times_gpoly gpoly_sum)
    apply (simp add: gpoly_inc [where n=n] gpoly_add N_def a_def r_def)
    done
  finally have (1 + sum x {..<Suc n}) ^ p = 1 + gpoly (Suc n) x N a r .
}
moreover have Suc N  $\leq$  (p + 1) ^ Suc n
proof -
have Suc N = ( $\sum q \leq p$ . Suc (Nf q))
  by (simp add: N_def peq)
also have ...  $\leq$  ( $\sum q \leq p$ . (q+1) ^ n)
  by (meson Nle sum_mono)
also have ...  $\leq$  ( $\sum q \leq p$ . (p+1) ^ n)
  by (force intro!: sum_mono power_mono)
also have ...  $\leq$  (p + 1) ^ Suc n
  by simp
finally show Suc N  $\leq$  (p + 1) ^ Suc n .

```

```

qed
ultimately show ?case
  by blast
qed

```

lemma *Kronecker_simult_aux3_uniq_exponents*:

obtains $N a r$ **where** $\bigwedge x. (1 + (\sum_{i < n. x i})^p = 1 + \text{gpoly } n x N a r \text{ Suc } N \leq (p+1)^{\wedge n}$

nontriv_exponents n N r uniq_exponents n N r nonzero_coeffs

$N a$

proof –

obtain $N0 a0 r0$ **where** $\bigwedge x. (1 + (\sum_{r < n. x r})^p = 1 + \text{gpoly } n x N0 a0 r0$

and $\text{Suc } N0 \leq (p+1)^{\wedge n}$ *nontriv_exponents n N0 r0*

using *Kronecker_simult_aux3* **by** *blast*

with *le_trans Suc_le_mono gpoly_uniq_exponents [of n N0 a0 r0]* **that show**
thesis

by (*metis (no_types, lifting)*)

qed

8.3.4 And finally Kroncker's theorem itself

Statement of Theorem 7.9

theorem *Kronecker_thm_1*:

fixes $\alpha \vartheta :: \text{nat} \Rightarrow \text{real}$ **and** $n :: \text{nat}$

assumes *indp: module.independent* $(\lambda r. (*) (\text{real_of_int } r)) (\vartheta \text{ ' } \{..<n\})$

and *inj* $\vartheta: \text{inj_on } \vartheta \{..<n\}$ **and** $\varepsilon > 0$

obtains $t h$ **where** $\bigwedge i. i < n \implies |t * \vartheta i - \text{of_int } (h i) - \alpha i| < \varepsilon$

proof (*cases n > 0*)

case *False* **then show** *?thesis*

using *that* **by** *blast*

next

case *True*

interpret *Modules.module* $(\lambda r. (*) (\text{real_of_int } r))$

by (*simp add: Modules.module.intro distrib_left mult.commute*)

define F **where** $F \equiv \lambda t. 1 + (\sum_{i < n. \text{exp}(i * \text{of_real } (2 * \text{pi} * (t * \vartheta i - \alpha i)))$

define L **where** $L \equiv \text{Sup } (\text{range } (\text{norm} \circ F))$

have [*continuous_intros*]: $0 < T \implies \text{continuous_on } \{0..T\} F$ **for** T

unfolding *F_def* **by** (*intro continuous_intros*)

have *nft_Sucn*: $\text{norm } (F t) \leq \text{Suc } n$ **for** t

unfolding *F_def* **by** (*rule norm_triangle_le order_trans [OF norm_sum] |*

simp)+

then have $L_le: L \leq \text{Suc } n$

by (*simp add: L_def cSUP_least*)

have *nft_L*: $\text{norm } (F t) \leq L$ **for** t

by (*metis L_def UNIV_I bdd_aboveI2 cSUP_upper nft_Sucn o_apply*)

have $L = \text{Suc } n$

proof –

{ **fix** $p :: \text{nat}$

```

assume  $p > 0$ 
obtain  $N a r$  where  $\exists: \bigwedge x. (1 + (\sum_{r < n}. x r)) ^ p = 1 + \text{gpoly } n x N a r$ 
and  $\text{Suc} N: \text{Suc } N \leq (p+1) ^ n$ 
and  $\text{nontriv}: \text{nontriv\_exponents } n N r$  and  $\text{uniq}: \text{uniq\_exponents } n N r$ 
and  $\text{apos}: \text{nonzero\_coeffs } N a$ 
using  $\text{Kronecker\_simult\_aux3\_uniq\_exponents}$  by  $\text{blast}$ 
have  $N \neq 0$ 
proof
assume  $N = 0$ 
have  $2 ^ p = (1::\text{complex})$ 
using  $\exists$   $[\text{of } (\lambda_. 0)(0:=1)] \text{True } \langle p > 0 \rangle \langle N = 0 \rangle$  by  $(\text{simp add: gpoly\_def})$ 
then have  $2 ^ p = \text{Suc } 0$ 
by  $(\text{metis of\_nat\_1 One\_nat\_def of\_nat\_eq iff of\_nat\_numeral of\_nat\_power})$ 
with  $\langle 0 < p \rangle$  show  $\text{False}$  by force
qed
define  $x$  where  $x \equiv \lambda t r. \exp(i * \text{of\_real } (2 * \pi i * (t * \vartheta r - \alpha r)))$ 
define  $f$  where  $f \equiv \lambda t. (F t) ^ p$ 
have  $\text{feq}: f t = 1 + \text{gpoly } n (x t) N a r$  for  $t$ 
unfolding  $f\_def F\_def x\_def$  by  $(\text{simp flip: } \exists)$ 
define  $c$  where  $c \equiv \lambda k. a k / \text{cis } (\sum_{i < n}. (\pi i * (2 * (\alpha i * \text{real } (r k i))))))$ 
define  $\eta$  where  $\eta \equiv \lambda k. 2 * \pi i * (\sum_{i < n}. r k i * \vartheta i)$ 
define  $\text{INTT}$  where  $\text{INTT} \equiv \lambda k T. (1/T) * \text{integral } \{0..T\} (\lambda t. f t * \exp(-i * \text{of\_real } t * \eta k))$ 
have  $a k * (\prod_{i < n}. x t i ^ r k i) = c k * \exp(i * t * \eta k)$  if  $k < N$  for  $k t$ 
apply  $(\text{simp add: } x\_def \eta\_def \text{sum\_distrib\_left flip: exp\_of\_nat\_mult exp\_sum})$ 
apply  $(\text{simp add: algebra\_simps sum\_subtractf exp\_diff c\_def sum\_distrib\_left cis\_conv\_exp})$ 
done
then have  $f\text{def}: f t = 1 + (\sum_{k < N}. c k * \exp(i * \text{of\_real } t * \eta k))$  for  $t$ 
by  $(\text{simp add: feq gpoly\_def})$ 
have  $\text{nzero}: \vartheta i \neq 0$  if  $i < n$  for  $i$ 
using  $\text{indp}$  that  $\text{local.dependent\_zero}$  by force
have  $\text{ind\_disj}: \bigwedge u. (\forall x < n. u (\vartheta x) = 0) \vee (\sum v \in \vartheta \{..<n\}. \text{of\_int } (u v) * v) \neq 0$ 
using  $\text{indp}$  by  $(\text{auto simp: dependent\_finite})$ 
have  $\text{inj}\eta: \text{inj\_on } \eta \{..<N\}$ 
proof
fix  $k k'$ 
assume  $k: k \in \{..<N\} k' \in \{..<N\}$  and  $\eta k = \eta k'$ 
then have  $\text{eq}: (\sum_{i < n}. \text{real } (r k i) * \vartheta i) = (\sum_{i < n}. \text{real } (r k' i) * \vartheta i)$ 
by  $(\text{auto simp: } \eta\_def)$ 
define  $f$  where  $f \equiv \lambda z. \text{let } i = \text{inv\_into } \{..<n\} \vartheta z \text{ in } \text{int } (r k i) - \text{int } (r k' i)$ 
show  $k = k'$ 
using  $\text{ind\_disj}$   $[\text{of } f] \text{inj}\vartheta \text{uniq eq } k$ 
apply  $(\text{simp add: } f\_def \text{Let\_def sum.reindex sum\_subtractf algebra\_simps uniq\_exponents\_def})$ 
by  $(\text{metis not\_less\_iff\_gr\_or\_eq})$ 

```

```

qed
moreover have  $0 \notin \eta \text{ ` } \{..<N\}$ 
  unfolding  $\eta\_def$ 
proof clarsimp
  fix  $k$ 
  assume *:  $(\sum_{i<n} \text{real } (r\ k\ i) * \vartheta\ i) = 0 \ k < N$ 
  define  $f$  where  $f \equiv \lambda z. \text{let } i = \text{inv\_into } \{..<n\} \ \vartheta\ z \text{ in } \text{int } (r\ k\ i)$ 
  obtain  $i$  where  $i < n$  and  $r\ k\ i > 0$ 
  by (meson  $\langle k < N \rangle$  nontriv nontriv_exponents_def zero_less_iff_neq_zero)
  with * nzero show False
  using  $\text{ind\_disj } [\text{of } f] \ \text{inj}\vartheta$  by (simp add:  $f\_def$  sum.reindex)
qed
ultimately have 15:  $(\text{INTT } k \longrightarrow c\ k) \text{ at\_top if } k < N \text{ for } k$ 
  unfolding  $f\_def$   $\text{INTT\_def}$  using  $\text{Kronecker\_simult\_aux1\_strict}$  that by
presburger
have  $\text{norm\_c}: \text{norm } (c\ k) \leq L^{\wedge} p$  if  $k < N$  for  $k$ 
proof (intro tendsto_le [of  $\_ \lambda \_ . L^{\wedge} p$ ])
  show  $((\text{norm} \circ \text{INTT } k) \longrightarrow \text{cmod } (c\ k)) \text{ at\_top}$ 
  using that 15 by (simp add:  $o\_def$  tendsto_norm)
  have  $\text{norm } (\text{INTT } k\ T) \leq L^{\wedge} p$  if  $T \geq 0$  for  $T::\text{real}$ 
  proof -
    have  $0 \leq L^{\wedge} p$ 
    by (meson  $\text{nft\_L } \text{norm\_ge\_zero } \text{order\_trans } \text{zero\_le\_power}$ )
    have  $\text{norm } (\text{integral } \{0..T\} (\lambda t. f\ t * \exp (- (i * t * \eta\ k))))$ 
       $\leq \text{integral } \{0..T\} (\lambda t. L^{\wedge} p)$  (is ?L  $\leq$  ?R) if  $T > 0$ 
    proof -
      have  $?L \leq \text{integral } \{0..T\} (\lambda t. \text{norm } (f\ t * \exp (- (i * t * \eta\ k))))$ 
      unfolding  $f\_def$  by (intro continuous_on_imp_absolutely_integrable_on
continuous_intros that)
      also have  $\dots \leq ?R$ 
      unfolding  $f\_def$ 
      by (intro  $\text{integral\_le } \text{continuous\_intros } \text{integrable\_continuous\_interval}$ 
that
| simp add:  $\text{nft\_L } \text{norm\_mult } \text{norm\_power } \text{power\_mono}$ ) +
      finally show ?thesis .
    qed
  with  $\langle T \geq 0 \rangle$  have  $\text{norm } (\text{INTT } k\ T) \leq (1/T) * \text{integral } \{0..T\} (\lambda t. L^{\wedge} p)$ 
  by (auto simp add:  $\text{INTT\_def } \text{norm\_divide } \text{divide\_simps } \text{simp } \text{del}:$ 
 $\text{integral\_const\_real}$ )
  also have  $\dots \leq L^{\wedge} p$ 
  using  $\langle T \geq 0 \rangle \langle 0 \leq L^{\wedge} p \rangle$  by simp
  finally show ?thesis .
qed
then show  $\forall_F x \text{ in } \text{at\_top}. (\text{norm} \circ \text{INTT } k) x \leq L^{\wedge} p$ 
  using eventually_at_top_linorder that by fastforce
qed auto
then have  $(\sum_{k < N} \text{cmod } (c\ k)) \leq N * L^{\wedge} p$ 
  by (metis  $\text{sum\_bounded\_above } \text{card\_lessThan } \text{lessThan\_iff}$ )

```

```

moreover
have  $Re((1 + (\sum r < n. 1)) ^ p) = Re(1 + gpoly\ n\ (\lambda_. 1)\ N\ a\ r)$ 
  using  $\mathcal{B}$  [of  $\lambda_. 1$ ] by metis
then have  $14: 1 + (\sum k < N. norm\ (c\ k)) = (1 + real\ n) ^ p$ 
  by (simp add: c_def norm_divide gpoly_def)
moreover
have  $L ^ p \geq 1$ 
  using norm_c [of 0]  $\langle N \neq 0 \rangle$  apos
  by (force simp add: c_def norm_divide nonzero_coeffs_def)
ultimately have  $*$ :  $(1 + real\ n) ^ p \leq Suc\ N * L ^ p$ 
  by (simp add: algebra_simps)
have [simp]:  $L > 0$ 
  using  $\langle 1 \leq L ^ p \rangle \langle 0 < p \rangle$  by (smt (verit, best) nft_L norm_ge_zero
power_eq_0_iff)
  have  $Suc\ n ^ p \leq (p+1) ^ n * L ^ p$ 
  by (smt (verit, best)  $*$  mult.commute  $\langle 1 \leq L ^ p \rangle$  SucN mult_left_mono
of_nat_1 of_nat_add of_nat_power_eq_of_nat_cancel_iff of_nat_power_le_of_nat_cancel_iff
plus_1_eq_Suc)
  then have  $(Suc\ n ^ p)\ powr\ (1/p) \leq ((p+1) ^ n * L ^ p)\ powr\ (1/p)$ 
  by (simp add: powr_mono2)
  then have  $(Suc\ n) \leq ((p+1) ^ n)\ powr\ (1/p) * L$ 
  using  $\langle p > 0 \rangle \langle 0 < L \rangle$  by (simp add: powr_powr powr_mult flip:
powr_realpow)
  also have  $\dots = (p+1)\ powr\ n)\ powr\ (1/p) * L$ 
  by (simp add: powr_realpow)
  also have  $\dots = (p+1)\ powr\ (n/p) * L$ 
  by (simp add: powr_powr)
  finally have  $(n + 1) / L \leq (p+1)\ powr\ (n/p)$ 
  by (simp add: divide_simps)
  then have  $\ln\ ((n + 1) / L) \leq \ln\ (real\ (p + 1)\ powr\ (real\ n / real\ p))$ 
  by simp
  also have  $\dots \leq (n/p) * \ln(p+1)$ 
  by (simp add: powr_def)
  finally have  $\ln\ ((n + 1) / L) \leq (n/p) * \ln(p+1) \wedge L > 0$ 
  by fastforce
} note  $*$  = this
then have [simp]:  $L > 0$ 
  by blast
have  $0: (\lambda p. (n/p) * \ln(p+1)) \longrightarrow 0$ 
  by real_asymp
have  $\ln\ (real\ (n + 1) / L) \leq 0$ 
  using  $*$  eventually_at_top_dense by (intro tendsto_lowerbound [OF 0]) auto
then have  $n+1 \leq L$ 
  by (simp add: ln_div)
then show ?thesis
  using  $L\_le$  by linarith
qed
with Kronecker_simult_aux2 [of  $n\ \vartheta\ \alpha$ ]  $\langle \varepsilon > 0 \rangle$  that show thesis
  by (auto simp: F_def L_def add.commute diff_diff_eq)

```

qed

Theorem 7.10

corollary *Kronecker_thm_2*:

```

fixes  $\alpha \ \vartheta :: \text{nat} \Rightarrow \text{real}$  and  $n :: \text{nat}$ 
assumes indp: module.independent ( $\lambda r x. \text{of\_int } r * x$ ) ( $\vartheta \text{ ' } \{..n\}$ )
and inj $\vartheta$ : inj_on  $\vartheta \ \{..n\}$  and [simp]:  $\vartheta \ n = 1$  and  $\varepsilon > 0$ 
obtains  $k \ m$  where  $\bigwedge i. i < n \Longrightarrow |\text{of\_int } k * \vartheta \ i - \text{of\_int } (m \ i) - \alpha \ i| < \varepsilon$ 
proof -
interpret Modules.module ( $\lambda r. (*)$ ) (real_of_int  $r$ )
by (simp add: Modules.module.intro distrib_left mult.commute)
have one_in $\vartheta$ :  $1 \in \vartheta \text{ ' } \{..n\}$ 
unfolding  $\langle \vartheta \ n = 1 \rangle$  [symmetric] by blast

have not_in_Ints:  $\vartheta \ i \notin \mathbb{Z}$  if  $i < n$  for  $i$ 
proof
assume  $\vartheta \ i \in \mathbb{Z}$ 
then obtain  $m$  where  $m: \vartheta \ i = \text{of\_int } m$ 
by (auto elim!: Ints_cases)
have not_one:  $\vartheta \ i \neq 1$ 
using inj_onD[OF inj $\vartheta$ , of i n]  $i$  by auto
define  $u :: \text{real} \Rightarrow \text{int}$  where  $u = (\lambda \_. 0)(\vartheta \ i := 1, 1 := -m)$ 
show False
using independentD[OF indp, of  $\vartheta \text{ ' } \{i, n\}$   $u \ \vartheta \ i$ ]  $\langle i < n \rangle$  not_one one_in $\vartheta$ 
by (auto simp: u_def simp flip: m)
qed

```

```

have inj $\vartheta'$ : inj_on (frac  $\circ \vartheta$ )  $\{..n\}$ 
proof (rule linorder_inj_onI')
fix  $i \ j$  assume ij:  $i \in \{..n\} \ j \in \{..n\} \ i < j$ 
show (frac  $\circ \vartheta$ )  $i \neq$  (frac  $\circ \vartheta$ )  $j$ 
proof (cases j = n)
case True
thus ?thesis
using not_in_Ints[of i]  $ij$  by auto
next
case False
hence  $j < n$ 
using  $ij$  by auto
have inj_on  $\vartheta$  (set  $[i, j, n]$ )
using inj $\vartheta$  by (rule inj_on_subset) (use ij in auto)
moreover have distinct  $[i, j, n]$ 
using  $\langle j < n \rangle$   $ij$  by auto
ultimately have distinct (map  $\vartheta$   $[i, j, n]$ )
unfolding distinct_map by blast
hence distinct: distinct  $[\vartheta \ i, \vartheta \ j, 1]$ 
by simp

show (frac  $\circ \vartheta$ )  $i \neq$  (frac  $\circ \vartheta$ )  $j$ 

```

```

proof
  assume eq: (frac ∘ ϑ) i = (frac ∘ ϑ) j
  define u :: real ⇒ int where u = (λ_. 0)(ϑ i := 1, ϑ j := -1, 1 := ⌊ϑ j⌋
- ⌊ϑ i⌋)
  have (∑ v∈{ϑ i, ϑ j, 1}. real_of_int (u v) * v) = frac (ϑ i) - frac (ϑ j)
    using distinct by (simp add: u_def frac_def)
  also have ... = 0
    using eq by simp
  finally have eq0: (∑ v∈{ϑ i, ϑ j, 1}. real_of_int (u v) * v) = 0 .
  show False
    using independentD[OF indep __ eq0, of ϑ i] one_in_ϑ ij distinct
    by (auto simp: u_def)
qed
qed
qed

have frac (ϑ n) = 0
  by auto
then have ϑno_int: of_int r ∉ ϑ ‘ {..<n} for r
  using injϑ' frac_of_int
  apply (simp add: inj_on_def Ball_def)
  by (metis ⟨frac (ϑ n) = 0⟩ frac_of_int imageE le_less lessThan_iff less_irrefl)
define ϑ' where ϑ' ≡ (frac ∘ ϑ)(n:=1)
have [simp]: {..<Suc n} ∩ {x. x ≠ n} = {..<n}
  by auto
have ϑimage[simp]: ϑ ‘ {..n} = insert 1 (ϑ ‘ {..<n})
  using lessThan_Suc lessThan_Suc_atMost by force
have module.independent (λr. (*) (of_int r)) (ϑ' ‘ {..<Suc n})
  unfolding dependent_explicit ϑ'_def
proof clarsimp
  fix T u v
  assume T: T ⊆ insert 1 ((λi. frac (ϑ i)) ‘ {..<n})
  and finite T
  and uv_eq0: (∑ v∈T. of_int (u v) * v) = 0
  and v ∈ T
  define invf where invf ≡ inv_into {..<n} (frac ∘ ϑ)
  have inj_on (λx. frac (ϑ x)) {..<n}
  using injϑ' by (auto simp: inj_on_def)
  then have invf [simp]: invf (frac (ϑ i)) = i if i < n for i
  using frac_lt_1 [of ϑ i] that by (auto simp: invf_def o_def inv_into_f_eq
[where x=i])
  define N where N ≡ invf ‘ (T - {1})
  have Nsub: N ⊆ {..n} and finite N
  using T ⟨finite T⟩ by (auto simp: N_def subset_iff)
  have inj_invf: inj_on invf (T - {1})
  using ϑno_int [of 1] ⟨ϑ n = 1⟩ inv_into_injective T
  by (fastforce simp: inj_on_def invf_def)
  have invf_iff: invf t = i ↔ (i < n ∧ t = frac (ϑ i)) if t ∈ T - {1} for i t
  using that T by auto

```



```

have sumN:  $(\sum i \in N. f i) = (\sum x \in T - \{1\}. f (invf x))$  for  $f :: nat \Rightarrow int$ 
  using inj_invf T by (simp add: N_def sum.reindex)
define T' where  $T' \equiv insert 1 (\vartheta ' N)$ 
have [simp]: finite T'  $1 \in T'$ 
  using T'_def N_def ⟨finite T⟩ by auto
have T'_sub:  $T' \subseteq \vartheta ' \{..n\}$ 
  using Nsub T'_def  $\vartheta image$  by blast
have in_N_not1:  $x \in N \implies \vartheta x \neq 1$  for  $x$ 
  using  $\vartheta no\_int [of 1]$  by (metis N_def image_iff invf_iff lessThan_iff
of_int_1)
define u' where  $u' = (u \circ frac)(1 := (if 1 \in T then u 1 else 0) + (\sum i \in N. - \lfloor \vartheta i \rfloor * u (frac (\vartheta i))))$ 
have  $(\sum v \in T'. real\_of\_int (u' v) * v) = u' 1 + (\sum v \in \vartheta ' N. real\_of\_int (u' v) * v)$ 
  using ⟨finite N⟩ by (simp add: T'_def image_iff in_N_not1)
also have ... =  $u' 1 + sum ((\lambda v. real\_of\_int (u' v) * v) \circ \vartheta) N$ 
  by (smt (verit) N_def ⟨finite N⟩ image_iff invf_iff sum.reindex_nontrivial)
also have ... =  $u' 1 + (\sum i \in N. of\_int ((u \circ frac) (\vartheta i)) * \vartheta i)$ 
  by (auto simp add: u'_def in_N_not1)
also have ... =  $u' 1 + (\sum i \in N. of\_int ((u \circ frac) (\vartheta i)) * (floor (\vartheta i) + frac(\vartheta i)))$ 
  by (simp add: frac_def cong: if_cong)
also have ... =  $(\sum v \in T. of\_int (u v) * v)$ 
proof (cases  $1 \in T$ )
case True
  then have T1:  $(\sum v \in T. real\_of\_int (u v) * v) = u 1 + (\sum v \in T - \{1\}. real\_of\_int (u v) * v)$ 
    by (simp add: ⟨finite T⟩ sum.remove)
  show ?thesis
    using inj_invf True T unfolding N_def u'_def
    by (auto simp: add.assoc distrib_left sum.reindex T1 simp flip: sum.distrib intro!: sum.cong)
next
case False
  then show ?thesis
    using inj_invf T unfolding N_def u'_def
    by (auto simp: algebra_simps sum.reindex simp flip: sum.distrib intro!: sum.cong)
qed
also have ... = 0
  using uv_eq0 by blast
finally have 0:  $(\sum v \in T'. real\_of\_int (u' v) * v) = 0$  .
have  $u v = 0$  if  $T'0: \bigwedge v. v \in T' \implies u' v = 0$ 
proof -
  have [simp]:  $u t = 0$  if  $t \in T - \{1\}$  for  $t$ 
  proof -
    have  $\vartheta (invf t) \in T'$ 
      using N_def T'_def that by blast
    then show ?thesis

```

```

    using that T'0 [of  $\vartheta$  (invf t)]
    by (smt (verit, best) in_N_not1 N_def fun_upd_other imageI invf_iff
o_apply u'_def)
  qed
  show ?thesis
  proof (cases v = 1)
    case True
    then have 1  $\in$  T
    using  $\langle v \in T \rangle$  by blast
    have  $(\sum v \in T. \text{real\_of\_int } (u v) * v) = u 1 + (\sum v \in T - \{1\}. \text{real\_of\_int } (u v) * v)$ 
    using True  $\langle \text{finite } T \rangle$   $\langle v \in T \rangle$  mk_disjoint_insert by fastforce
    then have 0 = u 1
    using uv_eq0 by auto
    with True show ?thesis by presburger
  next
    case False
    then have  $\vartheta$  (invf v)  $\in$   $\vartheta$  ' N
    using N_def  $\langle v \in T \rangle$  by blast
    then show ?thesis
    using that [of  $\vartheta$  (invf v)] False  $\langle v \in T \rangle$  T by (force simp: T'_def
in_N_not1 u'_def)
  qed
  qed
  with indep T'_sub  $\langle \text{finite } T' \rangle$  0 show u v = 0
  unfolding dependent_explicit by blast
  qed
  moreover have inj_on  $\vartheta'$  {.. $\text{Suc } n$ }
  using inj $\vartheta'$ 
  unfolding  $\vartheta'$ _def inj_on_def
  by (metis comp_def frac_lt_1 fun_upd_other fun_upd_same lessThan_Suc_atMost
less_irrefl)
  ultimately obtain t h where th:  $\bigwedge i. i < \text{Suc } n \implies |t * \vartheta' i - \text{of\_int } (h i) - (\alpha(n:=0)) i| < \varepsilon/2$ 
  using Kronecker_thm_1 [of  $\vartheta'$   $\text{Suc } n \varepsilon/2$ ] lessThan_Suc_atMost assms using
half_gt_zero by blast
  define k where k = h n
  define m where m  $\equiv \lambda i. h i + k * \lfloor \vartheta i \rfloor$ 
  show thesis
  proof
    fix i assume i < n
    have  $|k * \text{frac } (\vartheta i) - h i - \alpha i| < \varepsilon$ 
    proof -
      have  $|k * \text{frac } (\vartheta i) - h i - \alpha i| = |t * \text{frac } (\vartheta i) - h i - \alpha i + (k-t) * \text{frac } (\vartheta i)|$ 
      by (simp add: algebra_simps)
      also have ...  $\leq |t * \text{frac } (\vartheta i) - h i - \alpha i| + |(k-t) * \text{frac } (\vartheta i)|$ 
      by linarith
      also have ...  $\leq |t * \text{frac } (\vartheta i) - h i - \alpha i| + |k-t|$ 

```

```

    using frac_lt_1 [of  $\vartheta$   $i$ ] le_less by (fastforce simp add: abs_mult)
  also have ... <  $\varepsilon$ 
    using th[of  $i$ ] th[of  $n$ ] < $i < n$ >
    by (simp add: k_def  $\vartheta'$ _def) (smt (verit, best))
  finally show ?thesis .
qed
then show  $|k * \vartheta i - m i - \alpha i| < \varepsilon$ 
  by (simp add: algebra_simps frac_def m_def)
qed
qed
end

```

8.4 Bernstein-Weierstrass and Stone-Weierstrass

By L C Paulson (2015)

```

theory Weierstrass_Theorems
imports Uniform_Limit Path_Connected Derivative
begin

```

8.4.1 Bernstein polynomials

```

definition Bernstein :: [nat,nat,real]  $\Rightarrow$  real where
  Bernstein  $n$   $k$   $x$   $\equiv$  of_nat (n choose k) *  $x^k$  * (1 - x) $^{n-k}$ 

```

```

lemma Bernstein_nonneg:  $[0 \leq x; x \leq 1] \Longrightarrow 0 \leq$  Bernstein  $n$   $k$   $x$ 
  by (simp add: Bernstein_def)

```

```

lemma Bernstein_pos:  $[0 < x; x < 1; k \leq n] \Longrightarrow 0 <$  Bernstein  $n$   $k$   $x$ 
  by (simp add: Bernstein_def)

```

```

lemma sum_Bernstein [simp]:  $(\sum_{k \leq n} \text{Bernstein } n \ k \ x) = 1$ 
  using binomial_ring [of  $x$   $1-x$   $n$ ]
  by (simp add: Bernstein_def)

```

```

lemma binomial_deriv1:
   $(\sum_{k \leq n} (\text{of\_nat } k * \text{of\_nat } (n \ \text{choose } k)) * a^{k-1} * b^{n-k}) = \text{real\_of\_nat}$ 
 $n * (a+b)^{n-1}$ 
  apply (rule DERIV_unique [where  $f = \lambda a. (a+b)^n$  and  $x=a$ ])
  apply (subst binomial_ring)
  apply (rule derivative_eq_intros sum.cong | simp add: atMost_atLeast0)+
  done

```

```

lemma binomial_deriv2:
   $(\sum_{k \leq n} (\text{of\_nat } k * \text{of\_nat } (k-1) * \text{of\_nat } (n \ \text{choose } k)) * a^{k-2} * b^{n-k}) =$ 

```

2752

```

    of_nat n * of_nat (n-1) * (a+b::real)^(n-2)
  apply (rule DERIV_unique [where f = λa. of_nat n * (a+b::real)^(n-1) and
x=a])
  apply (subst binomial_deriv1 [symmetric])
  apply (rule derivative_eq_intros sum.cong | simp add: Num.numeral_2_eq_2)+
done

```

```

lemma sum_k_Bernstein [simp]: (∑ k≤n. real k * Bernstein n k x) = of_nat n
* x
  apply (subst binomial_deriv1 [of n x 1-x, simplified, symmetric])
  apply (simp add: sum_distrib_right)
  apply (auto simp: Bernstein_def algebra_simps power_eq_if intro!: sum.cong)
done

```

```

lemma sum_kk_Bernstein [simp]: (∑ k≤n. real k * (real k - 1) * Bernstein n k
x) = real n * (real n - 1) * x^2

```

```

proof -
  have (∑ k≤n. real k * (real k - 1) * Bernstein n k x) =
    (∑ k≤n. real k * real (k - Suc 0) * real (n choose k) * x^(k-2) * (1 -
x)^(n-k) * x^2)
  proof (rule sum.cong [OF refl], simp)
    fix k
    assume k ≤ n
    then consider k = 0 | k = 1 | k' where k = Suc (Suc k')
    by (metis One_nat_def not0_implies_Suc)
    then show k = 0 ∨
      (real k - 1) * Bernstein n k x =
        real (k - Suc 0) *
          (real (n choose k) * (x^(k-2) * ((1-x)^(n-k) * x^2)))
    by cases (auto simp add: Bernstein_def power2_eq_square algebra_simps)
  qed
  also have ... = real_of_nat n * real_of_nat (n - Suc 0) * x^2
    by (subst binomial_deriv2 [of n x 1-x, simplified, symmetric]) (simp add:
sum_distrib_right)
  also have ... = n * (n - 1) * x^2
    by auto
  finally show ?thesis
    by auto
qed

```

8.4.2 Explicit Bernstein version of the 1D Weierstrass approximation theorem

theorem *Bernstein_Weierstrass*:

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes *contf*: *continuous_on* {0..1} f **and** $e: 0 < e$

shows $\exists N. \forall n x. N \leq n \wedge x \in \{0..1\}$

$\longrightarrow |f x - (\sum k \leq n. f(k/n) * \text{Bernstein } n \text{ } k \text{ } x)| < e$

proof -

```

have bounded (f ' {0..1})
  using compact_continuous_image compact_imp_bounded contf by blast
then obtain M where M:  $\bigwedge x. 0 \leq x \implies x \leq 1 \implies |f x| \leq M$ 
  by (force simp add: bounded_iff)
then have 0 ≤ M by force
have ucontf: uniformly_continuous_on {0..1} f
  using compact_uniformly_continuous_contf by blast
then obtain d where d:  $d > 0 \bigwedge x x'. \llbracket x \in \{0..1\}; x' \in \{0..1\}; |x' - x| < d \rrbracket$ 
 $\implies |f x' - f x| < e/2$ 
  apply (rule uniformly_continuous_onE [where e = e/2])
  using e by (auto simp: dist_norm)
{ fix n::nat and x::real
  assume n:  $\text{Suc}(\text{nat}\lceil 4 * M / (e * d^2) \rceil) \leq n$  and x:  $0 \leq x \leq 1$ 
  have 0 < n using n by simp
  have ed0:  $-(e * d^2) < 0$ 
    using e ⟨0 < d⟩ by simp
  also have ... ≤ M * 4
    using ⟨0 ≤ M⟩ by simp
  finally have [simp]:  $\text{real\_of\_int}(\text{nat}\lceil 4 * M / (e * d^2) \rceil) = \text{real\_of\_int}\lceil 4 * M / (e * d^2) \rceil$ 
    using ⟨0 ≤ M⟩ e ⟨0 < d⟩
    by (simp add: field_simps)
  have  $4 * M / (e * d^2) + 1 \leq \text{real}(\text{Suc}(\text{nat}\lceil 4 * M / (e * d^2) \rceil))$ 
    by (simp add: real_nat_ceiling_ge)
  also have ... ≤ real n
    using n by (simp add: field_simps)
  finally have nbig:  $4 * M / (e * d^2) + 1 \leq \text{real } n$  .
  have sum_bern:  $(\sum_{k \leq n}. (x - k/n)^2 * \text{Bernstein } n \ k \ x) = x * (1 - x) / n$ 
  proof -
    have *:  $\bigwedge a \ b \ x::\text{real}. (a - b)^2 * x = a * (a - 1) * x + (1 - 2 * b) * a * x + b * b * x$ 
      by (simp add: algebra_simps power2_eq_square)
    have  $(\sum_{k \leq n}. (k - n * x)^2 * \text{Bernstein } n \ k \ x) = n * x * (1 - x)$ 
      apply (simp add: * sum.distrib)
      apply (simp flip: sum_distrib_left add: mult.assoc)
      apply (simp add: algebra_simps power2_eq_square)
      done
    then have  $(\sum_{k \leq n}. (k - n * x)^2 * \text{Bernstein } n \ k \ x) / n^2 = x * (1 - x) / n$ 
      by (simp add: power2_eq_square)
    then show ?thesis
      using n by (simp add: sum_divide_distrib field_split_simps power2_commute)
  qed
}
fix k
assume k:  $k \leq n$ 
then have kn:  $0 \leq k / n \leq 1$ 
  by (auto simp: field_split_simps)
consider (lessd)  $|x - k / n| < d \mid$  (ged)  $d \leq |x - k / n|$ 
  by linarith
then have  $|(f x - f (k/n))| \leq e/2 + 2 * M / d^2 * (x - k/n)^2$ 

```

```

proof cases
  case lessd
    then have  $|f x - f (k/n)| < e/2$ 
      using  $d x kn$  by (simp add: abs_minus_commute)
    also have  $\dots \leq (e/2 + 2 * M / d^2 * (x - k/n)^2)$ 
      using  $\langle M \geq 0 \rangle d$  by simp
    finally show ?thesis by simp
  next
    case ged
      then have  $dle: d^2 \leq (x - k/n)^2$ 
        by (metis d(1) less_eq_real_def power2_abs power_mono)
      have  $\S: 1 \leq (x - \text{real } k / \text{real } n)^2 / d^2$ 
        using  $dle \langle d > 0 \rangle$  by auto
      have  $|f x - f (k/n)| \leq |f x| + |f (k/n)|$ 
        by (rule abs_triangle_ineq4)
      also have  $\dots \leq M + M$ 
        by (meson M add_mono_thms_linordered_semiring(1) kn x)
      also have  $\dots \leq 2 * M * ((x - k/n)^2 / d^2)$ 
        using  $\S \langle M \geq 0 \rangle \text{mult\_left\_mono}$  by fastforce
      also have  $\dots \leq e/2 + 2 * M / d^2 * (x - k/n)^2$ 
        using  $e$  by simp
      finally show ?thesis .
    qed
  } note  $*$  = this
  have  $|f x - (\sum_{k \leq n}. f(k/n) * \text{Bernstein } n k x)| \leq |\sum_{k \leq n}. (f x - f(k/n))$ 
  * Bernstein } n k x|
    by (simp add: sum_subtractf sum_distrib_left [symmetric] algebra_simps)
  also have  $\dots \leq (\sum_{k \leq n}. |f x - f(k/n)| * \text{Bernstein } n k x|)$ 
    by (rule sum_abs)
  also have  $\dots \leq (\sum_{k \leq n}. (e/2 + (2 * M / d^2) * (x - k/n)^2) * \text{Bernstein } n$ 
   $k x)$ 
    using  $*$ 
    by (force simp add: abs_mult Bernstein_nonneg x mult_right_mono intro: sum_mono)
  also have  $\dots \leq e/2 + (2 * M) / (d^2 * n)$ 
    unfolding sum.distrib Rings.semiring_class.distrib_right sum_distrib_left
    [symmetric] mult.assoc sum_bern
    using  $\langle d > 0 \rangle x$  by (simp add: divide_simps \langle M \geq 0 \rangle mult_le_one mult_left_le)
  also have  $\dots < e$ 
    using  $\langle d > 0 \rangle \text{nbig } e \langle n > 0 \rangle$ 
    apply (simp add: field_split_simps)
    using ed0 by linarith
  finally have  $|f x - (\sum_{k \leq n}. f (\text{real } k / \text{real } n) * \text{Bernstein } n k x)| < e .$ 
  }
then show ?thesis
  by auto
qed

```

8.4.3 General Stone-Weierstrass theorem

Source: Bruno Brosowski and Frank Deutsch. An Elementary Proof of the Stone-Weierstrass Theorem. Proceedings of the American Mathematical Society Volume 81, Number 1, January 1981. DOI: 10.2307/2043993 <https://www.jstor.org/stable/2043993>

```

locale function_ring_on =
  fixes R :: ('a::t2_space  $\Rightarrow$  real) set and S :: 'a set
  assumes compact: compact S
  assumes continuous:  $f \in R \Rightarrow$  continuous_on S f
  assumes add:  $f \in R \Rightarrow g \in R \Rightarrow (\lambda x. f x + g x) \in R$ 
  assumes mult:  $f \in R \Rightarrow g \in R \Rightarrow (\lambda x. f x * g x) \in R$ 
  assumes const:  $(\lambda_. c) \in R$ 
  assumes separable:  $x \in S \Rightarrow y \in S \Rightarrow x \neq y \Rightarrow \exists f \in R. f x \neq f y$ 

begin
lemma minus:  $f \in R \Rightarrow (\lambda x. - f x) \in R$ 
  by (frule mult [OF const [of -1]]) simp

lemma diff:  $f \in R \Rightarrow g \in R \Rightarrow (\lambda x. f x - g x) \in R$ 
  unfolding diff_conv_add_uminus by (metis add minus)

lemma power:  $f \in R \Rightarrow (\lambda x. f x ^ n) \in R$ 
  by (induct n) (auto simp: const mult)

lemma sum:  $[\![$ finite I;  $\bigwedge i. i \in I \Rightarrow f i \in R]\!] \Rightarrow (\lambda x. \sum i \in I. f i x) \in R$ 
  by (induct I rule: finite_induct; simp add: const add)

lemma prod:  $[\![$ finite I;  $\bigwedge i. i \in I \Rightarrow f i \in R]\!] \Rightarrow (\lambda x. \prod i \in I. f i x) \in R$ 
  by (induct I rule: finite_induct; simp add: const mult)

definition normf :: ('a::t2_space  $\Rightarrow$  real)  $\Rightarrow$  real
  where normf f  $\equiv$  SUP  $x \in S. |f x|$ 

lemma normf_upper:
  assumes continuous_on S f  $x \in S$  shows  $|f x| \leq$  normf f
proof -
  have bdd_above  $((\lambda x. |f x|) ` S)$ 
  by (simp add: asms(1) bounded_imp_bdd_above compact_compact_continuous_image
compact_imp_bounded continuous_on_rabs)
  then show ?thesis
  using asms cSUP_upper normf_def by fastforce
qed

lemma normf_least:  $S \neq \{\}$   $\Rightarrow (\bigwedge x. x \in S \Rightarrow |f x| \leq M) \Rightarrow$  normf f  $\leq$  M
  by (simp add: normf_def cSUP_least)

end

```

lemma (in *function_ring_on*) one:

assumes U : open U **and** $t0$: $t0 \in S$ $t0 \in U$ **and** $t1$: $t1 \in S-U$

shows $\exists V$. open $V \wedge t0 \in V \wedge S \cap V \subseteq U \wedge$

$(\forall e > 0. \exists f \in R. f \text{ ' } S \subseteq \{0..1\} \wedge (\forall t \in S \cap V. f t < e) \wedge (\forall t \in S - U. f t > 1 - e))$

proof –

have $\exists pt \in R. pt \ t0 = 0 \wedge pt \ t > 0 \wedge pt \text{ ' } S \subseteq \{0..1\}$ **if** t : $t \in S - U$ **for** t

proof –

have $t \neq t0$ **using** $t \ t0$ **by** *auto*

then obtain g **where** g : $g \in R$ $g \ t \neq g \ t0$

using *separable* $t0$ **by** (*metis* *Diff_subset* *subset_eq* t)

define h **where** [*abs_def*]: $h \ x = g \ x - g \ t0$ **for** x

have $h \in R$

unfolding h_def **by** (*fast intro*: g *const* *diff*)

then have hsq : $(\lambda w. (h \ w)^2) \in R$

by (*simp add*: *power2_eq_square* *mult*)

have $h \ t \neq h \ t0$

by (*simp add*: h_def g)

then have $h \ t \neq 0$

by (*simp add*: h_def)

then have $ht2$: $0 < (h \ t)^2$

by *simp*

also have $\dots \leq \text{normf} (\lambda w. (h \ w)^2)$

using t *normf_upper* [**where** $x=t$] *continuous* [*OF* hsq] **by** *force*

finally have nfp : $0 < \text{normf} (\lambda w. (h \ w)^2)$.

define p **where** [*abs_def*]: $p \ x = (1 / \text{normf} (\lambda w. (h \ w)^2)) * (h \ x)^2$ **for** x

have $p \in R$

unfolding p_def **by** (*fast intro*: hsq *const* *mult*)

moreover have $p \ t0 = 0$

by (*simp add*: p_def h_def)

moreover have $p \ t > 0$

using nfp $ht2$ **by** (*simp add*: p_def)

moreover have $\bigwedge x. x \in S \implies p \ x \in \{0..1\}$

using nfp *normf_upper* [*OF* *continuous* [*OF* hsq]] **by** (*auto simp*: p_def)

ultimately show $\exists pt \in R. pt \ t0 = 0 \wedge pt \ t > 0 \wedge pt \text{ ' } S \subseteq \{0..1\}$

by *auto*

qed

then obtain pf **where** pf : $\bigwedge t. t \in S-U \implies pf \ t \in R \wedge pf \ t \ t0 = 0 \wedge pf \ t \ t > 0$

and $pf01$: $\bigwedge t. t \in S-U \implies pf \ t \text{ ' } S \subseteq \{0..1\}$

by *metis*

have com_sU : *compact* $(S-U)$

using *compact_closed_Int_compact* U **by** (*simp add*: *Diff_eq* *compact_Int_closed* *open_closed*)

have $\bigwedge t. t \in S-U \implies \exists A. \text{open } A \wedge A \cap S = \{x \in S. 0 < pf \ t \ x\}$

apply (*rule* *open_Collect_positive*)

by (*metis* pf *continuous*)

then obtain Uf **where** Uf : $\bigwedge t. t \in S-U \implies \text{open } (Uf \ t) \wedge (Uf \ t) \cap S = \{x \in S. 0 < pf \ t \ x\}$


```

    by metis
  then have open_Uf:  $\bigwedge t. t \in S-U \implies \text{open } (Uf\ t)$ 
    by blast
  have tUft:  $\bigwedge t. t \in S-U \implies t \in Uf\ t$ 
    using pf Uf by blast
  then have *:  $S-U \subseteq (\bigcup x \in S-U. Uf\ x)$ 
    by blast
  obtain subU where subU:  $subU \subseteq S - U$  finite subU  $S - U \subseteq (\bigcup x \in subU. Uf\ x)$ 
    by (blast intro: that compactE_image [OF com_sU open_Uf *])
  then have [simp]:  $subU \neq \{\}$ 
    using t1 by auto
  then have cardp:  $\text{card } subU > 0$  using subU
    by (simp add: card_gt_0_iff)
  define p where [abs_def]:  $p\ x = (1 / \text{card } subU) * (\sum t \in subU. pf\ t\ x)$  for x
  have pR:  $p \in R$ 
    unfolding p_def using subU pf by (fast intro: pf const mult sum)
  have pt0 [simp]:  $p\ t0 = 0$ 
    using subU pf by (auto simp: p_def intro: sum.neutral)
  have pt_pos:  $p\ t > 0$  if  $t: t \in S-U$  for t
  proof -
    obtain i where  $i: i \in subU\ t \in Uf\ i$  using subU t by blast
    show ?thesis
      using subU i t
      apply (clarsimp simp: p_def field_split_simps)
      apply (rule sum_pos2 [OF ‹finite subU›])
      using Uf t pf01 apply auto
      apply (force elim!: subsetCE)
      done
  qed
  have p01:  $p\ x \in \{0..1\}$  if  $t: x \in S$  for x
  proof -
    have  $0 \leq p\ x$ 
      using subU cardp t pf01
      by (fastforce simp add: p_def field_split_simps intro: sum_nonneg)
    moreover have  $p\ x \leq 1$ 
      using subU cardp t
      apply (simp add: p_def field_split_simps)
      apply (rule sum_bounded_above [where 'a=real and K=1, simplified])
      using pf01 by force
    ultimately show ?thesis
      by auto
  qed
  have compact (p ‹(S-U)›)
    by (meson Diff_subset com_sU compact_continuous_image continuous continuous_on_subset pR)
  then have open (-(p ‹(S-U)›))
    by (simp add: compact_imp_closed open_Comp1)
  moreover have  $0 \in -(p ‹(S-U)›)$ 

```

```

    by (metis (no_types) ComplI image_iff not_less_iff_gr_or_eq pt_pos)
  ultimately obtain delta0 where delta0: delta0 > 0 ball 0 delta0 ⊆ - (p ‘
(S-U))
  by (auto simp: elim!: openE)
  then have pt_delta:  $\bigwedge x. x \in S-U \implies p x \geq \text{delta0}$ 
  by (force simp: ball_def dist_norm dest: p01)
  define  $\delta$  where  $\delta = \text{delta0}/2$ 
  have delta0 ≤ 1 using delta0 p01 [of t1] t1
  by (force simp: ball_def dist_norm dest: p01)
  with delta0 have  $\delta 01: 0 < \delta \delta < 1$ 
  by (auto simp:  $\delta$ _def)
  have pt_δ:  $\bigwedge x. x \in S-U \implies p x \geq \delta$ 
  using pt_delta delta0 by (force simp:  $\delta$ _def)
  have  $\exists A. \text{open } A \wedge A \cap S = \{x \in S. p x < \delta/2\}$ 
  by (rule open_Collect_less_Int [OF continuous [OF pR] continuous_on_const])
  then obtain V where V: open V V ∩ S = {x ∈ S. p x < δ/2}
  by blast
  define k where k = nat[1/δ] + 1
  have k>0 by (simp add: k_def)
  have k-1 ≤ 1/δ
  using  $\delta 01$  by (simp add: k_def)
  with  $\delta 01$  have k ≤ (1+δ)/δ
  by (auto simp: algebra_simps add_divide_distrib)
  also have ... < 2/δ
  using  $\delta 01$  by (auto simp: field_split_simps)
  finally have k2δ: k < 2/δ .
  have 1/δ < k
  using  $\delta 01$  unfolding k_def by linarith
  with  $\delta 01$  k2δ have kδ: 1 < k*δ k*δ < 2
  by (auto simp: field_split_simps)
  define q where [abs_def]: q n t = (1 - p tδ)(kn) for n t
  have qR: q n ∈ R for n
  by (simp add: q_def const diff power pR)
  have q01:  $\bigwedge n t. t \in S \implies q n t \in \{0..1\}$ 
  using p01 by (simp add: q_def power_le_one algebra_simps)
  have qt0 [simp]:  $\bigwedge n. n > 0 \implies q n t0 = 1$ 
  using t0 pf by (simp add: q_def power_0_left)
  { fix t and n::nat
  assume t: t ∈ S ∩ V
  with ⟨k>0⟩ V have k * p t < k * δ / 2
  by force
  then have 1 - (k * δ / 2)n ≤ 1 - (k * p t)n
  using ⟨k>0⟩ p01 t by (simp add: power_mono)
  also have ... ≤ q n t
  using Bernoulli_inequality [of - ((p t)δ) kn]
  apply (simp add: q_def)
  by (metis IntE atLeastAtMost_iff p01 power_le_one power_mult_distrib t)
  finally have 1 - (k * δ / 2)n ≤ q n t .
} note limitV = this

```

```

{ fix t and n::nat
  assume t: t ∈ S - U
  with ⟨k>0⟩ U have k * δ ≤ k * p t
    by (simp add: pt_δ)
  with kδ have kpt: 1 < k * p t
    by (blast intro: less_le_trans)
  have ptn_pos: 0 < p t^n
    using pt_pos [OF t] by simp
  have ptn_le: p t^n ≤ 1
    by (meson DiffE atLeastAtMost_iff p01 power_le_one t)
  have q n t = (1/(k^n * (p t)^n)) * (1 - p t^n)^(k^n) * k^n * (p t)^n
    using pt_pos [OF t] ⟨k>0⟩ by (simp add: q_def)
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + k^n * (p t)^n)
    using pt_pos [OF t] ⟨k>0⟩
    by (simp add: divide_simps mult_left_mono ptn_le)
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + (p t)^n)^(k^n)
  proof (rule mult_left_mono [OF Bernoulli_inequality])
    show 0 ≤ 1 / (real k * p t)^n * (1 - p t^n)^(k^n)
      using ptn_pos ptn_le by (auto simp: power_mult_distrib)
  qed (use ptn_pos in auto)
  also have ... = (1/(k * (p t))^n) * (1 - p t^(2*n))^(k^n)
    using pt_pos [OF t] ⟨k>0⟩
  by (simp add: algebra_simps power_mult power2_eq_square flip: power_mult_distrib)
  also have ... ≤ (1/(k * (p t))^n) * 1
    using pt_pos ⟨k>0⟩ p01 power_le_one t
    by (intro mult_left_mono [OF power_le_one]) auto
  also have ... ≤ (1 / (k*δ))^n
    using ⟨k>0⟩ δ01 power_mono pt_δ t
    by (fastforce simp: field_simps)
  finally have q n t ≤ (1 / (real k * δ))^n .
} note limitNonU = this
define NN
  where NN e = 1 + nat [max (ln e / ln (real k * δ / 2)) (- ln e / ln (real k
* δ))] for e
  have NN: of_nat (NN e) > ln e / ln (real k * δ / 2) of_nat (NN e) > - ln e
/ ln (real k * δ)
    if 0 < e for e
  unfolding NN_def by linarith+
  have NN1: (k * δ / 2)^NN e < e if e > 0 for e
  proof -
    have ln ((real k * δ / 2)^NN e) = real (NN e) * ln (real k * δ / 2)
      by (simp add: ⟨δ>0⟩ ⟨0 < k⟩ ln_realpow)
    also have ... < ln e
      using NN kδ that by (force simp add: field_simps)
    finally show ?thesis
      by (simp add: ⟨δ>0⟩ ⟨0 < k⟩ that)
  qed
  have NN0: (1/(k*δ))^(NN e) < e if e > 0 for e
  proof -

```

```

have 0 < ln (real k) + ln δ
  using δ01(1) ⟨0 < k⟩ kδ(1) ln_gt_zero ln_mult by fastforce
then have real (NN e) * ln (1 / (real k * δ)) < ln e
  using kδ(1) NN(2) [of e] that by (simp add: ln_div divide_simps)
then have exp (real (NN e) * ln (1 / (real k * δ))) < e
  by (metis exp_less_mono exp_ln that)
then show ?thesis
  by (simp add: δ01(1) ⟨0 < k⟩ exp_of_nat_mult)
qed
{ fix t and e::real
  assume e>0
  have t ∈ S ∩ V ⇒ 1 - q (NN e) t < e t ∈ S - U ⇒ q (NN e) t < e
  proof -
    assume t: t ∈ S ∩ V
    show 1 - q (NN e) t < e
      by (metis add_commute diff_le_eq not_le limitV [OF t] less_le_trans [OF
NN1 [OF ⟨e>0⟩]])
    next
      assume t: t ∈ S - U
      show q (NN e) t < e
        using limitNonU [OF t] less_le_trans [OF NN0 [OF ⟨e>0⟩]] not_le by
blast
    qed
  } then have ∧e. e > 0 ⇒ ∃f∈R. f ' S ⊆ {0..1} ∧ (∀t ∈ S ∩ V. f t < e) ∧
(∀t ∈ S - U. 1 - e < f t)
  using q01
  by (rule_tac x=λx. 1 - q (NN e) x in bexI) (auto simp: algebra_simps intro:
diff const qR)
  moreover have t0V: t0 ∈ V S ∩ V ⊆ U
  using pt_δ t0 U V δ01 by fastforce+
  ultimately show ?thesis using V t0V
  by blast
}
qed

```

Non-trivial case, with A and B both non-empty

lemma (in *function_ring_on*) *two_special*:

```

assumes A: closed A A ⊆ S a ∈ A
  and B: closed B B ⊆ S b ∈ B
  and disj: A ∩ B = {}
  and e: 0 < e e < 1
shows ∃f ∈ R. f ' S ⊆ {0..1} ∧ (∀x ∈ A. f x < e) ∧ (∀x ∈ B. f x > 1 - e)
proof -
{ fix w
  assume w ∈ A
  then have open (- B) b ∈ S w ∉ B w ∈ S
    using asms by auto
  then have ∃V. open V ∧ w ∈ V ∧ S ∩ V ⊆ -B ∧
    (∀e>0. ∃f ∈ R. f ' S ⊆ {0..1} ∧ (∀x ∈ S ∩ V. f x < e) ∧ (∀x ∈ S
∩ B. f x > 1 - e))

```

```

    using one [of -B w b] assms ⟨w ∈ A⟩ by simp
  }
  then obtain Vf where Vf:
     $\bigwedge w. w \in A \implies \text{open } (Vf\ w) \wedge w \in Vf\ w \wedge S \cap Vf\ w \subseteq -B \wedge$ 
     $(\forall e > 0. \exists f \in R. f \text{ ' } S \subseteq \{0..1\} \wedge (\forall x \in S \cap Vf\ w. f\ x < e) \wedge$ 
     $(\forall x \in S \cap B. f\ x > 1 - e))$ 
    by metis
  then have open_Vf:  $\bigwedge w. w \in A \implies \text{open } (Vf\ w)$ 
    by blast
  have tVft:  $\bigwedge w. w \in A \implies w \in Vf\ w$ 
    using Vf by blast
  then have sum_max_0:  $A \subseteq (\bigcup x \in A. Vf\ x)$ 
    by blast
  have com_A: compact A using A
    by (metis compact compact_Int_closed inf.absorb_iff2)
  obtain subA where subA:  $subA \subseteq A$  finite subA  $A \subseteq (\bigcup x \in subA. Vf\ x)$ 
    by (blast intro: that compactE_image [OF com_A open_Vf sum_max_0])
  then have [simp]:  $subA \neq \{\}$ 
    using ⟨a ∈ A⟩ by auto
  then have cardp:  $\text{card } subA > 0$  using subA
    by (simp add: card_gt_0_iff)
  have  $\bigwedge w. w \in A \implies \exists f \in R. f \text{ ' } S \subseteq \{0..1\} \wedge (\forall x \in S \cap Vf\ w. f\ x < e / \text{card}$ 
   $subA) \wedge (\forall x \in S \cap B. f\ x > 1 - e / \text{card } subA)$ 
    using Vf e cardp by simp
  then obtain ff where ff:
     $\bigwedge w. w \in A \implies ff\ w \in R \wedge ff\ w \text{ ' } S \subseteq \{0..1\} \wedge$ 
     $(\forall x \in S \cap Vf\ w. ff\ w\ x < e / \text{card } subA) \wedge (\forall x \in S \cap B. ff\ w$ 
   $x > 1 - e / \text{card } subA)$ 
    by metis
  define pff where [abs_def]:  $pff\ x = (\prod w \in subA. ff\ w\ x)$  for x
  have pffR:  $pff \in R$ 
    unfolding pff_def using subA ff by (auto simp: intro: prod)
  moreover
  have pff01:  $pff\ x \in \{0..1\}$  if t:  $x \in S$  for x
  proof -
    have  $0 \leq pff\ x$ 
      using subA cardp t ff
    by (fastforce simp: pff_def field_split_simps sum_nonneg intro: prod_nonneg)
    moreover have  $pff\ x \leq 1$ 
      using subA cardp t ff
    by (fastforce simp add: pff_def field_split_simps sum_nonneg intro: prod_mono
    [where g =  $\lambda x. 1$ , simplified])
    ultimately show ?thesis
      by auto
  qed
  moreover
  { fix v x
    assume v:  $v \in subA$  and x:  $x \in Vf\ v \wedge x \in S$ 
    from subA v have  $pff\ x = ff\ v\ x * (\prod w \in subA - \{v\}. ff\ w\ x)$ 

```

```

    unfolding pff_def by (metis prod.remove)
  also have ... ≤ ff v x * 1
  proof -
    have  $\bigwedge i. i \in \text{subA} - \{v\} \implies 0 \leq \text{ff } i \ x \wedge \text{ff } i \ x \leq 1$ 
    by (metis Diff_subset atLeastAtMost_iff ff_image_subset_iff subA(1) subsetD
    x(2))
    moreover have  $0 \leq \text{ff } v \ x$ 
      using ff_subA(1) v x(2) by fastforce
    ultimately show ?thesis
      by (metis mult_left_mono prod_mono [where g =  $\lambda x. 1$ , simplified])
  qed
  also have ... < e / card subA
    using ff_subA(1) v x by auto
  also have ... ≤ e
    using cardp e by (simp add: field_split_simps)
  finally have pff x < e .
}
then have  $\bigwedge x. x \in A \implies \text{pff } x < e$ 
  using A Vf subA by (metis UN_E contra_subsetD)
moreover
{ fix x
  assume x:  $x \in B$ 
  then have  $x \in S$ 
    using B by auto
  have  $1 - e \leq (1 - e / \text{card subA})^{\text{card subA}}$ 
    using Bernoulli_inequality [of  $-e / \text{card subA}$   $\text{card subA}$ ] e cardp
    by (auto simp: field_simps)
  also have ... =  $(\prod w \in \text{subA}. 1 - e / \text{card subA})$ 
    by (simp add: subA(2))
  also have ... < pff x
  proof -
    have  $\bigwedge i. i \in \text{subA} \implies e / \text{real } (\text{card subA}) \leq 1 \wedge 1 - e / \text{real } (\text{card subA})$ 
    < ff i x
      using e ⟨B ⊆ S⟩ ff_subA(1) x by (force simp: field_split_simps)
    then show ?thesis
      using prod_mono_strict[of subA  $\lambda x. 1 - e / \text{card subA}$ ] subA
      unfolding pff_def by (smt (verit, best) UN_E assms(3) subsetD)
  qed
  finally have  $1 - e < \text{pff } x$  .
}
ultimately show ?thesis by blast
qed

```

lemma (in function_ring_on) two:

```

assumes A: closed A A ⊆ S
        and B: closed B B ⊆ S
        and disj: A ∩ B = {}
        and e: 0 < e e < 1
shows  $\exists f \in R. f \text{ ' } S \subseteq \{0..1\} \wedge (\forall x \in A. f \ x < e) \wedge (\forall x \in B. f \ x > 1 - e)$ 

```

```

proof (cases  $A \neq \{\} \wedge B \neq \{\}$ )
  case True then show ?thesis
    using assms
    by (force simp flip: ex_in_conv intro!: two_special)
next
  case False
  then consider  $A=\{\} \mid B=\{\}$  by force
  then show ?thesis
  proof cases
    case 1
    with e show ?thesis
    by (rule_tac  $x=\lambda x. 1$  in bexI) (auto simp: const)
  next
    case 2
    with e show ?thesis
    by (rule_tac  $x=\lambda x. 0$  in bexI) (auto simp: const)
  qed
qed

```

The special case where f is non-negative and $e < (1::'a) / (3::'a)$

```

lemma (in function_ring_on) Stone_Weierstrass_special:
  assumes f: continuous_on S f and fpos:  $\bigwedge x. x \in S \implies f x \geq 0$ 
  and e:  $0 < e \wedge e < 1/3$ 
  shows  $\exists g \in R. \forall x \in S. |f x - g x| < 2 * e$ 
proof -
  define n where  $n = 1 + \text{nat } \lceil \text{norm } f / e \rceil$ 
  define A where  $A j = \{x \in S. f x \leq (j - 1/3) * e\}$  for  $j :: \text{nat}$ 
  define B where  $B j = \{x \in S. f x \geq (j + 1/3) * e\}$  for  $j :: \text{nat}$ 
  have ngt:  $(n-1) * e \geq \text{norm } f$ 
    using e pos_divide_le_eq real_nat_ceiling_ge [of norm f / e]
    by (fastforce simp add: divide_simps n_def)
  moreover have  $n \geq 1$ 
    by (simp_all add: n_def)
  ultimately have ge_fx:  $(n-1) * e \geq f x$  if  $x \in S$  for  $x$ 
    using f normf_upper that by fastforce
  have closed S
    by (simp add: compact compact_imp_closed)
  { fix j
    have closed (A j) A j  $\subseteq$  S
      using  $\langle \text{closed } S \rangle$  continuous_on_closed_Collect_le [OF f continuous_on_const]
      by (simp_all add: A_def Collect_restrict)
    moreover have closed (B j) B j  $\subseteq$  S
      using  $\langle \text{closed } S \rangle$  continuous_on_closed_Collect_le [OF f continuous_on_const
f]
      by (simp_all add: B_def Collect_restrict)
    moreover have  $(A j) \cap (B j) = \{\}$ 
      using e by (auto simp: A_def B_def field_simps)
    ultimately have  $\exists f \in R. f ' S \subseteq \{0..1\} \wedge (\forall x \in A j. f x < e/n) \wedge (\forall x \in B j. f x > 1 - e/n)$ 

```

```

    using e ⟨1 ≤ n⟩ by (auto intro: two)
  }
then obtain xf where xfR:  $\bigwedge j. xf\ j \in R$  and xf01:  $\bigwedge j. xf\ j \in S \subseteq \{0..1\}$ 
    and xfA:  $\bigwedge x\ j. x \in A\ j \implies xf\ j\ x < e/n$ 
    and xfB:  $\bigwedge x\ j. x \in B\ j \implies xf\ j\ x > 1 - e/n$ 
  by metis
define g where [abs_def]:  $g\ x = e * (\sum i \leq n. xf\ i\ x)$  for x
have gR:  $g \in R$ 
  unfolding g_def by (fast intro: mult const sum xfR)
have gge0:  $\bigwedge x. x \in S \implies g\ x \geq 0$ 
  using e xf01 by (simp add: g_def zero_le_mult_iff image_subset_iff sum_nonneg)
have A0:  $A\ 0 = \{\}$ 
  using fpos e by (fastforce simp: A_def)
have An:  $A\ n = S$ 
  using e ngt ⟨n ≥ 1⟩ f normf_upper by (fastforce simp: A_def field_simps
of_nat_diff)
have Asub:  $A\ j \subseteq A\ i$  if  $i \geq j$  for  $i\ j$ 
  using e that by (force simp: A_def intro: order_trans)
{ fix t
  assume t:  $t \in S$ 
  define j where  $j = (LEAST\ j. t \in A\ j)$ 
  have jn:  $j \leq n$ 
    using t An by (simp add: Least_le j_def)
  have Aj:  $t \in A\ j$ 
    using t An by (fastforce simp add: j_def intro: LeastI)
  then have Ai:  $t \in A\ i$  if  $i \geq j$  for  $i$ 
    using Asub [OF that] by blast
  then have fj1:  $f\ t \leq (j - 1/3)*e$ 
    by (simp add: A_def)
  then have Anj:  $t \notin A\ i$  if  $i < j$  for  $i$ 
    using Aj ⟨i < j⟩ not_less_Least by (fastforce simp add: j_def)
  have j1:  $1 \leq j$ 
    using A0 Aj j_def not_less_eq_eq by (fastforce simp add: j_def)
  then have Anj:  $t \notin A\ (j-1)$ 
    using Least_le by (fastforce simp add: j_def)
  then have fj2:  $(j - 4/3)*e < f\ t$ 
    using j1 t by (simp add: A_def of_nat_diff)
  have xf_le1:  $\bigwedge i. xf\ i\ t \leq 1$ 
    using xf01 t by force
  have g t =  $e * (\sum i \leq n. xf\ i\ t)$ 
    by (simp add: g_def flip: distrib_left)
  also have ... =  $e * (\sum i \in \{..j\} \cup \{j..n\}. xf\ i\ t)$ 
    by (simp add: ivl_disj_un_one(4) jn)
  also have ... =  $e * (\sum i < j. xf\ i\ t) + e * (\sum i = j..n. xf\ i\ t)$ 
    by (simp add: distrib_left ivl_disj_int sum.union_disjoint)
  also have ...  $\leq e*j + e * ((Suc\ n - j)*e/n)$ 
  proof (intro add_mono mult_left_mono)
    show  $(\sum i < j. xf\ i\ t) \leq j$ 
      by (rule sum_bounded_above [OF xf_le1, where A = lessThan j, simplified])

```



```

have xfi t ≤ e/n if i ≥ j for i
  using xfiA [OF Ai] that by (simp add: less_eq_real_def)
then show (∑ i = j..n. xfi t) ≤ real (Suc n - j) * e / real n
  using sum_bounded_above [of {j..n} λi. xfi t]
  by fastforce
qed (use e in auto)
also have ... ≤ j*e + e*(n - j + 1)*e/n
  using <1 ≤ n> e by (simp add: field_simps del: of_nat_Suc)
also have ... ≤ j*e + e*e
  using <1 ≤ n> e j1 by (simp add: field_simps del: of_nat_Suc)
also have ... < (j + 1/3)*e
  using e by (auto simp: field_simps)
finally have gj1: g t < (j + 1 / 3) * e .
have gj2: (j - 4/3)*e < g t
proof (cases 2 ≤ j)
  case False
  then have j=1 using j1 by simp
  with t gge0 e show ?thesis by force
next
  case True
  then have (j - 4/3)*e < (j-1)*e - e^2
    using e by (auto simp: of_nat_diff algebra_simps power2_eq_square)
  also have ... < (j-1)*e - ((j - 1)/n) * e^2
    using e True jn by (simp add: power2_eq_square field_simps)
  also have ... = e * (j-1) * (1 - e/n)
    by (simp add: power2_eq_square field_simps)
  also have ... ≤ e * (∑ i ≤ j-2. xfi t)
proof -
  { fix i
    assume i+2 ≤ j
    then obtain d where i+2+d = j
      using le_Suc_ex that by blast
    then have t ∈ B i
      using Anj e ge_fx [OF t] <1 ≤ n> fpos [OF t] t
      unfolding A_def B_def
      by (auto simp add: field_simps of_nat_diff not_le intro: order_trans
[of _ e * 2 + e * d * 3 + e * i * 3])
    then have xfi t > 1 - e/n
      by (rule xfiB)
  }
  moreover have real (j - Suc 0) * (1 - e / real n) ≤ real (card {..j - 2})
* (1 - e / real n)
  using Suc_diff_le True by fastforce
ultimately show ?thesis
  using e True by (auto intro: order_trans [OF _ sum_bounded_below [OF
less_imp_le]])
qed
also have ... ≤ g t
  using jn e xfi01 t

```

```

    by (auto intro!: Groups_Big.sum_mono2 simp add: g_def zero_le_mult_iff
image_subset_iff sum_nonneg)
    finally show ?thesis .
  qed
  have |f t - g t| < 2 * e
    using fj1 fj2 gj1 gj2 by (simp add: abs_less_iff field_simps)
}
then show ?thesis
  by (rule_tac x=g in bexI) (auto intro: gR)
qed

```

The “unpretentious” formulation

```

proposition (in function_ring_on) Stone_Weierstrass_basic:
  assumes f: continuous_on S f and e: e > 0
  shows  $\exists g \in R. \forall x \in S. |f x - g x| < e$ 
proof -
  have  $\exists g \in R. \forall x \in S. |(f x + normf f) - g x| < 2 * \min (e/2) (1/4)$ 
proof (rule Stone_Weierstrass_special)
  show continuous_on S ( $\lambda x. f x + normf f$ )
    by (force intro: Limits.continuous_on_add [OF f Topological_Spaces.continuous_on_const])
  show  $\bigwedge x. x \in S \implies 0 \leq f x + normf f$ 
    using normf_upper [OF f] by force
qed (use e in auto)
  then obtain g where  $g \in R \forall x \in S. |g x - (f x + normf f)| < e$ 
    by force
  then show ?thesis
    by (rule_tac x= $\lambda x. g x - normf f$  in bexI) (auto simp: algebra_simps intro:
diff const)
qed

```

```

theorem (in function_ring_on) Stone_Weierstrass:
  assumes f: continuous_on S f
  shows  $\exists F \in UNIV \rightarrow R. LIM n \text{ sequentially. } F n :> \text{uniformly\_on } S f$ 
proof -
  define h where  $h \equiv \lambda n::nat. \text{SOME } g. g \in R \wedge (\forall x \in S. |f x - g x| < 1 / (1 + n))$ 
  show ?thesis
proof
  { fix e::real
    assume e:  $0 < e$ 
    then obtain N::nat where  $N: 0 < N \ 0 < \text{inverse } N \ \text{inverse } N < e$ 
      by (auto simp: real_arch_inverse [of e])
    { fix n :: nat and x :: 'a and g :: 'a  $\implies$  real
      assume n:  $N \leq n \ \forall x \in S. |f x - g x| < 1 / (1 + \text{real } n)$ 
      assume x:  $x \in S$ 
      have  $\neg \text{real } (\text{Suc } n) < \text{inverse } e$ 
        using  $\langle N \leq n \rangle N$  using less_imp_inverse_less by force
      then have  $1 / (1 + \text{real } n) \leq e$ 

```

```

    using e by (simp add: field_simps)
  then have |f x - g x| < e
    using n(2) x by auto
}
then have  $\forall_F n$  in sequentially.  $\forall x \in S. |f x - h n x| < e$ 
  unfolding h_def
  by (force intro: someI2_bex [OF Stone_Weierstrass_basic [OF f]] eventually_sequentiallyI [of N])
}
then show uniform_limit S h f sequentially
  unfolding uniform_limit_iff by (auto simp: dist_norm abs_minus_commute)
show h  $\in$  UNIV  $\rightarrow$  R
  unfolding h_def by (force intro: someI2_bex [OF Stone_Weierstrass_basic [OF f]])
qed
qed

```

A HOL Light formulation

corollary *Stone_Weierstrass_HOL*:

```

fixes R :: ('a::t2_space  $\Rightarrow$  real) set and S :: 'a set
assumes compact S  $\wedge$  c. P( $\lambda$ x. c::real)
   $\wedge$  f. P f  $\Longrightarrow$  continuous_on S f
   $\wedge$  f g. P(f)  $\wedge$  P(g)  $\Longrightarrow$  P( $\lambda$ x. f x + g x)  $\wedge$  f g. P(f)  $\wedge$  P(g)  $\Longrightarrow$  P( $\lambda$ x. f
x * g x)
   $\wedge$  x y. x  $\in$  S  $\wedge$  y  $\in$  S  $\wedge$  x  $\neq$  y  $\Longrightarrow$   $\exists$  f. P(f)  $\wedge$  f x  $\neq$  f y
  continuous_on S f
  0 < e
shows  $\exists$  g. P(g)  $\wedge$  ( $\forall$  x  $\in$  S. |f x - g x| < e)
proof -
  interpret PR: function_ring_on Collect P
  by unfold_locales (use assms in auto)
  show ?thesis
  using PR.Stone_Weierstrass_basic [OF  $\langle$ continuous_on S f $\rangle$   $\langle$ 0 < e $\rangle$ ]
  by blast
qed

```

8.4.4 Polynomial functions

inductive *real_polynomial_function* :: ('a::real_normed_vector \Rightarrow real) \Rightarrow bool

where

```

  linear: bounded_linear f  $\Longrightarrow$  real_polynomial_function f
| const: real_polynomial_function ( $\lambda$ x. c)
| add:  $\llbracket$ real_polynomial_function f; real_polynomial_function g $\rrbracket$   $\Longrightarrow$  real_polynomial_function
( $\lambda$ x. f x + g x)
| mult:  $\llbracket$ real_polynomial_function f; real_polynomial_function g $\rrbracket$   $\Longrightarrow$  real_polynomial_function
( $\lambda$ x. f x * g x)

```

declare *real_polynomial_function.intros* [intro]

definition *polynomial_function* :: ('a::real_normed_vector \Rightarrow 'b::real_normed_vector) \Rightarrow bool

where

polynomial_function p \equiv ($\forall f$. bounded_linear f \longrightarrow real_polynomial_function (f o p))

lemma *real_polynomial_function_eq*: real_polynomial_function p = polynomial_function p

unfolding *polynomial_function_def*

proof

assume *real_polynomial_function* p

then show $\forall f$. bounded_linear f \longrightarrow real_polynomial_function (f o p)

proof (*induction* p *rule*: real_polynomial_function.induct)

case (*linear* h) **then show** ?case

by (*auto simp*: bounded_linear_compose real_polynomial_function.linear)

next

case (*const* h) **then show** ?case

by (*simp add*: real_polynomial_function.const)

next

case (*add* h) **then show** ?case

by (*force simp add*: bounded_linear_def linear_add real_polynomial_function.add)

next

case (*mult* h) **then show** ?case

by (*force simp add*: real_bounded_linear_const real_polynomial_function.mult)

qed

next

assume [*rule_format*, *OF* bounded_linear_ident]: $\forall f$. bounded_linear f \longrightarrow real_polynomial_function (f o p)

then show real_polynomial_function p

by (*simp add*: o_def)

qed

lemma *polynomial_function_const* [*iff*]: polynomial_function (λx . c)

by (*simp add*: polynomial_function_def o_def const)

lemma *polynomial_function_bounded_linear*:

bounded_linear f \implies polynomial_function f

by (*simp add*: polynomial_function_def o_def bounded_linear_compose real_polynomial_function.linear)

lemma *polynomial_function_id* [*iff*]: polynomial_function(λx . x)

by (*simp add*: polynomial_function_bounded_linear)

lemma *polynomial_function_add* [*intro*]:

\llbracket polynomial_function f; polynomial_function g $\rrbracket \implies$ polynomial_function (λx . f x + g x)

by (*auto simp*: polynomial_function_def bounded_linear_def linear_add real_polynomial_function.add o_def)

lemma *polynomial_function_mult* [*intro*]:

assumes f : *polynomial_function* f **and** g : *polynomial_function* g
shows *polynomial_function* $(\lambda x. f\ x *_{\mathbb{R}} g\ x)$
proof –
have *real_polynomial_function* $(\lambda x. h\ (g\ x))$ **if** *bounded_linear* h **for** h
using g **that** **unfolding** *polynomial_function_def* *o_def* *bounded_linear_def*
by (*auto simp: real_polynomial_function_eq*)
moreover **have** *real_polynomial_function* f
by (*simp add: f real_polynomial_function_eq*)
ultimately show *?thesis*
unfolding *polynomial_function_def* *bounded_linear_def* *o_def*
by (*auto simp: linear.scaleR*)
qed

lemma *polynomial_function_cmul* [*intro*]:
assumes f : *polynomial_function* f
shows *polynomial_function* $(\lambda x. c *_{\mathbb{R}} f\ x)$
by (*rule polynomial_function_mult [OF polynomial_function_const f]*)

lemma *polynomial_function_minus* [*intro*]:
assumes f : *polynomial_function* f
shows *polynomial_function* $(\lambda x. - f\ x)$
using *polynomial_function_cmul* [*OF f, of -1*] **by** *simp*

lemma *polynomial_function_diff* [*intro*]:
 $\llbracket \text{polynomial_function } f; \text{polynomial_function } g \rrbracket \implies \text{polynomial_function } (\lambda x. f\ x - g\ x)$
unfolding *add_uminus_conv_diff* [*symmetric*]
by (*metis polynomial_function_add polynomial_function_minus*)

lemma *polynomial_function_sum* [*intro*]:
 $\llbracket \text{finite } I; \bigwedge i. i \in I \implies \text{polynomial_function } (\lambda x. f\ x\ i) \rrbracket \implies \text{polynomial_function } (\lambda x. \text{sum } (f\ x)\ I)$
by (*induct I rule: finite_induct*) *auto*

lemma *real_polynomial_function_minus* [*intro*]:
 $\text{real_polynomial_function } f \implies \text{real_polynomial_function } (\lambda x. - f\ x)$
using *polynomial_function_minus* [*of f*]
by (*simp add: real_polynomial_function_eq*)

lemma *real_polynomial_function_diff* [*intro*]:
 $\llbracket \text{real_polynomial_function } f; \text{real_polynomial_function } g \rrbracket \implies \text{real_polynomial_function } (\lambda x. f\ x - g\ x)$
using *polynomial_function_diff* [*of f*]
by (*simp add: real_polynomial_function_eq*)

lemma *real_polynomial_function_divide* [*intro*]:
assumes p : *real_polynomial_function* p **shows** *real_polynomial_function* $(\lambda x. p\ x / c)$
proof –

2770

```
have real_polynomial_function ( $\lambda x. p x * Fields.inverse c$ )
  using assms by auto
then show ?thesis
  by (simp add: divide_inverse)
qed
```

```
lemma real_polynomial_function_sum [intro]:
   $\llbracket finite I; \bigwedge i. i \in I \implies real\_polynomial\_function (\lambda x. f x i) \rrbracket \implies real\_polynomial\_function$ 
 $(\lambda x. sum (f x) I)$ 
  using polynomial_function_sum [of I f]
  by (simp add: real_polynomial_function_eq)
```

```
lemma real_polynomial_function_prod [intro]:
   $\llbracket finite I; \bigwedge i. i \in I \implies real\_polynomial\_function (\lambda x. f x i) \rrbracket \implies real\_polynomial\_function$ 
 $(\lambda x. prod (f x) I)$ 
  by (induct I rule: finite_induct) auto
```

```
lemma real_polynomial_function_gchoose:
  obtains p where real_polynomial_function p  $\bigwedge x. x gchoose r = p x$ 
proof
  show real_polynomial_function ( $\lambda x. (\prod i = 0..<r. x - real i) / fact r$ )
    by force
qed (simp add: gbinomial_prod_rev)
```

```
lemma real_polynomial_function_power [intro]:
  real_polynomial_function f  $\implies real\_polynomial\_function (\lambda x. f x ^ n)$ 
  by (induct n) (simp_all add: const mult)
```

```
lemma real_polynomial_function_compose [intro]:
  assumes f: polynomial_function f and g: real_polynomial_function g
  shows real_polynomial_function (g o f)
  using g
proof (induction g rule: real_polynomial_function.induct)
  case (linear f)
  then show ?case
    using f polynomial_function_def by blast
next
  case (add f g)
  then show ?case
    using f add by (auto simp: polynomial_function_def)
next
  case (mult f g)
  then show ?case
    using f mult by (auto simp: polynomial_function_def)
qed auto
```

```
lemma polynomial_function_compose [intro]:
  assumes f: polynomial_function f and g: polynomial_function g
  shows polynomial_function (g o f)
```

using *g real_polynomial_function_compose* [OF *f*]
 by (auto simp: *polynomial_function_def o_def*)

lemma *sum_max_0*:

fixes *x::real*

shows $(\sum_{i \leq \max m n} x^i * (\text{if } i \leq m \text{ then } a \ i \text{ else } 0)) = (\sum_{i \leq m} x^i * a \ i)$

proof -

have $(\sum_{i \leq \max m n} x^i * (\text{if } i \leq m \text{ then } a \ i \text{ else } 0)) = (\sum_{i \leq \max m n} (\text{if } i \leq m \text{ then } x^i * a \ i \text{ else } 0))$

by (auto simp: *algebra_simps intro: sum.cong*)

also have ... = $(\sum_{i \leq m} (\text{if } i \leq m \text{ then } x^i * a \ i \text{ else } 0))$

by (*rule sum_mono_neutral_right*) auto

also have ... = $(\sum_{i \leq m} x^i * a \ i)$

by (auto simp: *algebra_simps intro: sum.cong*)

finally show ?thesis .

qed

lemma *real_polynomial_function_imp_sum*:

assumes *real_polynomial_function f*

shows $\exists a \ n::\text{nat}. f = (\lambda x. \sum_{i \leq n} a \ i * x^i)$

using *assms*

proof (*induct f*)

case (*linear f*)

then obtain *c* where *f*: *f* = $(\lambda x. x * c)$

by (auto simp *add: real_bounded_linear*)

have $x * c = (\sum_{i \leq 1} (\text{if } i = 0 \text{ then } 0 \text{ else } c) * x^i)$ for *x*

by (*simp add: mult_ac*)

with *f* show ?case

by *fastforce*

next

case (*const c*)

have $c = (\sum_{i \leq 0} c * x^i)$ for *x*

by *auto*

then show ?case

by *fastforce*

case (*add f1 f2*)

then obtain *a1 n1 a2 n2* where

f1 = $(\lambda x. \sum_{i \leq n1} a1 \ i * x^i)$ *f2* = $(\lambda x. \sum_{i \leq n2} a2 \ i * x^i)$

by *auto*

then have $f1 \ x + f2 \ x = (\sum_{i \leq \max n1 \ n2} ((\text{if } i \leq n1 \text{ then } a1 \ i \text{ else } 0) + (\text{if } i \leq n2 \text{ then } a2 \ i \text{ else } 0)) * x^i)$

for *x*

using *sum_max_0* [where *m=n1* and *n=n2*] *sum_max_0* [where *m=n2* and *n=n1*]

by (*simp add: sum.distrib algebra_simps max.commute*)

then show ?case

by *force*

case (*mult f1 f2*)

then obtain *a1 n1 a2 n2* where

2772

```

    f1 = (λx. ∑ i≤n1. a1 i * x^i) f2 = (λx. ∑ i≤n2. a2 i * x^i)
  by auto
  then obtain b1 b2 where
    f1 = (λx. ∑ i≤n1. b1 i * x^i) f2 = (λx. ∑ i≤n2. b2 i * x^i)
    b1 = (λi. if i≤n1 then a1 i else 0) b2 = (λi. if i≤n2 then a2 i else 0)
  by auto
  then have f1 x * f2 x = (∑ i≤n1 + n2. (∑ k≤i. b1 k * b2 (i - k)) * x ^ i)
for x
  using polynomial_product [of n1 b1 n2 b2] by (simp add: Set_Interval.atLeast0AtMost)
  then show ?case
    by force
qed

```

```

lemma real_polynomial_function_iff_sum:
  real_polynomial_function f ↔ (∃ a n. f = (λx. ∑ i≤n. a i * x^i)) (is ?lhs
= ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis real_polynomial_function_imp_sum)
next
  assume ?rhs then show ?lhs
    by (auto simp: linear_mult_const real_polynomial_function_power real_polynomial_function_sum)
qed

```

```

lemma polynomial_function_iff_Basis_inner:
  fixes f :: 'a::real_normed_vector ⇒ 'b::euclidean_space
  shows polynomial_function f ↔ (∀ b∈Basis. real_polynomial_function (λx.
inner (f x) b))
  (is ?lhs = ?rhs)
unfolding polynomial_function_def
proof (intro iffI allI impI)
  assume ∀ h. bounded_linear h → real_polynomial_function (h ∘ f)
  then show ?rhs
    by (force simp add: bounded_linear_inner_left o_def)
next
  fix h :: 'b ⇒ real
  assume rp: ∀ b∈Basis. real_polynomial_function (λx. f x · b) and h: bounded_linear
h
  have real_polynomial_function (h ∘ (λx. ∑ b∈Basis. (f x · b) *R b))
    using rp
    by (force simp: real_polynomial_function_eq polynomial_function_mult
      intro!: real_polynomial_function_compose [OF _ linear [OF h]])
  then show real_polynomial_function (h ∘ f)
    by (simp add: euclidean_representation_sum_fun)
qed

```

8.4.5 Stone-Weierstrass theorem for polynomial functions

First, we need to show that they are continuous, differentiable and separable.


```

lemma continuous_real_polynomial_function:
  assumes real_polynomial_function f
  shows continuous (at x) f
using assms
by (induct f) (auto simp: linear_continuous_at)

lemma continuous_polynomial_function:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space
  assumes polynomial_function f
  shows continuous (at x) f
proof (rule euclidean_isCont)
  show  $\bigwedge b. b \in \text{Basis} \implies \text{isCont} (\lambda x. (f\ x \cdot b) *_R b) x$ 
  using assms continuous_real_polynomial_function
  by (force simp: polynomial_function_iff_Basis_inner intro: isCont_scaleR)
qed

lemma continuous_on_polynomial_function:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  'b::euclidean_space
  assumes polynomial_function f
  shows continuous_on S f
using continuous_polynomial_function [OF assms] continuous_at_imp_continuous_on
by blast

lemma has_real_derivative_polynomial_function:
  assumes real_polynomial_function p
  shows  $\exists p'. \text{real\_polynomial\_function } p' \wedge$ 
     $(\forall x. (p \text{ has\_real\_derivative } (p' x)) (at\ x))$ 
using assms
proof (induct p)
  case (linear p)
  then show ?case
  by (force simp: real_bounded_linear const intro!: derivative_eq_intros)
next
  case (const c)
  show ?case
  by (rule_tac  $x=\lambda x. 0$  in exI) auto
  case (add f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1  $\bigwedge x. (f1 \text{ has\_real\_derivative } p1\ x) (at\ x)$ 
    real_polynomial_function p2  $\bigwedge x. (f2 \text{ has\_real\_derivative } p2\ x) (at\ x)$ 
  by auto
  then show ?case
  by (rule_tac  $x=\lambda x. p1\ x + p2\ x$  in exI) (auto intro!: derivative_eq_intros)
  case (mult f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1  $\bigwedge x. (f1 \text{ has\_real\_derivative } p1\ x) (at\ x)$ 
    real_polynomial_function p2  $\bigwedge x. (f2 \text{ has\_real\_derivative } p2\ x) (at\ x)$ 
  by auto
  then show ?case

```

2774

```

using mult
by (rule_tac x= $\lambda x. f1\ x * p2\ x + f2\ x * p1\ x$  in exI) (auto intro!: derivative_eq_intros)
qed

```

lemma *has_vector_derivative_polynomial_function*:

```

fixes p :: real  $\Rightarrow$  'a::euclidean_space
assumes polynomial_function p
obtains p' where polynomial_function p'  $\wedge$  x. (p has_vector_derivative (p' x))
(at x)
proof -
  { fix b :: 'a
    assume b  $\in$  Basis
    then
      obtain p' where p': real_polynomial_function p' and pd:  $\wedge$  x. (( $\lambda x. p\ x \cdot b$ )
has_real_derivative p' x) (at x)
      using assms [unfolded polynomial_function_iff_Basis_inner] has_real_derivative_polynomial_function
      by blast
      have polynomial_function ( $\lambda x. p' x *_{\mathbb{R}} b$ )
      using <b  $\in$  Basis> p' const [where 'a=real and c=0]
      by (simp add: polynomial_function_iff_Basis_inner inner_Basis)
      then have  $\exists q. polynomial\_function\ q \wedge (\forall x. ((\lambda u. (p\ u \cdot b) *_{\mathbb{R}} b) has\_vector\_derivative\ q\ x))$ 
      (at x)
      by (fastforce intro: derivative_eq_intros pd)
    }
  then obtain qf where qf:
     $\wedge b. b \in Basis \implies polynomial\_function\ (qf\ b)$ 
     $\wedge b\ x. b \in Basis \implies ((\lambda u. (p\ u \cdot b) *_{\mathbb{R}} b) has\_vector\_derivative\ qf\ b\ x)$  (at x)
  by metis
  show ?thesis
  proof
    show  $\wedge x. (p\ has\_vector\_derivative\ (\sum_{b \in Basis} qf\ b\ x))$  (at x)
    apply (subst euclidean_representation_sum_fun [of p, symmetric])
    by (auto intro: has_vector_derivative_sum qf)
  qed (force intro: qf)
qed

```

lemma *real_polynomial_function_separable*:

```

fixes x :: 'a::euclidean_space
assumes x  $\neq$  y shows  $\exists f. real\_polynomial\_function\ f \wedge f\ x \neq f\ y$ 
proof -
  have real_polynomial_function ( $\lambda u. \sum_{b \in Basis} (inner\ (x-u)\ b) ^2$ )
  proof (rule real_polynomial_function_sum)
    show  $\wedge i. i \in Basis \implies real\_polynomial\_function\ (\lambda u. ((x - u) \cdot i)^2)$ 
    by (auto simp: algebra_simps real_polynomial_function_diff const linear
bounded_linear_inner_left)
  qed auto
  moreover have  $(\sum_{b \in Basis} ((x - y) \cdot b)^2) \neq 0$ 
  using assms by (force simp add: euclidean_eq_iff [of x y] sum_nonneg_eq_0_iff)

```

algebra_simps)

ultimately show *?thesis*

by auto

qed

lemma *Stone_Weierstrass_real_polynomial_function*:

fixes $f :: 'a::euclidean_space \Rightarrow \text{real}$

assumes *compact S continuous_on S f* $0 < e$

obtains g **where** *real_polynomial_function g* $\bigwedge x. x \in S \implies |f x - g x| < e$

proof –

interpret *PR: function_ring_on Collect real_polynomial_function*

proof *unfold_locales*

qed (*use assms continuous_on_polynomial_function real_polynomial_function_eq*

in *(auto intro: real_polynomial_function_separable)*)

show *?thesis*

using *PR.Stone_Weierstrass_basic [OF (continuous_on S f) (0 < e)]* **that by**

blast

qed

theorem *Stone_Weierstrass_polynomial_function*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes *S: compact S*

and *f: continuous_on S f*

and $e: 0 < e$

shows $\exists g. \text{polynomial_function } g \wedge (\forall x \in S. \text{norm}(f x - g x) < e)$

proof –

{ **fix** $b :: 'b$

assume $b \in \text{Basis}$

have $\exists p. \text{real_polynomial_function } p \wedge (\forall x \in S. |f x \cdot b - p x| < e / \text{DIM}('b))$

proof (*rule Stone_Weierstrass_real_polynomial_function [OF S _, of $\lambda x. f x \cdot b$ e / card Basis]*)

show *continuous_on S ($\lambda x. f x \cdot b$)*

using f **by** (*auto intro: continuous_intros*)

qed (*use e in auto*)

}

then obtain pf **where** pf :

$\bigwedge b. b \in \text{Basis} \implies \text{real_polynomial_function } (pf b) \wedge (\forall x \in S. |f x \cdot b - pf b x| < e / \text{DIM}('b))$

by *metis*

let $?g = \lambda x. \sum b \in \text{Basis}. pf b x *_R b$

{ **fix** x

assume $x \in S$

have $\text{norm} (\sum b \in \text{Basis}. (f x \cdot b) *_R b - pf b x *_R b) \leq (\sum b \in \text{Basis}. \text{norm} ((f x \cdot b) *_R b - pf b x *_R b))$

by (*rule norm_sum*)

also have $\dots < \text{of_nat } \text{DIM}('b) * (e / \text{DIM}('b))$

proof (*rule sum_bounded_above_strict*)

show $\bigwedge i. i \in \text{Basis} \implies \text{norm} ((f x \cdot i) *_R i - pf i x *_R i) < e / \text{real } \text{DIM}('b)$

```

    by (simp add: Real_Vector_Spaces.scaleR_diff_left [symmetric] pf ⟨x ∈ S⟩)
  qed (rule DIM_positive)
  also have ... = e
    by (simp add: field_simps)
  finally have norm (∑ b∈Basis. (f x · b) *R b - pf b x *R b) < e .
}
then have ∀ x∈S. norm ((∑ b∈Basis. (f x · b) *R b) - ?g x) < e
  by (auto simp flip: sum_subtractf)
moreover
have polynomial_function ?g
  using pf by (simp add: polynomial_function_sum polynomial_function_mult
real_polynomial_function_eq)
ultimately show ?thesis
  using euclidean_representation_sum_fun [of f] by (metis (no_types, lifting))
qed

```

proposition *Stone_Weierstrass_uniform_limit:*

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes S: compact S
  and f: continuous_on S f
obtains g where uniform_limit S g f sequentially ∧ n. polynomial_function (g
n)
proof -
  have pos: inverse (Suc n) > 0 for n by auto
  obtain g where g: ∧ n. polynomial_function (g n) ∧ x n. x ∈ S ⇒ norm(f x
- g n x) < inverse (Suc n)
  using Stone_Weierstrass_polynomial_function[OF S f pos]
  by metis
  have uniform_limit S g f sequentially
  proof (rule uniform_limitI)
    fix e::real assume 0 < e
    with LIMSEQ_inverse_real_of_nat have ∀F n in sequentially. inverse (Suc
n) < e
    by (rule order_tendstoD)
    moreover have ∀F n in sequentially. ∀ x∈S. dist (g n x) (f x) < inverse (Suc
n)
    using g by (simp add: dist_norm norm_minus_commute)
    ultimately show ∀F n in sequentially. ∀ x∈S. dist (g n x) (f x) < e
    by (eventually_elim) auto
  qed
  then show ?thesis using g(1) ..
qed

```

8.4.6 Polynomial functions as paths

One application is to pick a smooth approximation to a path, or just pick a smooth path anyway in an open connected set

lemma *path_polynomial_function:*

```

fixes g :: real ⇒ 'b::euclidean_space

```

shows *polynomial_function* $g \implies \text{path } g$
by (*simp add: path_def continuous_on_polynomial_function*)

lemma *path_approx_polynomial_function*:

fixes $g :: \text{real} \Rightarrow 'b::\text{euclidean_space}$

assumes $\text{path } g \ 0 < e$

obtains p **where** *polynomial_function* p $\text{pathstart } p = \text{pathstart } g$ $\text{pathfinish } p = \text{pathfinish } g$

$\bigwedge t. t \in \{0..1\} \implies \text{norm}(p \ t - g \ t) < e$

proof –

obtain q **where** *polynomial_function* q **and** $\text{noq}: \bigwedge x. x \in \{0..1\} \implies \text{norm}(g \ x - q \ x) < e/4$

using *Stone_Weierstrass_polynomial_function* [*of* $\{0..1\}$ $g \ e/4$] *assms*

by (*auto simp: path_def*)

define pf **where** $pf \equiv \lambda t. q \ t + (g \ 0 - q \ 0) + t *_R (g \ 1 - q \ 1 - (g \ 0 - q \ 0))$

show *thesis*

proof

show *polynomial_function* pf

by (*force simp add: poq pf_def*)

show $\text{norm}(pf \ t - g \ t) < e$

if $t \in \{0..1\}$ **for** t

proof –

have $*$: $\text{norm}(((q \ t - g \ t) + (g \ 0 - q \ 0)) + (t *_R (g \ 1 - q \ 1) + t *_R (q \ 0 - g \ 0))) < (e/4 + e/4) + (e/4 + e/4)$

proof (*intro Real_Vector_Spaces.norm_add_less*)

show $\text{norm}(q \ t - g \ t) < e / 4$

by (*metis noq norm_minus_commute that*)

show $\text{norm}(t *_R (g \ 1 - q \ 1)) < e / 4$

using *noq that le_less_trans [OF mult_left_le_one_le noq]*

by *auto*

show $\text{norm}(t *_R (q \ 0 - g \ 0)) < e / 4$

using *noq that le_less_trans [OF mult_left_le_one_le noq]*

by *simp (metis norm_minus_commute order_refl zero_le_one)*

qed (*use noq norm_minus_commute that in auto*)

then show *?thesis*

by (*auto simp add: algebra_simps pf_def*)

qed

qed (*auto simp add: path_defs pf_def*)

qed

proposition *connected_open_polynomial_connected*:

fixes $S :: 'a::\text{euclidean_space}$ *set*

assumes S : *open* S *connected* S

and $x \in S$ $y \in S$

shows $\exists g. \text{polynomial_function } g \wedge \text{path_image } g \subseteq S \wedge \text{pathstart } g = x \wedge \text{pathfinish } g = y$

proof –

have *path_connected* S **using** *assms*

by (*simp add: connected_open_path_connected*)

```

with  $\langle x \in S \rangle \langle y \in S \rangle$  obtain  $p$  where  $p: \text{path } p \text{ path\_image } p \subseteq S \text{ pathstart } p$ 
 $= x \text{ pathfinish } p = y$ 
  by (force simp: path_connected_def)
have  $\exists e. 0 < e \wedge (\forall x \in \text{path\_image } p. \text{ball } x \ e \subseteq S)$ 
proof (cases  $S = \text{UNIV}$ )
  case True then show ?thesis
    by (simp add: gt_ex)
next
  case False
show ?thesis
proof (intro exI conjI ballI)
  show  $\bigwedge x. x \in \text{path\_image } p \implies \text{ball } x \ (\text{setdist } (\text{path\_image } p) \ (-S)) \subseteq S$ 
    using setdist_le_dist [of _ path_image p _ -S] by fastforce
  show  $0 < \text{setdist } (\text{path\_image } p) \ (-S)$ 
    using S p False
  by (fastforce simp add: setdist_gt_0_compact_closed compact_path_image
open_closed)
  qed
qed
then obtain  $e$  where  $0 < e$  and  $eb: \bigwedge x. x \in \text{path\_image } p \implies \text{ball } x \ e \subseteq S$ 
  by auto
obtain  $pf$  where polynomial_function  $pf$  and  $pf: \text{pathstart } pf = \text{pathstart } p$ 
 $\text{pathfinish } pf = \text{pathfinish } p$ 
  and  $pf\_e: \bigwedge t. t \in \{0..1\} \implies \text{norm}(pf \ t - p \ t) < e$ 
  using path_approx_polynomial_function [OF  $\langle \text{path } p \rangle \langle 0 < e \rangle$ ] by blast
show ?thesis
proof (intro exI conjI)
  show polynomial_function  $pf$ 
    by fact
  show  $\text{pathstart } pf = x \ \text{pathfinish } pf = y$ 
    by (simp_all add: p pf)
  show  $\text{path\_image } pf \subseteq S$ 
    unfolding path_image_def
proof clarsimp
  fix  $x'::\text{real}$ 
  assume  $0 \leq x' \ x' \leq 1$ 
  then have  $\text{dist } (p \ x') \ (pf \ x') < e$ 
    by (metis atLeastAtMost_iff dist_commute dist_norm pf_e)
  then show  $pf \ x' \in S$ 
    by (metis  $\langle 0 \leq x' \rangle \langle x' \leq 1 \rangle$  atLeastAtMost_iff eb imageI mem_ball
path_image_def subset_iff)
  qed
qed
qed

lemma differentiable_componentwise_within:
   $f$  differentiable (at  $a$  within  $S$ )  $\iff$ 
   $(\forall i \in \text{Basis}. (\lambda x. f \ x \cdot i)$  differentiable at  $a$  within  $S$ )
proof -

```

```

{ assume  $\forall i \in \text{Basis}. \exists D. ((\lambda x. f x \cdot i) \text{ has\_derivative } D) \text{ (at } a \text{ within } S)$ 
  then obtain  $f'$  where  $f'$ :
     $\bigwedge i. i \in \text{Basis} \implies ((\lambda x. f x \cdot i) \text{ has\_derivative } f' i) \text{ (at } a \text{ within } S)$ 
  by metis
  have eq:  $(\lambda x. (\sum j \in \text{Basis}. f' j x *_{\mathbb{R}} j) \cdot i) = f' i$  if  $i \in \text{Basis}$  for  $i$ 
    using that by (simp add: inner_add_left inner_add_right)
  have  $\exists D. \forall i \in \text{Basis}. ((\lambda x. f x \cdot i) \text{ has\_derivative } (\lambda x. D x \cdot i)) \text{ (at } a \text{ within } S)$ 
    apply (rule_tac x= $\lambda x::'a. (\sum j \in \text{Basis}. f' j x *_{\mathbb{R}} j) :: 'b$  in exI)
    apply (simp add: eq f')
  done
}
then show ?thesis
  apply (simp add: differentiable_def)
  using has_derivative_componentwise_within
  by blast
qed

```

```

lemma polynomial_function_inner [intro]:
  fixes  $i :: 'a::\text{euclidean\_space}$ 
  shows  $\text{polynomial\_function } g \implies \text{polynomial\_function } (\lambda x. g x \cdot i)$ 
  apply (subst euclidean_representation [where  $x=i$ , symmetric])
  apply (force simp: inner_sum_right polynomial_function_iff_Basis_inner polynomial_function_sum)
  done

```

Differentiability of real and vector polynomial functions.

```

lemma differentiable_at_real_polynomial_function:
   $\text{real\_polynomial\_function } f \implies f \text{ differentiable (at } a \text{ within } S)$ 
  by (induction f rule: real_polynomial_function.induct)
    (simp_all add: bounded_linear_imp_differentiable)

```

```

lemma differentiable_on_real_polynomial_function:
   $\text{real\_polynomial\_function } p \implies p \text{ differentiable\_on } S$ 
  by (simp add: differentiable_at_imp_differentiable_on differentiable_at_real_polynomial_function)

```

```

lemma differentiable_at_polynomial_function:
  fixes  $f :: \_ \Rightarrow 'a::\text{euclidean\_space}$ 
  shows  $\text{polynomial\_function } f \implies f \text{ differentiable (at } a \text{ within } S)$ 
  by (metis differentiable_at_real_polynomial_function polynomial_function_iff_Basis_inner
    differentiable_componentwise_within)

```

```

lemma differentiable_on_polynomial_function:
  fixes  $f :: \_ \Rightarrow 'a::\text{euclidean\_space}$ 
  shows  $\text{polynomial\_function } f \implies f \text{ differentiable\_on } S$ 
  by (simp add: differentiable_at_polynomial_function differentiable_on_def)

```

```

lemma vector_eq_dot_span:
  assumes  $x \in \text{span } B$   $y \in \text{span } B$  and  $i: \bigwedge i. i \in B \implies i \cdot x = i \cdot y$ 
  shows  $x = y$ 

```

proof –
have $\bigwedge i. i \in B \implies \text{orthogonal } (x - y) i$
by (*simp add: i inner_commute inner_diff_right orthogonal_def*)
moreover have $x - y \in \text{span } B$
by (*simp add: assms span_diff*)
ultimately have $x - y = 0$
using *orthogonal_to_span orthogonal_self* **by** *blast*
then show *?thesis* **by** *simp*
qed

lemma *orthonormal_basis_expand*:
assumes B : *pairwise orthogonal B*
and 1 : $\bigwedge i. i \in B \implies \text{norm } i = 1$
and $x \in \text{span } B$
and *finite B*
shows $(\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = x$
proof (*rule vector_eq_dot_span [OF _ ⟨x ∈ span B⟩]*)
show $(\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) \in \text{span } B$
by (*simp add: span_clauses span_sum*)
show $i \cdot (\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = i \cdot x$ **if** $i \in B$ **for** i
proof –
have [*simp*]: $i \cdot j = (\text{if } j = i \text{ then } 1 \text{ else } 0)$ **if** $j \in B$ **for** j
using B 1 *that* $\langle i \in B \rangle$
by (*force simp: norm_eq_1 orthogonal_def pairwise_def*)
have $i \cdot (\sum_{i \in B}. (x \cdot i) *_{\mathbb{R}} i) = (\sum_{j \in B}. x \cdot j * (i \cdot j))$
by (*simp add: inner_sum_right*)
also have $\dots = (\sum_{j \in B}. \text{if } j = i \text{ then } x \cdot i \text{ else } 0)$
by (*rule sum.cong; simp*)
also have $\dots = i \cdot x$
by (*simp add: ⟨finite B⟩ that inner_commute*)
finally show *?thesis* .
qed
qed

theorem *Stone_Weierstrass_polynomial_function_subspace*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *compact S*
and *contf: continuous_on S f*
and $0 < e$
and *subspace T f ' S ⊆ T*
obtains g **where** *polynomial_function g g ' S ⊆ T*
 $\bigwedge x. x \in S \implies \text{norm}(f x - g x) < e$
proof –
obtain B **where** $B \subseteq T$ **and** *orthB: pairwise orthogonal B*
and $B1: \bigwedge x. x \in B \implies \text{norm } x = 1$
and *independent B and cardB: card B = dim T*
and *spanB: span B = T*
using *orthonormal_basis_subspace ⟨subspace T⟩* **by** *metis*


```

then have finite B
  by (simp add: independent_imp_finite)
then obtain n::nat and b where B = b ‘ {i. i < n} inj_on b {i. i < n}
  using finite_imp_nat_seg_image_inj_on by metis
with cardB have n = card B dim T = n
  by (auto simp: card_image)
have fx: (∑ i∈B. (f x · i) *R i) = f x if x ∈ S for x
  by (metis (no_types, lifting) B1 ‹finite B› assms(5) image_subset_iff orthB
  orthonormal_basis_expand spanB sum.cong that)
have cont: continuous_on S (λx. ∑ i∈B. (f x · i) *R i)
  by (intro continuous_intros contf)
obtain g where polynomial_function g
  and g: ∧x. x ∈ S ⇒ norm ((∑ i∈B. (f x · i) *R i) - g x) < e / (n+2)
  using Stone_Weierstrass_polynomial_function [OF ‹compact S› cont, of e /
  real (n + 2)] ‹0 < e›
  by auto
with fx have g: ∧x. x ∈ S ⇒ norm (f x - g x) < e / (n+2)
  by auto
show ?thesis
proof
  show polynomial_function (λx. ∑ i∈B. (g x · i) *R i)
    using ‹polynomial_function g› by (force intro: ‹finite B›)
  show (λx. ∑ i∈B. (g x · i) *R i) ‘ S ⊆ T
    using ‹B ⊆ T›
    by (blast intro: subspace_sum subspace_mul ‹subspace T›)
  show norm (f x - (∑ i∈B. (g x · i) *R i)) < e if x ∈ S for x
  proof -
    have orth': pairwise (λi j. orthogonal ((f x · i) *R i - (g x · i) *R i)
      ((f x · j) *R j - (g x · j) *R j)) B
      by (auto simp: orthogonal_def inner_diff_right inner_diff_left intro: pairwise_mono [OF orthB])
    then have (norm (∑ i∈B. (f x · i) *R i - (g x · i) *R i))2 =
      (∑ i∈B. (norm ((f x · i) *R i - (g x · i) *R i))2)
      by (simp add: norm_sum_Pythagorean [OF ‹finite B› orth'])
    also have ... = (∑ i∈B. (norm (((f x - g x) · i) *R i))2)
      by (simp add: algebra_simps)
    also have ... ≤ (∑ i∈B. (norm (f x - g x))2)
  proof -
    have ∧i. i ∈ B ⇒ ((f x - g x) · i)2 ≤ (norm (f x - g x))2
      by (metis B1 Cauchy_Schwarz_ineq inner_commute mult.left_neutral
      norm_eq_1 power2_norm_eq_inner)
    then show ?thesis
      by (intro sum_mono) (simp add: sum_mono B1)
  qed
  also have ... = n * norm (f x - g x)2
    by (simp add: ‹n = card B›)
  also have ... ≤ n * (e / (n+2))2
  proof (rule mult_left_mono)
    show (norm (f x - g x))2 ≤ (e / real (n + 2))2

```

```

    by (meson dual_order.order_iff_strict g norm_ge_zero power_mono that)
  qed auto
  also have ... ≤ e^2 / (n+2)
    using ‹0 < e› by (simp add: divide_simps power2_eq_square)
  also have ... < e^2
    using ‹0 < e› by (simp add: divide_simps)
  finally have (norm (∑ i∈B. (f x · i) *R i - (g x · i) *R i))^2 < e^2 .
  then have (norm (∑ i∈B. (f x · i) *R i - (g x · i) *R i)) < e
    by (simp add: ‹0 < e› norm_lt_square power2_norm_eq_inner)
  then show ?thesis
    using fx that by (simp add: sum_subtractf)
  qed
qed
qed

```

```
hide_fact linear add mult const
```

```
end
```

8.5 Radon-Nikodým Derivative

```

theory Radon_Nikodym
imports Bochner_Integration
begin

```

```
definition diff_measure :: 'a measure ⇒ 'a measure ⇒ 'a measure
```

```
where
```

```
diff_measure M N = measure_of (space M) (sets M) (λA. emeasure M A -
emeasure N A)
```

```
lemma
```

```

shows space_diff_measure[simp]: space (diff_measure M N) = space M
and sets_diff_measure[simp]: sets (diff_measure M N) = sets M
by (auto simp: diff_measure_def)

```

```
lemma emeasure_diff_measure:
```

```

assumes fin: finite_measure M finite_measure N and sets_eq: sets M = sets N
assumes pos: ⋀A. A ∈ sets M ⇒ emeasure N A ≤ emeasure M A and A: A
∈ sets M

```

```

shows emeasure (diff_measure M N) A = emeasure M A - emeasure N A (is
_ = ?μ A)

```

```
unfolding diff_measure_def
```

```
proof (rule emeasure_measure_of_sigma)
```

```
show sigma_algebra (space M) (sets M) ..
```

```
show positive (sets M) ?μ
```

```
using pos by (simp add: positive_def)
```

```
show countably_additive (sets M) ?μ
```

```
proof (rule countably_additiveI)
```

```

fix A :: nat  $\Rightarrow$  _ assume A: range A  $\subseteq$  sets M and disjoint_family A
then have suminf:
  ( $\sum$  i. emeasure M (A i)) = emeasure M ( $\bigcup$  i. A i)
  ( $\sum$  i. emeasure N (A i)) = emeasure N ( $\bigcup$  i. A i)
  by (simp_all add: suminf_emeasure sets_eq)
with A have ( $\sum$  i. emeasure M (A i) - emeasure N (A i)) =
  ( $\sum$  i. emeasure M (A i)) - ( $\sum$  i. emeasure N (A i))
  using fin_pos[of A _]
  by (intro ennreal_suminf_minus)
  (auto simp: sets_eq finite_measure.emeasure_eq_measure suminf_emeasure)
then show ( $\sum$  i. emeasure M (A i) - emeasure N (A i)) =
  emeasure M ( $\bigcup$  i. A i) - emeasure N ( $\bigcup$  i. A i)
  by (simp add: suminf)
qed
qed fact

```

An equivalent characterization of sigma-finite spaces is the existence of integrable positive functions (or, still equivalently, the existence of a probability measure which is in the same measure class as the original measure).

proposition (in *sigma_finite_measure*) *obtain_positive_integrable_function*:

obtains $f::'a \Rightarrow \text{real}$ **where**

$f \in \text{borel_measurable } M$

$\bigwedge x. f\ x > 0$

$\bigwedge x. f\ x \leq 1$

integrable M f

proof -

obtain A :: nat \Rightarrow 'a set **where** range A \subseteq sets M (\bigcup i. A i) = space M \bigwedge i.
emeasure M (A i) $\neq \infty$

using *sigma_finite* **by** *auto*

then have [*measurable*]: A n \in sets M **for** n **by** *auto*

define g **where** $g = (\lambda x. (\sum n. (1/2)^{\wedge}(\text{Suc } n) * \text{indicator } (A\ n)\ x / (1 + \text{measure } M\ (A\ n))))$

have [*measurable*]: g \in borel_measurable M **unfolding** g_def **by** *auto*

have *: *summable* ($\lambda n. (1/2)^{\wedge}(\text{Suc } n) * \text{indicator } (A\ n)\ x / (1 + \text{measure } M\ (A\ n))$) **for** x

apply (*rule summable_comparison_test'[of $\lambda n. (1/2)^{\wedge}(\text{Suc } n)\ 0$]*)

using *power_half_series summable_def* **by** (*auto simp add: indicator_def divide_simps*)

have $g\ x \leq (\sum n. (1/2)^{\wedge}(\text{Suc } n))$ **for** x **unfolding** g_def

apply (*rule suminf_le*) **using** * *power_half_series summable_def* **by** (*auto simp add: indicator_def divide_simps*)

then have g_le_1: $g\ x \leq 1$ **for** x **using** *power_half_series sums_unique* **by** *fastforce*

have g_pos: $g\ x > 0$ **if** x \in space M **for** x

unfolding g_def **proof** (*subst suminf_pos_iff[OF *[of x], auto*)

obtain i **where** x \in A i **using** $\langle \bigcup i. A\ i \rangle = \text{space } M \rangle \langle x \in \text{space } M \rangle$ **by** *auto*

then have $0 < (1/2)^{\wedge} \text{Suc } i * \text{indicator } (A\ i)\ x / (1 + \text{Sigma_Algebra.measure } M\ (A\ i))$

unfolding *indicator_def* **apply** (*auto simp add: divide_simps*) **using** *measure_nonneg[of M A i]*
by (*auto, meson add_nonneg_nonneg linorder_not_le mult_nonneg_nonneg zero_le_numeral zero_le_one zero_le_power*)
then show $\exists i. 0 < (1/2)^i * \text{indicator } (A\ i) \ x / (2 + 2 * \text{Sigma_Algebra.measure } M\ (A\ i))$
by auto
qed

have *integrable M g*
unfolding *g_def* **proof** (*rule integrable_suminf*)
fix *n*
show *integrable M* ($\lambda x. (1/2)^{\text{Suc } n} * \text{indicator } (A\ n) \ x / (1 + \text{Sigma_Algebra.measure } M\ (A\ n))$)
using $\langle \text{emeasure } M\ (A\ n) \neq \infty \rangle$
by (*auto intro!: integrable_mult_right integrable_divide_zero integrable_real_indicator simp add: top.not_eq_extremum*)
next
show *AE x in M. summable* ($\lambda n. \text{norm } ((1/2)^{\text{Suc } n} * \text{indicator } (A\ n) \ x / (1 + \text{Sigma_Algebra.measure } M\ (A\ n)))$)
using * **by auto**
show *summable* ($\lambda n. (\int x. \text{norm } ((1/2)^{\text{Suc } n} * \text{indicator } (A\ n) \ x / (1 + \text{Sigma_Algebra.measure } M\ (A\ n))) \ \partial M)$)
apply (*rule summable_comparison_test'[of $\lambda n. (1/2)^{\text{Suc } n} \ 0$], auto*)
using *power_half_series summable_def* **apply** *auto[1]*
apply (*auto simp add: field_split_simps*) **using** *measure_nonneg[of M]*
not_less **by fastforce**
qed

define *f* **where** $f = (\lambda x. \text{if } x \in \text{space } M \text{ then } g\ x \text{ else } 1)$
have $f\ x > 0$ **for** *x* **unfolding** *f_def* **using** *g_pos* **by auto**
moreover **have** $f\ x \leq 1$ **for** *x* **unfolding** *f_def* **using** *g_le_1* **by auto**
moreover **have** [*measurable*]: $f \in \text{borel_measurable } M$ **unfolding** *f_def* **by auto**
moreover **have** *integrable M f*
apply (*subst integrable_cong[of _ _ _ g]*) **unfolding** *f_def* **using** $\langle \text{integrable } M\ g \rangle$ **by auto**
ultimately show $(\bigwedge f. f \in \text{borel_measurable } M \implies (\bigwedge x. 0 < f\ x) \implies (\bigwedge x. f\ x \leq 1) \implies \text{integrable } M\ f \implies \text{thesis}) \implies \text{thesis}$
by (*meson that*)
qed

lemma (*in sigma_finite_measure*) *Ex_finite_integrable_function*:
 $\exists h \in \text{borel_measurable } M. \text{integral}^N\ M\ h \neq \infty \wedge (\forall x \in \text{space } M. 0 < h\ x \wedge h\ x < \infty)$

proof –

obtain *A* :: *nat* \Rightarrow 'a **set** **where**
range[*measurable*]: $\text{range } A \subseteq \text{sets } M$ **and**
space: $(\bigcup i. A\ i) = \text{space } M$ **and**
measure: $\bigwedge i. \text{emeasure } M\ (A\ i) \neq \infty$ **and**

```

    disjoint: disjoint_family A
    using sigma_finite_disjoint by blast
  let ?B =  $\lambda i. 2^{\wedge} \text{Suc } i * \text{emeasure } M (A i)$ 
  have [measurable]:  $\bigwedge i. A i \in \text{sets } M$ 
    using range by fastforce+
  have  $\forall i. \exists x. 0 < x \wedge x < \text{inverse } (?B i)$ 
  proof
    fix i show  $\exists x. 0 < x \wedge x < \text{inverse } (?B i)$ 
      using measure[of i]
    by (auto intro!: dense simp: ennreal_inverse_positive ennreal_mult_eq_top_iff
power_eq_top_ennreal)
  qed
  from choice[OF this] obtain n where  $n: \bigwedge i. 0 < n i$ 
     $\bigwedge i. n i < \text{inverse } (2^{\wedge} \text{Suc } i * \text{emeasure } M (A i))$  by auto
  { fix i have  $0 \leq n i$  using n(1)[of i] by auto } note pos = this
  let ?h =  $\lambda x. \sum i. n i * \text{indicator } (A i) x$ 
  show ?thesis
  proof (safe intro!: beXI[of _ ?h] del: notI)
    have  $\text{integral}^N M ?h = (\sum i. n i * \text{emeasure } M (A i))$  using pos
      by (simp add: nn_integral_suminf nn_integral_cmult_indicator)
    also have  $\dots \leq (\sum i. \text{ennreal } ((1/2)^{\wedge} \text{Suc } i))$ 
  proof (intro suminf_le allI)
    fix N
    have  $n N * \text{emeasure } M (A N) \leq \text{inverse } (2^{\wedge} \text{Suc } N * \text{emeasure } M (A N))$ 
      *  $\text{emeasure } M (A N)$ 
      using n[of N] by (intro mult_right_mono) auto
    also have  $\dots = (1/2)^{\wedge} \text{Suc } N * (\text{inverse } (\text{emeasure } M (A N)) * \text{emeasure } M (A N))$ 
      using measure[of N]
    by (simp add: ennreal_inverse_power divide_ennreal_def ennreal_inverse_mult
power_eq_top_ennreal less_top[symmetric] mult_ac
del: power_Suc)
    also have  $\dots \leq \text{inverse } (\text{ennreal } 2)^{\wedge} \text{Suc } N$ 
      using measure[of N]
    by (cases emeasure M (A N); cases emeasure M (A N) = 0)
      (auto simp: inverse_ennreal ennreal_mult[symmetric] divide_ennreal_def
simp del: power_Suc)
    also have  $\dots = \text{ennreal } (\text{inverse } 2)^{\wedge} \text{Suc } N$ 
      by (subst ennreal_power[symmetric], simp) (simp add: inverse_ennreal)
    finally show  $n N * \text{emeasure } M (A N) \leq \text{ennreal } ((1/2)^{\wedge} \text{Suc } N)$ 
      by simp
  qed auto
  also have  $\dots < \text{top}$ 
    unfolding less_top[symmetric]
    by (rule ennreal_suminf_neq_top)
      (auto simp: summable_geometric summable_Suc_iff simp del: power_Suc)
  finally show  $\text{integral}^N M ?h \neq \infty$ 
    by (auto simp: top_unique)
  next

```

```

{ fix x assume x ∈ space M
  then obtain i where x ∈ A i using space[symmetric] by auto
  with disjoint n have ?h x = n i
  by (auto intro!: suminf_cmult_indicator intro: less_imp_le)
  then show 0 < ?h x and ?h x < ∞ using n[of i] by (auto simp: less_top[symmetric])
}
note pos = this
qed measurable
qed

```

8.5.1 Absolutely continuous

definition *absolutely_continuous* :: 'a measure ⇒ 'a measure ⇒ bool **where**
absolutely_continuous M N ⇔ null_sets M ⊆ null_sets N

lemma *absolutely_continuousI_count_space*: *absolutely_continuous* (count_space A) M

unfolding *absolutely_continuous_def* **by** (auto simp: null_sets_count_space)

lemma *absolutely_continuousI_density*:

$f \in \text{borel_measurable } M \implies \text{absolutely_continuous } M \text{ (density } M f)$

by (force simp add: *absolutely_continuous_def* null_sets_density_iff dest: AE_not_in)

lemma *absolutely_continuousI_point_measure_finite*:

$(\bigwedge x. \llbracket x \in A ; f x \leq 0 \rrbracket \implies g x \leq 0) \implies \text{absolutely_continuous} \text{ (point_measure } A f) \text{ (point_measure } A g)$

unfolding *absolutely_continuous_def* **by** (force simp: null_sets_point_measure_iff)

lemma *absolutely_continuousD*:

$\text{absolutely_continuous } M N \implies A \in \text{sets } M \implies \text{emeasure } M A = 0 \implies \text{emeasure } N A = 0$

by (auto simp: *absolutely_continuous_def* null_sets_def)

lemma *absolutely_continuous_AE*:

assumes *sets_eq*: sets M' = sets M

and *absolutely_continuous* M M' AE x in M. P x

shows AE x in M'. P x

proof –

from ⟨AE x in M. P x⟩ **obtain** N **where** N: N ∈ null_sets M {x ∈ space M. ¬ P x} ⊆ N

unfolding *eventually_ae_filter* **by** auto

show AE x in M'. P x

proof (rule AE_I')

show {x ∈ space M'. ¬ P x} ⊆ N **using** *sets_eq_imp_space_eq*[OF *sets_eq*] N(2) **by** simp

from ⟨*absolutely_continuous* M M'⟩ **show** N ∈ null_sets M'

using N **unfolding** *absolutely_continuous_def* *sets_eq* *null_sets_def* **by** auto

qed

qed

8.5.2 Existence of the Radon-Nikodym derivative

proposition

```
(in finite_measure) Radon_Nikodym_finite_measure:
  assumes finite_measure N and sets_eq[simp]: sets N = sets M
  assumes absolutely_continuous M N
  shows  $\exists f \in \text{borel\_measurable } M. \text{density } M f = N$ 
proof -
  interpret N: finite_measure N by fact
  define G where  $G = \{g \in \text{borel\_measurable } M. \forall A \in \text{sets } M. (\int^+ x. g x * \text{indicator } A x \partial M) \leq N A\}$ 
  have [measurable_dest]:  $f \in G \implies f \in \text{borel\_measurable } M$ 
  and G_D:  $\bigwedge A. f \in G \implies A \in \text{sets } M \implies (\int^+ x. f x * \text{indicator } A x \partial M) \leq N A$  for f
  by (auto simp: G_def)
  note this[measurable_dest]
  have  $(\lambda x. 0) \in G$  unfolding G_def by auto
  hence  $G \neq \{\}$  by auto
  { fix f g assume f[measurable]:  $f \in G$  and g[measurable]:  $g \in G$ 
    have  $(\lambda x. \max (g x) (f x)) \in G$  (is ?max  $\in G$ ) unfolding G_def
    proof safe
      let ?A =  $\{x \in \text{space } M. f x \leq g x\}$ 
      have ?A  $\in \text{sets } M$  using f g unfolding G_def by auto
      fix A assume [measurable]:  $A \in \text{sets } M$ 
      have union:  $((?A \cap A) \cup ((\text{space } M - ?A) \cap A)) = A$ 
        using sets.sets_into_space[OF  $\langle A \in \text{sets } M \rangle$ ] by auto
      have  $\bigwedge x. x \in \text{space } M \implies \max (g x) (f x) * \text{indicator } A x = g x * \text{indicator } (?A \cap A) x + f x * \text{indicator } ((\text{space } M - ?A) \cap A) x$ 
        by (auto simp: indicator_def max_def)
      hence  $(\int^+ x. \max (g x) (f x) * \text{indicator } A x \partial M) = (\int^+ x. g x * \text{indicator } (?A \cap A) x \partial M) + (\int^+ x. f x * \text{indicator } ((\text{space } M - ?A) \cap A) x \partial M)$ 
        by (auto cong: nn_integral_cong intro!: nn_integral_add)
      also have  $\dots \leq N (?A \cap A) + N ((\text{space } M - ?A) \cap A)$ 
        using f g unfolding G_def by (auto intro!: add_mono)
      also have  $\dots = N A$ 
        using union by (subst plus_emeasure) auto
      finally show  $(\int^+ x. \max (g x) (f x) * \text{indicator } A x \partial M) \leq N A$  .
    qed auto }
  note max_in_G = this
  { fix f assume incseq f and f:  $\bigwedge i. f i \in G$ 
    then have [measurable]:  $\bigwedge i. f i \in \text{borel\_measurable } M$  by (auto simp: G_def)
    have  $(\lambda x. \text{SUP } i. f i x) \in G$  unfolding G_def
    proof safe
      show  $(\lambda x. \text{SUP } i. f i x) \in \text{borel\_measurable } M$  by measurable
    next
      fix A assume  $A \in \text{sets } M$ 
```

```

have ( $\int^{+x}. (SUP\ i.\ f\ i\ x) * indicator\ A\ x\ \partial M$ ) =
  ( $\int^{+x}. (SUP\ i.\ f\ i\ x * indicator\ A\ x)\ \partial M$ )
  by (intro nn_integral_cong) (simp split: split_indicator)
also have ... = ( $SUP\ i.\ (\int^{+x}. f\ i\ x * indicator\ A\ x\ \partial M)$ )
  using  $\langle incseq\ f \rangle f\ \langle A \in\ sets\ M \rangle$ 
  by (intro nn_integral_monotone_convergence_SUP)
  (auto simp: G_def incseq_Suc_iff le_fun_def split: split_indicator)
finally show ( $\int^{+x}. (SUP\ i.\ f\ i\ x) * indicator\ A\ x\ \partial M$ )  $\leq N\ A$ 
  using  $f\ \langle A \in\ sets\ M \rangle$  by (auto intro!: SUP_least simp: G_D)
qed }
note SUP_in_G = this
let  $?y = SUP\ g \in\ G.\ integral^N\ M\ g$ 
have  $y\_le: ?y \leq N\ (space\ M)$  unfolding G_def
proof (safe intro!: SUP_least)
  fix  $g$  assume  $\forall A \in\ sets\ M.\ (\int^{+x}. g\ x * indicator\ A\ x\ \partial M) \leq N\ A$ 
  from this[THEN bspec, OF sets.top] show  $integral^N\ M\ g \leq N\ (space\ M)$ 
  by (simp cong: nn_integral_cong)
qed
from ennreal_SUP_countable_SUP [OF  $\langle G \neq \{\} \rangle$ , of integral^N M]
obtain  $ys :: nat \Rightarrow ennreal$ 
  where  $ys: range\ ys \subseteq integral^N\ M\ 'G \wedge Sup\ (integral^N\ M\ 'G) = Sup\ (range\ ys)$ 
  by auto
then have  $\forall n.\ \exists g.\ g \in G \wedge integral^N\ M\ g = ys\ n$ 
proof safe
  fix  $n$  assume  $range\ ys \subseteq integral^N\ M\ 'G$ 
  hence  $ys\ n \in integral^N\ M\ 'G$  by auto
  thus  $\exists g.\ g \in G \wedge integral^N\ M\ g = ys\ n$  by auto
qed
from choice[OF this] obtain  $gs$  where  $\bigwedge i.\ gs\ i \in G \wedge \bigwedge n.\ integral^N\ M\ (gs\ n) =$ 
 $ys\ n$  by auto
hence  $y\_eq: ?y = (SUP\ i.\ integral^N\ M\ (gs\ i))$  using  $ys$  by auto
let  $?g = \lambda i\ x.\ Max\ ((\lambda n.\ gs\ n\ x)\ ' \{..i\})$ 
define  $f$  where [abs_def]:  $f\ x = (SUP\ i.\ ?g\ i\ x)$  for  $x$ 
let  $?F = \lambda A\ x.\ f\ x * indicator\ A\ x$ 
have  $gs\_not\_empty: \bigwedge i\ x.\ (\lambda n.\ gs\ n\ x)\ ' \{..i\} \neq \{\}$  by auto
{ fix  $i$  have  $?g\ i \in G$ 
  proof (induct i)
    case 0 thus  $?case$  by simp fact
  next
    case (Suc i)
    with  $Suc\ gs\_not\_empty\ \langle gs\ (Suc\ i) \in G \rangle$  show  $?case$ 
    by (auto simp add: atMost_Suc intro!: max_in_G)
  qed }
note  $g\_in\_G = this$ 
have  $incseq\ ?g$  using  $gs\_not\_empty$ 
  by (auto intro!: incseq_SucI le_funI simp add: atMost_Suc)

from SUP_in_G[OF this g_in_G] have [measurable]:  $f \in G$  unfolding f_def

```



```

then have [measurable]:  $f \in \text{borel\_measurable } M$  unfolding  $G\_def$  by auto

have  $\text{integral}^N M f = (\text{SUP } i. \text{integral}^N M (?g \ i))$  unfolding  $f\_def$ 
using  $g\_in\_G \langle \text{incseq } ?g \rangle$  by (auto intro!:  $\text{nn\_integral\_monotone\_convergence\_SUP}$ 
simp:  $G\_def$ )
also have  $\dots = ?y$ 
proof (rule antisym)
  show  $(\text{SUP } i. \text{integral}^N M (?g \ i)) \leq ?y$ 
    using  $g\_in\_G$  by (auto intro!:  $\text{SUP\_mono}$ )
  show  $?y \leq (\text{SUP } i. \text{integral}^N M (?g \ i))$  unfolding  $y\_eq$ 
    by (auto intro!:  $\text{SUP\_mono } \text{nn\_integral\_mono } \text{Max\_ge}$ )
qed
finally have  $\text{int\_f\_eq\_y}: \text{integral}^N M f = ?y$  .

have  $\text{upper\_bound}: \forall A \in \text{sets } M. N \ A \leq \text{density } M \ f \ A$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain  $A$  where  $A[\text{measurable}]: A \in \text{sets } M$  and  $f\_less\_N: \text{density } M \ f$ 
 $A < N \ A$ 
    by (auto simp!:  $\text{not\_le}$ )
  then have  $\text{pos\_A}: 0 < M \ A$ 
    using  $\langle \text{absolutely\_continuous } M \ N \rangle$  [ $\text{THEN } \text{absolutely\_continuousD}, \text{OF } A$ ]
    by (auto simp!:  $\text{zero\_less\_iff\_neq\_zero}$ )

  define  $b$  where  $b = (N \ A - \text{density } M \ f \ A) / M \ A / 2$ 
  with  $f\_less\_N \ \text{pos\_A}$  have  $0 < b \ b \neq \text{top}$ 
  by (auto intro!:  $\text{diff\_gr0\_ennreal } \text{simp!}: \text{zero\_less\_iff\_neq\_zero } \text{diff\_eq\_0\_iff\_ennreal}$ 
 $\text{ennreal\_divide\_eq\_top\_iff}$ )

  let  $?f = \lambda x. f \ x + b$ 
  have  $\text{nn\_integral } M \ f \neq \text{top}$ 
    using  $\langle f \in G \rangle$  [ $\text{THEN } G\_D, \text{of } \text{space } M$ ] by (auto simp!:  $\text{top\_unique } \text{cong!}:$ 
 $\text{nn\_integral\_cong}$ )
  with  $\langle b \neq \text{top} \rangle$  interpret  $Mf: \text{finite\_measure } \text{density } M \ ?f$ 
    by (intro  $\text{finite\_measureI}$ )
    (auto simp!:  $\text{field\_simps } \text{mult\_indicator\_subset } \text{ennreal\_mult\_eq\_top\_iff}$ 
 $\text{emeasure\_density } \text{nn\_integral\_cmult\_indicator } \text{nn\_integral\_add}$ 
 $\text{cong!}: \text{nn\_integral\_cong}$ )

  from  $\text{unsigned\_Hahn\_decomposition}$  [ $\text{of } \text{density } M \ ?f \ N \ A$ ]
  obtain  $Y$  where  $[\text{measurable}]: Y \in \text{sets } M$  and  $[\text{simp}]: Y \subseteq A$ 
    and  $Y1: \bigwedge C. C \in \text{sets } M \implies C \subseteq Y \implies \text{density } M \ ?f \ C \leq N \ C$ 
    and  $Y2: \bigwedge C. C \in \text{sets } M \implies C \subseteq A \implies C \cap Y = \{\} \implies N \ C \leq \text{density}$ 
 $M \ ?f \ C$ 
    by auto

  let  $?f' = \lambda x. f \ x + b * \text{indicator } Y \ x$ 
  have  $M \ Y \neq 0$ 

```

```

proof
  assume  $M \ Y = 0$ 
  then have  $N \ Y = 0$ 
    using  $\langle \text{absolutely\_continuous } M \ N \rangle [ \text{THEN absolutely\_continuous } D, \text{ of } Y ]$ 
by auto
  then have  $N \ A = N \ (A - Y)$ 
    by (subst emeasure_Diff) auto
  also have  $\dots \leq \text{density } M \ ?f \ (A - Y)$ 
    by (rule Y2) auto
  also have  $\dots \leq \text{density } M \ ?f \ A - \text{density } M \ ?f \ Y$ 
    by (subst emeasure_Diff) auto
  also have  $\dots \leq \text{density } M \ ?f \ A - 0$ 
    by (intro ennreal_minus_mono) auto
  also have  $\text{density } M \ ?f \ A = b * M \ A + \text{density } M \ f \ A$ 
    by (simp add: emeasure_density_field_simps mult_indicator_subset nn_integral_add
nn_integral_cmult_indicator)
  also have  $\dots < N \ A$ 
    using f_less_N_pos_A
    by (cases density M f A; cases M A; cases N A)
      (auto simp: b_def ennreal_less_iff ennreal_minus divide_ennreal en-
nnreal_numeral[symmetric]
        ennreal_plus[symmetric] ennreal_mult[symmetric] field_simps
        simp del: ennreal_numeral ennreal_plus)
  finally show False
    by simp
qed
then have  $\text{nn\_integral } M \ f < \text{nn\_integral } M \ ?f'$ 
  using  $\langle 0 < b \rangle \langle \text{nn\_integral } M \ f \neq \text{top} \rangle$ 
  by (simp add: nn_integral_add nn_integral_cmult_indicator ennreal_zero_less_mult_iff
zero_less_iff_neq_zero)
moreover
  have  $?f' \in G$ 
    unfolding G_def
  proof safe
    fix  $X$  assume [measurable]:  $X \in \text{sets } M$ 
    have  $(\int^+ x. ?f' \ x * \text{indicator } X \ x \ \partial M) = \text{density } M \ f \ (X - Y) + \text{density}$ 
 $M \ ?f \ (X \cap Y)$ 
      by (auto simp add: emeasure_density nn_integral_add[symmetric] split:
split_indicator intro!: nn_integral_cong)
    also have  $\dots \leq N \ (X - Y) + N \ (X \cap Y)$ 
      using G_D[OF  $\langle f \in G \rangle$ ] by (intro add_mono Y1) (auto simp: emea-
sure_density)
    also have  $\dots = N \ X$ 
      by (subst plus_emeasure) (auto intro!: arg_cong2[where f=emeasure])
    finally show  $(\int^+ x. ?f' \ x * \text{indicator } X \ x \ \partial M) \leq N \ X .$ 
  qed simp
then have  $\text{nn\_integral } M \ ?f' \leq ?y$ 
  by (rule SUP_upper)
ultimately show False

```

```

    by (simp add: int_f_eq_y)
  qed
  show ?thesis
  proof (intro bexI[of _ f] measure_eqI conjI antisym)
    fix A assume A ∈ sets (density M f) then show emeasure (density M f) A ≤
emeasure N A
    by (auto simp: emeasure_density intro!: G_D[OF ⟨f ∈ G⟩])
  next
    fix A assume A: A ∈ sets (density M f) then show emeasure N A ≤ emeasure
(density M f) A
    using upper_bound by auto
  qed auto
qed

```

lemma (in *finite_measure*) *split_space_into_finite_sets_and_rest*:

```

  assumes ac: absolutely_continuous M N and sets_eq[simp]: sets N = sets M
  shows ∃ B::nat ⇒ 'a set. disjoint_family B ∧ range B ⊆ sets M ∧ (∀ i. N (B i)
≠ ∞) ∧
  (∀ A ∈ sets M. A ∩ (⋃ i. B i) = {} → (emeasure M A = 0 ∧ N A = 0) ∨
(emeasure M A > 0 ∧ N A = ∞))
  proof -
    let ?Q = {Q ∈ sets M. N Q ≠ ∞}
    let ?a = SUP Q ∈ ?Q. emeasure M Q
    have {} ∈ ?Q by auto
    then have Q_not_empty: ?Q ≠ {} by blast
    have ?a ≤ emeasure M (space M) using sets_sets_into_space
    by (auto intro!: SUP_least emeasure_mono)
    then have ?a ≠ ∞
    using finite_emeasure_space
    by (auto simp: less_top[symmetric] top_unique simp del: SUP_eq_top_iff
Sup_eq_top_iff)
    from ennreal_SUP_countable_SUP [OF Q_not_empty, of emeasure M]
    obtain Q'' where range Q'' ⊆ emeasure M ' ?Q and a: ?a = (SUP i::nat. Q''
i)
    by auto
    then have ∀ i. ∃ Q'. Q'' i = emeasure M Q' ∧ Q' ∈ ?Q by auto
    from choice[OF this] obtain Q' where Q': ∧ i. Q'' i = emeasure M (Q' i) ∧ i.
Q' i ∈ ?Q
    by auto
    then have a_Lim: ?a = (SUP i. emeasure M (Q' i)) using a by simp
    let ?O = λ n. ⋃ i ≤ n. Q' i
    have Union: (SUP i. emeasure M (?O i)) = emeasure M (⋃ i. ?O i)
    proof (rule SUP_emeasure_incseq[of ?O])
      show range ?O ⊆ sets M using Q' by auto
      show incseq ?O by (fastforce intro!: incseq_SucI)
    qed
    have Q'_sets[measurable]: ∧ i. Q' i ∈ sets M using Q' by auto
    have O_sets: ∧ i. ?O i ∈ sets M using Q' by auto
    then have O_in_G: ∧ i. ?O i ∈ ?Q

```

```

proof (safe del: notI)
  fix  $i$  have  $Q' \text{ ' } \{..i\} \subseteq \text{sets } M$  using  $Q'$  by auto
  then have  $N (?O i) \leq (\sum i \leq i. N (Q' i))$ 
    by (simp add: emeasure_subadditive_finite)
  also have  $\dots < \infty$  using  $Q'$  by (simp add: less_top)
  finally show  $N (?O i) \neq \infty$  by simp
qed auto
have  $O\_mono: \bigwedge n. ?O n \subseteq ?O (Suc n)$  by fastforce
have  $a\_eq: ?a = \text{emeasure } M (\bigcup i. ?O i)$  unfolding Union[symmetric]
proof (rule antisym)
  show  $?a \leq (SUP i. \text{emeasure } M (?O i))$  unfolding  $a\_Lim$ 
    using  $Q'$  by (auto intro!: SUP_mono emeasure_mono)
  show  $(SUP i. \text{emeasure } M (?O i)) \leq ?a$ 
proof (safe intro!: Sup_mono, unfold bex_simps)
  fix  $i$ 
  have  $*$ :  $(\bigcup (Q' \text{ ' } \{..i\})) = ?O i$  by auto
  then show  $\exists x. (x \in \text{sets } M \wedge N x \neq \infty) \wedge$ 
     $\text{emeasure } M (\bigcup (Q' \text{ ' } \{..i\})) \leq \text{emeasure } M x$ 
    using  $O\_in\_G[\text{of } i]$  by (auto intro!: exI[of _ ?O i])
qed
qed
let  $?O\_0 = (\bigcup i. ?O i)$ 
have  $?O\_0 \in \text{sets } M$  using  $Q'$  by auto
have disjointed  $Q' i \in \text{sets } M$  for  $i$ 
  using sets.range_disjointed_sets[of  $Q' M$ ] using  $Q'\_sets$  by (auto simp: sub-set_eq)
note  $Q\_sets = \text{this}$ 
show ?thesis
proof (intro bexI exI conjI ballI impI allI)
  show disjoint_family (disjointed  $Q'$ )
    by (rule disjoint_family_disjointed)
  show range (disjointed  $Q'$ )  $\subseteq \text{sets } M$ 
    using  $Q'\_sets$  by (intro sets.range_disjointed_sets) auto
  { fix  $A$  assume  $A: A \in \text{sets } M \wedge A \cap (\bigcup i. \text{disjointed } Q' i) = \{\}$ 
    then have  $A1: A \cap (\bigcup i. Q' i) = \{\}$ 
      unfolding UN_disjointed_eq by auto
    show  $\text{emeasure } M A = 0 \wedge N A = 0 \vee 0 < \text{emeasure } M A \wedge N A = \infty$ 
proof (rule disjCI, simp)
      assume  $*$ :  $\text{emeasure } M A = 0 \vee N A \neq \text{top}$ 
      show  $\text{emeasure } M A = 0 \wedge N A = 0$ 
proof (cases emeasure M A = 0)
      case True
      with  $ac A$  have  $N A = 0$ 
        unfolding absolutely_continuous_def by auto
      with True show ?thesis by simp
next
      case False
      with  $*$  have  $N A \neq \infty$  by auto
      with  $A$  have  $\text{emeasure } M ?O\_0 + \text{emeasure } M A = \text{emeasure } M (?O\_0$ 

```

```

     $\cup A$ )
      using  $Q' A1$  by (auto intro!: plus_emeasure simp: set_eq_iff)
      also have ... = (SUP  $i$ . emeasure  $M$  ( $?O i \cup A$ ))
    proof (rule SUP_emeasure_incseq[of  $\lambda i. ?O i \cup A$ , symmetric, simplified])
      show range ( $\lambda i. ?O i \cup A$ )  $\subseteq$  sets  $M$ 
        using  $\langle N A \neq \infty \rangle$  O_sets  $A$  by auto
      qed (fastforce intro!: incseq_SucI)
      also have ...  $\leq ?a$ 
    proof (safe intro!: SUP_least)
      fix  $i$  have  $?O i \cup A \in ?Q$ 
    proof (safe del: notI)
      show  $?O i \cup A \in$  sets  $M$  using O_sets  $A$  by auto
      from O_in_G[of  $i$ ] have  $N (?O i \cup A) \leq N (?O i) + N A$ 
        using emeasure_subadditive[of  $?O i N A$ ] A O_sets by auto
      with O_in_G[of  $i$ ] show  $N (?O i \cup A) \neq \infty$ 
        using  $\langle N A \neq \infty \rangle$  by (auto simp: top_unique)
      qed
    then show emeasure  $M$  ( $?O i \cup A$ )  $\leq ?a$  by (rule SUP_upper)
    qed
  finally have emeasure  $M A = 0$ 
    unfolding a_eq using measure_nonneg[of  $M A$ ] by (simp add: emeasure_eq_measure)
  with  $\langle$ emeasure  $M A \neq 0$  $\rangle$  show ?thesis by auto
  qed
qed }
{ fix  $i$ 
  have  $N$  (disjointed  $Q' i$ )  $\leq N$  ( $Q' i$ )
    by (auto intro!: emeasure_mono simp: disjointed_def)
  then show  $N$  (disjointed  $Q' i$ )  $\neq \infty$ 
    using  $Q'(2)$ [of  $i$ ] by (auto simp: top_unique) }
qed
qed

```

proposition (in finite_measure) Radon_Nikodym_finite_measure_infinite:

assumes absolutely_continuous $M N$ and sets_eq: sets $N =$ sets M

shows $\exists f \in$ borel_measurable M . density $M f = N$

proof –

from split_space_into_finite_sets_and_rest[OF assms]

obtain $Q ::$ nat \Rightarrow 'a set

where Q : disjoint_family Q range $Q \subseteq$ sets M

and in_Q0: $\bigwedge A. A \in$ sets $M \implies A \cap (\bigcup i. Q i) = \{\} \implies$ emeasure $M A = 0 \wedge N A = 0 \vee 0 <$ emeasure $M A \wedge N A = \infty$

and Q_fin: $\bigwedge i. N$ ($Q i$) $\neq \infty$ by force

from Q have Q_sets: $\bigwedge i. Q i \in$ sets M by auto

let $?N = \lambda i. density N$ (indicator ($Q i$)) and $?M = \lambda i. density M$ (indicator ($Q i$))

have $\forall i. \exists f \in$ borel_measurable ($?M i$). density ($?M i$) $f = ?N i$

proof (intro allI finite_measure.Radon_Nikodym_finite_measure)

fix i

```

from  $Q$  show finite_measure ( $?M\ i$ )
  by (auto intro!: finite_measureI cong: nn_integral_cong
      simp add: emeasure_density subset_eq sets_eq)
from  $Q$  have emeasure ( $?N\ i$ ) (space  $N$ ) = emeasure  $N$  ( $Q\ i$ )
by (simp add: sets_eq[symmetric] emeasure_density subset_eq cong: nn_integral_cong)
with  $Q\_fin$  show finite_measure ( $?N\ i$ )
  by (auto intro!: finite_measureI)
show sets ( $?N\ i$ ) = sets ( $?M\ i$ ) by (simp add: sets_eq)
have [measurable]:  $\bigwedge A. A \in \text{sets } M \implies A \in \text{sets } N$  by (simp add: sets_eq)
show absolutely_continuous ( $?M\ i$ ) ( $?N\ i$ )
  using  $\langle \text{absolutely\_continuous } M\ N \rangle \langle Q\ i \in \text{sets } M \rangle$ 
  by (auto simp: absolutely_continuous_def null_sets_density_iff sets_eq
      intro!: absolutely_continuous_AE[OF sets_eq])

qed
from choice[OF this[unfolded Bex_def]]
obtain  $f$  where borel:  $\bigwedge i. f\ i \in \text{borel\_measurable } M \bigwedge i\ x. 0 \leq f\ i\ x$ 
  and f_density:  $\bigwedge i. \text{density } (?M\ i)\ (f\ i) = ?N\ i$ 
  by force
{ fix  $A\ i$  assume  $A \in \text{sets } M$ 
  with  $Q$  borel have  $(\int^{+x}. f\ i\ x * \text{indicator } (Q\ i \cap A)\ x\ \partial M) = \text{emeasure}$ 
(density ( $?M\ i$ ) ( $f\ i$ ))  $A$ 
  by (auto simp add: emeasure_density nn_integral_density subset_eq
      intro!: nn_integral_cong split_indicator)
  also have  $\dots = \text{emeasure } N\ (Q\ i \cap A)$ 
  using  $A\ Q$  by (simp add: f_density emeasure_restricted subset_eq sets_eq)
  finally have emeasure  $N\ (Q\ i \cap A) = (\int^{+x}. f\ i\ x * \text{indicator } (Q\ i \cap A)\ x$ 
 $\partial M) .. \}$ 
  note integral_eq = this
  let  $?f = \lambda x. (\sum i. f\ i\ x * \text{indicator } (Q\ i)\ x) + \infty * \text{indicator } (\text{space } M - (\bigcup i. Q\ i))\ x$ 
  show ?thesis
  proof (safe intro!: bexI[of _ ?f])
    show  $?f \in \text{borel\_measurable } M$  using borel  $Q\_sets$ 
    by (auto intro!: measurable_I)
    show density  $M\ ?f = N$ 
    proof (rule measure_eqI)
      fix  $A$  assume  $A \in \text{sets } (density\ M\ ?f)$ 
      then have  $A \in \text{sets } M$  by simp
      have  $Q_i: \bigwedge i. Q\ i \in \text{sets } M$  using  $Q$  by auto
      have [intro,simp]:  $\bigwedge i. (\lambda x. f\ i\ x * \text{indicator } (Q\ i \cap A)\ x) \in \text{borel\_measurable}$ 
 $M$ 
       $\bigwedge i. AE\ x\ \text{in } M. 0 \leq f\ i\ x * \text{indicator } (Q\ i \cap A)\ x$ 
      using borel  $Q_i \langle A \in \text{sets } M \rangle$  by auto
      have  $(\int^{+x}. ?f\ x * \text{indicator } A\ x\ \partial M) = (\int^{+x}. (\sum i. f\ i\ x * \text{indicator } (Q\ i$ 
 $\cap A)\ x) + \infty * \text{indicator } ((\text{space } M - (\bigcup i. Q\ i)) \cap A)\ x\ \partial M)$ 
      using borel by (intro nn_integral_cong) (auto simp: indicator_def)
      also have  $\dots = (\int^{+x}. (\sum i. f\ i\ x * \text{indicator } (Q\ i \cap A)\ x)\ \partial M) + \infty * \text{emeasure } M$ 
 $((\text{space } M - (\bigcup i. Q\ i)) \cap A)$ 
      using borel  $Q_i \langle A \in \text{sets } M \rangle$ 

```

```

    by (subst nn_integral_add)
      (auto simp add: nn_integral_cmult_indicator sets.Int intro!: suminf_0_le)
  also have ... = ( $\sum i. N (Q i \cap A)$ ) +  $\infty * \text{emeasure } M ((\text{space } M - (\bigcup i. Q i)) \cap A)$ 
  by (subst integral_eq[OF  $\langle A \in \text{sets } M \rangle$ ], subst nn_integral_suminf) auto
  finally have ( $\int^+ x. ?f x * \text{indicator } A x \partial M$ ) = ( $\sum i. N (Q i \cap A)$ ) +  $\infty * \text{emeasure } M ((\text{space } M - (\bigcup i. Q i)) \cap A)$  .
  moreover have ( $\sum i. N (Q i \cap A)$ ) =  $N ((\bigcup i. Q i) \cap A)$ 
    using Q_Q_sets  $\langle A \in \text{sets } M \rangle$ 
    by (subst suminf_emeasure) (auto simp: disjoint_family_on_def sets_eq)
  moreover
  have ( $\text{space } M - (\bigcup x. Q x) \cap A \cap (\bigcup x. Q x) = \{\}$ )
    by auto
  then have  $\infty * \text{emeasure } M ((\text{space } M - (\bigcup i. Q i)) \cap A) = N ((\text{space } M - (\bigcup i. Q i)) \cap A)$ 
    using in_Q0[of ( $\text{space } M - (\bigcup i. Q i) \cap A$ )  $\langle A \in \text{sets } M \rangle$  Q] by (auto simp: ennreal_top_mult)
  moreover have ( $\text{space } M - (\bigcup i. Q i) \cap A \in \text{sets } M ((\bigcup i. Q i) \cap A) \in \text{sets } M$ )
    using Q_sets  $\langle A \in \text{sets } M \rangle$  by auto
  moreover have  $((\bigcup i. Q i) \cap A) \cup ((\text{space } M - (\bigcup i. Q i)) \cap A) = A ((\bigcup i. Q i) \cap A) \cap ((\text{space } M - (\bigcup i. Q i)) \cap A) = \{\}$ 
    using  $\langle A \in \text{sets } M \rangle$  sets_sets_into_space by auto
  ultimately have  $N A = (\int^+ x. ?f x * \text{indicator } A x \partial M)$ 
    using plus_emeasure[of  $(\bigcup i. Q i) \cap A$   $N (\text{space } M - (\bigcup i. Q i)) \cap A$ ] by (simp add: sets_eq)
  with  $\langle A \in \text{sets } M \rangle$  borel Q show  $\text{emeasure } (\text{density } M ?f) A = N A$ 
    by (auto simp: subset_eq_emeasure_density)
  qed (simp add: sets_eq)
qed
qed

```

theorem (in *sigma_finite_measure*) *Radon_Nikodym*:

assumes *ac*: *absolutely_continuous* *M N* **assumes** *sets_eq*: *sets N = sets M*

shows $\exists f \in \text{borel_measurable } M. \text{density } M f = N$

proof –

from *Ex_finite_integrable_function*

obtain *h* **where** *finite*: $\text{integral}^N M h \neq \infty$ **and**

borel: $h \in \text{borel_measurable } M$ **and**

nn: $\bigwedge x. 0 \leq h x$ **and**

pos: $\bigwedge x. x \in \text{space } M \implies 0 < h x$ **and**

$\bigwedge x. x \in \text{space } M \implies h x < \infty$ **by** *auto*

let $?T = \lambda A. (\int^+ x. h x * \text{indicator } A x \partial M)$

let $?MT = \text{density } M h$

from *borel_finite_nn* **interpret** *T*: *finite_measure* $?MT$

by (*auto* *intro!*: *finite_measureI* *cong*: *nn_integral_cong* *simp*: *emeasure_density*)

have *absolutely_continuous* $?MT N$ *sets N = sets ?MT*

proof (*unfold absolutely_continuous_def, safe*)

fix *A* **assume** $A \in \text{null_sets } ?MT$

```

with borel have A ∈ sets M AE x in M. x ∈ A → h x ≤ 0
  by (auto simp add: null_sets_density_iff)
with pos sets.sets_into_space have AE x in M. x ∉ A
  by (elim eventually_mono) (auto simp: not_le[symmetric])
then have A ∈ null_sets M
  using ⟨A ∈ sets M⟩ by (simp add: AE_iff_null_sets)
with ac show A ∈ null_sets N
  by (auto simp: absolutely_continuous_def)
qed (auto simp add: sets_eq)
from T.Radon_Nikodym_finite_measure_infinite[OF this]
obtain f where f_borel: f ∈ borel_measurable M density ?MT f = N by auto
with nn borel show ?thesis
  by (auto intro!: borel[of _ λx. h x * f x] simp: density_density_eq)
qed

```

8.5.3 Uniqueness of densities

lemma finite_density_unique:

```

assumes borel: f ∈ borel_measurable M g ∈ borel_measurable M
assumes pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ g x
and fin: integralN M f ≠ ∞
shows density M f = density M g ↔ (AE x in M. f x = g x)
proof (intro iffI ballI)
  fix A assume eq: AE x in M. f x = g x
  with borel show density M f = density M g
    by (auto intro: density_cong)
next
let ?P = λf A. ∫+ x. f x * indicator A x ∂M
assume density M f = density M g
with borel have eq: ∀ A ∈ sets M. ?P f A = ?P g A
  by (simp add: emeasure_density[symmetric])
from this[THEN bspec, OF sets.top] fin
have g_fin: integralN M g ≠ ∞ by (simp cong: nn_integral_cong)
{ fix f g assume borel: f ∈ borel_measurable M g ∈ borel_measurable M
  and pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ g x
  and g_fin: integralN M g ≠ ∞ and eq: ∀ A ∈ sets M. ?P f A = ?P g A
  let ?N = {x ∈ space M. g x < f x}
  have N: ?N ∈ sets M using borel by simp
  have ?P g ?N ≤ integralN M g using pos
    by (intro nn_integral_mono_AE) (auto split: split_indicator)
  then have Pg_fin: ?P g ?N ≠ ∞ using g_fin by (auto simp: top_unique)
  have ?P (λx. (f x - g x)) ?N = (∫+ x. f x * indicator ?N x - g x * indicator
    ?N x ∂M)
    by (auto intro!: nn_integral_cong simp: indicator_def)
  also have ... = ?P f ?N - ?P g ?N
  proof (rule nn_integral_diff)
    show (λx. f x * indicator ?N x) ∈ borel_measurable M (λx. g x * indicator
    ?N x) ∈ borel_measurable M
    using borel N by auto

```



```

    show AE x in M. g x * indicator ?N x ≤ f x * indicator ?N x
      using pos by (auto split: split_indicator)
  qed fact
  also have ... = 0
    unfolding eq[THEN bspec, OF N] using Pg_fin by auto
  finally have AE x in M. f x ≤ g x
    using pos borel nn_integral_PInf_AE[OF borel(2) g_fin]
    by (subst (asm) nn_integral_0_iff_AE)
      (auto split: split_indicator simp: not_less ennreal_minus_eq_0) }
  from this[OF borel pos g_fin eq] this[OF borel(2,1) pos(2,1) fin] eq
  show AE x in M. f x = g x by auto
qed

lemma (in finite_measure) density_unique_finite_measure:
  assumes borel: f ∈ borel_measurable M f' ∈ borel_measurable M
  assumes pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ f' x
  assumes f:  $\bigwedge A. A \in \text{sets } M \implies (\int^+ x. f x * \text{indicator } A x \partial M) = (\int^+ x. f' x * \text{indicator } A x \partial M)$ 
    (is  $\bigwedge A. A \in \text{sets } M \implies ?P f A = ?P f' A$ )
  shows AE x in M. f x = f' x
proof -
  let ?D =  $\lambda f. \text{density } M f$ 
  let ?N =  $\lambda A. ?P f A$  and ?N' =  $\lambda A. ?P f' A$ 
  let ?f =  $\lambda A x. f x * \text{indicator } A x$  and ?f' =  $\lambda A x. f' x * \text{indicator } A x$ 

  have ac: absolutely_continuous M (density M f) sets (density M f) = sets M
    using borel by (auto intro!: absolutely_continuousI_density)
  from split_space_into_finite_sets_and_rest[OF this]
  obtain Q :: nat  $\Rightarrow$  'a set
    where Q: disjoint_family Q range Q  $\subseteq$  sets M
      and in_Q0:  $\bigwedge A. A \in \text{sets } M \implies A \cap (\bigcup i. Q i) = \{\} \implies \text{emeasure } M A = 0 \wedge ?D f A = 0 \vee 0 < \text{emeasure } M A \wedge ?D f A = \infty$ 
      and Q_fin:  $\bigwedge i. ?D f (Q i) \neq \infty$  by force
    with borel pos have in_Q0:  $\bigwedge A. A \in \text{sets } M \implies A \cap (\bigcup i. Q i) = \{\} \implies \text{emeasure } M A = 0 \wedge ?N A = 0 \vee 0 < \text{emeasure } M A \wedge ?N A = \infty$ 
      and Q_fin:  $\bigwedge i. ?N (Q i) \neq \infty$  by (auto simp: emeasure_density_subset_eq)

  from Q have Q_sets[measurable]:  $\bigwedge i. Q i \in \text{sets } M$  by auto
  let ?D = {x ∈ space M. f x ≠ f' x}
  have ?D ∈ sets M using borel by auto
  have *:  $\bigwedge i x A. \bigwedge y :: \text{ennreal}. y * \text{indicator } (Q i) x * \text{indicator } A x = y * \text{indicator } (Q i \cap A) x$ 
    unfolding indicator_def by auto
  have  $\forall i. AE x \text{ in } M. ?f (Q i) x = ?f' (Q i) x$  using borel Q_fin Q pos
    by (intro finite_density_unique[THEN iffD1] allI)
      (auto intro!: f_measure_eqI simp: emeasure_density * subset_eq)
  moreover have AE x in M. ?f (space M -  $\bigcup i. Q i$ ) x = ?f' (space M -  $\bigcup i. Q i$ ) x
  proof (rule AE_I')

```

```

{ fix f :: 'a ⇒ ennreal assume borel: f ∈ borel_measurable M
  and eq:  $\bigwedge A. A \in \text{sets } M \implies ?N A = (\int^+ x. f x * \text{indicator } A x \partial M)$ 
  let ?A =  $\lambda i. (\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x < (i::\text{nat})\}$ 
  have  $(\bigcup i. ?A i) \in \text{null\_sets } M$ 
  proof (rule null_sets_UN)
    fix i :: nat have ?A i ∈ sets M
      using borel by auto
    have  $?N (?A i) \leq (\int^+ x. (i::\text{ennreal}) * \text{indicator } (?A i) x \partial M)$ 
      unfolding eq[OF  $\langle ?A i \in \text{sets } M \rangle$ ]
      by (auto intro!: nn_integral_mono simp: indicator_def)
    also have ... = i * emeasure M (?A i)
      using  $\langle ?A i \in \text{sets } M \rangle$  by (auto intro!: nn_integral_cmult_indicator)
    also have ... <  $\infty$  using emeasure_real[of ?A i] by (auto simp: ennreal_mult_less_top of_nat_less_top)
    finally have ?N (?A i)  $\neq \infty$  by simp
    then show ?A i ∈ null_sets M using in_Q0[OF  $\langle ?A i \in \text{sets } M \rangle$ ]  $\langle ?A i \in \text{sets } M \rangle$  by auto
  qed
  also have  $(\bigcup i. ?A i) = (\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\}$ 
    by (auto simp: ennreal_Ex_less_of_nat_less_top[symmetric])
  finally have  $(\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\} \in \text{null\_sets } M$  by simp }
  from this[OF borel(1) refl] this[OF borel(2) f]
  have  $(\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\} \in \text{null\_sets } M$  (space M -  $(\bigcup i. Q i) \cap \{x \in \text{space } M. f' x \neq \infty\} \in \text{null\_sets } M$  by simp_all
  then show  $((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\}) \cup ((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f' x \neq \infty\}) \in \text{null\_sets } M$  by (rule null_sets.Un)
  show  $\{x \in \text{space } M. ?f (\text{space } M - (\bigcup i. Q i)) x \neq ?f' (\text{space } M - (\bigcup i. Q i)) x\} \subseteq ((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f x \neq \infty\}) \cup ((\text{space } M - (\bigcup i. Q i)) \cap \{x \in \text{space } M. f' x \neq \infty\})$  by (auto simp: indicator_def)
  qed
  moreover have AE x in M.  $(?f (\text{space } M - (\bigcup i. Q i)) x = ?f' (\text{space } M - (\bigcup i. Q i)) x) \longrightarrow (\forall i. ?f (Q i) x = ?f' (Q i) x) \longrightarrow ?f (\text{space } M) x = ?f' (\text{space } M) x$ 
    by (auto simp: indicator_def)
  ultimately have AE x in M.  $?f (\text{space } M) x = ?f' (\text{space } M) x$ 
    unfolding AE_all_countable[symmetric]
    by eventually_elim (auto split: if_split_asm simp: indicator_def)
  then show AE x in M.  $f x = f' x$  by auto
qed

```

proposition (in sigma_finite_measure) density_unique:

assumes $f: f \in \text{borel_measurable } M$

assumes $f': f' \in \text{borel_measurable } M$

assumes density_eq: $\text{density } M f = \text{density } M f'$

shows AE x in M. $f x = f' x$

proof –

obtain h where h_borel: $h \in \text{borel_measurable } M$

```

and fin: integralN M h ≠ ∞ and pos:  $\bigwedge x. x \in \text{space } M \implies 0 < h\ x \wedge h\ x < \infty \wedge x. 0 \leq h\ x$ 
using Ex_finite_integrable_function by auto
then have h_nn: AE x in M.  $0 \leq h\ x$  by auto
let ?H = density M h
interpret h: finite_measure ?H
using fin_h_borel_pos
by (intro finite_measureI) (simp cong: nn_integral_cong emeasure_density
add: fin)
let ?fM = density M f
let ?f'M = density M f'
{ fix A assume A ∈ sets M
then have  $\{x \in \text{space } M. h\ x * \text{indicator } A\ x \neq 0\} = A$ 
using pos(1) sets.sets_into_space by (force simp: indicator_def)
then have  $(\int^{+x}. h\ x * \text{indicator } A\ x\ \partial M) = 0 \iff A \in \text{null\_sets } M$ 
using h_borel  $\langle A \in \text{sets } M \rangle$  h_nn by (subst nn_integral_0_iff) auto }
note h_null_sets = this
{ fix A assume A ∈ sets M
have  $(\int^{+x}. f\ x * (h\ x * \text{indicator } A\ x)\ \partial M) = (\int^{+x}. h\ x * \text{indicator } A\ x\ \partial ?fM)$ 
using  $\langle A \in \text{sets } M \rangle$  h_borel h_nn f f'
by (intro nn_integral_density[symmetric]) auto
also have  $\dots = (\int^{+x}. h\ x * \text{indicator } A\ x\ \partial ?f'M)$ 
by (simp_all add: density_eq)
also have  $\dots = (\int^{+x}. f'\ x * (h\ x * \text{indicator } A\ x)\ \partial M)$ 
using  $\langle A \in \text{sets } M \rangle$  h_borel h_nn f f'
by (intro nn_integral_density) auto
finally have  $(\int^{+x}. h\ x * (f\ x * \text{indicator } A\ x)\ \partial M) = (\int^{+x}. h\ x * (f'\ x * \text{indicator } A\ x)\ \partial M)$ 
by (simp add: ac_simps)
then have  $(\int^{+x}. (f\ x * \text{indicator } A\ x)\ \partial ?H) = (\int^{+x}. (f'\ x * \text{indicator } A\ x)\ \partial ?H)$ 
using  $\langle A \in \text{sets } M \rangle$  h_borel h_nn f f'
by (subst (asm) (1 2) nn_integral_density[symmetric]) auto }
then have AE x in ?H. f x = f' x using h_borel h_nn f f'
by (intro h.density_unique_finite_measure absolutely_continuous_AE[of M])
auto
with AE_space[of M] pos show AE x in M. f x = f' x
unfolding AE_density[OF h_borel] by auto
qed

```

lemma (*in sigma_finite_measure*) *density_unique_iff*:
assumes *f*: *f* ∈ *borel_measurable* *M* **and** *f'*: *f'* ∈ *borel_measurable* *M*
shows *density* *M* *f* = *density* *M* *f'* \iff (*AE* *x* *in* *M*. *f* *x* = *f'* *x*)
using *density_unique*[*OF* *assms*] *density_cong*[*OF* *f* *f'*] **by** *auto*

lemma *sigma_finite_density_unique*:
assumes *borel*: *f* ∈ *borel_measurable* *M* *g* ∈ *borel_measurable* *M*
and *fin*: *sigma_finite_measure* (*density* *M* *f*)

2800

```
shows density M f = density M g  $\longleftrightarrow$  (AE x in M. f x = g x)
proof
  assume AE x in M. f x = g x with borel show density M f = density M g
  by (auto intro: density_cong)
next
  assume eq: density M f = density M g
  interpret f: sigma_finite_measure density M f by fact
  from f.sigma_finite_incseq obtain A where cover: range A  $\subseteq$  sets (density M
f)
   $\bigcup$  (range A) = space (density M f)
   $\bigwedge i$ . emeasure (density M f) (A i)  $\neq$   $\infty$ 
  incseq A
  by auto
  have AE x in M.  $\forall i$ . x  $\in$  A i  $\longrightarrow$  f x = g x
  unfolding AE_all_countable
  proof
    fix i
    have density (density M f) (indicator (A i)) = density (density M g) (indicator
(A i))
    unfolding eq ..
    moreover have ( $\int$   $^+$ x. f x * indicator (A i) x  $\partial$ M)  $\neq$   $\infty$ 
    using cover(1) cover(3)[of i] borel by (auto simp: emeasure_density sub-
set_eq)
    ultimately have AE x in M. f x * indicator (A i) x = g x * indicator (A i) x
    using borel cover(1)
    by (intro finite_density_unique[THEN iffD1]) (auto simp: density_density_eq
subset_eq)
    then show AE x in M. x  $\in$  A i  $\longrightarrow$  f x = g x
    by auto
  qed
  with AE_space show AE x in M. f x = g x
  apply eventually_elim
  using cover(2)[symmetric]
  apply auto
  done
qed
```

```
lemma (in sigma_finite_measure) sigma_finite_iff_density_finite':
  assumes f: f  $\in$  borel_measurable M
  shows sigma_finite_measure (density M f)  $\longleftrightarrow$  (AE x in M. f x  $\neq$   $\infty$ )
  (is sigma_finite_measure ?N  $\longleftrightarrow$  _)
proof
  assume sigma_finite_measure ?N
  then interpret N: sigma_finite_measure ?N .
  from N.Ex_finite_integrable_function obtain h where
  h: h  $\in$  borel_measurable M integralN ?N h  $\neq$   $\infty$  and
  fin:  $\forall x \in$  space M. 0 < h x  $\wedge$  h x <  $\infty$ 
  by auto
  have AE x in M. f x * h x  $\neq$   $\infty$ 
```

```

proof (rule AE_I')
  have  $\text{integral}^N \ ?N \ h = (\int^{+x}. f \ x * h \ x \ \partial M)$ 
    using  $f \ h$  by (auto intro!: nn_integral_density)
  then have  $(\int^{+x}. f \ x * h \ x \ \partial M) \neq \infty$ 
    using  $h(2)$  by simp
  then show  $(\lambda x. f \ x * h \ x) - \{ \infty \} \cap \text{space } M \in \text{null\_sets } M$ 
    using  $f \ h(1)$  by (auto intro!: nn_integral_PInf[unfolded_infinity_ennreal_def]
borel_measurable_vimage)
  qed auto
  then show  $AE \ x \ \text{in } M. f \ x \neq \infty$ 
    using  $fin$  by (auto elim!: AE_Ball_mp simp: less_top ennreal_mult_less_top)
next
  assume  $AE: AE \ x \ \text{in } M. f \ x \neq \infty$ 
  from  $\text{sigma\_finite}$  obtain  $Q :: \text{nat} \Rightarrow 'a \ \text{set}$ 
    where  $Q: \text{range } Q \subseteq \text{sets } M \cup (\text{range } Q) = \text{space } M \wedge i. \text{emeasure } M \ (Q \ i)$ 
 $\neq \infty$ 
    by auto
  define  $A$  where  $A \ i =$ 
 $f - \{ \text{case } i \ \text{of } 0 \Rightarrow \{ \infty \} \mid \text{Suc } n \Rightarrow \{ .. \text{ennreal}(\text{of\_nat } (\text{Suc } n)) \} \} \cap \text{space } M$ 
for  $i$ 
  { fix  $i \ j$  have  $A \ i \cap Q \ j \in \text{sets } M$ 
    unfolding  $A\_def$  using  $f \ Q$ 
    apply (rule_tac sets.Int)
    by (cases  $i$ ) (auto intro: measurable_sets[OF  $f(1)$ ]) }
  note  $A\_in\_sets = \text{this}$ 

show  $\text{sigma\_finite\_measure } ?N$ 
proof (standard, intro exI conjI ballI)
  show  $\text{countable } (\text{range } (\lambda(i, j). A \ i \cap Q \ j))$ 
    by auto
  show  $\text{range } (\lambda(i, j). A \ i \cap Q \ j) \subseteq \text{sets } (\text{density } M \ f)$ 
    using  $A\_in\_sets$  by auto
next
  have  $\bigcup (\text{range } (\lambda(i, j). A \ i \cap Q \ j)) = (\bigcup i \ j. A \ i \cap Q \ j)$ 
    by auto
  also have  $\dots = (\bigcup i. A \ i) \cap \text{space } M$  using  $Q$  by auto
  also have  $(\bigcup i. A \ i) = \text{space } M$ 
proof safe
    fix  $x$  assume  $x: x \in \text{space } M$ 
    show  $x \in (\bigcup i. A \ i)$ 
proof (cases  $f \ x$  rule: ennreal_cases)
      case  $\text{top}$  with  $x$  show  $?thesis$  unfolding  $A\_def$  by (auto intro: exI[of _
0])
    next
      case (real  $r$ )
      with  $\text{ennreal\_Ex\_less\_of\_nat}[of \ f \ x]$  obtain  $n :: \text{nat}$  where  $f \ x < n$ 
        by auto
      also have  $n < (\text{Suc } n :: \text{ennreal})$ 
        by simp

```

```

      finally show ?thesis
        using x real by (auto simp: A_def ennreal_of_nat_eq_real_of_nat intro!:
exI[of _ Suc n])
      qed
      qed (auto simp: A_def)
      finally show  $\bigcup (\text{range } (\lambda(i, j). A\ i \cap Q\ j)) = \text{space } ?N$  by simp
    next
      fix X assume  $X \in \text{range } (\lambda(i, j). A\ i \cap Q\ j)$ 
      then obtain i j where [simp]:  $X = A\ i \cap Q\ j$  by auto
      have  $(\int^+ x. f\ x * \text{indicator } (A\ i \cap Q\ j)\ x\ \partial M) \neq \infty$ 
      proof (cases i)
        case 0
          have AE x in M.  $f\ x * \text{indicator } (A\ i \cap Q\ j)\ x = 0$ 
            using AE by (auto simp: A_def ‹i = 0›)
          from nn_integral_cong_AE[OF this] show ?thesis by simp
        next
          case (Suc n)
            then have  $(\int^+ x. f\ x * \text{indicator } (A\ i \cap Q\ j)\ x\ \partial M) \leq$ 
               $(\int^+ x. (\text{Suc } n :: \text{ennreal}) * \text{indicator } (Q\ j)\ x\ \partial M)$ 
            by (auto intro!: nn_integral_mono simp: indicator_def A_def ennreal_of_nat_eq_real_of_nat)
            also have  $\dots = \text{Suc } n * \text{emeasure } M\ (Q\ j)$ 
              using Q by (auto intro!: nn_integral_cmult_indicator)
            also have  $\dots < \infty$ 
              using Q by (auto simp: ennreal_mult_less_top less_top_of_nat_less_top)
            finally show ?thesis by simp
          qed
        then show emeasure ?N X  $\neq \infty$ 
          using A_in_sets Q f by (auto simp: emeasure_density)
        qed
      qed

```

```

lemma (in sigma_finite_measure) sigma_finite_iff_density_finite:
   $f \in \text{borel\_measurable } M \implies \text{sigma\_finite\_measure } (\text{density } M\ f) \iff (AE\ x\ \text{in } M. f\ x \neq \infty)$ 
  by (subst sigma_finite_iff_density_finite')
  (auto simp: max_def intro!: measurable>If)

```

8.5.4 Radon-Nikodym derivative

```

definition RN_deriv :: 'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  'a  $\Rightarrow$  ennreal where
  RN_deriv M N =
    (if  $\exists f. f \in \text{borel\_measurable } M \wedge \text{density } M\ f = N$ 
     then SOME  $f. f \in \text{borel\_measurable } M \wedge \text{density } M\ f = N$ 
     else  $(\lambda_. 0)$ )

```

```

lemma RN_derivI:
  assumes  $f \in \text{borel\_measurable } M$   $\text{density } M\ f = N$ 
  shows  $\text{density } M\ (\text{RN\_deriv } M\ N) = N$ 
proof -

```

```

have *:  $\exists f. f \in \text{borel\_measurable } M \wedge \text{density } M f = N$ 
  using assms by auto
then have  $\text{density } M (\text{SOME } f. f \in \text{borel\_measurable } M \wedge \text{density } M f = N)$ 
=  $N$ 
  by (rule someI2_ex) auto
with * show ?thesis
  by (auto simp: RN_deriv_def)
qed

```

```

lemma borel_measurable_RN_deriv[measurable]:  $\text{RN\_deriv } M N \in \text{borel\_measurable } M$ 
proof –
  { assume ex:  $\exists f. f \in \text{borel\_measurable } M \wedge \text{density } M f = N$ 
    have 1:  $(\text{SOME } f. f \in \text{borel\_measurable } M \wedge \text{density } M f = N) \in \text{borel\_measurable } M$ 
      using ex by (rule someI2_ex) auto }
  from this show ?thesis
    by (auto simp: RN_deriv_def)
qed

```

```

lemma density_RN_deriv_density:
assumes  $f: f \in \text{borel\_measurable } M$ 
shows  $\text{density } M (\text{RN\_deriv } M (\text{density } M f)) = \text{density } M f$ 
by (rule RN_derivI[OF f]) simp

```

```

lemma (in sigma_finite_measure) density_RN_deriv:
 $\text{absolutely\_continuous } M N \implies \text{sets } N = \text{sets } M \implies \text{density } M (\text{RN\_deriv } M N) = N$ 
by (metis RN_derivI Radon_Nikodym)

```

```

lemma (in sigma_finite_measure) RN_deriv_nn_integral:
assumes  $N: \text{absolutely\_continuous } M N$   $\text{sets } N = \text{sets } M$ 
and  $f: f \in \text{borel\_measurable } M$ 
shows  $\text{integral}^N N f = (\int^+ x. \text{RN\_deriv } M N x * f x \partial M)$ 
proof –
  have  $\text{integral}^N N f = \text{integral}^N (\text{density } M (\text{RN\_deriv } M N)) f$ 
    using  $N$  by (simp add: density_RN_deriv)
  also have  $\dots = (\int^+ x. \text{RN\_deriv } M N x * f x \partial M)$ 
    using  $f$  by (simp add: nn_integral_density)
  finally show ?thesis by simp
qed

```

```

lemma (in sigma_finite_measure) RN_deriv_unique:
assumes  $f: f \in \text{borel\_measurable } M$ 
and  $eq: \text{density } M f = N$ 
shows  $\text{AE } x \text{ in } M. f x = \text{RN\_deriv } M N x$ 
unfolding eq[symmetric]
by (intro density_unique_iff[THEN iffD1] f borel_measurable_RN_deriv density_RN_deriv_density[symmetric])

```

lemma *RN_deriv_unique_sigma_finite*:
assumes $f: f \in \text{borel_measurable } M$
and $eq: \text{density } M f = N$ **and** $fin: \text{sigma_finite_measure } N$
shows $AE\ x\ \text{in } M. f\ x = \text{RN_deriv } M\ N\ x$
using fin **unfolding** $eq[\text{symmetric}]$
by (*intro sigma_finite_density_unique[THEN iffD1]*) $f \text{ borel_measurable_RN_deriv}$
 $\text{density_RN_deriv_density}[\text{symmetric}]$)

lemma (*in sigma_finite_measure*) *RN_deriv_distr*:
fixes $T :: 'a \Rightarrow 'b$
assumes $T: T \in \text{measurable } M\ M'$ **and** $T': T' \in \text{measurable } M'\ M$
and $inv: \forall x \in \text{space } M. T'\ (T\ x) = x$
and $ac[\text{simp}]: \text{absolutely_continuous } (\text{distr } M\ M'\ T)\ (\text{distr } N\ M'\ T)$
and $N: \text{sets } N = \text{sets } M$
shows $AE\ x\ \text{in } M. \text{RN_deriv } (\text{distr } M\ M'\ T)\ (\text{distr } N\ M'\ T)\ (T\ x) = \text{RN_deriv}$
 $M\ N\ x$
proof (*rule RN_deriv_unique*)
have $[\text{simp}]: \text{sets } N = \text{sets } M$ **by** *fact*
note $\text{sets_eq_imp_space_eq}[OF\ N, \text{simp}]$
have $\text{measurable_N}[\text{simp}]: \bigwedge M'. \text{measurable } N\ M' = \text{measurable } M\ M'$ **by** (*auto*
 $\text{simp: measurable_def}$)
{ **fix** A **assume** $A \in \text{sets } M$
with $inv\ T\ T'$ $\text{sets.sets_into_space}[OF\ \text{this}]$
have $T\ -' T'\ -' A \cap T\ -' \text{space } M' \cap \text{space } M = A$
by (*auto simp: measurable_def*) **}**
note $eq = \text{this}[\text{simp}]$
{ **fix** A **assume** $A \in \text{sets } M$
with $inv\ T\ T'$ $\text{sets.sets_into_space}[OF\ \text{this}]$
have $(T' \circ T)\ -' A \cap \text{space } M = A$
by (*auto simp: measurable_def*) **}**
note $eq2 = \text{this}[\text{simp}]$
let $?M' = \text{distr } M\ M'\ T$ **and** $?N' = \text{distr } N\ M'\ T$
interpret $M': \text{sigma_finite_measure } ?M'$
proof
from *sigma_finite_countable* **obtain** F
where $F: \text{countable } F \wedge F \subseteq \text{sets } M \wedge \bigcup F = \text{space } M \wedge (\forall a \in F. \text{emeasure}$
 $M\ a \neq \infty) ..$
show $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (\text{distr } M\ M'\ T) \wedge \bigcup A = \text{space } (\text{distr } M\ M'$
 $T) \wedge (\forall a \in A. \text{emeasure } (\text{distr } M\ M'\ T)\ a \neq \infty)$
proof (*intro exI conjI ballI*)
show $*$: $(\lambda A. T'\ -' A \cap \text{space } ?M')\ 'F \subseteq \text{sets } ?M'$
using $F\ T'$ **by** (*auto simp: measurable_def*)
show $\bigcup ((\lambda A. T'\ -' A \cap \text{space } ?M')\ 'F) = \text{space } ?M'$
using $F\ T'$ $[THEN\ \text{measurable_space}]$ **by** (*auto simp: set_eq_iff*)
next
fix X **assume** $X \in (\lambda A. T'\ -' A \cap \text{space } ?M')\ 'F$
then **obtain** A **where** $[\text{simp}]: X = T'\ -' A \cap \text{space } ?M'$ **and** $A \in F$ **by**
auto


```

    have  $X \in \text{sets } M'$  using  $F T' \langle A \in F \rangle$  by auto
    moreover
    have  $Fi: A \in \text{sets } M$  using  $F \langle A \in F \rangle$  by auto
    ultimately show  $\text{emeasure } ?M' X \neq \infty$ 
      using  $F T T' \langle A \in F \rangle$  by (simp add: emeasure_distr)
  qed (use  $F$  in auto)
qed
have  $(RN\_deriv ?M' ?N') \circ T \in \text{borel\_measurable } M$ 
  using  $T$  ac by measurable
then show  $(\lambda x. RN\_deriv ?M' ?N' (T x)) \in \text{borel\_measurable } M$ 
  by (simp add: comp_def)

have  $N = \text{distr } N M (T' \circ T)$ 
  by (subst measure_of_of_measure[of  $N$ , symmetric])
  (auto simp add: distr_def sets.sigma_sets_eq intro!: measure_of_eq sets.space_closed)
also have  $\dots = \text{distr } (\text{distr } N M' T) M T'$ 
  using  $T T'$  by (simp add: distr_distr)
also have  $\dots = \text{distr } (\text{density } (\text{distr } M M' T) (RN\_deriv (\text{distr } M M' T) (\text{distr } N M' T))) M T'$ 
  using ac by (simp add:  $M'.\text{density\_RN\_deriv}$ )
also have  $\dots = \text{density } M (RN\_deriv (\text{distr } M M' T) (\text{distr } N M' T) \circ T)$ 
  by (simp add: distr_density_distr[OF  $T T'$ , OF inv])
finally show  $\text{density } M (\lambda x. RN\_deriv (\text{distr } M M' T) (\text{distr } N M' T) (T x)) = N$ 
  by (simp add: comp_def)
qed

lemma (in sigma_finite_measure) RN_deriv_finite:
  assumes  $N: \text{sigma\_finite\_measure } N$  and  $ac: \text{absolutely\_continuous } M N$  sets  $N = \text{sets } M$ 
  shows  $\forall x \in M. RN\_deriv M N x \neq \infty$ 
proof -
  interpret  $N: \text{sigma\_finite\_measure } N$  by fact
  from  $N$  show ?thesis
    using sigma_finite_iff_density_finite[OF borel_measurable_RN_deriv, of  $N$ ]
    density_RN_deriv[OF ac]
    by simp
qed

lemma (in sigma_finite_measure)
  assumes  $N: \text{sigma\_finite\_measure } N$  and  $ac: \text{absolutely\_continuous } M N$  sets  $N = \text{sets } M$ 
  and  $f: f \in \text{borel\_measurable } M$ 
  shows  $RN\_deriv\_integrable: \text{integrable } N f \iff$ 
     $\text{integrable } M (\lambda x. \text{enn2real } (RN\_deriv M N x) * f x)$  (is ?integrable)
  and  $RN\_deriv\_integral: \text{integral}^L N f = (\int x. \text{enn2real } (RN\_deriv M N x) * f x \partial M)$  (is ?integral)
proof -
  note ac(2)[simp] and sets_eq_imp_space_eq[OF ac(2), simp]

```

interpret N : *sigma_finite_measure* N **by fact**

have eq : $density\ M\ (RN_deriv\ M\ N) = density\ M\ (\lambda x. enn2real\ (RN_deriv\ M\ N\ x))$

proof (*rule density_cong*)

from $RN_deriv_finite[OF\ assms(1,2,3)]$

show $AE\ x\ in\ M. RN_deriv\ M\ N\ x = ennreal\ (enn2real\ (RN_deriv\ M\ N\ x))$

by *eventually_elim* (*auto simp: less_top*)

qed (*insert ac, auto*)

show *?integrable*

apply (*subst density_RN_deriv[OF ac, symmetric]*)

unfolding *eq*

apply (*intro integrable_real_density f AE_I2 enn2real_nonneg*)

apply (*insert ac, auto*)

done

show *?integral*

apply (*subst density_RN_deriv[OF ac, symmetric]*)

unfolding *eq*

apply (*intro integral_real_density f AE_I2 enn2real_nonneg*)

apply (*insert ac, auto*)

done

qed

proposition (*in sigma_finite_measure*) *real_RN_deriv*:

assumes *finite_measure* N

assumes *ac*: *absolutely_continuous* $M\ N$ *sets* $N = sets\ M$

obtains D **where** $D \in borel_measurable\ M$

and $AE\ x\ in\ M. RN_deriv\ M\ N\ x = ennreal\ (D\ x)$

and $AE\ x\ in\ N. 0 < D\ x$

and $\bigwedge x. 0 \leq D\ x$

proof

interpret N : *finite_measure* N **by fact**

note $RN = borel_measurable_RN_deriv\ density_RN_deriv[OF\ ac]$

let $?RN = \lambda t. \{x \in space\ M. RN_deriv\ M\ N\ x = t\}$

show $(\lambda x. enn2real\ (RN_deriv\ M\ N\ x)) \in borel_measurable\ M$

using RN **by auto**

have $N\ (?RN\ \infty) = (\int^+ x. RN_deriv\ M\ N\ x * indicator\ (?RN\ \infty)\ x\ \partial M)$

using $RN(1)$ **by** (*subst RN(2)[symmetric]*) (*auto simp: emeasure_density*)

also have $\dots = (\int^+ x. \infty * indicator\ (?RN\ \infty)\ x\ \partial M)$

by (*intro nn_integral_cong*) (*auto simp: indicator_def*)

also have $\dots = \infty * emeasure\ M\ (?RN\ \infty)$

using RN **by** (*intro nn_integral_cmult_indicator*) *auto*

finally have $eq: N\ (?RN\ \infty) = \infty * emeasure\ M\ (?RN\ \infty)$.

```

moreover
have emeasure  $M$  (?RN  $\infty$ ) = 0
proof (rule ccontr)
  assume emeasure  $M$  { $x \in \text{space } M. \text{RN\_deriv } M \ N \ x = \infty$ }  $\neq 0$ 
  then have  $0 < \text{emeasure } M$  { $x \in \text{space } M. \text{RN\_deriv } M \ N \ x = \infty$ }
    by (auto simp: zero_less_iff_neq_zero)
  with eq have  $N$  (?RN  $\infty$ ) =  $\infty$  by (simp add: ennreal_mult_eq_top_iff)
  with  $N.emeasure\_finite$ [of ?RN  $\infty$ ]  $RN$  show False by auto
qed
ultimately have  $AE \ x \ \text{in } M. \text{RN\_deriv } M \ N \ x < \infty$ 
  using  $RN$  by (intro AE_iff_measurable[THEN iffD2]) (auto simp: less_top[symmetric])
then show  $AE \ x \ \text{in } M. \text{RN\_deriv } M \ N \ x = \text{ennreal } (\text{enn2real } (\text{RN\_deriv } M \ N \ x))$ 
  by auto
then have eq: AE x in N. RN_deriv M N x = ennreal (enn2real (RN_deriv M N x))
  using ac absolutely_continuous_AE by auto

have  $N$  (?RN 0) =  $(\int^+ x. \text{RN\_deriv } M \ N \ x * \text{indicator } (?RN \ 0) \ x \ \partial M)$ 
  by (subst RN(2)[symmetric]) (auto simp: emeasure_density)
also have  $\dots = (\int^+ x. 0 \ \partial M)$ 
  by (intro nn_integral_cong) (auto simp: indicator_def)
finally have  $AE \ x \ \text{in } N. \text{RN\_deriv } M \ N \ x \neq 0$ 
  using  $RN$  by (subst AE_iff_measurable[OF _ refl]) (auto simp: ac cong: sets_eq_imp_space_eq)
  with eq show  $AE \ x \ \text{in } N. 0 < \text{enn2real } (\text{RN\_deriv } M \ N \ x)$ 
  by (auto simp: enn2real_positive_iff less_top[symmetric] zero_less_iff_neq_zero)
qed (rule enn2real_nonneg)

lemma (in sigma_finite_measure) RN_deriv_singleton:
  assumes ac: absolutely_continuous M N sets N = sets M
  and  $x: \{x\} \in \text{sets } M$ 
  shows  $N \ \{x\} = \text{RN\_deriv } M \ N \ x * \text{emeasure } M \ \{x\}$ 
proof -
  from  $\langle \{x\} \in \text{sets } M \rangle$ 
  have density  $M$  ( $\text{RN\_deriv } M \ N$ )  $\{x\} = (\int^+ w. \text{RN\_deriv } M \ N \ x * \text{indicator } \{x\} \ w \ \partial M)$ 
  by (auto simp: indicator_def emeasure_density intro!: nn_integral_cong)
  with  $x$  density_RN_deriv[OF ac] show ?thesis
  by (auto simp: max_def)
qed

end

```


Chapter 9

Integrals over a Set

```
theory Set_Integral
  imports Radon_Nikodym
begin
```

9.1 Notation

```
definition set_borel_measurable M A f ≡ (λx. indicator A x *R f x) ∈ borel_measurable M
```

```
definition set_integrable M A f ≡ integrable M (λx. indicator A x *R f x)
```

```
definition set_lebesgue_integral M A f ≡ lebesgue_integral M (λx. indicator A x *R f x)
```

syntax

```
ascii_set_lebesgue_integral :: pttrn ⇒ 'a set ⇒ 'a measure ⇒ real ⇒ real
((4LINT (λx. indicator A x *R f x) [0,0,0,10] 10)
```

translations

```
LINT x:A|M. f == CONST set_lebesgue_integral M A (λx. f)
```

syntax

```
lebesgue_borel_integral :: pttrn ⇒ real ⇒ real
((2LBINT _./ _ [0,10] 10)
```

syntax

```
set_lebesgue_borel_integral :: pttrn ⇒ real set ⇒ real ⇒ real
((3LBINT _:./ _ [0,0,10] 10)
```

9.2 Basic properties

lemma *set_integrable_cong*:

assumes $M = M' \ A = A' \ \wedge x. x \in A \implies f x = f' x$

shows $\text{set_integrable } M \ A \ f = \text{set_integrable } M' \ A' \ f'$

proof –

have $(\lambda x. \text{indicator } A \ x \ *_{\mathbb{R}} \ f \ x) = (\lambda x. \text{indicator } A' \ x \ *_{\mathbb{R}} \ f' \ x)$

using *assms* **by** (*auto simp: indicator_def of_bool_def*)

thus *?thesis* **by** (*simp add: set_integrable_def assms*)

qed

lemma *set_borel_measurable_sets*:

fixes $f :: _ \Rightarrow _ :: \text{real_normed_vector}$

assumes $\text{set_borel_measurable } M \ X \ f \ B \in \text{sets } \text{borel } X \in \text{sets } M$

shows $f \ -' \ B \cap X \in \text{sets } M$

proof –

have $f \in \text{borel_measurable } (\text{restrict_space } M \ X)$

using *assms* **unfolding** *set_borel_measurable_def* **by** (*subst borel_measurable_restrict_space_iff*)

auto

then have $f \ -' \ B \cap \text{space } (\text{restrict_space } M \ X) \in \text{sets } (\text{restrict_space } M \ X)$

by (*rule measurable_sets*) *fact*

with $\langle X \in \text{sets } M \rangle$ **show** *?thesis*

by (*subst (asm) sets_restrict_space_iff*) (*auto simp: space_restrict_space*)

qed

lemma *set_integrable_bound*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second_countable_topology}\}$

and $g :: 'a \Rightarrow 'c :: \{\text{banach}, \text{second_countable_topology}\}$

assumes $\text{set_integrable } M \ A \ f \ \text{set_borel_measurable } M \ A \ g$

assumes $A \ E \ x \ \text{in } M. x \in A \implies \text{norm } (g \ x) \leq \text{norm } (f \ x)$

shows $\text{set_integrable } M \ A \ g$

unfolding *set_integrable_def*

proof (*rule Bochner_Integration.integrable_bound*)

from *assms*(1) **show** $\text{integrable } M \ (\lambda x. \text{indicator } A \ x \ *_{\mathbb{R}} \ f \ x)$

by (*simp add: set_integrable_def*)

from *assms*(2) **show** $(\lambda x. \text{indicat_real } A \ x \ *_{\mathbb{R}} \ g \ x) \in \text{borel_measurable } M$

by (*simp add: set_borel_measurable_def*)

from *assms*(3) **show** $A \ E \ x \ \text{in } M. \text{norm } (\text{indicat_real } A \ x \ *_{\mathbb{R}} \ g \ x) \leq \text{norm } (\text{indicat_real } A \ x \ *_{\mathbb{R}} \ f \ x)$

by *eventually_elim* (*simp add: indicator_def*)

qed

lemma *set_lebesgue_integral_zero* [*simp*]: $\text{set_lebesgue_integral } M \ A \ (\lambda x. 0) = 0$

by (*auto simp: set_lebesgue_integral_def*)

lemma *set_lebesgue_integral_cong*:

assumes $A \in \text{sets } M$ **and** $\forall x. x \in A \implies f \ x = g \ x$

shows $(\text{LINT } x:A | M. f \ x) = (\text{LINT } x:A | M. g \ x)$

```

unfolding set_lebesgue_integral_def
using assms
by (metis indicator_simps(2) real_vector.scale_zero_left)

lemma set_lebesgue_integral_cong_AE:
  assumes [measurable]:  $A \in \text{sets } M$   $f \in \text{borel\_measurable } M$   $g \in \text{borel\_measurable } M$ 
  assumes  $AE$   $x \in A$  in  $M$ .  $f x = g x$ 
  shows  $(LINT x:A|M. f x) = (LINT x:A|M. g x)$ 
proof -
  have  $AE$   $x$  in  $M$ .  $\text{indicator } A x *_R f x = \text{indicator } A x *_R g x$ 
  using assms by auto
  thus ?thesis
  unfolding set_lebesgue_integral_def by (intro integral_cong_AE) auto
qed

lemma set_integrable_cong_AE:
   $f \in \text{borel\_measurable } M \implies g \in \text{borel\_measurable } M \implies$ 
   $AE x \in A$  in  $M$ .  $f x = g x \implies A \in \text{sets } M \implies$ 
   $\text{set\_integrable } M A f = \text{set\_integrable } M A g$ 
  unfolding set_integrable_def
  by (rule integrable_cong_AE) auto

lemma set_integrable_subset:
  fixes  $M A B$  and  $f :: \_ \Rightarrow \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $\text{set\_integrable } M A f$   $B \in \text{sets } M$   $B \subseteq A$ 
  shows  $\text{set\_integrable } M B f$ 
proof -
  have  $\text{set\_integrable } M B (\lambda x. \text{indicator } A x *_R f x)$ 
  using assms  $\text{integrable\_mult\_indicator set\_integrable\_def}$  by blast
  with  $\langle B \subseteq A \rangle$  show ?thesis
  unfolding set_integrable_def
  by (simp add: indicator_inter_arith[symmetric] Int_absorb2)
qed

lemma set_integrable_restrict_space:
  fixes  $f :: 'a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $f: \text{set\_integrable } M S f$  and  $T: T \in \text{sets } (\text{restrict\_space } M S)$ 
  shows  $\text{set\_integrable } M T f$ 
proof -
  obtain  $T'$  where  $T\_eq: T = S \cap T'$  and  $T' \in \text{sets } M$ 
  using  $T$  by (auto simp: sets_restrict_space)
  have  $\langle \text{integrable } M (\lambda x. \text{indicator } T' x *_R (\text{indicator } S x *_R f x)) \rangle$ 
  using  $\langle T' \in \text{sets } M \rangle$   $f$   $\text{integrable\_mult\_indicator set\_integrable\_def}$  by blast
  then show ?thesis
  unfolding set_integrable_def
  unfolding  $T\_eq$   $\text{indicator\_inter\_arith}$  by (simp add: ac_simps)
qed

```

```

lemma set_integral_scaleR_left:
  assumes  $A \in \text{sets } M \ c \neq 0 \implies \text{integrable } M \ f$ 
  shows  $(\text{LINT } t:A|M. \ f \ t \ *_{R} \ c) = (\text{LINT } t:A|M. \ f \ t) \ *_{R} \ c$ 
  unfolding set_lebesgue_integral_def
  using integrable_mult_indicator[OF assms]
  by (subst integral_scaleR_left[symmetric], auto)

lemma set_integral_scaleR_right [simp]:  $(\text{LINT } t:A|M. \ a \ *_{R} \ f \ t) = a \ *_{R} \ (\text{LINT } t:A|M. \ f \ t)$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_scaleR_right[symmetric]) (auto intro!: Bochner_Integration.integral_cong)

lemma set_integral_mult_right [simp]:
  fixes  $a :: 'a :: \{\text{real\_normed\_field, second\_countable\_topology}\}$ 
  shows  $(\text{LINT } t:A|M. \ a \ * \ f \ t) = a \ * \ (\text{LINT } t:A|M. \ f \ t)$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_mult_right_zero[symmetric]) auto

lemma set_integral_mult_left [simp]:
  fixes  $a :: 'a :: \{\text{real\_normed\_field, second\_countable\_topology}\}$ 
  shows  $(\text{LINT } t:A|M. \ f \ t \ * \ a) = (\text{LINT } t:A|M. \ f \ t) \ * \ a$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_mult_left_zero[symmetric]) auto

lemma set_integral_divide_zero [simp]:
  fixes  $a :: 'a :: \{\text{real\_normed\_field, field, second\_countable\_topology}\}$ 
  shows  $(\text{LINT } t:A|M. \ f \ t \ / \ a) = (\text{LINT } t:A|M. \ f \ t) \ / \ a$ 
  unfolding set_lebesgue_integral_def
  by (subst integral_divide_zero[symmetric], intro Bochner_Integration.integral_cong)
  (auto split: split_indicator)

lemma set_integrable_scaleR_right [simp, intro]:
  shows  $(a \neq 0 \implies \text{set\_integrable } M \ A \ f) \implies \text{set\_integrable } M \ A \ (\lambda t. \ a \ *_{R} \ f \ t)$ 
  unfolding set_integrable_def
  unfolding scaleR_left_commute by (rule integrable_scaleR_right)

lemma set_integrable_scaleR_left [simp, intro]:
  fixes  $a :: \_ :: \{\text{banach, second\_countable\_topology}\}$ 
  shows  $(a \neq 0 \implies \text{set\_integrable } M \ A \ f) \implies \text{set\_integrable } M \ A \ (\lambda t. \ f \ t \ *_{R} \ a)$ 
  unfolding set_integrable_def
  using integrable_scaleR_left[of a M \lambda x. indicator A x *_{R} f x] by simp

lemma set_integrable_mult_right [simp, intro]:
  fixes  $a :: 'a :: \{\text{real\_normed\_field, second\_countable\_topology}\}$ 
  shows  $(a \neq 0 \implies \text{set\_integrable } M \ A \ f) \implies \text{set\_integrable } M \ A \ (\lambda t. \ a \ * \ f \ t)$ 
  unfolding set_integrable_def

```


using *integrable_mult_right*[of a M $\lambda x.$ *indicator* A x $*_R$ f x] by *simp*

lemma *set_integrable_mult_right_iff* [*simp*]:
fixes $a :: 'a :: \{\text{real_normed_field, second_countable_topology}\}$
assumes $a \neq 0$
shows $\text{set_integrable } M A (\lambda t. a * f t) \longleftrightarrow \text{set_integrable } M A f$
proof
assume $\text{set_integrable } M A (\lambda t. a * f t)$
then have $\text{set_integrable } M A (\lambda t. 1/a * (a * f t))$
using *set_integrable_mult_right* by *blast*
then show $\text{set_integrable } M A f$
using *assms* by *auto*
qed *auto*

lemma *set_integrable_mult_left* [*simp, intro*]:
fixes $a :: 'a :: \{\text{real_normed_field, second_countable_topology}\}$
shows $(a \neq 0 \implies \text{set_integrable } M A f) \implies \text{set_integrable } M A (\lambda t. f t * a)$
unfolding *set_integrable_def*
using *integrable_mult_left*[of a M $\lambda x.$ *indicator* A x $*_R$ f x] by *simp*

lemma *set_integrable_mult_left_iff* [*simp*]:
fixes $a :: 'a :: \{\text{real_normed_field, second_countable_topology}\}$
assumes $a \neq 0$
shows $\text{set_integrable } M A (\lambda t. f t * a) \longleftrightarrow \text{set_integrable } M A f$
using *assms* by (*subst set_integrable_mult_right_iff* [*symmetric*]) (*auto simp: mult.commute*)

lemma *set_integrable_divide* [*simp, intro*]:
fixes $a :: 'a :: \{\text{real_normed_field, field, second_countable_topology}\}$
assumes $a \neq 0 \implies \text{set_integrable } M A f$
shows $\text{set_integrable } M A (\lambda t. f t / a)$
proof –
have $\text{integrable } M (\lambda x. \text{indicator } A x *_R f x / a)$
using *assms* **unfolding** *set_integrable_def* by (*rule integrable_divide_zero*)
also have $(\lambda x. \text{indicator } A x *_R f x / a) = (\lambda x. \text{indicator } A x *_R (f x / a))$
by (*auto split: split_indicator*)
finally show *?thesis*
unfolding *set_integrable_def* .
qed

lemma *set_integrable_mult_divide_iff* [*simp*]:
fixes $a :: 'a :: \{\text{real_normed_field, second_countable_topology}\}$
assumes $a \neq 0$
shows $\text{set_integrable } M A (\lambda t. f t / a) \longleftrightarrow \text{set_integrable } M A f$
by (*simp add: divide_inverse assms*)

lemma *set_integral_add* [*simp, intro*]:
fixes $f g :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $\text{set_integrable } M A f$ $\text{set_integrable } M A g$

shows $set_integrable\ M\ A\ (\lambda x. f\ x + g\ x)$
and $(LINT\ x:A|M. f\ x + g\ x) = (LINT\ x:A|M. f\ x) + (LINT\ x:A|M. g\ x)$
using *assms* **unfolding** $set_integrable_def\ set_lebesgue_integral_def$ **by** $(simp_all\ add:\ scaleR_add_right)$

lemma $set_integral_diff$ [*simp, intro*]:
assumes $set_integrable\ M\ A\ f\ set_integrable\ M\ A\ g$
shows $set_integrable\ M\ A\ (\lambda x. f\ x - g\ x)$ **and** $(LINT\ x:A|M. f\ x - g\ x) =$
 $(LINT\ x:A|M. f\ x) - (LINT\ x:A|M. g\ x)$
using *assms* **unfolding** $set_integrable_def\ set_lebesgue_integral_def$ **by** $(simp_all\ add:\ scaleR_diff_right)$

lemma $set_integral_uminus$: $set_integrable\ M\ A\ f \implies (LINT\ x:A|M. - f\ x) =$
 $- (LINT\ x:A|M. f\ x)$
unfolding $set_integrable_def\ set_lebesgue_integral_def$
by $(subst\ integral_minus[symmetric])\ simp_all$

lemma $set_integral_complex_of_real$:
 $(LINT\ x:A|M. complex_of_real\ (f\ x)) = of_real\ (LINT\ x:A|M. f\ x)$
unfolding $set_lebesgue_integral_def$
by $(subst\ integral_complex_of_real[symmetric])$
 $(auto\ intro!: Bochner_Integration.integral_cong\ split:\ split_indicator)$

lemma $set_integral_mono$:
fixes $f\ g :: _ \Rightarrow real$
assumes $set_integrable\ M\ A\ f\ set_integrable\ M\ A\ g$
 $\bigwedge x. x \in A \implies f\ x \leq g\ x$
shows $(LINT\ x:A|M. f\ x) \leq (LINT\ x:A|M. g\ x)$
using *assms* **unfolding** $set_integrable_def\ set_lebesgue_integral_def$
by $(auto\ intro:\ integral_mono\ split:\ split_indicator)$

lemma $set_integral_mono_AE$:
fixes $f\ g :: _ \Rightarrow real$
assumes $set_integrable\ M\ A\ f\ set_integrable\ M\ A\ g$
 $AE\ x \in A\ in\ M. f\ x \leq g\ x$
shows $(LINT\ x:A|M. f\ x) \leq (LINT\ x:A|M. g\ x)$
using *assms* **unfolding** $set_integrable_def\ set_lebesgue_integral_def$
by $(auto\ intro:\ integral_mono_AE\ split:\ split_indicator)$

lemma $set_integrable_abs$: $set_integrable\ M\ A\ f \implies set_integrable\ M\ A\ (\lambda x. |f\ x|)$
 $:: real)$
using $integrable_abs[of\ M\ \lambda x. f\ x * indicator\ A\ x]$ **unfolding** $set_integrable_def$
by $(simp\ add:\ abs_mult\ ac_simps)$

lemma $set_integrable_abs_iff$:
fixes $f :: _ \Rightarrow real$
shows $set_borel_measurable\ M\ A\ f \implies set_integrable\ M\ A\ (\lambda x. |f\ x|) = set_integrable\ M\ A\ f$
unfolding $set_integrable_def\ set_borel_measurable_def$

by (subst (2) integrable_abs_iff[symmetric]) (simp_all add: abs_mult ac_simps)

lemma set_integrable_abs_iff':

fixes $f :: _ \Rightarrow \text{real}$

shows $f \in \text{borel_measurable } M \Longrightarrow A \in \text{sets } M \Longrightarrow$

$\text{set_integrable } M A (\lambda x. |f x|) = \text{set_integrable } M A f$

by (simp add: set_borel_measurable_def set_integrable_abs_iff)

lemma set_integrable_discrete_difference:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$

assumes countable X

assumes $\text{diff}: (A - B) \cup (B - A) \subseteq X$

assumes $\bigwedge x. x \in X \Longrightarrow \text{emeasure } M \{x\} = 0 \bigwedge x. x \in X \Longrightarrow \{x\} \in \text{sets } M$

shows $\text{set_integrable } M A f \longleftrightarrow \text{set_integrable } M B f$

unfolding set_integrable_def

proof (rule integrable_discrete_difference[where $X=X$])

show $\bigwedge x. x \in \text{space } M \Longrightarrow x \notin X \Longrightarrow \text{indicator } A x *_{\mathbb{R}} f x = \text{indicator } B x *_{\mathbb{R}} f x$

using diff by (auto split: split_indicator)

qed fact+

lemma set_integral_discrete_difference:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$

assumes countable X

assumes $\text{diff}: (A - B) \cup (B - A) \subseteq X$

assumes $\bigwedge x. x \in X \Longrightarrow \text{emeasure } M \{x\} = 0 \bigwedge x. x \in X \Longrightarrow \{x\} \in \text{sets } M$

shows $\text{set_lebesgue_integral } M A f = \text{set_lebesgue_integral } M B f$

unfolding set_lebesgue_integral_def

proof (rule integral_discrete_difference[where $X=X$])

show $\bigwedge x. x \in \text{space } M \Longrightarrow x \notin X \Longrightarrow \text{indicator } A x *_{\mathbb{R}} f x = \text{indicator } B x *_{\mathbb{R}} f x$

using diff by (auto split: split_indicator)

qed fact+

lemma set_integrable_Un:

fixes $f g :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$

assumes $f_A: \text{set_integrable } M A f$ and $f_B: \text{set_integrable } M B f$

and [measurable]: $A \in \text{sets } M B \in \text{sets } M$

shows $\text{set_integrable } M (A \cup B) f$

proof -

have $\text{set_integrable } M (A - B) f$

using f_A by (rule set_integrable_subset) auto

with f_B have $\text{integrable } M (\lambda x. \text{indicator } (A - B) x *_{\mathbb{R}} f x + \text{indicator } B x *_{\mathbb{R}} f x)$

unfolding set_integrable_def using integrable_add by blast

then show ?thesis

unfolding set_integrable_def

by (rule integrable_cong[THEN iffD1, rotated 2]) (auto split: split_indicator)

qed

lemma *set_integrable_empty* [*simp*]: *set_integrable* $M \ \{\}$ f
by (*auto simp: set_integrable_def*)

lemma *set_integrable_UN*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes *finite* $I \wedge i. i \in I \implies \text{set_integrable } M \ (A \ i) \ f$
 $\wedge i. i \in I \implies A \ i \in \text{sets } M$
shows *set_integrable* $M \ (\bigcup i \in I. A \ i) \ f$
using *assms*
by (*induct I*) (*auto simp: set_integrable_Un sets.finite_UN*)

lemma *set_integral_Un*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes $A \cap B = \{\}$
and *set_integrable* $M \ A \ f$
and *set_integrable* $M \ B \ f$
shows $(LINT \ x:A \cup B | M. f \ x) = (LINT \ x:A | M. f \ x) + (LINT \ x:B | M. f \ x)$
using *assms*
unfolding *set_integrable_def set_lebesgue_integral_def*
by (*auto simp add: indicator_union_arith indicator_inter_arith[symmetric] scaleR_add_left*)

lemma *set_integral_cong_set*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes *set_borel_measurable* $M \ A \ f$ *set_borel_measurable* $M \ B \ f$
and $ae: AE \ x \ \text{in } M. x \in A \longleftrightarrow x \in B$
shows $(LINT \ x:B | M. f \ x) = (LINT \ x:A | M. f \ x)$
unfolding *set_lebesgue_integral_def*
proof (*rule integral_cong_AE*)
show $AE \ x \ \text{in } M. \text{indicator } B \ x \ *_{\mathbb{R}} \ f \ x = \text{indicator } A \ x \ *_{\mathbb{R}} \ f \ x$
using ae **by** (*auto simp: subset_eq split: split_indicator*)
qed (*use assms in <auto simp: set_borel_measurable_def>*)

proposition *set_borel_measurable_subset*:
fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$
assumes [*measurable*]: *set_borel_measurable* $M \ A \ f$ $B \in \text{sets } M$ **and** $B \subseteq A$
shows *set_borel_measurable* $M \ B \ f$
proof–
have *set_borel_measurable* $M \ B \ (\lambda x. \text{indicator } A \ x \ *_{\mathbb{R}} \ f \ x)$
using *assms* **unfolding** *set_borel_measurable_def*
using *borel_measurable_indicator borel_measurable_scaleR* **by** *blast*
moreover **have** $(\lambda x. \text{indicator } B \ x \ *_{\mathbb{R}} \ \text{indicator } A \ x \ *_{\mathbb{R}} \ f \ x) = (\lambda x. \text{indicator } B \ x \ *_{\mathbb{R}} \ f \ x)$
using $\langle B \subseteq A \rangle$ **by** (*auto simp: fun_eq_iff split: split_indicator*)
ultimately **show** *?thesis*
unfolding *set_borel_measurable_def* **by** *simp*
qed

lemma *set_integral_Un_AE*:

```

fixes f ::  $\_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
assumes ae:  $AE\ x\ in\ M. \neg (x \in A \wedge x \in B)$  and [measurable]:  $A \in sets\ M\ B \in sets\ M$ 
and set_integrable M A f
and set_integrable M B f
shows  $(LINT\ x:A \cup B|M. f\ x) = (LINT\ x:A|M. f\ x) + (LINT\ x:B|M. f\ x)$ 
proof -
  have f: set_integrable M (A  $\cup$  B) f
    by (intro set_integrable_Un assms)
  then have f': set_borel_measurable M (A  $\cup$  B) f
    using integrable_iff_bounded set_borel_measurable_def set_integrable_def by blast
  have  $(LINT\ x:A \cup B|M. f\ x) = (LINT\ x:(A - A \cap B) \cup (B - A \cap B)|M. f\ x)$ 
  proof (rule set_integral_cong_set)
    show  $AE\ x\ in\ M. (x \in A - A \cap B \cup (B - A \cap B)) = (x \in A \cup B)$ 
      using ae by auto
    show set_borel_measurable M (A - A  $\cap$  B  $\cup$  (B - A  $\cap$  B)) f
      using f' by (rule set_borel_measurable_subset) auto
  qed fact
  also have  $\dots = (LINT\ x:(A - A \cap B)|M. f\ x) + (LINT\ x:(B - A \cap B)|M. f\ x)$ 
    by (auto intro!: set_integral_Un set_integrable_subset[OF f])
  also have  $\dots = (LINT\ x:A|M. f\ x) + (LINT\ x:B|M. f\ x)$ 
    using ae
    by (intro arg_cong2[where f=(+)] set_integral_cong_set)
      (auto intro!: set_borel_measurable_subset[OF f'])
  finally show ?thesis .
qed

```

lemma set_integral_finite_Union:

```

fixes f ::  $\_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$ 
assumes finite I disjoint_family_on A I
  and  $\bigwedge i. i \in I \implies set\_integrable\ M\ (A\ i)\ f$   $\bigwedge i. i \in I \implies A\ i \in sets\ M$ 
shows  $(LINT\ x:(\bigcup i \in I. A\ i)|M. f\ x) = (\sum i \in I. LINT\ x:A\ i|M. f\ x)$ 
using assms
proof induction
  case (insert x F)
  then have  $A\ x \cap \bigcup (A\ ' F) = \{\}$ 
    by (meson disjoint_family_on_insert)
  with insert show ?case
    by (simp add: set_integral_Un set_integrable_Un set_integrable_UN disjoint_family_on_insert)
qed (simp add: set_lebesgue_integral_def)

```

lemma pos_integrable_to_top:

```

fixes l::real
assumes  $\bigwedge i. A\ i \in sets\ M$  mono A
assumes nneg:  $\bigwedge x\ i. x \in A\ i \implies 0 \leq f\ x$ 

```

```

and intgbl:  $\bigwedge i::\text{nat. set\_integrable } M (A\ i) f$ 
and lim:  $(\lambda i::\text{nat. LINT } x:A\ i|M. f\ x) \longrightarrow l$ 
shows set\_integrable  $M (\bigcup i. A\ i) f$ 
  unfolding set\_integrable\_def
  apply (rule integrable\_monotone\_convergence[where  $f = \lambda i::\text{nat. } \lambda x. \text{indicator}$ 
 $(A\ i) x *_R f\ x$  and  $x = l$ ])
    apply (rule intgbl [unfolded set\_integrable\_def])
    prefer 3 apply (rule lim [unfolded set\_lebesgue\_integral\_def])
    apply (rule AE\_I2)
  using  $\langle \text{mono } A \rangle$  apply (auto simp: mono\_def nneg split: split\_indicator) []
proof (rule AE\_I2)
  { fix  $x$  assume  $x \in \text{space } M$ 
    show  $(\lambda i. \text{indicator } (A\ i) x *_R f\ x) \longrightarrow \text{indicator } (\bigcup i. A\ i) x *_R f\ x$ 
    proof cases
      assume  $\exists i. x \in A\ i$ 
      then obtain  $i$  where  $x \in A\ i ..$ 
      then have  $\forall_F i$  in sequentially.  $x \in A\ i$ 
      using  $\langle x \in A\ i \rangle \langle \text{mono } A \rangle$  by (auto simp: eventually\_sequentially\_mono\_def)
      with eventually\_mono have  $\forall_F i$  in sequentially.  $\text{indicat\_real } (A\ i) x *_R f\ x = \text{indicat\_real } (\bigcup ( \text{range } A)) x *_R f\ x$ 
      by fastforce
      then show ?thesis
      by (intro tendsto\_eventually)
    qed auto }
  then show  $(\lambda x. \text{indicator } (\bigcup i. A\ i) x *_R f\ x) \in \text{borel\_measurable } M$ 
  apply (rule borel\_measurable\_LIMSEQ\_real)
  apply assumption
  using intgbl set\_integrable\_def by blast
qed

```

Proof from Royden, *Real Analysis*, p. 91.

```

lemma lebesgue\_integral\_countable\_add:
  fixes  $f :: \_ \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes meas[intro]:  $\bigwedge i::\text{nat. } A\ i \in \text{sets } M$ 
  and disj:  $\bigwedge i\ j. i \neq j \implies A\ i \cap A\ j = \{\}$ 
  and intgbl: set\_integrable  $M (\bigcup i. A\ i) f$ 
  shows  $(\text{LINT } x:(\bigcup i. A\ i)|M. f\ x) = (\sum i. (\text{LINT } x:(A\ i)|M. f\ x))$ 
  unfolding set\_lebesgue\_integral\_def
proof (subst integral\_suminf[symmetric])
  show int\_A: integrable  $M (\lambda x. \text{indicat\_real } (A\ i) x *_R f\ x)$  for  $i$ 
  using intgbl unfolding set\_integrable\_def [symmetric]
  by (rule set\_integrable\_subset) auto
  { fix  $x$  assume  $x \in \text{space } M$ 
    have  $(\lambda i. \text{indicator } (A\ i) x *_R f\ x)$  sums  $(\text{indicator } (\bigcup i. A\ i) x *_R f\ x)$ 
    by (intro sums\_scaleR\_left indicator\_sums) fact }
  note sums = this

  have norm\_f:  $\bigwedge i. \text{set\_integrable } M (A\ i) (\lambda x. \text{norm } (f\ x))$ 
  using int\_A[THEN integrable\_norm] unfolding set\_integrable\_def by auto

```

```

show  $AE\ x\ in\ M.\ summable\ (\lambda i.\ norm\ (indicator\ (A\ i)\ x\ *_R\ f\ x))$ 
using disj by (intro AE_I2) (auto intro!: summable_mult2_sums_summable[OF
indicator_sums])

show  $summable\ (\lambda i.\ LINT\ x|M.\ norm\ (indicator\ (A\ i)\ x\ *_R\ f\ x))$ 
proof (rule summableI_nonneg_bounded)
  fix n
  show  $0 \leq LINT\ x|M.\ norm\ (indicator\ (A\ n)\ x\ *_R\ f\ x)$ 
  using norm_f by (auto intro!: integral_nonneg_AE)

  have  $(\sum\ i < n.\ LINT\ x|M.\ norm\ (indicator\ (A\ i)\ x\ *_R\ f\ x)) = (\sum\ i < n.\ LINT$ 
 $x:A\ i|M.\ norm\ (f\ x))$ 
  by (simp add: abs_mult_set_lebesgue_integral_def)
  also have  $\dots = set\_lebesgue\_integral\ M\ (\bigcup\ i < n.\ A\ i)\ (\lambda x.\ norm\ (f\ x))$ 
  using norm_f
  by (subst set_integral_finite_Union) (auto simp: disjoint_family_on_def
disj)
  also have  $\dots \leq set\_lebesgue\_integral\ M\ (\bigcup\ i.\ A\ i)\ (\lambda x.\ norm\ (f\ x))$ 
  using intgbl[unfolded set_integrable_def, THEN integrable_norm] norm_f
  unfolding set_lebesgue_integral_def set_integrable_def
  apply (intro integral_mono set_integrable_UN[of  $\{.. < n\}$ ], unfolded set_integrable_def)
  apply (auto split: split_indicator)
  done
  finally show  $(\sum\ i < n.\ LINT\ x|M.\ norm\ (indicator\ (A\ i)\ x\ *_R\ f\ x)) \leq$ 
 $set\_lebesgue\_integral\ M\ (\bigcup\ i.\ A\ i)\ (\lambda x.\ norm\ (f\ x))$ 
  by simp
qed
show  $LINT\ x|M.\ indicator\ (\bigcup\ (A\ 'UNIV))\ x\ *_R\ f\ x = LINT\ x|M.\ (\sum\ i.\ indicator$ 
 $(A\ i)\ x\ *_R\ f\ x)$ 
  by (metis (no_types, lifting) integral_cong_sums_sums_unique)
qed

lemma set_integral_cont_up:
  fixes  $f :: \_ \Rightarrow 'a :: \{banach,\ second\_countable\_topology\}$ 
  assumes [measurable]:  $\bigwedge i.\ A\ i \in sets\ M$  and  $A: incseq\ A$ 
  and intgbl:  $set\_integrable\ M\ (\bigcup\ i.\ A\ i)\ f$ 
shows  $(\lambda i.\ LINT\ x:(A\ i)|M.\ f\ x) \longrightarrow (LINT\ x:(\bigcup\ i.\ A\ i)|M.\ f\ x)$ 
  unfolding set_lebesgue_integral_def
proof (intro integral_dominated_convergence[where  $w = \lambda x.\ indicator\ (\bigcup\ i.\ A\ i)$ 
 $x\ *_R\ norm\ (f\ x)$ ])
  have  $int\_A: \bigwedge i.\ set\_integrable\ M\ (A\ i)\ f$ 
  using intgbl by (rule set_integrable_subset) auto
  show  $\bigwedge i.\ (\lambda x.\ indicator\ (A\ i)\ x\ *_R\ f\ x) \in borel\_measurable\ M$ 
  using int_A_integrable_iff_bounded_set_integrable_def by blast
  show  $(\lambda x.\ indicator\ (\bigcup\ (A\ 'UNIV))\ x\ *_R\ f\ x) \in borel\_measurable\ M$ 
  using integrable_iff_bounded_intgbl_set_integrable_def by blast
  show  $integrable\ M\ (\lambda x.\ indicator\ (\bigcup\ i.\ A\ i)\ x\ *_R\ norm\ (f\ x))$ 
  using int_A_intgbl_integrable_norm unfolding set_integrable_def

```

```

    by fastforce
  { fix x i assume x ∈ A i
  with A have (λxa. indicator (A xa) x::real) → 1 ↔ (λxa. 1::real) →
  1
    by (intro filterlim_cong refl)
      (fastforce simp: eventually_sequentially_incseq_def subset_eq intro!: exI[of
  _ i]) }
  then show AE x in M. (λi. indicator (A i) x *R f x) → indicator (⋃ i. A
  i) x *R f x
    by (intro AE_I2 tendsto_intros) (auto split: split_indicator)
  qed (auto split: split_indicator)

```

lemma *set_integral_cont_down*:

```

  fixes f :: _ ⇒ 'a :: {banach, second_countable_topology}
  assumes [measurable]: ∧i. A i ∈ sets M and A: decseq A
  and int0: set_integrable M (A 0) f
  shows (λi::nat. LINT x:(A i)|M. f x) → (LINT x:(⋂ i. A i)|M. f x)
  unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
  have int_A: ∧i. set_integrable M (A i) f
    using int0 by (rule set_integrable_subset) (insert A, auto simp: decseq_def)
  have integrable M (λc. norm (indicat_real (A 0) c *R f c))
    by (metis (no_types) int0 integrable_norm set_integrable_def)
  then show integrable M (λx. indicator (A 0) x *R norm (f x))
    by force
  have set_integrable M (⋂ i. A i) f
    using int0 by (rule set_integrable_subset) (insert A, auto simp: decseq_def)
  with int_A show (λx. indicat_real (⋂ (A ' UNIV)) x *R f x) ∈ borel_measurable
  M
    ∧i. (λx. indicat_real (A i) x *R f x) ∈ borel_measurable M
    by (auto simp: set_integrable_def)
  show ∧i. AE x in M. norm (indicator (A i) x *R f x) ≤ indicator (A 0) x *R
  norm (f x)
    using A by (auto split: split_indicator simp: decseq_def)
  { fix x i assume x ∈ space M x ∉ A i
  with A have (λi. indicator (A i) x::real) → 0 ↔ (λi. 0::real) → 0
    by (intro filterlim_cong refl)
      (auto split: split_indicator simp: eventually_sequentially_decseq_def intro!:
  exI[of _ i]) }
  then show AE x in M. (λi. indicator (A i) x *R f x) → indicator (⋂ i. A
  i) x *R f x
    by (intro AE_I2 tendsto_intros) (auto split: split_indicator)
  qed

```

lemma *set_integral_at_point*:

```

  fixes a :: real
  assumes set_integrable M {a} f
  and [simp]: {a} ∈ sets M and (emeasure M) {a} ≠ ∞

```



```

  shows (LINT x:{a} | M. f x) = f a * measure M {a}
proof -
  have set_lebesgue_integral M {a} f = set_lebesgue_integral M {a} (%x. f a)
    by (intro set_lebesgue_integral_cong) simp_all
  then show ?thesis using assms
    unfolding set_lebesgue_integral_def by simp
qed

```

9.3 Complex integrals

abbreviation $\text{complex_integrable} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{bool}$ **where**
 $\text{complex_integrable } M f \equiv \text{integrable } M f$

abbreviation $\text{complex_lebesgue_integral} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{complex}) \Rightarrow \text{complex}$ (integral^C) **where**
 $\text{integral}^C M f == \text{integral}^L M f$

syntax

```

_complex_lebesgue_integral :: pttrn  $\Rightarrow$  complex  $\Rightarrow$  'a measure  $\Rightarrow$  complex
( $\int^C \_ . \_ \partial \_ [0,0] 110$ )

```

translations

```

 $\int^C x. f \partial M == \text{CONST } \text{complex\_lebesgue\_integral } M (\lambda x. f)$ 

```

syntax

```

_ascii_complex_lebesgue_integral :: pttrn  $\Rightarrow$  'a measure  $\Rightarrow$  real  $\Rightarrow$  real
(( $\int \text{CLINT } \_ | \_ . \_ [0,0,10] 10$ )

```

translations

```

CLINT x|M. f == CONST complex_lebesgue_integral M ( $\lambda x. f$ )

```

lemma $\text{complex_integrable_cnj}$ [simp]:

```

complex_integrable M ( $\lambda x. \text{cnj } (f x)$ )  $\longleftrightarrow$  complex_integrable M f

```

proof

```

assume complex_integrable M ( $\lambda x. \text{cnj } (f x)$ )
then have complex_integrable M ( $\lambda x. \text{cnj } (\text{cnj } (f x))$ )
  by (rule integrable_cnj)
then show complex_integrable M f
  by simp

```

qed simp

lemma $\text{complex_of_real_integrable_eq}$:

```

complex_integrable M ( $\lambda x. \text{complex\_of\_real } (f x)$ )  $\longleftrightarrow$  integrable M f

```

proof

```

assume complex_integrable M ( $\lambda x. \text{complex\_of\_real } (f x)$ )
then have integrable M ( $\lambda x. \text{Re } (\text{complex\_of\_real } (f x))$ )
  by (rule integrable_Re)
then show integrable M f
  by simp

```

2822

qed simp

abbreviation *complex_set_integrable* :: 'a measure \Rightarrow 'a set \Rightarrow ('a \Rightarrow complex) \Rightarrow bool **where**
 complex_set_integrable M A f \equiv *set_integrable* M A f

abbreviation *complex_set_lebesgue_integral* :: 'a measure \Rightarrow 'a set \Rightarrow ('a \Rightarrow complex) \Rightarrow complex **where**
 complex_set_lebesgue_integral M A f \equiv *set_lebesgue_integral* M A f

syntax

_ascii_complex_set_lebesgue_integral :: pttrn \Rightarrow 'a set \Rightarrow 'a measure \Rightarrow real \Rightarrow real
 ((*CLINT* _:_|. _) [0,0,0,10] 10)

translations

CLINT x:A|M. f == *CONST* *complex_set_lebesgue_integral* M A (λ x. f)

lemma *set_measurable_continuous_on_ivl*:

assumes *continuous_on* {a..b} (f :: real \Rightarrow real)

shows *set_borel_measurable* borel {a..b} f

unfolding *set_borel_measurable_def*

by (rule *borel_measurable_continuous_on_indicator*[*OF* _ *assms*]) simp

9.4 NN Set Integrals

This notation is from Sébastien Gouëzel: His use is not directly in line with the notations in this file, they are more in line with sum, and more readable he thinks.

abbreviation *set_nn_integral* M A f \equiv *nn_integral* M (λ x. f x * *indicator* A x)

syntax

_set_nn_integral :: pttrn \Rightarrow 'a set \Rightarrow 'a measure \Rightarrow ereal \Rightarrow ereal
 ((*f* ⁺((_) \in (_) / _) / ∂ _) [0,0,0,110] 10)

_set_lebesgue_integral :: pttrn \Rightarrow 'a set \Rightarrow 'a measure \Rightarrow ereal \Rightarrow ereal
 ((*f* ((_) \in (_) / _) / ∂ _) [0,0,0,110] 10)

translations

\int ⁺x \in A. f ∂ M == *CONST* *set_nn_integral* M A (λ x. f)

\int x \in A. f ∂ M == *CONST* *set_lebesgue_integral* M A (λ x. f)

lemma *set_nn_integral_cong*:

assumes M = M' A = B \wedge x. x \in space M \cap A \implies f x = g x

shows *set_nn_integral* M A f = *set_nn_integral* M' B g

by (*metis* (*mono_tags*, *lifting*) *IntI* *assms* *indicator_simps*(2) *mult_eq_0_iff* *nn_integral_cong*)

lemma *nn_integral_disjoint_pair*:

assumes [*measurable*]: $f \in \text{borel_measurable } M$

$B \in \text{sets } M \ C \in \text{sets } M$

$B \cap C = \{\}$

shows $(\int^{+x} \in B \cup C. f x \ \partial M) = (\int^{+x} \in B. f x \ \partial M) + (\int^{+x} \in C. f x \ \partial M)$

proof –

have *mes*: $\bigwedge D. D \in \text{sets } M \implies (\lambda x. f x * \text{indicator } D x) \in \text{borel_measurable } M$

by *simp*

have *pos*: $\bigwedge D. \text{AE } x \text{ in } M. f x * \text{indicator } D x \geq 0$ **using** *assms(2)* **by** *auto*

have $\bigwedge x. f x * \text{indicator } (B \cup C) x = f x * \text{indicator } B x + f x * \text{indicator } C x$

using *assms(4)*

by (*auto split: split_indicator*)

then have $(\int^{+x}. f x * \text{indicator } (B \cup C) x \ \partial M) = (\int^{+x}. f x * \text{indicator } B x + f x * \text{indicator } C x \ \partial M)$

by *simp*

also have $\dots = (\int^{+x}. f x * \text{indicator } B x \ \partial M) + (\int^{+x}. f x * \text{indicator } C x \ \partial M)$

by (*rule nn_integral_add*) (*auto simp add: assms mes pos*)

finally show *?thesis* **by** *simp*

qed

lemma *nn_integral_disjoint_pair_countspace*:

assumes $B \cap C = \{\}$

shows $(\int^{+x} \in B \cup C. f x \ \partial \text{count_space } UNIV) = (\int^{+x} \in B. f x \ \partial \text{count_space } UNIV) + (\int^{+x} \in C. f x \ \partial \text{count_space } UNIV)$

by (*rule nn_integral_disjoint_pair*) (*simp_all add: assms*)

lemma *nn_integral_null_delta*:

assumes $A \in \text{sets } M \ B \in \text{sets } M$

$(A - B) \cup (B - A) \in \text{null_sets } M$

shows $(\int^{+x} \in A. f x \ \partial M) = (\int^{+x} \in B. f x \ \partial M)$

proof (*rule nn_integral_cong_AE*)

have ***: $\text{AE } a \text{ in } M. a \notin (A - B) \cup (B - A)$

using *assms(3)* *AE_not_in* **by** *blast*

then show $\langle \text{AE } x \text{ in } M. f x * \text{indicator } A x = f x * \text{indicator } B x \rangle$

by *auto*

qed

proposition *nn_integral_disjoint_family*:

assumes [*measurable*]: $f \in \text{borel_measurable } M \ \bigwedge (n::\text{nat}). B n \in \text{sets } M$

and *disjoint_family* *B*

shows $(\int^{+x} \in (\bigcup n. B n). f x \ \partial M) = (\sum n. (\int^{+x} \in B n. f x \ \partial M))$

proof –

have $(\int^{+x} x. (\sum n. f x * \text{indicator } (B n) x) \ \partial M) = (\sum n. (\int^{+x} x. f x * \text{indicator } (B n) x \ \partial M))$

by (*rule nn_integral_suminf*) *simp*

moreover have $(\sum n. f x * \text{indicator } (B n) x) = f x * \text{indicator } (\bigcup n. B n) x$
for *x*

```

proof (cases)
  assume  $x \in (\bigcup n. B n)$ 
  then obtain  $n$  where  $x \in B n$  by blast
  have  $a: \text{finite } \{n\}$  by simp
  have  $\bigwedge i. i \neq n \implies x \notin B i$  using  $\langle x \in B n \rangle$  assms(3) disjoint_family_on_def
    by (metis IntI UNIV_I empty_iff)
  then have  $\bigwedge i. i \notin \{n\} \implies \text{indicator } (B i) x = (0::\text{ennreal})$  using indicator_def
by simp
  then have  $b: \bigwedge i. i \notin \{n\} \implies f x * \text{indicator } (B i) x = (0::\text{ennreal})$  by simp

  define  $h$  where  $h = (\lambda i. f x * \text{indicator } (B i) x)$ 
  then have  $\bigwedge i. i \notin \{n\} \implies h i = 0$  using  $b$  by simp
  then have  $(\sum i. h i) = (\sum i \in \{n\}. h i)$ 
    by (metis sums_unique[OF sums_finite[OF a]])
  then have  $(\sum i. h i) = h n$  by simp
  then have  $(\sum n. f x * \text{indicator } (B n) x) = f x * \text{indicator } (B n) x$  using
h_def by simp
  then have  $(\sum n. f x * \text{indicator } (B n) x) = f x$  using  $\langle x \in B n \rangle$  indicator_def
by simp
  then show ?thesis using  $\langle x \in (\bigcup n. B n) \rangle$  by auto
next
  assume  $x \notin (\bigcup n. B n)$ 
  then have  $\bigwedge n. f x * \text{indicator } (B n) x = 0$  by simp
  have  $(\sum n. f x * \text{indicator } (B n) x) = 0$ 
    by (simp add:  $\langle \bigwedge n. f x * \text{indicator } (B n) x = 0 \rangle$ )
  then show ?thesis using  $\langle x \notin (\bigcup n. B n) \rangle$  by auto
qed
ultimately show ?thesis by simp
qed

```

lemma *nn_set_integral_add*:

```

assumes [measurable]:  $f \in \text{borel\_measurable } M$   $g \in \text{borel\_measurable } M$ 
   $A \in \text{sets } M$ 
shows  $(\int^+ x \in A. (f x + g x) \partial M) = (\int^+ x \in A. f x \partial M) + (\int^+ x \in A. g x \partial M)$ 
proof -
  have  $(\int^+ x \in A. (f x + g x) \partial M) = (\int^+ x. (f x * \text{indicator } A x + g x * \text{indicator } A x) \partial M)$ 
    by (auto simp add: indicator_def intro!: nn_integral_cong)
  also have  $\dots = (\int^+ x. f x * \text{indicator } A x \partial M) + (\int^+ x. g x * \text{indicator } A x \partial M)$ 
    by (rule nn_integral_add) using assms(1) assms(2) by auto
  finally show ?thesis by simp
qed

```

lemma *nn_set_integral_cong*:

```

assumes  $AE x \text{ in } M. f x = g x$ 
shows  $(\int^+ x \in A. f x \partial M) = (\int^+ x \in A. g x \partial M)$ 
apply (rule nn_integral_cong_AE) using assms(1) by auto

```

lemma *nn_set_integral_set_mono*:

$A \subseteq B \implies (\int^+ x \in A. f x \partial M) \leq (\int^+ x \in B. f x \partial M)$

by (*auto intro!*: *nn_integral_mono split: split_indicator*)

lemma *nn_set_integral_mono*:

assumes [*measurable*]: $f \in \text{borel_measurable } M$ $g \in \text{borel_measurable } M$
 $A \in \text{sets } M$

and $AE\ x \in A\ \text{in } M. f\ x \leq g\ x$

shows $(\int^+ x \in A. f x \partial M) \leq (\int^+ x \in A. g x \partial M)$

by (*auto intro!*: *nn_integral_mono_AE split: split_indicator simp: assms*)

lemma *nn_set_integral_space [simp]*:

shows $(\int^+ x \in \text{space } M. f x \partial M) = (\int^+ x. f x \partial M)$

by (*metis (mono_tags, lifting) indicator_simps(1) mult.right_neutral nn_integral_cong*)

lemma *nn_integral_count_compose_inj*:

assumes *inj_on* $g\ A$

shows $(\int^+ x \in A. f (g\ x) \partial \text{count_space } UNIV) = (\int^+ y \in g'A. f\ y \partial \text{count_space } UNIV)$

proof –

have $(\int^+ x \in A. f (g\ x) \partial \text{count_space } UNIV) = (\int^+ x. f (g\ x) \partial \text{count_space } A)$

by (*auto simp add: nn_integral_count_space_indicator[symmetric]*)

also have $\dots = (\int^+ y. f\ y \partial \text{count_space } (g'A))$

by (*simp add: assms nn_integral_bij_count_space inj_on_imp_bij_betw*)

also have $\dots = (\int^+ y \in g'A. f\ y \partial \text{count_space } UNIV)$

by (*auto simp add: nn_integral_count_space_indicator[symmetric]*)

finally show *?thesis* **by** *simp*

qed

lemma *nn_integral_count_compose_bij*:

assumes *bij_betw* $g\ A\ B$

shows $(\int^+ x \in A. f (g\ x) \partial \text{count_space } UNIV) = (\int^+ y \in B. f\ y \partial \text{count_space } UNIV)$

proof –

have *inj_on* $g\ A$ **using** *assms bij_betw_def* **by** *auto*

then have $(\int^+ x \in A. f (g\ x) \partial \text{count_space } UNIV) = (\int^+ y \in g'A. f\ y \partial \text{count_space } UNIV)$

by (*rule nn_integral_count_compose_inj*)

then show *?thesis* **using** *assms* **by** (*simp add: bij_betw_def*)

qed

lemma *set_integral_null_delta*:

fixes $f :: _ \Rightarrow _ :: \{\text{banach, second_countable_topology}\}$

assumes [*measurable*]: *integrable* $M\ f\ A \in \text{sets } M\ B \in \text{sets } M$

and *null*: $(A - B) \cup (B - A) \in \text{null_sets } M$

shows $(\int x \in A. f x \partial M) = (\int x \in B. f x \partial M)$

proof (*rule set_integral_cong_set*)

have $*$: $AE\ a\ \text{in } M. a \notin (A - B) \cup (B - A)$

using *null AE_not_in* **by** *blast*

```

then show  $\forall x \in M. (x \in B) = (x \in A)$ 
  by auto
qed (simp_all add: set_borel_measurable_def)

```

```

lemma set_integral_space:
  assumes integrable M f
  shows  $(\int x \in \text{space } M. f x \partial M) = (\int x. f x \partial M)$ 
  by (metis (no_types, lifting) indicator_simps(1) integral_cong scaleR_one set_lebesgue_integral_def)

```

```

lemma null_if_pos_func_has_zero_nn_int:
  fixes  $f :: 'a \Rightarrow \text{ennreal}$ 
  assumes [measurable]:  $f \in \text{borel\_measurable } M$   $A \in \text{sets } M$ 
  and  $\forall x \in A \text{ in } M. f x > 0 \implies (\int^+ x \in A. f x \partial M) = 0$ 
  shows  $A \in \text{null\_sets } M$ 
proof -
  have  $\forall x \text{ in } M. f x * \text{indicator } A x = 0$ 
    by (subst nn_integral_0_iff_AE[symmetric], auto simp add: assms(4))
  then have  $\forall x \in A \text{ in } M. \text{False}$  using assms(3) by auto
  then show  $A \in \text{null\_sets } M$  using assms(2) by (simp add: AE_iff_null_sets)
qed

```

```

lemma null_if_pos_func_has_zero_int:
  assumes [measurable]:  $f \in \text{integrable } M$   $A \in \text{sets } M$ 
  and  $\forall x \in A \text{ in } M. f x > 0 \implies (\int x \in A. f x \partial M) = (0 :: \text{real})$ 
  shows  $A \in \text{null\_sets } M$ 
proof -
  have  $\forall x \text{ in } M. \text{indicator } A x * f x = 0$ 
    apply (subst integral_nonneg_eq_0_iff_AE[symmetric])
    using assms integrable_mult_indicator[OF A \in sets M] assms(1)
    by (auto simp: set_lebesgue_integral_def)
  then have  $\forall x \in A \text{ in } M. f x = 0$  by auto
  then have  $\forall x \in A \text{ in } M. \text{False}$  using assms(3) by auto
  then show  $A \in \text{null\_sets } M$  using assms(2) by (simp add: AE_iff_null_sets)
qed

```

The next lemma is a variant of *density_unique*. Note that it uses the notation for nonnegative set integrals introduced earlier.

```

lemma (in sigma_finite_measure) density_unique2:
  assumes [measurable]:  $f \in \text{borel\_measurable } M$   $f' \in \text{borel\_measurable } M$ 
  assumes density_eq:  $\bigwedge A. A \in \text{sets } M \implies (\int^+ x \in A. f x \partial M) = (\int^+ x \in A. f' x \partial M)$ 
  shows  $\forall x \text{ in } M. f x = f' x$ 
proof (rule density_unique)
  show  $\text{density } M f = \text{density } M f'$ 
    by (intro measure_eqI (auto simp: emeasure_density intro!: density_eq))
qed (auto simp add: assms)

```

The next lemma implies the same statement for Banach-space valued functions using Hahn-Banach theorem and linear forms. Since they are not yet

easily available, I only formulate it for real-valued functions.

lemma *density_unique_real*:

```

fixes  $f f': \_ \Rightarrow \text{real}$ 
assumes  $M[\text{measurable}]: \text{integrable } M f \text{ integrable } M f'$ 
assumes  $\text{density\_eq}: \bigwedge A. A \in \text{sets } M \implies (\int x \in A. f x \partial M) = (\int x \in A. f' x \partial M)$ 
shows  $AE\ x \text{ in } M. f\ x = f'\ x$ 
proof -
  define  $A$  where  $A = \{x \in \text{space } M. f\ x < f'\ x\}$ 
  then have  $[\text{measurable}]: A \in \text{sets } M$  by simp
  have  $(\int x \in A. (f'\ x - f\ x) \partial M) = (\int x \in A. f'\ x \partial M) - (\int x \in A. f\ x \partial M)$ 
    using  $\langle A \in \text{sets } M \rangle M \text{ integrable\_mult\_indicator set\_integrable\_def}$  by blast
  then have  $(\int x \in A. (f'\ x - f\ x) \partial M) = 0$  using assms(3) by simp
  then have  $A \in \text{null\_sets } M$ 
    using  $A\_def \text{ null\_if\_pos\_func\_has\_zero\_int}[\text{where } ?f = \lambda x. f'\ x - f\ x \text{ and } ?A = A]$  assms by auto
  then have  $AE\ x \text{ in } M. x \notin A$  by  $(\text{simp add: } AE\_not\_in)$ 
  then have  $*$ :  $AE\ x \text{ in } M. f'\ x \leq f\ x$  unfolding  $A\_def$  by auto

  define  $B$  where  $B = \{x \in \text{space } M. f'\ x < f\ x\}$ 
  then have  $[\text{measurable}]: B \in \text{sets } M$  by simp
  have  $(\int x \in B. (f\ x - f'\ x) \partial M) = (\int x \in B. f\ x \partial M) - (\int x \in B. f'\ x \partial M)$ 
    using  $\langle B \in \text{sets } M \rangle M \text{ integrable\_mult\_indicator set\_integrable\_def}$  by blast
  then have  $(\int x \in B. (f\ x - f'\ x) \partial M) = 0$  using assms(3) by simp
  then have  $B \in \text{null\_sets } M$ 
    using  $B\_def \text{ null\_if\_pos\_func\_has\_zero\_int}[\text{where } ?f = \lambda x. f\ x - f'\ x \text{ and } ?A = B]$  assms by auto
  then have  $AE\ x \text{ in } M. x \notin B$  by  $(\text{simp add: } AE\_not\_in)$ 
  then have  $AE\ x \text{ in } M. f'\ x \geq f\ x$  unfolding  $B\_def$  by auto
  then show  $?thesis$  using  $*$  by auto
qed

```

9.5 Scheffé's lemma

The next lemma shows that L^1 convergence of a sequence of functions follows from almost everywhere convergence and the weaker condition of the convergence of the integrated norms (or even just the nontrivial inequality about them). Useful in a lot of contexts! This statement (or its variations) are known as Scheffe lemma.

The formalization is more painful as one should jump back and forth between reals and ereals and justify all the time positivity or integrability (thankfully, measurability is handled more or less automatically).

proposition *Scheffe_lemma1*:

```

assumes  $\bigwedge n. \text{integrable } M (F\ n) \text{ integrable } M f$ 
   $AE\ x \text{ in } M. (\lambda n. F\ n\ x) \longrightarrow f\ x$ 
   $\limsup (\lambda n. \int^+ x. \text{norm}(F\ n\ x) \partial M) \leq (\int^+ x. \text{norm}(f\ x) \partial M)$ 
shows  $(\lambda n. \int^+ x. \text{norm}(F\ n\ x - f\ x) \partial M) \longrightarrow 0$ 

```

proof –

```

have [measurable]:  $\bigwedge n. F\ n \in \text{borel\_measurable } M\ f \in \text{borel\_measurable } M$ 
  using assms(1) assms(2) by simp_all
define G where  $G = (\lambda n\ x. \text{norm}(f\ x) + \text{norm}(F\ n\ x) - \text{norm}(F\ n\ x - f\ x))$ 
have [measurable]:  $\bigwedge n. G\ n \in \text{borel\_measurable } M$  unfolding G_def by simp
have G_pos[simp]:  $\bigwedge n\ x. G\ n\ x \geq 0$ 
  unfolding G_def by (metis ge_iff_diff_ge_0 norm_minus_commute norm_triangle_ineq4)
have finint:  $(\int^+ x. \text{norm}(f\ x)\ \partial M) \neq \infty$ 
  using has_bochner_integral_implies_finite_norm[OF has_bochner_integral_integrable[OF
<integrable M f>]]
  by simp
then have fin2:  $2 * (\int^+ x. \text{norm}(f\ x)\ \partial M) \neq \infty$ 
  by (auto simp: ennreal_mult_eq_top_iff)

{
  fix x assume *:  $(\lambda n. F\ n\ x) \longrightarrow f\ x$ 
  then have  $(\lambda n. \text{norm}(F\ n\ x)) \longrightarrow \text{norm}(f\ x)$  using tendsto_norm by blast
  moreover have  $(\lambda n. \text{norm}(F\ n\ x - f\ x)) \longrightarrow 0$  using * Lim_null tend-
sto_norm_zero_iff by fastforce
  ultimately have a:  $(\lambda n. \text{norm}(F\ n\ x) - \text{norm}(F\ n\ x - f\ x)) \longrightarrow \text{norm}(f$ 
x) using tendsto_diff by fastforce
  have  $(\lambda n. \text{norm}(f\ x) + (\text{norm}(F\ n\ x) - \text{norm}(F\ n\ x - f\ x))) \longrightarrow \text{norm}(f$ 
x)  $+ \text{norm}(f\ x)$ 
  by (rule tendsto_add) (auto simp add: a)
  moreover have  $\bigwedge n. G\ n\ x = \text{norm}(f\ x) + (\text{norm}(F\ n\ x) - \text{norm}(F\ n\ x - f$ 
x)) unfolding G_def by simp
  ultimately have  $(\lambda n. G\ n\ x) \longrightarrow 2 * \text{norm}(f\ x)$  by simp
  then have  $(\lambda n. \text{ennreal}(G\ n\ x)) \longrightarrow \text{ennreal}(2 * \text{norm}(f\ x))$  by simp
  then have liminf  $(\lambda n. \text{ennreal}(G\ n\ x)) = \text{ennreal}(2 * \text{norm}(f\ x))$ 
  using sequentially_bot tendsto_iff_Liminf_eq_Limsup by blast
}
then have AE x in M. liminf  $(\lambda n. \text{ennreal}(G\ n\ x)) = \text{ennreal}(2 * \text{norm}(f\ x))$ 
using assms(3) by auto
then have  $(\int^+ x. \text{liminf } (\lambda n. \text{ennreal } (G\ n\ x))\ \partial M) = (\int^+ x. 2 * \text{ennreal}(\text{norm}(f$ 
x))\ \partial M)
  by (simp add: nn_integral_cong_AE ennreal_mult)
  also have  $\dots = 2 * (\int^+ x. \text{norm}(f\ x)\ \partial M)$  by (rule nn_integral_cmult) auto
  finally have int_liminf:  $(\int^+ x. \text{liminf } (\lambda n. \text{ennreal } (G\ n\ x))\ \partial M) = 2 * (\int^+$ 
x. norm}(f\ x)\ \partial M)
  by simp

have  $(\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M) = (\int^+ x. \text{norm}(f$ 
x)\ \partial M)  $+ (\int^+ x. \text{norm}(F\ n\ x)\ \partial M)$  for n
  by (rule nn_integral_add) (auto simp add: assms)
then have limsup  $(\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f\ x)) + \text{ennreal}(\text{norm}(F\ n\ x))\ \partial M))$ 
=
  limsup  $(\lambda n. (\int^+ x. \text{norm}(f\ x)\ \partial M) + (\int^+ x. \text{norm}(F\ n\ x)\ \partial M))$ 
  by simp
also have  $\dots = (\int^+ x. \text{norm}(f\ x)\ \partial M) + \text{limsup } (\lambda n. (\int^+ x. \text{norm}(F\ n\ x)\ \partial M))$ 

```


by (rule *Limsup_const_add*, auto simp add: *finint*)
 also have $\dots \leq (\int^+ x. \text{norm}(f x) \partial M) + (\int^+ x. \text{norm}(f x) \partial M)$
 using *assms(4)* by (simp add: *add_left_mono*)
 also have $\dots = 2 * (\int^+ x. \text{norm}(f x) \partial M)$
 unfolding *one_add_one[symmetric]* *distrib_right* by *simp*
 ultimately have $a: \text{limsup } (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) \partial M)) \leq$
 $2 * (\int^+ x. \text{norm}(f x) \partial M)$ by *simp*

have $le: \text{ennreal } (\text{norm } (F n x - f x)) \leq \text{ennreal } (\text{norm } (f x)) + \text{ennreal } (\text{norm } (F n x))$ for $n x$
 by (simp add: *norm_minus_commute* *norm_triangle_ineq4* *ennreal_minus_flip: ennreal_plus*)
 then have $le2: (\int^+ x. \text{ennreal } (\text{norm } (F n x - f x)) \partial M) \leq (\int^+ x. \text{ennreal } (\text{norm } (f x)) + \text{ennreal } (\text{norm } (F n x)) \partial M)$ for n
 by (rule *nn_integral_mono*)

have $2 * (\int^+ x. \text{norm}(f x) \partial M) = (\int^+ x. \text{liminf } (\lambda n. \text{ennreal } (G n x)) \partial M)$
 by (simp add: *int_liminf*)
 also have $\dots \leq \text{liminf } (\lambda n. (\int^+ x. G n x \partial M))$
 by (rule *nn_integral_liminf*) auto
 also have $\text{liminf } (\lambda n. (\int^+ x. G n x \partial M)) =$
 $\text{liminf } (\lambda n. (\int^+ x. \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) \partial M) - (\int^+ x. \text{norm}(F n x - f x) \partial M))$
 proof (intro *arg_cong[where f=liminf]* ext)
 fix n
 have $\bigwedge x. \text{ennreal}(G n x) = \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) - \text{ennreal}(\text{norm}(F n x - f x))$
 unfolding *G_def* by (simp add: *ennreal_minus_flip: ennreal_plus*)
 moreover have $(\int^+ x. \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) - \text{ennreal}(\text{norm}(F n x - f x)) \partial M)$
 $= (\int^+ x. \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) \partial M) - (\int^+ x. \text{norm}(F n x - f x) \partial M)$
 proof (rule *nn_integral_diff*)
 from le show $AE x \text{ in } M. \text{ennreal } (\text{norm } (F n x - f x)) \leq \text{ennreal } (\text{norm } (f x)) + \text{ennreal } (\text{norm } (F n x))$
 by *simp*
 from $le2$ have $(\int^+ x. \text{ennreal } (\text{norm } (F n x - f x)) \partial M) < \infty$ using *assms(1)* *assms(2)*
 by (metis *has_bochner_integral_implies_finite_norm_integrable_simps* *Bochner_Integration.integrable_diff*)
 then show $(\int^+ x. \text{ennreal } (\text{norm } (F n x - f x)) \partial M) \neq \infty$ by *simp*
 qed (auto simp add: *assms*)
 ultimately show $(\int^+ x. G n x \partial M) = (\int^+ x. \text{ennreal}(\text{norm}(f x)) + \text{ennreal}(\text{norm}(F n x)) \partial M) - (\int^+ x. \text{norm}(F n x - f x) \partial M)$
 by *simp*
 qed
 finally have $2 * (\int^+ x. \text{norm}(f x) \partial M) + \text{limsup } (\lambda n. (\int^+ x. \text{norm}(F n x - f x) \partial M)) \leq$

```

    liminf (λn. (∫+x. ennreal(norm(f x)) + ennreal(norm(F n x)) ∂M) - (∫+x.
norm(F n x - f x) ∂M)) +
    limsup (λn. (∫+x. norm(F n x - f x) ∂M))
  by (intro add_mono) auto
  also have ... ≤ (limsup (λn. ∫+x. ennreal(norm(f x)) + ennreal(norm(F n x))
∂M) - limsup (λn. ∫+x. norm(F n x - f x) ∂M)) +
    limsup (λn. (∫+x. norm(F n x - f x) ∂M))
  by (intro add_mono liminf_minus_ennreal le2) auto
  also have ... = limsup (λn. (∫+x. ennreal(norm(f x)) + ennreal(norm(F n x))
∂M))
  by (intro diff_add_cancel_ennreal Limsup_mono always_eventually_all le2)
  also have ... ≤ 2 * (∫+x. norm(f x) ∂M)
  by fact
  finally have limsup (λn. (∫+x. norm(F n x - f x) ∂M)) = 0
  using fin2 by simp
  then show ?thesis
  by (rule tendsto_0_if_Limsup_eq_0_ennreal)
qed

```

proposition *Scheffe_lemma2*:

```

  fixes F::nat ⇒ 'a ⇒ 'b::{banach, second_countable_topology}
  assumes ∧ n::nat. F n ∈ borel_measurable M integrable M f
    AE x in M. (λn. F n x) ⟶ f x
    ∧n. (∫+x. norm(F n x) ∂M) ≤ (∫+x. norm(f x) ∂M)
  shows (λn. ∫+x. norm(F n x - f x) ∂M) ⟶ 0
proof (rule Scheffe_lemma1)
  fix n::nat
  have (∫+x. norm(f x) ∂M) < ∞ using assms(2) by (metis has_bochner_integral_implies_finite_norm
integrable.cases)
  then have (∫+x. norm(F n x) ∂M) < ∞ using assms(4)[of n] by auto
  then show integrable M (F n) by (subst integrable_iff_bounded, simp add:
assms(1)[of n])
qed (auto simp add: assms Limsup_bounded)

```

lemma *tendsto_set_lebesgue_integral_at_right*:

```

  fixes a b :: real and f :: real ⇒ 'a :: {banach, second_countable_topology}
  assumes a < b and sets: ∧a'. a' ∈ {a<..b} ⟹ {a'..b} ∈ sets M
    and set_integrable M {a<..b} f
  shows ((λa'. set_lebesgue_integral M {a'..b} f) ⟶
    set_lebesgue_integral M {a<..b} f) (at_right a)
proof (rule tendsto_at_right_sequentially[OF assms(1)], goal_cases)
  case (1 S)
  have eq: (⋃ n. {S n..b}) = {a<..b}
  proof safe
    fix x n assume x ∈ {S n..b}
    with 1(1,2)[of n] show x ∈ {a<..b} by auto
  next
    fix x assume x ∈ {a<..b}
    with order_tendstoD[OF ‹S ⟶ a›, of x] show x ∈ (⋃ n. {S n..b})

```

```

    by (force simp: eventually_at_top_linorder dest: less_imp_le)
  qed
  have ( $\lambda n. \text{set\_lebesgue\_integral } M \{S\ n..b\} f$ )  $\longrightarrow$   $\text{set\_lebesgue\_integral } M$ 
  ( $\bigcup n. \{S\ n..b\}$ )  $f$ 
    by (rule set_integral_cont_up) (insert assms 1, auto simp: eq_incseq_def dec-
seq_def less_imp_le)
  with eq show ?case by simp
  qed

```

9.6 Convergence of integrals over an interval

The next lemmas relate convergence of integrals over an interval to improper integrals.

```

lemma tendsto_set_lebesgue_integral_at_left:
  fixes  $a\ b :: \text{real}$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes  $a < b$  and sets:  $\bigwedge b'. b' \in \{a..<b\} \implies \{a..b'\} \in \text{sets } M$ 
    and  $\text{set\_integrable } M \{a..<b\} f$ 
  shows  $((\lambda b'. \text{set\_lebesgue\_integral } M \{a..b'\} f) \longrightarrow$ 
 $\text{set\_lebesgue\_integral } M \{a..<b\} f)$  (at_left b)
proof (rule tendsto_at_left_sequentially[OF assms(1)], goal_cases)
  case (1 S)
  have eq:  $(\bigcup n. \{a..S\ n\}) = \{a..<b\}$ 
  proof safe
    fix  $x\ n$  assume  $x \in \{a..S\ n\}$ 
    with 1(1,2)[of n] show  $x \in \{a..<b\}$  by auto
  next
    fix  $x$  assume  $x \in \{a..<b\}$ 
    with order_tendstoD[OF  $\langle S \longrightarrow b \rangle$ , of x] show  $x \in (\bigcup n. \{a..S\ n\})$ 
    by (force simp: eventually_at_top_linorder dest: less_imp_le)
  qed
  have ( $\lambda n. \text{set\_lebesgue\_integral } M \{a..S\ n\} f$ )  $\longrightarrow$   $\text{set\_lebesgue\_integral } M$ 
  ( $\bigcup n. \{a..S\ n\}$ )  $f$ 
    by (rule set_integral_cont_up) (insert assms 1, auto simp: eq_incseq_def dec-
seq_def less_imp_le)
  with eq show ?case by simp
  qed

```

```

proposition tendsto_set_lebesgue_integral_at_top:
  fixes  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
  assumes sets:  $\bigwedge b. b \geq a \implies \{a..b\} \in \text{sets } M$ 
    and int:  $\text{set\_integrable } M \{a..\} f$ 
  shows  $((\lambda b. \text{set\_lebesgue\_integral } M \{a..b\} f) \longrightarrow \text{set\_lebesgue\_integral } M$ 
 $\{a..\} f)$  at_top
proof (rule tendsto_at_topI_sequentially)
  fix  $X :: \text{nat} \Rightarrow \text{real}$  assume filterlim X at_top sequentially
  show  $(\lambda n. \text{set\_lebesgue\_integral } M \{a..X\ n\} f) \longrightarrow \text{set\_lebesgue\_integral } M$ 
 $\{a..\} f$ 
    unfolding set_lebesgue_integral_def

```

```

proof (rule integral_dominated_convergence)
  show integrable M (λx. indicat_real {a..} x *R norm (f x))
    using integrable_norm[OF int[unfolded set_integrable_def]] by simp
  show AE x in M. (λn. indicator {a..X n} x *R f x) → indicat_real {a..}
x *R f x
  proof
    fix x
    from ⟨filterlim X at_top sequentially⟩
    have eventually (λn. x ≤ X n) sequentially
      unfolding filterlim_at_top_ge[where c=x] by auto
    then show (λn. indicator {a..X n} x *R f x) → indicat_real {a..} x *R
f x
      by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
    qed
  fix n show AE x in M. norm (indicator {a..X n} x *R f x) ≤
    indicator {a..} x *R norm (f x)
    by (auto split: split_indicator)
  next
  from int show (λx. indicat_real {a..} x *R f x) ∈ borel_measurable M
    by (simp add: set_integrable_def)
  next
  fix n :: nat
  from sets have {a..X n} ∈ sets M by (cases X n ≥ a) auto
  with int have set_integrable M {a..X n} f
    by (rule set_integrable_subset) auto
  thus (λx. indicat_real {a..X n} x *R f x) ∈ borel_measurable M
    by (simp add: set_integrable_def)
  qed
qed

proposition tendsto_set_lebesgue_integral_at_bot:
  fixes f :: real ⇒ 'a::{banach, second_countable_topology}
  assumes sets: ∧a. a ≤ b ⇒ {a..b} ∈ sets M
  and int: set_integrable M {..b} f
  shows ((λa. set_lebesgue_integral M {a..b} f) → set_lebesgue_integral M
{..b} f) at_bot
proof (rule tendsto_at_botI_sequentially)
  fix X :: nat ⇒ real assume filterlim X at_bot sequentially
  show (λn. set_lebesgue_integral M {X n..b} f) → set_lebesgue_integral M
{..b} f
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
  show integrable M (λx. indicat_real {..b} x *R norm (f x))
    using integrable_norm[OF int[unfolded set_integrable_def]] by simp
  show AE x in M. (λn. indicator {X n..b} x *R f x) → indicat_real {..b}
x *R f x
  proof
    fix x

```

```

from ⟨filterlim X at bot sequentially⟩
have eventually (λn. x ≥ X n) sequentially
  unfolding filterlim_at_bot_le[where c=x] by auto
then show (λn. indicator {X n..b} x *R f x) ⟶ indicat_real {..b} x *R
f x
  by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
qed
fix n show AE x in M. norm (indicator {X n..b} x *R f x) ≤
  indicator {..b} x *R norm (f x)
  by (auto split: split_indicator)
next
from int show (λx. indicat_real {..b} x *R f x) ∈ borel_measurable M
  by (simp add: set_integrable_def)
next
fix n :: nat
from sets have {X n..b} ∈ sets M by (cases X n ≤ b) auto
with int have set_integrable M {X n..b} f
  by (rule set_integrable_subset) auto
thus (λx. indicat_real {X n..b} x *R f x) ∈ borel_measurable M
  by (simp add: set_integrable_def)
qed
qed

```

theorem integral_Markov_inequality':

```

fixes u :: 'a ⇒ real
assumes [measurable]: set_integrable M A u and A ∈ sets M
assumes AE x in M. x ∈ A ⟶ u x ≥ 0 and 0 < (c::real)
shows emeasure M {x∈A. u x ≥ c} ≤ (1/c::real) * (∫ x∈A. u x ∂M)
proof -
have (λx. u x * indicator A x) ∈ borel_measurable M
  using assms by (auto simp: set_integrable_def mult_ac)
hence (λx. ennreal (u x * indicator A x)) ∈ borel_measurable M
  by measurable
also have (λx. ennreal (u x * indicator A x)) = (λx. ennreal (u x) * indicator
A x)
  by (intro ext) (auto simp: indicator_def)
finally have meas: ... ∈ borel_measurable M .
from assms(3) have AE: AE x in M. 0 ≤ u x * indicator A x
  by eventually_elim (auto simp: indicator_def)
have nonneg: set_lebesgue_integral M A u ≥ 0
  unfolding set_lebesgue_integral_def
  by (intro Bochner_Integration.integral_nonneg_AE eventually_mono[OF AE])
(auto simp: mult_ac)

have A: A ⊆ space M
  using ⟨A ∈ sets M⟩ by (simp add: sets.sets_into_space)

```

```

have { $x \in A. u x \geq c$ } = { $x \in A. \text{ennreal}(1/c) * u x \geq 1$ }
  using < $c > 0$ >  $A$  by (auto simp: ennreal_mult'[symmetric])
then have emeasure  $M$  { $x \in A. u x \geq c$ } = emeasure  $M$  ({ $x \in A. \text{ennreal}(1/c)$ 
*  $u x \geq 1$ })
  by simp
also have ...  $\leq \text{ennreal}(1/c) * (\int^+ x. \text{ennreal}(u x) * \text{indicator } A x \partial M)$ 
  by (intro nn_integral_Markov_inequality meas assms)
also have ( $\int^+ x. \text{ennreal}(u x) * \text{indicator } A x \partial M$ ) = ennreal (set_lebesgue_integral
 $M A u$ )
  unfolding set_lebesgue_integral_def nn_integral_set_ennreal using assms
 $AE$ 
  by (subst nn_integral_eq_integral) (simp_all add: mult_ac set_integrable_def)
finally show ?thesis
  using < $c > 0$ > nonneg by (subst ennreal_mult) auto
qed

```

```

theorem integral_Markov_inequality'_measure:
  assumes [measurable]: set_integrable  $M A u$  and  $A \in \text{sets } M$ 
  and  $AE x \text{ in } M. x \in A \longrightarrow 0 \leq u x \ 0 < (c::\text{real})$ 
  shows measure  $M$  { $x \in A. u x \geq c$ }  $\leq (\int x \in A. u x \partial M) / c$ 
proof -
  have nonneg: set_lebesgue_integral  $M A u \geq 0$ 
  unfolding set_lebesgue_integral_def
  by (intro Bochner_Integration.integral_nonneg_AE eventually_mono[OF assms(3)])
  (auto simp: mult_ac)
  have le: emeasure  $M$  { $x \in A. u x \geq c$ }  $\leq \text{ennreal} ((1/c) * (\int x \in A. u x \partial M))$ 
  by (rule integral_Markov_inequality') (use assms in auto)
  also have ...  $< \text{top}$ 
  by auto
  finally have ennreal (measure  $M$  { $x \in A. u x \geq c$ }) = emeasure  $M$  { $x \in A. u x \geq$ 
 $c$ }
  by (intro emeasure_eq_ennreal_measure [symmetric]) auto
  also note le
  finally show ?thesis using nonneg
  by (subst (asm) ennreal_le_iff)
  (auto intro!: divide_nonneg_pos Bochner_Integration.integral_nonneg_AE
  assms)
qed

```

```

theorem (in finite_measure) Chernoff_ineq_ge:
  assumes  $s: s > 0$ 
  assumes integrable: set_integrable  $M A (\lambda x. \text{exp } (s * f x))$  and  $A \in \text{sets } M$ 
  shows measure  $M$  { $x \in A. f x \geq a$ }  $\leq \text{exp } (-s * a) * (\int x \in A. \text{exp } (s * f x) \partial M)$ 
proof -
  have { $x \in A. f x \geq a$ } = { $x \in A. \text{exp } (s * f x) \geq \text{exp } (s * a)$ }
  using  $s$  by auto
  also have measure  $M$  ...  $\leq \text{set\_lebesgue\_integral } M A (\lambda x. \text{exp } (s * f x)) / \text{exp}$ 
( $s * a$ )
  by (intro integral_Markov_inequality'_measure assms) auto

```

```

finally show ?thesis
  by (simp add: exp_minus_field_simps)
qed

```

```

theorem (in finite_measure) Chernoff_ineq_le:
  assumes s: s > 0
  assumes integrable: set_integrable M A ( $\lambda x. \exp(-s * f x)$ ) and A  $\in$  sets M
  shows measure M {x  $\in$  A. f x  $\leq$  a}  $\leq$  exp(s * a) * ( $\int$  x  $\in$  A. exp(-s * f x)  $\partial$ M)
proof -
  have {x  $\in$  A. f x  $\leq$  a} = {x  $\in$  A. exp(-s * f x)  $\geq$  exp(-s * a)}
    using s by auto
  also have measure M ...  $\leq$  set_lebesgue_integral M A ( $\lambda x. \exp(-s * f x)$ ) /
    exp(-s * a)
    by (intro integral_Markov_inequality'_measure assms) auto
  finally show ?thesis
    by (simp add: exp_minus_field_simps)
qed

```

9.7 Integrable Simple Functions

This section is from the Martingales AFP entry, by Ata Keskin, TU München

We restate some basic results concerning Bochner-integrable functions.

```

lemma integrable_implies_simple_function_sequence:
  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes integrable M f
  obtains s where  $\bigwedge i. \text{simple\_function } M (s i)$ 
    and  $\bigwedge i. \text{emeasure } M \{y \in \text{space } M. s i y \neq 0\} \neq \infty$ 
    and  $\bigwedge x. x \in \text{space } M \implies (\lambda i. s i x) \longrightarrow f x$ 
    and  $\bigwedge x i. x \in \text{space } M \implies \text{norm } (s i x) \leq 2 * \text{norm } (f x)$ 
proof -
  have f: f  $\in$  borel_measurable M ( $\int^+ x. \text{norm } (f x) \partial M$ )  $< \infty$  using assms
unfolding integrable_iff_bounded by auto
  obtain s where s:  $\bigwedge i. \text{simple\_function } M (s i) \bigwedge x. x \in \text{space } M \implies (\lambda i. s i x) \longrightarrow f x \bigwedge i x. x \in \text{space } M \implies \text{norm } (s i x) \leq 2 * \text{norm } (f x)$  using
    borel_measurable_implies_sequence_metric[OF f(1)] unfolding norm_conv_dist
by metis
  {
    fix i
    have ( $\int^+ x. \text{norm } (s i x) \partial M$ )  $\leq$  ( $\int^+ x. \text{ennreal } (2 * \text{norm } (f x)) \partial M$ ) using
      s by (intro nn_integral_mono, auto)
    also have ...  $< \infty$  using f by (simp add: nn_integral_cmult ennreal_mult_less_top
      ennreal_mult)
    finally have sbi: Bochner_Integration.simple_bochner_integrable M (s i) using
      s by (intro simple_bochner_integrableI_bounded) auto
    hence emeasure M {y  $\in$  space M. s i y  $\neq$  0}  $\neq \infty$  by (auto intro: integrableI_simple_bochner_integrable simple_bochner_integrable.cases)
  }

```

thus ?thesis using that s by blast
qed

Simple functions can be represented by sums of indicator functions.

lemma *simple_function_indicator_representation_banach*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach}\}$
assumes *simple_function* $M f x \in \text{space } M$
shows $f x = (\sum y \in f^{-1} \text{space } M. \text{indicator } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$
(is ?l = ?r)
proof –
have $?r = (\sum y \in f^{-1} \text{space } M. (\text{if } y = f x \text{ then indicator } (f^{-1} \{y\} \cap \text{space } M) x *_R y \text{ else } 0))$
by (*auto intro!: sum.cong*)
also have $\dots = \text{indicator } (f^{-1} \{f x\} \cap \text{space } M) x *_R f x$ **using** *assms* **by**
(*auto dest: simple_functionD*)
also have $\dots = f x$ **using** *assms* **by** (*auto simp: indicator_def*)
finally show ?thesis **by** *auto*
qed

lemma *simple_function_indicator_representation_AE*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach}\}$
assumes *simple_function* $M f$
shows *AE* x *in* $M. f x = (\sum y \in f^{-1} \text{space } M. \text{indicator } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$
by (*metis (mono_tags, lifting) AE_I2 simple_function_indicator_representation_banach assms*)

lemmas *simple_function_scaleR*[*intro*] = *simple_function_compose2*[**where** $h = (*_R)$]

lemmas *integrable_simple_function* = *simple_bochner_integrable.intros*[*THEN has_bochner_integral. THEN integrable.intros*]

Induction rule for simple integrable functions.

lemma *integrable_simple_function_induct*[*consumes 2, case_names cong indicator add, induct set: simple_function*]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach}\}$
assumes $f: \text{simple_function } M f \text{ emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty$
assumes *cong*: $\bigwedge f g. \text{simple_function } M f \Longrightarrow \text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty$
 $\Longrightarrow \text{simple_function } M g \Longrightarrow \text{emeasure } M \{y \in \text{space } M. g y \neq 0\} \neq \infty$
 $\Longrightarrow (\bigwedge x. x \in \text{space } M \Longrightarrow f x = g x) \Longrightarrow P f \Longrightarrow P g$
assumes *indicator*: $\bigwedge A y. A \in \text{sets } M \Longrightarrow \text{emeasure } M A < \infty \Longrightarrow P (\lambda x. \text{indicator } A x *_R y)$
assumes *add*: $\bigwedge f g. \text{simple_function } M f \Longrightarrow \text{emeasure } M \{y \in \text{space } M. f y \neq 0\} \neq \infty \Longrightarrow$
 $\text{simple_function } M g \Longrightarrow \text{emeasure } M \{y \in \text{space } M. g y \neq 0\} \neq \infty \Longrightarrow$
 $(\bigwedge z. z \in \text{space } M \Longrightarrow \text{norm } (f z + g z) = \text{norm } (f z) + \text{norm } (g z)) \Longrightarrow$

$P f \implies P g \implies P (\lambda x. f x + g x)$

shows $P f$

proof –

let $?f = \lambda x. (\sum y \in f^{-1} \text{ space } M. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$

have $f_ae_eq: f x = ?f x$ **if** $x \in \text{space } M$ **for** x **using** *simple_function_indicator_representation_banach* [OF $f(1)$ *that*].

moreover have $\text{emeasure } M \{y \in \text{space } M. ?f y \neq 0\} \neq \infty$ **by** (*metis* (*no_types*, *lifting*) *Collect_cong* *calculation* $f(2)$)

moreover have $P (\lambda x. \sum y \in S. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$
 $\text{simple_function } M (\lambda x. \sum y \in S. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$

$\text{emeasure } M \{y \in \text{space } M. (\sum x \in S. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x) \neq 0\} \neq \infty$

if $S \subseteq f^{-1} \text{ space } M$ **for** S **using** *simple_functionD(1)* [OF *assms(1)*],
THEN rev_finite_subset, OF that **that**

proof (*induction* *rule: finite_induct*)

case *empty*

{

case *1*

then show $?case$ **using** *indicator* [of {}] **by** *force*

next

case *2*

then show $?case$ **by** *force*

next

case *3*

then show $?case$ **by** *force*

}

next

case (*insert* $x F$)

have $(f^{-1} \{x\} \cap \text{space } M) \subseteq \{y \in \text{space } M. f y \neq 0\}$ **if** $x \neq 0$ **using** *that* **by** *blast*

moreover have $\{y \in \text{space } M. f y \neq 0\} = \text{space } M - (f^{-1} \{0\} \cap \text{space } M)$
by *blast*

moreover have $\text{space } M - (f^{-1} \{0\} \cap \text{space } M) \in \text{sets } M$ **using** *simple_functionD(2)* [OF $f(1)$] **by** *blast*

ultimately have $\text{fin}_0: \text{emeasure } M (f^{-1} \{x\} \cap \text{space } M) < \infty$ **if** $x \neq 0$ **using** *that* **by** (*metis* *emeasure_mono* $f(2)$ *infinity_ennreal_def* *top.not_eq_extremum* *top_unique*)

hence $\text{fin}_1: \text{emeasure } M \{y \in \text{space } M. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x \neq 0\} \neq \infty$ **if** $x \neq 0$ **by** (*metis* (*mono_tags*, *lifting*) *emeasure_mono* $f(1)$ *indicator_simps(2)* *linorder_not_less* *mem_Collect_eq* *scaleR_eq_0_iff* *simple_functionD(2)* *subsetI* *that*)

have $*$: $(\sum y \in \text{insert } x F. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y) =$
 $(\sum y \in F. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y) + \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) x *_R x$ **for** x **by** (*metis* (*no_types*, *lifting*) *Diff_empty* *Diff_insert0* *add commute* *insert.hyps(1)* *insert.hyps(2)* *sum.insert_remove*)

have $**$: $\{y \in \text{space } M. (\sum x \in \text{insert } x F. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x) \neq 0\} \subseteq \{y \in \text{space } M. (\sum x \in F. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x)$

```

≠ 0} ∪ {y ∈ space M. indicat_real (f -' {x} ∩ space M) y *R x ≠ 0} unfolding
* by fastforce
{
  case 1
  hence x: x ∈ f ' space M and F: F ⊆ f ' space M by auto
  show ?case
  proof (cases x = 0)
    case True
    then show ?thesis unfolding * using insert(3)[OF F] by simp
  next
  case False
  have norm_argument: norm ((∑ y∈F. indicat_real (f -' {y} ∩ space M) z
*_R y) + indicat_real (f -' {x} ∩ space M) z *_R x) = norm (∑ y∈F. indicat_real
(f -' {y} ∩ space M) z *_R y) + norm (indicat_real (f -' {x} ∩ space M) z *_R
x) if z: z ∈ space M for z
  proof (cases f z = x)
    case True
    have indicat_real (f -' {y} ∩ space M) z *_R y = 0 if y ∈ F for y
    using True insert(2) z that 1 unfolding indicator_def by force
    hence (∑ y∈F. indicat_real (f -' {y} ∩ space M) z *_R y) = 0 by (meson
sum.neutral)
    then show ?thesis by force
  next
  case False
  then show ?thesis by force
  qed
  show ?thesis
    using False simple_functionD(2)[OF f(1)] insert(3,5)[OF F] sim-
ple_function_scaleR fin_0 fin_1
    by (subst *, subst add, subst simple_function_sum) (blast intro: norm_argument
indicator)+
  qed
next
  case 2
  hence x: x ∈ f ' space M and F: F ⊆ f ' space M by auto
  show ?case
  proof (cases x = 0)
    case True
    then show ?thesis unfolding * using insert(4)[OF F] by simp
  next
  case False
  then show ?thesis unfolding * using insert(4)[OF F] simple_functionD(2)[OF
f(1)] by fast
  qed
next
  case 3
  hence x: x ∈ f ' space M and F: F ⊆ f ' space M by auto
  show ?case
  proof (cases x = 0)

```

```

    case True
  then show ?thesis unfolding * using insert(5)[OF F] by simp
next
case False
  have emeasure M {y ∈ space M. (∑ x∈insert x F. indicat_real (f - ' {x}
  ∩ space M) y *R x) ≠ 0} ≤ emeasure M ({y ∈ space M. (∑ x∈F. indicat_real (f
  - ' {x} ∩ space M) y *R x) ≠ 0} ∪ {y ∈ space M. indicat_real (f - ' {x} ∩ space
  M) y *R x ≠ 0})
  using ** simple_functionD(2)[OF insert(4)[OF F]] simple_functionD(2)[OF
  f(1)] by (intro emeasure_mono, force+)
  also have ... ≤ emeasure M {y ∈ space M. (∑ x∈F. indicat_real (f - '
  {x} ∩ space M) y *R x) ≠ 0} + emeasure M {y ∈ space M. indicat_real (f - '
  {x} ∩ space M) y *R x ≠ 0}
  using simple_functionD(2)[OF insert(4)[OF F]] simple_functionD(2)[OF
  f(1)] by (intro emeasure_subadditive, force+)
  also have ... < ∞ using insert(5)[OF F] fn_1[OF False] by (simp add:
  less_top)
  finally show ?thesis by simp
qed
}
qed
moreover have simple_function M (λx. ∑ y∈f ' space M. indicat_real (f - '
{y} ∩ space M) x *R y) using calculation by blast
moreover have P (λx. ∑ y∈f ' space M. indicat_real (f - ' {y} ∩ space M) x
*R y) using calculation by blast
ultimately show ?thesis by (intro cong[OF _ _ f(1,2)], blast, presburger+)
qed

```

Induction rule for non-negative simple integrable functions

lemma *integrable_simple_function_induct_nn*[consumes 3, case_names cong indicator add, induct set: simple_function]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach, linorder_topology, ordered_real_vector}\}$

assumes f : *simple_function* M f *emeasure* M $\{y \in \text{space } M. f\ y \neq 0\} \neq \infty \wedge x. x \in \text{space } M \longrightarrow f\ x \geq 0$

assumes *cong*: $\bigwedge f\ g. \text{simple_function } M\ f \Longrightarrow \text{emeasure } M\ \{y \in \text{space } M. f\ y \neq 0\} \neq \infty \Longrightarrow (\bigwedge x. x \in \text{space } M \Longrightarrow f\ x \geq 0) \Longrightarrow \text{simple_function } M\ g \Longrightarrow \text{emeasure } M\ \{y \in \text{space } M. g\ y \neq 0\} \neq \infty \Longrightarrow (\bigwedge x. x \in \text{space } M \Longrightarrow g\ x \geq 0) \Longrightarrow (\bigwedge x. x \in \text{space } M \Longrightarrow f\ x = g\ x) \Longrightarrow P\ f \Longrightarrow P\ g$

assumes *indicator*: $\bigwedge A\ y. y \geq 0 \Longrightarrow A \in \text{sets } M \Longrightarrow \text{emeasure } M\ A < \infty \Longrightarrow P\ (\lambda x. \text{indicator } A\ x *_{R}\ y)$

assumes *add*: $\bigwedge f\ g. (\bigwedge x. x \in \text{space } M \Longrightarrow f\ x \geq 0) \Longrightarrow \text{simple_function } M\ f \Longrightarrow \text{emeasure } M\ \{y \in \text{space } M. f\ y \neq 0\} \neq \infty \Longrightarrow$

$(\bigwedge x. x \in \text{space } M \Longrightarrow g\ x \geq 0) \Longrightarrow \text{simple_function } M\ g \Longrightarrow \text{emeasure } M\ \{y \in \text{space } M. g\ y \neq 0\} \neq \infty \Longrightarrow$

$(\bigwedge z. z \in \text{space } M \Longrightarrow \text{norm } (f\ z + g\ z) = \text{norm } (f\ z) + \text{norm } (g\ z)) \Longrightarrow$

$P\ f \Longrightarrow P\ g \Longrightarrow P\ (\lambda x. f\ x + g\ x)$

shows $P\ f$

proof –

let $?f = \lambda x. (\sum y \in f^{-1} \text{space } M. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$
have $f_ae_eq: f x = ?f x$ **if** $x \in \text{space } M$ **for** x **using** *simple_function_indicator_representation_banach*
f(1) that .

moreover have $\text{emeasure } M \{y \in \text{space } M. ?f y \neq 0\} \neq \infty$ **by** (*metis (no_types, lifting) Collect_cong* calculation *f(2)*)

moreover have $P (\lambda x. \sum y \in S. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$
 $\text{simple_function } M (\lambda x. \sum y \in S. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M)$
 $x *_R y)$

$\text{emeasure } M \{y \in \text{space } M. (\sum x \in S. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x) \neq 0\} \neq \infty$
 $\wedge x. x \in \text{space } M \implies 0 \leq (\sum y \in S. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y)$

if $S \subseteq f^{-1} \text{space } M$ **for** S **using** *simple_functionD(1)[OF assms(1), THEN rev_finite_subset, OF that]* **that**

proof (*induction rule: finite_subset_induct'*)

case *empty*

{

case *1*

then show $?case$ **using** *indicator[of 0 {}]* **by** *force*

next

case *2*

then show $?case$ **by** *force*

next

case *3*

then show $?case$ **by** *force*

next

case *4*

then show $?case$ **by** *force*

}

next

case (*insert x F*)

have $(f^{-1} \{x\} \cap \text{space } M) \subseteq \{y \in \text{space } M. f y \neq 0\}$ **if** $x \neq 0$

using *that* **by** *blast*

moreover have $\{y \in \text{space } M. f y \neq 0\} = \text{space } M - (f^{-1} \{0\} \cap \text{space } M)$

by *blast*

moreover have $\text{space } M - (f^{-1} \{0\} \cap \text{space } M) \in \text{sets } M$

using *simple_functionD(2)[OF f(1)]* **by** *blast*

ultimately have $\text{fin_0}: \text{emeasure } M (f^{-1} \{x\} \cap \text{space } M) < \infty$ **if** $x \neq 0$

using *that* **by** (*metis emeasure_mono f(2) infinity_ennreal_def top.not_eq_extremum top_unique*)

hence $\text{fin_1}: \text{emeasure } M \{y \in \text{space } M. \text{indicat_real } (f^{-1} \{x\} \cap \text{space } M) y *_R x \neq 0\} \neq \infty$ **if** $x \neq 0$

by (*metis (mono_tags, lifting) emeasure_mono f(1) indicator_simps(2) linorder_not_less mem_Collect_eq scaleR_eq_0_iff simple_functionD(2) subsetI that*)

have *nonneg_x*: $x \geq 0$ **using** *insert f* **by** *blast*

have $*$: $(\sum y \in \text{insert } x F. \text{indicat_real } (f^{-1} \{y\} \cap \text{space } M) x *_R y) =$

```

( $\sum y \in F. \text{indicat\_real } (f - \{y\} \cap \text{space } M) \ x \ *_{\mathbb{R}} \ y$ ) +  $\text{indicat\_real } (f - \{x\} \cap \text{space } M) \ x \ *_{\mathbb{R}} \ x$  for  $x \in F$  by (metis (no_types, lifting) add commute insert.hyps(1) insert.hyps(4) sum.insert)
have **:  $\{y \in \text{space } M. (\sum x \in \text{insert } x \ F. \text{indicat\_real } (f - \{x\} \cap \text{space } M) \ y \ *_{\mathbb{R}} \ x) \neq 0\} \subseteq \{y \in \text{space } M. (\sum x \in F. \text{indicat\_real } (f - \{x\} \cap \text{space } M) \ y \ *_{\mathbb{R}} \ x) \neq 0\} \cup \{y \in \text{space } M. \text{indicat\_real } (f - \{x\} \cap \text{space } M) \ y \ *_{\mathbb{R}} \ x \neq 0\}$  unfolding
* by fastforce
{
  case 1
  show ?case
  proof (cases  $x = 0$ )
    case True
    then show ?thesis unfolding * using insert by simp
  next
  case False
  have norm_argument:  $\text{norm } ((\sum y \in F. \text{indicat\_real } (f - \{y\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ y) + \text{indicat\_real } (f - \{x\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ x)$ 
    =  $\text{norm } (\sum y \in F. \text{indicat\_real } (f - \{y\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ y) + \text{norm } (\text{indicat\_real } (f - \{x\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ x)$ 
    if  $z: z \in \text{space } M$  for  $z$ 
    proof (cases  $f \ z = x$ )
      case True
      have  $\text{indicat\_real } (f - \{y\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ y = 0$  if  $y \in F$  for  $y$ 
        using True insert  $z$  that 1 unfolding indicator_def by force
      hence  $(\sum y \in F. \text{indicat\_real } (f - \{y\} \cap \text{space } M) \ z \ *_{\mathbb{R}} \ y) = 0$ 
        by (meson sum.neutral)
      thus ?thesis by force
    qed (force)
    show ?thesis using False fin_0 fin_1 f norm_argument
      by (subst *, subst add, presburger add: insert, intro insert, intro insert,
fastforce simp add: indicator_def intro!: insert(2) f(3), auto intro!: indicator insert nonneg_x)
    qed
  next
  case 2
  show ?case
  proof (cases  $x = 0$ )
    case True
    then show ?thesis unfolding * using insert by simp
  next
  case False
  then show ?thesis unfolding * using insert simple_functionD(2)[OF f(1)]
by fast
  qed
next
  case 3
  show ?case
  proof (cases  $x = 0$ )
    case True

```

```

then show ?thesis unfolding * using insert by simp
next
  case False
    have emeasure M {y ∈ space M. (∑ x∈insert x F. indicat_real (f -‘ {x}
    ∩ space M) y *R x) ≠ 0}
      ≤ emeasure M ({y ∈ space M. (∑ x∈F. indicat_real (f -‘ {x} ∩ space
    M) y *R x) ≠ 0} ∪ {y ∈ space M. indicat_real (f -‘ {x} ∩ space M) y *R x ≠
    0})
    using ** simple_functionD(2)[OF insert(6)] simple_functionD(2)[OF
    f(1)] insert.IH(2) by (intro emeasure_mono, blast, simp)
    also have ... ≤ emeasure M {y ∈ space M. (∑ x∈F. indicat_real (f -‘
    {x} ∩ space M) y *R x) ≠ 0} + emeasure M {y ∈ space M. indicat_real (f -‘
    {x} ∩ space M) y *R x ≠ 0}
    using simple_functionD(2)[OF insert(6)] simple_functionD(2)[OF f(1)]
by (intro emeasure_subadditive, force+)
    also have ... < ∞ using insert(7) fin_1[OF False] by (simp add: less_top)
    finally show ?thesis by simp
  qed
next
  case (4 ξ)
    thus ?case using insert nonneg_x f(3) by (auto simp add: scaleR_nonneg_nonneg
    intro: sum_nonneg)
  }
qed
moreover have simple_function M (λx. ∑ y∈f -‘ space M. indicat_real (f -‘
  {y} ∩ space M) x *R y)
  using calculation by blast
  moreover have P (λx. ∑ y∈f -‘ space M. indicat_real (f -‘ {y} ∩ space M) x
  *R y)
  using calculation by blast
  moreover have ∧x. x ∈ space M ⇒ 0 ≤ f x using f(3) by simp
  ultimately show ?thesis
  by (smt (verit) Collect_cong f(1) local.cong)
qed

lemma finite_nn_integral_imp_ae_finite:
  fixes f :: 'a ⇒ ennreal
  assumes f ∈ borel_measurable M (∫+x. f x ∂M) < ∞
  shows AE x in M. f x < ∞
proof (rule ccontr, goal_cases)
  case 1
  let ?A = space M ∩ {x. f x = ∞}
  have *: emeasure M ?A > 0 using 1 assms(1)
  by (metis (mono_tags, lifting) assms(2) eventually_mono infinity_ennreal_def
  nn_integral_noteq_infinite top.not_eq_extremum)
  have (∫+x. f x * indicator ?A x ∂M) = (∫+x. ∞ * indicator ?A x ∂M)
  by (metis (mono_tags, lifting) indicator_inter_arith indicator_simps(2) mem_Collect_eq
  mult_eq_0_iff)
  also have ... = ∞ * emeasure M ?A

```

```

    using assms(1) by (intro nn_integral_cmult_indicator, simp)
  also have ... =  $\infty$ 
    using * by fastforce
  finally have  $(\int^+ x. f x * \text{indicator } ?A x \partial M) = \infty$  .
  moreover have  $(\int^+ x. f x * \text{indicator } ?A x \partial M) \leq (\int^+ x. f x \partial M)$ 
    by (intro nn_integral_mono, simp add: indicator_def)
  ultimately show ?case using assms(2) by simp
qed

```

Convergence in L1-Norm implies existence of a subsequence which converges almost everywhere. This lemma is easier to use than the existing one in *HOL-Analysis.Bochner_Integration*

lemma *cauchy_L1_AE_cauchy_subseq*:

```

  fixes s :: nat  $\Rightarrow$  'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes [measurable]:  $\bigwedge n. \text{integrable } M (s n)$ 
    and  $\bigwedge e. e > 0 \implies \exists N. \forall i \geq N. \forall j \geq N. \text{LINT } x|M. \text{norm } (s i x - s j x) < e$ 
  obtains r where strict_mono r AE x in M. Cauchy ( $\lambda i. s (r i) x$ )

```

proof –

```

  have  $\exists r. \forall n. (\forall i \geq r n. \forall j \geq r n. \text{LINT } x|M. \text{norm } (s i x - s j x) < (1 / 2) ^ n) \wedge (r (\text{Suc } n) > r n)$ 

```

```

  proof (intro dependent_nat_choice, goal_cases)

```

```

    case 1

```

```

    then show ?case using assms(2) by presburger

```

```

  next

```

```

    case (2 x n)

```

```

    obtain N where *:  $\text{LINT } x|M. \text{norm } (s i x - s j x) < (1 / 2) ^ \text{Suc } n$  if  $i \geq N$  for  $j \geq N$ 

```

```

    using assms(2)[of  $(1 / 2) ^ \text{Suc } n$ ] by auto

```

```

    {
      fix i j assume  $i \geq \max N (\text{Suc } x) \ j \geq \max N (\text{Suc } x)$ 

```

```

      hence  $\text{integral}^L M (\lambda x. \text{norm } (s i x - s j x)) < (1 / 2) ^ \text{Suc } n$  using * by
fastforce

```

```

    }

```

```

    then show ?case by fastforce

```

```

  qed

```

```

  then obtain r where strict_mono: strict_mono r and  $\forall i \geq r n. \forall j \geq r n. \text{LINT } x|M. \text{norm } (s i x - s j x) < (1 / 2) ^ n$  for n

```

```

    using strict_mono_Suc_iff by blast

```

```

    hence r_is:  $\text{LINT } x|M. \text{norm } (s (r (\text{Suc } n)) x - s (r n) x) < (1 / 2) ^ n$  for n by (simp add: strict_mono_leD)

```

```

    define g where  $g = (\lambda n x. (\sum_{i \leq n. \text{ennreal } (\text{norm } (s (r (\text{Suc } i)) x - s (r i) x))))$ 

```

```

    define g' where  $g' = (\lambda x. \sum i. \text{ennreal } (\text{norm } (s (r (\text{Suc } i)) x - s (r i) x)))$ 

```

```

    have integrable_g:  $(\int^+ x. g n x \partial M) < 2$  for n

```

```

  proof –

```

```

    have  $(\int^+ x. g n x \partial M) = (\int^+ x. (\sum_{i \leq n. \text{ennreal } (\text{norm } (s (r (\text{Suc } i)) x - s (r i) x)))) \partial M$ 

```

```

    using g_def by simp
    also have ... = ( $\sum_{i \leq n}. (\int^+ x. \text{ennreal} (\text{norm} (s (r (Suc i)) x - s (r i) x))$ 
 $\partial M$ ))
    by (intro nn_integral_sum, simp)
    also have ... = ( $\sum_{i \leq n}. \text{LINT } x[M. \text{norm} (s (r (Suc i)) x - s (r i) x)$ 
    unfolding dist_norm using assms(1) by (subst nn_integral_eq_integral[OF
    integrable_norm], auto)
    also have ... < ennreal ( $\sum_{i \leq n}. (1 / 2) ^ i$ )
    by (intro ennreal_lessI[OF sum_pos sum_strict_mono[OF finite_atMost _
    r_is]], auto)
    also have ...  $\leq$  ennreal 2
    unfolding sum_gp0[of 1 / 2 n] by (intro ennreal_leI, auto)
    finally show ( $\int^+ x. g \ n \ x \ \partial M$ ) < 2 by simp
  qed

  have integrable_g': ( $\int^+ x. g' \ x \ \partial M$ )  $\leq$  2
  proof -
    have incseq ( $\lambda n. g \ n \ x$ ) for x by (intro incseq_SucI, auto simp add: g_def
    ennreal_leI)
    hence convergent ( $\lambda n. g \ n \ x$ ) for x
    unfolding convergent_def using LIMSEQ_incseq_SUP by fast
    hence ( $\lambda n. g \ n \ x$ )  $\longrightarrow$  g' x for x
    unfolding g_def g'_def by (intro summable_iff_convergent'[THEN iffD2,
    THEN summable_LIMSEQ'], blast)
    hence ( $\int^+ x. g' \ x \ \partial M$ ) = ( $\int^+ x. \text{liminf} (\lambda n. g \ n \ x) \ \partial M$ ) by (metis lim_imp_Liminf
    trivial_limit_sequentially)
    also have ...  $\leq$  liminf ( $\lambda n. \int^+ x. g \ n \ x \ \partial M$ ) by (intro nn_integral_liminf,
    simp add: g_def)
    also have ...  $\leq$  liminf ( $\lambda n. 2$ ) using integrable_g by (intro Liminf_mono)
    (simp add: order_le_less)
    also have ... = 2
    using sequentially_bot_tendsto_iff_Liminf_eq_Limsup by blast
    finally show ?thesis .
  qed
  hence AE x in M. g' x <  $\infty$ 
  by (intro finite_nn_integral_imp_ae_finite) (auto simp add: order_le_less_trans
  g'_def)
  moreover have summable ( $\lambda i. \text{norm} (s (r (Suc i)) x - s (r i) x)$ ) if g' x  $\neq$   $\infty$ 
  for x
    using that unfolding g'_def by (intro summable_suminf_not_top) fastforce+

  ultimately have ae_summable: AE x in M. summable ( $\lambda i. s (r (Suc i)) x - s$ 
  ( $r \ i$ ) x)
    using summable_norm_cancel unfolding dist_norm by force

  {
    fix x assume summable ( $\lambda i. s (r (Suc i)) x - s (r i) x$ )
    hence ( $\lambda n. \sum_{i < n}. s (r (Suc i)) x - s (r i) x$ )  $\longrightarrow$  ( $\sum i. s (r (Suc i)) x -$ 
    s ( $r \ i$ ) x)
  }

```



```

    using summable_LIMSEQ by blast
    moreover have  $(\lambda n. (\sum i < n. s (r (Suc i)) x - s (r i) x)) = (\lambda n. s (r n) x - s (r 0) x)$ 
    using sum_lessThan_telescope by fastforce
    ultimately have  $(\lambda n. s (r n) x - s (r 0) x) \longrightarrow (\sum i. s (r (Suc i)) x - s (r i) x)$  by argo
    hence  $(\lambda n. s (r n) x - s (r 0) x + s (r 0) x) \longrightarrow (\sum i. s (r (Suc i)) x - s (r i) x) + s (r 0) x$ 
    by (intro isCont_tendsto_compose[of  $\lambda z. z + s (r 0) x$ ], auto)
    hence Cauchy  $(\lambda n. s (r n) x)$  by (simp add: LIMSEQ_imp_Cauchy)
  }
  hence AE x in M. Cauchy  $(\lambda i. s (r i) x)$  using ae_summable by fast
  thus ?thesis by (rule that[OF strict_mono(1)])
qed

```

9.7.1 Totally Ordered Banach Spaces

```

lemma integrable_max[simp, intro]:
  fixes f :: 'a  $\Rightarrow$  'b :: {second_countable_topology, banach, linorder_topology}
  assumes fg[measurable]: integrable M f integrable M g
  shows integrable M  $(\lambda x. \max (f x) (g x))$ 
proof (rule Bochner_Integration.integrable_bound)
  {
    fix x y :: 'b
    have norm (max x y)  $\leq \max (norm x) (norm y)$  by linarith
    also have ...  $\leq norm x + norm y$  by simp
    finally have norm (max x y)  $\leq norm (norm x + norm y)$  by simp
  }
  thus AE x in M. norm (max (f x) (g x))  $\leq norm (norm (f x) + norm (g x))$  by
  simp
qed (auto intro: Bochner_Integration.integrable_add[OF integrable_norm[OF fg(1)]
  integrable_norm[OF fg(2)]])

```

```

lemma integrable_min[simp, intro]:
  fixes f :: 'a  $\Rightarrow$  'b :: {second_countable_topology, banach, linorder_topology}
  assumes [measurable]: integrable M f integrable M g
  shows integrable M  $(\lambda x. \min (f x) (g x))$ 
proof -
  have norm (min (f x) (g x))  $\leq norm (f x) \vee norm (min (f x) (g x)) \leq norm (g x)$ 
  for x by linarith
  thus ?thesis
    by (intro integrable_bound[OF integrable_max[OF integrable_norm(1,1), OF
  assms]], auto)
qed

```

Restatement of *integral_nonneg_AE* for functions taking values in a Banach space.

```

lemma integral_nonneg_AE_banach:
  fixes f :: 'a  $\Rightarrow$  'b :: {second_countable_topology, banach, linorder_topology, or-

```

```

dered_real_vector}
  assumes [measurable]: f ∈ borel_measurable M and nonneg: AE x in M. 0 ≤ f
  x
  shows 0 ≤ integralL M f
proof cases
  assume integrable: integrable M f
  hence max: (λx. max 0 (f x)) ∈ borel_measurable M ∧ x. 0 ≤ max 0 (f x)
integrable M (λx. max 0 (f x)) by auto
  hence 0 ≤ integralL M (λx. max 0 (f x))
proof -
  obtain s where *: ∧i. simple_function M (s i)
    ∧i. emeasure M {y ∈ space M. s i y ≠ 0} ≠ ∞
    ∧x. x ∈ space M ⇒ (λi. s i x) → f x
    ∧x i. x ∈ space M ⇒ norm (s i x) ≤ 2 * norm (f x)
  using integrable_implies_simple_function_sequence[OF integrable] by blast
  have simple: ∧i. simple_function M (λx. max 0 (s i x))
  using * by fast
  have ∧i. {y ∈ space M. max 0 (s i y) ≠ 0} ⊆ {y ∈ space M. s i y ≠ 0}
  unfolding max_def by force
  moreover have ∧i. {y ∈ space M. s i y ≠ 0} ∈ sets M
  using * by measurable
  ultimately have ∧i. emeasure M {y ∈ space M. max 0 (s i y) ≠ 0} ≤
emeasure M {y ∈ space M. s i y ≠ 0}
  by (rule emeasure_mono)
  hence **: ∧i. emeasure M {y ∈ space M. max 0 (s i y) ≠ 0} ≠ ∞
  using *(2) by (auto intro: order.strict_trans1 simp add: top.not_eq_extremum)
  have ∧x. x ∈ space M ⇒ (λi. max 0 (s i x)) → max 0 (f x)
  using *(3) tendsto_max by blast
  moreover have ∧x i. x ∈ space M ⇒ norm (max 0 (s i x)) ≤ norm (2 *R
f x)
  using *(4) unfolding max_def by auto
  ultimately have tendsto: (λi. integralL M (λx. max 0 (s i x))) → integralL
M (λx. max 0 (f x))
  using borel_measurable_simple_function simple integrable by (intro inte-
gral_dominated_convergence[OF max(1) _ integrable_norm[OF integrable_scaleR_right],
of _ 2 f], blast+)
  {
    fix h :: 'a ⇒ 'b :: {second_countable_topology, banach, linorder_topology,
ordered_real_vector}
    assume simple_function M h emeasure M {y ∈ space M. h y ≠ 0} ≠ ∞ ∧x.
x ∈ space M → h x ≥ 0
    hence *: integralL M h ≥ 0
    proof (induct rule: integrable_simple_function_induct_nn)
    case (cong f g)
    then show ?case using Bochner_Integration.integral_cong by force
    next
    case (indicator A y)
    hence A ≠ {} ⇒ y ≥ 0 using sets.sets_into_space by fastforce
    then show ?case using indicator by (cases A = {}, auto simp add:

```

```

scaleR_nonneg_nonneg)
  next
  case (add f g)
  then show ?case by (simp add: integrable_simple_function)
  qed
}
thus ?thesis using LIMSEQ_le_const[OF tendsto, of 0] ** simple by fastforce
qed
also have ... = integralL M f using nonneg by (auto intro: integral_cong_AE)
finally show ?thesis .
qed (simp add: not_integrable_integral_eq)

```

lemma *integral_mono_AE_banach*:

```

fixes f g :: 'a ⇒ 'b :: {second_countable_topology, banach, linorder_topology,
ordered_real_vector}
assumes integrable M f integrable M g AE x in M. f x ≤ g x
shows integralL M f ≤ integralL M g
using integral_nonneg_AE_banach[of λx. g x - f x] assms Bochner_Integration.integral_diff[OF
assms(1,2)] by force

```

lemma *integral_mono_banach*:

```

fixes f g :: 'a ⇒ 'b :: {second_countable_topology, banach, linorder_topology,
ordered_real_vector}
assumes integrable M f integrable M g ∧ x. x ∈ space M ⇒ f x ≤ g x
shows integralL M f ≤ integralL M g
using integral_mono_AE_banach assms by blast

```

9.7.2 Auxiliary Lemmas for Set Integrals

lemma *nn_set_integral_eq_set_integral*:

```

assumes [measurable]: integrable M f
and AE x ∈ A in M. 0 ≤ f x A ∈ sets M
shows (∫+ x ∈ A. f x ∂M) = (∫ x ∈ A. f x ∂M)

```

proof –

```

have (∫+ x. indicator A x *R f x ∂M) = (∫ x ∈ A. f x ∂M)
  unfolding set_lebesgue_integral_def
  using assms(2) by (intro nn_integral_eq_integral[of _ λx. indicat_real A x
*_R f x], blast intro: assms integrable_mult_indicator, fastforce)
moreover have (∫+ x. indicator A x *R f x ∂M) = (∫+ x ∈ A. f x ∂M)
  by (metis ennreal_0 indicator_simps(1) indicator_simps(2) mult.commute
mult_1 mult_zero_left real_scaleR_def)
ultimately show ?thesis by argo
qed

```

lemma *set_integral_restrict_space*:

```

fixes f :: 'a ⇒ 'b :: {banach, second_countable_topology}
assumes Ω ∩ space M ∈ sets M
shows set_lebesgue_integral (restrict_space M Ω) A f = set_lebesgue_integral
M A (λx. indicator Ω x *R f x)

```

unfolding *set_lebesgue_integral_def*
by (*subst integral_restrict_space*, *auto intro!*: *integrable_mult_indicator assms*
simp: *mult.commute*)

lemma *set_integral_const*:
fixes $c :: 'b :: \{\text{banach, second_countable_topology}\}$
assumes $A \in \text{sets } M \text{ emeasure } M A \neq \infty$
shows $\text{set_lebesgue_integral } M A (\lambda _ . c) = \text{measure } M A *_R c$
unfolding *set_lebesgue_integral_def*
using *assms* **by** (*metis has_bochner_integral_indicator has_bochner_integral_integral_eq*
infinity_ennreal_def less_top)

lemma *set_integral_mono_banach*:
fixes $f g :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach, linorder_topology,}$
*ordered_real_vector}\}
assumes $\text{set_integrable } M A f \text{ set_integrable } M A g$
 $\bigwedge x. x \in A \implies f x \leq g x$
shows $(\text{LINT } x:A|M. f x) \leq (\text{LINT } x:A|M. g x)$
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def*
by (*auto intro: integral_mono_banach split: split_indicator*)*

lemma *set_integral_mono_AE_banach*:
fixes $f g :: 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach, linorder_topology,}$
*ordered_real_vector}\}
assumes $\text{set_integrable } M A f \text{ set_integrable } M A g \text{ AE } x \in A \text{ in } M. f x \leq g x$
shows $\text{set_lebesgue_integral } M A f \leq \text{set_lebesgue_integral } M A g$ **using** *assms*
unfolding *set_lebesgue_integral_def* **by** (*auto simp add: set_integrable_def in-*
*trol: integral_mono_AE_banach[of M \lambda x. indicator A x *_R f x \lambda x. indicator A x*
**_R g x], simp add: indicator_def*)*

9.7.3 Integrability and Measurability of the Diameter

context

fixes $s :: \text{nat} \Rightarrow 'a \Rightarrow 'b :: \{\text{second_countable_topology, banach}\}$ **and** M
assumes *bounded*: $\bigwedge x. x \in \text{space } M \implies \text{bounded } (\lambda i. s i x)$

begin

lemma *borel_measurable_diameter*:

assumes [*measurable*]: $\bigwedge i. (s i) \in \text{borel_measurable } M$
shows $(\lambda x. \text{diameter } \{s i x \mid i. n \leq i\}) \in \text{borel_measurable } M$

proof –

have $\{s i x \mid i. N \leq i\} \neq \{\}$ **for** $x N$ **by** *blast*

hence *diameter_SUP*: $\text{diameter } \{s i x \mid i. N \leq i\} = (\text{SUP } (i, j) \in \{N.. \} \times \{N.. \}.$
 $\text{dist } (s i x) (s j x))$ **for** $x N$ **unfolding** *diameter_def* **by** (*auto intro!*: *arg_cong[of*
 $_ _ \text{Sup}]$)

have *case_prod dist* ‘ $(\{s i x \mid i. n \leq i\} \times \{s i x \mid i. n \leq i\}) = ((\lambda(i, j). \text{dist } (s i$
 $x) (s j x)) \text{ ‘ } (\{n.. \} \times \{n.. \}))$ **for** x **by** *fast*

hence *: $(\lambda x. \text{diameter } \{s i x \mid i. n \leq i\}) = (\lambda x. \text{Sup } ((\lambda(i, j). \text{dist } (s i x) (s j$

```

x)) ‘({n..} × {n..})) using diameter_SUP by (simp add: case_prod_beta’)

  have bounded ((λ(i, j). dist (s i x) (s j x)) ‘({n..} × {n..})) if x ∈ space M for
x by (rule bounded_imp_dist_bounded[OF bounded, OF that])
  hence bdd: bdd_above ((λ(i, j). dist (s i x) (s j x)) ‘({n..} × {n..})) if x ∈ space
M for x using that bounded_imp_bdd_above by presburger
  have fst p ∈ borel_measurable M snd p ∈ borel_measurable M if p ∈ s ‘ {n..} ×
s ‘ {n..} for p using that by fastforce+
  hence (λx. fst p x - snd p x) ∈ borel_measurable M if p ∈ s ‘ {n..} × s ‘ {n..}
for p using that borel_measurable_diff by simp
  hence (λx. case p of (f, g) ⇒ dist (f x) (g x)) ∈ borel_measurable M if p ∈ s ‘
{n..} × s ‘ {n..} for p unfolding dist_norm using that by measurable
  moreover have countable (s ‘ {n..} × s ‘ {n..}) by (intro countable_SIGMA
countable_image, auto)
  ultimately show ?thesis unfolding * by (auto intro!: borel_measurable_cSUP
bdd)
qed

```

lemma *integrable_bound_diameter*:

```

  fixes f :: 'a ⇒ real
  assumes integrable M f
    and [measurable]: ∧i. (s i) ∈ borel_measurable M
    and ∧x i. x ∈ space M ⇒ norm (s i x) ≤ f x
    shows integrable M (λx. diameter {s i x | i. n ≤ i})
  proof -
    have {s i x | i. N ≤ i} ≠ {} for x N by blast
    hence diameter_SUP: diameter {s i x | i. N ≤ i} = (SUP (i, j) ∈ {N..} × {N..}.
dist (s i x) (s j x)) for x N unfolding diameter_def by (auto intro!: arg_cong[of
_ _ Sup])
    {
      fix x assume x: x ∈ space M
      let ?S = (λ(i, j). dist (s i x) (s j x)) ‘({n..} × {n..})
      have case_prod_dist ‘({s i x | i. n ≤ i} × {s i x | i. n ≤ i}) = (λ(i, j). dist (s
i x) (s j x)) ‘({n..} × {n..}) by fast
      hence *: diameter {s i x | i. n ≤ i} = Sup ?S using diameter_SUP by (simp
add: case_prod_beta’)

      have bounded ?S by (rule bounded_imp_dist_bounded[OF bounded[OF x]])
      hence Sup_S_nonneg: 0 ≤ Sup ?S by (auto intro!: cSup_upper2 x bounded_imp_bdd_above)

      have dist (s i x) (s j x) ≤ 2 * f x for i j by (intro dist_triangle2[THEN
order_trans, of _ 0]) (metis norm_conv_dist assms(3) x add_mono mult_2)
      hence ∀c ∈ ?S. c ≤ 2 * f x by force
      hence Sup ?S ≤ 2 * f x by (intro cSup_least, auto)
      hence norm (Sup ?S) ≤ 2 * norm (f x) using Sup_S_nonneg by auto
      also have ... = norm (2 *R f x) by simp
      finally have norm (diameter {s i x | i. n ≤ i}) ≤ norm (2 *R f x) unfolding
* .
    }
  }

```

hence $\forall x \text{ in } M. \text{norm}(\text{diameter } \{s \ i \ x \mid i. n \leq i\}) \leq \text{norm}(2 *_{\mathbb{R}} f x)$ **by** *blast*
thus $\text{integrable } M(\lambda x. \text{diameter } \{s \ i \ x \mid i. n \leq i\})$ **using** *borel_measurable_diameter*
by (*intro Bochner_Integration.integrable_bound[OF assms(1)][THEN integrable_scaleR_right[of 2]]*, *measurable*)
qed
end

9.7.4 Averaging Theorem

We aim to lift results from the real case to arbitrary Banach spaces. Our fundamental tool in this regard will be the averaging theorem. The proof of this theorem is due to Serge Lang (Real and Functional Analysis) [5]. The theorem allows us to make statements about a function's value almost everywhere, depending on the value its integral takes on various sets of the measure space.

Before we introduce and prove the averaging theorem, we will first show the following lemma which is crucial for our proof. While not stated exactly in this manner, our proof makes use of the characterization of second-countable topological spaces given in the book General Topology by Ryszard Engelking (Theorem 4.1.15) [4]. (Engelking's book *General Topology*)

lemma *balls_countable_basis*:

obtains $D :: 'a :: \{\text{metric_space}, \text{second_countable_topology}\}$ *set*
where *topological_basis* (*case_prod ball ' (D × (ℚ ∩ {0<..}))*)
and *countable* D
and $D \neq \{\}$

proof –

obtain $D :: 'a$ *set* **where** *dense_subset: countable* $D \ D \neq \{\}$ [*open* U ; $U \neq \{\}$]
 $\implies \exists y \in D. y \in U$ **for** U **using** *countable_dense_exists* **by** *blast*

have *topological_basis* (*case_prod ball ' (D × (ℚ ∩ {0<..}))*)

proof (*intro topological_basis_iff[THEN iffD2]*, *fast*, *clarify*)

fix U **and** $x :: 'a$ **assume** *asm: open* $U \ x \in U$

obtain e **where** $e > 0$ *ball* $x \ e \subseteq U$ **using** *asm openE* **by** *blast*

obtain y **where** $y \in D \ y \in \text{ball } x \ (e / 3)$ **using** *dense_subset(3)[OF open_ball, of x e / 3] centre_in_ball[THEN iffD2, OF divide_pos_pos[OF e(1), of 3]]* **by** *force*

obtain r **where** $r \in \mathbb{Q} \cap \{e/3 < .. < e/2\}$ **unfolding** *Rats_def* **using**
of_rat_dense[OF divide_strict_left_mono[OF _ e(1)], of 2 3] **by** *auto*

have $*$: $x \in \text{ball } y \ r$ **using** $r \ y$ **by** (*simp add: dist_commute*)

hence $\text{ball } y \ r \subseteq U$ **using** r **by** (*intro order_trans[OF _ e(2)]*, *simp*, *metric*)

moreover **have** $\text{ball } y \ r \in (\text{case_prod ball ' (D × (ℚ ∩ {0<..}))})$ **using** $y(1)$
 r **by** *force*

ultimately show $\exists B' \in (\text{case_prod ball ' (D × (ℚ ∩ {0<..}))}) . x \in B' \wedge B' \subseteq U$
using $*$ **by** *meson*

qed

thus *?thesis* **using** *that dense_subset* **by** *blast*

qed

context *sigma_finite_measure*
begin

To show statements concerning σ -finite measure spaces, one usually shows the statement for finite measure spaces and uses a limiting argument to show it for the σ -finite case. The following induction scheme formalizes this.

lemma *sigma_finite_measure_induct*[*case_names finite_measure, consumes 0*]:

assumes $\bigwedge(N :: 'a \text{ measure}) \Omega. \text{finite_measure } N$
 $\implies N = \text{restrict_space } M \ \Omega$
 $\implies \Omega \in \text{sets } M$
 $\implies \text{emeasure } N \ \Omega \neq \infty$
 $\implies \text{emeasure } N \ \Omega \neq 0$
 $\implies \text{almost_everywhere } N \ Q$

and [*measurable*]: *Measurable.pred* $M \ Q$

shows *almost_everywhere* $M \ Q$

proof –

have *: *almost_everywhere* $N \ Q$ **if** *finite_measure* $N \ N = \text{restrict_space } M \ \Omega \ \Omega \in \text{sets } M \ \text{emeasure } N \ \Omega \neq \infty$ **for** $N \ \Omega$ **using** *that* **by** (*cases* *emeasure* $N \ \Omega = 0$, *auto* *intro*: *emeasure_0_AE* *assms*(1))

obtain $A :: \text{nat} \Rightarrow 'a \text{ set}$ **where** $A: \text{range } A \subseteq \text{sets } M \ (\bigcup i. A \ i) = \text{space } M$ **and** *emeasure_finite*: *emeasure* $M \ (A \ i) \neq \infty$ **for** i **using** *sigma_finite* **by** *metis*

note $A(1)$ [*measurable*]

have *space_restr*: *space* (*restrict_space* $M \ (A \ i)$) = $A \ i$ **for** i **unfolding** *space_restrict_space* **by** *simp*

{
fix i

have *: $\{x \in A \ i \cap \text{space } M. \ Q \ x\} = \{x \in \text{space } M. \ Q \ x\} \cap (A \ i)$ **by** *fast*

have *Measurable.pred* (*restrict_space* $M \ (A \ i)$) Q **using** A **by** (*intro* *measurableI*, *auto* *simp* *add*: *space_restr* *intro*!: *sets_restrict_space_iff*[*THEN iffD2*], *measurable*, *auto*)

}

note *this*[*measurable*]

{
fix i

have *finite_measure* (*restrict_space* $M \ (A \ i)$) **using** *emeasure_finite* **by** (*intro* *finite_measureI*, *subst* *space_restr*, *subst* *emeasure_restrict_space*, *auto*)

hence *emeasure* (*restrict_space* $M \ (A \ i)$) $\{x \in A \ i. \neg Q \ x\} = 0$ **using** *emeasure_finite* **by** (*intro* *AE_iff_measurable*[*THEN iffD1*, *OF _ _ **], *measurable*, *subst* *space_restr*[*symmetric*], *intro* *sets.top*, *auto* *simp* *add*: *emeasure_restrict_space*)

hence *emeasure* $M \ \{x \in A \ i. \neg Q \ x\} = 0$ **by** (*subst* *emeasure_restrict_space*[*symmetric*], *auto*)

}

hence *emeasure* $M \ (\bigcup i. \{x \in A \ i. \neg Q \ x\}) = 0$ **by** (*intro* *emeasure_UN_eq_0*, *auto*)

moreover **have** $(\bigcup i. \{x \in A \ i. \neg Q \ x\}) = \{x \in \text{space } M. \neg Q \ x\}$ **using** A **by** *auto*

ultimately show *?thesis* **by** (*intro* *AE_iff_measurable*[*THEN iffD2*], *auto*)

qed

Real Functional Analysis - Lang

The Averaging Theorem allows us to make statements concerning how a function behaves almost everywhere, depending on its behaviour on average.

lemma *averaging_theorem*:

fixes $f :: _ \Rightarrow 'b :: \{\text{second_countable_topology, banach}\}$

assumes [*measurable*]: *integrable M f*

and *closed*: *closed S*

and $\bigwedge A. A \in \text{sets } M \implies \text{measure } M A > 0 \implies (1 / \text{measure } M A) *_{\mathbb{R}} \text{set_lebesgue_integral } M A f \in S$

shows $\text{AE } x \text{ in } M. f x \in S$

proof (*induct rule: sigma_finite_measure_induct*)

case (*finite_measure N Ω*)

interpret *finite_measure N* **by** (*rule finite_measure*)

have *integrable[measurable]: integrable N f* **using** *assms finite_measure* **by** (*auto simp: integrable_restrict_space integrable_mult_indicator*)

have *average: (1 / Sigma_Algebra.measure N A) *_{\mathbb{R}} \text{set_lebesgue_integral } N A f \in S* **if** $A \in \text{sets } N$ **measure** $N A > 0$ **for** A

proof –

have $*$: $A \in \text{sets } M$ **using** *that* **by** (*simp add: sets_restrict_space_iff finite_measure*)

have $A = A \cap \Omega$ **by** (*metis finite_measure(2,3) inf.orderE sets.sets_into_space space_restrict_space that(1)*)

hence $\text{set_lebesgue_integral } N A f = \text{set_lebesgue_integral } M A f$ **unfolding** *finite_measure* **by** (*subst set_integral_restrict_space, auto simp add: finite_measure set_lebesgue_integral_def indicator_inter_arith[symmetric]*)

moreover **have** $\text{measure } N A = \text{measure } M A$ **using** *that* **by** (*auto intro!: measure_restrict_space simp add: finite_measure sets_restrict_space_iff*)

ultimately show *?thesis* **using** *that * assms(3)* **by** *presburger*

qed

obtain $D :: 'b$ **set** **where** *balls_basis: topological_basis (case_prod ball ' (D \times ($\mathbb{Q} \cap \{0 < ..\}$)))* **and** *countable_D: countable D* **using** *balls_countable_basis* **by** *blast*

have *countable_balls: countable (case_prod ball ' (D \times ($\mathbb{Q} \cap \{0 < ..\}$)))* **using** *countable_rat countable_D* **by** *blast*

obtain B **where** *B_balls: B \subseteq case_prod ball ' (D \times ($\mathbb{Q} \cap \{0 < ..\}$))) $\cup B = -S$* **using** *topological_basis[THEN iffD1, OF balls_basis] open_Cmpl[OF assms(2)]* **by** *meson*

hence *countable_B: countable B* **using** *countable_balls countable_subset* **by** *fast*

define b **where** $b = \text{from_nat_into } (B \cup \{\{\}\})$

have $B \cup \{\{\}\} \neq \{\}$ **by** *simp*

have *range_b: range b = B \cup $\{\{\}\}$* **using** *countable_B* **by** (*auto simp add: b_def intro!: range_from_nat_into*)


```

have open_b: open (b i) for i unfolding b_def using B_balls open_ball from_nat_into[of
B ∪ {{{}} i] by force
have Union_range_b: ⋃(range b) = -S using B_balls range_b by simp

{
  fix v r assume ball_in_Compl: ball v r ⊆ -S
  define A where A = f - ` ball v r ∩ space N
  have dist_less: dist (f x) v < r if x ∈ A for x using that unfolding A_def
vimage_def by (simp add: dist_commute)
  hence AE_less: AE x ∈ A in N. norm (f x - v) < r by (auto simp add:
dist_norm)
  have *: A ∈ sets N unfolding A_def by simp
  have emeasure N A = 0
  proof -
    {
      assume asm: emeasure N A > 0
      hence measure_pos: measure N A > 0 unfolding emeasure_eq_measure
by simp
      hence (1 / measure N A) *R set_lebesgue_integral N A f - v
= (1 / measure N A) *R set_lebesgue_integral N A (λx. f x - v)
      using integrable_integrable_const *
      by (subst set_integral_diff(2), auto simp add: set_integrable_def set_integral_const[OF
*] algebra_simps intro!: integrable_mult_indicator)
      moreover have norm (∫ x∈A. (f x - v)∂N) ≤ (∫ x∈A. norm (f x - v)∂N)

      using * by (auto intro!: integral_norm_bound[of N λx. indicator A x
*_R (f x - v), THEN order_trans] integrable_mult_indicator integrable simp add:
set_lebesgue_integral_def)
      ultimately have norm ((1 / measure N A) *R set_lebesgue_integral N A
f - v)
≤ set_lebesgue_integral N A (λx. norm (f x - v)) / measure N
A using asm by (auto intro: divide_right_mono)
      also have ... < set_lebesgue_integral N A (λx. r) / measure N A
      unfolding set_lebesgue_integral_def
      using asm * integrable_integrable_const AE_less measure_pos
      by (intro divide_strict_right_mono integral_less_AE[of _ _ A] inte-
grable_mult_indicator)
      (fastforce simp add: dist_less dist_norm indicator_def)+
      also have ... = r using * measure_pos by (simp add: set_integral_const)
      finally have dist ((1 / measure N A) *R set_lebesgue_integral N A f) v <
r by (subst dist_norm)
      hence False using average[OF * measure_pos] by (metis ComplD dist_commute
in_mono mono mem_ball ball_in_Compl)
    }
  }
  thus ?thesis by fastforce
qed
}
note * = this
{

```

```

    fix b' assume b' ∈ B
    hence ball_subset_Compl: b' ⊆ -S and ball_radius_pos: ∃ v ∈ D. ∃ r > 0. b'
= ball v r using B_balls by (blast, fast)
  }
  note ** = this
  hence emeasure N (f -' b i ∩ space N) = 0 for i by (cases b i = {}, simp)
(metis UnE singletonD * range_b[THEN eq_refl, THEN range_subsetD])
  hence emeasure N (⋃ i. f -' b i ∩ space N) = 0 using open_b by (intro
emeasure_UN_eq_0) fastforce+
  moreover have (⋃ i. f -' b i ∩ space N) = f -' (⋃ (range b)) ∩ space N by
blast
  ultimately have emeasure N (f -' (-S) ∩ space N) = 0 using Union_range_b
by argo
  hence AE x in N. f x ∉ -S using open_Compl[OF assms(2)] by (intro AE_iff_measurable[THEN
iffD2], auto)
  thus ?case by force
qed (simp add: pred_sets2[OF borel_closed] assms(2))

```

```

lemma density_zero:
  fixes f::'a ⇒ 'b::{second_countable_topology, banach}
  assumes integrable M f
    and density_0: ⋀ A. A ∈ sets M ⇒ set_lebesgue_integral M A f = 0
  shows AE x in M. f x = 0
  using averaging_theorem[OF assms(1), of {0}] assms(2)
  by (simp add: scaleR_nonneg_nonneg)

```

The following lemma shows that densities are unique in Banach spaces.

```

lemma density_unique_banach:
  fixes f f': 'a ⇒ 'b::{second_countable_topology, banach}
  assumes integrable M f integrable M f'
    and density_eq: ⋀ A. A ∈ sets M ⇒ set_lebesgue_integral M A f =
set_lebesgue_integral M A f'
  shows AE x in M. f x = f' x
proof -
  {
    fix A assume asm: A ∈ sets M
    hence LINT x|M. indicat_real A x *_R (f x - f' x) = 0 using density_eq
assms(1,2) by (simp add: set_lebesgue_integral_def algebra_simps Bochner_Integration.integral_diff[O
integrable_mult_indicator(1,1)])
  }
  thus ?thesis using density_zero[OF Bochner_Integration.integrable_diff[OF assms(1,2)]]
by (simp add: set_lebesgue_integral_def)
qed

```

```

lemma density_nonneg:
  fixes f::_ ⇒ 'b::{second_countable_topology, banach, linorder_topology, ordered_real_vector}
  assumes integrable M f
    and ⋀ A. A ∈ sets M ⇒ set_lebesgue_integral M A f ≥ 0
  shows AE x in M. f x ≥ 0

```

```

using averaging_theorem[OF assms(1), of {0..}, OF closed_atLeast] assms(2)
by (simp add: scaleR_nonneg_nonneg)

```

corollary *integral_nonneg_eq_0_iff_AE_banach*:

```

fixes f :: 'a  $\Rightarrow$  'b :: {second_countable_topology, banach, linorder_topology, or-
  dered_real_vector}

```

```

assumes f[measurable]: integrable M f and nonneg: AE x in M. 0  $\leq$  f x

```

```

shows integralL M f = 0  $\longleftrightarrow$  (AE x in M. f x = 0)

```

proof

```

assume *: integralL M f = 0

```

```

{

```

```

  fix A assume asm: A  $\in$  sets M

```

```

  have 0  $\leq$  integralL M ( $\lambda$ x. indicator A x *R f x) using nonneg by (subst inte-
    gral_zero[of M, symmetric], intro integral_mono_AE_banach integrable_mult_indicator
    asm f integrable_zero, auto simp add: indicator_def)

```

```

  moreover have ...  $\leq$  integralL M f using nonneg by (intro integral_mono_AE_banach
    integrable_mult_indicator asm f, auto simp add: indicator_def)

```

```

  ultimately have set_lebesgue_integral M A f = 0 unfolding set_lebesgue_integral_def
using * by force

```

```

}

```

```

thus AE x in M. f x = 0 by (intro density_zero f, blast)

```

```

qed (auto simp add: integral_eq_zero_AE)

```

corollary *integral_eq_mono_AE_eq_AE*:

```

fixes f g :: 'a  $\Rightarrow$  'b :: {second_countable_topology, banach, linorder_topology,
  ordered_real_vector}

```

```

assumes integrable M f integrable M g integralL M f = integralL M g AE x in
  M. f x  $\leq$  g x

```

```

shows AE x in M. f x = g x

```

proof –

```

define h where h = ( $\lambda$ x. g x - f x)

```

```

have AE x in M. h x = 0 unfolding h_def using assms by (subst inte-
  gral_nonneg_eq_0_iff_AE_banach[symmetric]) auto

```

```

then show ?thesis unfolding h_def by auto

```

```

qed

```

end

end

9.8 Homeomorphism Theorems

```

theory Homeomorphism

```

```

imports Homotopy

```

```

begin

```

lemma *homeomorphic_spheres'*:

```

fixes a :: 'a::euclidean_space and b :: 'b::euclidean_space

```

```

assumes 0 <  $\delta$  and dimeq: DIM('a) = DIM('b)

```

shows (sphere a δ) homeomorphic (sphere b δ)
proof –
 obtain $f :: 'a \Rightarrow 'b$ and g where linear f linear g
 and $fg: \bigwedge x. \text{norm}(f\ x) = \text{norm}\ x \bigwedge y. \text{norm}(g\ y) = \text{norm}\ y \bigwedge x. g(f\ x) = x$
 $\bigwedge y. f(g\ y) = y$
 by (blast intro: isomorphisms_UNIV_UNIV [OF dimeq])
 then have continuous_on UNIV f continuous_on UNIV g
 using linear_continuous_on linear_linear by blast+
 then show ?thesis
 unfolding homeomorphic_minimal
 apply(rule_tac $x = \lambda x. b + f(x - a)$ in exI)
 apply(rule_tac $x = \lambda x. a + g(x - b)$ in exI)
 using assms
 apply (force intro: continuous_intros
 continuous_on_compose2 [of _ f] continuous_on_compose2 [of _
 g] simp: dist_commute dist_norm fg)
 done
qed

lemma homeomorphic_spheres_gen:
 fixes $a :: 'a::\text{euclidean_space}$ and $b :: 'b::\text{euclidean_space}$
 assumes $0 < r\ 0 < s$ $\text{DIM}('a::\text{euclidean_space}) = \text{DIM}('b::\text{euclidean_space})$
 shows (sphere a r homeomorphic sphere b s)
 using assms homeomorphic_trans [OF homeomorphic_spheres homeomorphic_spheres']
 by auto

9.8.1 Homeomorphism of all convex compact sets with nonempty interior

proposition
 fixes $S :: 'a::\text{euclidean_space}$ set
 assumes compact S and $0: 0 \in \text{rel_interior}\ S$
 and $\text{star}: \bigwedge x. x \in S \implies \text{open_segment}\ 0\ x \subseteq \text{rel_interior}\ S$
 shows starlike_compact_projective1_0:
 $S - \text{rel_interior}\ S$ homeomorphic sphere $0\ 1 \cap \text{affine hull}\ S$
 (is ?SMINUS homeomorphic ?SPHER)
 and starlike_compact_projective2_0:
 S homeomorphic cball $0\ 1 \cap \text{affine hull}\ S$
 (is S homeomorphic ?CBALL)
proof –
 have starI: $(u *_R\ x) \in \text{rel_interior}\ S$ if $x \in S\ 0 \leq u\ u < 1$ for $x\ u$
proof (cases $x=0 \vee u=0$)
 case True with 0 show ?thesis by force
 next
 case False with that show ?thesis
 by (auto simp: in_segment intro: star [THEN subsetD])
qed
 have $0 \in S$ using assms rel_interior_subset by auto
 define proj where $\text{proj} \equiv \lambda x::'a. x /_R \text{norm}\ x$

```

have eqI:  $x = y$  if  $\text{proj } x = \text{proj } y$   $\text{norm } x = \text{norm } y$  for  $x y$ 
  using that by (force simp: proj_def)
then have iff_eq:  $\bigwedge x y. (\text{proj } x = \text{proj } y \wedge \text{norm } x = \text{norm } y) \longleftrightarrow x = y$ 
  by blast
have projI:  $x \in \text{affine hull } S \implies \text{proj } x \in \text{affine hull } S$  for  $x$ 
  by (metis  $\langle 0 \in S \rangle$  affine_hull_span_0 hull_inc span_mul proj_def)
have nproj1 [simp]:  $x \neq 0 \implies \text{norm}(\text{proj } x) = 1$  for  $x$ 
  by (simp add: proj_def)
have proj0_iff [simp]:  $\text{proj } x = 0 \longleftrightarrow x = 0$  for  $x$ 
  by (simp add: proj_def)
have cont_proj: continuous_on (UNIV - {0}) proj
  unfolding proj_def by (rule continuous_intros | force)+
have proj_spherI:  $\bigwedge x. \llbracket x \in \text{affine hull } S; x \neq 0 \rrbracket \implies \text{proj } x \in ?SPHER$ 
  by (simp add: projI)
have bounded_S_closed_S
  using  $\langle \text{compact } S \rangle$  compact_eq_bounded_closed by blast+
have inj_on_proj: inj_on proj (S - rel_interior S)
proof
  fix  $x y$ 
  assume  $x: x \in S - \text{rel\_interior } S$ 
    and  $y: y \in S - \text{rel\_interior } S$  and eq:  $\text{proj } x = \text{proj } y$ 
  then have xynot:  $x \neq 0 \wedge y \neq 0 \wedge x \in S \wedge y \in S \wedge x \notin \text{rel\_interior } S \wedge y \notin \text{rel\_interior } S$ 
  S
    using 0 by auto
  consider  $\text{norm } x = \text{norm } y \mid \text{norm } x < \text{norm } y \mid \text{norm } x > \text{norm } y$  by linarith
  then show  $x = y$ 
  proof cases
    assume  $\text{norm } x = \text{norm } y$ 
      with iff_eq eq show  $x = y$  by blast
    next
      assume *:  $\text{norm } x < \text{norm } y$ 
      have  $x /_R \text{norm } x = \text{norm } x *_R (x /_R \text{norm } x) /_R \text{norm } (\text{norm } x *_R (x /_R \text{norm } x))$ 
        norm x)
      by force
      then have proj ((norm x / norm y) *_R y) = proj x
      by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)
      then have [simp]:  $(\text{norm } x / \text{norm } y) *_R y = x$ 
      by (rule eqI) (simp add:  $\langle y \neq 0 \rangle$ )
      have no:  $0 \leq \text{norm } x / \text{norm } y \wedge \text{norm } x / \text{norm } y < 1$ 
      using * by (auto simp: field_split_simps)
      then show  $x = y$ 
      using starI [OF  $\langle y \in S \rangle$  no] xynot by auto
    next
      assume *:  $\text{norm } x > \text{norm } y$ 
      have  $y /_R \text{norm } y = \text{norm } y *_R (y /_R \text{norm } y) /_R \text{norm } (\text{norm } y *_R (y /_R \text{norm } y))$ 
        norm y)
      by force
      then have proj ((norm y / norm x) *_R x) = proj y
      by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)

```

```

then have [simp]: (norm y / norm x) *R x = y
  by (rule eqI) (simp add: ⟨x ≠ 0⟩)
have no: 0 ≤ norm y / norm x norm y / norm x < 1
  using * by (auto simp: field_split_simps)
then show x = y
  using starI [OF ⟨x ∈ S⟩ no] xynot by auto
qed
qed
have ∃ surf. homeomorphism (S - rel_interior S) ?SPHER proj surf
proof (rule homeomorphism_compact)
  show compact (S - rel_interior S)
    using ⟨compact S⟩ compact_rel_boundary by blast
  show continuous_on (S - rel_interior S) proj
    using 0 by (blast intro: continuous_on_subset [OF cont_proj])
  show proj ' (S - rel_interior S) = ?SPHER
  proof
    show proj ' (S - rel_interior S) ⊆ ?SPHER
      using 0 by (force simp: hull_inc projI intro: nproj1)
    show ?SPHER ⊆ proj ' (S - rel_interior S)
      proof (clarsimp simp: proj_def)
        fix x
        assume x ∈ affine hull S and nox: norm x = 1
        then have x ≠ 0 by auto
        obtain d where 0 < d and dx: (d *R x) ∈ rel_frontier S
          and ri: ∧e. [0 ≤ e; e < d] ⇒ (e *R x) ∈ rel_interior S
          using ray_to_rel_frontier [OF ⟨bounded S⟩ 0] ⟨x ∈ affine hull S⟩ ⟨x ≠
0⟩ by auto
        show x ∈ (λx. x /R norm x) ' (S - rel_interior S)
          proof
            show x = d *R x /R norm (d *R x)
              using ⟨0 < d⟩ by (auto simp: nox)
            show d *R x ∈ S - rel_interior S
              using dx ⟨closed S⟩ by (auto simp: rel_frontier_def)
          qed
        qed
      qed
    qed
  qed (rule inj_on_proj)
then obtain surf where surf: homeomorphism (S - rel_interior S) ?SPHER
proj surf
  by blast
then have cont_surf: continuous_on (proj ' (S - rel_interior S)) surf
  by (auto simp: homeomorphism_def)
have surf_nz: ∧x. x ∈ ?SPHER ⇒ surf x ≠ 0
  by (metis 0 DiffE homeomorphism_def imageI surf)
have cont_nosp: continuous_on (?SPHER) (λx. norm x *R ((surf o proj) x))
proof (intro continuous_intros)
  show continuous_on (sphere 0 1 ∩ affine hull S) proj
    by (rule continuous_on_subset [OF cont_proj], force)
  show continuous_on (proj ' (sphere 0 1 ∩ affine hull S)) surf

```

```

    by (intro continuous_on_subset [OF cont_surf]) (force simp: homeomorphism_image1 [OF surf] dest: proj_spherI)
  qed
  have surfpS:  $\bigwedge x. \llbracket \text{norm } x = 1; x \in \text{affine hull } S \rrbracket \implies \text{surf } (\text{proj } x) \in S$ 
    by (metis (full_types) DiffE  $\langle 0 \in S \rangle$  homeomorphism_def image_eqI norm_zero proj_spherI real_vector.scale_zero_left scaleR_one surf)
  have *:  $\exists y. \text{norm } y = 1 \wedge y \in \text{affine hull } S \wedge x = \text{surf } (\text{proj } y)$ 
    if  $x \in S \wedge x \notin \text{rel\_interior } S$  for  $x$ 
  proof -
    have proj_x  $\in$  ?SPHER
      by (metis (full_types) 0 hull_inc proj_spherI that)
    moreover have surf (proj x) = x
      by (metis Diff_iff homeomorphism_def surf that)
    ultimately show ?thesis
      by (metis  $\langle \bigwedge x. x \in ?SPHER \implies \text{surf } x \neq 0 \rangle$  hull_inc inverse_1 local.proj_def norm_sgn projI scaleR_one sgn_div_norm that(1))
  qed
  have surfp_notin:  $\bigwedge x. \llbracket \text{norm } x = 1; x \in \text{affine hull } S \rrbracket \implies \text{surf } (\text{proj } x) \notin \text{rel\_interior } S$ 
    by (metis (full_types) DiffE one_neq_zero homeomorphism_def image_eqI norm_zero proj_spherI surf)
  have no_sp_im:  $(\lambda x. \text{norm } x *_R \text{surf } (\text{proj } x)) ' (?SPHER) = S - \text{rel\_interior } S$ 
    by (auto simp: surfpS image_def Bex_def surfp_notin *)
  have inj_spher: inj_on  $(\lambda x. \text{norm } x *_R \text{surf } (\text{proj } x))$  ?SPHER
  proof
    fix x y
    assume xy:  $x \in ?SPHER \wedge y \in ?SPHER$ 
    and eq:  $\text{norm } x *_R \text{surf } (\text{proj } x) = \text{norm } y *_R \text{surf } (\text{proj } y)$ 
    then have norm_x = 1 norm_y = 1  $x \in \text{affine hull } S \wedge y \in \text{affine hull } S$ 
      using 0 by auto
    with eq show x = y
      by (simp add: proj_def) (metis surf_xy homeomorphism_def)
  qed
  have co01: compact ?SPHER
    by (simp add: compact_Int_closed)
  show ?SMINUS homeomorphic ?SPHER
    using homeomorphic_def surf by blast
  have proj_scaleR:  $\bigwedge a x. 0 < a \implies \text{proj } (a *_R x) = \text{proj } x$ 
    by (simp add: proj_def)
  have cont_sp0: continuous_on (affine hull S - {0}) (surf o proj)
  proof (rule continuous_on_compose [OF continuous_on_subset [OF cont_proj]])
    show continuous_on (proj ' (affine hull S - {0})) surf
      using homeomorphism_image1 proj_spherI surf by (intro continuous_on_subset [OF cont_surf]) fastforce
  qed auto
  obtain B where B > 0 and B:  $\bigwedge x. x \in S \implies \text{norm } x \leq B$ 
    by (metis compact_imp_bounded  $\langle \text{compact } S \rangle$  bounded_pos_less less_eq_real_def)
  have cont_nosp: continuous (at x within ?CBALL)  $(\lambda x. \text{norm } x *_R \text{surf } (\text{proj } x))$ 

```

```

x))
  if  $\text{norm } x \leq 1 \ x \in \text{affine hull } S$  for  $x$ 
proof (cases  $x=0$ )
  case True
  have ( $\text{norm } \longrightarrow 0$ ) (at 0 within  $\text{cball } 0 \ 1 \cap \text{affine hull } S$ )
  by (simp add: tendsto_norm_zero eventually_at)
  with True show ?thesis
  apply (simp add: continuous_within)
  apply (rule lim_null_scaleR_bounded [where  $B=B$ ])
  using  $B \langle 0 < B \rangle \text{local.proj\_def projI surfpS}$  by (auto simp: eventually_at)
next
  case False
  then have  $\forall_F x$  in at  $x$ . ( $x \in \text{affine hull } S - \{0\} = (x \in \text{affine hull } S)$ )
  by (force simp: False eventually_at)
  moreover
  have continuous (at  $x$  within  $\text{affine hull } S - \{0\}$ ) ( $\lambda x$ . surf (proj  $x$ ))
  using cont_sp0 False that by (auto simp add: continuous_on_eq_continuous_within)
  ultimately have *: continuous (at  $x$  within  $\text{affine hull } S$ ) ( $\lambda x$ . surf (proj  $x$ ))
  by (simp add: continuous_within Lim_transform_within_set continuous_on_eq_continuous_within)
  show ?thesis
  by (intro continuous_within_subset [where  $S = \text{affine hull } S$ , OF Int_lower2]
continuous_intros *)
qed
  have cont_nosp2: continuous_on ?CBALL ( $\lambda x$ . norm  $x$  *R ((surf o proj)  $x$ ))
  by (simp add: continuous_on_eq_continuous_within cont_nosp)
  have norm  $y$  *R surf (proj  $y$ )  $\in S$  if  $y \in \text{cball } 0 \ 1$  and  $y_{\text{aff}}$ :  $y \in \text{affine hull } S$ 
for  $y$ 
proof (cases  $y=0$ )
  case True then show ?thesis
  by (simp add:  $\langle 0 \in S \rangle$ )
next
  case False
  then have norm  $y$  *R surf (proj  $y$ ) = norm  $y$  *R surf (proj ( $y /_R$  norm  $y$ ))
  by (simp add: proj_def)
  have norm  $y \leq 1$  using that by simp
  have surf (proj ( $y /_R$  norm  $y$ ))  $\in S$ 
  using False local.proj_def nproj1 projI surfpS  $y_{\text{aff}}$  by blast
  then have surf (proj  $y$ )  $\in S$ 
  by (simp add: False proj_def)
  then show norm  $y$  *R surf (proj  $y$ )  $\in S$ 
  by (metis dual_order.antisym le_less_linear norm_ge_zero rel_interior_subset
scaleR_one
starI subset_eq  $\langle \text{norm } y \leq 1 \rangle$ )
qed
moreover have  $x \in (\lambda x$ . norm  $x$  *R surf (proj  $x$ )) ' (?CBALL) if  $x \in S$  for  $x$ 
proof (cases  $x=0$ )
  case True with that hull_inc show ?thesis by fastforce
next
  case False

```



```

then have psp: proj (surf (proj x)) = proj x
  by (metis homeomorphism_def hull_inc proj_spherI surf that)
have nxx: norm x *R proj x = x
  by (simp add: False local.proj_def)
have affineI: (1 / norm (surf (proj x))) *R x ∈ affine hull S
  by (metis ‹0 ∈ S› affine_hull_span_0 hull_inc span_clauses(4) that)
have sproj_nz: surf (proj x) ≠ 0
  by (metis False proj0_iff psp)
then have proj x = proj (proj x)
  by (metis False nxx proj_scaleR zero_less_norm_iff)
moreover have scaleproj: ∧ a r. r *R proj a = (r / norm a) *R a
  by (simp add: divide_inverse local.proj_def)
ultimately have (norm (surf (proj x)) / norm x) *R x ∉ rel_interior S
  by (metis (no_types) sproj_nz divide_self_if hull_inc norm_eq_zero nproj1
projI psp scaleR_one surfp_notin that)
  then have (norm (surf (proj x)) / norm x) ≥ 1
  using starI [OF that] by (meson starI [OF that] le_less_linear norm_ge_zero
zero_le_divide_iff)
  then have nole: norm x ≤ norm (surf (proj x))
  by (simp add: le_divide_eq_1)
let ?inx = x /R norm (surf (proj x))
show ?thesis
proof
  show x = norm ?inx *R surf (proj ?inx)
  by (simp add: field_simps) (metis inverse_eq_divide nxx positive_imp_inverse_positive
proj_scaleR psp scaleproj sproj_nz zero_less_norm_iff)
  qed (auto simp: field_split_simps nole affineI)
qed
ultimately have im_cball: (λx. norm x *R surf (proj x)) ‘ ?CBALL = S
  by blast
have inj_cball: inj_on (λx. norm x *R surf (proj x)) ?CBALL
proof
  fix x y
  assume x ∈ ?CBALL y ∈ ?CBALL
  and eq: norm x *R surf (proj x) = norm y *R surf (proj y)
  then have x: x ∈ affine hull S and y: y ∈ affine hull S
  using 0 by auto
  show x = y
  proof (cases x=0 ∨ y=0)
  case True then show x = y using eq proj_spherI surf_nz x y by force
  next
  case False
  with x y have speq: surf (proj x) = surf (proj y)
  by (metis eq_homeomorphism_apply2 proj_scaleR proj_spherI surf zero_less_norm_iff)
  then have norm x = norm y
  by (metis ‹x ∈ affine hull S› ‹y ∈ affine hull S› eq proj_spherI real_vector.scale_cancel_right
surf_nz)
  moreover have proj x = proj y
  by (metis (no_types) False speq homeomorphism_apply2 proj_spherI surf x

```

2862

y)
ultimately show $x = y$
using *eq eqI* by *blast*
qed
qed
have *co01*: *compact ?CBALL*
by (*simp add: compact_Int_closed*)
show *S* *homeomorphic ?CBALL*
using *homeomorphic_compact [OF co01 cont_nosp2 [unfolded o_def] im_cball inj_cball] homeomorphic_sym* by *blast*
qed

corollary

fixes *S* :: 'a::euclidean_space set
assumes *compact S* and *a*: $a \in \text{rel_interior } S$
and *star*: $\bigwedge x. x \in S \implies \text{open_segment } a \ x \subseteq \text{rel_interior } S$
shows *starlike_compact_projective1*:
 $S - \text{rel_interior } S \text{ homeomorphic sphere } a \ 1 \cap \text{affine hull } S$
and *starlike_compact_projective2*:
 $S \text{ homeomorphic cball } a \ 1 \cap \text{affine hull } S$

proof –

have *1*: *compact ((+) (-a) ' S)* by (*meson assms compact_translation*)
have *2*: $0 \in \text{rel_interior } ((+) (-a) ' S)$
using *a rel_interior_translation [of - a S]* by (*simp cong: image_cong_simp*)
have *3*: $\text{open_segment } 0 \ x \subseteq \text{rel_interior } ((+) (-a) ' S)$ if $x \in ((+) (-a) ' S)$

for *x*

proof –

have $x+a \in S$ using *that* by *auto*
then have $\text{open_segment } a \ (x+a) \subseteq \text{rel_interior } S$ by (*metis star*)
then show *?thesis* using *open_segment_translation [of a 0 x]*
using *rel_interior_translation [of - a S]* by (*fastforce simp add: ac_simps image_iff cong: image_cong_simp*)

qed

have $S - \text{rel_interior } S \text{ homeomorphic } ((+) (-a) ' S) - \text{rel_interior } ((+) (-a) ' S)$

by (*metis rel_interior_translation translation_diff homeomorphic_translation*)

also have ... *homeomorphic sphere 0 1 \cap affine hull ((+) (-a) ' S)*

by (*rule starlike_compact_projective1_0 [OF 1 2 3]*)

also have ... = $(+) (-a) ' (\text{sphere } a \ 1 \cap \text{affine hull } S)$

by (*metis affine_hull_translation left_minus sphere_translation translation_Int*)

also have ... *homeomorphic sphere a 1 \cap affine hull S*

using *homeomorphic_translation homeomorphic_sym* by *blast*

finally show $S - \text{rel_interior } S \text{ homeomorphic sphere } a \ 1 \cap \text{affine hull } S$.

have *S* *homeomorphic ((+) (-a) ' S)*

by (*metis homeomorphic_translation*)

also have ... *homeomorphic cball 0 1 \cap affine hull ((+) (-a) ' S)*

by (*rule starlike_compact_projective2_0 [OF 1 2 3]*)

also have ... = $(+) (-a) ' (\text{cball } a \ 1 \cap \text{affine hull } S)$

```

  by (metis affine_hull_translation left_minus cball_translation translation_Int)
  also have ... homeomorphic cball a 1  $\cap$  affine hull S
    using homeomorphic_translation homeomorphic_sym by blast
  finally show S homeomorphic cball a 1  $\cap$  affine hull S .
qed

```

corollary *starlike_compact_projective_special*:

```

  assumes compact S
    and cb01: cball (0::'a::euclidean_space) 1  $\subseteq$  S
    and scale:  $\bigwedge x u. \llbracket x \in S; 0 \leq u; u < 1 \rrbracket \implies u *_R x \in S - \text{frontier } S$ 
  shows S homeomorphic (cball (0::'a::euclidean_space) 1)
proof -
  have ball 0 1  $\subseteq$  interior S
    using cb01 interior_cball interior_mono by blast
  then have 0: 0  $\in$  rel_interior S
    by (meson centre_in_ball subsetD interior_subset_rel_interior le_numeral_extra(2)
not_le)
  have [simp]: affine hull S = UNIV
    using  $\langle$ ball 0 1  $\subseteq$  interior S $\rangle$  by (auto intro!: affine_hull_nonempty_interior)
  have star: open_segment 0 x  $\subseteq$  rel_interior S if x  $\in$  S for x
  proof
    fix p assume p  $\in$  open_segment 0 x
    then obtain u where x  $\neq$  0 and u: 0  $\leq$  u u < 1 and p: u *_R x = p
      by (auto simp: in_segment)
    then show p  $\in$  rel_interior S
      using scale [OF that u] closure_subset frontier_def interior_subset_rel_interior
  by fastforce
  qed
  show ?thesis
    using starlike_compact_projective2_0 [OF  $\langle$ compact S $\rangle$  0 star] by simp
qed

```

lemma *homeomorphic_convex_lemma*:

```

  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S compact S convex T compact T
    and affeq: aff_dim S = aff_dim T
  shows (S - rel_interior S) homeomorphic (T - rel_interior T)  $\wedge$ 
    S homeomorphic T
proof (cases rel_interior S = {}  $\vee$  rel_interior T = {})
  case True
    then show ?thesis
      by (metis Diff_empty affeq  $\langle$ convex S $\rangle$   $\langle$ convex T $\rangle$  aff_dim_empty homeomorphic_empty
rel_interior_eq_empty aff_dim_empty)
  next
  case False
    then obtain a b where a: a  $\in$  rel_interior S and b: b  $\in$  rel_interior T by auto
    have starS:  $\bigwedge x. x \in S \implies$  open_segment a x  $\subseteq$  rel_interior S
      using rel_interior_closure_convex_segment
      a  $\langle$ convex S $\rangle$  closure_subset subsetCE by blast

```

```

have starT:  $\bigwedge x. x \in T \implies \text{open\_segment } b \ x \subseteq \text{rel\_interior } T$ 
  using rel_interior_closure_convex_segment
  b <convex T> closure_subset subsetCE by blast
let ?aS = (+) (-a) ' S and ?bT = (+) (-b) ' T
have 0:  $0 \in \text{affine\_hull } ?aS \ 0 \in \text{affine\_hull } ?bT$ 
  by (metis a b subsetD hull_inc image_eqI left_minus rel_interior_subset)+
have subs: subspace (span ?aS) subspace (span ?bT)
  by (rule subspace_span)+
moreover
have dim (span ((+) (- a) ' S)) = dim (span ((+) (- b) ' T))
  by (metis 0 aff_dim_translation_eq aff_dim_zero affeq dim_span nat_int)
ultimately obtain f g where linear f linear g
  and fim:  $f ' \text{span } ?aS = \text{span } ?bT$ 
  and gim:  $g ' \text{span } ?bT = \text{span } ?aS$ 
  and fno:  $\bigwedge x. x \in \text{span } ?aS \implies \text{norm}(f \ x) = \text{norm } x$ 
  and gno:  $\bigwedge x. x \in \text{span } ?bT \implies \text{norm}(g \ x) = \text{norm } x$ 
  and gf:  $\bigwedge x. x \in \text{span } ?aS \implies g(f \ x) = x$ 
  and fg:  $\bigwedge x. x \in \text{span } ?bT \implies f(g \ x) = x$ 
  by (rule isometries_subspaces) blast
have [simp]: continuous_on A f for A
  using <linear f> linear_conv_bounded_linear linear_continuous_on by blast
have [simp]: continuous_on B g for B
  using <linear g> linear_conv_bounded_linear linear_continuous_on by blast
have eqspanS:  $\text{affine\_hull } ?aS = \text{span } ?aS$ 
  by (metis a affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)
have eqspanT:  $\text{affine\_hull } ?bT = \text{span } ?bT$ 
  by (metis b affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)
have S homeomorphic cball a 1  $\cap$  affine hull S
  by (rule starlike_compact_projective2 [OF <compact S> a starS])
also have ... homeomorphic (+) (-a) ' (cball a 1  $\cap$  affine hull S)
  by (metis homeomorphic_translation)
also have ... = cball 0 1  $\cap$  (+) (-a) ' (affine hull S)
  by (auto simp: dist_norm)
also have ... = cball 0 1  $\cap$  span ?aS
  using eqspanS affine_hull_translation by blast
also have ... homeomorphic cball 0 1  $\cap$  span ?bT
proof (rule homeomorphicI)
  show fim1:  $f ' (\text{cball } 0 \ 1 \cap \text{span } ?aS) = \text{cball } 0 \ 1 \cap \text{span } ?bT$ 
  proof
    show  $f ' (\text{cball } 0 \ 1 \cap \text{span } ?aS) \subseteq \text{cball } 0 \ 1 \cap \text{span } ?bT$ 
      using fim fno by auto
    show  $\text{cball } 0 \ 1 \cap \text{span } ?bT \subseteq f ' (\text{cball } 0 \ 1 \cap \text{span } ?aS)$ 
      by clarify (metis IntI fg gim gno image_eqI mem_cball_0)
  qed
  show  $g ' (\text{cball } 0 \ 1 \cap \text{span } ?bT) = \text{cball } 0 \ 1 \cap \text{span } ?aS$ 
  proof
    show  $g ' (\text{cball } 0 \ 1 \cap \text{span } ?bT) \subseteq \text{cball } 0 \ 1 \cap \text{span } ?aS$ 
      using gim gno by auto
    show  $\text{cball } 0 \ 1 \cap \text{span } ?aS \subseteq g ' (\text{cball } 0 \ 1 \cap \text{span } ?bT)$ 

```

```

    by clarify (metis IntI fim1 gf image_eqI)
  qed
qed (auto simp: fg gf)
also have ... = cball 0 1  $\cap$  (+) (-b) ' (affine hull T)
  using eqspanT affine_hull_translation by blast
also have ... = (+) (-b) ' (cball b 1  $\cap$  affine hull T)
  by (auto simp: dist_norm)
also have ... homeomorphic (cball b 1  $\cap$  affine hull T)
  by (metis homeomorphic_translation homeomorphic_sym)
also have ... homeomorphic T
  by (metis starlike_compact_projective2 [OF <compact T> b starT] homeomor-
    phic_sym)
finally have 1: S homeomorphic T .

have S - rel_interior S homeomorphic sphere a 1  $\cap$  affine hull S
  by (rule starlike_compact_projective1 [OF <compact S> a starS])
also have ... homeomorphic (+) (-a) ' (sphere a 1  $\cap$  affine hull S)
  by (metis homeomorphic_translation)
also have ... = sphere 0 1  $\cap$  (+) (-a) ' (affine hull S)
  by (auto simp: dist_norm)
also have ... = sphere 0 1  $\cap$  span ?aS
  using eqspanS affine_hull_translation by blast
also have ... homeomorphic sphere 0 1  $\cap$  span ?bT
proof (rule homeomorphicI)
  show fim1: f ' (sphere 0 1  $\cap$  span ?aS) = sphere 0 1  $\cap$  span ?bT
proof
  show f ' (sphere 0 1  $\cap$  span ?aS)  $\subseteq$  sphere 0 1  $\cap$  span ?bT
  using fim fno by auto
  show sphere 0 1  $\cap$  span ?bT  $\subseteq$  f ' (sphere 0 1  $\cap$  span ?aS)
  by clarify (metis IntI fg gim gno image_eqI mem_sphere_0)
qed
show g ' (sphere 0 1  $\cap$  span ?bT) = sphere 0 1  $\cap$  span ?aS
proof
  show g ' (sphere 0 1  $\cap$  span ?bT)  $\subseteq$  sphere 0 1  $\cap$  span ?aS
  using gim gno by auto
  show sphere 0 1  $\cap$  span ?aS  $\subseteq$  g ' (sphere 0 1  $\cap$  span ?bT)
  by clarify (metis IntI fim1 gf image_eqI)
qed
qed (auto simp: fg gf)
also have ... = sphere 0 1  $\cap$  (+) (-b) ' (affine hull T)
  using eqspanT affine_hull_translation by blast
also have ... = (+) (-b) ' (sphere b 1  $\cap$  affine hull T)
  by (auto simp: dist_norm)
also have ... homeomorphic (sphere b 1  $\cap$  affine hull T)
  by (metis homeomorphic_translation homeomorphic_sym)
also have ... homeomorphic T - rel_interior T
  by (metis starlike_compact_projective1 [OF <compact T> b starT] homeomor-
    phic_sym)
finally have 2: S - rel_interior S homeomorphic T - rel_interior T .

```

```

show ?thesis
  using 1 2 by blast
qed

```

```

lemma homeomorphic_convex_compact_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S compact S convex T compact T
    and affeq: aff_dim S = aff_dim T
  shows S homeomorphic T
using homeomorphic_convex_lemma [OF assms] assms
by (auto simp: rel_frontier_def)

```

```

lemma homeomorphic_rel_frontiers_convex_bounded_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S bounded S convex T bounded T
    and affeq: aff_dim S = aff_dim T
  shows rel_frontier S homeomorphic rel_frontier T
using assms homeomorphic_convex_lemma [of closure S closure T]
by (simp add: rel_frontier_def convex_rel_interior_closure)

```

9.8.2 Homeomorphisms between punctured spheres and affine sets

Including the famous stereoscopic projection of the 3-D sphere to the complex plane

The special case with centre 0 and radius 1

```

lemma homeomorphic_punctured_affine_sphere_affine_01:
  assumes b ∈ sphere 0 1 affine T 0 ∈ T b ∈ T affine p
    and affT: aff_dim T = aff_dim p + 1
  shows (sphere 0 1 ∩ T) - {b} homeomorphic p
proof -
  have [simp]: norm b = 1 b·b = 1
    using assms by (auto simp: norm_eq_1)
  have [simp]: T ∩ {v. b·v = 0} ≠ {}
    using ⟨0 ∈ T⟩ by auto
  have [simp]: ¬ T ⊆ {v. b·v = 0}
    using ⟨norm b = 1⟩ ⟨b ∈ T⟩ by auto
  define f where f ≡ λx. 2 *R b + (2 / (1 - b·x)) *R (x - b)
  define g where g ≡ λy. b + (4 / (norm y ^ 2 + 4)) *R (y - 2 *R b)
  have fg[simp]: ∧x. [x ∈ T; b·x = 0] ⇒ f (g x) = x
    unfolding f_def g_def by (simp add: algebra_simps field_split_simps add_nonneg_eq_0_iff)
  have no: (norm (f x))^2 = 4 * (1 + b·x) / (1 - b·x)
    if norm x = 1 and b·x ≠ 1 for x
    using that sum_sqs_eq [of 1 b·x]
  apply (simp flip: dot_square_norm add: norm_eq_1 nonzero_eq_divide_eq)
  apply (simp add: f_def vector_add_divide_simps inner_simps)
  apply (auto simp add: field_split_simps inner_commute)
done

```

```

have [simp]:  $\bigwedge u::\text{real}. 8 + u * (u * 8) = u * 16 \longleftrightarrow u=1$ 
  by algebra
have gf[simp]:  $\bigwedge x. \llbracket \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies g(fx) = x$ 
  unfolding g_def no by (auto simp: f_def field_split_simps)
have g1:  $\text{norm}(gx) = 1$  if  $x \in T$  and  $b \cdot x = 0$  for  $x$ 
  using that
  apply (simp only: g_def)
  apply (rule power2_eq_imp_eq)
  apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps)
  apply (simp add: algebra_simps inner_commute)
  done
have ne1:  $b \cdot gx \neq 1$  if  $x \in T$  and  $b \cdot x = 0$  for  $x$ 
  using that unfolding g_def
  apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps
add_nonneg_eq_0_iff)
  apply (auto simp: algebra_simps)
  done
have subspace T
  by (simp add: assms subspace_affine)
have gT:  $\bigwedge x. \llbracket x \in T; b \cdot x = 0 \rrbracket \implies gx \in T$ 
  unfolding g_def
  by (blast intro:  $\langle \text{subspace } T \rangle \langle b \in T \rangle$  subspace_add subspace_mul subspace_diff)
have f'  $\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \subseteq \{x. b \cdot x = 0\}$ 
  unfolding f_def using  $\langle \text{norm } b = 1 \rangle$  norm_eq_1
  by (force simp: field_simps inner_add_right inner_diff_right)
moreover have f'  $T \subseteq T$ 
  unfolding f_def using assms  $\langle \text{subspace } T \rangle$ 
  by (auto simp add: inner_add_right inner_diff_right mem_affine_3_minus
subspace_mul)
moreover have  $\{x. b \cdot x = 0\} \cap T \subseteq f'(\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T)$ 
  by clarify (metis (mono_tags) IntI ne1 fg gT g1 imageI mem_Collect_eq)
ultimately have  $\text{im}f: f'(\{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\} \cap T) = \{x. b \cdot x = 0\} \cap T$ 
  by blast
have no4:  $\bigwedge y. b \cdot y = 0 \implies \text{norm}((y \cdot y + 4) *_{\mathbb{R}} b + 4 *_{\mathbb{R}} (y - 2 *_{\mathbb{R}} b)) = y \cdot y + 4$ 
  apply (rule power2_eq_imp_eq)
  apply (simp_all flip: dot_square_norm)
  apply (auto simp: power2_eq_square algebra_simps inner_commute)
  done
have [simp]:  $\bigwedge x. \llbracket \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies b \cdot fx = 0$ 
  by (simp add: f_def algebra_simps field_split_simps)
have [simp]:  $\bigwedge x. \llbracket x \in T; \text{norm } x = 1; b \cdot x \neq 1 \rrbracket \implies fx \in T$ 
  unfolding f_def
  by (blast intro:  $\langle \text{subspace } T \rangle \langle b \in T \rangle$  subspace_add subspace_mul subspace_diff)
have g'  $\{x. b \cdot x = 0\} \subseteq \{x. \text{norm } x = 1 \wedge b \cdot x \neq 1\}$ 
  unfolding g_def
  apply (clarify simp: no4 vector_add_divide_simps divide_simps add_nonneg_eq_0_iff
dot_square_norm [symmetric])

```

```

    apply (auto simp: algebra_simps)
  done
  moreover have  $g \text{ ' } T \subseteq T$ 
    unfolding  $g\_def$ 
  by (blast intro: ‹subspace T› ‹ $b \in T$ ›  $subspace\_add$   $subspace\_mul$   $subspace\_diff$ )
  moreover have  $\{x. norm\ x = 1 \wedge b \cdot x \neq 1\} \cap T \subseteq g \text{ ' } (\{x. b \cdot x = 0\} \cap T)$ 
    by clarify (metis (mono_tags, lifting) IntI gf_image_iff imf mem_Collect_eq)
  ultimately have  $img: g \text{ ' } (\{x. b \cdot x = 0\} \cap T) = \{x. norm\ x = 1 \wedge b \cdot x \neq 1\} \cap T$ 
  T
    by blast
  have  $aff: affine\ (\{x. b \cdot x = 0\} \cap T)$ 
    by (blast intro: affine_hyperplane assms)
  have  $contf: continuous\_on\ (\{x. norm\ x = 1 \wedge b \cdot x \neq 1\} \cap T) f$ 
    unfolding  $f\_def$  by (rule continuous_intros | force)+
  have  $contg: continuous\_on\ (\{x. b \cdot x = 0\} \cap T) g$ 
    unfolding  $g\_def$  by (rule continuous_intros | force simp: add_nonneg_eq_0_iff)+
  have  $(sphere\ 0\ 1 \cap T) - \{b\} = \{x. norm\ x = 1 \wedge (b \cdot x \neq 1)\} \cap T$ 
    using ‹ $norm\ b = 1$ › by (auto simp: norm_eq_1) (metis vector_eq ‹ $b \cdot b = 1$ ›)
  also have ... homeomorphic  $\{x. b \cdot x = 0\} \cap T$ 
    by (rule homeomorphicI [OF imf img contf contg]) auto
  also have ... homeomorphic p
  proof (rule homeomorphic_affine_sets [OF aff ‹affine p›])
    show  $aff\_dim\ (\{x. b \cdot x = 0\} \cap T) = aff\_dim\ p$ 
      by (simp add: Int_commute aff_dim_affine_Int_hyperplane [OF ‹affine T›])
  qed
  finally show ?thesis .
qed

```

```

theorem homeomorphic_punctured_affine_sphere_affine:
  fixes  $a :: 'a :: euclidean\_space$ 
  assumes  $0 < r$   $b \in sphere\ a\ r$   $affine\ T$   $a \in T$   $b \in T$   $affine\ p$ 
  and  $aff: aff\_dim\ T = aff\_dim\ p + 1$ 
  shows  $(sphere\ a\ r \cap T) - \{b\}$  homeomorphic p
proof -
  have  $a \neq b$  using assms by auto
  then have  $inj: inj\ (\lambda x::'a. x /_R norm\ (a - b))$ 
    by (simp add: inj_on_def)
  have  $((sphere\ a\ r \cap T) - \{b\})$  homeomorphic
    (+)  $(-a) \text{ ' } ((sphere\ a\ r \cap T) - \{b\})$ 
    by (rule homeomorphic_translation)
  also have ... homeomorphic  $(*_R)\ (inverse\ r) \text{ ' } (+)\ (-\ a) \text{ ' } (sphere\ a\ r \cap T - \{b\})$ 
    by (metis ‹ $0 < r$ › homeomorphic_scaling inverse_inverse_eq inverse_zero less_irrefl)
  also have ... =  $sphere\ 0\ 1 \cap ((*_R)\ (inverse\ r) \text{ ' } (+)\ (-\ a) \text{ ' } T) - \{(b - a) /_R r\}$ 
    using assms by (auto simp: dist_norm norm_minus_commute divide_simps)
  also have ... homeomorphic p

```


using *assms* *affine_translation* [*symmetric*, *of - a*] *aff_dim_translation_eq* [*of - a*]
by (*intro* *homeomorphic_punctured_affine_sphere_affine_01*) (*auto simp:* *dist_norm norm_minus_commute affine_scaling inj*)
finally show *?thesis* .
qed

corollary *homeomorphic_punctured_sphere_affine:*

fixes *a* :: '*a* :: *euclidean_space*
assumes $0 < r$ **and** *b*: $b \in \text{sphere } a \ r$
and *affine T* **and** *affS*: $\text{aff_dim } T + 1 = \text{DIM}('a)$
shows (*sphere* *a r - {b}*) *homeomorphic T*
using *homeomorphic_punctured_affine_sphere_affine* [*of r b a UNIV T*] *assms*
by *auto*

corollary *homeomorphic_punctured_sphere_hyperplane:*

fixes *a* :: '*a* :: *euclidean_space*
assumes $0 < r$ **and** *b*: $b \in \text{sphere } a \ r$
and $c \neq 0$
shows (*sphere* *a r - {b}*) *homeomorphic {x::'a. c · x = d}*
using *assms*
by (*intro* *homeomorphic_punctured_sphere_affine*) (*auto simp:* *affine_hyperplane of_nat_diff*)

proposition *homeomorphic_punctured_sphere_affine_gen:*

fixes *a* :: '*a* :: *euclidean_space*
assumes *convex S bounded S* **and** *a*: $a \in \text{rel_frontier } S$
and *affine T* **and** *affS*: $\text{aff_dim } S = \text{aff_dim } T + 1$
shows $\text{rel_frontier } S - \{a\}$ *homeomorphic T*
proof -
obtain *U* :: '*a* *set* **where** *affine U convex U* **and** *affdS*: $\text{aff_dim } U = \text{aff_dim } S$
using *choose_affine_subset* [*OF* *affine_UNIV aff_dim_geq*]
by (*meson* *aff_dim_affine_hull affine_affine_hull affine_imp_convex*)
have $S \neq \{\}$ **using** *assms* **by** *auto*
then obtain *z* **where** $z \in U$
by (*metis* *aff_dim_negative_iff equals0I affdS*)
then have *bne*: $\text{ball } z \ 1 \cap U \neq \{\}$ **by** *force*
then have [*simp*]: $\text{aff_dim}(\text{ball } z \ 1 \cap U) = \text{aff_dim } U$
using *aff_dim_convex_Int_open* [*OF* $\langle \text{convex } U \rangle$ *open_ball*]
by (*fastforce* *simp* *add: Int_commute*)
have $\text{rel_frontier } S$ *homeomorphic* $\text{rel_frontier}(\text{ball } z \ 1 \cap U)$
by (*rule* *homeomorphic_rel_frontiers_convex_bounded_sets*)
*(auto simp: $\langle \text{affine } U \rangle$ *affine_imp_convex convex_Int affdS assms*)
also have $\dots = \text{sphere } z \ 1 \cap U$
using *convex_affine_rel_frontier_Int* [*of* $\text{ball } z \ 1 \ U$]
by (*simp* *add: $\langle \text{affine } U \rangle$ bne*)
finally have $\text{rel_frontier } S$ *homeomorphic* $\text{sphere } z \ 1 \cap U$.
then obtain *h k* **where** *him*: $h \text{ ' rel_frontier } S = \text{sphere } z \ 1 \cap U$
and *kim*: $k \text{ ' } (\text{sphere } z \ 1 \cap U) = \text{rel_frontier } S$*

```

and hcon: continuous_on (rel_frontier S) h
and kcon: continuous_on (sphere z 1 ∩ U) k
and kh:  $\bigwedge x. x \in \text{rel\_frontier } S \implies k(h(x)) = x$ 
and hk:  $\bigwedge y. y \in \text{sphere } z \ 1 \cap U \implies h(k(y)) = y$ 
unfolding homeomorphic_def homeomorphism_def by auto
have rel_frontier S - {a} homeomorphic (sphere z 1 ∩ U) - {h a}
proof (rule homeomorphicI)
  show h:  $h \text{ ' (rel\_frontier } S - \{a\}) = \text{sphere } z \ 1 \cap U - \{h \ a\}$ 
    using him a kh by auto metis
  show k:  $k \text{ ' (sphere } z \ 1 \cap U - \{h \ a\}) = \text{rel\_frontier } S - \{a\}$ 
    by (force simp: h [symmetric] image_comp o_def kh)
qed (auto intro: continuous_on_subset hcon kcon simp: kh hk)
also have ... homeomorphic T
  by (rule homeomorphic_punctured_affine_sphere_affine)
    (use a him in ⟨auto simp: affS affdS ⟨affine T⟩ ⟨affine U⟩ ⟨z ∈ U⟩⟩)
finally show ?thesis .
qed

```

When dealing with AR, ANR and ANR later, it's useful to know that every set is homeomorphic to a closed subset of a convex set, and if the set is locally compact we can take the convex set to be the universe.

proposition *homeomorphic_closedin_convex*:

```

fixes S :: 'm::euclidean_space set
assumes aff_dim S < DIM('n)
obtains U and T :: 'n::euclidean_space set
  where convex U U ≠ {} closedin (top_of_set U) T
    S homeomorphic T
proof (cases S = {})
  case True then show ?thesis
    by (rule_tac U=UNIV and T={ } in that) auto
next
  case False
  then obtain a where a ∈ S by auto
  obtain i::'n where i: i ∈ Basis i ≠ 0
    using SOME_Basis Basis_zero by force
  have 0 ∈ affine hull ((+) (- a) ' S)
    by (simp add: ⟨a ∈ S⟩ hull_inc)
  then have dim ((+) (- a) ' S) = aff_dim ((+) (- a) ' S)
    by (simp add: aff_dim_zero)
  also have ... < DIM('n)
    by (simp add: aff_dim_translation_eq_subtract assms cong: image_cong_simp)
  finally have dd: dim ((+) (- a) ' S) < DIM('n)
    by linarith
  have span: span {x. i · x = 0} = {x. i · x = 0}
    using span_eq_iff [symmetric, of {x. i · x = 0}] subspace_hyperplane [of i]
by simp
  have dim ((+) (- a) ' S) ≤ dim {x. i · x = 0}
    using dd by (simp add: dim_hyperplane [OF ⟨i ≠ 0⟩])
  then obtain T where subspace T and Tsub: T ⊆ {x. i · x = 0}

```

```

  and dimT: dim T = dim ((+) (- a) ' S)
  by (rule choose_subspace_of_subspace) (simp add: span)
  have subspace (span ((+) (- a) ' S))
  using subspace_span by blast
  then obtain h k where linear h linear k
    and heq: h ' span ((+) (- a) ' S) = T
    and keq: k ' T = span ((+) (- a) ' S)
    and hinv [simp]:  $\bigwedge x. x \in \text{span } ((+) (- a) ' S) \implies k(h x) = x$ 
    and kinv [simp]:  $\bigwedge x. x \in T \implies h(k x) = x$ 
  by (auto simp: dimT intro: isometries_subspaces [OF_ <subspace T>] dimT)
  have hcont: continuous_on A h and kcont: continuous_on B k for A B
  using <linear h> <linear k> linear_continuous_on linear_conv_bounded_linear
  by blast+
  have ihhh [simp]:  $\bigwedge x. x \in S \implies i \cdot h (x - a) = 0$ 
  using Tsub [THEN subsetD] heq span_superset by fastforce
  have sphere 0 1 - {i} homeomorphic {x. i \cdot x = 0}
  proof (rule homeomorphic_punctured_sphere_affine)
    show affine {x. i \cdot x = 0}
    by (auto simp: affine_hyperplane)
    show aff_dim {x. i \cdot x = 0} + 1 = int DIM('n)
    using i by clarsimp (metis DIM_positive Suc_pred add.commute of_nat_Suc)
  qed (use i in auto)
  then obtain f g where fg: homeomorphism (sphere 0 1 - {i}) {x. i \cdot x = 0} f
  g
  by (force simp: homeomorphic_def)
  show ?thesis
  proof
    have h ' (+) (- a) ' S  $\subseteq$  T
    using heq span_superset span_linear_image by blast
    then have g ' h ' (+) (- a) ' S  $\subseteq$  g ' {x. i \cdot x = 0}
    using Tsub by (simp add: image_mono)
    also have ...  $\subseteq$  sphere 0 1 - {i}
    by (simp add: fg [unfolded homeomorphism_def])
    finally have gh_sub_sph: (g \circ h) ' (+) (- a) ' S  $\subseteq$  sphere 0 1 - {i}
    by (metis image_comp)
    then have gh_sub_cb: (g \circ h) ' (+) (- a) ' S  $\subseteq$  cball 0 1
    by (metis Diff_subset order_trans sphere_cball)
    have [simp]:  $\bigwedge u. u \in S \implies \text{norm } (g (h (u - a))) = 1$ 
    using gh_sub_sph [THEN subsetD] by (auto simp: o_def)
    show convex (ball 0 1  $\cup$  (g \circ h) ' (+) (- a) ' S)
    by (meson ball_subset_cball convex_intermediate_ball gh_sub_cb sup.bounded_iff
    sup.cobounded1)
    show closedin (top_of_set (ball 0 1  $\cup$  (g \circ h) ' (+) (- a) ' S)) ((g \circ h) ' (+)
    (- a) ' S)
    unfolding closedin_closed
    by (rule_tac x=sphere 0 1 in exI) auto
    have ghcont: continuous_on ((\lambda x. x - a) ' S) (\lambda x. g (h x))
    by (rule continuous_on_compose2 [OF homeomorphism_cont2 [OF fg] hcont],
    force)

```

```

have kfcont: continuous_on ((λx. g (h (x - a))) ' S) (λx. k (f x))
proof (rule continuous_on_compose2 [OF kcont])
  show continuous_on ((λx. g (h (x - a))) ' S) f
    using homeomorphism_cont1 [OF fg] gh_sub_sph by (fastforce intro:
continuous_on_subset)
qed auto
have S homeomorphic (+) (- a) ' S
  by (fact homeomorphic_translation)
also have ... homeomorphic (g ∘ h) ' (+) (- a) ' S
apply (simp add: homeomorphic_def homeomorphism_def cong: image_cong_simp)
apply (rule_tac x=g ∘ h in exI)
apply (rule_tac x=k ∘ f in exI)
apply (auto simp: ghcont kfcont span_base homeomorphism_apply2 [OF fg]
image_comp cong: image_cong_simp)
done
finally show S homeomorphic (g ∘ h) ' (+) (- a) ' S .
qed auto
qed

```

9.8.3 Locally compact sets in an open set

Locally compact sets are closed in an open set and are homeomorphic to an absolutely closed set if we have one more dimension to play with.

lemma *locally_compact_open_Int_closure*:

fixes $S :: 'a :: \text{metric_space}$ set

assumes *locally_compact* S

obtains T **where** *open* T S = T ∩ closure S

proof –

have $\forall x \in S. \exists T v u. u = S \cap T \wedge x \in u \wedge u \subseteq v \wedge v \subseteq S \wedge \text{open } T \wedge \text{compact } v$

by (*metis* *assms* *locally_compact* *openin_open*)

then obtain t v **where**

$tv: \bigwedge x. x \in S$

$\implies v x \subseteq S \wedge \text{open } (t x) \wedge \text{compact } (v x) \wedge (\exists u. x \in u \wedge u \subseteq v x \wedge u$

$= S \cap t x)$

by *metis*

then have o: *open* ($\bigcup (t ' S)$)

by *blast*

have S = $\bigcup (v ' S)$

using tv **by** *blast*

also have ... = $\bigcup (t ' S) \cap \text{closure } S$

proof

show $\bigcup (v ' S) \subseteq \bigcup (t ' S) \cap \text{closure } S$

by *clarify* (*meson* *IntD2* *IntI* *UN_I* *closure_subset* *subsetD* tv)

have t x ∩ closure S $\subseteq v x$ **if** x ∈ S **for** x

proof –

have t x ∩ closure S $\subseteq \text{closure } (t x \cap S)$

by (*simp* *add*: *open_Int_closure_subset* that tv)

also have ... $\subseteq v x$

```

    by (metis Int_commute closure_minimal compact_imp_closed that tv)
  finally show ?thesis .
qed
then show  $\bigcup (t \text{ ' } S) \cap \text{closure } S \subseteq \bigcup (v \text{ ' } S)$ 
  by blast
qed
finally have  $e: S = \bigcup (t \text{ ' } S) \cap \text{closure } S$  .
show ?thesis
  by (rule that [OF o e])
qed

```

```

lemma locally_compact_closedin_open:
  fixes  $S :: 'a :: \text{metric\_space set}$ 
  assumes locally_compact  $S$ 
  obtains  $T$  where open  $T$  closedin (top_of_set  $T$ )  $S$ 
  by (metis locally_compact_open_Int_closure [OF assms] closed_closure closedin_closed_Int)

```

```

lemma locally_compact_homeomorphism_projection_closed:
  assumes locally_compact  $S$ 
  obtains  $T$  and  $f :: 'a \Rightarrow 'a :: \text{euclidean\_space} \times 'b :: \text{euclidean\_space}$ 
  where closed  $T$  homeomorphism  $S$   $T$   $f$  fst
proof (cases closed  $S$ )
case True
  show ?thesis
  proof
    show homeomorphism  $S$  ( $S \times \{0\}$ ) ( $\lambda x. (x, 0)$ ) fst
    by (auto simp: homeomorphism_def continuous_intros)
  qed (use True closed_Times in auto)
next
case False
  obtain  $U$  where open  $U$  and  $US: U \cap \text{closure } S = S$ 
  by (metis locally_compact_open_Int_closure [OF assms])
  with False have  $U\text{comp}: -U \neq \{\}$ 
  using closure_eq by auto
  have [simp]:  $\text{closure } (-U) = -U$ 
  by (simp add: <open  $U$ > closed_Compl)
  define  $f :: 'a \Rightarrow 'a \times 'b$  where  $f \equiv \lambda x. (x, \text{One } /_R \text{ setdist } \{x\} (-U))$ 
  have continuous_on  $U$  ( $\lambda x. (x, \text{One } /_R \text{ setdist } \{x\} (-U))$ )
  proof (intro continuous_intros continuous_on_setdist)
    show  $\forall x \in U. \text{setdist } \{x\} (-U) \neq 0$ 
    by (simp add:  $U\text{comp setdist_eq_0\_sing_1}$ )
  qed
  qed
  then have  $\text{hom}U: \text{homeomorphism } U (f'U) f$  fst
  by (auto simp:  $f$ _def homeomorphism_def image_iff continuous_intros)
  have  $\text{clo}S: \text{closedin } (\text{top\_of\_set } U) S$ 
  by (metis  $US$  closed_closure closedin_closed_Int)
  have  $\text{cont}: \text{isCont } ((\lambda x. \text{setdist } \{x\} (-U)) \circ f) z$  for  $z :: 'a \times 'b$ 

```

```

    by (rule continuous_at_compose_continuous_intros continuous_at_setdist)+
    have setdist1D: setdist {a} (- U) *R b = One  $\implies$  setdist {a} (- U)  $\neq$  0 for
a::'a and b::'b
    by force
    have *: r *R b = One  $\implies$  b = (1 / r) *R One for r and b::'b
    by (metis One_non_0 nonzero_divide_eq_eq real_vector.scale_eq_0_iff
real_vector.scale_scale scaleR_one)
    have  $\bigwedge$  a b::'b. setdist {a} (- U) *R b = One  $\implies$  (a,b)  $\in$  ( $\lambda$ x. (x, (1 / setdist
{x} (- U)) *R One)) ' U
    by (metis (mono_tags, lifting) * ComplI image_eqI setdist1D setdist_sing_in_set)
    then have f ' U = ( $\lambda$ z. (setdist {fst z} (- U) *R snd z)) - ' {One}
    by (auto simp: f_def setdist_eq_0_sing_1 field_simps Ucomp)
    then have clfU: closed (f ' U)
    by (force intro: continuous_intros cont [unfolded o_def] continuous_closed_vimage)
    have closed (f ' S)
    by (metis closedin_closed_trans [OF clfU] homeomorphism_imp_closed_map
[OF homU cloS])
    then show ?thesis
    by (metis US homU homeomorphism_of_subsets inf_sup_ord(1) that)
qed

```

lemma *locally_compact_closed_Int_open*:

```

    fixes S :: 'a :: euclidean_space set
    shows locally_compact S  $\longleftrightarrow$  ( $\exists$  U V. closed U  $\wedge$  open V  $\wedge$  S = U  $\cap$  V) (is
?lhs = ?rhs)
proof
    show ?lhs  $\implies$  ?rhs
    by (metis closed_closure inf_commute locally_compact_open_Int_closure)
    show ?rhs  $\implies$  ?lhs
    by (meson closed_imp_locally_compact locally_compact_Int_open_imp_locally_compact)
qed

```

lemma *lowerdim_embeddings*:

```

    assumes DIM('a) < DIM('b)
    obtains f :: 'a::euclidean_space*real  $\Rightarrow$  'b::euclidean_space
    and g :: 'b  $\Rightarrow$  'a*real
    and j :: 'b
    where linear f linear g  $\bigwedge$  z. g (f z) = z j  $\in$  Basis  $\bigwedge$  x. f(x,0)  $\cdot$  j = 0
proof -
    let ?B = Basis :: ('a*real) set
    have b01: (0,1)  $\in$  ?B
    by (simp add: Basis_prod_def)
    have DIM('a * real)  $\leq$  DIM('b)
    by (simp add: Suc_leI assms)
    then obtain basf :: 'a*real  $\Rightarrow$  'b where sbf: basf ' ?B  $\subseteq$  Basis and injbf: inj_on
basf Basis
    by (metis finite_Basis card_le_inj)
    define basg:: 'b  $\Rightarrow$  'a * real where
    basg  $\equiv$   $\lambda$ i. if i  $\in$  basf ' Basis then inv_into Basis basf i else (0,1)

```

```

have bgf[simp]: basg (basf i) = i if i ∈ Basis for i
  using inv_into_f_f injbf that by (force simp: basg_def)
have sbg: basg 'Basis ⊆ ?B
  by (force simp: basg_def injbf b01)
define f :: 'a*real ⇒ 'b where f ≡ λu. ∑ j∈Basis. (u · basg j) *R j
define g :: 'b ⇒ 'a*real where g ≡ λz. (∑ i∈Basis. (z · basf i) *R i)
show ?thesis
proof
  show linear f
    unfolding f_def
    by (intro linear_compose_sum linearI ballI) (auto simp: algebra_simps)
  show linear g
    unfolding g_def
    by (intro linear_compose_sum linearI ballI) (auto simp: algebra_simps)
  have *: (∑ a ∈ Basis. a · basf b * (x · basg a)) = x · b if b ∈ Basis for x b
    using sbf that by auto
  show gf: g (f x) = x for x
  proof (rule euclidean_eqI)
    show ∧ b. b ∈ Basis ⇒ g (f x) · b = x · b
      using f_def g_def sbf by auto
  qed
  show basf(0,1) ∈ Basis
    using b01 sbf by auto
  then show f(x,0) · basf(0,1) = 0 for x
    unfolding f_def inner_sum_left
    using b01 inner_not_same_Basis
    by (fastforce intro: comm_monoid_add_class.sum.neutral)
  qed
qed

proposition locally_compact_homeomorphic_closed:
  fixes S :: 'a::euclidean_space set
  assumes locally_compact S and dimlt: DIM('a) < DIM('b)
  obtains T :: 'b::euclidean_space set where closed T S homeomorphic T
proof –
  obtain U:: ('a*real)set and h
    where closed U and homU: homeomorphism S U h fst
    using locally_compact_homeomorphism_projection_closed assms by metis
  obtain f :: 'a*real ⇒ 'b and g :: 'b ⇒ 'a*real
    where linear f linear g and gf [simp]: ∧ z. g (f z) = z
    using lowerdim_embeddings [OF dimlt] by metis
  then have inj f
    by (metis injI)
  have gfU: g 'f ' U = U
    by (simp add: image_comp o_def)
  have S homeomorphic U
    using homU homeomorphic_def by blast
  also have ... homeomorphic f ' U
  proof (rule homeomorphicI [OF refl gfU])

```

```

    show continuous_on U f
    by (meson ⟨inj f⟩ ⟨linear f⟩ homeomorphism_cont2 linear_homeomorphism_image)
    show continuous_on (f ' U) g
    using ⟨linear g⟩ linear_continuous_on linear_conv_bounded_linear by blast
  qed (auto simp: o_def)
  finally show ?thesis
    using ⟨closed U⟩ ⟨inj f⟩ ⟨linear f⟩ closed_injective_linear_image that by blast
  qed

```

lemma *homeomorphic_convex_compact_lemma*:

```

  fixes S :: 'a::euclidean_space set
  assumes convex S
    and compact S
    and cball 0 1 ⊆ S
  shows S homeomorphic (cball (0::'a) 1)
proof (rule starlike_compact_projective_special[OF assms(2-3)])
  fix x u
  assume x ∈ S and 0 ≤ u and u < (1::real)
  have open (ball (u *_R x) (1 - u))
    by (rule open_ball)
  moreover have u *_R x ∈ ball (u *_R x) (1 - u)
    unfolding centre_in_ball using ⟨u < 1⟩ by simp
  moreover have ball (u *_R x) (1 - u) ⊆ S
proof
  fix y
  assume y ∈ ball (u *_R x) (1 - u)
  then have dist (u *_R x) y < 1 - u
    unfolding mem_ball .
  with ⟨u < 1⟩ have inverse (1 - u) *_R (y - u *_R x) ∈ cball 0 1
    by (simp add: dist_norm inverse_eq_divide norm_minus_commute)
  with assms(3) have inverse (1 - u) *_R (y - u *_R x) ∈ S ..
  with assms(1) have (1 - u) *_R ((y - u *_R x) /_R (1 - u)) + u *_R x ∈ S
    using ⟨x ∈ S⟩ ⟨0 ≤ u⟩ ⟨u < 1⟩ [THEN less_imp_le] by (rule convexD_alt)
  then show y ∈ S using ⟨u < 1⟩
    by simp
  qed
  ultimately have u *_R x ∈ interior S ..
  then show u *_R x ∈ S - frontier S
    using frontier_def and interior_subset by auto
  qed

```

proposition *homeomorphic_convex_compact_cball*:

```

  fixes e :: real
  and S :: 'a::euclidean_space set
  assumes S: convex S compact S interior S ≠ {} and e > 0
  shows S homeomorphic (cball (b::'a) e)
proof (rule homeomorphic_trans[OF _ homeomorphic_balls(2)])
  obtain a where a ∈ interior S

```



```

using assms by auto
then show S homeomorphic cball (0::'a) 1
  by (metis (no_types) aff_dim_cball S compact_cball convex_cball
      homeomorphic_convex_lemma interior_rel_interior_gen zero_less_one)
qed (use <e>0) in auto

```

corollary *homeomorphic_convex_compact*:

```

fixes S :: 'a::euclidean_space set
  and T :: 'a set
assumes convex S compact S interior S ≠ {}
  and convex T compact T interior T ≠ {}
shows S homeomorphic T
using assms
by (meson zero_less_one homeomorphic_trans homeomorphic_convex_compact_cball
    homeomorphic_sym)

```

lemma *homeomorphic_closed_intervals*:

```

fixes a :: 'a::euclidean_space and b and c :: 'a::euclidean_space and d
assumes box a b ≠ {} and box c d ≠ {}
shows (cbox a b) homeomorphic (cbox c d)
by (simp add: assms homeomorphic_convex_compact)

```

lemma *homeomorphic_closed_intervals_real*:

```

fixes a::real and b and c::real and d
assumes a<b and c<d
shows {a..b} homeomorphic {c..d}
using assms by (auto intro: homeomorphic_convex_compact)

```

9.8.4 Covering spaces and lifting results for them

definition *covering_space*

```

:: 'a::topological_space set ⇒ ('a ⇒ 'b) ⇒ 'b::topological_space set ⇒ bool

```

where

```

covering_space c p S ≡
  continuous_on c p ∧ p ' c = S ∧
  (∀ x ∈ S. ∃ T. x ∈ T ∧ openin (top_of_set S) T ∧
    (∃ v. ∪ v = c ∩ p - ' T ∧
      (∀ u ∈ v. openin (top_of_set c) u) ∧
      pairwise disjnt v ∧
      (∀ u ∈ v. ∃ q. homeomorphism u T p q)))

```

lemma *covering_spaceI* [*intro?*]:

```

assumes continuous_on c p p ' c = S
  ∧ x. x ∈ S ⇒ ∃ T. x ∈ T ∧ openin (top_of_set S) T ∧
    (∃ v. ∪ v = c ∩ p - ' T ∧ (∀ u ∈ v. openin (top_of_set c)
  u) ∧
    pairwise disjnt v ∧ (∀ u ∈ v. ∃ q. homeomorphism u T p q))
shows covering_space c p S
using assms unfolding covering_space_def by auto

```

lemma *covering_space_imp_continuous*: $\text{covering_space } c \ p \ S \implies \text{continuous_on } c \ p$
by (*simp add: covering_space_def*)

lemma *covering_space_imp_surjective*: $\text{covering_space } c \ p \ S \implies p \text{ ' } c = S$
by (*simp add: covering_space_def*)

lemma *homeomorphism_imp_covering_space*: $\text{homeomorphism } S \ T \ f \ g \implies \text{covering_space } S \ f \ T$
apply (*clarsimp simp add: homeomorphism_def covering_space_def*)
apply (*rule_tac x=T in exI, simp*)
apply (*rule_tac x={S} in exI, auto*)
done

lemma *covering_space_local_homeomorphism*:
assumes *covering_space* $c \ p \ S \ x \in c$
obtains $T \ u \ q$ **where** $x \in T \text{ openin } (\text{top_of_set } c) \ T$
 $p \ x \in u \text{ openin } (\text{top_of_set } S) \ u$
 $\text{homeomorphism } T \ u \ p \ q$
using *assms*
by (*clarsimp simp add: covering_space_def*) (*metis IntI UnionE vimage_eq*)

lemma *covering_space_local_homeomorphism_alt*:
assumes p : *covering_space* $c \ p \ S$ **and** $y \in S$
obtains $x \ T \ U \ q$ **where** $p \ x = y$
 $x \in T \text{ openin } (\text{top_of_set } c) \ T$
 $y \in U \text{ openin } (\text{top_of_set } S) \ U$
 $\text{homeomorphism } T \ U \ p \ q$
proof –
obtain x **where** $p \ x = y \ x \in c$
using *assms covering_space_imp_surjective* **by** *blast*
show *?thesis*
using *that* $\langle p \ x = y \rangle$ **by** (*auto intro: covering_space_local_homeomorphism*
 $[\text{OF } p \ \langle x \in c \rangle]$)
qed

proposition *covering_space_open_map*:
fixes $S :: 'a :: \text{metric_space set}$ **and** $T :: 'b :: \text{metric_space set}$
assumes p : *covering_space* $c \ p \ S$ **and** T : *openin* $(\text{top_of_set } c) \ T$
shows *openin* $(\text{top_of_set } S) \ (p \text{ ' } T)$

proof –
have pce : $p \text{ ' } c = S$
and $covs$:
 $\bigwedge x. x \in S \implies$
 $\exists X \ VS. x \in X \wedge \text{openin } (\text{top_of_set } S) \ X \wedge$
 $\bigcup VS = c \cap p \text{ ' } X \wedge$
 $(\forall u \in VS. \text{openin } (\text{top_of_set } c) \ u) \wedge$

```

      pairwise disjnt VS  $\wedge$ 
      ( $\forall u \in VS. \exists q. \text{homeomorphism } u \ X \ p \ q$ )
    using p by (auto simp: covering_space_def)
  have  $T \subseteq c$  by (metis openin_euclidean_subtopology_iff T)
  have  $\exists X. \text{openin } (\text{top\_of\_set } S) \ X \wedge y \in X \wedge X \subseteq p^{-1} T$ 
    if  $y \in p^{-1} T$  for y
  proof -
    have  $y \in S$  using  $\langle T \subseteq c \rangle$  pce that by blast
    obtain U VS where  $y \in U$  and  $U: \text{openin } (\text{top\_of\_set } S) \ U$ 
      and  $VS: \bigcup VS = c \cap p^{-1} U$ 
      and  $\text{openVS}: \forall V \in VS. \text{openin } (\text{top\_of\_set } c) \ V$ 
      and  $\text{homVS}: \bigwedge V. V \in VS \implies \exists q. \text{homeomorphism } V \ U \ p \ q$ 
    using covs [OF  $\langle y \in S \rangle$ ] by auto
    obtain x where  $x \in c \ p \ x \in U \ x \in T \ p \ x = y$ 
      using T [unfolded openin_euclidean_subtopology_iff]  $\langle y \in U \rangle \langle y \in p^{-1} T \rangle$ 
  by blast
  with VS obtain V where  $x \in V \ V \in VS$  by auto
  then obtain q where  $q: \text{homeomorphism } V \ U \ p \ q$  using homVS by blast
  then have ptV:  $p^{-1} (T \cap V) = U \cap q^{-1} (T \cap V)$ 
    using VS  $\langle V \in VS \rangle$  by (auto simp: homeomorphism_def)
  have ocv:  $\text{openin } (\text{top\_of\_set } c) \ V$ 
    by (simp add:  $\langle V \in VS \rangle$  openVS)
  have  $\text{openin } (\text{top\_of\_set } (q^{-1} U)) \ (T \cap V)$ 
    using q unfolding homeomorphism_def
  by (metis T inf.absorb_iff2 ocv openin_imp_subset openin_subtopology_Int
subtopology_subtopology)
  then have  $\text{openin } (\text{top\_of\_set } U) \ (U \cap q^{-1} (T \cap V))$ 
    using continuous_on_open homeomorphism_def q by blast
  then have os:  $\text{openin } (\text{top\_of\_set } S) \ (U \cap q^{-1} (T \cap V))$ 
    using openin_trans [of U] by (simp add: Collect_conj_eq U)
  show ?thesis
  proof (intro exI conjI)
    show  $\text{openin } (\text{top\_of\_set } S) \ (p^{-1} (T \cap V))$ 
      by (simp only: ptV os)
    qed (use  $\langle p \ x = y \rangle \langle x \in V \rangle \langle x \in T \rangle$  in auto)
  qed
  with openin_subopen show ?thesis by blast
qed

lemma covering_space_lift_unique_gen:
  fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
  fixes g1 :: 'a  $\Rightarrow$  'c::real_normed_vector
  assumes cov: covering_space c p S
    and eq: g1 a = g2 a
    and f: continuous_on T f f  $\in T \rightarrow S$ 
    and g1: continuous_on T g1 g1  $\in T \rightarrow c$ 
    and fg1:  $\bigwedge x. x \in T \implies f \ x = p(g1 \ x)$ 
    and g2: continuous_on T g2 g2  $\in T \rightarrow c$ 
    and fg2:  $\bigwedge x. x \in T \implies f \ x = p(g2 \ x)$ 

```

```

    and u_compt:  $U \in \text{components } T$  and  $a \in U$   $x \in U$ 
    shows  $g1\ x = g2\ x$ 
  proof -
    have  $U \subseteq T$  by (rule in_components_subset [OF u_compt])
    define G12 where  $G12 \equiv \{x \in U. g1\ x - g2\ x = 0\}$ 
    have connected  $U$  by (rule in_components_connected [OF u_compt])
    have contu: continuous_on  $U$  g1 continuous_on  $U$  g2
      using  $\langle U \subseteq T \rangle$  continuous_on_subset g1 g2 by blast+
    have o12: openin (top_of_set  $U$ ) G12
    unfolding G12_def
    proof (subst openin_subopen, clarify)
      fix  $z$ 
      assume  $z: z \in U$   $g1\ z - g2\ z = 0$ 
      obtain  $v\ w\ q$  where  $g1\ z \in v$  and ocv: openin (top_of_set  $c$ )  $v$ 
        and  $p$  ( $g1\ z$ )  $\in w$  and osw: openin (top_of_set  $S$ )  $w$ 
        and hom: homeomorphism  $v\ w\ p\ q$ 
      proof (rule covering_space_local_homeomorphism [OF cov])
        show  $g1\ z \in c$ 
          using  $\langle U \subseteq T \rangle$   $\langle z \in U \rangle$  g1(2) by blast
        qed auto
        have  $g2\ z \in v$  using  $\langle g1\ z \in v \rangle$   $z$  by auto
        have gg:  $U \cap g - ' v = U \cap g - ' (v \cap g - ' U)$  for  $g$ 
          by auto
        have openin (top_of_set ( $g1 - ' U$ )) ( $v \cap g1 - ' U$ )
          using ocv  $\langle U \subseteq T \rangle$  g1 by (fastforce simp add: openin_open)
        then have 1: openin (top_of_set  $U$ ) ( $U \cap g1 - ' v$ )
          unfolding gg by (blast intro: contu continuous_on_open [THEN iffD1,
            rule_format])
        have openin (top_of_set ( $g2 - ' U$ )) ( $v \cap g2 - ' U$ )
          using ocv  $\langle U \subseteq T \rangle$  g2 by (fastforce simp add: openin_open)
        then have 2: openin (top_of_set  $U$ ) ( $U \cap g2 - ' v$ )
          unfolding gg by (blast intro: contu continuous_on_open [THEN iffD1,
            rule_format])
        let  $?T = (U \cap g1 - ' v) \cap (U \cap g2 - ' v)$ 
        show  $\exists T. \text{openin} (\text{top\_of\_set } U) T \wedge z \in T \wedge T \subseteq \{z \in U. g1\ z - g2\ z = 0\}$ 
          proof (intro exI conjI)
            show openin (top_of_set  $U$ )  $?T$ 
              using 1 2 by blast
            show  $z \in ?T$ 
              using  $z$  by (simp add:  $\langle g1\ z \in v \rangle$   $\langle g2\ z \in v \rangle$ )
            show  $?T \subseteq \{z \in U. g1\ z - g2\ z = 0\}$ 
              using hom
              by (clarsimp simp: homeomorphism_def) (metis  $\langle U \subseteq T \rangle$  fg1 fg2 subsetD)
          qed
        qed
      have c12: closedin (top_of_set  $U$ ) G12
      unfolding G12_def
      by (intro continuous_intros continuous_closedin_preimage_constant contu)

```

```

have  $G12 = \{\} \vee G12 = U$ 
  by (intro connected_clopen [THEN iffD1, rule_format] <connected U> conjI
o12 c12)
  with eq <a ∈ U> have  $\bigwedge x. x \in U \implies g1\ x - g2\ x = 0$  by (auto simp: G12_def)
  then show ?thesis
    using <x ∈ U> by force
qed

```

proposition *covering_space_lift_unique:*

```

fixes f :: 'a::topological_space  $\Rightarrow$  'b::topological_space
fixes g1 :: 'a  $\Rightarrow$  'c::real_normed_vector
assumes covering_space c p S
  g1 a = g2 a
  continuous_on T f f ∈ T  $\rightarrow$  S
  continuous_on T g1 g1 ∈ T  $\rightarrow$  c  $\bigwedge x. x \in T \implies f\ x = p(g1\ x)$ 
  continuous_on T g2 g2 ∈ T  $\rightarrow$  c  $\bigwedge x. x \in T \implies f\ x = p(g2\ x)$ 
  connected T a ∈ T x ∈ T
shows g1 x = g2 x
using covering_space_lift_unique_gen [of c p S] in_components_self assms
ex_in_conv
by blast

```

lemma *covering_space_locally:*

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
assumes loc: locally  $\varphi\ C$  and cov: covering_space C p S
  and pim:  $\bigwedge T. \llbracket T \subseteq C; \varphi\ T \rrbracket \implies \psi(p\ ' T)$ 
shows locally  $\psi\ S$ 
proof -
  have locally  $\psi\ (p\ ' C)$ 
proof (rule locally_open_map_image [OF loc])
  show continuous_on C p
    using cov covering_space_imp_continuous by blast
  show  $\bigwedge T. \text{openin}\ (top\_of\_set\ C)\ T \implies \text{openin}\ (top\_of\_set\ (p\ ' C))\ (p\ ' T)$ 
    using cov covering_space_imp_surjective covering_space_open_map by blast
qed (simp add: pim)
then show ?thesis
  using covering_space_imp_surjective [OF cov] by metis
qed

```

proposition *covering_space_locally_eq:*

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
assumes cov: covering_space C p S
  and pim:  $\bigwedge T. \llbracket T \subseteq C; \varphi\ T \rrbracket \implies \psi(p\ ' T)$ 
  and qim:  $\bigwedge q\ U. \llbracket U \subseteq S; \text{continuous\_on}\ U\ q; \psi\ U \rrbracket \implies \varphi(q\ ' U)$ 
shows locally  $\psi\ S \longleftrightarrow$  locally  $\varphi\ C$ 
  (is ?lhs = ?rhs)

```

proof

```

assume L: ?lhs

```

```

show ?rhs
proof (rule locallyI)
  fix V x
  assume V: openin (top_of_set C) V and x ∈ V
  have p x ∈ p ' C
    by (metis IntE V ⟨x ∈ V⟩ imageI openin_open)
  then obtain T V where p x ∈ T
    and opeT: openin (top_of_set S) T
    and veq:  $\bigcup V = C \cap p -' T$ 
    and ope:  $\forall U \in V. \text{openin } (top\_of\_set C) U$ 
    and hom:  $\forall U \in V. \exists q. \text{homeomorphism } U T p q$ 
  using cov unfolding covering_space_def by (blast intro: that)
  have x ∈  $\bigcup V$ 
    using V veq ⟨p x ∈ T⟩ ⟨x ∈ V⟩ openin_imp_subset by fastforce
  then obtain U where x ∈ U U ∈ V
    by blast
  then obtain q where opeU: openin (top_of_set C) U and q: homeomorphism
    U T p q
    using ope hom by blast
  with V have openin (top_of_set C) (U ∩ V)
    by blast
  then have UV: openin (top_of_set S) (p '(U ∩ V))
    using cov covering_space_open_map by blast
  obtain W W' where opeW: openin (top_of_set S) W and ψ W' p x ∈ W
    W ⊆ W' and W'sub: W' ⊆ p '(U ∩ V)
    using locallyE [OF L UV] ⟨x ∈ U⟩ ⟨x ∈ V⟩ by blast
  then have W ⊆ T
    by (metis Int_lower1 q homeomorphism_image1 image_Int_subset order_trans)
  show  $\exists U Z. \text{openin } (top\_of\_set C) U \wedge$ 
     $\varphi Z \wedge x \in U \wedge U \subseteq Z \wedge Z \subseteq V$ 
  proof (intro exI conjI)
    have openin (top_of_set T) W
      by (meson opeW opeT openin_imp_subset openin_subset_trans ⟨W ⊆ T⟩)
    then have openin (top_of_set U) (q ' W)
      by (meson homeomorphism_imp_open_map homeomorphism_symD q)
    then show openin (top_of_set C) (q ' W)
      using opeU openin_trans by blast
    show  $\varphi (q ' W')$ 
      by (metis (mono_tags, lifting) Int_subset_iff UV W'sub ⟨ψ W'⟩ continu-
        ous_on_subset dual_order.trans homeomorphism_def image_Int_subset openin_imp_subset
        q qim)
    show x ∈ q ' W
      by (metis ⟨p x ∈ W⟩ ⟨x ∈ U⟩ homeomorphism_def imageI q)
    show q ' W ⊆ q ' W'
      using ⟨W ⊆ W'⟩ by blast
    have W' ⊆ p ' V
      using W'sub by blast
    then show q ' W' ⊆ V
      using W'sub homeomorphism_apply1 [OF q] by auto
  
```

```

      qed
    qed
  next
    assume ?rhs
    then show ?lhs
      using cov covering_space_locally_pim by blast
    qed

lemma covering_space_locally_compact_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_compact S  $\longleftrightarrow$  locally_compact C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{compact } T \rrbracket \Longrightarrow \text{compact } (p \text{ ' } T)$ 
    by (meson assms compact_continuous_image continuous_on_subset covering_space_imp_continuous)
qed (use compact_continuous_image in blast)

lemma covering_space_locally_connected_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_connected S  $\longleftrightarrow$  locally_connected C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{connected } T \rrbracket \Longrightarrow \text{connected } (p \text{ ' } T)$ 
    by (meson connected_continuous_image assms continuous_on_subset covering_space_imp_continuous)
qed (use connected_continuous_image in blast)

lemma covering_space_locally_path_connected_eq:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes covering_space C p S
  shows locally_path_connected S  $\longleftrightarrow$  locally_path_connected C
proof (rule covering_space_locally_eq [OF assms])
  show  $\bigwedge T. \llbracket T \subseteq C; \text{path\_connected } T \rrbracket \Longrightarrow \text{path\_connected } (p \text{ ' } T)$ 
    by (meson path_connected_continuous_image assms continuous_on_subset covering_space_imp_continuous)
qed (use path_connected_continuous_image in blast)

lemma covering_space_locally_compact:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes locally_compact C covering_space C p S
  shows locally_compact S
  using assms covering_space_locally_compact_eq by blast

lemma covering_space_locally_connected:
  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes locally_connected C covering_space C p S

```

shows *locally connected S*
 using *assms covering_space_locally_connected_eq* by *blast*

lemma *covering_space_locally_path_connected*:
 fixes $p :: 'a::real_normed_vector \Rightarrow 'b::real_normed_vector$
 assumes *locally_path_connected C covering_space C p S*
 shows *locally_path_connected S*
 using *assms covering_space_locally_path_connected_eq* by *blast*

proposition *covering_space_lift_homotopy*:
 fixes $p :: 'a::real_normed_vector \Rightarrow 'b::real_normed_vector$
 and $h :: real \times 'c::real_normed_vector \Rightarrow 'b$
 assumes *cov: covering_space C p S*
 and *conth: continuous_on* $(\{0..1\} \times U)$ h
 and *him: $h \in (\{0..1\} \times U) \rightarrow S$*
 and *heq: $\bigwedge y. y \in U \implies h(0, y) = p(f y)$*
 and *conf: continuous_on U f* and *fim: $f \in U \rightarrow C$*
 obtains k where *continuous_on* $(\{0..1\} \times U)$ k
 $k \in (\{0..1\} \times U) \rightarrow C$
 $\bigwedge y. y \in U \implies k(0, y) = f y$
 $\bigwedge z. z \in \{0..1\} \times U \implies h z = p(k z)$

proof –

have $\exists V k. \text{openin } (\text{top_of_set } U) V \wedge y \in V \wedge$
 $\text{continuous_on } (\{0..1\} \times V) k \wedge k' (\{0..1\} \times V) \subseteq C \wedge$
 $(\forall z \in V. k(0, z) = f z) \wedge (\forall z \in \{0..1\} \times V. h z = p(k z))$
 if $y \in U$ for y

proof –

obtain UU where $UU: \bigwedge s. s \in S \implies s \in (UU s) \wedge \text{openin } (\text{top_of_set } S)$
 $(UU s) \wedge$

$$(\exists \mathcal{V}. \bigcup \mathcal{V} = C \cap p - ' UU s \wedge$$

$$(\forall U \in \mathcal{V}. \text{openin } (\text{top_of_set } C) U) \wedge$$

$$\text{pairwise disjoint } \mathcal{V} \wedge$$

$$(\forall U \in \mathcal{V}. \exists q. \text{homeomorphism } U (UU s) p q))$$

using *cov unfolding covering_space_def* by *(metis (mono_tags))*

then have *ope: $\bigwedge s. s \in S \implies s \in (UU s) \wedge \text{openin } (\text{top_of_set } S) (UU s)$*
 by *blast*

have $\exists k n i. \text{open } k \wedge \text{open } n \wedge$

$t \in k \wedge y \in n \wedge i \in S \wedge h' ((\{0..1\} \cap k) \times (U \cap n)) \subseteq UU i$ if t
 $\in \{0..1\}$ for t

proof –

have *hinS: $h(t, y) \in S$*

using $\langle y \in U \rangle$ *him that* by *blast*

then have $(t, y) \in (\{0..1\} \times U) \cap h - ' UU(h(t, y))$

using $\langle y \in U \rangle \langle t \in \{0..1\} \rangle$ by *(auto simp: ope)*

moreover have *ope_01U: openin* $(\text{top_of_set } (\{0..1\} \times U)) ((\{0..1\} \times U)$
 $\cap h - ' UU(h(t, y)))$

using *hinS ope continuous_on_open_gen [OF him] conth* by *blast*

ultimately obtain $V W$ where *opeV: open V* and $t \in \{0..1\} \cap V$ $t \in$
 $\{0..1\} \cap V$


```

and opeW: open W and y ∈ U y ∈ W
and VW: ({0..1} ∩ V) × (U ∩ W) ⊆ (({0..1} × U) ∩
h - ' UU(h(t, y))
  by (rule Times_in_interior_subtopology) (auto simp: openin_open)
  then show ?thesis
  using hinS by blast
qed
then obtain K NN X where
  K:  $\bigwedge t. t \in \{0..1\} \implies \text{open } (K t)$ 
  and NN:  $\bigwedge t. t \in \{0..1\} \implies \text{open } (NN t)$ 
  and inUS:  $\bigwedge t. t \in \{0..1\} \implies t \in K t \wedge y \in NN t \wedge X t \in S$ 
  and him:  $\bigwedge t. t \in \{0..1\} \implies h ' ((\{0..1\} \cap K t) \times (U \cap NN t)) \subseteq UU$ 
(X t)
  by (metis (mono_tags))
obtain  $\mathcal{T}$  where  $\mathcal{T} \subseteq ((\lambda i. K i \times NN i) ' \{0..1\} \text{ finite } \mathcal{T} \{0::\text{real}..1\} \times \{y\}$ 
 $\subseteq \bigcup \mathcal{T}$ 
proof (rule compactE)
  show compact ( $\{0::\text{real}..1\} \times \{y\}$ )
  by (simp add: compact_Times)
  show  $\{0..1\} \times \{y\} \subseteq (\bigcup i \in \{0..1\}. K i \times NN i)$ 
  using K inUS by auto
  show  $\bigwedge B. B \in (\lambda i. K i \times NN i) ' \{0..1\} \implies \text{open } B$ 
  using K NN by (auto simp: open_Times)
qed blast
then obtain tk where  $tk \subseteq \{0..1\} \text{ finite } tk$ 
  and  $tk: \{0::\text{real}..1\} \times \{y\} \subseteq (\bigcup i \in tk. K i \times NN i)$ 
  by (metis (no_types, lifting) finite_subset_image)
then have  $tk \neq \{\}$ 
  by auto
define n where  $n = \bigcap (NN ' tk)$ 
have  $y \in n$  open n
  using inUS NN  $\langle tk \subseteq \{0..1\} \rangle \langle \text{finite } tk \rangle$ 
  by (auto simp: n_def open_INT subset_iff)
obtain  $\delta$  where  $0 < \delta$  and  $\delta: \bigwedge T. \llbracket T \subseteq \{0..1\}; \text{diameter } T < \delta \rrbracket \implies \exists B \in K$ 
 $' tk. T \subseteq B$ 
proof (rule Lebesgue_number_lemma [of  $\{0..1\}$  K ' tk])
  show  $K ' tk \neq \{\}$ 
  using  $\langle tk \neq \{\} \rangle$  by auto
  show  $\{0..1\} \subseteq \bigcup (K ' tk)$ 
  using tk by auto
  show  $\bigwedge B. B \in K ' tk \implies \text{open } B$ 
  using  $\langle tk \subseteq \{0..1\} \rangle$  K by auto
qed auto
obtain N::nat where  $N: N > 1 / \delta$ 
  using reals_Archimedean2 by blast
then have  $N > 0$ 
  using  $\langle 0 < \delta \rangle$  order.asym by force
have *:  $\exists V k. \text{openin } (\text{top\_of\_set } U) V \wedge y \in V \wedge$ 
  continuous_on ( $\{0..of\_nat n / N\} \times V$ ) k  $\wedge$ 

```

```

      k ' ({0..of_nat n / N} × V) ⊆ C ∧
      (∀ z ∈ V. k (0, z) = f z) ∧
      (∀ z ∈ {0..of_nat n / N} × V. h z = p (k z)) if n ≤ N for n
    using that
  proof (induction n)
  case 0
  show ?case
    apply (rule_tac x=U in exI)
    apply (rule_tac x=f ∘ snd in exI)
    apply (intro conjI ⟨y ∈ U⟩ continuous_intros continuous_on_subset [OF
contf])
    using fim apply (auto simp: heq)
  done
  next
  case (Suc n)
  then obtain V k where opeUV: openin (top_of_set U) V
    and y ∈ V
    and contk: continuous_on ({0..n/N} × V) k
    and kim: k ' ({0..n/N} × V) ⊆ C
    and keq: ∧z. z ∈ V ⇒ k (0, z) = f z
    and heq: ∧z. z ∈ {0..n/N} × V ⇒ h z = p (k z)
    using Suc_leD by auto
  have n ≤ N
    using Suc.prem by auto
  obtain t where t ∈ tk and t: {n/N .. (1 + real n) / N} ⊆ K t
  proof (rule bexE [OF δ])
    show {n/N .. (1 + real n) / N} ⊆ {0..1}
      using Suc.prem by (auto simp: field_split_simps)
    show diameter_less: diameter {n/N .. (1 + real n) / N} < δ
      using ⟨0 < δ⟩ N by (auto simp: field_split_simps)
  qed blast
  have t01: t ∈ {0..1}
    using ⟨t ∈ tk⟩ ⟨tk ⊆ {0..1}⟩ by blast
  obtain V where V: ⋃V = C ∩ p -' UU (X t)
    and opeC: ∧U. U ∈ V ⇒ openin (top_of_set C) U
    and pairwise disjnt V
    and homuu: ∧U. U ∈ V ⇒ ∃ q. homeomorphism U (UU (X t)) p q
    using inUS [OF t01] UU by meson
  have n_div_N_in: n/N ∈ {n/N .. (1 + real n) / N}
    using N by (auto simp: field_split_simps)
  with t have nN_in_kkt: n/N ∈ K t
    by blast
  have k (n/N, y) ∈ C ∩ p -' UU (X t)
  proof (simp, rule conjI)
    show k (n/N, y) ∈ C
      using ⟨y ∈ V⟩ kim keq by force
    have p (k (n/N, y)) = h (n/N, y)
      by (simp add: ⟨y ∈ V⟩ heq)
    also have ... ∈ h ' (({0..1} ∩ K t) × (U ∩ NN t))

```

```

    using ⟨y ∈ V⟩ t01 ⟨n ≤ N⟩
    by (simp add: nN_in_kkt ⟨y ∈ U⟩ inUS field_split_simps)
  also have ... ⊆ UU (X t)
    using him t01 by blast
  finally show p (k (n/N, y)) ∈ UU (X t) .
qed
with V have k (n/N, y) ∈ ⋃V
  by blast
then obtain W where W: k (n/N, y) ∈ W and W ∈ V
  by blast
then obtain p' where opeC': openin (top_of_set C) W
  and hom': homeomorphism W (UU (X t)) p p'
  using homuu opeC by blast
then have W ⊆ C
  using openin_imp_subset by blast
define W' where W' = UU(X t)
have opeVW: openin (top_of_set V) (V ∩ (k ∘ Pair (n / N)) -' W)
proof (rule continuous_openin_preimage [OF _ _ opeC'])
  show continuous_on V (k ∘ Pair (n/N))
    by (intro continuous_intros continuous_on_subset [OF contk], auto)
  show (k ∘ Pair (n/N)) ∈ V → C
    using kim by (auto simp: ⟨y ∈ V⟩ W)
qed
obtain N' where opeUN': openin (top_of_set U) N'
  and y ∈ N' and kimw: k ' ({(n/N)} × N') ⊆ W
proof
  show openin (top_of_set U) (V ∩ (k ∘ Pair (n/N)) -' W)
    using opeUV opeVW openin_trans by blast
qed (use ⟨y ∈ V⟩ W in ⟨force+⟩)
obtain Q Q' where opeUQ: openin (top_of_set U) Q
  and cloUQ': closedin (top_of_set U) Q'
  and y ∈ Q Q ⊆ Q'
  and Q': Q' ⊆ (U ∩ NN(t)) ∩ N' ∩ V
proof -
  obtain VO VX where open VO open VX and VO: V = U ∩ VO and
  VX: N' = U ∩ VX
    using opeUV opeUN' by (auto simp: openin_open)
  then have open (NN(t) ∩ VO ∩ VX)
    using NN t01 by blast
  then obtain e where e > 0 and e: cball y e ⊆ NN(t) ∩ VO ∩ VX
    by (metis Int_iff ⟨N' = U ∩ VX⟩ ⟨V = U ∩ VO⟩ ⟨y ∈ N'⟩ ⟨y ∈ V⟩ inUS
  open_contains_cball t01)
  show ?thesis
proof
  show openin (top_of_set U) (U ∩ ball y e)
    by blast
  show closedin (top_of_set U) (U ∩ cball y e)
    using e by (auto simp: closedin_closed)
qed (use ⟨y ∈ U⟩ ⟨e > 0⟩ VO VX e in auto)

```

```

qed
then have  $y \in Q' \ Q \subseteq (U \cap NN(t)) \cap N' \cap V$ 
  by blast+
have  $neg: \{0..n/N\} \cup \{n/N..(1 + real\ n) / N\} = \{0..(1 + real\ n) / N\}$ 
  apply (auto simp: field_split_simps)
by (metis not_less_of_nat_0_le_iff_of_nat_0_less_iff order_trans zero_le_mult_iff)
then have  $negQ': \{0..n/N\} \times Q' \cup \{n/N..(1 + real\ n) / N\} \times Q' = \{0..(1 + real\ n) / N\} \times Q'$ 
+  $real\ n) / N\} \times Q'$ 
  by blast
have  $cont: continuous\_on (\{0..(1 + real\ n) / N\} \times Q') (\lambda x. if\ x \in \{0..n/N\} \times Q' then\ k\ x\ else\ (p' \circ h)\ x)$ 
  unfolding  $negQ'$  [symmetric]
proof (rule continuous_on_cases_local, simp_all add:  $negQ'$  del: comp_apply)
  have  $\exists T. closed\ T \wedge \{0..n/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$ 
    using  $n\_div\_N\_in$ 
  by (rule_tac  $x = \{0..n/N\} \times UNIV$  in  $exI$ ) (auto simp: closed_Times)
  then show  $closedin\ (top\_of\_set\ (\{0..(1 + real\ n) / N\} \times Q')) (\{0..n/N\} \times Q')$ 
   $\times Q'$ 
    by (simp add: closedin_closed)
  have  $\exists T. closed\ T \wedge \{n/N..(1+n)/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$ 
  by (rule_tac  $x = \{n/N..(1+n)/N\} \times UNIV$  in  $exI$ ) (auto simp: closed_Times
order_trans [rotated])
  then show  $closedin\ (top\_of\_set\ (\{0..(1 + real\ n) / N\} \times Q')) (\{n/N..(1 + real\ n) / N\} \times Q')$ 
+  $real\ n) / N\} \times Q'$ 
    by (simp add: closedin_closed)
  show  $continuous\_on (\{0..n/N\} \times Q') k$ 
    using  $Q'$  by (auto intro: continuous_on_subset [OF contk])
  have  $continuous\_on (\{n/N..(1 + real\ n) / N\} \times Q') h$ 
  proof (rule continuous_on_subset [OF conth])
    show  $\{n/N..(1 + real\ n) / N\} \times Q' \subseteq \{0..1\} \times U$ 
    proof (clarsimp, intro conjI)
      fix  $a\ b$ 
      assume  $b \in Q'$  and  $a: n/N \leq a \leq (1 + real\ n) / N$ 
      have  $0 \leq n/N \ (1 + real\ n) / N \leq 1$ 
        using  $a\_Suc.prem$  by (auto simp: divide_simps)
      with  $a$  show  $0 \leq a \leq 1$ 
        by linarith+
      show  $b \in U$ 
        using  $\langle b \in Q' \rangle\ cloUQ'\ closedin\_imp\_subset$  by blast
    qed
  qed
qed
qed
moreover have  $continuous\_on (h \ ' (\{n/N..(1 + real\ n) / N\} \times Q')) p'$ 
proof (rule continuous_on_subset [OF homeomorphism_cont2 [OF hom]])
  have  $h \ ' (\{n/N..(1 + real\ n) / N\} \times Q') \subseteq h \ ' ((\{0..1\} \cap K\ t) \times (U \cap NN\ t))$ 
  proof (rule image_mono)
    show  $\{n/N..(1 + real\ n) / N\} \times Q' \subseteq (\{0..1\} \cap K\ t) \times (U \cap NN\ t)$ 
    proof (clarsimp, intro conjI)
      fix  $a::real$  and  $b$ 

```

```

    assume  $b \in Q'$   $n/N \leq a$   $a \leq (1 + \text{real } n) / N$ 
    show  $0 \leq a$ 
      by (meson  $\langle n/N \leq a \rangle$  divide_nonneg_nonneg_of_nat_0_le_iff
order_trans)
    show  $a \leq 1$ 
      using Suc.premis  $\langle a \leq (1 + \text{real } n) / N \rangle$  order_trans by force
    show  $a \in K$  t
      using  $\langle a \leq (1 + \text{real } n) / N \rangle$   $\langle n/N \leq a \rangle$  t by auto
    show  $b \in U$ 
      using  $\langle b \in Q' \rangle$  cloUQ' closedin_imp_subset by blast
    show  $b \in NN$  t
      using  $Q'$   $\langle b \in Q' \rangle$  by auto
  qed
qed
with him show  $h' (\{n/N..(1 + \text{real } n) / N\} \times Q') \subseteq UU$  (X t)
  using t01 by blast
qed
ultimately show continuous_on ( $\{n/N..(1 + \text{real } n) / N\} \times Q'$ ) ( $p' \circ h$ )
  by (rule continuous_on_compose)
have  $k (n/N, b) = p' (h (n/N, b))$  if  $b \in Q'$  for b
proof -
  have  $k (n/N, b) \in W$ 
    using that  $Q'$  kimw by force
  then have  $k (n/N, b) = p' (p (k (n/N, b)))$ 
    by (simp add: homeomorphism_apply1 [OF hom])
  then show ?thesis
    using  $Q'$  that by (force simp: heq)
qed
then show  $\bigwedge x. x \in \{n/N..(1 + \text{real } n) / N\} \times Q' \wedge$ 
 $x \in \{0..n/N\} \times Q' \implies k x = (p' \circ h) x$ 
  by auto
qed
have  $h\_in\_UU: h (x, y) \in UU$  (X t) if  $y \in Q \neg x \leq n/N$   $0 \leq x$   $x \leq (1 +$ 
 $\text{real } n) / N$  for x y
proof -
  have  $x \leq 1$ 
    using Suc.premis that order_trans by force
  moreover have  $x \in K$  t
    by (meson atLeastAtMost_iff le_less not_le subset_eq t that)
  moreover have  $y \in U$ 
    using  $\langle y \in Q \rangle$  opeUQ openin_imp_subset by blast
  moreover have  $y \in NN$  t
    using  $Q'$   $\langle Q \subseteq Q' \rangle$   $\langle y \in Q \rangle$  by auto
  ultimately have  $(x, y) \in ((\{0..1\} \cap K t) \times (U \cap NN t))$ 
    using that by auto
  then have  $h (x, y) \in h' ((\{0..1\} \cap K t) \times (U \cap NN t))$ 
    by blast
  also have  $\dots \subseteq UU$  (X t)
    by (metis him t01)

```

```

    finally show ?thesis .
  qed
  let ?k = (λx. if x ∈ {0..n/N} × Q' then k x else (p' ∘ h) x)
  show ?case
  proof (intro exI conjI)
    show continuous_on ({0..real (Suc n) / N} × Q) ?k
      using ⟨Q ⊆ Q'⟩ by (auto intro: continuous_on_subset [OF cont])
    have ∧x y. [x ≤ n/N; y ∈ Q'; 0 ≤ x] ⇒ k (x, y) ∈ C
      using kim Q' by force
    moreover have p' (h (x, y)) ∈ C if y ∈ Q ∧ x ≤ n/N 0 ≤ x x ≤ (1 +
real n) / N for x y
    proof (rule ⟨W ⊆ C⟩ [THEN subsetD])
      show p' (h (x, y)) ∈ W
        using homeomorphism_image2 [OF hom', symmetric] h_in_UU Q'
⟨Q ⊆ Q'⟩ ⟨W ⊆ C⟩ that by auto
    qed
    ultimately show ?k ' ({0..real (Suc n) / N} × Q) ⊆ C
      using Q' ⟨Q ⊆ Q'⟩ by force
    show ∀z ∈ Q. ?k (0, z) = f z
      using Q' keq ⟨Q ⊆ Q'⟩ by auto
    show ∀z ∈ {0..real (Suc n) / N} × Q. h z = p(?k z)
      using ⟨Q ⊆ U ∩ NN t ∩ N' ∩ V⟩ heq Q' ⟨Q ⊆ Q'⟩
      by (auto simp: homeomorphism_apply2 [OF hom'] dest: h_in_UU)
    qed (auto simp: ⟨y ∈ Q⟩ opeUQ)
  qed
  show ?thesis
    using *[OF order_refl] N ⟨0 < δ⟩ by (simp add: split: if_split_asm)
  qed
  then obtain V fs where opeV: ∧y. y ∈ U ⇒ openin (top_of_set U) (V y)
    and V: ∧y. y ∈ U ⇒ y ∈ V y
    and contfs: ∧y. y ∈ U ⇒ continuous_on ({0..1} × V y) (fs y)
    and *: ∧y. y ∈ U ⇒ (fs y) ' ({0..1} × V y) ⊆ C ∧
      (∀z ∈ V y. fs y (0, z) = f z) ∧
      (∀z ∈ {0..1} × V y. h z = p(fs y z))
    by (metis (mono_tags))
  then have VU: ∧y. y ∈ U ⇒ V y ⊆ U
    by (meson openin_imp_subset)
  obtain k where contk: continuous_on ({0..1} × U) k
    and k: ∧x i. [i ∈ U; x ∈ {0..1} × U ∩ {0..1} × V i] ⇒ k x = fs i x
  proof (rule pasting_lemma_exists)
    let ?X = top_of_set ({0..1::real} × U)
    show topspace ?X ⊆ (∪i ∈ U. {0..1} × V i)
      using V by force
    show ∧i. i ∈ U ⇒ openin (top_of_set ({0..1} × U)) ({0..1} × V i)
      by (simp add: Abstract_Topology.openin_Times opeV)
    show ∧i. i ∈ U ⇒ continuous_map
      (subtopology (top_of_set ({0..1} × U)) ({0..1} × V i)) euclidean (fs
i)
      by (metis contfs subtopology_subtopology continuous_map_iff_continuous

```

```

Times_Int_Times VU inf.absorb_iff2 inf.idem)
  show fs i x = fs j x if i ∈ U j ∈ U and x: x ∈ topspace ?X ∩ {0..1} × V i
  ∩ {0..1} × V j
    for i j x
  proof -
    obtain u y where x = (u, y) y ∈ V i y ∈ V j 0 ≤ u u ≤ 1
    using x by auto
    show ?thesis
  proof (rule covering_space_lift_unique [OF cov, of_ (0,y) _ {0..1} × {y}
h])
    show fs i (0, y) = fs j (0, y)
      using * V by (simp add: ⟨y ∈ V i⟩ ⟨y ∈ V j⟩ that)
    show conth_y: continuous_on ({0..1} × {y}) h
      using VU ⟨y ∈ V j⟩ that by (auto intro: continuous_on_subset [OF
conth])
    show h ∈ ({0..1} × {y}) → S
      using ⟨y ∈ V i⟩ assms(3) VU that by fastforce
    show continuous_on ({0..1} × {y}) (fs i)
      using continuous_on_subset [OF contfs] ⟨i ∈ U⟩
      by (simp add: ⟨y ∈ V i⟩ subset_iff)
    show fs i ∈ ({0..1} × {y}) → C
      using * ⟨y ∈ V i⟩ ⟨i ∈ U⟩ by fastforce
    show ∧x. x ∈ {0..1} × {y} ⇒ h x = p (fs i x)
      using * ⟨y ∈ V i⟩ ⟨i ∈ U⟩ by blast
    show continuous_on ({0..1} × {y}) (fs j)
      using continuous_on_subset [OF contfs] ⟨j ∈ U⟩
      by (simp add: ⟨y ∈ V j⟩ subset_iff)
    show fs j ∈ ({0..1} × {y}) → C
      using * ⟨y ∈ V j⟩ ⟨j ∈ U⟩ by fastforce
    show ∧x. x ∈ {0..1} × {y} ⇒ h x = p (fs j x)
      using * ⟨y ∈ V j⟩ ⟨j ∈ U⟩ by blast
    show connected ({0..1::real} × {y})
      using connected_Icc connected_Times connected_sing by blast
    show (0, y) ∈ {0..1::real} × {y}
      by force
    show x ∈ {0..1} × {y}
      using ⟨x = (u, y)⟩ x by blast
  qed
qed
qed force
show ?thesis
proof
  show k ∈ ({0..1} × U) → C
    using V*k VU by fastforce
  show ∧y. y ∈ U ⇒ k (0, y) = f y
    by (simp add: V*k)
  show ∧z. z ∈ {0..1} × U ⇒ h z = p (k z)
    using V*k by auto
qed (auto simp: contk)

```

qed

corollary *covering_space_lift_homotopy_alt:*

fixes $p :: 'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$
and $h :: 'c::\text{real_normed_vector} \times \text{real} \Rightarrow 'b$

assumes $\text{cov}: \text{covering_space } C \text{ } p \text{ } S$

and $\text{conth}: \text{continuous_on } (U \times \{0..1\}) \text{ } h$

and $\text{him}: h \in (U \times \{0..1\}) \rightarrow S$

and $\text{heq}: \bigwedge y. y \in U \implies h(y, 0) = p(f y)$

and $\text{contf}: \text{continuous_on } U \text{ } f$ **and** $\text{fim}: f \in U \rightarrow C$

obtains k **where** $\text{continuous_on } (U \times \{0..1\}) \text{ } k$

$k \in (U \times \{0..1\}) \rightarrow C$

$\bigwedge y. y \in U \implies k(y, 0) = f y$

$\bigwedge z. z \in U \times \{0..1\} \implies h z = p(k z)$

proof –

have $\text{continuous_on } (\{0..1\} \times U) (h \circ (\lambda z. (\text{snd } z, \text{fst } z)))$

by (*intro continuous_intros continuous_on_subset [OF conth]*) *auto*

then obtain k **where** $\text{contk}: \text{continuous_on } (\{0..1\} \times U) \text{ } k$

and $\text{kim}: k \text{ ' } (\{0..1\} \times U) \subseteq C$

and $\text{k0}: \bigwedge y. y \in U \implies k(0, y) = f y$

and $\text{heqp}: \bigwedge z. z \in \{0..1\} \times U \implies (h \circ (\lambda z. \text{Pair } (\text{snd } z) (\text{fst } z)))$

$z = p(k z)$

apply (*rule covering_space_lift_homotopy [OF cov _ _ _ contf fim]*)

using him **by** (*auto simp: contf heq*)

show *?thesis*

proof

show $\text{continuous_on } (U \times \{0..1\}) (k \circ (\lambda z. (\text{snd } z, \text{fst } z)))$

by (*intro continuous_intros continuous_on_subset [OF contk]*) *auto*

qed (*use kim heqp in <auto simp: k0>*)

qed

corollary *covering_space_lift_homotopic_function:*

fixes $p :: 'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$ **and** $g :: 'c::\text{real_normed_vector} \Rightarrow 'a$

assumes $\text{cov}: \text{covering_space } C \text{ } p \text{ } S$

and $\text{contg}: \text{continuous_on } U \text{ } g$

and $\text{gim}: g \in U \rightarrow C$

and $\text{pgeq}: \bigwedge y. y \in U \implies p(g y) = f y$

and $\text{hom}: \text{homotopic_with_canon } (\lambda x. \text{True}) \text{ } U \text{ } S \text{ } f \text{ } f'$

obtains g' **where** $\text{continuous_on } U \text{ } g' \text{ image } g' \text{ } U \subseteq C \bigwedge y. y \in U \implies p(g' y) = f' y$

proof –

obtain h **where** $\text{conth}: \text{continuous_on } (\{0..1::\text{real}\} \times U) \text{ } h$

and $\text{him}: h \in (\{0..1\} \times U) \rightarrow S$

and $\text{h0}: \bigwedge x. h(0, x) = f x$

and $\text{h1}: \bigwedge x. h(1, x) = f' x$

using hom **by** (*auto simp: homotopic_with_def*)

have $\bigwedge y. y \in U \implies h(0, y) = p(g y)$

by (*simp add: h0 pgeq*)


```

then obtain  $k$  where  $contk: continuous\_on (\{0..1\} \times U) k$ 
  and  $kim: k \text{ ' } (\{0..1\} \times U) \subseteq C$ 
  and  $k0: \bigwedge y. y \in U \implies k(0, y) = g y$ 
  and  $heq: \bigwedge z. z \in \{0..1\} \times U \implies h z = p(k z)$ 
  using  $covering\_space\_lift\_homotopy [OF cov conth him\_contg gim]$  by ( $metis$ 
 $image\_subset\_iff\_funcset$ )
  show  $?thesis$ 
  proof
    show  $continuous\_on U (k \circ Pair 1)$ 
    by ( $meson contk atLeastAtMost\_iff continuous\_on\_o\_Pair order\_refl zero\_le\_one$ )
    show  $(k \circ Pair 1) \text{ ' } U \subseteq C$ 
    using  $kim$  by  $auto$ 
    show  $\bigwedge y. y \in U \implies p ((k \circ Pair 1) y) = f' y$ 
    by ( $auto simp: h1 heq [symmetric]$ )
  qed
qed

corollary  $covering\_space\_lift\_inessential\_function$ :
  fixes  $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$  and  $U :: 'c::real\_normed\_vector$ 
   $set$ 
  assumes  $cov: covering\_space C p S$ 
    and  $hom: homotopic\_with\_canon (\lambda x. True) U S f (\lambda x. a)$ 
  obtains  $g$  where  $continuous\_on U g g \text{ ' } U \subseteq C \bigwedge y. y \in U \implies p(g y) = f y$ 
proof ( $cases U = \{\}$ )
  case  $True$ 
    then show  $?thesis$ 
    using  $that continuous\_on\_empty$  by  $blast$ 
  next
  case  $False$ 
    then obtain  $b$  where  $b: b \in C p b = a$ 
    using  $covering\_space\_imp\_surjective [OF cov] homotopic\_with\_imp\_subset2$ 
    [ $OF hom$ ]
    by  $auto$ 
    then have  $gim: (\lambda y. b) \in U \rightarrow C$ 
    by  $blast$ 
    show  $?thesis$ 
  proof ( $rule covering\_space\_lift\_homotopic\_function [OF cov continuous\_on\_const$ 
 $gim]$ )
    show  $\bigwedge y. y \in U \implies p b = a$ 
    using  $b$  by  $auto$ 
  qed ( $use that homotopic\_with\_symD [OF hom]$  in  $auto$ )
qed

```

9.8.5 Lifting of general functions to covering space

```

proposition  $covering\_space\_lift\_path\_strong$ :
  fixes  $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$ 
    and  $f :: 'c::real\_normed\_vector \Rightarrow 'b$ 
  assumes  $cov: covering\_space C p S$  and  $a \in C$ 

```

and $\text{path } g$ **and** $\text{pag: path_image } g \subseteq S$ **and** $\text{pas: pathstart } g = p \ a$
obtains h **where** $\text{path } h$ $\text{path_image } h \subseteq C$ $\text{pathstart } h = a$
and $\bigwedge t. t \in \{0..1\} \implies p(h \ t) = g \ t$

proof –

obtain $k:: \text{real} \times 'c \Rightarrow 'a$
where $\text{contk: continuous_on } (\{0..1\} \times \{\text{undefined}\}) \ k$
and $\text{kim: } k \ ' (\{0..1\} \times \{\text{undefined}\}) \subseteq C$
and $\text{k0: } k \ (0, \text{undefined}) = a$
and $\text{pk: } \bigwedge z. z \in \{0..1\} \times \{\text{undefined}\} \implies p(k \ z) = (g \circ \text{fst}) \ z$
proof (*rule covering_space_lift_homotopy* [*OF cov, of* $\{\text{undefined}\} \ g \circ \text{fst}$])
show $\text{continuous_on } (\{0..1::\text{real}\} \times \{\text{undefined}::'c\}) \ (g \circ \text{fst})$
using $\langle \text{path } g \rangle$ **by** (*intro continuous_intros*) (*simp add: path_def*)
show $(g \circ \text{fst}) \in (\{0..1\} \times \{\text{undefined}\}) \rightarrow S$
using pag **by** (*auto simp: path_image_def*)
show $(g \circ \text{fst}) \ (0, y) = p \ a$ **if** $y \in \{\text{undefined}\}$ **for** $y::'c$
by (*metis comp_def fst_conv pas pathstart_def*)
qed (*use assms in auto*)
show *?thesis*
proof
show $\text{path } (k \circ (\lambda t. \text{Pair } t \ \text{undefined}))$
unfolding path_def
by (*intro continuous_on_compose continuous_intros continuous_on_subset*
[*OF contk*]) *auto*
show $\text{path_image } (k \circ (\lambda t. (t, \text{undefined}))) \subseteq C$
using kim **by** (*auto simp: path_image_def*)
show $\text{pathstart } (k \circ (\lambda t. (t, \text{undefined}))) = a$
by (*auto simp: pathstart_def k0*)
show $\bigwedge t. t \in \{0..1\} \implies p \ ((k \circ (\lambda t. (t, \text{undefined}))) \ t) = g \ t$
by (*auto simp: pk*)
qed
qed

corollary *covering_space_lift_path*:

fixes $p:: 'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$
assumes $\text{cov: covering_space } C \ p \ S$ **and** $\text{path } g$ **and** $\text{pig: path_image } g \subseteq S$
obtains h **where** $\text{path } h$ $\text{path_image } h \subseteq C$ $\bigwedge t. t \in \{0..1\} \implies p(h \ t) = g \ t$
proof –
obtain a **where** $a \in C$ $\text{pathstart } g = p \ a$
by (*metis pig cov covering_space_imp_surjective imageE pathstart_in_path_image subsetCE*)
show *?thesis*
using *covering_space_lift_path_strong* [*OF cov* $\langle a \in C \rangle \langle \text{path } g \rangle \text{ pig}$]
by (*metis* $\langle \text{pathstart } g = p \ a \rangle$ *that*)
qed

proposition *covering_space_lift_homotopic_paths*:

fixes $p:: 'a::\text{real_normed_vector} \Rightarrow 'b::\text{real_normed_vector}$
assumes $\text{cov: covering_space } C \ p \ S$

```

    and path g1 and pig1: path_image g1  $\subseteq$  S
    and path g2 and pig2: path_image g2  $\subseteq$  S
    and hom: homotopic_paths S g1 g2
    and path h1 and pih1: path_image h1  $\subseteq$  C and ph1:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h1 t) = g1 t
    and path h2 and pih2: path_image h2  $\subseteq$  C and ph2:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h2 t) = g2 t
    and h1h2: pathstart h1 = pathstart h2
    shows homotopic_paths C h1 h2
proof -
  obtain h :: real  $\times$  real  $\implies$  'b
  where conth: continuous_on ( $\{0..1\} \times \{0..1\}$ ) h
    and him: h  $\in$  ( $\{0..1\} \times \{0..1\}$ )  $\rightarrow$  S
    and h0:  $\bigwedge x. h (0, x) = g1 x$  and h1:  $\bigwedge x. h (1, x) = g2 x$ 
    and heq0:  $\bigwedge t. t \in \{0..1\} \implies h (t, 0) = g1 0$ 
    and heq1:  $\bigwedge t. t \in \{0..1\} \implies h (t, 1) = g1 1$ 
  using hom by (auto simp: homotopic_paths_def homotopic_with_def path-
start_def pathfinish_def image_subset_iff_funcset)
  obtain k where kontk: continuous_on ( $\{0..1\} \times \{0..1\}$ ) k
    and kim: k  $\in$  ( $\{0..1\} \times \{0..1\}$ )  $\rightarrow$  C
    and kh2:  $\bigwedge y. y \in \{0..1\} \implies k (y, 0) = h2 0$ 
    and hpk:  $\bigwedge z. z \in \{0..1\} \times \{0..1\} \implies h z = p (k z)$ 
proof (rule covering_space_lift_homotopy_alt [OF cov conth him])
  show  $\bigwedge y. y \in \{0..1\} \implies h (y, 0) = p (h2 0)$ 
  by (metis atLeastAtMost_iff h1h2 heq0 order_refl pathstart_def ph1 zero_le_one)
qed (use path_image_def pih2 in <fastforce+>)
have contg1: continuous_on  $\{0..1\}$  g1 and contg2: continuous_on  $\{0..1\}$  g2
  using <path g1> <path g2> path_def by blast+
have g1im: g1  $\in$   $\{0..1\} \rightarrow$  S and g2im: g2  $\in$   $\{0..1\} \rightarrow$  S
  using path_image_def pig1 pig2 by auto
have conth1: continuous_on  $\{0..1\}$  h1 and conth2: continuous_on  $\{0..1\}$  h2
  using <path h1> <path h2> path_def by blast+
have h1im: h1  $\in$   $\{0..1\} \rightarrow$  C and h2im: h2  $\in$   $\{0..1\} \rightarrow$  C
  using path_image_def pih1 pih2 by auto
show ?thesis
  unfolding homotopic_paths pathstart_def pathfinish_def
proof (intro exI conjI ballI)
  show keqh1: k(0, x) = h1 x if x  $\in$   $\{0..1\}$  for x
  proof (rule covering_space_lift_unique [OF cov _ contg1 g1im])
    show k (0,0) = h1 0
    by (metis atLeastAtMost_iff h1h2 kh2 order_refl pathstart_def zero_le_one)
    show continuous_on  $\{0..1\}$  ( $\lambda a. k (0, a)$ )
    by (intro continuous_intros continuous_on_compose2 [OF kontk]) auto
    show  $\bigwedge x. x \in \{0..1\} \implies g1 x = p (k (0, x))$ 
    by (metis atLeastAtMost_iff h0 hpk zero_le_one mem_Sigma_iff order_refl)
  qed (folding conth1 h1im kim that in <auto simp: ph1>)
  show k(1, x) = h2 x if x  $\in$   $\{0..1\}$  for x
  proof (rule covering_space_lift_unique [OF cov _ contg2 g2im])
    show k (1,0) = h2 0

```

```

    by (metis atLeastAtMost_iff kh2 order_refl zero_le_one)
  show continuous_on {0..1} (λa. k (1, a))
    by (intro continuous_intros continuous_on_compose2 [OF contk]) auto
  show  $\bigwedge x. x \in \{0..1\} \implies g2\ x = p\ (k\ (1, x))$ 
    by (metis atLeastAtMost_iff h1 hpk mem_Sigma_iff order_refl zero_le_one)
qed (use conth2 h2im kim that in ⟨auto simp: ph2⟩)
show  $\bigwedge t. t \in \{0..1\} \implies (k \circ \text{Pair } t)\ 0 = h1\ 0$ 
  by (metis comp_apply h1h2 kh2 pathstart_def)
show  $(k \circ \text{Pair } t)\ 1 = h1\ 1$  if  $t \in \{0..1\}$  for  $t$ 
proof (rule covering_space_lift_unique
  [OF cov, of λa. (k ∘ Pair a) 1 0 λa. h1 1 {0..1} λx. g1 1])
  show  $(k \circ \text{Pair } 0)\ 1 = h1\ 1$ 
    using keq1 by auto
  show continuous_on {0..1} (λa. (k ∘ Pair a) 1)
    by (auto intro!: continuous_intros continuous_on_compose2 [OF contk])
  show  $\bigwedge x. x \in \{0..1\} \implies g1\ 1 = p\ ((k \circ \text{Pair } x)\ 1)$ 
    using heq1 hpk by auto
qed (use contk kim g1im h1im that in ⟨auto simp: ph1⟩)
qed (use contk kim in auto)
qed

```

corollary *covering_space_monodromy:*

```

  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes cov: covering_space C p S
    and path g1 and pig1: path_image g1  $\subseteq$  S
    and path g2 and pig2: path_image g2  $\subseteq$  S
    and hom: homotopic_paths S g1 g2
    and path h1 and pih1: path_image h1  $\subseteq$  C and ph1:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h1 t) = g1 t
    and path h2 and pih2: path_image h2  $\subseteq$  C and ph2:  $\bigwedge t. t \in \{0..1\} \implies$ 
p(h2 t) = g2 t
    and h1h2: pathstart h1 = pathstart h2
  shows pathfinish h1 = pathfinish h2
  using covering_space_lift_homotopic_paths [OF assms] homotopic_paths_imp_pathfinish
  by blast

```

corollary *covering_space_lift_homotopic_path:*

```

  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  assumes cov: covering_space C p S
    and hom: homotopic_paths S f f'
    and path g and pig: path_image g  $\subseteq$  C
    and a: pathstart g = a and b: pathfinish g = b
    and pgeq:  $\bigwedge t. t \in \{0..1\} \implies p(g\ t) = f\ t$ 
  obtains g' where path g' path_image g'  $\subseteq$  C
    pathstart g' = a pathfinish g' = b  $\bigwedge t. t \in \{0..1\} \implies p(g'\ t) = f'\ t$ 
proof (rule covering_space_lift_path_strong [OF cov, of a f'])
  show a  $\in$  C

```

```

  using a pig by auto
  show path f' path_image f'  $\subseteq$  S
  using hom homotopic_paths_imp_path homotopic_paths_imp_subset by blast+
  show pathstart f' = p a
  by (metis a atLeastAtMost_iff hom homotopic_paths_imp_pathstart order_refl
  pathstart_def pgeq zero_le_one)
  qed (metis (mono_tags, lifting) assms cov covering_space_monodromy hom ho-
  motopic_paths_imp_path homotopic_paths_imp_subset pgeq pig)

```

proposition *covering_space_lift_general*:

```

  fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  and f :: 'c::real_normed_vector  $\Rightarrow$  'b
  assumes cov: covering_space C p S and a  $\in$  C z  $\in$  U
  and U: path_connected U locally_path_connected U
  and contf: continuous_on U f and fim: f  $\in$  U  $\rightarrow$  S
  and feq: f z = p a
  and hom:  $\bigwedge r. \llbracket \text{path } r; \text{path\_image } r \subseteq U; \text{pathstart } r = z; \text{pathfinish } r = z \rrbracket$ 
     $\implies \exists q. \text{path } q \wedge \text{path\_image } q \subseteq C \wedge$ 
      pathstart q = a  $\wedge$  pathfinish q = a  $\wedge$ 
      homotopic_paths S (f  $\circ$  r) (p  $\circ$  q)
  obtains g where continuous_on U g g  $\in$  U  $\rightarrow$  C g z = a  $\wedge \forall y. y \in U \implies p(g$ 
  y) = f y
  proof -
  have *:  $\exists g h. \text{path } g \wedge \text{path\_image } g \subseteq U \wedge$ 
    pathstart g = z  $\wedge$  pathfinish g = y  $\wedge$ 
    path h  $\wedge$  path_image h  $\subseteq$  C  $\wedge$  pathstart h = a  $\wedge$ 
    ( $\forall t \in \{0..1\}. p(h t) = f(g t)$ )
  if y  $\in$  U for y
  proof -
  obtain g where path g path_image g  $\subseteq$  U and pastg: pathstart g = z
  and pafig: pathfinish g = y
  using U  $\langle z \in U \rangle \langle y \in U \rangle$  by (force simp: path_connected_def)
  obtain h where path h path_image h  $\subseteq$  C pathstart h = a
  and  $\bigwedge t. t \in \{0..1\} \implies p(h t) = (f \circ g) t$ 
  proof (rule covering_space_lift_path_strong [OF cov  $\langle a \in C \rangle$ ])
  show path (f  $\circ$  g)
  using  $\langle \text{path } g \rangle \langle \text{path\_image } g \subseteq U \rangle$  contf continuous_on_subset path_continuous_image
  by blast
  show path_image (f  $\circ$  g)  $\subseteq$  S
  by (metis  $\langle \text{path\_image } g \subseteq U \rangle$  fim image_mono path_image_compose
  subset_trans image_subset_iff_funcset)
  show pathstart (f  $\circ$  g) = p a
  by (simp add: feq pastg pathstart_compose)
  qed auto
  then show ?thesis
  by (metis  $\langle \text{path } g \rangle \langle \text{path\_image } g \subseteq U \rangle$  comp_apply pafig pastg)
  qed
  have  $\exists l. \forall g h. \text{path } g \wedge \text{path\_image } g \subseteq U \wedge \text{pathstart } g = z \wedge \text{pathfinish } g =$ 

```

$y \wedge$

$path\ h \wedge path_image\ h \subseteq C \wedge pathstart\ h = a \wedge$
 $(\forall t \in \{0..1\}. p(h\ t) = f(g\ t)) \longrightarrow pathfinish\ h = l$ **for** y

proof –

have $pathfinish\ h = pathfinish\ h'$
if $g: path\ g\ path_image\ g \subseteq U\ pathstart\ g = z\ pathfinish\ g = y$
and $h: path\ h\ path_image\ h \subseteq C\ pathstart\ h = a$
and $phg: \bigwedge t. t \in \{0..1\} \implies p(h\ t) = f(g\ t)$
and $g': path\ g'\ path_image\ g' \subseteq U\ pathstart\ g' = z\ pathfinish\ g' = y$
and $h': path\ h'\ path_image\ h' \subseteq C\ pathstart\ h' = a$
and $phg': \bigwedge t. t \in \{0..1\} \implies p(h'\ t) = f(g'\ t)$
for $g\ h\ g'\ h'$

proof –

obtain q **where** $path\ q$ **and** $piq: path_image\ q \subseteq C$ **and** $pastq: pathstart\ q$
 $= a$ **and** $pafiq: pathfinish\ q = a$
and $homS: homotopic_paths\ S\ (f \circ g\ +++\ reversepath\ g')\ (p \circ q)$
using $g\ g'\ hom$ [of $g\ +++\ reversepath\ g'$] **by** (*auto simp: subset_path_image_join*)
have $papq: path\ (p \circ q)$
using $homS\ homotopic_paths_imp_path$ **by** *blast*
have $pipq: path_image\ (p \circ q) \subseteq S$
using $homS\ homotopic_paths_imp_subset$ **by** *blast*
obtain q' **where** $path\ q'\ path_image\ q' \subseteq C$
and $pathstart\ q' = pathstart\ q\ pathfinish\ q' = pathfinish\ q$
and $pq'_eq: \bigwedge t. t \in \{0..1\} \implies p\ (q'\ t) = (f \circ g\ +++\ reversepath$

$g')$ t

using *covering_space_lift_homotopic_path* [*OF cov homotopic_paths_sym*
[*OF homS*] $\langle path\ q \rangle\ piq\ refl\ refl$]

by *auto*

have $q'\ t = (h \circ (*_R)\ 2)\ t$ **if** $0 \leq t \leq 1/2$ **for** t

proof (*rule covering_space_lift_unique* [*OF cov, of q' 0 h \circ (*_R)\ 2 \{0..1/2\}*
 $f \circ g \circ (*_R)\ 2\ t$])

show $q'\ 0 = (h \circ (*_R)\ 2)\ 0$

by (*metis* $\langle pathstart\ q' = pathstart\ q \rangle\ comp_def\ h(3)\ pastq\ pathstart_def$
 $pth_4(2)$)

show *continuous_on* $\{0..1/2\}$ $(f \circ g \circ (*_R)\ 2)$

proof (*intro continuous_intros continuous_on_path* [*OF* $\langle path\ g \rangle$] *contin-*
uous_on_subset [*OF contf*])

show $g\ ' (*_R)\ 2\ ' \{0..1/2\} \subseteq U$

using $g\ path_image_def$ **by** *fastforce*

qed *auto*

show $(f \circ g \circ (*_R)\ 2) \in \{0..1/2\} \rightarrow S$

using $g(2)\ fim$ **by** (*fastforce simp: path_image_def image_subset_iff_funcset*)

show $(h \circ (*_R)\ 2) \in \{0..1/2\} \rightarrow C$

using $h\ path_image_def$ **by** *fastforce*

show $q' \in \{0..1/2\} \rightarrow C$

using $\langle path_image\ q' \subseteq C \rangle\ path_image_def$ **by** *fastforce*

show $\bigwedge x. x \in \{0..1/2\} \implies (f \circ g \circ (*_R)\ 2)\ x = p\ (q'\ x)$

by (*auto simp: joinpaths_def pq'_eq*)

show $\bigwedge x. x \in \{0..1/2\} \implies (f \circ g \circ (*_R)\ 2)\ x = p\ ((h \circ (*_R)\ 2)\ x)$

```

    by (simp add: phg)
  show continuous_on {0..1/2} q'
    by (simp add: continuous_on_path ⟨path q'⟩)
  show continuous_on {0..1/2} (h ∘ (*R) 2)
    by (intro continuous_intros continuous_on_path [OF ⟨path h⟩]) auto
  qed (use that in auto)
  moreover have q' t = (reversepath h' ∘ (λt. 2 *R t - 1)) t if 1/2 < t t ≤
  1 for t
  proof (rule covering_space_lift_unique [OF cov, of q' 1 reversepath h' ∘ (λt.
  2 *R t - 1) {1/2<..1} f ∘ reversepath g' ∘ (λt. 2 *R t - 1) t])
    show q' 1 = (reversepath h' ∘ (λt. 2 *R t - 1)) 1
      using h' ⟨pathfinish q' = pathfinish q⟩ pafiq
    by (simp add: pathstart_def pathfinish_def reversepath_def)
    show continuous_on {1/2<..1} (f ∘ reversepath g' ∘ (λt. 2 *R t - 1))
      proof (intro continuous_intros continuous_on_path ⟨path g'⟩ continu-
ous_on_subset [OF contf])
        show reversepath g' ' (λt. 2 *R t - 1) ' {1/2<..1} ⊆ U
          using g' by (auto simp: path_image_def reversepath_def)
        qed (use g' in auto)
        show (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) ∈ {1/2<..1} → S
          using g'(2) path_image_def fim by (auto simp: image_subset_iff
path_image_def reversepath_def)
        show q' ∈ {1/2<..1} → C
          using ⟨path_image q' ⊆ C⟩ path_image_def by fastforce
        show (reversepath h' ∘ (λt. 2 *R t - 1)) ∈ {1/2<..1} → C
          using h' by (simp add: path_image_def reversepath_def subset_eq)
        show ∧x. x ∈ {1/2<..1} ⇒ (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) x =
p (q' x)
          by (auto simp: joinpaths_def pq'_eq)
        show ∧x. x ∈ {1/2<..1} ⇒
          (f ∘ reversepath g' ∘ (λt. 2 *R t - 1)) x = p ((reversepath h' ∘ (λt.
2 *R t - 1)) x)
          by (simp add: phg' reversepath_def)
        show continuous_on {1/2<..1} q'
          by (auto intro: continuous_on_path [OF ⟨path q'⟩])
        show continuous_on {1/2<..1} (reversepath h' ∘ (λt. 2 *R t - 1))
          by (intro continuous_intros continuous_on_path ⟨path h'⟩) (use h' in
auto)
        qed (use that in auto)
      ultimately have q' t = (h +++ reversepath h') t if 0 ≤ t t ≤ 1 for t
        using that by (simp add: joinpaths_def)
      then have path(h +++ reversepath h')
        by (auto intro: path_eq [OF ⟨path q'⟩])
      then show ?thesis
        by (auto simp: ⟨path h⟩ ⟨path h'⟩)
    qed
  then show ?thesis by metis
  qed
  then obtain l :: 'c ⇒ 'a

```

where $l: \bigwedge y g h. \llbracket \text{path } g; \text{path_image } g \subseteq U; \text{pathstart } g = z; \text{pathfinish } g = y;$
 $\text{path } h; \text{path_image } h \subseteq C; \text{pathstart } h = a;$
 $\bigwedge t. t \in \{0..1\} \implies p(h \ t) = f(g \ t) \rrbracket \implies \text{pathfinish } h = l \ y$

by *metis*
show *?thesis*
proof
show $\text{pleq: } p \ (l \ y) = f \ y$ **if** $y \in U$ **for** y
using* $\langle OF \ \langle y \in U \rangle \rangle$ **by** (*metis* l *atLeastAtMost_iff* *order_refl* *pathfinish_def* *zero_le_one*)
show $l \ z = a$
using l [*of* *linepath* $z \ z \ z$ *linepath* $a \ a$] **by** (*auto simp: assms*)
show $LC: l \in U \rightarrow C$
by (*clarify* *dest!*: $*$) (*metis* (*full_types*) l *pathfinish_in_path_image* *subsetCE*)
have $\exists T. \text{openin } (\text{top_of_set } U) \ T \wedge y \in T \wedge T \subseteq U \cap l \ -' \ X$
if $X: \text{openin } (\text{top_of_set } C) \ X$ **and** $y \in U \ l \ y \in X$ **for** $X \ y$
proof $-$
have $X \subseteq C$
using X *openin_euclidean_subtopology_iff* **by** *blast*
have $f \ y \in S$
using *fm* $\langle y \in U \rangle$ **by** *blast*
then obtain $W \ \mathcal{V}$
where $WV: f \ y \in W \wedge \text{openin } (\text{top_of_set } S) \ W \wedge$
 $(\bigcup \mathcal{V} = C \cap p \ -' \ W \wedge$
 $(\forall U \in \mathcal{V}. \text{openin } (\text{top_of_set } C) \ U) \wedge$
 $\text{pairwise disjoint } \mathcal{V} \wedge$
 $(\forall U \in \mathcal{V}. \exists q. \text{homeomorphism } U \ W \ p \ q))$
using *cov* **by** (*force simp: covering_space_def*)
then have $l \ y \in \bigcup \mathcal{V}$
using $\langle X \subseteq C \rangle$ *pleq* **that** **by** *auto*
then obtain W' **where** $l \ y \in W'$ **and** $W' \in \mathcal{V}$
by *blast*
with WV **obtain** p' **where** $\text{ope}CW': \text{openin } (\text{top_of_set } C) \ W'$
and $\text{hom}UW': \text{homeomorphism } W' \ W \ p \ p'$
by *blast*
then have $\text{cont}p': \text{continuous_on } W \ p'$ **and** $p'im: p' \ -' \ W \subseteq W'$
using $\text{hom}UW'$ *homeomorphism_image2* *homeomorphism_cont2* **by** *fast-force+*
obtain V **where** $y \in V \ y \in U$ **and** $\text{fm}W: f \ -' \ V \subseteq W \ V \subseteq U$
and $\text{path_connected } V$ **and** $\text{ope}UV: \text{openin } (\text{top_of_set } U) \ V$
proof $-$
have $\text{openin } (\text{top_of_set } U) \ (U \cap f \ -' \ W)$
using WV *contf* *continuous_on_open_gen* *fm* **by** *auto*
then obtain UO **where** $\text{openin } (\text{top_of_set } U) \ UO \wedge \text{path_connected } UO$
 $\wedge y \in UO \wedge UO \subseteq U \cap f \ -' \ W$
using $U \ WV \ \langle y \in U \rangle$ *unfolding* *locally_path_connected* **by** (*meson* *IntI* *vimage_eq*)
then show *?thesis*
by (*meson* $\langle y \in U \rangle$ *image_subset_iff_subset_vimage* *le_inf_iff* *that*)


```

qed
have  $W' \subseteq C$   $W \subseteq S$ 
  using opeCW' WV openin_imp_subset by auto
have p'im:  $p' \text{ ` } W \subseteq W'$ 
  using homUW' homeomorphism_image2 by fastforce
show ?thesis
proof (intro exI conjI)
  have openin (top_of_set S) (W  $\cap$  p' ` (W'  $\cap$  X))
  proof (rule openin_trans)
    show openin (top_of_set W) (W  $\cap$  p' ` (W'  $\cap$  X))
      using  $X \langle W' \subseteq C \rangle$ 
      by (metis continuous_on_open contp' homUW' homeomorphism_image2
inf.assoc inf.orderE openin_open)
    show openin (top_of_set S) W
      using WV by blast
  qed
  then show openin (top_of_set U) (V  $\cap$  (U  $\cap$  (f ` (W  $\cap$  (p' ` (W'  $\cap$ 
X))))))
    by (blast intro: opeUV openin_subtopology_self continuous_openin_preimage
[OF contf fim])
    have  $p' (f y) \in X$ 
      using  $\langle l y \in W' \rangle$  homeomorphism_apply1 [OF homUW'] pleq  $\langle y \in U \rangle$   $\langle l$ 
 $y \in X \rangle$  by fastforce
    then show  $y \in V \cap (U \cap f \text{ ` } (W \cap p' \text{ ` } (W' \cap X)))$ 
      using  $\langle y \in U \rangle$   $\langle y \in V \rangle$  WV p'im by auto
    show  $V \cap (U \cap f \text{ ` } (W \cap p' \text{ ` } (W' \cap X))) \subseteq U \cap l \text{ ` } X$ 
    proof (intro subsetI IntI; clarify)
      fix  $y'$ 
      assume  $y': y' \in V$   $y' \in U$   $f y' \in W$   $p' (f y') \in W'$   $p' (f y') \in X$ 
      then obtain  $\gamma$  where path  $\gamma$  path_image  $\gamma \subseteq V$  pathstart  $\gamma = y$  pathfinish
 $\gamma = y'$ 
        by (meson  $\langle$ path_connected  $V \rangle$   $\langle y \in V \rangle$  path_connected_def)
      obtain pp qq where pp: path pp path_image pp  $\subseteq U$  pathstart pp =  $z$ 
pathfinish pp =  $y$ 
        and qq: path qq path_image qq  $\subseteq C$  pathstart qq =  $a$ 
        and pqqeq:  $\bigwedge t. t \in \{0..1\} \implies p(qq t) = f(pp t)$ 
      using* [OF  $\langle y \in U \rangle$ ] by blast
      have finW:  $\bigwedge x. [0 \leq x; x \leq 1] \implies f(\gamma x) \in W$ 
      using  $\langle$ path_image  $\gamma \subseteq V \rangle$  by (auto simp: image_subset_iff path_image_def
finW [THEN subsetD])
      have pathfinish ( $qq$  +++ ( $p' \circ f \circ \gamma$ )) =  $l y'$ 
      proof (rule l [of pp +++  $\gamma$  y' qq +++ ( $p' \circ f \circ \gamma$ )])
        show path ( $pp$  +++  $\gamma$ )
          by (simp add:  $\langle$ path  $\gamma \rangle$   $\langle$ path pp $\rangle$   $\langle$ pathfinish pp =  $y \rangle$   $\langle$ pathstart  $\gamma$  =  $y \rangle$ )
        show path_image ( $pp$  +++  $\gamma$ )  $\subseteq U$ 
          using  $\langle V \subseteq U \rangle$   $\langle$ path_image  $\gamma \subseteq V \rangle$   $\langle$ path_image pp  $\subseteq U \rangle$ 
not_in_path_image_join by blast
        show pathstart ( $pp$  +++  $\gamma$ ) =  $z$ 
          by (simp add:  $\langle$ pathstart pp =  $z \rangle$ )

```

```

show pathfinish (pp +++  $\gamma$ ) =  $y'$ 
  by (simp add:  $\langle$ pathfinish  $\gamma$  =  $y'$  $\rangle$ )
have pathfinish qq =  $l y$ 
  using  $\langle$ path pp $\rangle$   $\langle$ path qq $\rangle$   $\langle$ path_image pp  $\subseteq U$  $\rangle$   $\langle$ path_image qq  $\subseteq C$  $\rangle$ 
 $\langle$ pathfinish pp =  $y$  $\rangle$   $\langle$ pathstart pp =  $z$  $\rangle$   $\langle$ pathstart qq =  $a$  $\rangle$   $l$  pqeqq by blast
also have ... =  $p' (f y)$ 
  using  $\langle$ l  $y \in W'$  $\rangle$  homUW' homeomorphism_apply1 pleq that(2) by
fastforce
finally have pathfinish qq =  $p' (f y)$  .
then have paqq: pathfinish qq = pathstart ( $p' \circ f \circ \gamma$ )
  by (simp add:  $\langle$ pathstart  $\gamma$  =  $y$  $\rangle$  pathstart_compose)
have continuous_on (path_image  $\gamma$ ) ( $p' \circ f$ )
proof (rule continuous_on_compose)
  show continuous_on (path_image  $\gamma$ )  $f$ 
    using  $\langle$ path_image  $\gamma \subseteq V$  $\rangle$   $\langle V \subseteq U$  $\rangle$  contf continuous_on_subset
by blast
  show continuous_on ( $f \text{ ` path_image } \gamma$ )  $p'$ 
  proof (rule continuous_on_subset [OF contp'])
    show  $f \text{ ` path_image } \gamma \subseteq W$ 
      by (auto simp: path_image_def pathfinish_def pathstart_def finW)
  qed
qed
then show path (qq +++ ( $p' \circ f \circ \gamma$ ))
  using  $\langle$ path  $\gamma$  $\rangle$   $\langle$ path qq $\rangle$  paqq path_continuous_image path_join_imp
by blast
  show path_image (qq +++ ( $p' \circ f \circ \gamma$ ))  $\subseteq C$ 
  proof (rule subset_path_image_join)
    show path_image qq  $\subseteq C$ 
      by (simp add:  $\langle$ path_image qq  $\subseteq C$  $\rangle$ )
    show path_image ( $p' \circ f \circ \gamma$ )  $\subseteq C$ 
      by (metis  $\langle W' \subseteq C$  $\rangle$   $\langle$ path_image  $\gamma \subseteq V$  $\rangle$  dual_order.trans fimW(1)
image_comp image_mono p'im path_image_compose)
  qed
  show pathstart (qq +++ ( $p' \circ f \circ \gamma$ )) =  $a$ 
    by (simp add:  $\langle$ pathstart qq =  $a$  $\rangle$ )
  show  $p ((qq +++ ( $p' \circ f \circ \gamma$ ))  $\xi$ ) =  $f ((pp +++ \gamma) \xi)$  if  $\xi$ :  $\xi \in \{0..1\}$ 
for  $\xi$ 
  proof (simp add: joinpaths_def, safe)
    show  $p (qq (2*\xi)) = f (pp (2*\xi))$  if  $\xi*2 \leq 1$ 
      using  $\langle \xi \in \{0..1\} \rangle$  pqeqq that by auto
    show  $p (p' (f (\gamma (2*\xi - 1)))) = f (\gamma (2*\xi - 1))$  if  $\neg \xi*2 \leq 1$ 
      using that  $\xi$  by (auto intro: homeomorphism_apply2 [OF homUW'
finW])
  qed
qed
with  $\langle$ pathfinish  $\gamma$  =  $y'$  $\rangle$   $\langle p' (f y') \in X \rangle$  show  $y' \in l \text{ - ` } X$ 
  unfolding pathfinish_join by (simp add: pathfinish_def)
qed
qed$ 
```

```

qed
then show continuous_on U l
  using vimage_eq openin_subopen continuous_on_open_gen [OF LC]
  by (metis IntD1 IntD2 vimage_eq openin_subopen continuous_on_open_gen
[OF LC])
qed
qed

```

corollary *covering_space_lift_stronger*:

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
and f :: 'c::real_normed_vector  $\Rightarrow$  'b
assumes cov: covering_space C p S a  $\in$  C z  $\in$  U
and U: path_connected U locally_path_connected U
and contf: continuous_on U f and fim: f  $\in$  U  $\rightarrow$  S
and feq: f z = p a
and hom:  $\bigwedge r. \llbracket$ path r; path_image r  $\subseteq$  U; pathstart r = z; pathfinish r = z $\rrbracket$ 
 $\implies \exists b. \text{homotopic\_paths } S (f \circ r) (\text{linepath } b b)$ 
obtains g where continuous_on U g g  $\in$  U  $\rightarrow$  C g z = a  $\bigwedge y. y \in U \implies p(g$ 
y) = f y
proof (rule covering_space_lift_general [OF cov U contf fim feq])
  fix r
  assume path r path_image r  $\subseteq$  U pathstart r = z pathfinish r = z
  then obtain b where b: homotopic_paths S (f  $\circ$  r) (linepath b b)
    using hom by blast
  then have f (pathstart r) = b
    by (metis homotopic_paths_imp_pathstart pathstart_compose pathstart_linepath)
  then have homotopic_paths S (f  $\circ$  r) (linepath (f z) (f z))
    by (simp add: b  $\langle$ pathstart r = z $\rangle$ )
  then have homotopic_paths S (f  $\circ$  r) (p  $\circ$  linepath a a)
    by (simp add: o_def feq linepath_def)
  then show  $\exists q. \text{path } q \wedge$ 
    path_image q  $\subseteq$  C  $\wedge$ 
    pathstart q = a  $\wedge$  pathfinish q = a  $\wedge$  homotopic_paths S (f  $\circ$  r) (p
 $\circ$  q)
    by (force simp:  $\langle a \in C \rangle$ )
qed auto

```

corollary *covering_space_lift_strong*:

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
and f :: 'c::real_normed_vector  $\Rightarrow$  'b
assumes cov: covering_space C p S a  $\in$  C z  $\in$  U
and scU: simply_connected U and lpcU: locally_path_connected U
and contf: continuous_on U f and fim: f  $\in$  U  $\rightarrow$  S
and feq: f z = p a
obtains g where continuous_on U g g  $\in$  U  $\rightarrow$  C g z = a  $\bigwedge y. y \in U \implies p(g$ 
y) = f y
proof (rule covering_space_lift_stronger [OF cov _ lpcU contf fim feq])
  show path_connected U
    using scU simply_connected_eq_contractible_loop_some by blast

```

```

fix r
assume r: path r path_image r  $\subseteq$  U pathstart r = z pathfinish r = z
have linepath (f z) (f z) = f  $\circ$  linepath z z
  by (simp add: o_def linepath_def)
then have homotopic_paths S (f  $\circ$  r) (linepath (f z) (f z))
  by (metis r contf fim homotopic_paths_continuous_image scU simply_connected_eq_contractible_pat)
then show  $\exists$  b. homotopic_paths S (f  $\circ$  r) (linepath b b)
  by blast
qed blast

```

corollary covering_space_lift:

```

fixes p :: 'a::real_normed_vector  $\Rightarrow$  'b::real_normed_vector
  and f :: 'c::real_normed_vector  $\Rightarrow$  'b
assumes cov: covering_space C p S
  and U: simply_connected U locally_path_connected U
  and contf: continuous_on U f and fim: f  $\in$  U  $\rightarrow$  S
  obtains g where continuous_on U g g  $\in$  U  $\rightarrow$  C  $\wedge$  y. y  $\in$  U  $\implies$  p(g y) = f y
proof (cases U = {})
  case True
    with that show ?thesis by auto
  next
    case False
      then obtain z where z  $\in$  U by blast
      then obtain a where a  $\in$  C f z = p a
      by (metis cov covering_space_imp_surjective fim_image_iff Pi_iff)
      then show ?thesis
      by (metis that covering_space_lift_strong [OF cov _  $\langle$ z  $\in$  U $\rangle$  U contf fim])
qed

```

9.8.6 Homeomorphisms of arc images

lemma homeomorphism_arc:

```

fixes g :: real  $\Rightarrow$  'a::t2_space
assumes arc g
obtains h where homeomorphism {0..1} (path_image g) g h
using assms by (force simp: arc_def homeomorphism_compact path_def path_image_def)

```

lemma homeomorphic_arc_image_interval:

```

fixes g :: real  $\Rightarrow$  'a::t2_space and a::real
assumes arc g a < b
shows (path_image g) homeomorphic {a..b}
proof -
  have (path_image g) homeomorphic {0..1::real}
  by (meson assms(1) homeomorphic_def homeomorphic_sym homeomorphism_arc)
  also have ... homeomorphic {a..b}
  using assms by (force intro: homeomorphic_closed_intervals_real)
  finally show ?thesis .
qed

```

```

lemma homeomorphic_arc_images:
  fixes  $g :: \text{real} \Rightarrow 'a::t2\_space$  and  $h :: \text{real} \Rightarrow 'b::t2\_space$ 
  assumes arc g arc h
  shows  $(\text{path\_image } g) \text{ homeomorphic } (\text{path\_image } h)$ 
proof -
  have  $(\text{path\_image } g) \text{ homeomorphic } \{0..1::\text{real}\}$ 
  by (meson assms homeomorphic_def homeomorphic_sym homeomorphism_arc)
  also have ... homeomorphic  $(\text{path\_image } h)$ 
  by (meson assms homeomorphic_def homeomorphism_arc)
  finally show ?thesis .
qed

end

```

```

theory Equivalence_Lebesgue_Henstock_Integration

```

```

imports

```

```

  Lebesgue_Measure
  Henstock_Kurzweil_Integration
  Complete_Measure
  Set_Integral
  Homeomorphism
  Cartesian_Euclidean_Space

```

```

begin

```

```

lemma LIMSEQ_if_less:  $(\lambda k. \text{if } i < k \text{ then } a \text{ else } b) \longrightarrow a$ 
  by (rule_tac k=Suc i in LIMSEQ_offset) auto

```

Note that the rhs is an implication. This lemma plays a specific role in one proof.

```

lemma le_left_mono:  $x \leq y \implies y \leq a \longrightarrow x \leq (a::'a::preorder)$ 
  by (auto intro: order_trans)

```

```

lemma ball_trans:

```

```

  assumes  $y \in \text{ball } z \ q \ r + q \leq s$  shows  $\text{ball } y \ r \subseteq \text{ball } z \ s$ 
  using assms by metric

```

```

lemma has_integral_implies_lebesgue_measurable_cbox:

```

```

  fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $f: (f \text{ has\_integral } I) (\text{cbox } x \ y)$ 
  shows  $f \in \text{lebesgue\_on } (\text{cbox } x \ y) \rightarrow_M \text{borel}$ 
proof (rule cld_measure.borel_measurable_cld)
  let ?L = lebesgue_on  $(\text{cbox } x \ y)$ 
  let ? $\mu$  = emeasure ?L
  let ? $\mu'$  = outer_measure_of ?L
  interpret L: finite_measure ?L
proof
  show ? $\mu$   $(\text{space } ?L) \neq \infty$ 
  by (simp add: emeasure_restrict_space space_restrict_space emeasure_lborel_cbox_eq)

```

qed

show *cld_measure* ?L

proof

fix B A assume $B \subseteq A$ $A \in \text{null_sets } ?L$

then show $B \in \text{sets } ?L$

using *null_sets_completion_subset*[OF $\langle B \subseteq A \rangle$, of *lborel*]

by (auto simp add: *null_sets_restrict_space* *sets_restrict_space_iff* intro:)

next

fix A assume $A \subseteq \text{space } ?L \wedge B. B \in \text{sets } ?L \implies ?\mu B < \infty \implies A \cap B \in \text{sets } ?L$

from *this(1)* *this(2)*[of *space* ?L] show $A \in \text{sets } ?L$

by (auto simp: *Int_absorb2* *less_top*[*symmetric*])

qed auto

then interpret *cld_measure* ?L

.

have *content_eq_L*: $A \in \text{sets borel} \implies A \subseteq \text{cbox } x y \implies \text{content } A = \text{measure } ?L A$ for A

by (subst *measure_restrict_space*) (auto simp: *measure_def*)

fix E and a b :: real assume $E \in \text{sets } ?L$ $a < b$ $0 < ?\mu E$ $?\mu E < \infty$

then obtain M :: real where $?\mu E = M$ $0 < M$

by (cases ? μE) auto

define e where $e = M / (4 + 2 / (b - a))$

from $\langle a < b \rangle \langle 0 < M \rangle$ have $0 < e$

by (auto intro!: *divide_pos_pos* simp: *field_simps* *e_def*)

have $e < M / (3 + 2 / (b - a))$

using $\langle a < b \rangle \langle 0 < M \rangle$

unfolding *e_def* by (intro *divide_strict_left_mono* *add_strict_right_mono* *mult_pos_pos*) (auto simp: *field_simps*)

then have $2 * e < (b - a) * (M - e * 3)$

using $\langle 0 < M \rangle \langle 0 < e \rangle \langle a < b \rangle$ by (simp add: *field_simps*)

have *e_less_M*: $e < M / 1$

unfolding *e_def* using $\langle a < b \rangle \langle 0 < M \rangle$ by (intro *divide_strict_left_mono*) (auto simp: *field_simps*)

obtain d

where *gauge* d

and *integral_f*: $\forall p. p \text{ tagged_division_of cbox } x y \wedge d \text{ fine } p \longrightarrow$

$\text{norm } ((\sum (x,k) \in p. \text{content } k *_{\mathbb{R}} f x) - I) < e$

using $\langle 0 < e \rangle$ f unfolding *has_integral* by auto

define C where $C X m = X \cap \{x. \text{ball } x (1/\text{Suc } m) \subseteq d x\}$ for X m

have *incseq* (C X) for X

unfolding *C_def* [*abs_def*]

by (intro *monoI* *Collect_mono* *conj_mono* *imp_refl* *le_left_mono* *subset_ball*)

divide_left_mono Int_mono) auto

```

{ fix X assume X  $\subseteq$  space ?L and eq: ? $\mu'$  X = ? $\mu$  E
  have (SUP m. outer_measure_of ?L (C X m)) = outer_measure_of ?L ( $\bigcup$  m.
C X m)
  using  $\langle$ X  $\subseteq$  space ?L $\rangle$  by (intro SUP_outer_measure_of_incseq  $\langle$ incseq (C
X) $\rangle$ ) (auto simp: C_def)
  also have ( $\bigcup$  m. C X m) = X
  proof -
    { fix x
      obtain e where 0 < e ball x e  $\subseteq$  d x
      using gaugeD[OF  $\langle$ gauge d $\rangle$ , of x] unfolding open_contains_ball by auto
      moreover
      obtain n where 1 / (1 + real n) < e
      using reals_Archimedean[OF  $\langle$ 0 < e $\rangle$ ] by (auto simp: inverse_eq_divide)
      then have ball x (1 / (1 + real n))  $\subseteq$  ball x e
      by (intro subset_ball) auto
      ultimately have  $\exists$  n. ball x (1 / (1 + real n))  $\subseteq$  d x
      by blast }
    then show ?thesis
      by (auto simp: C_def)
  qed
  finally have (SUP m. outer_measure_of ?L (C X m)) = ? $\mu$  E
  using eq by auto
  also have ... > M - e
  using  $\langle$ 0 < M $\rangle$   $\langle$ ? $\mu$  E = M $\rangle$   $\langle$ 0 < e $\rangle$  by (auto intro!: ennreal_lessI)
  finally have  $\exists$  m. M - e < outer_measure_of ?L (C X m)
  unfolding less_SUP_iff by auto }
note C = this

let ?E = {x  $\in$  E. f x  $\leq$  a} and ?F = {x  $\in$  E. b  $\leq$  f x}

have  $\neg$  (? $\mu'$  ?E = ? $\mu$  E  $\wedge$  ? $\mu'$  ?F = ? $\mu$  E)
proof
  assume eq: ? $\mu'$  ?E = ? $\mu$  E  $\wedge$  ? $\mu'$  ?F = ? $\mu$  E
  with C[of ?E] C[of ?F]  $\langle$ E  $\in$  sets ?L $\rangle$ [THEN sets.sets_into_space] obtain ma
mb
  where M - e < outer_measure_of ?L (C ?E ma) M - e < outer_measure_of
?L (C ?F mb)
  by auto
  moreover define m where m = max ma mb
  ultimately have M_minus_e: M - e < outer_measure_of ?L (C ?E m) M
- e < outer_measure_of ?L (C ?F m)
  using
    incseqD[OF  $\langle$ incseq (C ?E) $\rangle$ , of ma m, THEN outer_measure_of_mono]
    incseqD[OF  $\langle$ incseq (C ?F) $\rangle$ , of mb m, THEN outer_measure_of_mono]
  by (auto intro: less_le_trans)
  define d' where d' x = d x  $\cap$  ball x (1 / (3 * Suc m)) for x
  have gauge d'

```

```

unfolding  $d'_def$  by (intro gauge_Int ‹gauge d› gauge_ball) auto
then obtain  $p$  where  $p$ :  $p$  tagged_division_of cbox  $x$   $y$   $d'$  fine  $p$ 
by (rule fine_division_exists)
then have  $d$  fine  $p$ 
unfolding  $d'_def$ [abs_def] fine_def by auto

define  $s$  where  $s = \{(x::'a, k). k \cap (C \ ?E \ m) \neq \{\} \wedge k \cap (C \ ?F \ m) \neq \{\}\}$ 
define  $T$  where  $T \ E \ k = (SOME \ x. x \in k \cap C \ E \ m)$  for  $E \ k$ 
let  $?A = (\lambda(x, k). (T \ ?E \ k, k)) \ ' (p \cap s) \cup (p - s)$ 
let  $?B = (\lambda(x, k). (T \ ?F \ k, k)) \ ' (p \cap s) \cup (p - s)$ 

{ fix  $X$  assume  $X\_eq$ :  $X = ?E \vee X = ?F$ 
let  $?T = (\lambda(x, k). (T \ X \ k, k))$ 
let  $?p = ?T \ ' (p \cap s) \cup (p - s)$ 

have  $in\_s$ :  $(x, k) \in s \implies T \ X \ k \in k \cap C \ X \ m$  for  $x \ k$ 
using someI_ex[of  $\lambda x. x \in k \cap C \ X \ m$ ]  $X\_eq$  unfolding  $ex\_in\_conv$  by
(auto simp:  $T\_def \ s\_def$ )

{ fix  $x \ k$  assume  $(x, k) \in p \ (x, k) \in s$ 
have  $k \subseteq ball \ x \ (1 / (3 * Suc \ m))$ 
using ‹ $d'$  fine  $p$ ›[THEN fineD, OF ‹ $(x, k) \in p$ ›] by (auto simp:  $d'_def$ )
then have  $x \in ball \ (T \ X \ k) \ (1 / (3 * Suc \ m))$ 
using  $in\_s$ [OF ‹ $(x, k) \in s$ ›] by (auto simp: C_def subset_eq dist_commute)
then have  $ball \ x \ (1 / (3 * Suc \ m)) \subseteq ball \ (T \ X \ k) \ (1 / Suc \ m)$ 
by (rule ball_trans) (auto simp: field_split_simps)
with  $k \ in\_s$ [OF ‹ $(x, k) \in s$ ›] have  $k \subseteq d \ (T \ X \ k)$ 
by (auto simp: C_def) }
then have  $d$  fine  $?p$ 
using ‹ $d$  fine  $p$ › by (auto intro!: fineI)
moreover
have  $?p$  tagged_division_of cbox  $x \ y$ 
proof (rule tagged_division_ofI)
show finite  $?p$ 
using  $p(1)$  by auto
next
fix  $z \ k$  assume  $*$ :  $(z, k) \in ?p$ 
then consider  $(z, k) \in p \ (z, k) \notin s$ 
|  $x'$  where  $(x', k) \in p \ (x', k) \in s \ z = T \ X \ k$ 
by (auto simp:  $T\_def$ )
then have  $z \in k \wedge k \subseteq cbox \ x \ y \wedge (\exists a \ b. k = cbox \ a \ b)$ 
using  $p(1)$  by cases (auto dest:  $in\_s$ )
then show  $z \in k \ k \subseteq cbox \ x \ y \ \exists a \ b. k = cbox \ a \ b$ 
by auto
next
fix  $z \ k \ z' \ k'$  assume  $(z, k) \in ?p \ (z', k') \in ?p \ (z, k) \neq (z', k')$ 
with tagged_division_ofD(5)[OF  $p(1)$ , of  $\_ k \_ k'$ ]
show interior  $k \cap interior \ k' = \{\}$ 
by (auto simp:  $T\_def \ dest: in\_s$ )

```



```

next
  have  $\{k. \exists x. (x, k) \in ?p\} = \{k. \exists x. (x, k) \in p\}$ 
    by (auto simp: T_def image_iff Bex_def)
  then show  $\bigcup \{k. \exists x. (x, k) \in ?p\} = \text{cbox } x \ y$ 
    using p(1) by auto
qed
ultimately have  $I: \text{norm } ((\sum (x,k) \in ?p. \text{content } k *_R f x) - I) < e$ 
  using integral_f by auto

have  $(\sum (x,k) \in ?p. \text{content } k *_R f x) =$ 
   $(\sum (x,k) \in ?T '(p \cap s). \text{content } k *_R f x) + (\sum (x,k) \in p - s. \text{content } k$ 
*_R f x)
  using p(1)[THEN tagged_division_ofD(1)]
  by (safe intro!: sum.union_inter_neutral) (auto simp: s_def T_def)
  also have  $(\sum (x,k) \in ?T '(p \cap s). \text{content } k *_R f x) = (\sum (x,k) \in p \cap s.$ 
content k *_R f (T X k))
  proof (subst sum.reindex_nontrivial, safe)
    fix x1 x2 k assume 1:  $(x1, k) \in p$   $(x1, k) \in s$  and 2:  $(x2, k) \in p$   $(x2, k)$ 
 $\in s$ 
    and eq:  $\text{content } k *_R f (T X k) \neq 0$ 
    with tagged_division_ofD(5)[OF p(1), of x1 k x2 k] tagged_division_ofD(4)[OF
p(1), of x1 k]
    show  $x1 = x2$ 
      by (auto simp: content_eq_0_interior)
  qed (use p in <auto intro!: sum.cong>)
  finally have eq:  $(\sum (x,k) \in ?p. \text{content } k *_R f x) =$ 
 $(\sum (x,k) \in p \cap s. \text{content } k *_R f (T X k)) + (\sum (x,k) \in p - s. \text{content } k$ 
*_R f x) .

have in_T:  $(x, k) \in s \implies T X k \in X$  for x k
  using in_s[of x k] by (auto simp: C_def)

note I eq in_T }
note parts = this

have p_in_L:  $(x, k) \in p \implies k \in \text{sets } ?L$  for x k
  using tagged_division_ofD(3, 4)[OF p(1), of x k] by (auto simp: sets_restrict_space)

have [simp]: finite p
  using tagged_division_ofD(1)[OF p(1)] .

have  $(M - 3*e) * (b - a) \leq (\sum (x,k) \in p \cap s. \text{content } k) * (b - a)$ 
proof (intro mult_right_mono)
  have fin:  $?\mu (E \cap \bigcup \{k \in \text{snd}' p. k \cap C X m = \{\}\}) < \infty$  for X
    using  $\langle ?\mu E < \infty \rangle$  by (rule le_less_trans[rotated]) (auto intro!: emea-
sure_mono <E \in sets ?L>)
  have sets:  $(E \cap \bigcup \{k \in \text{snd}' p. k \cap C X m = \{\}\}) \in \text{sets } ?L$  for X
    using tagged_division_ofD(1)[OF p(1)] by (intro sets.Diff <E \in sets ?L>
sets.finite_Union sets.Int) (auto intro: p_in_L)

```

```

{ fix X assume X ⊆ E M - e < ?μ' (C X m)
  have M - e ≤ ?μ' (C X m)
    by (rule less_imp_le) fact
  also have ... ≤ ?μ' (E - (E ∩ ⋃ {k∈snd'p. k ∩ C X m = {}}))
  proof (intro outer_measure_of_mono subsetI)
    fix v assume v ∈ C X m
    then have v ∈ cbox x y v ∈ E
      using ⟨E ⊆ space ?L⟩ ⟨X ⊆ E⟩ by (auto simp: space_restrict_space
C_def)
    then obtain z k where (z, k) ∈ p v ∈ k
      using tagged_division_ofD(6)[OF p(1), symmetric] by auto
    then show v ∈ E - E ∩ (⋃ {k∈snd'p. k ∩ C X m = {}})
      using ⟨v ∈ C X m⟩ ⟨v ∈ E⟩ by auto
  qed
  also have ... = ?μ E - ?μ (E ∩ ⋃ {k∈snd'p. k ∩ C X m = {}})
    using ⟨E ∈ sets ?L⟩ fin[of X] sets[of X] by (auto intro!: emeasure_Diff)
  finally have ?μ (E ∩ ⋃ {k∈snd'p. k ∩ C X m = {}}) ≤ e
    using ⟨0 < e⟩ e_less_M
    by (cases ?μ (E ∩ ⋃ {k∈snd'p. k ∩ C X m = {}})) (auto simp add: ⟨?μ
E = M⟩ ennreal_minus ennreal_le_iff2)
    note this }
  note upper_bound = this

  have ?μ (E ∩ ⋃ (snd'(p - s))) =
    ?μ ((E ∩ ⋃ {k∈snd'p. k ∩ C ?E m = {}}) ∪ (E ∩ ⋃ {k∈snd'p. k ∩ C ?F
m = {}}))
    by (intro arg_cong[where f=?μ]) (auto simp: s_def image_def Bex_def)
  also have ... ≤ ?μ (E ∩ ⋃ {k∈snd'p. k ∩ C ?E m = {}}) + ?μ (E ∩
⋃ {k∈snd'p. k ∩ C ?F m = {}})
    using sets[of ?E] sets[of ?F] M_minus_e by (intro emeasure_subadditive)
  auto
  also have ... ≤ e + ennreal e
    using upper_bound[of ?E] upper_bound[of ?F] M_minus_e by (intro
add_mono) auto
  finally have ?μ E - 2*e ≤ ?μ (E - (E ∩ ⋃ (snd'(p - s))))
    using ⟨0 < e⟩ ⟨E ∈ sets ?L⟩ tagged_division_ofD(1)[OF p(1)]
    by (subst emeasure_Diff)
      (auto simp: top_unique simp flip: ennreal_plus
intro!: sets.Int sets.finite_UN ennreal_mono_minus intro: p_in_L)
  also have ... ≤ ?μ (⋃ x∈p ∩ s. snd x)
  proof (safe intro!: emeasure_mono subsetI)
    fix v assume v ∈ E and not: v ∉ (⋃ x∈p ∩ s. snd x)
    then have v ∈ cbox x y
      using ⟨E ⊆ space ?L⟩ by (auto simp: space_restrict_space)
    then obtain z k where (z, k) ∈ p v ∈ k
      using tagged_division_ofD(6)[OF p(1), symmetric] by auto
    with not show v ∈ ⋃ (snd '(p - s))
      by (auto intro!: bexI[of _ (z, k)] elim: ballE[of _ _ (z, k)])
  qed (auto intro!: sets.Int sets.finite_UN ennreal_mono_minus intro: p_in_L)

```

```

also have ... = measure ?L ( $\bigcup x \in p \cap s. \text{snd } x$ )
  by (auto intro!: emeasure_eq_ennreal_measure)
finally have  $M - 2 * e \leq \text{measure } ?L (\bigcup x \in p \cap s. \text{snd } x)$ 
  unfolding ⟨?μ E = M⟩ using ⟨0 < e⟩ by (simp add: ennreal_minus)
also have  $\text{measure } ?L (\bigcup x \in p \cap s. \text{snd } x) = \text{content } (\bigcup x \in p \cap s. \text{snd } x)$ 
  using tagged_division_ofD(1,3,4) [OF p(1)]
  by (intro content_eq_L[symmetric])
  (fastforce intro!: sets.finite_UN UN_least del: subsetI)+
also have  $\text{content } (\bigcup x \in p \cap s. \text{snd } x) \leq (\sum k \in p \cap s. \text{content } (\text{snd } k))$ 
using p(1) by (auto simp: emeasure_lborel_cbox_eq intro!: measure_subadditive_finite
  dest!: p(1)[THEN tagged_division_ofD(4)])
finally show  $M - 3 * e \leq (\sum (x, y) \in p \cap s. \text{content } y)$ 
  using ⟨0 < e⟩ by (simp add: split_beta)
qed (use ⟨a < b⟩ in auto)
also have ... =  $(\sum (x, k) \in p \cap s. \text{content } k * (b - a))$ 
  by (simp add: sum_distrib_right split_beta')
also have ... ≤  $(\sum (x, k) \in p \cap s. \text{content } k * (f (T ?F k) - f (T ?E k)))$ 
  using parts(3) by (auto intro!: sum_mono mult_left_mono diff_mono)
also have ... =  $(\sum (x, k) \in p \cap s. \text{content } k * f (T ?F k)) - (\sum (x, k) \in p \cap$ 
s. content k * f (T ?E k))
  by (auto intro!: sum.cong simp: field_simps sum_subtractf[symmetric])
also have ... =  $(\sum (x, k) \in ?B. \text{content } k *_{\mathbb{R}} f x) - (\sum (x, k) \in ?A. \text{content } k$ 
 $*_{\mathbb{R}} f x)$ 
  by (subst (1 2) parts) auto
also have ... ≤ norm  $((\sum (x, k) \in ?B. \text{content } k *_{\mathbb{R}} f x) - (\sum (x, k) \in ?A.$ 
 $\text{content } k *_{\mathbb{R}} f x))$ 
  by auto
also have ... ≤ e + e
  using parts(1)[of ?E] parts(1)[of ?F] by (intro norm_diff_triangle_le[of _
I]) auto
finally show False
  using ⟨2 * e < (b - a) * (M - e * 3)⟩ by (auto simp: field_simps)
qed
moreover have ?μ' ?E ≤ ?μ E ?μ' ?F ≤ ?μ E
  unfolding outer_measure_of_eq[OF ⟨E ∈ sets ?L⟩, symmetric] by (auto intro!:
outer_measure_of_mono)
  ultimately show min (?μ' ?E) (?μ' ?F) < ?μ E
  unfolding min_less_iff_disj by (auto simp: less_le)
qed

lemma has_integral_implies_lebesgue_measurable_real:
  fixes f :: 'a :: euclidean_space ⇒ real
  assumes f: (f has_integral I) Ω
  shows  $(\lambda x. f x * \text{indicator } \Omega x) \in \text{lebesgue} \rightarrow_M \text{borel}$ 
proof -
  define B :: nat ⇒ 'a set where B n = cbox (- real n *ℝ One) (real n *ℝ One)
for n
  show  $(\lambda x. f x * \text{indicator } \Omega x) \in \text{lebesgue} \rightarrow_M \text{borel}$ 
  proof (rule measurable_piecewise_restrict)

```

```

have (⋃ n. box (− real n *R One) (real n *R One)) ⊆ ⋃ (B ‘ UNIV)
  unfolding B_def by (intro UN_mono box_subset_cbox order_refl)
then show countable (range B) space lebesgue ⊆ ⋃ (B ‘ UNIV)
  by (auto simp: B_def UN_box_eq_UNIV)
next
fix Ω' assume Ω' ∈ range B
then obtain n where Ω': Ω' = B n by auto
then show Ω' ∩ space lebesgue ∈ sets lebesgue
  by (auto simp: B_def)

have f integrable_on Ω
  using f by auto
then have (λx. f x * indicator Ω x) integrable_on Ω
  by (auto simp: integrable_on_def cong: has_integral_cong)
then have (λx. f x * indicator Ω x) integrable_on (Ω ∪ B n)
  by (rule integrable_on_superset) auto
then have (λx. f x * indicator Ω x) integrable_on B n
  unfolding B_def by (rule integrable_on_subcbox) auto
then show (λx. f x * indicator Ω x) ∈ lebesgue_on Ω' →M borel
  unfolding B_def Ω' by (auto intro: has_integral_implies_lebesgue_measurable_cbox
simp: integrable_on_def)
qed
qed

```

```

lemma has_integral_implies_lebesgue_measurable:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f: (f has_integral I) Ω
  shows (λx. indicator Ω x *R f x) ∈ lebesgue →M borel
proof (intro borel_measurable_euclidean_space[where 'c='b, THEN iffD2] ballI)
  fix i :: 'b assume i ∈ Basis
  have (λx. (f x · i) * indicator Ω x) ∈ borel_measurable (completion lborel)
    using has_integral_linear[OF f bounded_linear_inner_left, of i]
  by (intro has_integral_implies_lebesgue_measurable_real) (auto simp: comp_def)
  then show (λx. indicator Ω x *R f x · i) ∈ borel_measurable (completion lborel)
    by (simp add: ac_simps)
qed

```

9.8.7 Equivalence Lebesgue integral on *lborel* and HK-integral

```

lemma has_integral_measure_lborel:
  fixes A :: 'a::euclidean_space set
  assumes A[measurable]: A ∈ sets borel and finite: emeasure lborel A < ∞
  shows ((λx. 1) has_integral measure lborel A) A
proof −
  { fix l u :: 'a
    have ((λx. 1) has_integral measure lborel (box l u)) (box l u)
    proof cases
      assume ∀ b ∈ Basis. l · b ≤ u · b
      then show ?thesis

```

```

    using has_integral_const[of 1::real l u]
  by (simp flip: has_integral_restrict[OF box_subset_cbox] add: has_integral_spike_interior)
next
  assume  $\neg (\forall b \in \text{Basis}. l \cdot b \leq u \cdot b)$ 
  then have  $\text{box } l \ u = \{\}$ 
    unfolding box_eq_empty by (auto simp: not_le intro: less_imp_le)
  then show ?thesis
    by simp
  qed }
note has_integral_box = this

{ fix a b :: 'a let ?M =  $\lambda A. \text{measure } l\text{borel } (A \cap \text{box } a \ b)$ 
  have Int_stable (range ( $\lambda(a, b). \text{box } a \ b$ ))
    by (auto simp: Int_stable_def box_Int_box)
  moreover have (range ( $\lambda(a, b). \text{box } a \ b$ ))  $\subseteq \text{Pow UNIV}$ 
    by auto
  moreover have  $A \in \text{sigma\_sets UNIV}$  (range ( $\lambda(a, b). \text{box } a \ b$ ))
    using A unfolding borel_eq_box by simp
  ultimately have (( $\lambda x. 1$ ) has_integral ?M A) (A  $\cap \text{box } a \ b$ )
  proof (induction rule: sigma_sets_induct_disjoint)
    case (basic A) then show ?case
      by (auto simp: box_Int_box has_integral_box)
  next
    case empty then show ?case
      by simp
  next
    case (compl A)
    then have [measurable]:  $A \in \text{sets borel}$ 
      by (simp add: borel_eq_box)

    have (( $\lambda x. 1$ ) has_integral ?M (box a b)) (box a b)
      by (simp add: has_integral_box)
    moreover have (( $\lambda x. \text{if } x \in A \cap \text{box } a \ b \text{ then } 1 \text{ else } 0$ ) has_integral ?M A)
      (box a b)
      by (subst has_integral_restrict) (auto intro: compl)
    ultimately have (( $\lambda x. 1 - (\text{if } x \in A \cap \text{box } a \ b \text{ then } 1 \text{ else } 0)$ ) has_integral
      ?M (box a b) - ?M A) (box a b)
      by (rule has_integral_diff)
    then have (( $\lambda x. (\text{if } x \in (\text{UNIV} - A) \cap \text{box } a \ b \text{ then } 1 \text{ else } 0)$ ) has_integral
      ?M (box a b) - ?M A) (box a b)
      by (rule has_integral_cong[THEN iffD1, rotated 1]) auto
    then have (( $\lambda x. 1$ ) has_integral ?M (box a b) - ?M A) ((UNIV - A)  $\cap \text{box}$ 
      a b)
      by (subst (asm) has_integral_restrict) auto
    also have ?M (box a b) - ?M A = ?M (UNIV - A)
      by (subst measure_Diff[symmetric]) (auto simp: emeasure_lborel_box_eq
        Diff_Int_distrib2)
    finally show ?case .
  next

```

```

case (union F)
then have [measurable]:  $\bigwedge i. F\ i \in \text{sets borel}$ 
  by (simp add: borel_eq_box_subset_eq)
  have (( $\lambda x. \text{if } x \in \bigcup (F \text{ ' UNIV}) \cap \text{box } a\ b \text{ then } 1 \text{ else } 0$ ) has_integral ?M
( $\bigcup i. F\ i$ )) (box a b)
proof (rule has_integral_monotone_convergence_increasing)
  let ?f =  $\lambda k\ x. \sum i < k. \text{if } x \in F\ i \cap \text{box } a\ b \text{ then } 1 \text{ else } 0 :: \text{real}$ 
  show  $\bigwedge k. (?f\ k \text{ has\_integral } (\sum i < k. ?M\ (F\ i)))\ (\text{box } a\ b)$ 
    using union.IH by (auto intro!: has_integral_sum simp del: Int_iff)
  show  $\bigwedge k\ x. ?f\ k\ x \leq ?f\ (\text{Suc } k)\ x$ 
    by (intro sum_mono2) auto
  from union(1) have *:  $\bigwedge x\ i\ j. x \in F\ i \implies x \in F\ j \longleftrightarrow j = i$ 
    by (auto simp add: disjoint_family_on_def)
  show ( $\lambda k. ?f\ k\ x \longrightarrow (\text{if } x \in \bigcup (F \text{ ' UNIV}) \cap \text{box } a\ b \text{ then } 1 \text{ else } 0)$  for
x
  by (auto simp: * sum.If_cases Iio_Int_singleton if_distrib LIMSEQ_if_less
cong: if_cong)
  have *:  $\text{emeasure lborel } ((\bigcup x. F\ x) \cap \text{box } a\ b) \leq \text{emeasure lborel } (\text{box } a\ b)$ 
    by (intro emeasure_mono) auto

  with union(1) show ( $\lambda k. \sum i < k. ?M\ (F\ i) \longrightarrow ?M\ (\bigcup i. F\ i)$ )
    unfolding sums_def[symmetric] UN_extend_simps
    by (intro measure_UNION) (auto simp: disjoint_family_on_def emea-
sure_lborel_box_eq top_unique)
  qed
  then show ?case
    by (subst (asm) has_integral_restrict) auto
  qed }
note * = this

show ?thesis
proof (rule has_integral_monotone_convergence_increasing)
  let ?B =  $\lambda n::\text{nat}. \text{box } (-\ \text{real } n\ *_R\ \text{One})\ (\text{real } n\ *_R\ \text{One}) :: 'a\ \text{set}$ 
  let ?f =  $\lambda n::\text{nat}. \lambda x. \text{if } x \in A \cap ?B\ n \text{ then } 1 \text{ else } 0 :: \text{real}$ 
  let ?M =  $\lambda n. \text{measure lborel } (A \cap ?B\ n)$ 

  show  $\bigwedge n::\text{nat}. (?f\ n \text{ has\_integral } ?M\ n)\ A$ 
    using * by (subst has_integral_restrict) simp_all
  show  $\bigwedge k\ x. ?f\ k\ x \leq ?f\ (\text{Suc } k)\ x$ 
    by (auto simp: box_def)
  { fix x assume x ∈ A
  moreover have ( $\lambda k. \text{indicator } (A \cap ?B\ k)\ x :: \text{real} \longrightarrow \text{indicator } (\bigcup k::\text{nat}.
A \cap ?B\ k)\ x$ )
    by (intro LIMSEQ_indicator_incseq) (auto simp: incseq_def box_def)
  ultimately show ( $\lambda k. \text{if } x \in A \cap ?B\ k \text{ then } 1 \text{ else } 0::\text{real} \longrightarrow 1$ )
    by (simp add: indicator_def of_bool_def UN_box_eq_UNIV) }

  have ( $\lambda n. \text{emeasure lborel } (A \cap ?B\ n) \longrightarrow \text{emeasure lborel } (\bigcup n::\text{nat}. A \cap
?B\ n)$ )

```

```

    by (intro Lim_emeasure_incseq) (auto simp: incseq_def box_def)
  also have  $(\lambda n. \text{emeasure } \text{lborel } (A \cap ?B \ n)) = (\lambda n. \text{measure } \text{lborel } (A \cap ?B \ n))$ 
  proof (intro ext_emeasure_eq_ennreal_measure)
    fix n have  $\text{emeasure } \text{lborel } (A \cap ?B \ n) \leq \text{emeasure } \text{lborel } (?B \ n)$ 
    by (intro_emeasure_mono) auto
    then show  $\text{emeasure } \text{lborel } (A \cap ?B \ n) \neq \text{top}$ 
    by (auto simp: top_unique)
  qed
  finally show  $(\lambda n. \text{measure } \text{lborel } (A \cap ?B \ n)) \longrightarrow \text{measure } \text{lborel } A$ 
  using_emeasure_eq_ennreal_measure[of lborel A] finite
  by (simp add: UN_box_eq_UNIV less_top)
qed
qed

lemma nn_integral_has_integral:
  fixes f::'a::euclidean_space  $\Rightarrow$  real
  assumes f:  $f \in \text{borel\_measurable } \text{borel } \wedge x. 0 \leq f \ x \ (\int^+ x. f \ x \ \partial \text{lborel}) = \text{ennreal}$ 
  r 0  $\leq$  r
  shows (f has_integral r) UNIV
using f proof (induct f arbitrary: r rule: borel_measurable_induct_real)
  case (set A)
  then have  $((\lambda x. 1) \text{ has\_integral } \text{measure } \text{lborel } A) \ A$ 
  by (intro_has_integral_measure_lborel) (auto simp: ennreal_indicator)
  with set show ?case
  by (simp add: ennreal_indicator_measure_def) (simp add: indicator_def of_bool_def)
next
  case (mult g c)
  then have  $\text{ennreal } c * (\int^+ x. g \ x \ \partial \text{lborel}) = \text{ennreal } r$ 
  by (subst nn_integral_cmult[symmetric]) (auto simp: ennreal_mult)
  with  $\langle 0 \leq r \rangle \langle 0 \leq c \rangle$ 
  obtain r' where  $(c = 0 \wedge r = 0) \vee (0 \leq r' \wedge (\int^+ x. \text{ennreal } (g \ x) \ \partial \text{lborel}) =$ 
   $\text{ennreal } r' \wedge r = c * r')$ 
  by (cases  $\int^+ x. \text{ennreal } (g \ x) \ \partial \text{lborel}$  rule: ennreal_cases)
  (auto split: if_split_asm simp: ennreal_mult_top ennreal_mult[symmetric])
  with mult show ?case
  by (auto intro!: has_integral_cmult_real)
next
  case (add g h)
  then have  $(\int^+ x. h \ x + g \ x \ \partial \text{lborel}) = (\int^+ x. h \ x \ \partial \text{lborel}) + (\int^+ x. g \ x \ \partial \text{lborel})$ 
  by (simp add: nn_integral_add)
  with add obtain a b where  $0 \leq a \ 0 \leq b \ (\int^+ x. h \ x \ \partial \text{lborel}) = \text{ennreal } a \ (\int^+$ 
   $x. g \ x \ \partial \text{lborel}) = \text{ennreal } b \ r = a + b$ 
  by (cases  $\int^+ x. h \ x \ \partial \text{lborel} \ \int^+ x. g \ x \ \partial \text{lborel}$  rule: ennreal2_cases)
  (auto simp: add_top nn_integral_add top_add simp flip: ennreal_plus)
  with add show ?case
  by (auto intro!: has_integral_add)
next
  case (seq U)
  note seq(1)[measurable] and f[measurable]

```

have $U_le_f: U\ i\ x \leq f\ x$ **for** $i\ x$
by (*metis* (*no_types*) *LIMSEQ_le_const UNIV_I incseq_def le_fun_def seq.hyps(4)*)
seq.hyps(5) space_borel)

{ fix i
have $(\int^{+x}. U\ i\ x\ \partial\text{borel}) \leq (\int^{+x}. f\ x\ \partial\text{borel})$
using *seq(2) f(2) U_le_f* **by** (*intro nn_integral_mono*) *simp*
then obtain p **where** $(\int^{+x}. U\ i\ x\ \partial\text{borel}) = \text{ennreal } p$ $p \leq r$ $0 \leq p$
using *seq(6) <0≤r>* **by** (*cases* $\int^{+x}. U\ i\ x\ \partial\text{borel}$ *rule: ennreal_cases*) (*auto simp: top_unique*)
moreover note *seq*
ultimately have $\exists p. (\int^{+x}. U\ i\ x\ \partial\text{borel}) = \text{ennreal } p \wedge 0 \leq p \wedge p \leq r \wedge (U\ i\ \text{has_integral } p)$ *UNIV*
by *auto* }
then obtain p **where** $p: \bigwedge i. (\int^{+x}. \text{ennreal } (U\ i\ x)\ \partial\text{borel}) = \text{ennreal } (p\ i)$
and $\text{bnd}: \bigwedge i. p\ i \leq r \wedge i. 0 \leq p\ i$
and $U_int: \bigwedge i. (U\ i\ \text{has_integral } (p\ i))$ *UNIV* **by** *metis*

have $\text{int_eq}: \bigwedge i. \text{integral } UNIV\ (U\ i) = p\ i$ **using** U_int **by** (*rule integral_unique*)

have $*$: f *integrable_on UNIV* $\wedge (\lambda k. \text{integral } UNIV\ (U\ k)) \longrightarrow \text{integral } UNIV\ f$

proof (*rule monotone_convergence_increasing*)
show $\bigwedge k. U\ k$ *integrable_on UNIV* **using** U_int **by** *auto*
show $\bigwedge k\ x. x \in UNIV \implies U\ k\ x \leq U\ (\text{Suc } k)\ x$ **using** $\langle \text{incseq } U \rangle$ **by** (*auto simp: incseq_def le_fun_def*)
then show *bounded* ($\text{range } (\lambda k. \text{integral } UNIV\ (U\ k))$)
using bnd int_eq **by** (*auto simp: bounded_real intro!: exI[of _ r]*)
show $\bigwedge x. x \in UNIV \implies (\lambda k. U\ k\ x) \longrightarrow f\ x$
using *seq* **by** *auto*

qed

moreover have $(\lambda i. (\int^{+x}. U\ i\ x\ \partial\text{borel})) \longrightarrow (\int^{+x}. f\ x\ \partial\text{borel})$

using *seq f(2) U_le_f* **by** (*intro nn_integral_dominated_convergence[where w=f]*) *auto*

ultimately have $\text{integral } UNIV\ f = r$

by (*auto simp add: bnd int_eq p seq intro: LIMSEQ_unique*)

with $*$ **show** *?case*

by (*simp add: has_integral_integral*)

qed

lemma *nn_integral_borel_eq_integral*:

fixes $f::'a::\text{euclidean_space} \Rightarrow \text{real}$

assumes $f: f \in \text{borel_measurable borel} \wedge x. 0 \leq f\ x$ $(\int^{+x}. f\ x\ \partial\text{borel}) < \infty$

shows $(\int^{+x}. f\ x\ \partial\text{borel}) = \text{integral } UNIV\ f$

proof –

from *f(3)* **obtain** r **where** $(\int^{+x}. f\ x\ \partial\text{borel}) = \text{ennreal } r$ $0 \leq r$

by (*cases* $\int^{+x}. f\ x\ \partial\text{borel}$ *rule: ennreal_cases*) *auto*

then show *?thesis*

using $nn_integral_has_integral[OF f(1,2) r]$ by (simp add: integral_unique)
qed

lemma $nn_integral_integrable_on$:

fixes $f::'a::euclidean_space \Rightarrow real$

assumes $f: f \in borel_measurable borel \wedge x. 0 \leq f x (\int^+ x. f x \partial lborel) < \infty$

shows f integrable_on UNIV

proof -

from $f(3)$ obtain r where $r: (\int^+ x. f x \partial lborel) = ennreal r \ 0 \leq r$

by (cases $\int^+ x. f x \partial lborel$ rule: ennreal_cases) auto

then show ?thesis

by (intro has_integral_integrable[where $i=r$] $nn_integral_has_integral$ [where $r=r$]) f)

qed

lemma $nn_integral_has_integral_lborel$:

fixes $f :: 'a::euclidean_space \Rightarrow real$

assumes $f_borel: f \in borel_measurable borel$ and nonneg: $\wedge x. 0 \leq f x$

assumes $I: (f$ has_integral $I)$ UNIV

shows $integral^N lborel f = I$

proof -

from f_borel have $(\lambda x. ennreal (f x)) \in borel_measurable lborel$ by auto

from $borel_measurable_implies_simple_function_sequence'[OF this]$

obtain F where $F: \wedge i. simple_function lborel (F i) incseq F$

$\wedge i x. F i x < top \wedge x. (SUP i. F i x) = ennreal (f x)$

by blast

then have $[measurable]: \wedge i. F i \in borel_measurable lborel$

by (metis $borel_measurable_simple_function$)

let $?B = \lambda i::nat. box (- (real i *_R One)) (real i *_R One) :: 'a set$

have $0 \leq I$

using I by (rule has_integral_nonneg) (simp add: nonneg)

have $F_le_f: enn2real (F i x) \leq f x$ for $i x$

using $F(3,4)[where x=x]$ nonneg SUP_upper[of i UNIV $\lambda i. F i x$]

by (cases $F i x$ rule: ennreal_cases) auto

let $?F = \lambda i x. F i x * indicator (?B i) x$

have $(\int^+ x. ennreal (f x) \partial lborel) = (SUP i. integral^N lborel (\lambda x. ?F i x))$

proof (subst $nn_integral_monotone_convergence_SUP[symmetric]$)

{ fix x

obtain j where $j: x \in ?B j$

using $UN_box_eq_UNIV$ by auto

have $ennreal (f x) = (SUP i. F i x)$

using $F(4)[of x]$ nonneg[of x] by (simp add: max_def)

also have $\dots = (SUP i. ?F i x)$

proof (rule SUP_eq)

fix i show $\exists j \in UNIV. F i x \leq ?F j x$

using $j F(2)$

```

      by (intro beXI[of _ max i j])
         (auto split: split_max split_indicator simp: incseq_def le_fun_def
box_def)
    qed (auto intro!: F split: split_indicator)
    finally have ennreal (f x) = (SUP i. ?F i x) . }
    then show (∫+ x. ennreal (f x) ∂lborel) = (∫+ x. (SUP i. ?F i x) ∂lborel)
      by simp
    qed (insert F, auto simp: incseq_def le_fun_def box_def split: split_indicator)
    also have ... ≤ ennreal I
    proof (rule SUP_least)
      fix i :: nat
      have finite_F: (∫+ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
< ∞
    proof (rule nn_integral_bound_simple_function)
      have emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) * indicator
(?B i) x) ≠ 0} ≤
        emeasure lborel (?B i)
      by (intro emeasure_mono) (auto split: split_indicator)
      then show emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) *
indicator (?B i) x) ≠ 0} < ∞
        by (auto simp: less_top[symmetric] top_unique)
    qed (auto split: split_indicator
      intro!: F simple_function_compose1 [where g=enn2real] simple_function_ennreal)

    have int_F: (λx. enn2real (F i x) * indicator (?B i) x) integrable_on UNIV
      using F(4) finite_F
    by (intro nn_integral_integrable_on) (auto split: split_indicator simp: enn2real_nonneg)

    have (∫+ x. F i x * indicator (?B i) x ∂lborel) =
      (∫+ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
      using F(3,4)
    by (intro nn_integral_cong) (auto simp: image_iff eq_commute split: split_indicator)
    also have ... = ennreal (integral UNIV (λx. enn2real (F i x) * indicator (?B
i) x))
      using F
      by (intro nn_integral_lborel_eq_integral[OF __ finite_F])
         (auto split: split_indicator intro: enn2real_nonneg)
    also have ... ≤ ennreal I
      by (auto intro!: has_integral_le[OF integrable_integral[OF int_F] I] nonneg
F_le_f
        simp: <0 ≤ I> split: split_indicator )
    finally show (∫+ x. F i x * indicator (?B i) x ∂lborel) ≤ ennreal I .
    qed
  finally have (∫+ x. ennreal (f x) ∂lborel) < ∞
    by (auto simp: less_top[symmetric] top_unique)
  from nn_integral_lborel_eq_integral[OF assms(1,2) this] I show ?thesis
    by (simp add: integral_unique)
qed

```

```

lemma has_integral_iff_emeasure_lborel:
  fixes A :: 'a::euclidean_space set
  assumes A[measurable]: A ∈ sets borel and [simp]: 0 ≤ r
  shows ((λx. 1) has_integral r) A ↔ emeasure lborel A = ennreal r
proof (cases emeasure lborel A = ∞)
  case emeasure_A: True
  have ¬ (λx. 1::real) integrable_on A
  proof
    assume int: (λx. 1::real) integrable_on A
    then have (indicator A::'a ⇒ real) integrable_on UNIV
      unfolding indicator_def of_bool_def integrable_restrict_UNIV .
    then obtain r where ((indicator A::'a⇒real) has_integral r) UNIV
      by auto
    from nn_integral_has_integral_lborel[OF _ _ this] emeasure_A show False
      by (simp add: ennreal_indicator)
  qed
  with emeasure_A show ?thesis
    by auto
next
  case False
  then have ((λx. 1) has_integral measure lborel A) A
    by (simp add: has_integral_measure_lborel less_top)
  with False show ?thesis
    by (auto simp: emeasure_eq_ennreal_measure has_integral_unique)
qed

lemma ennreal_max_0: ennreal (max 0 x) = ennreal x
  by (auto simp: max_def ennreal_neg)

lemma has_integral_integral_real:
  fixes f::'a::euclidean_space ⇒ real
  assumes f: integrable lborel f
  shows (f has_integral (integralL lborel f)) UNIV
proof -
  from integrableE[OF f] obtain r q
    where 0 ≤ r 0 ≤ q
      and r: (∫+ x. ennreal (max 0 (f x)) ∂lborel) = ennreal r
      and q: (∫+ x. ennreal (max 0 (- f x)) ∂lborel) = ennreal q
      and f: f ∈ borel_measurable lborel and eq: integralL lborel f = r - q
  unfolding ennreal_max_0 by auto
  then have ((λx. max 0 (f x)) has_integral r) UNIV ((λx. max 0 (- f x))
has_integral q) UNIV
  using nn_integral_has_integral[OF _ _ r] nn_integral_has_integral[OF _ _
q] by auto
  note has_integral_diff[OF this]
  moreover have (λx. max 0 (f x) - max 0 (- f x)) = f
    by auto
  ultimately show ?thesis
    by (simp add: eq)

```

2920

qed

lemma *has_integral_AE*:

assumes *ae*: $AE\ x\ in\ lborel.\ x \in \Omega \longrightarrow f\ x = g\ x$

shows $(f\ has_integral\ x)\ \Omega = (g\ has_integral\ x)\ \Omega$

proof –

from *ae* **obtain** *N*

where $N: N \in sets\ borel\ emeasure\ lborel\ N = 0\ \{x.\ \neg(x \in \Omega \longrightarrow f\ x = g\ x)\}$
 $\subseteq N$

by (*auto elim!*: *AE_E*)

then have *not_N*: $AE\ x\ in\ lborel.\ x \notin N$

by (*simp add*: *AE_iff_measurable*)

show *?thesis*

proof (*rule has_integral_spike_eq[symmetric]*)

show $\bigwedge x.\ x \in \Omega - N \implies f\ x = g\ x$ **using** *N(3)* **by** *auto*

show *negligible N*

unfolding *negligible_def*

proof (*intro allI*)

fix *a b* :: 'a

let $?F = \lambda x::'a.\ if\ x \in\ cbox\ a\ b\ then\ indicator\ N\ x\ else\ 0 :: real$

have *integrable lborel ?F = integrable lborel ($\lambda x::'a.\ 0::real$)*

using *not_N N(1)* **by** (*intro integrable_cong_AE*) *auto*

moreover have $(LINT\ x|lborel.\ ?F\ x) = (LINT\ x::'a|lborel.\ 0::real)$

using *not_N N(1)* **by** (*intro integral_cong_AE*) *auto*

ultimately have $(?F\ has_integral\ 0)\ UNIV$

using *has_integral_integral_real[of ?F]* **by** *simp*

then show $(indicator\ N\ has_integral\ (0::real))\ (cbox\ a\ b)$

unfolding *has_integral_restrict_UNIV* .

qed

qed

qed

lemma *nn_integral_has_integral_lebesgue*:

fixes *f* :: 'a::euclidean_space $\Rightarrow real$

assumes *nonneg*: $\bigwedge x.\ x \in \Omega \implies 0 \leq f\ x$ **and** *I*: $(f\ has_integral\ I)\ \Omega$

shows $integral^N\ lborel\ (\lambda x.\ indicator\ \Omega\ x * f\ x) = I$

proof –

from *I* **have** $(\lambda x.\ indicator\ \Omega\ x *_{\mathbb{R}}\ f\ x) \in lebesgue \rightarrow_M\ borel$

by (*rule has_integral_implies_lebesgue_measurable*)

then obtain *f'* :: 'a $\Rightarrow real$

where [*measurable*]: $f' \in borel \rightarrow_M\ borel$ **and** *eq*: $AE\ x\ in\ lborel.\ indicator\ \Omega\ x * f\ x = f'\ x$

by (*auto dest: completion_ex_borel_measurable_real*)

from *I* **have** $((\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x))\ has_integral\ I)\ UNIV$

using *nonneg* **by** (*simp add: indicator_def of_bool_def if_distrib[of $\lambda x.\ x * f\ y$ for *y*] cong: if_cong*)

also have $((\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x))\ has_integral\ I)\ UNIV \longleftrightarrow ((\lambda x.\ abs\ (f'\ x))\ has_integral\ I)\ UNIV$

```

  using eq by (intro has_integral_AE) auto
  finally have integralN lborel ( $\lambda x. \text{abs } (f' x) = I$ )
    by (rule nn_integral_has_integral_lborel[rotated 2]) auto
  also have integralN lborel ( $\lambda x. \text{abs } (f' x) = \text{integral}^N \text{lborel } (\lambda x. \text{abs } (\text{indicator } \Omega x * f x))$ )
    using eq by (intro nn_integral_cong_AE) auto
  also have ( $\lambda x. \text{abs } (\text{indicator } \Omega x * f x) = (\lambda x. \text{indicator } \Omega x * f x)$ )
    using nonneg by (auto simp: indicator_def fun_eq_iff)
  finally show ?thesis .
qed

```

lemma has_integral_iff_nn_integral_lebesgue:

```

  assumes f:  $\bigwedge x. 0 \leq f x$ 
  shows (f has_integral r) UNIV  $\longleftrightarrow$  (f  $\in$  lebesgue  $\rightarrow_M$  borel  $\wedge$  integralN lebesgue
  f = r  $\wedge$   $0 \leq r$ ) (is ?I = ?N)

```

proof

```

  assume ?I
  have  $0 \leq r$ 
  using has_integral_nonneg[OF  $\langle ?I \rangle$ ] f by auto
  then show ?N
  using nn_integral_has_integral_lebesgue[OF f  $\langle ?I \rangle$ ]
    has_integral_implies_lebesgue_measurable[OF  $\langle ?I \rangle$ ]
  by (auto simp: nn_integral_completion)

```

next

```

  assume ?N
  then obtain f' where f': f'  $\in$  borel  $\rightarrow_M$  borel AE x in lborel. f x = f' x
  by (auto dest: completion_ex_borel_measurable_real)
  moreover have ( $\int^+ x. \text{ennreal } |f' x| \partial \text{lborel}$ ) = ( $\int^+ x. \text{ennreal } |f x| \partial \text{lborel}$ )
  using f' by (intro nn_integral_cong_AE) auto
  moreover have (( $\lambda x. |f' x|$ ) has_integral r) UNIV  $\longleftrightarrow$  (( $\lambda x. |f x|$ ) has_integral
  r) UNIV
  using f' by (intro has_integral_AE) auto
  moreover note nn_integral_has_integral[of  $\lambda x. |f' x| r$ ]  $\langle ?N \rangle$ 
  ultimately show ?I
  using f by (auto simp: nn_integral_completion)

```

qed

lemma set_nn_integral_lborel_eq_integral:

```

  fixes f::'a::euclidean_space  $\Rightarrow$  real
  assumes set_borel_measurable borel A f
  assumes  $\bigwedge x. x \in A \implies 0 \leq f x$  ( $\int^+ x \in A. f x \partial \text{lborel}$ )  $< \infty$ 
  shows ( $\int^+ x \in A. f x \partial \text{lborel}$ ) = integral A f

```

proof -

```

  have eq: ( $\int^+ x \in A. f x \partial \text{lborel}$ ) = ( $\int^+ x. \text{ennreal } (\text{indicator } A x * f x) \partial \text{lborel}$ )
  by (intro nn_integral_cong) (auto simp: indicator_def)
  also have ... = integral UNIV ( $\lambda x. \text{indicator } A x * f x$ )
  using asms eq by (intro nn_integral_lborel_eq_integral)
  (auto simp: indicator_def set_borel_measurable_def)
  also have integral UNIV ( $\lambda x. \text{indicator } A x * f x$ ) = integral A ( $\lambda x. \text{indicator } A$ 

```

2922

$x * f x$
by (rule *integral_spike_set*) (auto intro: *empty_imp_negligible*)

also have $\dots = \text{integral } A f$
by (rule *integral_cong*) (auto simp: *indicator_def*)
finally show ?thesis .

qed

lemma *nn_integral_has_integral_lebesgue'*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow \text{real}$
assumes *nonneg*: $\bigwedge x. x \in \Omega \implies 0 \leq f x$ and $I: (f \text{ has_integral } I) \Omega$
shows *integral^N lborel* $(\lambda x. \text{ennreal } (f x) * \text{indicator } \Omega x) = \text{ennreal } I$

proof -

have *integral^N lborel* $(\lambda x. \text{ennreal } (f x) * \text{indicator } \Omega x) =$
integral^N lborel $(\lambda x. \text{ennreal } (\text{indicator } \Omega x * f x))$

by (intro *nn_integral_cong*) (auto simp: *indicator_def*)

also have $\dots = \text{ennreal } I$

using *assms* by (intro *nn_integral_has_integral_lebesgue*)

finally show ?thesis .

qed

context

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$

begin

lemma *has_integral_integral_lborel*:

assumes $f: \text{integrable lborel } f$

shows $(f \text{ has_integral } (\text{integral}^L \text{ lborel } f)) \text{ UNIV}$

proof -

have $((\lambda x. \sum_{b \in \text{Basis}} (f x \cdot b) *_R b) \text{ has_integral } (\sum_{b \in \text{Basis}} \text{integral}^L \text{ lborel } (\lambda x. f x \cdot b) *_R b)) \text{ UNIV}$

using f by (intro *has_integral_sum_finite_Basis_ballI* *has_integral_scaleR_left* *has_integral_integral_real*) auto

also have *eq_f*: $(\lambda x. \sum_{b \in \text{Basis}} (f x \cdot b) *_R b) = f$

by (*simp add: fun_eq_iff euclidean_representation*)

also have $(\sum_{b \in \text{Basis}} \text{integral}^L \text{ lborel } (\lambda x. f x \cdot b) *_R b) = \text{integral}^L \text{ lborel } f$

using f by (*subst* (2) *eq_f[symmetric]*) *simp*

finally show ?thesis .

qed

lemma *integrable_on_lborel*: $\text{integrable lborel } f \implies f \text{ integrable_on UNIV}$

using *has_integral_integral_lborel* by auto

lemma *integral_lborel*: $\text{integrable lborel } f \implies \text{integral UNIV } f = (\int x. f x \partial \text{lborel})$

using *has_integral_integral_lborel* by auto

end

context

begin

private lemma *has_integral_integral_lebesgue_real*:

fixes $f :: 'a::euclidean_space \Rightarrow \text{real}$

assumes f : *integrable lebesgue* f

shows $(f \text{ has_integral } (\text{integral}^L \text{ lebesgue } f)) \text{ UNIV}$

proof –

obtain f' **where** $f': f' \in \text{borel} \rightarrow_M \text{borel } AE \text{ } x \text{ in } \text{lborel}. f \ x = f' \ x$

using *completion_ex_borel_measurable_real*[*OF borel_measurable_integrable*[*OF f*]] **by** *auto*

moreover have $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial \text{lborel}) = (\int^+ x. \text{ennreal } (\text{norm } (f' \ x)) \ \partial \text{lborel})$

using f' **by** (*intro nn_integral_cong_AE*) *auto*

ultimately have *integrable lborel* f'

using f **by** (*auto simp: integrable_iff_bounded nn_integral_completion cong: nn_integral_cong_AE*)

note *has_integral_integral_real*[*OF this*]

moreover have *integral^L lebesgue* $f = \text{integral}^L \text{ lebesgue } f'$

using $f' \ f$ **by** (*intro integral_cong_AE*) (*auto intro: AE_completion_measurable_completion*)

moreover have *integral^L lebesgue* $f' = \text{integral}^L \text{ lborel } f'$

using f' **by** (*simp add: integral_completion*)

moreover have $(f' \text{ has_integral } \text{integral}^L \text{ lborel } f') \text{ UNIV} \longleftrightarrow (f \text{ has_integral } \text{integral}^L \text{ lborel } f') \text{ UNIV}$

using f' **by** (*intro has_integral_AE*) *auto*

ultimately show *?thesis*

by *auto*

qed

lemma *has_integral_integral_lebesgue*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes f : *integrable lebesgue* f

shows $(f \text{ has_integral } (\text{integral}^L \text{ lebesgue } f)) \text{ UNIV}$

proof –

have $((\lambda x. \sum_{b \in \text{Basis}} (f \ x \cdot b) *_R b) \text{ has_integral } (\sum_{b \in \text{Basis}} \text{integral}^L \text{ lebesgue } (\lambda x. f \ x \cdot b) *_R b)) \text{ UNIV}$

using f **by** (*intro has_integral_sum_finite_Basis_ballI has_integral_scaleR_left has_integral_integral_lebesgue_real*) *auto*

also have *eq_f*: $(\lambda x. \sum_{b \in \text{Basis}} (f \ x \cdot b) *_R b) = f$

by (*simp add: fun_eq_iff euclidean_representation*)

also have $(\sum_{b \in \text{Basis}} \text{integral}^L \text{ lebesgue } (\lambda x. f \ x \cdot b) *_R b) = \text{integral}^L \text{ lebesgue } f$

using f **by** (*subst (2) eq_f[symmetric]*) *simp*

finally show *?thesis* .

qed

lemma *has_integral_integral_lebesgue_on*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes *integrable (lebesgue_on S) f* $S \in \text{sets lebesgue}$

```

  shows (f has_integral (integralL (lebesgue_on S) f)) S
proof -
  let ?f =  $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ 
  have integrable_lebesgue ( $\lambda x. \text{indicat\_real } S x *_{\mathbb{R}} f x$ )
    using indicator_scaleR_eq_if [of S f] assms
  by (metis (full_types) integrable_restrict_space sets.Int_space_eq2)
  then have integrable_lebesgue ?f
    using indicator_scaleR_eq_if [of S f] assms by auto
  then have (?f has_integral (integralL lebesgue ?f)) UNIV
    by (rule has_integral_integral_lebesgue)
  then have (f has_integral (integralL lebesgue ?f)) S
    using has_integral_restrict_UNIV by blast
  moreover
  have  $S \cap \text{space lebesgue} \in \text{sets lebesgue}$ 
    by (simp add: assms)
  then have (integralL lebesgue ?f) = (integralL (lebesgue_on S) f)
    by (simp add: integral_restrict_space indicator_scaleR_eq_if)
  ultimately show ?thesis
    by auto
qed

```

```

lemma lebesgue_integral_eq_integral:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes integrable (lebesgue_on S) f  $S \in \text{sets lebesgue}$ 
  shows integralL (lebesgue_on S) f = integral S f
  by (metis has_integral_integral_lebesgue_on assms integral_unique)

```

```

lemma integrable_on_lebesgue:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows integrable lebesgue f  $\implies f \text{ integrable\_on UNIV}$ 
  using has_integral_integral_lebesgue by auto

```

```

lemma integral_lebesgue:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows integrable lebesgue f  $\implies \text{integral UNIV } f = (\int x. f x \partial \text{lebesgue})$ 
  using has_integral_integral_lebesgue by auto

```

end

9.8.8 Absolute integrability (this is the same as Lebesgue integrability)

translations

```
LBINT x. f == CONST lebesgue_integral CONST lborel ( $\lambda x. f$ )
```

translations

```
LBINT x:A. f == CONST set_lebesgue_integral CONST lborel A ( $\lambda x. f$ )
```

lemma set_integral_reflect:


```

fixes  $S$  and  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
shows  $(\text{LBINT } x : S. f x) = (\text{LBINT } x : \{x. -x \in S\}. f (-x))$ 
unfolding  $\text{set\_lebesgue\_integral\_def}$ 
by  $(\text{subst } \text{lborel\_integral\_real\_affine}[\text{where } c=-1 \text{ and } t=0])$ 
 $(\text{auto intro!: Bochner\_Integration.integral\_cong split: split\_indicator})$ 

```

lemma $\text{borel_integrable_atLeastAtMost}'$:

```

fixes  $f :: \text{real} \Rightarrow 'a :: \{\text{banach, second\_countable\_topology}\}$ 
assumes  $f: \text{continuous\_on } \{a..b\} f$ 
shows  $\text{set\_integrable } \text{lborel } \{a..b\} f$ 
unfolding  $\text{set\_integrable\_def}$ 
by  $(\text{intro } \text{borel\_integrable\_compact } \text{compact\_Icc } f)$ 

```

lemma $\text{integral_FTC_atLeastAtMost}$:

```

fixes  $f :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$ 
assumes  $a \leq b$ 
and  $F: \bigwedge x. a \leq x \implies x \leq b \implies (F \text{ has\_vector\_derivative } f x)$   $(\text{at } x \text{ within } \{a..b\})$ 
and  $f: \text{continuous\_on } \{a..b\} f$ 
shows  $\text{integral}^L \text{lborel } (\lambda x. \text{indicator } \{a..b\} x *_R f x) = F b - F a$ 
proof -
let  $?f = \lambda x. \text{indicator } \{a..b\} x *_R f x$ 
have  $(?f \text{ has\_integral } (\int x. ?f x \partial \text{lborel})) \text{ UNIV}$ 
using  $\text{borel\_integrable\_atLeastAtMost}[OF f]$ 
unfolding  $\text{set\_integrable\_def}$  by  $(\text{rule } \text{has\_integral\_integral\_lborel})$ 
moreover
have  $(f \text{ has\_integral } F b - F a) \{a..b\}$ 
by  $(\text{intro } \text{fundamental\_theorem\_of\_calculus } \text{ballI } \text{assms}) \text{ auto}$ 
then have  $(?f \text{ has\_integral } F b - F a) \{a..b\}$ 
by  $(\text{subst } \text{has\_integral\_cong}[\text{where } g=f]) \text{ auto}$ 
then have  $(?f \text{ has\_integral } F b - F a) \text{ UNIV}$ 
by  $(\text{intro } \text{has\_integral\_on\_superset}[\text{where } T=\text{UNIV} \text{ and } S=\{a..b\}]) \text{ auto}$ 
ultimately show  $\text{integral}^L \text{lborel } ?f = F b - F a$ 
by  $(\text{rule } \text{has\_integral\_unique})$ 
qed

```

lemma $\text{set_borel_integral_eq_integral}$:

```

fixes  $f :: 'a :: \text{euclidean\_space} \Rightarrow 'b :: \text{euclidean\_space}$ 
assumes  $\text{set\_integrable } \text{lborel } S f$ 
shows  $f \text{ integrable\_on } S (\text{LINT } x : S \mid \text{lborel. } f x) = \text{integral } S f$ 
proof -
let  $?f = \lambda x. \text{indicator } S x *_R f x$ 
have  $(?f \text{ has\_integral } (\text{LINT } x : S \mid \text{lborel. } f x)) \text{ UNIV}$ 
using  $\text{assms } \text{has\_integral\_integral\_lborel}$ 
unfolding  $\text{set\_integrable\_def } \text{set\_lebesgue\_integral\_def}$  by  $\text{blast}$ 
hence  $1: (f \text{ has\_integral } (\text{set\_lebesgue\_integral } \text{lborel } S f)) S$ 
by  $(\text{simp add: indicator\_scaleR\_eq\_if})$ 
thus  $f \text{ integrable\_on } S$ 
by  $(\text{auto simp add: integrable\_on\_def})$ 

```

```

with 1 have (f has_integral (integral S f)) S
  by (intro integrable_integral, auto simp add: integrable_on_def)
thus (LINT x : S | lborel. f x) = integral S f
  by (intro has_integral_unique [OF 1])
qed

```

```

lemma has_integral_set_lebesgue:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: set_integrable lebesgue S f
  shows (f has_integral (LINT x:S|lebesgue. f x)) S
  using has_integral_integral_lebesgue f
  by (fastforce simp add: set_integrable_def set_lebesgue_integral_def indicator_def
    of_bool_def if_distrib[of  $\lambda x. x *_R f \_$ ] cong: if_cong)

```

```

lemma set_lebesgue_integral_eq_integral:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: set_integrable lebesgue S f
  shows f integrable_on S (LINT x:S | lebesgue. f x) = integral S f
  using has_integral_set_lebesgue[OF f] by (auto simp: integral_unique integrable_on_def)

```

```

lemma lmeasurable_iff_has_integral:
   $S \in \text{lmeasurable} \iff ((\text{indicator } S) \text{ has\_integral } \text{measure lebesgue } S) \text{ UNIV}$ 
  by (subst has_integral_iff_nn_integral_lebesgue)
  (auto simp: ennreal_indicator_emeasure_eq_measure2 borel_masurable_indicator_iff
  intro!: fmeasurableI)

```

abbreviation

```

absolutely_integrable_on :: ('a::euclidean_space  $\Rightarrow$  'b::{banach, second_countable_topology})
 $\Rightarrow$  'a set  $\Rightarrow$  bool
(infixr absolutely_integrable'_on 46)
where f absolutely_integrable_on s  $\equiv$  set_integrable lebesgue s f

```

```

lemma absolutely_integrable_zero [simp]: ( $\lambda x. 0$ ) absolutely_integrable_on S
  by (simp add: set_integrable_def)

```

```

lemma absolutely_integrable_on_def:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows f absolutely_integrable_on S  $\iff$  f integrable_on S  $\wedge$  ( $\lambda x. \text{norm } (f x)$ )
  integrable_on S
proof safe
  assume f: f absolutely_integrable_on S
  then have nf: integrable lebesgue ( $\lambda x. \text{norm } (\text{indicator } S x *_R f x)$ )
    using integrable_norm set_integrable_def by blast
  show f integrable_on S
    by (rule set_lebesgue_integral_eq_integral[OF f])
  have ( $\lambda x. \text{norm } (\text{indicator } S x *_R f x)$ ) = ( $\lambda x. \text{if } x \in S \text{ then } \text{norm } (f x) \text{ else } 0$ )
    by auto

```

```

with integrable_on_lebesgue[OF nf] show ( $\lambda x. \text{norm } (f x)$ ) integrable_on S
  by (simp add: integrable_restrict_UNIV)
next
assume f: f integrable_on S and nf: ( $\lambda x. \text{norm } (f x)$ ) integrable_on S
show f absolutely_integrable_on S
  unfolding set_integrable_def
proof (rule integrableI_bounded)
  show ( $\lambda x. \text{indicator } S x *_{\mathbb{R}} f x$ )  $\in$  borel_measurable_lebesgue
    using f has_integral_implies_lebesgue_measurable[of f _ S] by (auto simp:
integrable_on_def)
  show ( $\int^+ x. \text{ennreal } (\text{norm } (\text{indicator } S x *_{\mathbb{R}} f x)) \partial \text{lebesgue}$ )  $< \infty$ 
    using nf nn_integral_has_integral_lebesgue[of _  $\lambda x. \text{norm } (f x)$ ]
    by (auto simp: integrable_on_def nn_integral_completion)
  qed
qed

```

```

lemma integrable_on_lebesgue_on:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: integrable (lebesgue_on S) f and S: S  $\in$  sets_lebesgue
  shows f integrable_on S
proof -
  have integrable_lebesgue ( $\lambda x. \text{indicator } S x *_{\mathbb{R}} f x$ )
    using S f inf_top.comm_neutral_integrable_restrict_space by blast
  then show ?thesis
    using absolutely_integrable_on_def set_integrable_def by blast
qed

```

```

lemma absolutely_integrable_imp_integrable:
  assumes f absolutely_integrable_on S S  $\in$  sets_lebesgue
  shows integrable (lebesgue_on S) f
  by (meson assms integrable_restrict_space set_integrable_def sets.Int sets.top)

```

```

lemma absolutely_integrable_on_null [intro]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  shows content (cbox a b) = 0  $\implies$  f absolutely_integrable_on (cbox a b)
  by (auto simp: absolutely_integrable_on_def)

```

```

lemma absolutely_integrable_on_open_interval:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  shows f absolutely_integrable_on box a b  $\longleftrightarrow$ 
    f absolutely_integrable_on cbox a b
  by (auto simp: integrable_on_open_interval absolutely_integrable_on_def)

```

```

lemma absolutely_integrable_restrict_UNIV:
  ( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) absolutely_integrable_on UNIV  $\longleftrightarrow$  f absolutely_integrable_on S
  unfolding set_integrable_def
  by (intro arg_cong2[where f=integrable]) auto

```

```

lemma absolutely_integrable_onI:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows  $f \text{ integrable\_on } S \implies (\lambda x. \text{norm } (f\ x)) \text{ integrable\_on } S \implies f \text{ absolutely\_integrable\_on } S$ 
  unfolding absolutely_integrable_on_def by auto

lemma nonnegative_absolutely_integrable_1:
  fixes  $f :: 'a :: euclidean\_space \Rightarrow \text{real}$ 
  assumes  $f: f \text{ integrable\_on } A$  and  $\bigwedge x. x \in A \implies 0 \leq f\ x$ 
  shows  $f \text{ absolutely\_integrable\_on } A$ 
  by (rule absolutely_integrable_onI [OF  $f$ ]) (use assms in  $\langle \text{simp add: integrable\_eq} \rangle$ )

lemma absolutely_integrable_on_iff_nonneg:
  fixes  $f :: 'a :: euclidean\_space \Rightarrow \text{real}$ 
  assumes  $\bigwedge x. x \in S \implies 0 \leq f\ x$  shows  $f \text{ absolutely\_integrable\_on } S \longleftrightarrow f \text{ integrable\_on } S$ 
  proof -
    { assume  $f \text{ integrable\_on } S$ 
      then have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ integrable\_on UNIV}$ 
        by (simp add: integrable_restrict_UNIV)
      then have  $(\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
        using  $\langle f \text{ integrable\_on } S \rangle$  absolutely_integrable_restrict_UNIV assms nonnegative_absolutely_integrable_1 by blast
      then have  $f \text{ absolutely\_integrable\_on } S$ 
        using absolutely_integrable_restrict_UNIV by blast
    }
    then show ?thesis
      unfolding absolutely_integrable_on_def by auto
  qed

lemma absolutely_integrable_on_scaleR_iff:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  shows
     $(\lambda x. c *_{\mathbb{R}} f\ x) \text{ absolutely\_integrable\_on } S \longleftrightarrow$ 
     $c = 0 \vee f \text{ absolutely\_integrable\_on } S$ 
  proof (cases  $c=0$ )
    case False
    then show ?thesis
      unfolding absolutely_integrable_on_def
      by (simp add: norm_mult)
  qed auto

lemma absolutely_integrable_spike:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \text{ absolutely\_integrable\_on } T$  and  $S: \text{negligible } S \bigwedge x. x \in T - S \implies$ 
 $g\ x = f\ x$ 
  shows  $g \text{ absolutely\_integrable\_on } T$ 
  using assms integrable_spike
  unfolding absolutely_integrable_on_def by metis

```

lemma *absolutely_integrable_negligible*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes *negligible* S
shows f *absolutely_integrable_on* S
using *assms* **by** (*simp* *add*: *absolutely_integrable_on_def* *integrable_negligible*)

lemma *absolutely_integrable_spike_eq*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes *negligible* $S \wedge x. x \in T - S \implies g\ x = f\ x$
shows (f *absolutely_integrable_on* $T \longleftrightarrow g$ *absolutely_integrable_on* T)
using *assms* **by** (*blast* *intro*: *absolutely_integrable_spike_sym*)

lemma *absolutely_integrable_spike_set_eq*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes *negligible* $\{x \in S - T. f\ x \neq 0\}$ *negligible* $\{x \in T - S. f\ x \neq 0\}$
shows (f *absolutely_integrable_on* $S \longleftrightarrow f$ *absolutely_integrable_on* T)

proof -

have ($\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0$) *absolutely_integrable_on* $UNIV \longleftrightarrow$
 $(\lambda x. \text{if } x \in T \text{ then } f\ x \text{ else } 0)$ *absolutely_integrable_on* $UNIV$

proof (*rule* *absolutely_integrable_spike_eq*)

show *negligible* $(\{x \in S - T. f\ x \neq 0\} \cup \{x \in T - S. f\ x \neq 0\})$

by (*rule* *negligible_Un* [*OF* *assms*])

qed *auto*

with *absolutely_integrable_restrict_UNIV* **show** *?thesis*

by *blast*

qed

lemma *absolutely_integrable_spike_set*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes f : f *absolutely_integrable_on* S **and** *neg*: *negligible* $\{x \in S - T. f\ x \neq 0\}$ *negligible* $\{x \in T - S. f\ x \neq 0\}$
shows f *absolutely_integrable_on* T
using *absolutely_integrable_spike_set_eq* f *neg* **by** *blast*

lemma *absolutely_integrable_reflect[simp]*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
shows ($\lambda x. f(-x)$) *absolutely_integrable_on* $cbox\ (-b)\ (-a) \longleftrightarrow f$ *absolutely_integrable_on* $cbox\ a\ b$
unfolding *absolutely_integrable_on_def*
by (*metis* (*mono_tags*, *lifting*) *integrable_eq* *integrable_reflect*)

lemma *absolutely_integrable_reflect_real[simp]*:

fixes $f :: \text{real} \Rightarrow 'b::euclidean_space$
shows ($\lambda x. f(-x)$) *absolutely_integrable_on* $\{-b .. -a\} \longleftrightarrow f$ *absolutely_integrable_on* $\{a..b::\text{real}\}$
unfolding *box_real[symmetric]* **by** (*rule* *absolutely_integrable_reflect*)

lemma *absolutely_integrable_on_subcbox*:

2930

```
fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
shows  $\llbracket f \text{ absolutely\_integrable\_on } S; \text{ cbox } a \ b \subseteq S \rrbracket \Longrightarrow f \text{ absolutely\_integrable\_on } \text{cbox } a \ b$ 
by (meson absolutely\_integrable\_on\_def integrable\_on\_subcbox)
```

```
lemma absolutely\_integrable\_on\_subinterval:
fixes f :: real  $\Rightarrow$  'b::euclidean_space
shows  $\llbracket f \text{ absolutely\_integrable\_on } S; \{a..b\} \subseteq S \rrbracket \Longrightarrow f \text{ absolutely\_integrable\_on } \{a..b\}$ 
using absolutely\_integrable\_on\_subcbox by fastforce
```

```
lemma integrable\_subinterval:
fixes f :: real  $\Rightarrow$  'a::euclidean_space
assumes integrable (lebesgue\_on {a..b}) f
and {c..d}  $\subseteq$  {a..b}
shows integrable (lebesgue\_on {c..d}) f
proof (rule absolutely\_integrable\_imp\_integrable)
show f absolutely\_integrable\_on {c..d}
proof -
have f integrable\_on {c..d}
using assms integrable\_on\_lebesgue\_on integrable\_on\_subinterval by fastforce
moreover have  $(\lambda x. \text{norm } (f \ x)) \text{ integrable\_on } \{c..d\}$ 
proof (rule integrable\_on\_subinterval)
show  $(\lambda x. \text{norm } (f \ x)) \text{ integrable\_on } \{a..b\}$ 
by (simp add: assms integrable\_on\_lebesgue\_on)
qed (use assms in auto)
ultimately show ?thesis
by (auto simp: absolutely\_integrable\_on\_def)
qed
qed auto
```

```
lemma indefinite\_integral\_continuous\_real:
fixes f :: real  $\Rightarrow$  'a::euclidean_space
assumes integrable (lebesgue\_on {a..b}) f
shows continuous\_on {a..b}  $(\lambda x. \text{integral}^L (\text{lebesgue\_on } \{a..x\}) f)$ 
proof -
have f integrable\_on {a..b}
by (simp add: assms integrable\_on\_lebesgue\_on)
then have continuous\_on {a..b}  $(\lambda x. \text{integral } \{a..x\} f)$ 
using indefinite\_integral\_continuous\_1 by blast
moreover have  $\text{integral}^L (\text{lebesgue\_on } \{a..x\}) f = \text{integral } \{a..x\} f$  if  $a \leq x \leq b$  for x
proof -
have  $\{a..x\} \subseteq \{a..b\}$ 
using that by auto
then have integrable (lebesgue\_on {a..x}) f
using integrable\_subinterval assms by blast
then show  $\text{integral}^L (\text{lebesgue\_on } \{a..x\}) f = \text{integral } \{a..x\} f$ 
```

```

    by (simp add: lebesgue_integral_eq_integral)
  qed
  ultimately show ?thesis
    by (metis (no_types, lifting) atLeastAtMost_iff continuous_on_cong)
  qed

lemma lmeasurable_iff_integrable_on:  $S \in \text{lmeasurable} \longleftrightarrow (\lambda x. 1::\text{real}) \text{ integrable\_on } S$ 
  by (subst absolutely_integrable_on_iff_nonneg[symmetric])
    (simp_all add: lmeasurable_iff_integrable_set_integrable_def)

lemma lmeasure_integral_UNIV:  $S \in \text{lmeasurable} \implies \text{measure lebesgue } S = \text{integral UNIV (indicator } S)$ 
  by (simp add: lmeasurable_iff_has_integral integral_unique)

lemma lmeasure_integral:  $S \in \text{lmeasurable} \implies \text{measure lebesgue } S = \text{integral } S$ 
  ( $\lambda x. 1::\text{real}$ )
  by (fastforce simp add: lmeasure_integral_UNIV indicator_def [abs_def] of_bool_def
    lmeasurable_iff_integrable_on)

lemma integrable_on_const:  $S \in \text{lmeasurable} \implies (\lambda x. c) \text{ integrable\_on } S$ 
  unfolding lmeasurable_iff_integrable
  by (metis (mono_tags, lifting) integrable_eq integrable_on_scaleR_left lmeasurable_iff_integrable
    lmeasurable_iff_integrable_on scaleR_one)

lemma integral_indicator:
  assumes  $(S \cap T) \in \text{lmeasurable}$ 
  shows  $\text{integral } T \text{ (indicator } S) = \text{measure lebesgue } (S \cap T)$ 
  proof -
    have  $\text{integral UNIV (indicator } (S \cap T)) = \text{integral UNIV } (\lambda a. \text{if } a \in S \cap T \text{ then } 1::\text{real} \text{ else } 0)$ 
    by (simp add: indicator_def [abs_def] of_bool_def)
    moreover have  $(\text{indicator } (S \cap T) \text{ has\_integral measure lebesgue } (S \cap T))$ 
    UNIV
    using assms by (simp add: lmeasurable_iff_has_integral)
    ultimately have  $\text{integral UNIV } (\lambda x. \text{if } x \in S \cap T \text{ then } 1 \text{ else } 0) = \text{measure lebesgue } (S \cap T)$ 
    by (metis (no_types) integral_unique)
    moreover have  $\text{integral } T \text{ (indicator } S) = \text{integral } (S \cap T \cap \text{UNIV}) \text{ (indicator } S)$ 
    by (simp add: Henstock_Kurzweil_Integration.integral_restrict_Int)
    moreover have  $\text{integral } T \text{ (indicator\_real } S) = \text{integral } T \text{ (indicator } S \text{ then } 1 \text{ else } 0)$ 
    by (simp add: indicator_def [abs_def] of_bool_def)
    ultimately show ?thesis
    by (simp add: assms lmeasure_integral)
  qed

lemma measurable_integrable:

```

fixes $S :: 'a::\text{euclidean_space set}$
shows $S \in \text{lmeasurable} \longleftrightarrow (\text{indicat_real } S) \text{ integrable_on UNIV}$
by (*auto simp: lmeasurable_iff_integrable absolutely_integrable_on_iff_nonneg [symmetric] set_integrable_def*)

lemma integrable_on_indicator:
fixes $S :: 'a::\text{euclidean_space set}$
shows $\text{indicat_real } S \text{ integrable_on } T \longleftrightarrow (S \cap T) \in \text{lmeasurable}$
unfolding *measurable_integrable*
unfolding *integrable_restrict_UNIV [of T, symmetric]*
by (*fastforce simp add: indicator_def elim: integrable_eq*)

lemma
assumes $\mathcal{D}: \mathcal{D} \text{ division_of } S$
shows *lmeasurable_division: $S \in \text{lmeasurable}$ (is ?l)*
and *content_division: $(\sum k \in \mathcal{D}. \text{measure lebesgue } k) = \text{measure lebesgue } S$ (is ?m)*
proof –
{ **fix** $d1\ d2$ **assume** $*$: $d1 \in \mathcal{D}\ d2 \in \mathcal{D}\ d1 \neq d2$
then obtain $a\ b\ c\ d$ **where** $d1 = \text{cbox } a\ b\ d2 = \text{cbox } c\ d$
using *division_ofD(4)[OF \mathcal{D}]* **by** *blast*
with *division_ofD(5)[OF \mathcal{D} $*$]*
have $d1 \in \text{sets lborel}\ d2 \in \text{sets lborel}\ d1 \cap d2 \subseteq (\text{cbox } a\ b - \text{box } a\ b) \cup (\text{cbox } c\ d - \text{box } c\ d)$
by *auto*
moreover have $(\text{cbox } a\ b - \text{box } a\ b) \cup (\text{cbox } c\ d - \text{box } c\ d) \in \text{null_sets lborel}$
by (*intro null_sets.Un null_sets_cbox_Diff_box*)
ultimately have $d1 \cap d2 \in \text{null_sets lborel}$
by (*blast intro: null_sets_subset*) **}**
then show *?l ?m*
unfolding *division_ofD(6)[OF \mathcal{D} , symmetric]*
using *division_ofD(1,4)[OF \mathcal{D}]*
by (*auto intro!: measure_Union_AE[symmetric] simp: completion.AE_iff_null_sets Int_def[symmetric] pairwise_def null_sets_def*)
qed

lemma has_measure_limit:
assumes $S \in \text{lmeasurable}\ e > 0$
obtains B **where** $B > 0$
 $\bigwedge a\ b. \text{ball } 0\ B \subseteq \text{cbox } a\ b \implies |\text{measure lebesgue } (S \cap \text{cbox } a\ b) - \text{measure lebesgue } S| < e$
using *assms* **unfolding** *lmeasurable_iff_has_integral has_integral_alt'*
by (*force simp: integral_indicator integrable_on_indicator*)

lemma lmeasurable_iff_indicator_has_integral:
fixes $S :: 'a::\text{euclidean_space set}$
shows $S \in \text{lmeasurable} \wedge m = \text{measure lebesgue } S \longleftrightarrow (\text{indicat_real } S \text{ has_integral } m) \text{ UNIV}$
by (*metis has_integral_iff_lmeasurable_iff_has_integral measurable_integrable*)


```

lemma has_measure_limit_iff:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'a::banach$ 
  shows  $S \in \text{lmeasurable} \wedge m = \text{measure lebesgue } S \longleftrightarrow$ 
     $(\forall e > 0. \exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
       $(S \cap \text{cbox } a b) \in \text{lmeasurable} \wedge |\text{measure lebesgue } (S \cap \text{cbox } a b) - m|$ 
     $< e)$  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (meson has_measure_limit fmeasurable.Int lmeasurable_cbox)
next
  assume RHS [rule_format]: ?rhs
  then show ?lhs
    apply (simp add: lmeasurable_iff_indicator_has_integral has_integral' [where
i=m])
    by (metis (full_types) integral_indicator integrable_integral integrable_on_indicator)
qed

```

9.8.9 Applications to Negligibility

```

lemma negligible_iff_null_sets:  $\text{negligible } S \longleftrightarrow S \in \text{null\_sets lebesgue}$ 
proof
  assume negligible S
  then have (indicator S has_integral (0::real)) UNIV
    by (auto simp: negligible)
  then show  $S \in \text{null\_sets lebesgue}$ 
    by (subst (asm) has_integral_iff_nn_integral_lebesgue)
    (auto simp: borel_measurable_indicator_iff nn_integral_0_iff_AE AE_iff_null_sets
indicator_eq_0_iff)
next
  assume  $S: S \in \text{null\_sets lebesgue}$ 
  show negligible S
    unfolding negligible_def
  proof (safe intro!: has_integral_iff_nn_integral_lebesgue[THEN iffD2]
    has_integral_restrict_UNIV[where s=cbox ___, THEN iffD1])
    fix  $a b$ 
    show  $(\lambda x. \text{if } x \in \text{cbox } a b \text{ then indicator } S x \text{ else } 0) \in \text{lebesgue} \rightarrow_M \text{borel}$ 
      using  $S$  by (auto intro!: measurable>If)
    then show  $(\int^+ x. \text{ennreal } (\text{if } x \in \text{cbox } a b \text{ then indicator } S x \text{ else } 0) \partial \text{lebesgue})$ 
     $= \text{ennreal } 0$ 
      using  $S$  [THEN AE_not_in] by (auto intro!: nn_integral_0_iff_AE[THEN
iffD2])
    qed auto
  qed

```

corollary *eventually_ae_filter_negligible*:

$\text{eventually } P \text{ (ae_filter lebesgue)} \longleftrightarrow (\exists N. \text{negligible } N \wedge \{x. \neg P x\} \subseteq N)$

by (*auto simp: completion.AE_iff_null_sets negligible_iff_null_sets null_sets_completion_subset*)

lemma *starlike_negligible*:

assumes *closed S*

and *eq1*: $\bigwedge c x. (a + c *_R x) \in S \implies 0 \leq c \implies a + x \in S \implies c = 1$

shows *negligible S*

proof –

have *negligible ((+) (-a) ‘ S)*

proof (*subst negligible_on_intervals, intro allI*)

fix *u v*

show *negligible ((+) (- a) ‘ S \cap cbox u v)*

using \langle *closed S* \rangle *eq1* **by** (*auto simp add: negligible_iff_null_sets algebra_simps*)

intro!: *closed_translation_subtract starlike_negligible_compact cong: image_cong_simp*)

(*metis add_diff_eq diff_add_cancel scale_right_diff_distrib*)

qed

then show *?thesis*

by (*rule negligible_translation_rev*)

qed

lemma *starlike_negligible_strong*:

assumes *closed S*

and *star*: $\bigwedge c x. [0 \leq c; c < 1; a + x \in S] \implies a + c *_R x \notin S$

shows *negligible S*

proof –

show *?thesis*

proof (*rule starlike_negligible [OF \langle closed S \rangle , of a]*)

fix *c x*

assume *cx*: $a + c *_R x \in S$ $0 \leq c$ $a + x \in S$

with *star* **have** $\neg (c < 1)$ **by** *auto*

moreover **have** $\neg (c > 1)$

using *star [of 1/c c *_R x] cx* **by** *force*

ultimately **show** $c = 1$ **by** *arith*

qed

qed

lemma *negligible_hyperplane*:

assumes $a \neq 0 \vee b \neq 0$ **shows** *negligible {x. a \cdot x = b}*

proof –

obtain *x* **where** $x: a \cdot x \neq b$

using *assms* **by** (*metis euclidean_all_zero_iff inner_zero_right*)

moreover **have** $c = 1$ **if** $a \cdot (x + c *_R w) = b$ $a \cdot (x + w) = b$ **for** $c w$

using *that*

by (*metis (no_types, opaque_lifting) add_diff_eq diff_0 diff_minus_eq_add inner_diff_right inner_scaleR_right mult_cancel_right2 right_minus_eq x*)

ultimately

show *?thesis*

using *starlike_negligible [OF closed_hyperplane, of x]* **by** *simp*

qed

```

lemma negligible_lowdim:
  fixes S :: 'N :: euclidean_space set
  assumes dim S < DIM('N)
  shows negligible S
proof -
  obtain a where a ≠ 0 and a: span S ⊆ {x. a · x = 0}
  using lowdim_subset_hyperplane [OF assms] by blast
  have negligible (span S)
  using ⟨a ≠ 0⟩ a negligible_hyperplane by (blast intro: negligible_subset)
  then show ?thesis
  using span_base by (blast intro: negligible_subset)
qed

proposition negligible_convex_frontier:
  fixes S :: 'N :: euclidean_space set
  assumes convex S
  shows negligible(frontier S)
proof -
  have nf: negligible(frontier S) if convex S 0 ∈ S for S :: 'N set
  proof -
    obtain B where B ⊆ S and indB: independent B
      and spanB: S ⊆ span B and cardB: card B = dim S
      by (metis basis_exists)
    consider dim S < DIM('N) | dim S = DIM('N)
    using dim_subset_UNIV le_eq_less_or_eq by auto
    then show ?thesis
  proof cases
    case 1
    show ?thesis
      by (rule negligible_subset [of closure S])
      (simp_all add: frontier_def negligible_lowdim 1)
    next
    case 2
    obtain a where a ∈ interior (convex hull insert 0 B)
    proof (rule interior_simplex_nonempty [OF indB])
      show finite B
        by (simp add: indB independent_imp_finite)
      show card B = DIM('N)
        by (simp add: cardB 2)
    qed
    then have a: a ∈ interior S
      by (metis ⟨B ⊆ S⟩ ⟨0 ∈ S⟩ ⟨convex S⟩ insert_absorb insert_subset interior_mono subset_hull)
    show ?thesis
  proof (rule starlike_negligible_strong [where a=a])
    fix c::real and x
    have eq: a + c *R x = (a + x) - (1 - c) *R ((a + x) - a)
      by (simp add: algebra_simps)
    assume 0 ≤ c c < 1 a + x ∈ frontier S
  
```

```

then show  $a + c *_R x \notin \text{frontier } S$ 
  using  $\text{eq mem\_interior\_closure\_convex\_shrink } [OF \langle \text{convex } S \rangle a, \text{ of } \_$ 
1-c]
  unfolding  $\text{frontier\_def}$ 
  by ( $\text{metis Diff\_iff add\_diff\_cancel\_left' add\_diff\_eq diff\_gt\_0\_iff\_gt}$ 
 $\text{group\_cancel.rule0 not\_le}$ )
  qed auto
qed
qed
show  $?thesis$ 
proof ( $\text{cases } S = \{\}$ )
  case  $\text{True}$  then show  $?thesis$  by  $\text{auto}$ 
next
  case  $\text{False}$ 
  then obtain  $a$  where  $a \in S$  by  $\text{auto}$ 
  show  $?thesis$ 
  using  $\text{nf } [of (\lambda x. -a + x) ' S]$ 
  by ( $\text{metis } \langle a \in S \rangle \text{add.left\_inverse assms convex\_translation\_eq frontier\_translation}$ 
 $\text{image\_eqI negligible\_translation\_rev}$ )
  qed
qed

```

```

corollary  $\text{negligible\_sphere: negligible (sphere } a \ e)$ 
  using  $\text{frontier\_cball negligible\_convex\_frontier convex\_cball}$ 
  by ( $\text{blast intro: negligible\_subset}$ )

```

```

lemma  $\text{non\_negligible\_UNIV } [simp]: \neg \text{negligible UNIV}$ 
  unfolding  $\text{negligible\_iff\_null\_sets}$  by ( $\text{auto simp: null\_sets\_def}$ )

```

```

lemma  $\text{negligible\_interval:}$ 
 $\text{negligible (cbox } a \ b) \iff \text{box } a \ b = \{\}$   $\text{negligible (box } a \ b) \iff \text{box } a \ b = \{\}$ 
  by ( $\text{auto simp: negligible\_iff\_null\_sets null\_sets\_def prod\_nonneg inner\_diff\_left}$ 
 $\text{box\_eq\_empty}$ 
 $\text{not\_le emeasure\_lborel\_cbox\_eq emeasure\_lborel\_box\_eq}$ 
 $\text{intro: eq\_refl antisym less\_imp\_le}$ )

```

```

proposition  $\text{open\_not\_negligible:}$ 

```

```

  assumes  $\text{open } S \ S \neq \{\}$ 
  shows  $\neg \text{negligible } S$ 

```

```

proof

```

```

  assume  $\text{neg: negligible } S$ 

```

```

  obtain  $a$  where  $a \in S$ 

```

```

  using  $\langle S \neq \{\} \rangle$  by  $\text{blast}$ 

```

```

  then obtain  $e$  where  $e > 0$   $\text{cball } a \ e \subseteq S$ 

```

```

  using  $\langle \text{open } S \rangle \text{open\_contains\_cball\_eq}$  by  $\text{blast}$ 

```

```

  let  $?p = a - (e / \text{DIM}('a)) *_R \text{One}$  let  $?q = a + (e / \text{DIM}('a)) *_R \text{One}$ 

```

```

  have  $\text{cbox } ?p \ ?q \subseteq \text{cball } a \ e$ 

```

```

  proof ( $\text{clarsimp simp: mem\_box dist\_norm}$ )

```

```

    fix  $x$ 

```

```

assume  $\forall i \in \text{Basis}. ?p \cdot i \leq x \cdot i \wedge x \cdot i \leq ?q \cdot i$ 
then have  $ax: |(a - x) \cdot i| \leq e / \text{real } \text{DIM}('a)$  if  $i \in \text{Basis}$  for  $i$ 
  using that by (auto simp: algebra_simps)
have  $\text{norm } (a - x) \leq (\sum i \in \text{Basis}. |(a - x) \cdot i|)$ 
  by (rule norm_le_l1)
also have  $\dots \leq \text{DIM}('a) * (e / \text{real } \text{DIM}('a))$ 
  by (intro sum_bounded_above ax)
also have  $\dots = e$ 
  by simp
finally show  $\text{norm } (a - x) \leq e$  .
qed
then have negligible (cbox  $?p$   $?q$ )
  by (meson  $\langle \text{cball } a \ e \subseteq S \rangle$  neg negligible_subset)
with  $\langle e > 0 \rangle$  show False
  by (simp add: negligible_interval box_eq_empty algebra_simps field_split_simps
mult_le_0_iff)
qed

```

lemma *negligible_convex_interior*:

```

 $\text{convex } S \implies (\text{negligible } S \longleftrightarrow \text{interior } S = \{\})$ 
by (metis Diff_empty closure_subset frontier_def interior_subset negligible_convex_frontier
negligible_subset open_interior open_not_negligible)

```

lemma *measure_eq_0_null_sets*: $S \in \text{null_sets } M \implies \text{measure } M \ S = 0$

by (*auto simp: measure_def null_sets_def*)

lemma *negligible_imp_measure0*: $\text{negligible } S \implies \text{measure lebesgue } S = 0$

by (*simp add: measure_eq_0_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_emeasure0*: $S \in \text{sets lebesgue} \implies (\text{negligible } S \longleftrightarrow \text{emeasure lebesgue } S = 0)$

by (*auto simp: measure_eq_0_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_measure0*: $S \in \text{lmeasurable} \implies (\text{negligible } S \longleftrightarrow \text{measure lebesgue } S = 0)$

by (*metis* (*full_types*) *completion.null_sets_outer negligible_iff_null_sets negligible_imp_measure0 order_refl*)

lemma *negligible_imp_sets*: $\text{negligible } S \implies S \in \text{sets lebesgue}$

by (*simp add: negligible_iff_null_sets null_setsD2*)

lemma *negligible_imp_measurable*: $\text{negligible } S \implies S \in \text{lmeasurable}$

by (*simp add: fmeasurableI_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_measure*: $\text{negligible } S \longleftrightarrow S \in \text{lmeasurable} \wedge \text{measure lebesgue } S = 0$

by (*fastforce simp: negligible_iff_measure0 negligible_imp_measurable dest: negligible_imp_measure0*)

lemma *negligible_outer*:

$negligible\ S \longleftrightarrow (\forall e > 0. \exists T. S \subseteq T \wedge T \in lmeasurable \wedge measure\ lebesgue\ T < e)$ (**is** $_ = ?rhs$)

proof

assume *negligible S then show ?rhs*

by (*metis negligible_iff_measure_order_refl*)

next

assume *?rhs then show negligible S*

by (*meson completion.null_sets_outer negligible_iff_null_sets*)

qed

lemma *negligible_outer_le*:

$negligible\ S \longleftrightarrow (\forall e > 0. \exists T. S \subseteq T \wedge T \in lmeasurable \wedge measure\ lebesgue\ T \leq e)$ (**is** $_ = ?rhs$)

proof

assume *negligible S then show ?rhs*

by (*metis dual_order.strict_implies_order negligible_imp_measurable negligible_imp_measure0_order_refl*)

next

assume *?rhs then show negligible S*

by (*metis le_less_trans negligible_outer_field_lbound_gt_zero*)

qed

lemma *negligible_UNIV*: $negligible\ S \longleftrightarrow (indicat_real\ S\ has_integral\ 0)$ UNIV (**is** $_ = ?rhs$)

by (*metis lmeasurable_iff_indicator_has_integral negligible_iff_measure*)

lemma *sets_negligible_symdiff*:

$\llbracket S \in sets\ lebesgue; negligible((S - T) \cup (T - S)) \rrbracket \implies T \in sets\ lebesgue$

by (*metis Diff_Diff_Int Int_Diff_Un inf_commute negligible_Un_eq negligible_imp_sets sets.Diff sets.Un*)

lemma *lmeasurable_negligible_symdiff*:

$\llbracket S \in lmeasurable; negligible((S - T) \cup (T - S)) \rrbracket \implies T \in lmeasurable$

using *integrable_spike_set_eq lmeasurable_iff_integrable_on* **by** *blast*

lemma *measure_Un3_negligible*:

assumes *meas: S ∈ lmeasurable T ∈ lmeasurable U ∈ lmeasurable*

and *neg: negligible(S ∩ T) negligible(S ∩ U) negligible(T ∩ U)* **and** *V: S ∪ T ∪ U = V*

shows *measure lebesgue V = measure lebesgue S + measure lebesgue T + measure lebesgue U*

proof –

have [*simp*]: *measure lebesgue (S ∩ T) = 0*

using *neg(1) negligible_imp_measure0* **by** *blast*

have [*simp*]: *measure lebesgue (S ∩ U ∪ T ∩ U) = 0*

using *neg(2) neg(3) negligible_Un negligible_imp_measure0* **by** *blast*

have *measure lebesgue V = measure lebesgue (S ∪ T ∪ U)*

using V **by** *simp*
also have $\dots = \text{measure lebesgue } S + \text{measure lebesgue } T + \text{measure lebesgue } U$
by (*simp add: measure_Un3 meas fmeasurable.Un Int_Un_distrib2*)
finally show *?thesis* .
qed

lemma *measure_translate_add*:
assumes *meas*: $S \in \text{lmeasurable } T \in \text{lmeasurable}$
and U : $S \cup ((+)a ' T) = U$ **and** *neg*: *negligible*($S \cap ((+)a ' T)$)
shows $\text{measure lebesgue } S + \text{measure lebesgue } T = \text{measure lebesgue } U$
proof –
have [*simp*]: $\text{measure lebesgue } (S \cap (+) a ' T) = 0$
using *neg negligible_imp_measure0* **by** *blast*
have $\text{measure lebesgue } (S \cup ((+)a ' T)) = \text{measure lebesgue } S + \text{measure lebesgue } T$
by (*simp add: measure_Un3 meas measurable_translation measure_translation fmeasurable.Un*)
then show *?thesis*
using U **by** *auto*
qed

lemma *measure_negligible_syndiff*:
assumes S : $S \in \text{lmeasurable}$
and *neg*: *negligible* ($S - T \cup (T - S)$)
shows $\text{measure lebesgue } T = \text{measure lebesgue } S$
proof –
have $\text{measure lebesgue } (S - T) = 0$
using *neg negligible_Un_eq negligible_imp_measure0* **by** *blast*
then show *?thesis*
by (*metis S Un_commute add.right_neutral lmeasurable_negligible_syndiff measure_Un2 neg negligible_Un_eq negligible_imp_measure0*)
qed

lemma *measure_closure*:
assumes *bounded S* **and** *neg*: *negligible* (*frontier S*)
shows $\text{measure lebesgue } (\text{closure } S) = \text{measure lebesgue } S$
proof –
have $\text{measure lebesgue } (\text{frontier } S) = 0$
by (*metis neg negligible_imp_measure0*)
then show *?thesis*
by (*metis assms lmeasurable_iff_integrable_on eq_iff_diff_eq_0 has_integral_interior integrable_on_def integral_unique lmeasurable_interior lmeasure_integral measure_frontier*)
qed

lemma *measure_interior*:
 $\llbracket \text{bounded } S; \text{negligible}(\text{frontier } S) \rrbracket \implies \text{measure lebesgue } (\text{interior } S) = \text{measure lebesgue } S$
using *measure_closure measure_frontier negligible_imp_measure0* **by** *fastforce*

lemma *measurable_Jordan*:

assumes *bounded S and neg: negligible (frontier S)*

shows $S \in \text{lmeasurable}$

proof –

have $\text{closure } S \in \text{lmeasurable}$

by (*metis lmeasurable_closure <bounded S>*)

moreover have $\text{interior } S \in \text{lmeasurable}$

by (*simp add: lmeasurable_interior <bounded S>*)

moreover have $\text{interior } S \subseteq S$

by (*simp add: interior_subset*)

ultimately show *?thesis*

using *assms* **by** (*metis (full_types) closure_subset completion.complete_sets_sandwich_fmeasurable measure_closure measure_interior*)

qed

lemma *measurable_convex*: $\llbracket \text{convex } S; \text{bounded } S \rrbracket \implies S \in \text{lmeasurable}$

by (*simp add: measurable_Jordan negligible_convex_frontier*)

lemma *content_cball_conv_ball*: $\text{content } (\text{cball } c \ r) = \text{content } (\text{ball } c \ r)$

proof –

have $\text{ball } c \ r - \text{cball } c \ r \cup (\text{cball } c \ r - \text{ball } c \ r) = \text{sphere } c \ r$

by *auto*

hence $\text{measure lebesgue } (\text{cball } c \ r) = \text{measure lebesgue } (\text{ball } c \ r)$

using *negligible_sphere[of c r]* **by** (*intro measure_negligible_syndiff simp_all*)

thus *?thesis* **by** *simp*

qed

9.8.10 Negligibility of image under non-injective linear map

lemma *negligible_Union_nat*:

assumes $\bigwedge n::\text{nat}. \text{negligible}(S \ n)$

shows $\text{negligible}(\bigcup n. S \ n)$

proof –

have $\text{negligible } (\bigcup m \leq k. S \ m)$ **for** *k*

using *assms* **by** *blast*

then have *0*: $\text{integral UNIV } (\text{indicat_real } (\bigcup m \leq k. S \ m)) = 0$

and *1*: $(\text{indicat_real } (\bigcup m \leq k. S \ m)) \text{ integrable_on UNIV for } k$

by (*auto simp: negligible_has_integral_iff*)

have *2*: $\bigwedge k \ x. \text{indicat_real } (\bigcup m \leq k. S \ m) \ x \leq (\text{indicat_real } (\bigcup m \leq \text{Suc } k. S \ m) \ x)$

by (*auto simp add: indicator_def*)

have *3*: $\bigwedge x. (\bigwedge k. \text{indicat_real } (\bigcup m \leq k. S \ m) \ x) \longrightarrow (\text{indicat_real } (\bigcup n. S \ n) \ x)$

by (*force simp: indicator_def eventually_sequentially intro: tendsto_eventually*)

have *4*: $\text{bounded } (\text{range } (\lambda k. \text{integral UNIV } (\text{indicat_real } (\bigcup m \leq k. S \ m))))$

by (*simp add: 0*)

have ***: $\text{indicat_real } (\bigcup n. S \ n) \text{ integrable_on UNIV } \wedge$

$(\lambda k. \text{integral UNIV } (\text{indicat_real } (\bigcup m \leq k. S \ m))) \longrightarrow (\text{integral UNIV } (\text{indicat_real } (\bigcup n. S \ n)))$


```

(indicat_real ( $\bigcup n. S n$ ))
  by (intro monotone_convergence_increasing 1 2 3 4)
  then have integral_UNIV (indicat_real ( $\bigcup n. S n$ )) = (0::real)
    using LIMSEQ_unique by (auto simp: 0)
  then show ?thesis
    using * by (simp add: negligible_UNIV_has_integral_iff)
qed

```

```

lemma negligible_linear_singular_image:
  fixes f :: 'n::euclidean_space  $\Rightarrow$  'n
  assumes linear f  $\neg$  inj f
  shows negligible (f ' S)
proof -
  obtain a where a  $\neq$  0  $\wedge$  S. f ' S  $\subseteq$  {x. a  $\cdot$  x = 0}
    using assms linear_singular_image_hyperplane by blast
  then show negligible (f ' S)
    by (metis negligible_hyperplane negligible_subset)
qed

```

```

lemma measure_negligible_finite_Union:
  assumes finite  $\mathcal{F}$ 
  and meas:  $\bigwedge S. S \in \mathcal{F} \implies S \in$  lmeasurable
  and djointish: pairwise ( $\lambda S T. negligible (S \cap T)$ )  $\mathcal{F}$ 
  shows measure_lebesgue ( $\bigcup \mathcal{F}$ ) = ( $\sum S \in \mathcal{F}. measure_lebesgue S$ )
  using assms
proof (induction)
  case empty
  then show ?case
    by auto
next
  case (insert S  $\mathcal{F}$ )
  then have S  $\in$  lmeasurable  $\bigcup \mathcal{F} \in$  lmeasurable pairwise ( $\lambda S T. negligible (S \cap T)$ )  $\mathcal{F}$ 
    by (simp_all add: fmeasurable.finite_Union insert.hyps(1) insert.premis(1) pairwise_insert subsetI)
  then show ?case
  proof (simp add: measure_Un3 insert)
    have *:  $\bigwedge T. T \in (\cap) S ' \mathcal{F} \implies negligible T$ 
      using insert by (force simp: pairwise_def)
    have negligible(S  $\cap \bigcup \mathcal{F}$ )
      unfolding Int_Union
      by (rule negligible_Union) (simp_all add: * insert.hyps(1))
    then show measure_lebesgue (S  $\cap \bigcup \mathcal{F}$ ) = 0
      using negligible_imp_measure0 by blast
  qed
qed

```

```

lemma measure_negligible_finite_Union_image:

```

```

assumes finite S
  and meas:  $\bigwedge x. x \in S \implies f x \in lmeasurable$ 
  and djointish: pairwise  $(\lambda x y. negligible (f x \cap f y)) S$ 
shows measure lebesgue  $(\bigcup (f ` S)) = (\sum_{x \in S}. measure lebesgue (f x))$ 
proof -
  have measure lebesgue  $(\bigcup (f ` S)) = sum (measure lebesgue) (f ` S)$ 
  using assms by (auto simp: pairwise_mono pairwise_image intro: measure_negligible_finite_Union)
  also have ... = sum (measure lebesgue  $\circ$  f) S
  using djointish [unfolded pairwise_def] by (metis inf.idem negligible_imp_measure0
sum.reindex_nontrivial [OF <finite S>])
  also have ... =  $(\sum_{x \in S}. measure lebesgue (f x))$ 
  by simp
  finally show ?thesis .
qed

```

9.8.11 Negligibility of a Lipschitz image of a negligible set

The bound will be eliminated by a sort of onion argument

lemma *locally_Lipschitz_negl_bounded:*

fixes $f :: 'M::euclidean_space \Rightarrow 'N::euclidean_space$

assumes $M \leq N$: $DIM('M) \leq DIM('N)$ $0 < B$ bounded S negligible S

and $lips$: $\bigwedge x. x \in S$

$\implies \exists T. open T \wedge x \in T \wedge$

$(\forall y \in S \cap T. norm(f y - f x) \leq B * norm(y - x))$

shows negligible $(f ` S)$

unfolding negligible_iff_null_sets

proof (clarsimp simp: completion.null_sets_outer)

fix $e::real$

assume $0 < e$

have $S \in lmeasurable$

using $\langle negligible S \rangle$ **by** (simp add: negligible_iff_null_sets fmeasurableI_null_sets)

then have $S \in sets lebesgue$

by blast

have $e22$: $0 < e/2 / (2 * B * real DIM('M)) \wedge DIM('N)$

using $\langle 0 < e \rangle \langle 0 < B \rangle$ **by** (simp add: field_split_simps)

obtain T **where** $open T$ $S \subseteq T$ $(T - S) \in lmeasurable$

$measure lebesgue (T - S) < e/2 / (2 * B * DIM('M)) \wedge DIM('N)$

using sets_lebesgue_outer_open [OF $\langle S \in sets lebesgue \rangle$ $e22$]

by (metis emeasure_eq_measure2 ennreal_leI linorder_not_le)

then have T : $measure lebesgue T \leq e/2 / (2 * B * DIM('M)) \wedge DIM('N)$

using $\langle negligible S \rangle$ **by** (simp add: measure_Diff_null_set negligible_iff_null_sets)

have $\exists r. 0 < r \wedge r \leq 1/2 \wedge$

$(x \in S \longrightarrow (\forall y. norm(y - x) < r$

$\longrightarrow y \in T \wedge (y \in S \longrightarrow norm(f y - f x) \leq B * norm(y - x))))$

for x

proof (cases $x \in S$)

case True

obtain U **where** $open U$ $x \in U$ **and** U : $\bigwedge y. y \in S \cap U \implies norm(f y - f x)$

$\leq B * norm(y - x)$

```

    using lips [OF ‹x ∈ S›] by auto
  have x ∈ T ∩ U
    using ‹S ⊆ T› ‹x ∈ U› ‹x ∈ S› by auto
  then obtain ε where 0 < ε ball x ε ⊆ T ∩ U
    by (metis ‹open T› ‹open U› openE open_Int)
  then show ?thesis
    by (rule_tac x=min (1/2) ε in exI) (simp add: U dist_norm norm_minus_commute
subset_iff)
  next
    case False
  then show ?thesis
    by (rule_tac x=1/4 in exI) auto
qed
then obtain R where R12: ∧x. 0 < R x ∧ R x ≤ 1/2
  and RT: ∧x y. [x ∈ S; norm(y - x) < R x] ⇒ y ∈ T
  and RB: ∧x y. [x ∈ S; y ∈ S; norm(y - x) < R x] ⇒ norm(f y -
f x) ≤ B * norm(y - x)
  by metis+
then have gaugeR: gauge (λx. ball x (R x))
  by (simp add: gauge_def)
obtain c where c: S ⊆ cbox (-c *R One) (c *R One) box (-c *R One:: 'M) (c
*R One) ≠ {}
proof -
  obtain B where B: ∧x. x ∈ S ⇒ norm x ≤ B
    using ‹bounded S› bounded_iff by blast
  show ?thesis
  proof (rule_tac c = abs B + 1 in that)
    show S ⊆ cbox (- (|B| + 1) *R One) ((|B| + 1) *R One)
      using norm_bound_Basis_le Basis_le_norm
      by (fastforce simp: mem_box dest!: B intro: order_trans)
    show box (- (|B| + 1) *R One) ((|B| + 1) *R One) ≠ {}
      by (simp add: box_eq_empty(1))
  qed
qed
obtain D where countable D
  and Dsub: ∪ D ⊆ cbox (-c *R One) (c *R One)
  and cbox: ∧K. K ∈ D ⇒ interior K ≠ {} ∧ (∃ c d. K = cbox c d)
  and pw: pairwise (λA B. interior A ∩ interior B = {}) D
  and Ksub: ∧K. K ∈ D ⇒ ∃ x ∈ S ∩ K. K ⊆ (λx. ball x (R x)) x
  and exN: ∧u v. cbox u v ∈ D ⇒ ∃ n. ∀ i ∈ Basis. v · i - u · i = (2*c) /
2n
  and S ⊆ ∪ D
  using covering_lemma [OF c gaugeR] by force
have ∃ u v z. K = cbox u v ∧ box u v ≠ {} ∧ z ∈ S ∧ z ∈ cbox u v ∧
  cbox u v ⊆ ball z (R z) if K ∈ D for K
proof -
  obtain u v where K = cbox u v
    using ‹K ∈ D› cbox by blast
  with that show ?thesis

```

by (*metis Int_iff interior_cbox cbox Ksub*)
qed
then obtain $uf\ vf\ zf$
where $uvz: \bigwedge K. K \in \mathcal{D} \implies$
 $K = cbox\ (uf\ K)\ (vf\ K) \wedge cbox\ (uf\ K)\ (vf\ K) \neq \{\} \wedge zf\ K \in S \wedge$
 $zf\ K \in cbox\ (uf\ K)\ (vf\ K) \wedge cbox\ (uf\ K)\ (vf\ K) \subseteq ball\ (zf\ K)\ (R\ (zf$
 $K))$
 by *metis*
define *prj1* **where** $prj1 \equiv \lambda x::'M. x \cdot (SOME\ i. i \in Basis)$
define *fbx* **where** $fbx \equiv \lambda D. cbox\ (f\ (zf\ D) - (B * DIM('M) * (prj1\ (vf\ D - uf$
 $D))) *_{R}\ One::'N)$
 $(f\ (zf\ D) + (B * DIM('M) * prj1\ (vf\ D - uf\ D))) *_{R}$
 $One)$
have $vu_pos: 0 < prj1\ (vf\ X - uf\ X)$ **if** $X \in \mathcal{D}$ **for** X
using uvz [*OF that*] **by** (*simp add: prj1_def box_ne_empty SOME_Basis*
inner_diff_left)
have $prj1_idem: prj1\ (vf\ X - uf\ X) = (vf\ X - uf\ X) \cdot i$ **if** $X \in \mathcal{D}$ $i \in Basis$
for $X\ i$
proof -
have $cbox\ (uf\ X)\ (vf\ X) \in \mathcal{D}$
using uvz $\langle X \in \mathcal{D} \rangle$ **by** *auto*
with exN **obtain** n **where** $\bigwedge i. i \in Basis \implies vf\ X \cdot i - uf\ X \cdot i = (2*c) /$
 2^n
by *blast*
then show *?thesis*
by (*simp add: \langle i \in Basis \rangle SOME_Basis inner_diff prj1_def*)
qed
have *countbl: countable* $(fbx\ 'D)$
using $\langle countable\ \mathcal{D} \rangle$ **by** *blast*
have $(\sum k \in fbx\ 'D'. measure\ lebesgue\ k) \leq e/2$ **if** $\mathcal{D}' \subseteq \mathcal{D}$ *finite* \mathcal{D}' **for** \mathcal{D}'
proof -
have $BM_ge0: 0 \leq B * (DIM('M) * prj1\ (vf\ X - uf\ X))$ **if** $X \in \mathcal{D}'$ **for** X
using $\langle 0 < B \rangle \langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ *that* vu_pos **by** *fastforce*
have $\{\} \notin \mathcal{D}'$
using $cbox\ \langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ *interior_empty* **by** *blast*
have $(\sum k \in fbx\ 'D'. measure\ lebesgue\ k) \leq sum\ (measure\ lebesgue\ o\ fbx)\ \mathcal{D}'$
by (*rule sum_image_le* [*OF \langle finite \mathcal{D}' \rangle*]) (*force simp: fbx_def*)
also have $\dots \leq (\sum X \in \mathcal{D}'. (2 * B * DIM('M)) \wedge DIM('N) * measure\ lebesgue$
 $X)$
proof (*rule sum_mono*)
fix X **assume** $X \in \mathcal{D}'$
then have $X \in \mathcal{D}$ **using** $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle$ **by** *blast*
then have $ufvf: cbox\ (uf\ X)\ (vf\ X) = X$
using uvz **by** *blast*
have $prj1\ (vf\ X - uf\ X) \wedge DIM('M) = (\prod i::'M \in Basis. prj1\ (vf\ X - uf$
 $X))$
by (*rule prod_constant* [*symmetric*])
also have $\dots = (\prod i \in Basis. vf\ X \cdot i - uf\ X \cdot i)$
by (*simp add: \langle X \in \mathcal{D} \rangle inner_diff_left prj1_idem cong: prod.cong*)

```

finally have prj1_eq: prj1 (vf X - uf X) ^ DIM('M) = (∏ i∈Basis. vf X ·
i - uf X · i) .
have uf X ∈ cbox (uf X) (vf X) vf X ∈ cbox (uf X) (vf X)
  using uvz [OF ‹X ∈ D›] by (force simp: mem_box)+
moreover have cbox (uf X) (vf X) ⊆ ball (zf X) (1/2)
  by (meson R12 order_trans subset_ball uvz [OF ‹X ∈ D›])
ultimately have uf X ∈ ball (zf X) (1/2) vf X ∈ ball (zf X) (1/2)
  by auto
then have dist (vf X) (uf X) ≤ 1
  unfolding mem_ball
  by (metis dist_commute dist_triangle_half_l dual_order.order_iff_strict)
then have 1: prj1 (vf X - uf X) ≤ 1
  unfolding prj1_def dist_norm using Basis_le_norm SOME_Basis_order_trans
by fastforce
  have 0: 0 ≤ prj1 (vf X - uf X)
    using ‹X ∈ D› prj1_def vu_pos by fastforce
  have (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N) * content
(cbox (uf X) (vf X))
    apply (simp add: fbx_def content_cbox_cases algebra_simps BM_ge0 ‹X
∈ D› ‹0 < B› flip: prj1_eq)
    using MleN 0 1 uvz ‹X ∈ D›
    by (fastforce simp add: box_ne_empty power_decreasing)
  also have ... = (2 * B * DIM('M)) ^ DIM('N) * measure lebesgue X
    by (subst (3) ufuf[symmetric]) simp
  finally show (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N)
* measure lebesgue X .
qed
also have ... = (2 * B * DIM('M)) ^ DIM('N) * sum (measure lebesgue) D'
  by (simp add: sum_distrib_left)
also have ... ≤ e/2
proof -
  have ∧K. K ∈ D' ⇒ ∃ a b. K = cbox a b
    using cbox_that by blast
  then have div: D' division_of ∪D'
    using pairwise_subset [OF pw ‹D' ⊆ D›] unfolding pairwise_def
    by (force simp: ‹finite D'› ‹{} ∉ D'› division_of_def)
  have le_meat: measure lebesgue (∪D') ≤ measure lebesgue T
  proof (rule measure_mono_fmeasurable)
    show (∪D') ∈ sets lebesgue
      using div lmeasurable_division by auto
    have ∪D' ⊆ ∪D
      using ‹D' ⊆ D› by blast
    also have ... ⊆ T
  proof (clarify)
    fix x D
    assume x ∈ D D ∈ D
    show x ∈ T
      using Ksub [OF ‹D ∈ D›]
      by (metis ‹x ∈ D› Int_iff dist_norm mem_ball norm_minus_commute

```

```

subsetD RT)
  qed
  finally show  $\bigcup \mathcal{D}' \subseteq T$  .
  show  $T \in \text{lmeasurable}$ 
    using  $\langle S \in \text{lmeasurable} \rangle \langle S \subseteq T \rangle \langle T - S \in \text{lmeasurable} \rangle \text{fmeasurable\_Diff\_D}$  by blast
  qed
  have  $\text{sum (measure lebesgue) } \mathcal{D}' = \text{sum content } \mathcal{D}'$ 
    using  $\langle \mathcal{D}' \subseteq \mathcal{D} \rangle \text{cbox by (force intro: sum.cong)}$ 
  then have  $(2 * B * \text{DIM('M)}) \wedge \text{DIM('N)} * \text{sum (measure lebesgue) } \mathcal{D}' =$ 
     $(2 * B * \text{real DIM('M)}) \wedge \text{DIM('N)} * \text{measure lebesgue } (\bigcup \mathcal{D}')$ 
    using  $\text{content\_division [OF div] by auto}$ 
  also have  $\dots \leq (2 * B * \text{real DIM('M)}) \wedge \text{DIM('N)} * \text{measure lebesgue } T$ 
    using  $\langle 0 < B \rangle$ 
    by  $(\text{intro mult\_left\_mono [OF le\_meaT]}) (\text{force simp add: algebra\_simps})$ 
  also have  $\dots \leq e/2$ 
    using  $T \langle 0 < B \rangle$  by  $(\text{simp add: field\_simps})$ 
  finally show ?thesis .
  qed
  finally show ?thesis .
qed
then have  $e2: \text{sum (measure lebesgue) } \mathcal{G} \leq e/2$  if  $\mathcal{G} \subseteq \text{fbx 'D}$  finite  $\mathcal{G}$  for  $\mathcal{G}$ 
  by  $(\text{metis finite\_subset\_image that})$ 
show  $\exists W \in \text{lmeasurable. } f ' S \subseteq W \wedge \text{measure lebesgue } W < e$ 
proof  $(\text{intro bexI conjI})$ 
  have  $\exists X \in \mathcal{D}. f y \in \text{fbx } X$  if  $y \in S$  for  $y$ 
  proof -
    obtain  $X$  where  $y \in X$   $X \in \mathcal{D}$ 
      using  $\langle S \subseteq \bigcup \mathcal{D} \rangle \langle y \in S \rangle$  by auto
    then have  $y: y \in \text{ball}(zf X) (R(zf X))$ 
      using  $\text{uvwz by fastforce}$ 
    have  $\text{conj\_le\_eq: } z - b \leq y \wedge y \leq z + b \longleftrightarrow \text{abs}(y - z) \leq b$  for  $z y b::\text{real}$ 
      by auto
    have  $yin: y \in \text{cbox (uf X) (vf X)}$  and  $zin: (zf X) \in \text{cbox (uf X) (vf X)}$ 
      using  $\text{uvwz } \langle X \in \mathcal{D} \rangle \langle y \in X \rangle$  by auto
    have  $\text{norm } (y - zf X) \leq (\sum i \in \text{Basis. } |(y - zf X) \cdot i|)$ 
      by  $(\text{rule norm\_le\_l1})$ 
    also have  $\dots \leq \text{real DIM('M)} * \text{prj1 (vf X - uf X)}$ 
    proof  $(\text{rule sum\_bounded\_above})$ 
      fix  $j::'M$  assume  $j: j \in \text{Basis}$ 
      show  $|(y - zf X) \cdot j| \leq \text{prj1 (vf X - uf X)}$ 
        using  $yin zin j$ 
        by  $(\text{fastforce simp add: mem\_box prj1\_idem [OF } \langle X \in \mathcal{D} \rangle j \text{ inner\_diff\_left})$ 
    qed
    finally have  $\text{nole: norm } (y - zf X) \leq \text{DIM('M)} * \text{prj1 (vf X - uf X)}$ 
      by  $\text{simp}$ 
    have  $\text{fle: } |f y \cdot i - f(zf X) \cdot i| \leq B * \text{DIM('M)} * \text{prj1 (vf X - uf X)}$  if  $i \in \text{Basis}$  for  $i$ 
    proof -

```

```

    have  $|f y \cdot i - f (zf X) \cdot i| = |(f y - f (zf X)) \cdot i|$ 
      by (simp add: algebra_simps)
    also have  $\dots \leq \text{norm } (f y - f (zf X))$ 
      by (simp add: Basis_le_norm that)
    also have  $\dots \leq B * \text{norm}(y - zf X)$ 
      by (metis uvz RB  $\langle X \in \mathcal{D} \rangle$  dist_commute dist_norm mem_ball  $\langle y \in S \rangle$ 
y)
    also have  $\dots \leq B * \text{real DIM}('M) * \text{prj1 } (vf X - uf X)$ 
      using  $\langle 0 < B \rangle$  by (simp add: none)
    finally show ?thesis .
qed
show ?thesis
  by (rule_tac x=X in bexI)
    (auto simp: fbx_def prj1_idem mem_box conj_le_eq inner_add inner_diff
fle  $\langle X \in \mathcal{D} \rangle$ )
qed
then show  $f ' S \subseteq (\bigcup D \in \mathcal{D}. \text{fbx } D)$  by auto
next
have 1:  $\bigwedge D. D \in \mathcal{D} \implies \text{fbx } D \in \text{lmeasurable}$ 
  by (auto simp: fbx_def)
have 2:  $I' \subseteq \mathcal{D} \implies \text{finite } I' \implies \text{measure lebesgue } (\bigcup D \in I'. \text{fbx } D) \leq e/2$  for
I'
  by (rule order_trans[OF measure_Union_le e2]) (auto simp: fbx_def)
show  $(\bigcup D \in \mathcal{D}. \text{fbx } D) \in \text{lmeasurable}$ 
  by (intro fmeasurable_UN_bound[OF  $\langle \text{countable } \mathcal{D} \rangle$  1 2])
have  $\text{measure lebesgue } (\bigcup D \in \mathcal{D}. \text{fbx } D) \leq e/2$ 
  by (intro measure_UN_bound[OF  $\langle \text{countable } \mathcal{D} \rangle$  1 2])
then show  $\text{measure lebesgue } (\bigcup D \in \mathcal{D}. \text{fbx } D) < e$ 
  using  $\langle 0 < e \rangle$  by linarith
qed
qed

proposition negligible_locally_Lipschitz_image:
fixes  $f :: 'M::\text{euclidean\_space} \Rightarrow 'N::\text{euclidean\_space}$ 
assumes  $MleN: \text{DIM}('M) \leq \text{DIM}('N)$  negligible  $S$ 
and lips:  $\bigwedge x. x \in S$ 
 $\implies \exists T B. \text{open } T \wedge x \in T \wedge$ 
 $(\forall y \in S \cap T. \text{norm}(f y - f x) \leq B * \text{norm}(y - x))$ 
shows negligible  $(f ' S)$ 
proof -
let  $?S = \lambda n. (\{x \in S. \text{norm } x \leq n \wedge$ 
 $(\exists T. \text{open } T \wedge x \in T \wedge$ 
 $(\forall y \in S \cap T. \text{norm } (f y - f x) \leq (\text{real } n + 1) * \text{norm } (y$ 
 $- x))\})$ 
have  $\text{negfn: } f ' ?S n \in \text{null\_sets lebesgue}$  for  $n::\text{nat}$ 
unfolding negligible_iff_null_sets[symmetric]
apply (rule_tac  $B = \text{real } n + 1$  in locally_Lipschitz_negl_bounded)
by (auto simp:  $MleN$  bounded_iff intro: negligible_subset [OF  $\langle \text{negligible } S \rangle$ ])
have  $S = (\bigcup n. ?S n)$ 

```

```

proof (intro set_eqI iffI)
  fix x assume x ∈ S
  with lips obtain T B where T: open T x ∈ T
    and B:  $\bigwedge y. y \in S \cap T \implies \text{norm}(f y - f x) \leq B * \text{norm}(y - x)$ 
  by metis+
  have no:  $\text{norm}(f y - f x) \leq (\text{nat} \lceil \max B (\text{norm } x) \rceil + 1) * \text{norm}(y - x)$  if
  y ∈ S ∩ T for y
  proof -
    have B *  $\text{norm}(y - x) \leq (\text{nat} \lceil \max B (\text{norm } x) \rceil + 1) * \text{norm}(y - x)$ 
    by (meson max.cobounded1 mult_right_mono nat_ceiling_le_eq nat_le_iff_add
  norm_ge_zero order_trans)
    then show ?thesis
    using B order_trans that by blast
  qed
  have norm x ≤ real (nat  $\lceil \max B (\text{norm } x) \rceil$ )
  by linarith
  then have x ∈ ?S (nat (ceiling (max B (norm x))))
  using T no by (force simp: ⟨x ∈ S⟩ algebra_simps)
  then show x ∈ (⋃ n. ?S n) by force
qed auto
then show ?thesis
by (rule ssubst) (auto simp: image_Union negligible_iff_null_sets intro: negfn)
qed

```

corollary negligible_differentiable_image_negligible:

```

fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
assumes MleN: DIM('M) ≤ DIM('N) negligible S
and diff_f: f differentiable_on S
shows negligible (f ' S)
proof -
  have  $\exists T B. \text{open } T \wedge x \in T \wedge (\forall y \in S \cap T. \text{norm}(f y - f x) \leq B * \text{norm}(y - x))$ 
  if x ∈ S for x
  proof -
    obtain f' where linear f'
    and f':  $\bigwedge e. e > 0 \implies \exists d > 0. \forall y \in S. \text{norm}(y - x) < d \implies \text{norm}(f y - f x - f'(y - x)) \leq e * \text{norm}(y - x)$ 
    using diff_f ⟨x ∈ S⟩
    by (auto simp: linear_linear_differentiable_on_def differentiable_def has_derivative_within_alt)
    obtain B where B > 0 and B:  $\forall x. \text{norm}(f' x) \leq B * \text{norm } x$ 
    using linear_bounded_pos ⟨linear f'⟩ by blast
    obtain d where d > 0
    and d:  $\bigwedge y. \llbracket y \in S; \text{norm}(y - x) < d \rrbracket \implies \text{norm}(f y - f x - f'(y - x)) \leq \text{norm}(y - x)$ 
    using f' [of 1] by (force simp:)
  show ?thesis
proof (intro exI conjI ballI)

```



```

show norm (f y - f x) ≤ (B + 1) * norm (y - x)
if y ∈ S ∩ ball x d for y
proof -
  have norm (f y - f x) - B * norm (y - x) ≤ norm (f y - f x) - norm (f'
(y - x))
    by (simp add: B)
  also have ... ≤ norm (f y - f x - f' (y - x))
    by (rule norm_triangle_ineq2)
  also have ... ≤ norm (y - x)
    by (metis IntE d dist_norm mem_ball norm_minus_commute that)
  finally show ?thesis
    by (simp add: algebra_simps)
qed
qed (use ‹d>0› in auto)
qed
with negligible_locally_Lipschitz_image assms show ?thesis by metis
qed

corollary negligible_differentiable_image_lowdim:
fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
assumes MlessN: DIM('M) < DIM('N) and diff_f: f differentiable_on S
shows negligible (f ' S)
proof -
have x ≤ DIM('M) ⇒ x ≤ DIM('N) for x
using MlessN by linarith
obtain lift :: 'M * real ⇒ 'N and drop :: 'N ⇒ 'M * real and j :: 'N
where linear lift linear drop and dropl [simp]: ∧z. drop (lift z) = z
and j ∈ Basis and j: ∧x. lift(x,0) · j = 0
using lowerdim_embeddings [OF MlessN] by metis
have negligible ((λx. lift (x, 0)) ' S)
proof -
have negligible {x. x·j = 0}
by (metis ‹j ∈ Basis› negligible_standard_hyperplane)
moreover have (λm. lift (m, 0)) ' S ⊆ {n. n · j = 0}
using j by force
ultimately show ?thesis
using negligible_subset by auto
qed
moreover
have f ∘ fst ∘ drop differentiable_on (λx. lift (x, 0)) ' S
using diff_f
apply (clarsimp simp add: differentiable_on_def)
apply (intro differentiable_chain_within linear_imp_differentiable [OF ‹linear
drop›]
  linear_imp_differentiable [OF linear_fst])
apply (force simp: image_comp o_def)
done
moreover
have f = f ∘ fst ∘ drop ∘ (λx. lift (x, 0))

```

by (simp add: o_def)
 ultimately show ?thesis
 by (metis (no_types) image_comp negligible_differentiable_image_negligible
 order_refl)
 qed

9.8.12 Measurability of countable unions and intersections of various kinds.

lemma

assumes $S: \bigwedge n. S\ n \in \text{lmeasurable}$
 and $leB: \bigwedge n. \text{measure lebesgue } (S\ n) \leq B$
 and $nest: \bigwedge n. S\ n \subseteq S(\text{Suc } n)$
 shows $\text{measurable_nested_Union}: (\bigcup n. S\ n) \in \text{lmeasurable}$
 and $\text{measure_nested_Union}: (\lambda n. \text{measure lebesgue } (S\ n)) \longrightarrow \text{measure lebesgue } (\bigcup n. S\ n)$ (is ?Lim)
 proof –
 have $\text{indicat_real } (\bigcup (\text{range } S)) \text{ integrable_on } UNIV \wedge$
 $(\lambda n. \text{integral } UNIV (\text{indicat_real } (S\ n)))$
 $\longrightarrow \text{integral } UNIV (\text{indicat_real } (\bigcup (\text{range } S)))$
 proof (rule monotone_convergence_increasing)
 show $\bigwedge n. (\text{indicat_real } (S\ n)) \text{ integrable_on } UNIV$
 using $S \text{ measurable_integrable}$ by blast
 show $\bigwedge n\ x::'a. \text{indicat_real } (S\ n)\ x \leq (\text{indicat_real } (S\ (\text{Suc } n))\ x)$
 by (simp add: indicator_leI nest rev_subsetD)
 have $\bigwedge x. (\exists n. x \in S\ n) \longrightarrow (\forall_F n \text{ in sequentially. } x \in S\ n)$
 by (metis eventually_sequentiallyI lift_Suc_mono_le nest subsetCE)
 then
 show $\bigwedge x. (\lambda n. \text{indicat_real } (S\ n)\ x) \longrightarrow (\text{indicat_real } (\bigcup (S\ ' UNIV))\ x)$
 by (simp add: indicator_def tendsto_eventually)
 show $\text{bounded } (\text{range } (\lambda n. \text{integral } UNIV (\text{indicat_real } (S\ n))))$
 using leB by (auto simp: lmeasure_integral_UNIV [symmetric] S bounded_iff)
 qed
 then have $(\bigcup n. S\ n) \in \text{lmeasurable} \wedge ?Lim$
 by (simp add: lmeasure_integral_UNIV S cong: conj_cong) (simp add: measurable_integrable)
 then show $(\bigcup n. S\ n) \in \text{lmeasurable} ?Lim$
 by auto
 qed

lemma

assumes $S: \bigwedge n. S\ n \in \text{lmeasurable}$
 and $djointish: \text{pairwise } (\lambda m\ n. \text{negligible } (S\ m \cap S\ n))\ UNIV$
 and $leB: \bigwedge n. (\sum k \leq n. \text{measure lebesgue } (S\ k)) \leq B$
 shows $\text{measurable_countable_negligible_Union}: (\bigcup n. S\ n) \in \text{lmeasurable}$
 and $\text{measure_countable_negligible_Union}: (\lambda n. (\text{measure lebesgue } (S\ n)))$
 $\text{sums measure lebesgue } (\bigcup n. S\ n)$ (is ?Sums)
 proof –
 have $1: \bigcup (S\ ' \{..n\}) \in \text{lmeasurable}$ for n

```

    using  $S$  by blast
  have 2:  $\text{measure lebesgue } (\bigcup (S \text{ ' } \{..n\})) \leq B$  for  $n$ 
  proof -
    have  $\text{measure lebesgue } (\bigcup (S \text{ ' } \{..n\})) \leq (\sum_{k \leq n} \text{measure lebesgue } (S k))$ 
      by (simp add:  $S$  fmeasurableD measure_UNION_le)
    with leB show ?thesis
      using order_trans by blast
  qed
  have 3:  $\bigwedge n. \bigcup (S \text{ ' } \{..n\}) \subseteq \bigcup (S \text{ ' } \{..Suc\ n\})$ 
    by (simp add: SUP_subset_mono)
  have eqS:  $(\bigcup n. S n) = (\bigcup n. \bigcup (S \text{ ' } \{..n\}))$ 
    using atLeastAtMost_iff by blast
  also have  $(\bigcup n. \bigcup (S \text{ ' } \{..n\})) \in \text{lmeasurable}$ 
    by (intro measurable_nested_Union [OF 1 2] 3)
  finally show  $(\bigcup n. S n) \in \text{lmeasurable}$  .
  have eqm:  $(\sum_{i \leq n} \text{measure lebesgue } (S i)) = \text{measure lebesgue } (\bigcup (S \text{ ' } \{..n\}))$ 
  for  $n$ 
    using assms by (simp add: measure_negligible_finite_Union_image pairwise_mono)
  have  $(\lambda n. (\text{measure lebesgue } (S n))) \text{ sums measure lebesgue } (\bigcup n. \bigcup (S \text{ ' } \{..n\}))$ 
    by (simp add: sums_def' eqm atLeast0AtMost) (intro measure_nested_Union [OF 1 2] 3)
  then show ?Sums
    by (simp add: eqS)
  qed

lemma negligible_countable_Union [intro]:
  assumes countable  $\mathcal{F}$  and meas:  $\bigwedge S. S \in \mathcal{F} \implies \text{negligible } S$ 
  shows negligible  $(\bigcup \mathcal{F})$ 
  proof (cases  $\mathcal{F} = \{\}$ )
  case False
  then show ?thesis
    by (metis from_nat_into range_from_nat_into assms negligible_Union_nat)
  qed simp

lemma
  assumes  $S$ :  $\bigwedge n. (S n) \in \text{lmeasurable}$ 
    and djointish: pairwise  $(\lambda m n. \text{negligible } (S m \cap S n))$  UNIV
    and bo: bounded  $(\bigcup n. S n)$ 
  shows measurable_countable_negligible_Union_bounded:  $(\bigcup n. S n) \in \text{lmeasurable}$ 
    and measure_countable_negligible_Union_bounded:  $(\lambda n. (\text{measure lebesgue } (S n))) \text{ sums measure lebesgue } (\bigcup n. S n)$  (is ?Sums)
  proof -
    obtain  $a b$  where  $ab$ :  $(\bigcup n. S n) \subseteq \text{cbox } a b$ 
      using bo bounded_subset_cbox_symmetric by metis
    then have  $B$ :  $(\sum_{k \leq n} \text{measure lebesgue } (S k)) \leq \text{measure lebesgue } (\text{cbox } a b)$ 
  for  $n$ 
  proof -

```

have $(\sum_{k \leq n}. \text{measure lebesgue } (S k)) = \text{measure lebesgue } (\bigcup (S \text{ '}\{..n\}\text{}))$
using *measure_negligible_finite_Union_image [OF _ _ pairwise_subset]*
djointish
by *(metis S finite_atMost_subset_UNIV)*
also have $\dots \leq \text{measure lebesgue } (cbox a b)$
proof *(rule measure_mono_fmeasurable)*
show $\bigcup (S \text{ '}\{..n\}) \in \text{sets lebesgue}$ **using** *S by blast*
qed *(use ab in auto)*
finally show *?thesis .*
qed
show $(\bigcup n. S n) \in \text{lmeasurable}$
by *(rule measurable_countable_negligible_Union [OF S djointish B])*
show *?Sums*
by *(rule measure_countable_negligible_Union [OF S djointish B])*
qed

lemma *measure_countable_Union_approachable:*

assumes *countable D e > 0 and measD: $\bigwedge d. d \in \mathcal{D} \implies d \in \text{lmeasurable}$*
and *B: $\bigwedge D'. \llbracket D' \subseteq \mathcal{D}; \text{finite } D' \rrbracket \implies \text{measure lebesgue } (\bigcup D') \leq B$*
obtains *D' where $D' \subseteq \mathcal{D}$ finite D' measure lebesgue $(\bigcup \mathcal{D}) - e < \text{measure lebesgue } (\bigcup D')$*
proof *(cases D = {})*
case *True*
then show *?thesis*
by *(simp add: <e > 0> that)*
next
case *False*
let *?S = $\lambda n. \bigcup_{k \leq n}. \text{from_nat_into } \mathcal{D} k$*
have $(\lambda n. \text{measure lebesgue } (?S n)) \longrightarrow \text{measure lebesgue } (\bigcup n. ?S n)$
proof *(rule measure_nested_Union)*
show *?S n \in lmeasurable for n*
by *(simp add: False_fmeasurable.finite_UN from_nat_into measD)*
show *measure lebesgue (?S n) $\leq B$ for n*
by *(metis (mono_tags, lifting) B False finite_atMost_finite_imageI from_nat_into image_iff_subsetI)*
show *?S n \subseteq ?S (Suc n) for n*
by *force*
qed
then obtain *N where $N: \bigwedge n. n \geq N \implies \text{dist } (\text{measure lebesgue } (?S n))$*
(measure lebesgue $(\bigcup n. ?S n)) < e$
using *metric_LIMSEQ_D <e > 0> by blast*
show *?thesis*
proof
show *from_nat_into D '}\{..N\} \subseteq D*
by *(auto simp: False from_nat_into)*
have *eq: $(\bigcup n. \bigcup_{k \leq n}. \text{from_nat_into } \mathcal{D} k) = (\bigcup \mathcal{D})$*
using *<countable D> False*
by *(auto intro: from_nat_into_dest: from_nat_into_surj [OF <countable D>])*
show *measure lebesgue $(\bigcup \mathcal{D}) - e < \text{measure lebesgue } (\bigcup (\text{from_nat_into } \mathcal{D}))$*

```

‘{..N}))
  using N [OF order_refl]
  by (auto simp: eq algebra_simps dist_norm)
qed auto
qed

```

9.8.13 Negligibility is a local property

lemma *locally_negligible_alt*:

$\text{negligible } S \iff (\forall x \in S. \exists U. \text{openin } (\text{top_of_set } S) U \wedge x \in U \wedge \text{negligible } U)$
 (is _ = ?rhs)

proof

assume *negligible S*

then show *?rhs*

using *openin_subtopology_self* **by** *blast*

next

assume *?rhs*

then obtain *U* **where** *ope*: $\bigwedge x. x \in S \implies \text{openin } (\text{top_of_set } S) (U x)$

and *cov*: $\bigwedge x. x \in S \implies x \in U x$

and *neg*: $\bigwedge x. x \in S \implies \text{negligible } (U x)$

by *metis*

obtain *F* **where** $\mathcal{F} \subseteq U \text{ ' } S$ *countable F* **and** *eq*: $\bigcup \mathcal{F} = \bigcup (U \text{ ' } S)$

using *ope* **by** (*force intro: Lindelof_openin [of U ' S S]*)

then have *negligible* $(\bigcup \mathcal{F})$

by (*metis imageE neg negligible_countable_Union subset_eq*)

with *eq* **have** *negligible* $(\bigcup (U \text{ ' } S))$

by *metis*

moreover have $S \subseteq \bigcup (U \text{ ' } S)$

using *cov* **by** *blast*

ultimately show *negligible S*

using *negligible_subset* **by** *blast*

qed

lemma *locally_negligible*: *locally negligible S* \iff *negligible S*

unfolding *locally_def*

by (*metis locally_negligible_alt negligible_subset openin_imp_subset openin_subtopology_self*)

9.8.14 Integral bounds

lemma *set_integral_norm_bound*:

fixes *f* :: $_ \Rightarrow \text{'a}$:: {*banach, second_countable_topology*}

shows *set_integrable M k f* $\implies \text{norm } (\text{LINT } x:k|M. f x) \leq (\text{LINT } x:k|M. \text{norm } (f x))$

using *integral_norm_bound*[*of M λx. indicator k x *_R f x*] **by** (*simp add: set_lebesgue_integral_def*)

lemma *set_integral_finite_UN_AE*:

fixes *f* :: $_ \Rightarrow _$:: {*banach, second_countable_topology*}

assumes *finite I*

```

    and ae:  $\bigwedge i j. i \in I \implies j \in I \implies AE\ x\ in\ M. (x \in A\ i \wedge x \in A\ j) \longrightarrow i = j$ 
    and [measurable]:  $\bigwedge i. i \in I \implies A\ i \in sets\ M$ 
    and f:  $\bigwedge i. i \in I \implies set\_integrable\ M\ (A\ i)\ f$ 
  shows (LINT  $x:(\bigcup i \in I. A\ i) | M. f\ x) = (\sum i \in I. LINT\ x:A\ i | M. f\ x)$ 
  using <finite I> order_refl[of I]
proof (induction I rule: finite_subset_induct')
  case (insert i I')
  have AE x in M. ( $\forall j \in I'. x \in A\ i \longrightarrow x \notin A\ j$ )
  proof (intro AE_ball_countable[THEN iffD2] ballI)
    fix j assume  $j \in I'$ 
    with <I'  $\subseteq$  I> <i  $\notin$  I'> have  $i \neq j$ 
    by auto
  then show AE x in M.  $x \in A\ i \longrightarrow x \notin A\ j$ 
    using ae[of i j] <i  $\in$  I> by auto
  qed (use <finite I'> in <rule countable_finite>)
  then have AE  $x \in A\ i$  in M.  $\forall xa \in I'. x \notin A\ xa$ 
    by auto
  with insert.hyps insert.IH[symmetric]
  show ?case
    by (auto intro!: set_integral_Un_AE sets.finite_UN f set_integrable_UN)
  qed (simp add: set_lebesgue_integral_def)

```

lemma set_integrable_norm:

```

  fixes f :: 'a  $\Rightarrow$  'b::{banach, second_countable_topology}
  assumes f: set_integrable M k f shows set_integrable M k ( $\lambda x. norm\ (f\ x)$ )
  using integrable_norm f by (force simp add: set_integrable_def)

```

lemma absolutely_integrable_bounded_variation:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: f absolutely_integrable_on UNIV
  obtains B where  $\forall d. d\ division\_of\ (\bigcup d) \longrightarrow sum\ (\lambda k. norm\ (integral\ k\ f))\ d \leq B$ 
proof (rule that[of integral UNIV ( $\lambda x. norm\ (f\ x)$ )]; safe)
  fix d :: 'a set set assume d: d division_of  $\bigcup d$ 
  have *:  $k \in d \implies f\ absolutely\_integrable\_on\ k$  for k
    using f[THEN set_integrable_subset, of k] division_ofD(2,4)[OF d, of k] by
  auto
  note d' = division_ofD[OF d]
  have ( $\sum k \in d. norm\ (integral\ k\ f)$ ) = ( $\sum k \in d. norm\ (LINT\ x:k | lebesgue. f\ x)$ )
    by (intro sum.cong refl arg_cong[where f=norm] set_lebesgue_integral_eq_integral(2)[symmetric]
  *)
  also have  $\dots \leq (\sum k \in d. LINT\ x:k | lebesgue. norm\ (f\ x))$ 
    by (intro sum_mono set_integral_norm_bound *)
  also have  $\dots = (\sum k \in d. integral\ k\ (\lambda x. norm\ (f\ x)))$ 
    by (intro sum.cong refl set_lebesgue_integral_eq_integral(2) set_integrable_norm
  *)
  also have  $\dots \leq integral\ (\bigcup d)\ (\lambda x. norm\ (f\ x))$ 
  using integrable_on_subdivision[OF d] assms f unfolding absolutely_integrable_on_def
  by (subst integral_combine_division_topdown[OF _ d]) auto

```

also have $\dots \leq \text{integral UNIV } (\lambda x. \text{norm } (f x))$
using *integrable_on_subdivision*[OF d] *assms* **unfolding** *absolutely_integrable_on_def*
by (*intro integral_subset_le*) *auto*
finally show $(\sum k \in d. \text{norm } (\text{integral } k f)) \leq \text{integral UNIV } (\lambda x. \text{norm } (f x))$.
qed

lemma *absdiff_norm_less*:

assumes $\text{sum } (\lambda x. \text{norm } (f x - g x)) S < e$
shows $|\text{sum } (\lambda x. \text{norm}(f x)) S - \text{sum } (\lambda x. \text{norm}(g x)) S| < e$ (**is** *?lhs < e*)
proof –
have $?lhs \leq (\sum i \in S. |\text{norm } (f i) - \text{norm } (g i)|)$
by (*metis (no_types) sum_abs sum_subtractf*)
also have $\dots \leq (\sum x \in S. \text{norm } (f x - g x))$
by (*simp add: norm_triangle_ineq3 sum_mono*)
also have $\dots < e$
using *assms*(1) **by** *blast*
finally show *?thesis* .
qed

proposition *bounded_variation_absolutely_integrable_interval*:

fixes $f :: 'n::\text{euclidean_space} \Rightarrow 'm::\text{euclidean_space}$
assumes $f: f \text{ integrable_on } \text{cbox } a \ b$
and $*$: $\bigwedge d. d \text{ division_of } (\text{cbox } a \ b) \implies \text{sum } (\lambda K. \text{norm}(\text{integral } K f)) d \leq B$
shows $f \text{ absolutely_integrable_on } \text{cbox } a \ b$
proof –
let $?f = \lambda d. \sum K \in d. \text{norm } (\text{integral } K f)$ **and** $?D = \{d. d \text{ division_of } (\text{cbox } a \ b)\}$
have $D_1: ?D \neq \{\}$
by (*rule elementary_interval*[of a b]) *auto*
have $D_2: \text{bdd_above } (?f' ?D)$
by (*metis * mem_Collect_eq bdd_aboveI2*)
note $D = D_1 \ D_2$
let $?S = \text{SUP } x \in ?D. ?f x$
have $*$: $\exists \gamma. \text{gauge } \gamma \wedge$
 $(\forall p. p \text{ tagged_division_of } \text{cbox } a \ b \wedge$
 $\gamma \text{ fine } p \implies$
 $\text{norm } ((\sum (x,k) \in p. \text{content } k *_{\mathbb{R}} \text{norm } (f x)) - ?S) < e)$
if $e: e > 0$ **for** e
proof –
have $?S - e/2 < ?S$ **using** $\langle e > 0 \rangle$ **by** *simp*
then obtain d **where** $d: d \text{ division_of } (\text{cbox } a \ b)$ $?S - e/2 < (\sum k \in d. \text{norm } (\text{integral } k f))$
unfolding *less_cSUP_iff*[OF D] **by** *auto*
note $d' = \text{division_of } D$ [OF *this*(1)]

have $\exists e > 0. \forall i \in d. x \notin i \implies \text{ball } x \ e \cap i = \{\}$ **for** x
proof –
have $\exists d' > 0. \forall x' \in \bigcup \{i \in d. x \notin i\}. d' \leq \text{dist } x \ x'$
proof (*rule separate_point_closed*)

```

    show closed ( $\bigcup \{i \in d. x \notin i\}$ )
      using  $d'$  by force
    show  $x \notin \bigcup \{i \in d. x \notin i\}$ 
      by auto
  qed
  then show ?thesis
    by force
  qed
  then obtain  $k$  where  $k: \bigwedge x. 0 < k x \wedge i x. \llbracket i \in d; x \notin i \rrbracket \implies \text{ball } x (k x) \cap$ 
 $i = \{\}$ 
    by metis
  have  $e/2 > 0$ 
    using  $e$  by auto
  with Henstock_lemma[OF  $f$ ]
  obtain  $\gamma$  where  $g: \text{gauge } \gamma$ 
     $\bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } \text{cbox } a b; \gamma \text{ fine } p \rrbracket$ 
     $\implies (\sum (x,k) \in p. \text{norm } (\text{content } k *_{\mathbb{R}} f x - \text{integral } k f)) < e/2$ 
  by (metis (no_types, lifting))
  let  $?g = \lambda x. \gamma x \cap \text{ball } x (k x)$ 
  show ?thesis
  proof (intro exI conjI allI impI)
    show gauge ?g
      using  $g(1) k(1)$  by (auto simp: gauge_def)
  next
    fix  $p$ 
    assume  $p \text{ tagged\_division\_of } (\text{cbox } a b) \wedge ?g \text{ fine } p$ 
    then have  $p: p \text{ tagged\_division\_of } \text{cbox } a b \gamma \text{ fine } p (\lambda x. \text{ball } x (k x)) \text{ fine } p$ 
      by (auto simp: fine_Int)
    note  $p' = \text{tagged\_division\_of } D[\text{OF } p(1)]$ 
    define  $p'$  where  $p' = \{(x,k) \mid x k. \exists i l. x \in i \wedge i \in d \wedge (x,l) \in p \wedge k = i \cap$ 
 $l\}$ 
    have  $gp': \gamma \text{ fine } p'$ 
      using  $p(2)$  by (auto simp:  $p'_\text{def}$  fine_def)
    have  $p'': p' \text{ tagged\_division\_of } (\text{cbox } a b)$ 
    proof (rule tagged_division_ofI)
      show finite  $p'$ 
        proof (rule finite_subset)
          show  $p' \subseteq (\lambda(k, x, l). (x, k \cap l)) \text{ ' } (d \times p)$ 
            by (force simp:  $p'_\text{def}$  image_iff)
          show finite  $((\lambda(k, x, l). (x, k \cap l)) \text{ ' } (d \times p))$ 
            by (simp add:  $d'(1) p'(1)$ )
        qed
      qed
    next
      fix  $x K$ 
      assume  $(x, K) \in p'$ 
      then have  $\exists i l. x \in i \wedge i \in d \wedge (x, l) \in p \wedge K = i \cap l$ 
        unfolding  $p'_\text{def}$  by auto
      then obtain  $i l$  where  $il: x \in i \wedge i \in d \wedge (x, l) \in p \wedge K = i \cap l$  by blast
      show  $x \in K$  and  $K \subseteq \text{cbox } a b$ 

```



```

    using p'(2-3)[OF il(3)] il by auto
  show  $\exists a b. K = \text{cbox } a b$ 
  unfolding il using p'(4)[OF il(3)] d'(4)[OF il(2)] by (meson Int_interval)
next
  fix x1 K1
  assume (x1, K1)  $\in p'$ 
  then have  $\exists i l. x1 \in i \wedge i \in d \wedge (x1, l) \in p \wedge K1 = i \cap l$ 
    unfolding p'_def by auto
  then obtain i1 l1 where il1:  $x1 \in i1 \wedge i1 \in d \wedge (x1, l1) \in p \wedge K1 = i1 \cap l1$ 
by blast
  fix x2 K2
  assume (x2, K2)  $\in p'$ 
  then have  $\exists i l. x2 \in i \wedge i \in d \wedge (x2, l) \in p \wedge K2 = i \cap l$ 
    unfolding p'_def by auto
  then obtain i2 l2 where il2:  $x2 \in i2 \wedge i2 \in d \wedge (x2, l2) \in p \wedge K2 = i2 \cap l2$ 
by blast
  assume (x1, K1)  $\neq (x2, K2)$ 
  then have interior i1  $\cap$  interior i2 = {}  $\vee$  interior l1  $\cap$  interior l2 = {}
    using d'(5)[OF il1(2) il2(2)] p'(5)[OF il1(3) il2(3)] by (auto simp: il1
il2)
  then show interior K1  $\cap$  interior K2 = {}
    unfolding il1 il2 by auto
next
  have *:  $\forall (x, X) \in p'. X \subseteq \text{cbox } a b$ 
    unfolding p'_def using d' by blast
  show  $\bigcup \{K. \exists x. (x, K) \in p'\} = \text{cbox } a b$ 
  proof
    show  $\bigcup \{k. \exists x. (x, k) \in p'\} \subseteq \text{cbox } a b$ 
      using * by auto
  next
    show  $\text{cbox } a b \subseteq \bigcup \{k. \exists x. (x, k) \in p'\}$ 
  proof
    fix y
    assume y:  $y \in \text{cbox } a b$ 
    obtain x L where xl:  $(x, L) \in p \wedge y \in L$ 
      using y unfolding p'(6)[symmetric] by auto
    obtain I where i:  $I \in d \wedge y \in I$ 
      using y unfolding d'(6)[symmetric] by auto
    have  $x \in I$ 
      using fineD[OF p(3) xl(1)] using k(2) i xl by auto
    then show  $y \in \bigcup \{K. \exists x. (x, K) \in p'\}$ 
  proof -
    obtain x l where xl:  $(x, l) \in p \wedge y \in l$ 
      using y unfolding p'(6)[symmetric] by auto
    obtain i where i:  $i \in d \wedge y \in i$ 
      using y unfolding d'(6)[symmetric] by auto
    have  $x \in i$ 
      using fineD[OF p(3) xl(1)] using k(2) i xl by auto
    then show ?thesis

```

```

      unfolding p'_def by (rule_tac X=i ∩ l in UnionI) (use i xl in auto)
    qed
  qed
  qed
  then have sum_less_e2: (∑ (x,K) ∈ p'. norm (content K *R f x - integral
K f)) < e/2
    using g(2) gp' tagged_division_of_def by blast

  have in_p': (x, I ∩ L) ∈ p' if x: (x, L) ∈ p I ∈ d and y: y ∈ I y ∈ L
    for x I L y
  proof -
    have x ∈ I
      using fineD[OF p(3) that(1)] k(2)[OF ⟨I ∈ d⟩] y by auto
    with x have (∃ i l. x ∈ i ∧ i ∈ d ∧ (x, l) ∈ p ∧ I ∩ L = i ∩ l)
      by blast
    then have (x, I ∩ L) ∈ p'
      by (simp add: p'_def)
    with y show ?thesis by auto
  qed
  moreover
  have Ex_p_p': ∃ y i l. (x, K) = (y, i ∩ l) ∧ (y, l) ∈ p ∧ i ∈ d ∧ i ∩ l ≠ {}
    if xK: (x,K) ∈ p' for x K
  proof -
    obtain i l where il: x ∈ i i ∈ d (x, l) ∈ p K = i ∩ l
      using xK unfolding p'_def by auto
    then show ?thesis
      using p'(2) by fastforce
  qed
  ultimately have p'alt: p' = {(x, I ∩ L) | x I L. (x,L) ∈ p ∧ I ∈ d ∧ I ∩ L
≠ {}}
    by auto
  have sum_p': (∑ (x,K) ∈ p'. norm (integral K f)) = (∑ k∈snd ' p'. norm
(integral k f))
  proof (rule sum.over_tagged_division_lemma[OF p''])
    show ∧ u v. box u v = {} ⇒ norm (integral (cbox u v) f) = 0
      by (auto intro: integral_null simp: content_eq_0_interior)
  qed
  have snd_p_div: snd ' p division_of cbox a b
    by (rule division_of_tagged_division[OF p(1)])
  note snd_p = division_ofD[OF snd_p_div]
  have fin_d_sndp: finite (d × snd ' p)
    by (simp add: d'(1) snd_p(1))

  have *: ∧ sni sni' sf sf'. [|sf' - sni'| < e/2; ?S - e/2 < sni; sni' ≤ ?S;
sni ≤ sni'; sf' = sf] ⇒ |sf - ?S| < e

  by arith
  show norm ((∑ (x,k) ∈ p. content k *R norm (f x)) - ?S) < e
    unfolding real_norm_def

```

```

proof (rule *)
  show  $|\left(\sum_{(x,K)\in p'}. \text{norm} (\text{content } K *_R f x)\right) - \left(\sum_{(x,k)\in p'}. \text{norm} (\text{integral } k f)\right)| < e/2$ 
    using  $p'' \text{ sum\_less\_e2}$  unfolding  $\text{split\_def}$  by (force intro!: absd-iff_norm_less)
    show  $\left(\sum_{(x,k)\in p'}. \text{norm} (\text{integral } k f)\right) \leq ?S$ 
      by (auto simp: sum_p' division_of_tagged_division[OF p''] D intro!: cSUP_upper)
    show  $\left(\sum_{k\in d}. \text{norm} (\text{integral } k f)\right) \leq \left(\sum_{(x,k)\in p'}. \text{norm} (\text{integral } k f)\right)$ 
    proof -
      have  $\ast: \{k \cap l \mid k l. k \in d \wedge l \in \text{snd } 'p\} = (\lambda(k,l). k \cap l) \text{ } '(d \times \text{snd } 'p)$ 
        by auto
      have  $\left(\sum_{K\in d}. \text{norm} (\text{integral } K f)\right) \leq \left(\sum_{i\in d}. \sum_{l\in \text{snd } 'p}. \text{norm} (\text{integral } (i \cap l) f)\right)$ 
    proof (rule sum_mono)
      fix  $K$  assume  $k: K \in d$ 
      from  $d'(4)$ [OF this] obtain  $u v$  where  $uv: K = \text{cbox } u v$  by metis
      define  $d'$  where  $d' = \{\text{cbox } u v \cap l \mid l. l \in \text{snd } 'p \wedge \text{cbox } u v \cap l \neq \{\}\}$ 
      have  $uvab: \text{cbox } u v \subseteq \text{cbox } a b$ 
        using  $d(1)$   $k uv$  by blast
      have  $d'_div: d' \text{ division\_of } \text{cbox } u v$ 
        unfolding  $d'_def$  by (rule division_inter_1 [OF snd_p_div uvab])
      moreover have  $\text{norm} \left(\sum_{i\in d'}. \text{integral } i f\right) \leq \left(\sum_{k\in d'}. \text{norm} (\text{integral } k f)\right)$ 
    by (simp add: sum_norm_le)
    moreover have  $f \text{ integrable\_on } K$ 
      using  $f \text{ integrable\_on\_subcbox } uv uvab$  by blast
    moreover have  $d' \text{ division\_of } K$ 
      using  $d'_div uv$  by blast
    ultimately have  $\text{norm} (\text{integral } K f) \leq \text{sum } (\lambda k. \text{norm} (\text{integral } k f)) d'$ 
      by (simp add: integral_combine_division_topdown)
    also have  $\dots = \left(\sum_{I\in\{K \cap L \mid L. L \in \text{snd } 'p\}}. \text{norm} (\text{integral } I f)\right)$ 
    proof (rule sum_mono_neutral_left)
      show  $\text{finite } \{K \cap L \mid L. L \in \text{snd } 'p\}$ 
        by (simp add: snd_p(1))
      show  $\forall i\in\{K \cap L \mid L. L \in \text{snd } 'p\}. \text{norm} (\text{integral } i f) = 0$ 
         $d' \subseteq \{K \cap L \mid L. L \in \text{snd } 'p\}$ 
        using  $d'_def \text{ image\_eqI } uv$  by auto
    qed
    also have  $\dots = \left(\sum_{l\in \text{snd } 'p}. \text{norm} (\text{integral } (K \cap l) f)\right)$ 
      unfolding  $\text{Setcompr\_eq\_image}$ 
    proof (rule sum_reindex_nontrivial [unfolded o_def])
      show  $\text{finite } (\text{snd } 'p)$ 
        using  $\text{snd\_p}(1)$  by blast
      show  $\text{norm} (\text{integral } (K \cap l) f) = 0$ 
        if  $l \in \text{snd } 'p \ y \in \text{snd } 'p \ l \neq y \ K \cap l = K \cap y$  for  $l y$ 
    proof -
      have  $\text{interior } (K \cap l) \subseteq \text{interior } (l \cap y)$ 
        by (metis Int_lower2 interior_mono le_inf_iff that(4))

```

```

then have interior (K ∩ l) = {}
  by (simp add: snd_p(5) that)
moreover from d'(4)[OF k] snd_p(4)[OF that(1)]
obtain u1 v1 u2 v2
  where uv: K = cbox u1 u2 l = cbox v1 v2 by metis
ultimately show ?thesis
  using that integral_null
  unfolding uv Int_interval content_eq_0_interior
  by (metis (mono_tags, lifting) norm_eq_zero)
qed
qed
finally show norm (integral K f) ≤ (∑ l ∈ snd ' p. norm (integral (K ∩
l) f)) .
qed
also have ... = (∑ (i,l) ∈ d × snd ' p. norm (integral (i ∩ l) f))
  by (simp add: sum.cartesian_product)
also have ... = (∑ x ∈ d × snd ' p. norm (integral (case_prod (∩) x) f))
  by (force simp: split_def intro!: sum.cong)
also have ... = (∑ k ∈ {i ∩ l | i l. i ∈ d ∧ l ∈ snd ' p}. norm (integral k
f))
proof -
have eq0: (integral (l1 ∩ k1) f) = 0
  if l1 ∩ k1 = l2 ∩ k2 (l1, k1) ≠ (l2, k2)
  l1 ∈ d (j1,k1) ∈ p l2 ∈ d (j2,k2) ∈ p
  for l1 l2 k1 k2 j1 j2
proof -
obtain u1 v1 u2 v2 where uv: l1 = cbox u1 u2 k1 = cbox v1 v2
  using ⟨j1, k1⟩ ∈ p ⟨l1 ∈ d⟩ d'(4) p'(4) by blast
have l1 ≠ l2 ∨ k1 ≠ k2
  using that by auto
then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
= {}
  by (meson d'(5) old.prod.inject p'(5) that(3) that(4) that(5) that(6))
moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
  by (simp add: that(1))
ultimately have interior(l1 ∩ k1) = {}
  by auto
then show ?thesis
  unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
qed
show ?thesis
  unfolding *
  apply (rule sum.reindex_nontrivial [OF fn_d_sndp, symmetric,
unfolded o_def])
  apply clarsimp
  by (metis eq0 fst_conv snd_conv)
qed
also have ... = (∑ (x,k) ∈ p'. norm (integral k f))
  unfolding sum_p'

```

```

proof (rule sum.mono_neutral_right)
  show finite {i ∩ l | i l. i ∈ d ∧ l ∈ snd ' p}
    by (metis * finite_imageI[OF fin_d_sndp])
  show snd ' p' ⊆ {i ∩ l | i l. i ∈ d ∧ l ∈ snd ' p}
    by (clarsimp simp: p'_def) (metis image_eqI snd_conv)
  show ∑ i ∈ {i ∩ l | i l. i ∈ d ∧ l ∈ snd ' p} - snd ' p'. norm (integral i f)
= 0
    by clarsimp (metis Henstock_Kurzweil_Integration.integral_empty
disjoint_iff_image_eqI in_p' snd_conv)
  qed
  finally show ?thesis .
qed
show (∑ (x,k) ∈ p'. norm (content k *R f x)) = (∑ (x,k) ∈ p. content k *R
norm (f x))
proof -
  let ?S = {(x, i ∩ l) | x i l. (x, l) ∈ p ∧ i ∈ d}
  have *: ?S = (λ(x,l,i). (fst xl, snd xl ∩ i)) ' (p × d)
    by force
  have fin_pd: finite (p × d)
    using finite_cartesian_product[OF p'(1) d'(1)] by metis
  have (∑ (x,k) ∈ p'. norm (content k *R f x)) = (∑ (x,k) ∈ ?S. |content
k| * norm (f x))
    unfolding norm_scaleR
  proof (rule sum.mono_neutral_left)
    show finite {(x, i ∩ l) | x i l. (x, l) ∈ p ∧ i ∈ d}
      by (simp add: * fin_pd)
    qed (use p'alt in ‹force+›)
  also have ... = (∑ ((x,l),i) ∈ p × d. |content (l ∩ i)| * norm (f x))
  proof -
    have |content (l1 ∩ k1)| * norm (f x1) = 0
      if (x1, l1) ∈ p (x2, l2) ∈ p k1 ∈ d k2 ∈ d
        x1 = x2 l1 ∩ k1 = l2 ∩ k2 x1 ≠ x2 ∨ l1 ≠ l2 ∨ k1 ≠ k2
      for x1 l1 k1 x2 l2 k2
    proof -
      obtain u1 v1 u2 v2 where uv: k1 = cbox u1 u2 l1 = cbox v1 v2
        by (meson ‹(x1, l1) ∈ p› ‹k1 ∈ d› d(1) division_ofD(4) p'(4))
      have l1 ≠ l2 ∨ k1 ≠ k2
        using that by auto
      then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
= {}
        using that p'(5) d'(5) by (metis snd_conv)
      moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
        unfolding that ..
      ultimately have interior (l1 ∩ k1) = {}
        by auto
      then show |content (l1 ∩ k1)| * norm (f x1) = 0
        unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
    qed
  then show ?thesis

```

```

    unfolding *
    apply (subst sum.reindex_nontrivial [OF fin_pd])
    unfolding split_paired_all o_def split_def prod.inject
    by force+
  qed
  also have ... = (∑ (x,k) ∈ p. content k *R norm (f x))
  proof -
    (f x) have sumeq: (∑ i∈d. content (l ∩ i) * norm (f x)) = content l * norm
      if (x, l) ∈ p for x l
    proof -
      note xl = p'(2-4)[OF that]
      then obtain u v where uv: l = cbox u v by blast
      have (∑ i∈d. |content (l ∩ i)|) = (∑ k∈d. content (k ∩ cbox u v))
        by (simp add: Int_commute uv)
      also have ... = sum content {k ∩ cbox u v | k. k ∈ d}
      proof -
        have eq0: content (k ∩ cbox u v) = 0
          if k ∈ d y ∈ d k ≠ y and eq: k ∩ cbox u v = y ∩ cbox u v for k y
        proof -
          from d'(4)[OF that(1)] d'(4)[OF that(2)]
          obtain α β where α: k ∩ cbox u v = cbox α β
            by (meson Int_interval)
          have {} = interior ((k ∩ y) ∩ cbox u v)
            by (simp add: d'(5) that)
          also have ... = interior (y ∩ (k ∩ cbox u v))
            by auto
          also have ... = interior (k ∩ cbox u v)
            unfolding eq by auto
          finally show ?thesis
            unfolding α content_eq_0_interior ..
        qed
      then show ?thesis
        unfolding Setcompr_eq_image
        by (fastforce intro: sum.reindex_nontrivial [OF ⟨finite d⟩, unfolded
o_def, symmetric])
    qed
    also have ... = sum content {cbox u v ∩ k | k. k ∈ d ∧ cbox u v ∩ k
≠ {}}
  proof (rule sum.mono_neutral_right)
    show finite {k ∩ cbox u v | k. k ∈ d}
      by (simp add: d'(1))
  qed (fastforce simp: inf.commute)+
  finally have (∑ i∈d. |content (l ∩ i)|) = content (cbox u v)
    using additive_content_division[OF division_inter_1[OF d(1)]] uv
xl(2) by auto
  then show (∑ i∈d. content (l ∩ i) * norm (f x)) = content l * norm
(f x)
    unfolding sum_distrib_right[symmetric] using uv by auto

```

```

      qed
      show ?thesis
      by (subst sum_Sigma_product[symmetric]) (auto intro!: sumeq sum.cong
p' d')
      qed
      finally show ?thesis .
    qed
    qed (rule d)
  qed
  then show ?thesis
  using absolutely_integrable_onI [OF f has_integral_integrable] has_integral[of
_ ?S]
  by blast
qed

```

lemma *bounded_variation_absolutely_integrable*:

```

  fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
  assumes f_integrable_on UNIV
  and  $\forall d. d \text{ division\_of } (\bigcup d) \longrightarrow \text{sum } (\lambda k. \text{norm } (\text{integral } k f)) d \leq B$ 
  shows f absolutely_integrable_on UNIV
proof (rule absolutely_integrable_onI, fact)
  let ?f =  $\lambda D. \sum_{k \in D}. \text{norm } (\text{integral } k f)$  and ?D =  $\{d. d \text{ division\_of } (\bigcup d)\}$ 
  define SDF where  $SDF \equiv \text{SUP } d \in ?D. ?f d$ 
  have D_1: ?D  $\neq \{\}$ 
  by (rule elementary_interval) auto
  have D_2: bdd_above (?f' ?D)
  using assms(2) by auto
  have f_int:  $\bigwedge a b. f \text{ absolutely\_integrable\_on } \text{cbox } a b$ 
  using assms integrable_on_subcbox
  by (blast intro!: bounded_variation_absolutely_integrable_interval)
  have  $\exists B > 0. \forall a b. \text{ball } 0 B \subseteq \text{cbox } a b \longrightarrow$ 
 $|\text{integral } (\text{cbox } a b) (\lambda x. \text{norm } (f x)) - SDF| < e$ 
  if  $0 < e$  for e
proof -
  have  $\exists y \in ?f' ?D. \neg y \leq SDF - e$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then have  $SDF \leq SDF - e$ 
  unfolding SDF_def
  by (metis (mono_tags) D_1 cSUP_least image_eqI)
  then show False
  using that by auto
qed
  then obtain d K where ddiv:  $d \text{ division\_of } \bigcup d$  and  $K = ?f d SDF - e < K$ 
  by (auto simp add: image_iff not_le)
  then have d:  $SDF - e < ?f d$ 
  by auto

```

```

note d'=division_ofD[OF ddiv]
have bounded (∪ d)
  using ddiv by blast
then obtain K where K: 0 < K ∀ x∈∪ d. norm x ≤ K
  using bounded_pos by blast
show ?thesis
proof (intro conjI impI allI exI)
  fix a b :: 'n
  assume ab: ball 0 (K + 1) ⊆ cbox a b
  have *: ∧ s s1. [|SDF - e < s1; s1 ≤ s; s < SDF + e|] ⇒ |s - SDF| < e
    by arith
  show |integral (cbox a b) (λx. norm (f x)) - SDF| < e
    unfolding real_norm_def
  proof (rule * [OF d])
    have ?f d ≤ sum (λk. integral k (λx. norm (f x))) d
    proof (intro sum_mono)
      fix k assume k ∈ d
      with d'(4) f_int show norm (integral k f) ≤ integral k (λx. norm (f x))
      by (force simp: absolutely_integrable_on_def integral_norm_bound_integral)
    qed
    also have ... = integral (∪ d) (λx. norm (f x))
      by (metis (full_types) absolutely_integrable_on_def d'(4) ddiv f_int
integral_combine_division_bottomup)
    also have ... ≤ integral (cbox a b) (λx. norm (f x))
    proof -
      have ∪ d ⊆ cbox a b
        using K(2) ab by fastforce
      then show ?thesis
        using integrable_on_subdivision[OF ddiv] f_int[of a b] unfolding
absolutely_integrable_on_def
        by (auto intro!: integral_subset_le)
    qed
    finally show ?f d ≤ integral (cbox a b) (λx. norm (f x)) .
next
  have e/2>0
    using ‹e > 0› by auto
  moreover
  have f: f integrable_on cbox a b (λx. norm (f x)) integrable_on cbox a b
    using f_int by (auto simp: absolutely_integrable_on_def)
  ultimately obtain d1 where gauge d1
    and d1: ∧ p. [|p tagged_division_of (cbox a b); d1 fine p|] ⇒
      norm ((∑ (x,k) ∈ p. content k *R norm (f x)) - integral (cbox a b) (λx.
norm (f x))) < e/2
    unfolding has_integral_integral has_integral by meson
  obtain d2 where gauge d2
    and d2: ∧ p. [|p tagged_partial_division_of (cbox a b); d2 fine p|] ⇒
      (∑ (x,k) ∈ p. norm (content k *R f x - integral k f)) < e/2
    by (blast intro: Henstock_lemma [OF f(1) ‹e/2>0])
  obtain p where

```



```

    p: p tagged_division_of (cbox a b) d1 fine p d2 fine p
  by (rule fine_division_exists [OF gauge_Int [OF ‹gauge d1› ‹gauge d2›],
of a b])
    (auto simp add: fine_Int)
  have *:  $\bigwedge sf\ sf'\ si\ di. \llbracket sf' = sf; si \leq SDF; |sf - si| < e/2; |sf' - di| < e/2 \rrbracket \implies di < SDF + e$ 
    by arith
  have integral (cbox a b) ( $\lambda x. norm (f x)$ ) < SDF + e
  proof (rule *)
    show  $|\left(\sum (x,k) \in p. norm (content\ k\ *_R\ f\ x)\right) - \left(\sum (x,k) \in p. norm (integral\ k\ f)\right)| < e/2$ 
      unfolding split_def
      proof (rule absdiff_norm_less)
        show  $\left(\sum p \in p. norm (content (snd\ p) *_R f (fst\ p) - integral (snd\ p) f)\right) < e/2$ 
          using d2[of p] p(1,3) by (auto simp: tagged_division_of_def split_def)
        qed
        show  $|\left(\sum (x,k) \in p. content\ k\ *_R\ norm (f\ x)\right) - integral (cbox\ a\ b) (\lambda x. norm(f\ x))| < e/2$ 
          using d1[OF p(1,2)] by (simp only: real_norm_def)
        show  $\left(\sum (x,k) \in p. content\ k\ *_R\ norm (f\ x)\right) = \left(\sum (x,k) \in p. norm (content\ k\ *_R\ f\ x)\right)$ 
          by (auto simp: split_paired_all sum.cong [OF refl])
        have  $\left(\sum (x,k) \in p. norm (integral\ k\ f)\right) = \left(\sum k \in snd\ ' p. norm (integral\ k\ f)\right)$ 
          apply (rule sum.over_tagged_division_lemma[OF p(1)])
          by (metis Henstock_Kurzweil_Integration.integral_empty integral_open_interval norm_zero)
        also have ...  $\leq SDF$ 
          using partial_division_of_tagged_division[of p cbox a b] p(1)
          by (auto simp: SDF_def tagged_partial_division_of_def intro!: cSUP_upper2 D_1 D_2)
        finally show  $\left(\sum (x,k) \in p. norm (integral\ k\ f)\right) \leq SDF$  .
        qed
      then show integral (cbox a b) ( $\lambda x. norm (f x)$ ) < SDF + e
        by simp
      qed
    qed (use K in auto)
  qed
  moreover have  $\bigwedge a\ b. (\lambda x. norm (f x))\ integrable\_on\ cbox\ a\ b$ 
    using absolutely_integrable_on_def f_int by auto
  ultimately
  have  $((\lambda x. norm (f x))\ has\_integral\ SDF)\ UNIV$ 
    by (auto simp: has_integral_alt')
  then show  $(\lambda x. norm (f x))\ integrable\_on\ UNIV$ 
    by blast
  qed

```

9.8.15 Outer and inner approximation of measurable sets by well-behaved sets.

proposition *measurable_outer_intervals_bounded*:

assumes $S \in \text{lmeasurable } S \subseteq \text{cbox } a \ b \ e > 0$

obtains \mathcal{D}

where *countable* \mathcal{D}

$\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \ b \wedge K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$

pairwise $(\lambda A \ B. \text{interior } A \cap \text{interior } B = \{\}) \ \mathcal{D}$

$\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$

$\bigwedge K. \llbracket K \in \mathcal{D}; \text{box } a \ b \neq \{\} \rrbracket \implies \text{interior } K \neq \{\}$

$S \subseteq \bigcup \mathcal{D} \cup \mathcal{D} \in \text{lmeasurable } \text{measure lebesgue } (\bigcup \mathcal{D}) \leq \text{measure lebesgue } S$

+ e

proof (*cases* $\text{box } a \ b = \{\}$)

case *True*

show *?thesis*

proof (*cases* $\text{cbox } a \ b = \{\}$)

case *True*

with *assms* **have** [*simp*]: $S = \{\}$

by *auto*

show *?thesis*

proof

show *countable* $\{\}$

by *simp*

qed (*use* $\langle e > 0 \rangle$ **in** *auto*)

next

case *False*

show *?thesis*

proof

show *countable* $\{\text{cbox } a \ b\}$

by *simp*

show $\bigwedge u \ v. \text{cbox } u \ v \in \{\text{cbox } a \ b\} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i - a \cdot i) / 2^n$

using *False* **by** (*force* *simp*: *eq_cbox* *intro*: *exI* [**where** $x=0$])

show $\text{measure lebesgue } (\bigcup \{\text{cbox } a \ b\}) \leq \text{measure lebesgue } S + e$

using *assms* **by** (*simp* *add*: *sum_content.box_empty_imp* [*OF* *True*])

qed (*use* *assms* $\langle \text{cbox } a \ b \neq \{\} \rangle$ **in** *auto*)

qed

next

case *False*

let $? \mu = \text{measure lebesgue}$

have $S \cap \text{cbox } a \ b \in \text{lmeasurable}$

using $\langle S \in \text{lmeasurable} \rangle$ **by** *blast*

then **have** *indS_int*: (*indicator* S *has_integral* $(? \mu \ S)$) $(\text{cbox } a \ b)$

by (*metis* *integral_indicator* $\langle S \subseteq \text{cbox } a \ b \rangle$ *has_integral_integrable_integral* *inf.orderE* *integrable_on_indicator*)

with $\langle e > 0 \rangle$ **obtain** γ **where** *gauge* γ **and** γ :

$\bigwedge \mathcal{D}. \llbracket \mathcal{D} \text{ tagged_division_of } (\text{cbox } a \ b); \gamma \text{ fine } \mathcal{D} \rrbracket \implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content}(K) *_{\mathbb{R}} \text{indicator } S \ x) - ? \mu \ S) < e$

by (*force* *simp*: *has_integral*)

```

have integ: integral (cbox a b) (indicator S) = integral UNIV (indicator S)
using assms by (metis has_integral_iff indS_int lmeasure_integral_UNIV)
obtain  $\mathcal{D}$  where  $\mathcal{D}$ : countable  $\mathcal{D} \cup \mathcal{D} \subseteq \text{cbox } a \ b$ 
and cbox:  $\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
and djointish: pairwise ( $\lambda A \ B. \text{interior } A \cap \text{interior } B = \{\}$ )  $\mathcal{D}$ 
and covered:  $\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq \gamma \ x$ 
and close:  $\bigwedge u \ v. \text{cbox } u \ v \in \mathcal{D} \implies \exists n. \forall i \in \text{Basis}. v \cdot i - u \cdot i = (b \cdot i -$ 
 $a \cdot i) / 2^n$ 
and covers:  $S \subseteq \bigcup \mathcal{D}$ 
using covering_lemma [of  $S \ a \ b \ \gamma$ ]  $\langle \text{gauge } \gamma \rangle \langle \text{box } a \ b \neq \{\} \rangle$  assms by force
show ?thesis
proof
show  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \ b \wedge K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
by (meson Sup_le_iff  $\mathcal{D}(2)$  cbox_interior_empty)
have negl_int: negligible( $K \cap L$ ) if  $K \in \mathcal{D} \ L \in \mathcal{D} \ K \neq L$  for  $K \ L$ 
proof -
have interior  $K \cap \text{interior } L = \{\}$ 
using djointish pairwise $\mathcal{D}$  that by fastforce
moreover obtain  $u \ v \ x \ y$  where  $K = \text{cbox } u \ v \ L = \text{cbox } x \ y$ 
using cbox  $\langle K \in \mathcal{D} \rangle \langle L \in \mathcal{D} \rangle$  by blast
ultimately show ?thesis
by (simp add: Int_interval box_Int_box negligible_interval(1))
qed
have fincase:  $\bigcup \mathcal{F} \in \text{lmeasurable} \wedge ?\mu (\bigcup \mathcal{F}) \leq ?\mu S + e$  if finite  $\mathcal{F} \ \mathcal{F} \subseteq \mathcal{D}$ 
for  $\mathcal{F}$ 
proof -
obtain  $t$  where  $t$ :  $\bigwedge K. K \in \mathcal{F} \implies t \ K \in S \cap K \wedge K \subseteq \gamma(t \ K)$ 
using covered  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  subset $\mathcal{D}$  by metis
have  $\forall K \in \mathcal{F}. \forall L \in \mathcal{F}. K \neq L \implies \text{interior } K \cap \text{interior } L = \{\}$ 
using that djointish by (simp add: pairwise_def) (metis subset $\mathcal{D}$ )
with cbox that  $\mathcal{D}$  have  $\mathcal{F} \text{ div: } \mathcal{F} \text{ division\_of } (\bigcup \mathcal{F})$ 
by (fastforce simp: division_of_def dest: cbox)
then have 1:  $\bigcup \mathcal{F} \in \text{lmeasurable}$ 
by blast
have norme:  $\bigwedge p. \llbracket p \text{ tagged\_division\_of } \text{cbox } a \ b; \gamma \text{ fine } p \rrbracket$ 
 $\implies \text{norm } ((\sum (x,K) \in p. \text{content } K * \text{indicator } S \ x) - \text{integral } (\text{cbox } a \ b)$ 
 $(\text{indicator } S)) < e$ 
by (auto simp: lmeasure_integral_UNIV assms integ dest:  $\gamma$ )
have  $\forall x \ K \ y \ L. (x,K) \in (\lambda K. (t \ K, K)) \wedge (y,L) \in (\lambda K. (t \ K, K)) \wedge \mathcal{F} \wedge$ 
 $(x,K) \neq (y,L) \implies \text{interior } K \cap \text{interior } L = \{\}$ 
using that djointish by (clarsimp simp: pairwise_def) (metis subset $\mathcal{D}$ )
with that  $\mathcal{D}$  have tagged:  $(\lambda K. (t \ K, K)) \wedge \mathcal{F} \text{ tagged\_partial\_division\_of}$ 
 $\text{cbox } a \ b$ 
by (auto simp: tagged_partial_division_of_def dest:  $t \ \text{cbox}$ )
have fine:  $\gamma \text{ fine } (\lambda K. (t \ K, K)) \wedge \mathcal{F}$ 
using  $t$  by (auto simp: fine_def)
have *:  $y \leq ?\mu S \implies |x - y| \leq e \implies x \leq ?\mu S + e$  for  $x \ y$ 
by arith
have  $?\mu (\bigcup \mathcal{F}) \leq ?\mu S + e$ 

```

```

proof (rule *)
  have  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) = ?\mu (\bigcup C \in \mathcal{F}. C \cap S)$ 
  proof (rule measure_negligible_finite_Union_image [OF ‹finite  $\mathcal{F}$ ›, symmetric])
    show  $\bigwedge K. K \in \mathcal{F} \implies K \cap S \in \text{lmeasurable}$ 
      using  $\mathcal{F} \text{ div } \langle S \in \text{lmeasurable} \rangle$  by blast
    show pairwise  $(\lambda K y. \text{negligible} (K \cap S \cap (y \cap S))) \mathcal{F}$ 
      unfolding pairwise_def
      by (metis inf.commute inf_sup_aci(3) negligible_Int subsetCE negl_int
 $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$ )
    qed
  also have  $\dots = ?\mu (\bigcup \mathcal{F} \cap S)$ 
    by simp
  also have  $\dots \leq ?\mu S$ 
    by (simp add: 1 ‹ $S \in \text{lmeasurable}$ › fmeasurableD measure_mono_fmeasurable sets.Int)
  finally show  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) \leq ?\mu S$  .
next
  have  $?\mu (\bigcup \mathcal{F}) = \text{sum } ?\mu \mathcal{F}$ 
    by (metis  $\mathcal{F} \text{ div content\_division}$ )
  also have  $\dots = (\sum K \in \mathcal{F}. \text{content } K)$ 
    using  $\mathcal{F} \text{ div}$  by (force intro: sum.cong)
  also have  $\dots = (\sum x \in \mathcal{F}. \text{content } x * \text{indicator } S (t x))$ 
    using  $t$  by auto
  finally have eq1:  $?\mu (\bigcup \mathcal{F}) = (\sum x \in \mathcal{F}. \text{content } x * \text{indicator } S (t x))$  .
  have eq2:  $(\sum K \in \mathcal{F}. ?\mu (K \cap S)) = (\sum K \in \mathcal{F}. \text{integral } K (\text{indicator } S))$ 
    apply (rule sum.cong [OF refl])
    by (metis integral_indicator  $\mathcal{F} \text{ div } \langle S \in \text{lmeasurable} \rangle \text{ division\_ofD}$ (4)
fmeasurable.Int inf.commute lmeasurable_cbox)
  have  $|\sum (x, K) \in (\lambda K. (t K, K)) ' \mathcal{F}. \text{content } K * \text{indicator } S x - \text{integral } K (\text{indicator } S)| \leq e$ 
    using Henstock_lemma_part1 [of indicator S::'a $\implies$ real, OF _ ‹ $e > 0$ ›
 $\langle \text{gauge } \gamma \rangle$  _ tagged fine]
    indS_int norme by auto
  then show  $|\sum ?\mu (\bigcup \mathcal{F}) - (\sum K \in \mathcal{F}. ?\mu (K \cap S))| \leq e$ 
    by (simp add: eq1 eq2 comm_monoid_add_class.sum.reindex inj_on_def sum_subtractf)
  qed
  with 1 show ?thesis by blast
qed
proof  $\bigcup \mathcal{D} \in \text{lmeasurable} \wedge ?\mu (\bigcup \mathcal{D}) \leq ?\mu S + e$ 
  proof (cases finite  $\mathcal{D}$ )
    case True
      with fincase show ?thesis
        by blast
    next
      case False
        let  $?T = \text{from\_nat\_into } \mathcal{D}$ 
        have  $T: \text{bij\_betw } ?T \text{ UNIV } \mathcal{D}$ 

```

```

  by (simp add: False  $\mathcal{D}(1)$  bij_betw_from_nat_into)
  have TM:  $\bigwedge n. ?T n \in \text{lmeasurable}$ 
  by (metis False cbox finite.emptyI from_nat_into lmeasurable_cbox)
  have TN:  $\bigwedge m n. m \neq n \implies \text{negligible } (?T m \cap ?T n)$ 
  by (simp add: False  $\mathcal{D}(1)$  from_nat_into infinite_imp_nonempty negl_int)
  have TB:  $(\sum_{k \leq n}. ?\mu (?T k)) \leq ?\mu S + e$  for  $n$ 
  proof -
    have  $(\sum_{k \leq n}. ?\mu (?T k)) = ?\mu (\bigcup (?T ' \{..n\}))$ 
    by (simp add: pairwise_def TM TN measure_negligible_finite_Union_image)
    also have  $?\mu (\bigcup (?T ' \{..n\})) \leq ?\mu S + e$ 
    using fincase [of  $?T ' \{..n\}$ ] T by (auto simp: bij_betw_def)
    finally show ?thesis .
  qed
  have  $\bigcup \mathcal{D} \in \text{lmeasurable}$ 
  by (metis lmeasurable_compact T  $\mathcal{D}(2)$  bij_betw_def cbox compact_cbox
  countable_Un_Int(1) fmeasurableD fmeasurableI2 rangeI)
  moreover
  have  $?\mu (\bigcup x. \text{from\_nat\_into } \mathcal{D} x) \leq ?\mu S + e$ 
  proof (rule measure_countable_Union_le [OF TM])
    show  $?\mu (\bigcup_{x \leq n}. \text{from\_nat\_into } \mathcal{D} x) \leq ?\mu S + e$  for  $n$ 
    by (metis (mono_tags, lifting) False fincase finite.emptyI finite_atMost
  finite_imageI from_nat_into imageE subsetI)
  qed
  ultimately show ?thesis by (metis T bij_betw_def)
  qed
  then show  $\bigcup \mathcal{D} \in \text{lmeasurable measure lebesgue } (\bigcup \mathcal{D}) \leq ?\mu S + e$  by blast+
  qed (use  $\mathcal{D}$  cbox djointish close covers in auto)
  qed

```

9.8.16 Transformation of measure by linear maps

lemma *emeasure_lebesgue_ball_conv_unit_ball*:

fixes $c :: 'a :: \text{euclidean_space}$

assumes $r \geq 0$

shows $\text{emeasure lebesgue } (\text{ball } c r) =$
 $\text{ennreal } (r \wedge \text{DIM}('a)) * \text{emeasure lebesgue } (\text{ball } (0 :: 'a) 1)$

proof (cases $r = 0$)

case False

with *assms* have $r: r > 0$ by auto

have $\text{emeasure lebesgue } ((\lambda x. c + x) ' (\lambda x. r *_R x) ' \text{ball } (0 :: 'a) 1) =$
 $r \wedge \text{DIM}('a) * \text{emeasure lebesgue } (\text{ball } (0 :: 'a) 1)$

unfolding *image_image* using *emeasure_lebesgue_affine*[of $r c \text{ball } 0 1$] *assms*

by (simp add: *add_ac*)

also have $(\lambda x. r *_R x) ' \text{ball } 0 1 = \text{ball } (0 :: 'a) r$

using r by (*subst ball_scale*) auto

also have $(\lambda x. c + x) ' \dots = \text{ball } c r$

by (*subst image_add_ball*) (*simp_all add: algebra_simps*)

finally show ?thesis by *simp*

qed auto

lemma *content_ball_conv_unit_ball*:

fixes $c :: 'a :: euclidean_space$

assumes $r \geq 0$

shows $\text{content } (\text{ball } c \ r) = r \wedge \text{DIM}('a) * \text{content } (\text{ball } (0 :: 'a) \ 1)$

proof –

have $\text{ennreal } (\text{content } (\text{ball } c \ r)) = \text{emeasure lebesgue } (\text{ball } c \ r)$

using *emeasure_lborel_ball_finite*[of $c \ r$] **by** (*subst emeasure_eq_ennreal_measure*)

auto

also have $\dots = \text{ennreal } (r \wedge \text{DIM}('a)) * \text{emeasure lebesgue } (\text{ball } (0 :: 'a) \ 1)$

using *assms* **by** (*intro emeasure_lebesgue_ball_conv_unit_ball*) *auto*

also have $\dots = \text{ennreal } (r \wedge \text{DIM}('a)) * \text{content } (\text{ball } (0 :: 'a) \ 1)$

using *emeasure_lborel_ball_finite*[of $0 :: 'a \ 1$] *assms*

by (*subst emeasure_eq_ennreal_measure*) (*auto simp: ennreal_mult'*)

finally show *?thesis*

using *assms* **by** (*subst (asm) ennreal_inj*) *auto*

qed

lemma *measurable_linear_image_interval*:

$\text{linear } f \implies f' \ (\text{cbox } a \ b) \in \text{lmeasurable}$

by (*metis bounded_linear_image_linear_linear_bounded_cbox_closure_bounded_linear_image_closure_cbox_compact_closure_lmeasurable_compact*)

proposition *measure_linear_sufficient*:

fixes $f :: 'n :: euclidean_space \Rightarrow 'n$

assumes *linear f* **and** $S: S \in \text{lmeasurable}$

and $\text{im}: \bigwedge a \ b. \text{measure lebesgue } (f' \ (\text{cbox } a \ b)) = m * \text{measure lebesgue } (\text{cbox } a \ b)$

shows $f' \ S \in \text{lmeasurable} \wedge m * \text{measure lebesgue } S = \text{measure lebesgue } (f' \ S)$

using *le_less_linear* [of $0 \ m$]

proof

assume $m < 0$

then show *?thesis*

using *im* [of $0 \ \text{One}$] **by** *auto*

next

assume $m \geq 0$

let $?\mu = \text{measure lebesgue}$

show *?thesis*

proof (*cases inj f*)

case *False*

then have $?\mu \ (f' \ S) = 0$

using $\langle \text{linear } f \rangle$ *negligible_imp_measure0 negligible_linear_singular_image*

by *blast*

then have $m * ?\mu \ (\text{cbox } 0 \ (\text{One})) = 0$

by (*metis False* $\langle \text{linear } f \rangle$ *cbox_borel content_unit im measure_completion negligible_imp_measure0 negligible_linear_singular_image sets_lborel*)

then show *?thesis*

using $\langle \text{linear } f \rangle$ *negligible_linear_singular_image negligible_imp_measure0*

False

```

    by (auto simp: lmeasurable_iff_has_integral negligible_UNIV)
  next
    case True
    then obtain h where linear h and hf:  $\bigwedge x. h (f x) = x$  and fh:  $\bigwedge x. f (h x) = x$ 
    using <linear f> linear_injective_isomorphism by blast
    have fBS:  $(f \text{ ' } S) \in \text{lmeasurable} \wedge m * ?\mu S = ?\mu (f \text{ ' } S)$ 
    if bounded  $S \in \text{lmeasurable}$  for  $S$ 
    proof -
      obtain a b where  $S \subseteq \text{cbox } a \ b$ 
      using <bounded S> bounded_subset_cbox_symmetric by metis
      have fUD:  $(f \text{ ' } \bigcup \mathcal{D}) \in \text{lmeasurable} \wedge ?\mu (f \text{ ' } \bigcup \mathcal{D}) = (m * ?\mu (\bigcup \mathcal{D}))$ 
      if countable  $\mathcal{D}$ 
      and cbox:  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \ b \wedge K \neq \{\} \wedge (\exists c \ d. K = \text{cbox } c \ d)$ 
      and intint: pairwise  $(\lambda A \ B. \text{interior } A \cap \text{interior } B = \{\}) \ \mathcal{D}$ 
      for  $\mathcal{D}$ 
      proof -
        have conv:  $\bigwedge K. K \in \mathcal{D} \implies \text{convex } K$ 
        using cbox_convex_box(1) by blast
        have neg: negligible  $(g \text{ ' } K \cap g \text{ ' } L)$  if linear  $g \ K \in \mathcal{D} \ L \in \mathcal{D} \ K \neq L$ 
        for  $K \ L$  and  $g :: 'n \implies 'n$ 
        proof (cases inj g)
          case True
          have negligible  $(\text{frontier}(g \text{ ' } K \cap g \text{ ' } L) \cup \text{interior}(g \text{ ' } K \cap g \text{ ' } L))$ 
          proof (rule negligible_Un)
            show negligible  $(\text{frontier } (g \text{ ' } K \cap g \text{ ' } L))$ 
            by (simp add: negligible_convex_frontier convex_Int conv_convex_linear_image that)
          next
            have  $\forall p \ N. \text{pairwise } p \ N = (\forall Na. (Na::'n \text{ set}) \in N \longrightarrow (\forall Nb. Nb \in N \wedge Na \neq Nb \longrightarrow p \ Na \ Nb))$ 
            by (metis pairwise_def)
            then have interior  $K \cap \text{interior } L = \{\}$ 
            using intint that(2) that(3) that(4) by presburger
            then show negligible  $(\text{interior } (g \text{ ' } K \cap g \text{ ' } L))$ 
            by (metis True empty_imp_negligible image_Int image_empty interior_Int interior_injective_linear_image that(1))
          qed
          moreover have  $g \text{ ' } K \cap g \text{ ' } L \subseteq \text{frontier } (g \text{ ' } K \cap g \text{ ' } L) \cup \text{interior } (g \text{ ' } K \cap g \text{ ' } L)$ 
          by (metis Diff_partition Int_commute calculation closure_Un_frontier frontier_def inf.absorb_iff2 inf_bot_right inf_sup_absorb negligible_Un_eq open_interior open_not_negligible sup_commute)
          ultimately show ?thesis
          by (rule negligible_subset)
        next
          case False
          then show ?thesis

```

```

    by (simp add: negligible_Int negligible_linear_singular_image ⟨linear g⟩)
  qed
  have negf: negligible ((f ' K) ∩ (f ' L))
  and negid: negligible (K ∩ L) if K ∈ D L ∈ D K ≠ L for K L
    using neg [OF ⟨linear f⟩] neg [OF linear_id] that by auto
  show ?thesis
  proof (cases finite D)
    case True
      then have ?μ (⋃ x∈D. f ' x) = (∑ x∈D. ?μ (f ' x))
        using ⟨linear f⟩ cbox measurable_linear_image_interval negf
        by (blast intro: measure_negligible_finite_Union_image [unfolded pairwise_def])
      also have ... = (∑ k∈D. m * ?μ k)
        by (metis (no_types, lifting) cbox im sum.cong)
      also have ... = m * ?μ (⋃ D)
        unfolding sum_distrib_left [symmetric]
        by (metis True cbox lmeasurable_cbox measure_negligible_finite_Union [unfolded pairwise_def] negid)
      finally show ?thesis
        by (metis True ⟨linear f⟩ cbox image_Union fmeasurable.finite_UN measurable_linear_image_interval)
    next
      case False
        with ⟨countable D⟩ obtain X :: nat ⇒ 'n set where S: bij_betw X UNIV
        D
          using bij_betw_from_nat_into by blast
        then have eq: (⋃ D) = (⋃ n. X n) (f ' ⋃ D) = (⋃ n. f ' X n)
          by (auto simp: bij_betw_def)
        have meas: ∧K. K ∈ D ⇒ K ∈ lmeasurable
          using cbox by blast
        with S have 1: ∧n. X n ∈ lmeasurable
          by (auto simp: bij_betw_def)
        have 2: pairwise (λm n. negligible (X m ∩ X n)) UNIV
          using S unfolding bij_betw_def pairwise_def by (metis injD negid range_eqI)
        have bounded (⋃ D)
          by (meson Sup_least bounded_cbox bounded_subset cbox)
        then have 3: bounded (⋃ n. X n)
          using S unfolding bij_betw_def by blast
        have (⋃ n. X n) ∈ lmeasurable
          by (rule measurable_countable_negligible_Union_bounded [OF 1 2 3])
        with S have f1: ∧n. f ' (X n) ∈ lmeasurable
        unfolding bij_betw_def by (metis assms(1) cbox measurable_linear_image_interval rangeI)
        have f2: pairwise (λm n. negligible (f ' (X m) ∩ f ' (X n))) UNIV
          using S unfolding bij_betw_def pairwise_def by (metis injD negf rangeI)
        have bounded (⋃ D)
          by (meson Sup_least bounded_cbox bounded_subset cbox)

```



```

then have f3: bounded ( $\bigcup n. f \text{ ' } X n$ )
  using S unfolding bij_betw_def
  by (metis bounded_linear_image linear_linear assms(1) image_Union
range_composition)
have ( $\lambda n. ?\mu (X n)$ ) sums  $?\mu (\bigcup n. X n)$ 
  by (rule measure_countable_negligible_Union_bounded [OF 1 2 3])
have meq:  $?\mu (\bigcup n. f \text{ ' } X n) = m * ?\mu (\bigcup (X \text{ ' } UNIV))$ 
proof (rule sums_unique2 [OF measure_countable_negligible_Union_bounded
[OF f1 f2 f3]])
  have m:  $\bigwedge n. ?\mu (f \text{ ' } X n) = (m * ?\mu (X n))$ 
    using S unfolding bij_betw_def by (metis cbox_im_rangeI)
  show ( $\lambda n. ?\mu (f \text{ ' } X n)$ ) sums  $(m * ?\mu (\bigcup (X \text{ ' } UNIV)))$ 
    unfolding m
    using measure_countable_negligible_Union_bounded [OF 1 2 3]
sums_mult by blast
  qed
  show ?thesis
    using measurable_countable_negligible_Union_bounded [OF f1 f2 f3]
meq
  by (auto simp: eq [symmetric])
qed
qed
show ?thesis
  unfolding completion.fmeasurable_measure_inner_outer_le
proof (intro conjI allI impI)
  fix e :: real
  assume e > 0
  have 1: cbox a b - S  $\in$  lmeasurable
    by (simp add: fmeasurable.Diff that)
  have 2:  $0 < e / (1 + |m|)$ 
    using  $\langle e > 0 \rangle$  by (simp add: field_split_simps abs_add_one_gt_zero)
  obtain  $\mathcal{D}$ 
    where countable  $\mathcal{D}$ 
    and cbox:  $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \text{ } b \wedge K \neq \{\} \wedge (\exists c \text{ } d. K = \text{cbox}$ 
c d)
    and intdisj: pairwise ( $\lambda A \text{ } B. \text{interior } A \cap \text{interior } B = \{\}$ )  $\mathcal{D}$ 
    and DD:  $\text{cbox } a \text{ } b - S \subseteq \bigcup \mathcal{D} \cup \mathcal{D} \in$  lmeasurable
    and le:  $?\mu (\bigcup \mathcal{D}) \leq ?\mu (\text{cbox } a \text{ } b - S) + e / (1 + |m|)$ 
    by (rule measurable_outer_intervals_bounded [of cbox a b - S a b e / (1
+ |m|)]; use 1 2 pairwise_def in force)
  show  $\exists T \in$  lmeasurable.  $T \subseteq f \text{ ' } S \wedge m * ?\mu S - e \leq ?\mu T$ 
proof (intro bexI conjI)
  show  $f \text{ ' } (\text{cbox } a \text{ } b) - f \text{ ' } (\bigcup \mathcal{D}) \subseteq f \text{ ' } S$ 
    using  $\langle \text{cbox } a \text{ } b - S \subseteq \bigcup \mathcal{D} \rangle$  by force
  have  $m * ?\mu S - e \leq m * (?\mu S - e / (1 + |m|))$ 
    using  $\langle m \geq 0 \rangle \langle e > 0 \rangle$  by (simp add: field_simps)
  also have  $\dots \leq ?\mu (f \text{ ' } \text{cbox } a \text{ } b) - ?\mu (f \text{ ' } (\bigcup \mathcal{D}))$ 
proof -
  have  $?\mu (\text{cbox } a \text{ } b - S) = ?\mu (\text{cbox } a \text{ } b) - ?\mu S$ 

```

```

    by (simp add: measurable_measure_Diff ⟨S ⊆ cbox a b⟩ fmeasurableD
that(2))
    then have (?μ S - e / (1 + m)) ≤ (content (cbox a b) - ?μ (⋃ D))
      using ⟨m ≥ 0⟩ le by auto
    then show ?thesis
      using ⟨m ≥ 0⟩ ⟨e > 0⟩
      by (simp add: mult_left_mono in fUD [OF ⟨countable D⟩ cbox intdisj]
flip: right_diff_distrib)
    qed
    also have ... = ?μ (f' cbox a b - f' ⋃ D)
    proof (rule measurable_measure_Diff [symmetric])
      show f' cbox a b ∈ lmeasurable
        by (simp add: asms(1) measurable_linear_image_interval)
      show f' ⋃ D ∈ sets lebesgue
        by (simp add: ⟨countable D⟩ cbox fUD fmeasurableD intdisj)
      show f' ⋃ D ⊆ f' cbox a b
        by (simp add: Sup_le_iff cbox image_mono)
    qed
    finally show m * ?μ S - e ≤ ?μ (f' cbox a b - f' ⋃ D) .
    show f' cbox a b - f' ⋃ D ∈ lmeasurable
      by (simp add: fUD ⟨countable D⟩ ⟨linear f⟩ cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
    qed
  next
  fix e :: real
  assume e > 0
  have em: 0 < e / (1 + |m|)
    using ⟨e > 0⟩ by (simp add: field_split_simps abs_add_one_gt_zero)
  obtain D
    where countable D
      and cbox: ⋀K. K ∈ D ⟹ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃ c d. K = cbox
c d)
      and intdisj: pairwise (λA B. interior A ∩ interior B = {}) D
      and DD: S ⊆ ⋃ D ∧ ⋃ D ∈ lmeasurable
      and le: ?μ (⋃ D) ≤ ?μ S + e / (1 + |m|)
    by (rule measurable_outer_intervals_bounded [of S a b e / (1 + |m|)]; use
⟨S ∈ lmeasurable⟩ ⟨S ⊆ cbox a b⟩ em in force)
  show ∃ U ∈ lmeasurable. f' S ⊆ U ∧ ?μ U ≤ m * ?μ S + e
  proof (intro bexI conjI)
    show f' S ⊆ f' (⋃ D)
      by (simp add: DD(1) image_mono)
    have ?μ (f' ⋃ D) ≤ m * (?μ S + e / (1 + |m|))
      using ⟨m ≥ 0⟩ le mult_left_mono
      by (auto simp: fUD ⟨countable D⟩ ⟨linear f⟩ cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
    also have ... ≤ m * ?μ S + e
      using ⟨m ≥ 0⟩ ⟨e > 0⟩ by (simp add: fUD [OF ⟨countable D⟩ cbox
intdisj] field_simps)
    finally show ?μ (f' ⋃ D) ≤ m * ?μ S + e .

```

```

    show  $f \upharpoonright \bigcup D \in \text{lmeasurable}$ 
      by (simp add: ‹countable D› cbox fUD intdisj)
  qed
qed
qed
show ?thesis
  unfolding has_measure_limit_iff
proof (intro allI impI)
  fix  $e :: \text{real}$ 
  assume  $e > 0$ 
  obtain  $B$  where  $B > 0$  and  $B$ :
     $\bigwedge a b. \text{ball } 0 B \subseteq \text{cbox } a b \implies |\int ?\mu (S \cap \text{cbox } a b) - \int ?\mu S| < e / (1 + |m|)$ 
    using has_measure_limit [OF S] ‹ $e > 0$ › by (metis abs_add_one_gt_zero
zero_less_divide_iff)
  obtain  $c d :: 'n$  where  $cd: \text{ball } 0 B \subseteq \text{cbox } c d$ 
    by (metis bounded_subset_cbox_symmetric_bounded_ball)
  with  $B$  have less:  $|\int ?\mu (S \cap \text{cbox } c d) - \int ?\mu S| < e / (1 + |m|)$  .
  obtain  $D$  where  $D > 0$  and  $D: \text{cbox } c d \subseteq \text{ball } 0 D$ 
    by (metis bounded_cbox_bounded_subset_ballD)
  obtain  $C$  where  $C > 0$  and  $C: \bigwedge x. \text{norm } (f x) \leq C * \text{norm } x$ 
    using linear_bounded_pos ‹linear f› by blast
  have  $f \upharpoonright S \cap \text{cbox } a b \in \text{lmeasurable} \wedge$ 
     $|\int ?\mu (f \upharpoonright S \cap \text{cbox } a b) - m * \int ?\mu S| < e$ 
    if  $\text{ball } 0 (D * C) \subseteq \text{cbox } a b$  for  $a b$ 
  proof -
    have bounded  $(S \cap h \upharpoonright \text{cbox } a b)$ 
      by (simp add: bounded_linear_image_linear_linear ‹linear h› bounded_Int)
    moreover have  $\text{Shab}: S \cap h \upharpoonright \text{cbox } a b \in \text{lmeasurable}$ 
      by (simp add: S ‹linear h› fmeasurable.Int measurable_linear_image_interval)
    moreover have  $\text{fim}: f \upharpoonright (S \cap h \upharpoonright (\text{cbox } a b)) = (f \upharpoonright S) \cap \text{cbox } a b$ 
      by (auto simp: hf_rev_image_eqI fh)
    ultimately have 1:  $(f \upharpoonright S) \cap \text{cbox } a b \in \text{lmeasurable}$ 
      and 2:  $\int ?\mu ((f \upharpoonright S) \cap \text{cbox } a b) = m * \int ?\mu (S \cap h \upharpoonright \text{cbox } a b)$ 
      using fBS [of  $S \cap (h \upharpoonright (\text{cbox } a b))$ ] by auto
    have *:  $\llbracket |z - m| < e; z \leq w; w \leq m \rrbracket \implies |w - m| \leq e$ 
      for  $w z m$  and  $e :: \text{real}$  by auto
    have meas_adiff:  $|\int ?\mu (S \cap h \upharpoonright \text{cbox } a b) - \int ?\mu S| \leq e / (1 + |m|)$ 
      proof (rule * [OF less])
        show  $\int ?\mu (S \cap \text{cbox } c d) \leq \int ?\mu (S \cap h \upharpoonright \text{cbox } a b)$ 
          proof (rule measure_mono_fmeasurable [OF __ Shab])
            have  $f \upharpoonright \text{ball } 0 D \subseteq \text{ball } 0 (C * D)$ 
              using  $C$  ‹ $C > 0$ ›
              apply (clarsimp simp: algebra_simps)
            by (meson le_less_trans linordered_comm_semiring_strict_class.comm_mult_strict_left_mono)
            then have  $f \upharpoonright \text{ball } 0 D \subseteq \text{cbox } a b$ 
              by (metis mult.commute order_trans that)
            have  $\text{ball } 0 D \subseteq h \upharpoonright \text{cbox } a b$ 
              by (metis ‹ $f \upharpoonright \text{ball } 0 D \subseteq \text{cbox } a b$ › hf_image_subset_iff subsetI)
            then show  $S \cap \text{cbox } c d \subseteq S \cap h \upharpoonright \text{cbox } a b$ 

```

```

      using D by blast
    next
      show S ∩ cbox c d ∈ sets lebesgue
      using S fmeasurable_cbox by blast
    qed
  next
    show ?μ (S ∩ h ' cbox a b) ≤ ?μ S
    by (simp add: S Shab fmeasurableD measure_mono_fmeasurable)
  qed
  have |?μ (f ' S ∩ cbox a b) - m * ?μ S| ≤ |?μ S - ?μ (S ∩ h ' cbox a b)|
* m
    by (metis 2 ⟨m ≥ 0⟩ abs_minus_commute abs_mult_pos mult.commute
order_refl right_diff_distrib')
  also have ... ≤ e / (1 + m) * m
    by (metis ⟨m ≥ 0⟩ abs_minus_commute abs_of_nonneg meas_adiff
mult.commute mult_left_mono)
  also have ... < e
    using ⟨e > 0⟩ ⟨m ≥ 0⟩ by (simp add: field_simps)
  finally have |?μ (f ' S ∩ cbox a b) - m * ?μ S| < e .
  with 1 show ?thesis by auto
  qed
  then show ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
    f ' S ∩ cbox a b ∈ lmeasurable ∧
    |?μ (f ' S ∩ cbox a b) - m * ?μ S| < e
  using ⟨C > 0⟩ ⟨D > 0⟩ by (metis mult_zero_left mult_less_cancel_right_pos)
  qed
  qed
  qed

```

9.8.17 Lemmas about absolute integrability

lemma *absolutely_integrable_linear*:

```

  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  and h :: 'n::euclidean_space ⇒ 'p::euclidean_space
  shows f absolutely_integrable_on s ⟹ bounded_linear h ⟹ (h ∘ f) abso-
lutely_integrable_on s
  using integrable_bounded_linear[of h lebesgue λx. indicator s x *R f x]
  by (simp add: linear_simps[of h] set_integrable_def)

```

lemma *absolutely_integrable_sum*:

```

  fixes f :: 'a ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes finite T and ⋀ a. a ∈ T ⟹ (f a) absolutely_integrable_on S
  shows (λx. sum (λa. f a x) T) absolutely_integrable_on S
  using assms by induction auto

```

lemma *absolutely_integrable_integrable_bound*:

```

  fixes f :: 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes le: ⋀ x. x ∈ S ⟹ norm (f x) ≤ g x and f: f integrable_on S and g: g
integrable_on S

```

```

shows  $f$  absolutely_integrable_on  $S$ 
  unfolding set_integrable_def
proof (rule Bochner_Integration.integrable_bound)
  have  $g$  absolutely_integrable_on  $S$ 
    unfolding absolutely_integrable_on_def
  proof
    show  $(\lambda x. \text{norm } (g\ x))$  integrable_on  $S$ 
      using le_norm_ge_zero[of f _]
      by (intro integrable_spike_finite[OF _ _ g, of {}])
        (auto intro!: abs_of_nonneg intro: order_trans simp del: norm_ge_zero)
    qed fact
  then show integrable_lebesgue  $(\lambda x. \text{indicat\_real } S\ x\ *_{\mathbb{R}}\ g\ x)$ 
    by (simp add: set_integrable_def)
  show  $(\lambda x. \text{indicat\_real } S\ x\ *_{\mathbb{R}}\ f\ x) \in \text{borel\_measurable\_lebesgue}$ 
    using  $f$  by (auto intro: has_integral_implies_lebesgue_measurable simp: integrable_on_def)
  qed (use le in  $\langle$ force intro!: always_eventually_split: split_indicator $\rangle$ )

```

```

corollary absolutely_integrable_on_const [simp]:
  fixes  $c :: 'a::\text{euclidean\_space}$ 
  assumes  $S \in \text{lmeasurable}$ 
  shows  $(\lambda x. c)$  absolutely_integrable_on  $S$ 
  by (metis (full_types) assms absolutely_integrable_integrable_bound integrable_on_const order_refl)

```

```

lemma absolutely_integrable_continuous:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  shows continuous_on (cbox  $a\ b$ )  $f \implies f$  absolutely_integrable_on cbox  $a\ b$ 
  using absolutely_integrable_integrable_bound
  by (simp add: absolutely_integrable_on_def continuous_on_norm integrable_continuous)

```

```

lemma absolutely_integrable_continuous_real:
  fixes  $f :: \text{real} \Rightarrow 'b::\text{euclidean\_space}$ 
  shows continuous_on  $\{a..b\}$   $f \implies f$  absolutely_integrable_on  $\{a..b\}$ 
  by (metis absolutely_integrable_continuous box_real(2))

```

```

lemma continuous_imp_integrable:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes continuous_on (cbox  $a\ b$ )  $f$ 
  shows integrable (lebesgue_on (cbox  $a\ b$ ))  $f$ 
proof –
  have  $f$  absolutely_integrable_on cbox  $a\ b$ 
    by (simp add: absolutely_integrable_continuous assms)
  then show ?thesis
    by (simp add: integrable_restrict_space set_integrable_def)
qed

```

```

lemma continuous_imp_integrable_real:
  fixes  $f :: \text{real} \Rightarrow 'b::\text{euclidean\_space}$ 

```

```

assumes continuous_on {a..b} f
shows integrable (lebesgue_on {a..b}) f
by (metis assms continuous_imp_integrable interval_cbox)

```

9.8.18 Componentwise

proposition *absolutely_integrable_componentwise_iff*:

```

shows f absolutely_integrable_on A  $\longleftrightarrow$  ( $\forall b \in \text{Basis. } (\lambda x. f x \cdot b)$  absolutely_integrable_on
A)

```

proof –

```

have *: ( $\lambda x. \text{norm } (f x)$ ) integrable_on A  $\longleftrightarrow$  ( $\forall b \in \text{Basis. } (\lambda x. \text{norm } (f x \cdot b))$ 
integrable_on A) (is ?lhs = ?rhs)

```

```

if f integrable_on A

```

proof

```

assume ?lhs

```

```

then show ?rhs

```

```

by (metis absolutely_integrable_on_def Topology_Euclidean_Space.norm_nth_le
absolutely_integrable_integrable_bound integrable_component that)

```

next

```

assume R: ?rhs

```

```

have f absolutely_integrable_on A

```

```

proof (rule absolutely_integrable_integrable_bound)

```

```

show ( $\lambda x. \sum i \in \text{Basis. norm } (f x \cdot i)$ ) integrable_on A

```

```

using R by (force intro: integrable_sum)

```

```

qed (use that norm_le_l1 in auto)

```

```

then show ?lhs

```

```

using absolutely_integrable_on_def by auto

```

qed

```

show ?thesis

```

```

unfolding absolutely_integrable_on_def

```

```

by (simp add: integrable_componentwise_iff [symmetric] ball_conj_distrib *
cong: conj_cong)

```

qed

lemma *absolutely_integrable_componentwise*:

```

shows ( $\bigwedge b. b \in \text{Basis} \implies (\lambda x. f x \cdot b)$  absolutely_integrable_on A)  $\implies$  f
absolutely_integrable_on A

```

```

using absolutely_integrable_componentwise_iff by blast

```

lemma *absolutely_integrable_component*:

```

f absolutely_integrable_on A  $\implies$  ( $\lambda x. f x \cdot (b :: 'b :: \text{euclidean\_space})$ ) absolutely_integrable_on A

```

```

by (drule absolutely_integrable_linear[OF _ bounded_linear_inner_left[of b]])
(simp add: o_def)

```

lemma *absolutely_integrable_scaleR_left*:

```

fixes f :: 'n::euclidean_space  $\Rightarrow$  'm::euclidean_space

```

```

assumes f absolutely_integrable_on S

```

```

  shows  $(\lambda x. c *_{\mathbb{R}} f x)$  absolutely_integrable_on S
proof -
  have  $(\lambda x. c *_{\mathbb{R}} x)$  o f absolutely_integrable_on S
  by (simp add: absolutely_integrable_linear assms bounded_linear_scaleR_right)
  then show ?thesis
  using assms by blast
qed

```

```

lemma absolutely_integrable_scaleR_right:
  assumes f absolutely_integrable_on S
  shows  $(\lambda x. f x *_{\mathbb{R}} c)$  absolutely_integrable_on S
  using assms by blast

```

```

lemma absolutely_integrable_norm:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  assumes f absolutely_integrable_on S
  shows (norm o f) absolutely_integrable_on S
  using assms by (simp add: absolutely_integrable_on_def o_def)

```

```

lemma absolutely_integrable_abs:
  fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
  assumes f absolutely_integrable_on S
  shows  $(\lambda x. \sum i \in \text{Basis}. |f x \cdot i| *_{\mathbb{R}} i)$  absolutely_integrable_on S
  (is ?g absolutely_integrable_on S)

```

```

proof -
  have *:  $(\lambda y. \sum j \in \text{Basis}. \text{if } j = i \text{ then } y *_{\mathbb{R}} j \text{ else } 0) \circ$ 
     $(\lambda x. \text{norm} (\sum j \in \text{Basis}. \text{if } j = i \text{ then } (x \cdot i) *_{\mathbb{R}} j \text{ else } 0)) \circ f$ 
    absolutely_integrable_on S
  if  $i \in \text{Basis}$  for i
proof -
  have bounded_linear  $(\lambda y. \sum j \in \text{Basis}. \text{if } j = i \text{ then } y *_{\mathbb{R}} j \text{ else } 0)$ 
  by (simp add: linear_linear algebra_simps linearI)
  moreover have  $(\lambda x. \text{norm} (\sum j \in \text{Basis}. \text{if } j = i \text{ then } (x \cdot i) *_{\mathbb{R}} j \text{ else } 0)) \circ f$ 
    absolutely_integrable_on S
  using assms  $\langle i \in \text{Basis} \rangle$ 
  unfolding o_def
  by (intro absolutely_integrable_norm [unfolded o_def])
  (auto simp: algebra_simps dest: absolutely_integrable_component)
  ultimately show ?thesis
  by (subst comp_assoc) (blast intro: absolutely_integrable_linear)
qed
  have eq: ?g =
     $(\lambda x. \sum i \in \text{Basis}. ((\lambda y. \sum j \in \text{Basis}. \text{if } j = i \text{ then } y *_{\mathbb{R}} j \text{ else } 0) \circ$ 
       $(\lambda x. \text{norm} (\sum j \in \text{Basis}. \text{if } j = i \text{ then } (x \cdot i) *_{\mathbb{R}} j \text{ else } 0)) \circ f) x)$ 
  by (simp)
  show ?thesis
  unfolding eq
  by (rule absolutely_integrable_sum) (force simp: intro!: *)+
qed

```

lemma *abs_absolutely_integrableI_1*:
fixes $f :: 'a :: euclidean_space \Rightarrow real$
assumes $f: f \text{ integrable_on } A$ **and** $(\lambda x. |f x|) \text{ integrable_on } A$
shows $f \text{ absolutely_integrable_on } A$
by (rule *absolutely_integrable_integrable_bound* [*OF _ assms*]) *auto*

lemma *abs_absolutely_integrableI*:
assumes $f: f \text{ integrable_on } S$ **and** $fcomp: (\lambda x. \sum_{i \in Basis} |f x \cdot i| *_R i) \text{ integrable_on } S$
shows $f \text{ absolutely_integrable_on } S$
proof –
have $(\lambda x. (f x \cdot i) *_R i) \text{ absolutely_integrable_on } S$ **if** $i \in Basis$ **for** i
proof –
have $(\lambda x. |f x \cdot i|) \text{ integrable_on } S$
using *assms integrable_component* [*OF fcomp, where y=i*] **that by simp**
then have $(\lambda x. f x \cdot i) \text{ absolutely_integrable_on } S$
using *abs_absolutely_integrableI_1* $f \text{ integrable_component}$ **by blast**
then show *?thesis*
by (rule *absolutely_integrable_scaleR_right*)
qed
then have $(\lambda x. \sum_{i \in Basis} (f x \cdot i) *_R i) \text{ absolutely_integrable_on } S$
by (*simp add: absolutely_integrable_sum*)
then show *?thesis*
by (*simp add: euclidean_representation*)
qed

lemma *absolutely_integrable_abs_iff*:
 $f \text{ absolutely_integrable_on } S \longleftrightarrow$
 $f \text{ integrable_on } S \wedge (\lambda x. \sum_{i \in Basis} |f x \cdot i| *_R i) \text{ integrable_on } S$
(is ?lhs = ?rhs)
proof
assume *?lhs* **then show** *?rhs*
using *absolutely_integrable_abs absolutely_integrable_on_def* **by blast**
next
assume *?rhs*
moreover
have $(\lambda x. \text{if } x \in S \text{ then } \sum_{i \in Basis} |f x \cdot i| *_R i \text{ else } 0) = (\lambda x. \sum_{i \in Basis} |(\text{if } x \in S \text{ then } f x \text{ else } 0) \cdot i| *_R i)$
by force
ultimately show *?lhs*
by (*simp only: absolutely_integrable_restrict_UNIV* [*of S, symmetric*] *integrable_restrict_UNIV* [*of S, symmetric*] *abs_absolutely_integrableI*)
qed

lemma *absolutely_integrable_max*:
fixes $f :: 'n::euclidean_space \Rightarrow 'm::euclidean_space$


```

assumes  $f$  absolutely_integrable_on  $S$   $g$  absolutely_integrable_on  $S$ 
shows  $(\lambda x. \sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i)$ 
  absolutely_integrable_on  $S$ 
proof -
  have  $(\lambda x. \sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i) =$ 
     $(\lambda x. (1/2) *_{\mathbb{R}} (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_{\mathbb{R}} i)))$ 
  proof (rule ext)
    fix  $x$ 
    have  $(\sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i) = (\sum_{i \in \text{Basis}} ((f x \cdot i + g x \cdot$ 
 $i + |f x \cdot i - g x \cdot i|) / 2) *_{\mathbb{R}} i)$ 
    by (force intro: sum.cong)
    also have  $\dots = (1 / 2) *_{\mathbb{R}} (\sum_{i \in \text{Basis}} (f x \cdot i + g x \cdot i + |f x \cdot i - g x \cdot i|$ 
 $*_{\mathbb{R}} i)$ 
    by (simp add: scaleR_right.sum)
    also have  $\dots = (1 / 2) *_{\mathbb{R}} (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_{\mathbb{R}} i))$ 
    by (simp add: sum.distrib algebra_simps euclidean_representation)
    finally
    show  $(\sum_{i \in \text{Basis}} \max (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i) =$ 
       $(1 / 2) *_{\mathbb{R}} (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_{\mathbb{R}} i)) .$ 
  qed
  moreover have  $(\lambda x. (1 / 2) *_{\mathbb{R}} (f x + g x + (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_{\mathbb{R}}$ 
 $i)))$ 
    absolutely_integrable_on  $S$ 
    using absolutely_integrable_abs [OF set_integral_diff(1)] [OF assms]
    by (intro set_integral_add absolutely_integrable_scaleR_left assms) (simp add:
algebra_simps)
  ultimately show ?thesis by metis
qed

```

corollary *absolutely_integrable_max_1*:

```

fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow \text{real}$ 
assumes  $f$  absolutely_integrable_on  $S$   $g$  absolutely_integrable_on  $S$ 
shows  $(\lambda x. \max (f x) (g x))$  absolutely_integrable_on  $S$ 
using absolutely_integrable_max [OF assms] by simp

```

lemma *absolutely_integrable_min*:

```

fixes  $f :: 'n::\text{euclidean\_space} \Rightarrow 'm::\text{euclidean\_space}$ 
assumes  $f$  absolutely_integrable_on  $S$   $g$  absolutely_integrable_on  $S$ 
shows  $(\lambda x. \sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i)$ 
  absolutely_integrable_on  $S$ 

```

proof -

```

have  $(\lambda x. \sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i) =$ 
   $(\lambda x. (1/2) *_{\mathbb{R}} (f x + g x - (\sum_{i \in \text{Basis}} |f x \cdot i - g x \cdot i| *_{\mathbb{R}} i)))$ 

```

proof (*rule ext*)

fix x

```

have  $(\sum_{i \in \text{Basis}} \min (f x \cdot i) (g x \cdot i) *_{\mathbb{R}} i) = (\sum_{i \in \text{Basis}} ((f x \cdot i + g x \cdot$ 
 $i - |f x \cdot i - g x \cdot i|) / 2) *_{\mathbb{R}} i)$ 

```

by (*force intro: sum.cong*)

```

also have  $\dots = (1 / 2) *_{\mathbb{R}} (\sum_{i \in \text{Basis}} (f x \cdot i + g x \cdot i - |f x \cdot i - g x \cdot i|)$ 

```

2982

$*_R i$)
by (simp add: scaleR_right.sum)
also have ... = $(1 / 2) *_R (f x + g x - (\sum i \in \text{Basis}. |f x \cdot i - g x \cdot i| *_R i))$
by (simp add: sum.distrib sum_subtractf algebra_simps euclidean_representation)
finally
show $(\sum i \in \text{Basis}. \min (f x \cdot i) (g x \cdot i) *_R i) =$
 $(1 / 2) *_R (f x + g x - (\sum i \in \text{Basis}. |f x \cdot i - g x \cdot i| *_R i))$.
qed
moreover have $(\lambda x. (1 / 2) *_R (f x + g x - (\sum i \in \text{Basis}. |f x \cdot i - g x \cdot i| *_R i)))$
 $\text{absolutely_integrable_on } S$
using absolutely_integrable_abs [OF set_integral_diff(1) [OF assms]]
by (intro set_integral_add set_integral_diff absolutely_integrable_scaleR_left
assms)
(simp add: algebra_simps)
ultimately show ?thesis by metis
qed

corollary absolutely_integrable_min_1:
fixes $f :: 'n :: \text{euclidean_space} \Rightarrow \text{real}$
assumes $f \text{ absolutely_integrable_on } S$ $g \text{ absolutely_integrable_on } S$
shows $(\lambda x. \min (f x) (g x)) \text{ absolutely_integrable_on } S$
using absolutely_integrable_min [OF assms] by simp

lemma nonnegative_absolutely_integrable:
fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$
assumes $f \text{ integrable_on } A$ and comp: $\bigwedge x b. \llbracket x \in A; b \in \text{Basis} \rrbracket \implies 0 \leq f x \cdot b$
shows $f \text{ absolutely_integrable_on } A$
proof -
have $(\lambda x. (f x \cdot i) *_R i) \text{ absolutely_integrable_on } A$ if $i \in \text{Basis}$ for i
proof -
have $(\lambda x. f x \cdot i) \text{ integrable_on } A$
by (simp add: assms(1) integrable_component)
then have $(\lambda x. f x \cdot i) \text{ absolutely_integrable_on } A$
by (metis that comp nonnegative_absolutely_integrable_1)
then show ?thesis
by (rule absolutely_integrable_scaleR_right)
qed
then have $(\lambda x. \sum i \in \text{Basis}. (f x \cdot i) *_R i) \text{ absolutely_integrable_on } A$
by (simp add: absolutely_integrable_sum)
then show ?thesis
by (simp add: euclidean_representation)
qed

lemma absolutely_integrable_component_ubound:
fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$
assumes $f: f \text{ integrable_on } A$ and $g: g \text{ absolutely_integrable_on } A$
and comp: $\bigwedge x b. \llbracket x \in A; b \in \text{Basis} \rrbracket \implies f x \cdot b \leq g x \cdot b$
shows $f \text{ absolutely_integrable_on } A$

proof –

have $(\lambda x. g\ x - (g\ x - f\ x))$ *absolutely_integrable_on* A

proof (*rule set_integral_diff* [*OF* g *nonnegative_absolutely_integrable*])

show $(\lambda x. g\ x - f\ x)$ *integrable_on* A

using *Henstock_Kurzweil_Integration.integrable_diff_absolutely_integrable_on_def*

f g **by** *blast*

qed (*simp add: comp_inner_diff_left*)

then show *?thesis*

by *simp*

qed

lemma *absolutely_integrable_component_lbound*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: euclidean_space$

assumes f : f *absolutely_integrable_on* A **and** g : g *integrable_on* A

and *comp*: $\bigwedge x\ b. \llbracket x \in A; b \in Basis \rrbracket \Longrightarrow f\ x \cdot b \leq g\ x \cdot b$

shows g *absolutely_integrable_on* A

proof –

have $(\lambda x. f\ x + (g\ x - f\ x))$ *absolutely_integrable_on* A

proof (*rule set_integral_add* [*OF* f *nonnegative_absolutely_integrable*])

show $(\lambda x. g\ x - f\ x)$ *integrable_on* A

using *Henstock_Kurzweil_Integration.integrable_diff_absolutely_integrable_on_def*

f g **by** *blast*

qed (*simp add: comp_inner_diff_left*)

then show *?thesis*

by *simp*

qed

lemma *integrable_on_1_iff*:

fixes $f :: 'a :: euclidean_space \Rightarrow real^1$

shows f *integrable_on* $S \longleftrightarrow (\lambda x. f\ x\ \$\ 1)$ *integrable_on* S

by (*auto simp: integrable_componentwise_iff* [*of f*] *Basis_vec_def cart_eq_inner_axis*)

lemma *integral_on_1_eq*:

fixes $f :: 'a :: euclidean_space \Rightarrow real^1$

shows *integral* $S\ f = vec$ (*integral* S $(\lambda x. f\ x\ \$\ 1)$)

by (*cases f integrable_on S*) (*simp_all add: integrable_on_1_iff vec_eq_iff not_integrable_integral*)

lemma *absolutely_integrable_on_1_iff*:

fixes $f :: 'a :: euclidean_space \Rightarrow real^1$

shows f *absolutely_integrable_on* $S \longleftrightarrow (\lambda x. f\ x\ \$\ 1)$ *absolutely_integrable_on*

S

unfolding *absolutely_integrable_on_def*

by (*auto simp: integrable_on_1_iff norm_real*)

lemma *absolutely_integrable_absolutely_integrable_lbound*:

fixes $f :: 'm :: euclidean_space \Rightarrow real$

assumes f : f *integrable_on* S **and** g : g *absolutely_integrable_on* S

and $*$: $\bigwedge x. x \in S \Longrightarrow g\ x \leq f\ x$

shows f *absolutely_integrable_on* S

by (rule absolutely_integrable_component_lbound [OF g f]) (simp add: *)

lemma absolutely_integrable_absolutely_integrable_ubound:

fixes $f :: 'm::\text{euclidean_space} \Rightarrow \text{real}$

assumes $fg: f \text{ integrable_on } S \ g \text{ absolutely_integrable_on } S$

and $*$: $\bigwedge x. x \in S \implies f\ x \leq g\ x$

shows $f \text{ absolutely_integrable_on } S$

by (rule absolutely_integrable_component_ubound [OF fg]) (simp add: *)

lemma has_integral_vec1_I_cbox:

fixes $f :: \text{real}^1 \Rightarrow 'a::\text{real_normed_vector}$

assumes $(f \text{ has_integral } y) \text{ (cbox } a\ b)$

shows $((f \circ \text{vec}) \text{ has_integral } y) \{a\$1..b\$1\}$

proof –

have $((\lambda x. f(\text{vec } x)) \text{ has_integral } (1 / 1) *_{\mathbb{R}} y) ((\lambda x. x \$ 1) \text{ ' cbox } a\ b)$

proof (rule has_integral_twiddle)

show $\exists w\ z::\text{real}^1. \text{vec ' cbox } u\ v = \text{cbox } w\ z$

content $(\text{vec ' cbox } u\ v :: (\text{real}^1) \text{ set}) = 1 * \text{content } (\text{cbox } u\ v)$ **for** $u\ v$

unfolding vec_cbox_1_eq

by (auto simp: content_cbox_if_cart_interval_eq_empty_cart)

show $\exists w\ z. (\lambda x. x \$ 1) \text{ ' cbox } u\ v = \text{cbox } w\ z$ **for** $u\ v :: \text{real}^1$

using vec_nth_cbox_1_eq **by** blast

qed (auto simp: continuous_vec assms)

then show ?thesis

by (simp add: o_def)

qed

lemma has_integral_vec1_I:

fixes $f :: \text{real}^1 \Rightarrow 'a::\text{real_normed_vector}$

assumes $(f \text{ has_integral } y) \ S$

shows $(f \circ \text{vec} \text{ has_integral } y) ((\lambda x. x \$ 1) \text{ ' } S)$

proof –

have $*$: $\exists z. ((\lambda x. \text{if } x \in (\lambda x. x \$ 1) \text{ ' } S \text{ then } (f \circ \text{vec})\ x \text{ else } 0) \text{ has_integral } z)$
 $\{a..b\} \wedge \text{norm } (z - y) < e$

if $\text{int}: \bigwedge a\ b. \text{ball } 0\ B \subseteq \text{cbox } a\ b \implies$

$(\exists z. ((\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0) \text{ has_integral } z) \text{ (cbox } a\ b) \wedge$
 $\text{norm } (z - y) < e)$

and $B: \text{ball } 0\ B \subseteq \{a..b\}$ **for** $e\ B\ a\ b$

proof –

have [simp]: $(\exists y \in S. x = y \$ 1) \iff \text{vec } x \in S$ **for** x

by force

have B' : $\text{ball } (0::\text{real}^1) B \subseteq \text{cbox } (\text{vec } a) (\text{vec } b)$

using B **by** (simp add: Basis_vec_def cart_eq_inner_axis [symmetric]
 $\text{mem_box_norm_real_subset_iff})$

show ?thesis

using int [OF B'] **by** (auto simp: image_iff o_def cong: if_cong dest!:
 $\text{has_integral_vec1_I_cbox})$

qed

show ?thesis

```

using assms
apply (subst has_integral_alt)
apply (subst (asm) has_integral_alt)
apply (simp add: has_integral_vec1_I_cbox split: if_split_asm)
subgoal by (metis vector_one_nth)
subgoal
  apply (erule all_forward imp_forward ex_forward asm_rl)+
  by (blast intro!: *)
done
qed

lemma has_integral_vec1_nth_cbox:
  fixes f :: real ⇒ 'a::real_normed_vector
  assumes (f has_integral y) {a..b}
  shows (( $\lambda x::\text{real}^1. f(x\$1)$ ) has_integral y) (cbox (vec a) (vec b))
proof -
  have (( $\lambda x::\text{real}^1. f(x\$1)$ ) has_integral ( $1 / 1$ ) *R y) (vec ' cbox a b)
  proof (rule has_integral_twiddle)
    show  $\exists w z::\text{real}. (\lambda x. x \$ 1) ' cbox u v = cbox w z$ 
       $\text{content } ((\lambda x. x \$ 1) ' cbox u v) = 1 * \text{content } (cbox u v)$  for u v::real^1
    unfolding vec_cbox_1_eq by (auto simp: content_cbox_if_cart_interval_eq_empty_cart)
    show  $\exists w z::\text{real}^1. \text{vec ' cbox } u v = cbox w z$  for u v :: real
      using vec_cbox_1_eq by auto
  qed (auto simp: continuous_vec assms)
  then show ?thesis
    using vec_cbox_1_eq by auto
qed

lemma has_integral_vec1_D_cbox:
  fixes f :: real^1 ⇒ 'a::real_normed_vector
  assumes ((f  $\circ$  vec) has_integral y) {a$1..b$1}
  shows (f has_integral y) (cbox a b)
  by (metis (mono_tags, lifting) assms comp_apply has_integral_eq has_integral_vec1_nth_cbox
vector_one_nth)

lemma has_integral_vec1_D:
  fixes f :: real^1 ⇒ 'a::real_normed_vector
  assumes ((f  $\circ$  vec) has_integral y) (( $\lambda x. x \$ 1$ ) ' S)
  shows (f has_integral y) S
proof -
  have *:  $\exists z. ((\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ has\_integral } z) (cbox a b) \wedge \text{norm } (z - y) < e$ 
  if int:  $\bigwedge a b. \text{ball } 0 B \subseteq \{a..b\} \implies$ 
     $(\exists z. ((\lambda x. \text{if } x \in (\lambda x. x \$ 1) ' S \text{ then } (f \circ \text{vec}) x \text{ else } 0)$ 
has_integral z) {a..b}  $\wedge \text{norm } (z - y) < e)$ 
  and B:  $\text{ball } 0 B \subseteq cbox a b$  for e B and a b::real^1
  proof -
  have B':  $\text{ball } 0 B \subseteq \{a\$1..b\$1\}$ 
  proof (clarsimp)

```

```

fix t
assume |t| < B then show a $ 1 ≤ t ∧ t ≤ b $ 1
  using subsetD [OF B]
    by (metis (mono_tags, opaque_lifting) mem_ball_0 mem_box_cart(2)
norm_real vec_component)
qed
have eq: (λx. if vec x ∈ S then f (vec x) else 0) = (λx. if x ∈ S then f x else
0) ∘ vec
  by force
have [simp]: (∃ y ∈ S. x = y $ 1) ↔ vec x ∈ S for x
  by force
show ?thesis
using int [OF B'] by (auto simp: image_iff eq cong: if_cong dest!: has_integral_vec1_D_cbox)
qed
show ?thesis
  using assms
  apply (subst has_integral_alt)
  apply (subst (asm) has_integral_alt)
  apply (simp add: has_integral_vec1_D_cbox eq_cbox split: if_split_asm, blast)
  apply (intro conjI impI)
  subgoal by (metis vector_one_nth)
  apply (erule thin_rl)
  apply (erule all_forward ex_forward conj_forward)+
  by (blast intro!: *)+
qed

```

lemma *integral_vec1_eq*:

```

fixes f :: real1 ⇒ 'a::real_normed_vector
shows integral S f = integral ((λx. x $ 1) ' S) (f ∘ vec)
using has_integral_vec1_I [of f] has_integral_vec1_D [of f]
by (metis has_integral_iff not_integrable_integral)

```

lemma *absolutely_integrable_drop*:

```

fixes f :: real1 ⇒ 'b::euclidean_space
shows f absolutely_integrable_on S ↔ (f ∘ vec) absolutely_integrable_on (λx.
x $ 1) ' S

```

```

unfolding absolutely_integrable_on_def integrable_on_def

```

proof safe

```

fix y r
assume (f has_integral y) S ((λx. norm (f x)) has_integral r) S
then show ∃ y. (f ∘ vec has_integral y) ((λx. x $ 1) ' S)
  ∃ y. ((λx. norm ((f ∘ vec) x)) has_integral y) ((λx. x $ 1) ' S)
  by (force simp: o_def dest!: has_integral_vec1_I)+

```

next

```

fix y :: 'b and r :: real
assume (f ∘ vec has_integral y) ((λx. x $ 1) ' S)
  ((λx. norm ((f ∘ vec) x)) has_integral r) ((λx. x $ 1) ' S)
then show ∃ y. (f has_integral y) S ∃ y. ((λx. norm (f x)) has_integral y) S

```

by (force simp: o_def intro: has_integral_vec1_D)+
qed

9.8.19 Dominated convergence

lemma *dominated_convergence*:

fixes $f :: \text{nat} \Rightarrow 'n::\text{euclidean_space} \Rightarrow 'm::\text{euclidean_space}$
assumes $f: \bigwedge k. (f\ k) \text{ integrable_on } S$ **and** $h: h \text{ integrable_on } S$
and $le: \bigwedge k\ x. x \in S \implies \text{norm } (f\ k\ x) \leq h\ x$
and $conv: \bigwedge x. x \in S \implies (\lambda k. f\ k\ x) \longrightarrow g\ x$
shows $g \text{ integrable_on } S \implies (\lambda k. \text{integral } S\ (f\ k)) \longrightarrow \text{integral } S\ g$
proof –
have $3: h \text{ absolutely_integrable_on } S$
unfolding *absolutely_integrable_on_def*
proof
show $(\lambda x. \text{norm } (h\ x)) \text{ integrable_on } S$
proof (*intro integrable_spike_finite[OF _ _ h, of {}] ballI*)
fix $x \text{ assume } x \in S - \{\}$ **then show** $\text{norm } (h\ x) = h\ x$
by (*metis Diff_empty_abs_of_nonneg bot_set_def le_norm_ge_zero order_trans real_norm_def*)
qed auto
qed fact
have $2: \text{set_borel_measurable } \text{lebesgue } S\ (f\ k) \text{ for } k$
unfolding *set_borel_measurable_def*
using f **by** (*auto intro: has_integral_implies_lebesgue_measurable simp: integrable_on_def*)
then have $1: \text{set_borel_measurable } \text{lebesgue } S\ g$
unfolding *set_borel_measurable_def*
by (*rule borel_measurable_LIMSEQ_metric*) (*use conv in <auto split: split_indicator>*)
have $4: \text{AE } x \text{ in } \text{lebesgue}. (\lambda i. \text{indicator } S\ x\ *_{\mathbb{R}}\ f\ i\ x) \longrightarrow \text{indicator } S\ x\ *_{\mathbb{R}}\ g\ x$
AE } $x \text{ in } \text{lebesgue}. \text{norm } (\text{indicator } S\ x\ *_{\mathbb{R}}\ f\ k\ x) \leq \text{indicator } S\ x\ *_{\mathbb{R}}\ h\ x \text{ for } k$
using $conv\ le$ **by** (*auto intro!: always_eventually split: split_indicator*)
have $g \text{ absolutely_integrable_on } S$
using $1\ 2\ 3\ 4$ **unfolding** *set_borel_measurable_def set_integrable_def*
by (*rule integrable_dominated_convergence*)
then show $g \text{ integrable_on } S$
by (*auto simp: absolutely_integrable_on_def*)
have $(\lambda k. (\text{LINT } x:S|\text{lebesgue}. f\ k\ x)) \longrightarrow (\text{LINT } x:S|\text{lebesgue}. g\ x)$
unfolding *set_borel_measurable_def set_lebesgue_integral_def*
using $1\ 2\ 3\ 4$ **unfolding** *set_borel_measurable_def set_lebesgue_integral_def set_integrable_def*
by (*rule integral_dominated_convergence*)
then show $(\lambda k. \text{integral } S\ (f\ k)) \longrightarrow \text{integral } S\ g$
using $g \text{ absolutely_integrable_integrable_bound}[OF\ le\ f\ h]$
by (*subst (asm) (1 2) set_lebesgue_integral_eq_integral*) *auto*
qed

lemma *has_integral_dominated_convergence*:

```

fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
assumes  $\bigwedge k. (f\ k\ \text{has\_integral}\ y\ k)\ S\ h\ \text{integrable\_on}\ S$ 
 $\bigwedge k. \forall x \in S. \text{norm}\ (f\ k\ x) \leq h\ x\ \forall x \in S. (\lambda k. f\ k\ x) \longrightarrow g\ x$ 
and x: y  $\longrightarrow$  x
shows (g has_integral x) S
proof -
have int_f:  $\bigwedge k. (f\ k)\ \text{integrable\_on}\ S$ 
using assms by (auto simp: integrable_on_def)
have (g has_integral (integral S g)) S
by (metis assms(2-4) dominated_convergence(1) has_integral_integral int_f)
moreover have integral S g = x
proof (rule LIMSEQ_unique)
show ( $\lambda i. \text{integral}\ S\ (f\ i)$ )  $\longrightarrow$  x
using integral_unique[OF assms(1)] x by simp
show ( $\lambda i. \text{integral}\ S\ (f\ i)$ )  $\longrightarrow$  integral S g
by (metis assms(2) assms(3) assms(4) dominated_convergence(2) int_f)
qed
ultimately show ?thesis
by simp
qed

```

```

lemma dominated_convergence_integrable_1:
fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  real
assumes f:  $\bigwedge k. f\ k\ \text{absolutely\_integrable\_on}\ S$ 
and h: h integrable_on S
and normg:  $\bigwedge x. x \in S \implies \text{norm}(g\ x) \leq (h\ x)$ 
and lim:  $\bigwedge x. x \in S \implies (\lambda k. f\ k\ x) \longrightarrow g\ x$ 
shows g integrable_on S
proof -
have habs: h absolutely_integrable_on S
using h normg nonnegative_absolutely_integrable_1 norm_ge_zero order_trans
by blast
let ?f =  $\lambda n\ x. (\min (\max (- h\ x) (f\ n\ x)) (h\ x))$ 
have h0: h x  $\geq 0$  if x  $\in$  S for x
using normg that by force
have leh: norm (?f k x)  $\leq h\ x$  if x  $\in$  S for k x
using h0 that by force
have limf: ( $\lambda k. ?f\ k\ x$ )  $\longrightarrow$  g x if x  $\in$  S for x
proof -
have  $\bigwedge e\ y. |f\ y\ x - g\ x| < e \implies |\min (\max (- h\ x) (f\ y\ x)) (h\ x) - g\ x| < e$ 
using h0 [OF that] normg [OF that] by simp
then show ?thesis
using lim [OF that] by (auto simp add: tendsto_iff dist_norm elim!: eventually_mono)
qed
show ?thesis
proof (rule dominated_convergence [of ?f S h g])
have ( $\lambda x. - h\ x$ ) absolutely_integrable_on S
using habs unfolding set_integrable_def by auto

```



```

then show ?f k integrable_on S for k
  by (intro set_lebesgue_integral_eq_integral absolutely_integrable_min_1 ab-
solutely_integrable_max_1 f habs)
  qed (use assms leh limf in auto)
qed

```

lemma dominated_convergence_integrable:

```

fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
assumes f:  $\bigwedge k. f k$  absolutely_integrable_on S
  and h: h integrable_on S
  and normg:  $\bigwedge x. x \in S \implies \text{norm}(g x) \leq (h x)$ 
  and lim:  $\bigwedge x. x \in S \implies (\lambda k. f k x) \longrightarrow g x$ 
shows g integrable_on S
using f
unfolding integrable_componentwise_iff [of g] absolutely_integrable_componentwise_iff
[where f = f k for k]
proof clarify
  fix b :: 'm
  assume fb [rule_format]:  $\bigwedge k. \forall b \in \text{Basis}. (\lambda x. f k x \cdot b)$  absolutely_integrable_on
S and b: b  $\in$  Basis
  show  $(\lambda x. g x \cdot b)$  integrable_on S
  proof (rule dominated_convergence_integrable_1 [OF fb h])
    fix x
    assume x  $\in$  S
    show norm (g x  $\cdot$  b)  $\leq$  h x
      using norm_nth_le  $\langle x \in S \rangle$  b normg order.trans by blast
    show  $(\lambda k. f k x \cdot b) \longrightarrow g x \cdot b$ 
      using  $\langle x \in S \rangle$  b lim tendsto_componentwise_iff by fastforce
  qed (use b in auto)
qed

```

lemma dominated_convergence_absolutely_integrable:

```

fixes f :: nat  $\Rightarrow$  'n::euclidean_space  $\Rightarrow$  'm::euclidean_space
assumes f:  $\bigwedge k. f k$  absolutely_integrable_on S
  and h: h integrable_on S
  and normg:  $\bigwedge x. x \in S \implies \text{norm}(g x) \leq (h x)$ 
  and lim:  $\bigwedge x. x \in S \implies (\lambda k. f k x) \longrightarrow g x$ 
shows g absolutely_integrable_on S
proof -
  have g integrable_on S
    by (rule dominated_convergence_integrable [OF assms])
  with assms show ?thesis
    by (blast intro: absolutely_integrable_integrable_bound [where g=h])
qed

```

proposition integral_countable_UN:

```

fixes f :: realm  $\Rightarrow$  realn
assumes f: f absolutely_integrable_on  $(\bigcup (\text{range } s))$ 

```

```

    and s:  $\bigwedge m. s\ m \in \text{sets lebesgue}$ 
  shows  $\bigwedge n. f \text{ absolutely\_integrable\_on } (\bigcup m \leq n. s\ m)$ 
    and  $(\lambda n. \text{integral } (\bigcup m \leq n. s\ m)\ f) \longrightarrow \text{integral } (\bigcup (s\ ' \text{ UNIV}))\ f \text{ (is ?F}$ 
 $\longrightarrow ?I)$ 
  proof -
  show  $fU: f \text{ absolutely\_integrable\_on } (\bigcup m \leq n. s\ m)$  for n
    using assms by (blast intro: set_integrable_subset [OF f])
  have fint:  $f \text{ integrable\_on } (\bigcup (\text{range } s))$ 
    using absolutely_integrable_on_def f by blast
  let ?h =  $\lambda x. \text{if } x \in \bigcup (s\ ' \text{ UNIV}) \text{ then norm}(f\ x) \text{ else } 0$ 
  have  $(\lambda n. \text{integral UNIV } (\lambda x. \text{if } x \in (\bigcup m \leq n. s\ m) \text{ then } f\ x \text{ else } 0))$ 
 $\longrightarrow \text{integral UNIV } (\lambda x. \text{if } x \in \bigcup (s\ ' \text{ UNIV}) \text{ then } f\ x \text{ else } 0)$ 
  proof (rule dominated_convergence)
  show  $(\lambda x. \text{if } x \in (\bigcup m \leq n. s\ m) \text{ then } f\ x \text{ else } 0) \text{ integrable\_on UNIV}$  for n
    unfolding integrable_restrict_UNIV
    using fU absolutely_integrable_on_def by blast
  show  $(\lambda x. \text{if } x \in \bigcup (s\ ' \text{ UNIV}) \text{ then norm}(f\ x) \text{ else } 0) \text{ integrable\_on UNIV}$ 
    by (metis (no_types) absolutely_integrable_on_def integrable_restrict_UNIV)
  show  $\bigwedge x. (\lambda n. \text{if } x \in (\bigcup m \leq n. s\ m) \text{ then } f\ x \text{ else } 0)$ 
 $\longrightarrow (\text{if } x \in \bigcup (s\ ' \text{ UNIV}) \text{ then } f\ x \text{ else } 0)$ 
    by (force intro: tendsto_eventually_eventually_sequentiallyI)
  qed auto
  then show ?F  $\longrightarrow ?I$ 
    by (simp only: integral_restrict_UNIV)
  qed

```

9.8.20 Fundamental Theorem of Calculus for the Lebesgue integral

For the positive integral we replace continuity with Borel-measurability.

lemma

```

  fixes f :: real  $\Rightarrow$  real
  assumes [measurable]:  $f \in \text{borel\_measurable borel}$ 
  assumes f:  $\bigwedge x. x \in \{a..b\} \implies \text{DERIV } F\ x \text{ :> } f\ x \wedge x \in \{a..b\} \implies 0 \leq f\ x$ 
  and a < b
  shows nn_integral_FTC_Icc:  $(\int^{+x} \text{ennreal } (f\ x) * \text{indicator } \{a..b\}\ x\ \partial \text{lborel})$ 
 $= F\ b - F\ a \text{ (is ?nn)}$ 
    and has_bochner_integral_FTC_Icc_nonneg:
      has_bochner_integral lborel  $(\lambda x. f\ x * \text{indicator } \{a..b\}\ x) (F\ b - F\ a) \text{ (is ?has)}$ 
    and integral_FTC_Icc_nonneg:  $(\int x. f\ x * \text{indicator } \{a..b\}\ x\ \partial \text{lborel}) = F\ b - F\ a \text{ (is ?eq)}$ 
    and integrable_FTC_Icc_nonneg:  $\text{integrable lborel } (\lambda x. f\ x * \text{indicator } \{a..b\}\ x) \text{ (is ?int)}$ 
  proof -
  have *:  $(\lambda x. f\ x * \text{indicator } \{a..b\}\ x) \in \text{borel\_measurable borel} \wedge x. 0 \leq f\ x * \text{indicator } \{a..b\}\ x$ 
    using f(2) by (auto split: split_indicator)

```

have $F_mono: a \leq x \implies x \leq y \implies y \leq b \implies F x \leq F y$ **for** $x y$
using f **by** $(intro\ DERIV_nonneg_imp_nondecreasing[of\ x\ y\ F])$ $(auto\ intro: order_trans)$

have $(f\ has_integral\ F\ b - F\ a)\ \{a..b\}$
by $(intro\ fundamental_theorem_of_calculus$
 $(auto\ simp: has_real_derivative_iff_has_vector_derivative[symmetric]$
 $intro: has_field_derivative_subset[OF\ f(1)]\ \langle a \leq b \rangle)$
then have $i: ((\lambda x. f\ x * indicator\ \{a..b\}\ x)\ has_integral\ F\ b - F\ a)$ $UNIV$
unfolding $indicator_def\ of_bool_def\ if_distrib$ **where** $f = \lambda x. a * x$ **for** a
by $(simp\ cong\ del: if_weak_cong\ del: atLeastAtMost_iff)$
then have $nn: (\int^+ x. f\ x * indicator\ \{a..b\}\ x\ \partial lborel) = F\ b - F\ a$
by $(rule\ nn_integral_has_integral_lborel[OF\ *])$
then show $?has$
by $(rule\ has_bochner_integral_nn_integral[rotated\ 3])$ $(simp_all\ add: * F_mono$
 $\langle a \leq b \rangle)$
then show $?eq\ ?int$
unfolding $has_bochner_integral_iff$ **by** $auto$
show $?nn$
by $(subst\ nn[symmetric])$
 $(auto\ intro!: nn_integral_cong\ simp\ add: ennreal_mult\ f\ split: split_indicator)$
qed

lemma

fixes $f :: real \Rightarrow 'a :: euclidean_space$
assumes $a \leq b$
assumes $\bigwedge x. a \leq x \implies x \leq b \implies (F\ has_vector_derivative\ f\ x)$ $(at\ x\ within\ \{a..b\})$
assumes $cont: continuous_on\ \{a..b\}\ f$
shows $has_bochner_integral_FTC_Icc:$
 $has_bochner_integral\ lborel\ (\lambda x. indicator\ \{a..b\}\ x *_{\mathbb{R}} f\ x)\ (F\ b - F\ a)$ **(is**
 $?has)$
and $integral_FTC_Icc: (\int x. indicator\ \{a..b\}\ x *_{\mathbb{R}} f\ x\ \partial lborel) = F\ b - F\ a$
(is $?eq)$
proof $-$
let $?f = \lambda x. indicator\ \{a..b\}\ x *_{\mathbb{R}} f\ x$
have $int: integrable\ lborel\ ?f$
using $borel_integrable_compact[OF\ _\ cont]$ **by** $auto$
have $(f\ has_integral\ F\ b - F\ a)\ \{a..b\}$
using $assms(1,2)$ **by** $(intro\ fundamental_theorem_of_calculus)\ auto$
moreover
have $(f\ has_integral\ integral^L\ lborel\ ?f)\ \{a..b\}$
using $has_integral_integral\ lborel[OF\ int]$
unfolding $indicator_def\ of_bool_def\ if_distrib$ **where** $f = \lambda x. x *_{\mathbb{R}} a$ **for** a
by $(simp\ cong\ del: if_weak_cong\ del: atLeastAtMost_iff)$
ultimately show $?eq$
by $(auto\ dest: has_integral_unique)$
then show $?has$
using int **by** $(auto\ simp: has_bochner_integral_iff)$

qed

lemma

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $a \leq b$ 
assumes  $\text{deriv}: \bigwedge x. a \leq x \implies x \leq b \implies \text{DERIV } F x :> f x$ 
assumes  $\text{cont}: \bigwedge x. a \leq x \implies x \leq b \implies \text{isCont } f x$ 
shows  $\text{has\_bochner\_integral\_FTC\_Icc\_real}$ :
   $\text{has\_bochner\_integral } \text{lborel } (\lambda x. f x * \text{indicator } \{a .. b\} x) (F b - F a)$  (is
 $\text{?has}$ )
  and  $\text{integral\_FTC\_Icc\_real}$ :  $(\int x. f x * \text{indicator } \{a .. b\} x \partial \text{lborel}) = F b -$ 
 $F a$  (is ?eq)
proof -
  have  $1: \bigwedge x. a \leq x \implies x \leq b \implies (F \text{ has\_vector\_derivative } f x)$  (at  $x$  within  $\{a$ 
 $.. b\}$ )
    unfolding  $\text{has\_real\_derivative\_iff\_has\_vector\_derivative}$ [ $\text{symmetric}$ ]
    using  $\text{deriv}$  by  $(\text{auto intro: DERIV\_subset})$ 
  have  $2: \text{continuous\_on } \{a .. b\} f$ 
    using  $\text{cont}$  by  $(\text{intro continuous\_at\_imp\_continuous\_on})$   $\text{auto}$ 
  show  $\text{?has ?eq}$ 
    using  $\text{has\_bochner\_integral\_FTC\_Icc}$ [ $\text{OF } \langle a \leq b \rangle 1 2$ ]  $\text{integral\_FTC\_Icc}$ [ $\text{OF } \langle a \leq b \rangle 1 2$ ]
    by  $(\text{auto simp: mult.commute})$ 
qed

```

lemma $\text{nn_integral_FTC_atLeast}$:

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $f\_borel: f \in \text{borel\_measurable borel}$ 
assumes  $f: \bigwedge x. a \leq x \implies \text{DERIV } F x :> f x$ 
assumes  $\text{nonneg}: \bigwedge x. a \leq x \implies 0 \leq f x$ 
assumes  $\text{lim}: (F \longrightarrow T)$   $\text{at\_top}$ 
shows  $(\int^{+x}. \text{ennreal } (f x) * \text{indicator } \{a ..\} x \partial \text{lborel}) = T - F a$ 
proof -
  let  $\text{?f} = \lambda(i::\text{nat}) (x::\text{real}). \text{ennreal } (f x) * \text{indicator } \{a..a + \text{real } i\} x$ 
  let  $\text{?fR} = \lambda x. \text{ennreal } (f x) * \text{indicator } \{a ..\} x$ 

  have  $F\_mono: a \leq x \implies x \leq y \implies F x \leq F y$  for  $x y$ 
    using  $f \text{ nonneg}$  by  $(\text{intro DERIV\_nonneg\_imp\_nondecreasing}$ [ $\text{of } x y F$ ])  $(\text{auto$ 
 $\text{intro: order\_trans})$ 
  then have  $F\_le\_T: a \leq x \implies F x \leq T$  for  $x$ 
    by  $(\text{intro tendsto\_lowerbound}$ [ $\text{OF } \text{lim}$ ])
     $(\text{auto simp: eventually\_at\_top\_linorder})$ 

  have  $(\text{SUP } i. \text{?f } i x) = \text{?fR } x$  for  $x$ 
proof  $(\text{rule LIMSEQ\_unique}$ [ $\text{OF LIMSEQ\_SUP}$ ])
  obtain  $n$  where  $x - a < \text{real } n$ 
    using  $\text{reals\_Archimedean2}$ [ $\text{of } x - a$ ]  $..$ 
  then have  $\text{eventually } (\lambda n. \text{?f } n x = \text{?fR } x)$   $\text{sequentially}$ 
    by  $(\text{auto intro!: eventually\_sequentiallyI}$ [where  $c=n$ ]  $\text{split: split\_indicator})$ 

```

```

    then show  $(\lambda n. ?f n x) \longrightarrow ?fR x$ 
      by (rule tendsto_eventually)
  qed (auto simp: nonneg incseq_def le_fun_def split: split_indicator)
  then have  $\text{integral}^N \text{lborel } ?fR = (\int^+ x. (\text{SUP } i. ?f i x) \partial \text{lborel})$ 
    by simp
  also have  $\dots = (\text{SUP } i. (\int^+ x. ?f i x \partial \text{lborel}))$ 
  proof (rule nn_integral_monotone_convergence_SUP)
    show incseq ?f
      using nonneg by (auto simp: incseq_def le_fun_def split: split_indicator)
    show  $\bigwedge i. (?f i) \in \text{borel\_measurable lborel}$ 
      using f_borel by auto
  qed
  also have  $\dots = (\text{SUP } i. \text{ennreal } (F (a + \text{real } i) - F a))$ 
    by (subst nn_integral_FTC_Icc[OF f_borel f_nonneg]) auto
  also have  $\dots = T - F a$ 
  proof (rule LIMSEQ_unique[OF LIMSEQ_SUP])
    have  $(\lambda x. F (a + \text{real } x)) \longrightarrow T$ 
      by (auto intro: filterlim_compose[OF lim_filterlim_tendsto_add_at_top]
        filterlim_real_sequentially)
    then show  $(\lambda n. \text{ennreal } (F (a + \text{real } n) - F a)) \longrightarrow \text{ennreal } (T - F a)$ 
      by (simp add: F_mono F_le_T tendsto_diff)
  qed (auto simp: incseq_def intro!: ennreal_le_iff[THEN iffD2] F_mono)
  finally show ?thesis .
qed

```

lemma *integral_power*:

```

 $a \leq b \implies (\int x. x^k * \text{indicator } \{a..b\} x \partial \text{lborel}) = (b^{\text{Suc } k} - a^{\text{Suc } k}) / \text{Suc } k$ 
proof (subst integral_FTC_Icc_real)
  fix x show  $\text{DERIV } (\lambda x. x^{\text{Suc } k} / \text{Suc } k) x \text{ :> } x^k$ 
    by (intro derivative_eq_intros) auto
qed (auto simp: field_simps simp del: of_nat_Suc)

```

9.8.21 Integration by parts

lemma *integral_by_parts_integrable*:

```

fixes f g F G::real  $\Rightarrow$  real
assumes  $a \leq b$ 
assumes  $\text{cont\_f}[intro]: !!x. a \leq x \implies x \leq b \implies \text{isCont } f x$ 
assumes  $\text{cont\_g}[intro]: !!x. a \leq x \implies x \leq b \implies \text{isCont } g x$ 
assumes  $[intro]: !!x. \text{DERIV } F x \text{ :> } f x$ 
assumes  $[intro]: !!x. \text{DERIV } G x \text{ :> } g x$ 
shows  $\text{integrable lborel } (\lambda x. (F x) * (g x) + (f x) * (G x)) * \text{indicator } \{a .. b\} x)$ 
by (auto intro!: borel_integrable_atLeastAtMost continuous_intros) (auto intro!:
  DERIV_isCont)

```

lemma *integral_by_parts*:

```

fixes f g F G::real  $\Rightarrow$  real
assumes  $[arith]: a \leq b$ 

```

```

assumes cont_f[intro]: !!x.  $a \leq x \implies x \leq b \implies \text{isCont } f \ x$ 
assumes cont_g[intro]: !!x.  $a \leq x \implies x \leq b \implies \text{isCont } g \ x$ 
assumes [intro]: !!x.  $\text{DERIV } F \ x \ :> f \ x$ 
assumes [intro]: !!x.  $\text{DERIV } G \ x \ :> g \ x$ 
shows  $(\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel})$ 
  =  $F \ b * G \ b - F \ a * G \ a - \int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ 
proof -
  have  $(\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel})$ 
    =  $(\text{LBINT } x. F \ x * g \ x * \text{indicat\_real } \{a..b\} \ x + \int x. f \ x * G \ x * \text{indicat\_real } \{a..b\} \ x)$ 
  by (meson vector_space_over_itself.scale_left_distrib)
  also have ... =  $(\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}) + \int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}$ 
  proof (intro Bochner_Integration.integral_add borel_integrable_atLeastAtMost cont_f cont_g continuous_intros)
    show  $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } F \ x \ \wedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } G \ x$ 
      using DERIV_isCont by blast+
    qed
  finally have  $(\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}) =$ 
     $(\int x. (F \ x * g \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}) + \int x. (f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel} .$ 
  moreover have  $(\int x. (F \ x * g \ x + f \ x * G \ x) * \text{indicator } \{a .. b\} \ x \ \partial \text{lborel}) =$ 
     $F \ b * G \ b - F \ a * G \ a$ 
  proof (intro integral_FTC_Icc_real derivative_eq_intros cont_f cont_g continuous_intros)
    show  $\bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } F \ x \ \wedge x. \llbracket a \leq x; x \leq b \rrbracket \implies \text{isCont } G \ x$ 
      using DERIV_isCont by blast+
    qed auto
  ultimately show ?thesis by auto
qed

```

lemma *integral_by_parts'*:

fixes *f g F G*::*real* \Rightarrow *real*

assumes $a \leq b$

assumes !!*x*. $a \leq x \implies x \leq b \implies \text{isCont } f \ x$

assumes !!*x*. $a \leq x \implies x \leq b \implies \text{isCont } g \ x$

assumes !!*x*. $\text{DERIV } F \ x \ :> f \ x$

assumes !!*x*. $\text{DERIV } G \ x \ :> g \ x$

shows $(\int x. \text{indicator } \{a .. b\} \ x *_{\mathbb{R}} (F \ x * g \ x) \ \partial \text{lborel})$

$$= F \ b * G \ b - F \ a * G \ a - \int x. \text{indicator } \{a .. b\} \ x *_{\mathbb{R}} (f \ x * G \ x)$$

∂lborel

using *integral_by_parts[OF assms]* **by** (*simp add: ac_simps*)

lemma *has_bochner_integral_even_function*:

fixes *f* :: *real* \Rightarrow '*a* :: {*banach*, *second_countable_topology*}

assumes *f*: *has_bochner_integral lborel* $(\lambda x. \text{indicator } \{0..\} \ x *_{\mathbb{R}} f \ x) \ x$

assumes *even*: $\bigwedge x. f \ (- \ x) = f \ x$

shows *has_bochner_integral lborel* $f \ (2 *_{\mathbb{R}} \ x)$

proof -

```

have indicator:  $\bigwedge x::\text{real}. \text{indicator } \{..0\} (-x) = \text{indicator } \{0..\} x$ 
  by (auto split: split_indicator)
have has_bochner_integral lborel ( $\lambda x. \text{indicator } \{..0\} x *_R f x$ )  $x$ 
  by (subst lborel_has_bochner_integral_real_affine_iff[where  $c=-1$  and  $t=0$ ])
    (auto simp: indicator_even f)
with  $f$  have has_bochner_integral lborel ( $\lambda x. \text{indicator } \{0..\} x *_R f x + \text{indicator}$ 
 $\{..0\} x *_R f x$ ) ( $x + x$ )
  by (rule has_bochner_integral_add)
then have has_bochner_integral lborel  $f (x + x)$ 
  by (rule has_bochner_integral_discrete_difference[where  $X=\{0\}$ , THEN iffD1,
rotated 4])
    (auto split: split_indicator)
then show ?thesis
  by (simp add: scaleR_2)
qed

```

lemma has_bochner_integral_odd_function:

```

fixes  $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{second\_countable\_topology}\}$ 
assumes  $f$ : has_bochner_integral lborel ( $\lambda x. \text{indicator } \{0..\} x *_R f x$ )  $x$ 
assumes odd:  $\bigwedge x. f (-x) = -f x$ 
shows has_bochner_integral lborel  $f 0$ 
proof -
have indicator:  $\bigwedge x::\text{real}. \text{indicator } \{..0\} (-x) = \text{indicator } \{0..\} x$ 
  by (auto split: split_indicator)
have has_bochner_integral lborel ( $\lambda x. - \text{indicator } \{..0\} x *_R f x$ )  $x$ 
  by (subst lborel_has_bochner_integral_real_affine_iff[where  $c=-1$  and  $t=0$ ])
    (auto simp: indicator_odd f)
from has_bochner_integral_minus[OF this]
have has_bochner_integral lborel ( $\lambda x. \text{indicator } \{..0\} x *_R f x$ ) ( $-x$ )
  by simp
with  $f$  have has_bochner_integral lborel ( $\lambda x. \text{indicator } \{0..\} x *_R f x + \text{indicator}$ 
 $\{..0\} x *_R f x$ ) ( $x + -x$ )
  by (rule has_bochner_integral_add)
then have has_bochner_integral lborel  $f (x + -x)$ 
  by (rule has_bochner_integral_discrete_difference[where  $X=\{0\}$ , THEN iffD1,
rotated 4])
    (auto split: split_indicator)
then show ?thesis
  by simp
qed

```

9.8.22 A non-negative continuous function whose integral is zero must be zero

lemma has_integral_0_closure_imp_0:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
assumes  $f$ : continuous_on (closure  $S$ )  $f$ 
  and nonneg_interior:  $\bigwedge x. x \in S \implies 0 \leq f x$ 
  and pos:  $0 < \text{emeasure lborel } S$ 

```

```

    and finite: emeasure lborel S < ∞
    and regular: emeasure lborel (closure S) = emeasure lborel S
    and opn: open S
  assumes int: (f has_integral 0) (closure S)
  assumes x: x ∈ closure S
  shows f x = 0
proof -
  have zero: emeasure lborel (frontier S) = 0
    using finite closure_subset regular
  unfolding frontier_def
  by (subst emeasure_Diff) (auto simp: frontier_def interior_open ‹open S›)
  have nonneg: 0 ≤ f x if x ∈ closure S for x
    using continuous_ge_on_closure[OF f that nonneg_interior] by simp
  have 0 = integral (closure S) f
    by (blast intro: int sym)
  also
  note intl = has_integral_integrable[OF int]
  have af: f absolutely_integrable_on (closure S)
    using nonneg
  by (intro absolutely_integrable_onI intl integrable_eq[OF intl]) simp
  then have integral (closure S) f = set_lebesgue_integral lebesgue (closure S) f
    by (intro set_lebesgue_integral_eq_integral(2)[symmetric])
  also have ... = 0 ↔ (AE x in lebesgue. indicator (closure S) x *R f x = 0)
    unfolding set_lebesgue_integral_def
  proof (rule integral_nonneg_eq_0_iff_AE)
    show integrable lebesgue (λx. indicat_real (closure S) x *R f x)
      by (metis af set_integrable_def)
  qed (use nonneg in ‹auto simp: indicator_def›)
  also have ... ↔ (AE x in lebesgue. x ∈ {x. x ∈ closure S → f x = 0})
    by (auto simp: indicator_def)
  finally have (AE x in lebesgue. x ∈ {x. x ∈ closure S → f x = 0}) by simp
  moreover have (AE x in lebesgue. x ∈ - frontier S)
    using zero
  by (auto simp: eventually_ae_filter null_sets_def intro!: exI[where x=frontier S])
  ultimately have ae: AE x ∈ S in lebesgue. x ∈ {x ∈ closure S. f x = 0} (is
    ?th)
    by eventually_elim (use closure_subset in ‹auto simp: ›)
  have closed {0::real} by simp
  with continuous_on_closed_vimage[OF closed_closure, of S f] f
  have closed (f -‘ {0} ∩ closure S) by blast
  then have closed {x ∈ closure S. f x = 0} by (auto simp: vimage_def Int_def
    conj_commute)
  with ‹open S› have x ∈ {x ∈ closure S. f x = 0} if x ∈ S for x using ae that
    by (rule mem_closed_if_AE_lebesgue_open)
  then have f x = 0 if x ∈ S for x using that by auto
  from continuous_constant_on_closure[OF f this ‹x ∈ closure S›]
  show f x = 0 .
qed

```



```

lemma has_integral_0_cbox_imp_0:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes continuous_on (cbox a b) f and  $\bigwedge x. x \in \text{box } a \ b \implies 0 \leq f \ x$ 
  assumes (f has_integral 0) (cbox a b)
  assumes ne: box a b  $\neq$  {}
  assumes x: x  $\in$  cbox a b
  shows f x = 0
proof -
  have 0 < emeasure lborel (box a b)
    using ne x unfolding emeasure_lborel_box_eq
  by (force intro!: prod_pos simp: mem_box algebra_simps)
  then show ?thesis using assms
  by (intro has_integral_0_closure_imp_0 [of box a b f x])
    (auto simp: emeasure_lborel_box_eq emeasure_lborel_cbox_eq algebra_simps
  mem_box)
qed

```

```

corollary integral_cbox_eq_0_iff:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes continuous_on (cbox a b) f and box a b  $\neq$  {}
  and  $\bigwedge x. x \in \text{cbox } a \ b \implies f \ x \geq 0$ 
  shows integral (cbox a b) f = 0  $\iff$  ( $\forall x \in \text{cbox } a \ b. f \ x = 0$ ) (is ?lhs = ?rhs)
proof
  assume int0: ?lhs
  show ?rhs
    using has_integral_0_cbox_imp_0 [of a b f] assms
  by (metis box_subset_cbox eq_integralD int0 integrable_continuous subsetD)
next
  assume ?rhs then show ?lhs
  by (meson has_integral_is_0_cbox integral_unique)
qed

```

```

lemma integral_eq_0_iff:
  fixes f :: real  $\Rightarrow$  real
  assumes continuous_on {a..b} f and a < b
  and  $\bigwedge x. x \in \{a..b\} \implies f \ x \geq 0$ 
  shows integral {a..b} f = 0  $\iff$  ( $\forall x \in \{a..b\}. f \ x = 0$ )
  using integral_cbox_eq_0_iff [of a b f] assms by simp

```

```

lemma integralL_eq_0_iff:
  fixes f :: real  $\Rightarrow$  real
  assumes contf: continuous_on {a..b} f and a < b
  and  $\bigwedge x. x \in \{a..b\} \implies f \ x \geq 0$ 
  shows integralL (lebesgue_on {a..b}) f = 0  $\iff$  ( $\forall x \in \{a..b\}. f \ x = 0$ )
  using integral_eq_0_iff [OF assms]
  by (simp add: contf continuous_imp_integrable_real lebesgue_integral_eq_integral)

```

In fact, strict inequality is required only at a single point within the box.

lemma *integral_less*:

fixes $f :: 'n::euclidean_space \Rightarrow real$
assumes *cont*: $continuous_on (cbox\ a\ b)\ f\ continuous_on (cbox\ a\ b)\ g$ **and** $box\ a\ b \neq \{\}$
and *fg*: $\bigwedge x. x \in box\ a\ b \implies f\ x < g\ x$
shows $integral\ (cbox\ a\ b)\ f < integral\ (cbox\ a\ b)\ g$
proof –
obtain *int*: $f\ integrable_on\ (cbox\ a\ b)\ g\ integrable_on\ (cbox\ a\ b)$
using *cont integrable_continuous* **by** *blast*
then have $integral\ (cbox\ a\ b)\ f \leq integral\ (cbox\ a\ b)\ g$
by (*metis fg integrable_on_open_interval integral_le integral_open_interval less_eq_real_def*)
moreover have $integral\ (cbox\ a\ b)\ f \neq integral\ (cbox\ a\ b)\ g$
proof (*rule ccontr*)
assume $\neg\ integral\ (cbox\ a\ b)\ f \neq integral\ (cbox\ a\ b)\ g$
then have $0: ((\lambda x. g\ x - f\ x)\ has_integral\ 0)\ (cbox\ a\ b)$
by (*metis (full_types) cancel_comm_monoid_add_class.diff_cancel has_integral_diff int_integrable_integral*)
have *cgf*: $continuous_on\ (cbox\ a\ b)\ (\lambda x. g\ x - f\ x)$
using *cont continuous_on_diff* **by** *blast*
show *False*
using *has_integral_0_cbox_imp_0* [*OF cgf_0*] *assms(3) box_subset_cbox fg less_eq_real_def* **by** *fastforce*
qed
ultimately show *?thesis*
by *linarith*
qed

lemma *integral_less_real*:

fixes $f :: real \Rightarrow real$
assumes *continuous_on* $\{a..b\}\ f\ continuous_on\ \{a..b\}\ g$ **and** $\{a <..<b\} \neq \{\}$
and $\bigwedge x. x \in \{a <..<b\} \implies f\ x < g\ x$
shows $integral\ \{a..b\}\ f < integral\ \{a..b\}\ g$
by (*metis assms box_real integral_less*)

9.8.23 Various common equivalent forms of function measurability

lemma *indicator_sum_eq*:

fixes $m::real$ **and** $f :: 'a \Rightarrow real$
assumes $|m| \leq 2^{\wedge}(2*n)\ m/2^{\wedge}n \leq f\ x\ f\ x < (m+1)/2^{\wedge}n\ m \in \mathbb{Z}$
shows $(\sum k::real \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). k/2^{\wedge}n * indicator\ \{y. k/2^{\wedge}n \leq f\ y \wedge f\ y < (k+1)/2^{\wedge}n\}\ x) = m/2^{\wedge}n$
(is sum ?f ?S = _)

proof –

have $sum\ ?f\ ?S = sum\ (\lambda k. k/2^{\wedge}n * indicator\ \{y. k/2^{\wedge}n \leq f\ y \wedge f\ y < (k+1)/2^{\wedge}n\}\ x)\ \{m\}$
proof (*rule comm_monoid_add_class.sum_mono_neutral_right*)
show *finite ?S*

```

    by (rule finite_abs_int_segment)
  show {m}  $\subseteq$  {k  $\in$   $\mathbb{Z}$ . |k|  $\leq$   $2^{(2*n)}$ }
    using assms by auto
  show  $\forall i \in \{k \in \mathbb{Z}. |k| \leq 2^{(2*n)}\} - \{m\}. ?f i = 0$ 
    using assms by (auto simp: indicator_def Ints_def abs_le_iff field_split_simps)
  qed
  also have ... =  $m/2^n$ 
    using assms by (auto simp: indicator_def not_less)
  finally show ?thesis .
  qed

```

lemma measurable_on_sf_limit_lemma1:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes  $\bigwedge a b. \{x \in S. a \leq f x \wedge f x < b\} \in \text{sets (lebesgue\_on } S)$ 
  obtains g where  $\bigwedge n. g n \in \text{borel\_measurable (lebesgue\_on } S)$ 
     $\bigwedge n. \text{finite (range (g n))}$ 
     $\bigwedge x. (\lambda n. g n x) \longrightarrow f x$ 
  proof
    show  $(\lambda x. \text{sum } (\lambda k::\text{real}. k/2^n * \text{indicator } \{y. k/2^n \leq f y \wedge f y < (k+1)/2^n\} x)$ 
       $\{k \in \mathbb{Z}. |k| \leq 2^{(2*n)}\}) \in \text{borel\_measurable (lebesgue\_on } S)$ 
      (is ?g  $\in$  _) for n
    proof -
      have  $\bigwedge k. \llbracket k \in \mathbb{Z}; |k| \leq 2^{(2*n)} \rrbracket$ 
         $\implies \text{Measurable.pred (lebesgue\_on } S) (\lambda x. k / (2^n) \leq f x \wedge f x < (k+1) / (2^n))$ 
        using assms by (force simp: pred_def space_restrict_space)
      then show ?thesis
        by (simp add: field_class.field_divide_inverse)
    qed
    show finite (range (?g n)) for n
    proof -
      have range (?g n)  $\subseteq$   $(\lambda k. k/2^n) \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{(2*n)}\}$ 
      proof clarify
        fix x
        show ?g n x  $\in$   $(\lambda k. k/2^n) \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{(2*n)}\}$ 
        proof (cases  $\exists k::\text{real}. k \in \mathbb{Z} \wedge |k| \leq 2^{(2*n)} \wedge k/2^n \leq (f x) \wedge (f x) < (k+1)/2^n$ )
          case True
            then show ?thesis
              apply clarify
              by (subst indicator_sum_eq) auto
          next
            case False
              then have ?g n x = 0 by auto
              then show ?thesis by force
        qed
      qed
    next
      case False
        then have ?g n x = 0 by auto
        then show ?thesis by force
    qed
  qed
  moreover have finite  $((\lambda k::\text{real}. (k/2^n)) \text{ ' } \{k \in \mathbb{Z}. |k| \leq 2^{(2*n)}\})$ 

```

```

    by (simp add: finite_abs_int_segment)
  ultimately show ?thesis
    using finite_subset by blast
qed
show (λn. ?g n x) → f x for x
proof (rule LIMSEQ_I)
  fix e::real
  assume e > 0
  obtain N1 where N1: |f x| < 2 ^ N1
    using real_arch_pow by fastforce
  obtain N2 where N2: (1/2) ^ N2 < e
    using real_arch_pow_inv ‹e > 0› by force
  have norm (?g n x - f x) < e if n: n ≥ max N1 N2 for n
  proof -
    define m where m ≡ floor(2 ^ n * (f x))
    have 1 ≤ |2 ^ n| * e
      using n N2 ‹e > 0› less_eq_real_def less_le_trans by (fastforce simp add:
field_split_simps)
    then have *: [|x| ≤ y; y < x + 1] ⇒ abs(x - y) < |2 ^ n| * e for x y::real
      by linarith
    have |2 ^ n| * |m/2 ^ n - f x| = |2 ^ n * (m/2 ^ n - f x)|
      by (simp add: abs_mult)
    also have ... = |real_of_int [2 ^ n * f x] - f x * 2 ^ n|
      by (simp add: algebra_simps m_def)
    also have ... < |2 ^ n| * e
      by (rule *; simp add: mult_commute)
    finally have |2 ^ n| * |m/2 ^ n - f x| < |2 ^ n| * e .
    then have me: |m/2 ^ n - f x| < e
      by simp
    have |real_of_int m| ≤ 2 ^ (2*n)
    proof (cases f x < 0)
    case True
      then have -|f x| ≤ [(2::real) ^ N1]
        using N1 le_floor_iff minus_le_iff by fastforce
      with n True have |real_of_int [f x]| ≤ 2 ^ N1
        by linarith
      also have ... ≤ 2 ^ n
        using n by (simp add: m_def)
      finally have |real_of_int [f x]| * 2 ^ n ≤ 2 ^ n * 2 ^ n
        by simp
    moreover
    have |real_of_int [2 ^ n * f x]| ≤ |real_of_int [f x]| * 2 ^ n
    proof -
      have |real_of_int [2 ^ n * f x]| = - (real_of_int [2 ^ n * f x])
        using True by (simp add: abs_if_mult_less_0_iff)
      also have ... ≤ - (real_of_int [(2::real) ^ n] * [f x])
        using le_mult_floor_Ints [of (2::real) ^ n] by simp
      also have ... ≤ |real_of_int [f x]| * 2 ^ n
        using True

```

```

      by simp
      finally show ?thesis .
    qed
    ultimately show ?thesis
    by (metis (no_types, opaque_lifting) m_def order_trans power2_eq_square
power_even_eq)
  next
  case False
  with n N1 have f x ≤ 2n
    by (simp add: not_less) (meson less_eq_real_def one_le_numeral or-
der_trans power_increasing)
  moreover have 0 ≤ m
    using False m_def by force
  ultimately show ?thesis
  by (metis abs_of_nonneg floor_mono le_floor_iff m_def of_int_0_le_iff
power2_eq_square power_mult mult_le_cancel_right_pos zero_less_numeral mult commute
zero_less_power)
  qed
  then have ?g n x = m/2n
    by (rule indicator_sum_eq) (auto simp add: m_def field_split_simps,
linarith)
  then have norm (?g n x - f x) = norm (m/2n - f x)
    by simp
  also have ... < e
    by (simp add: me)
  finally show ?thesis .
  qed
  then show ∃ no. ∀ n ≥ no. norm (?g n x - f x) < e
    by blast
  qed
qed

```

lemma *borel_measurable_simple_function_limit:*

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

shows $f \in \text{borel_measurable}(\text{lebesgue_on } S) \iff$

$(\exists g. (\forall n. (g n) \in \text{borel_measurable}(\text{lebesgue_on } S)) \wedge$
 $(\forall n. \text{finite}(\text{range}(g n))) \wedge (\forall x. (\lambda n. g n x) \longrightarrow f x))$

proof –

have $\exists g. (\forall n. (g n) \in \text{borel_measurable}(\text{lebesgue_on } S)) \wedge$
 $(\forall n. \text{finite}(\text{range}(g n))) \wedge (\forall x. (\lambda n. g n x) \longrightarrow f x)$

if $f: \bigwedge a i. i \in \text{Basis} \implies \{x \in S. f x \cdot i < a\} \in \text{sets}(\text{lebesgue_on } S)$

proof –

have $\exists g. (\forall n. (g n) \in \text{borel_measurable}(\text{lebesgue_on } S)) \wedge$
 $(\forall n. \text{finite}(\text{image}(g n) \text{ UNIV})) \wedge$
 $(\forall x. ((\lambda n. g n x) \longrightarrow f x \cdot i) \text{ if } i \in \text{Basis for } i$

proof (*rule measurable_on_sf_limit_lemma1 [of S λx. f x · i]*)

show $\{x \in S. a \leq f x \cdot i \wedge f x \cdot i < b\} \in \text{sets}(\text{lebesgue_on } S)$ **for** $a b$

proof –

```

      have {x ∈ S. a ≤ f x · i ∧ f x · i < b} = {x ∈ S. f x · i < b} - {x ∈ S. a
> f x · i}
      by auto
      also have ... ∈ sets (lebesgue_on S)
      using f that by blast
      finally show ?thesis .
    qed
  qed blast
  then obtain g where g:
    ∧ i n. i ∈ Basis ⇒ g i n ∈ borel_measurable (lebesgue_on S)
    ∧ i n. i ∈ Basis ⇒ finite(range (g i n))
    ∧ i x. i ∈ Basis ⇒ ((λn. g i n x) → f x · i)
  by metis
  show ?thesis
  proof (intro conjI allI exI)
    show (λx. ∑ i ∈ Basis. g i n x *R i) ∈ borel_measurable (lebesgue_on S) for
n
    by (intro borel_measurable_sum borel_measurable_scaleR) (auto intro: g)
    show finite (range (λx. ∑ i ∈ Basis. g i n x *R i)) for n
    proof -
      have range (λx. ∑ i ∈ Basis. g i n x *R i) ⊆ (λh. ∑ i ∈ Basis. h i *R i) ‘ PiE
Basis (λi. range (g i n))
      proof clarify
        fix x
        show (∑ i ∈ Basis. g i n x *R i) ∈ (λh. ∑ i ∈ Basis. h i *R i) ‘ (ΠE i ∈ Basis.
range (g i n))
        by (rule_tac x=λi ∈ Basis. g i n x in image_eqI) auto
      qed
      moreover have finite(PiE Basis (λi. range (g i n)))
      by (simp add: g finite_PiE)
      ultimately show ?thesis
      by (metis (mono_tags, lifting) finite_surj)
    qed
  show (λn. ∑ i ∈ Basis. g i n x *R i) → f x for x
  proof -
    have (λn. ∑ i ∈ Basis. g i n x *R i) → (∑ i ∈ Basis. (f x · i) *R i)
    by (auto intro!: tendsto_sum tendsto_scaleR g)
    moreover have (∑ i ∈ Basis. (f x · i) *R i) = f x
    using euclidean_representation by blast
    ultimately show ?thesis
    by metis
  qed
  qed
  moreover have f ∈ borel_measurable (lebesgue_on S)
  if meas_g: ∧ n. g n ∈ borel_measurable (lebesgue_on S)
  and fin: ∧ n. finite (range (g n))
  and to_f: ∧ x. (λn. g n x) → f x for g
  by (rule borel_measurable_LIMSEQ_metric [OF meas_g to_f])

```

ultimately show *?thesis*
 using *borel_measurable_vimage_halfspace_component_lt* by blast
 qed

9.8.24 Lebesgue sets and continuous images

proposition *lebesgue_regular_inner*:

assumes $S \in \text{sets lebesgue}$
 obtains $K \subseteq C$ where *negligible* $K \wedge n::\text{nat. compact}(C \ n) \ S = (\bigcup n. C \ n) \cup K$
 proof –
 have $\exists T. \text{closed } T \wedge T \subseteq S \wedge (S - T) \in \text{lmeasurable} \wedge \text{emeasure lebesgue } (S - T) < \text{ennreal } ((1/2)^n)$ for n
 using *sets_lebesgue_inner_closed assms*
 by (*metis sets_lebesgue_inner_closed zero_less_divide_1_iff zero_less_numeral zero_less_power*)
 then obtain C where *clo*: $\bigwedge n. \text{closed } (C \ n)$ and *subS*: $\bigwedge n. C \ n \subseteq S$
 and *mea*: $\bigwedge n. (S - C \ n) \in \text{lmeasurable}$
 and *less*: $\bigwedge n. \text{emeasure lebesgue } (S - C \ n) < \text{ennreal } ((1/2)^n)$
 by *metis*
 have $\exists F. (\forall n::\text{nat. compact}(F \ n)) \wedge (\bigcup n. F \ n) = C$ for $m::\text{nat}$
 by (*metis clo_closed_Union_compact_subsets*)
 then obtain $D :: [\text{nat}, \text{nat}] \Rightarrow 'a \text{ set}$ where $D: \bigwedge m \ n. \text{compact}(D \ m \ n) \wedge m. (\bigcup n. D \ m \ n) = C \ m$
 by *metis*
 let $?C = \text{from_nat_into } (\bigcup m. \text{range } (D \ m))$
 have *countable* $(\bigcup m. \text{range } (D \ m))$
 by *blast*
 have *range* $(\text{from_nat_into } (\bigcup m. \text{range } (D \ m))) = (\bigcup m. \text{range } (D \ m))$
 using *range_from_nat_into* by *simp*
 then have $CD: \exists m \ n. ?C \ k = D \ m \ n$ for k
 by (*metis (mono_tags, lifting) UN_iff rangeE_range_eqI*)
 show *thesis*
 proof
 show *negligible* $(S - (\bigcup n. C \ n))$
 proof (*clarsimp simp: negligible_outer_le*)
 fix $e :: \text{real}$
 assume $e > 0$
 then obtain n where $n: (1/2)^n < e$
 using *real_arch_pow_inv [of e 1/2]* by *auto*
 show $\exists T. S - (\bigcup n. C \ n) \subseteq T \wedge T \in \text{lmeasurable} \wedge \text{measure lebesgue } T \leq e$
 proof (*intro exI conjI*)
 show $S - (\bigcup n. C \ n) \subseteq S - C \ n$
 by *blast*
 show $S - C \ n \in \text{lmeasurable}$
 by (*simp add: mea*)
 show $\text{measure lebesgue } (S - C \ n) \leq e$
 using *less [of n] n*
 by (*simp add: emeasure_eq_measure2 less_le mea*)
 qed
 qed

```

qed
show compact (?C n) for n
  using CD D by metis
show  $S = (\bigcup n. ?C n) \cup (S - (\bigcup n. C n))$  (is _ = ?rhs)
proof
  show  $S \subseteq ?rhs$ 
  using D by fastforce
  show  $?rhs \subseteq S$ 
  using subS D CD by auto (metis Sup_upper range_eqI subsetCE)
qed
qed
qed

```

```

lemma sets_lebesgue_continuous_image:
  assumes T:  $T \in \text{sets lebesgue}$  and conf: continuous_on S f
  and negim:  $\bigwedge T. [\text{negligible } T; T \subseteq S] \implies \text{negligible}(f' T)$  and  $T \subseteq S$ 
  shows  $f' T \in \text{sets lebesgue}$ 
proof -
  obtain K C where negligible K and com:  $\bigwedge n::\text{nat}. \text{compact}(C n)$  and Teq:  $T = (\bigcup n. C n) \cup K$ 
  using lebesgue_regular_inner [OF T] by metis
  then have comf:  $\bigwedge n::\text{nat}. \text{compact}(f' C n)$ 
  by (metis Un_subset_iff Union_upper  $\langle T \subseteq S \rangle$  compact_continuous_image
  conf continuous_on_subset rangeI)
  have  $((\bigcup n. f' C n) \cup f' K) \in \text{sets lebesgue}$ 
  proof (rule sets.Un)
    have  $K \subseteq S$ 
    using Teq  $\langle T \subseteq S \rangle$  by blast
  show  $(\bigcup n. f' C n) \in \text{sets lebesgue}$ 
  proof (rule sets.countable_Union)
    show  $\text{range } (\lambda n. f' C n) \subseteq \text{sets lebesgue}$ 
    using borel_compact comf by (auto simp: borel_compact)
  qed auto
  show  $f' K \in \text{sets lebesgue}$ 
  by (simp add:  $\langle K \subseteq S \rangle \langle \text{negligible } K \rangle$  negim negligible_imp_sets)
qed
then show ?thesis
  by (simp add: Teq image_Un image_Union)
qed

```

```

lemma differentiable_image_in_sets_lebesgue:
  fixes f :: 'm::euclidean_space  $\Rightarrow$  'n::euclidean_space
  assumes S:  $S \in \text{sets lebesgue}$  and dim:  $\text{DIM}('m) \leq \text{DIM}('n)$  and f: f differentiable_on S
  shows  $f'S \in \text{sets lebesgue}$ 
proof (rule sets_lebesgue_continuous_image [OF S])
  show continuous_on S f
  by (meson differentiable_imp_continuous_on f)
  show  $\bigwedge T. [\text{negligible } T; T \subseteq S] \implies \text{negligible}(f' T)$ 

```



```

    using differentiable_on_subset f
    by (auto simp: intro!: negligible_differentiable_image_negligible [OF dim])
qed auto

lemma sets_lebesgue_on_continuous_image:
  assumes S: S ∈ sets lebesgue and X: X ∈ sets (lebesgue_on S) and contf:
    continuous_on S f
  and negim:  $\bigwedge T. \llbracket \text{negligible } T; T \subseteq S \rrbracket \implies \text{negligible}(f \text{ ` } T)$ 
  shows f ` X ∈ sets (lebesgue_on (f ` S))
proof -
  have X ⊆ S
  by (metis S X sets.Int_space_eq2 sets_restrict_space_iff)
  moreover have f ` S ∈ sets lebesgue
  using S contf negim sets_lebesgue_continuous_image by blast
  moreover have f ` X ∈ sets lebesgue
  by (metis S X contf negim sets_lebesgue_continuous_image sets_restrict_space_iff
    space_restrict_space space_restrict_space2)
  ultimately show ?thesis
  by (auto simp: sets_restrict_space_iff)
qed

lemma differentiable_image_in_sets_lebesgue_on:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes S: S ∈ sets lebesgue and X: X ∈ sets (lebesgue_on S) and dim:
    DIM('m) ≤ DIM('n)
  and f: f differentiable_on S
  shows f ` X ∈ sets (lebesgue_on (f ` S))
proof (rule sets_lebesgue_on_continuous_image [OF S X])
  show continuous_on S f
  by (meson differentiable_imp_continuous_on f)
  show  $\bigwedge T. \llbracket \text{negligible } T; T \subseteq S \rrbracket \implies \text{negligible}(f \text{ ` } T)$ 
  using differentiable_on_subset f
  by (auto simp: intro!: negligible_differentiable_image_negligible [OF dim])
qed

```

9.8.25 Affine lemmas

```

lemma borel_measurable_affine:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f: f ∈ borel_measurable lebesgue and c ≠ 0
  shows (λx. f(t + c *R x)) ∈ borel_measurable lebesgue
proof -
  { fix a b
  have {x. f x ∈ cbox a b} ∈ sets lebesgue
  using f cbox_borel lebesgue_measurable_vimage_borel by blast
  then have (λx. (x - t) /R c) ` {x. f x ∈ cbox a b} ∈ sets lebesgue
  proof (rule differentiable_image_in_sets_lebesgue)
    show (λx. (x - t) /R c) differentiable_on {x. f x ∈ cbox a b}
    unfolding differentiable_on_def differentiable_def

```

```

    by (rule ‹c ≠ 0› derivative_eq_intros strip exI | simp)+
  qed auto
  moreover
  have {x. f(t + c *R x) ∈ cbox a b} = (λx. (x-t) /R c) ‘ {x. f x ∈ cbox a b}
    using ‹c ≠ 0› by (auto simp: image_def)
  ultimately have {x. f(t + c *R x) ∈ cbox a b} ∈ sets lebesgue
    by (auto simp: borel_measurable_vimage_closed_interval) }
  then show ?thesis
  by (subst lebesgue_on_UNIV_eq [symmetric]; auto simp: borel_measurable_vimage_closed_interval)
qed

```

```

lemma lebesgue_integrable_real_affine:
  fixes f :: real ⇒ 'a :: euclidean_space
  assumes f: integrable lebesgue f and c ≠ 0
  shows integrable lebesgue (λx. f(t + c * x))
proof -
  have (λx. norm (f x)) ∈ borel_measurable lebesgue
    by (simp add: borel_measurable_integrable f)
  then show ?thesis
    using assms borel_measurable_affine [of f c]
    unfolding integrable_iff_bounded
    by (subst (asm) nn_integral_real_affine_lebesgue[where c=c and t=t]) (auto
  simp: ennreal_mult_less_top)
qed

```

```

lemma lebesgue_integrable_real_affine_iff:
  fixes f :: real ⇒ 'a :: euclidean_space
  shows c ≠ 0 ⇒ integrable lebesgue (λx. f(t + c * x)) ⟷ integrable lebesgue f
  using lebesgue_integrable_real_affine[of f c t]
    lebesgue_integrable_real_affine[of λx. f(t + c * x) 1/c -t/c]
  by (auto simp: field_simps)

```

```

lemma lebesgue_integral_real_affine:
  fixes f :: real ⇒ 'a :: euclidean_space and c :: real
  assumes c: c ≠ 0 shows (∫ x. f x ∂ lebesgue) = |c| *R (∫ x. f(t + c * x)
  ∂ lebesgue)
proof cases
  have (λx. t + c * x) ∈ lebesgue →M lebesgue
    using lebesgue_affine_measurable[where c= λx::real. c] ‹c ≠ 0› by simp
  moreover
  assume integrable lebesgue f
  ultimately show ?thesis
    by (subst lebesgue_real_affine[OF c, of t]) (auto simp: integral_density inte-
  gral_distr)
next
  assume ¬ integrable lebesgue f with c show ?thesis
    by (simp add: lebesgue_integrable_real_affine_iff not_integrable_integral_eq)
qed

```

lemma *has_bochner_integral_lebesgue_real_affine_iff*:
fixes $i :: 'a :: euclidean_space$
shows $c \neq 0 \implies$
 $has_bochner_integral\ lebesgue\ f\ i \longleftrightarrow$
 $has_bochner_integral\ lebesgue\ (\lambda x. f(t + c * x))\ (i /_R |c|)$
unfolding *has_bochner_integral_iff_lebesgue_integrable_real_affine_iff*
by (*simp_all add: lebesgue_integral_real_affine[symmetric] divideR_right cong: conj_cong*)

lemma *has_bochner_integral_reflect_real_lemma[intro]*:
fixes $f :: real \Rightarrow 'a :: euclidean_space$
assumes $has_bochner_integral\ (lebesgue_on\ \{a..b\})\ f\ i$
shows $has_bochner_integral\ (lebesgue_on\ \{-b..-a\})\ (\lambda x. f(-x))\ i$
proof -
have $eq: indicat_real\ \{a..b\}\ (-x) *_R f(-x) = indicat_real\ \{-b..-a\}\ x *_R f(-x)$
for x
by (*auto simp: indicator_def*)
have $i: has_bochner_integral\ lebesgue\ (\lambda x. indicator\ \{a..b\}\ x *_R f\ x)\ i$
using *assms* **by** (*auto simp: has_bochner_integral_restrict_space*)
then have $has_bochner_integral\ lebesgue\ (\lambda x. indicator\ \{-b..-a\}\ x *_R f(-x))\ i$
using *has_bochner_integral_lebesgue_real_affine_iff* [*of -1* ($\lambda x. indicator\ \{a..b\}\ x *_R f\ x$) $i\ 0$]
by (*auto simp: eq*)
then show *?thesis*
by (*auto simp: has_bochner_integral_restrict_space*)
qed

lemma *has_bochner_integral_reflect_real[simp]*:
fixes $f :: real \Rightarrow 'a :: euclidean_space$
shows $has_bochner_integral\ (lebesgue_on\ \{-b..-a\})\ (\lambda x. f(-x))\ i \longleftrightarrow has_bochner_integral\ (lebesgue_on\ \{a..b\})\ f\ i$
by (*auto simp: dest: has_bochner_integral_reflect_real_lemma*)

lemma *integrable_reflect_real[simp]*:
fixes $f :: real \Rightarrow 'a :: euclidean_space$
shows $integrable\ (lebesgue_on\ \{-b..-a\})\ (\lambda x. f(-x)) \longleftrightarrow integrable\ (lebesgue_on\ \{a..b\})\ f$
by (*metis has_bochner_integral_iff has_bochner_integral_reflect_real*)

lemma *integral_reflect_real[simp]*:
fixes $f :: real \Rightarrow 'a :: euclidean_space$
shows $integral^L\ (lebesgue_on\ \{-b..-a\})\ (\lambda x. f(-x)) = integral^L\ (lebesgue_on\ \{a..b::real\})\ f$
using *has_bochner_integral_reflect_real* [*of b a f*]
by (*metis has_bochner_integral_iff not_integrable_integral_eq*)

9.8.26 More results on integrability

lemma *integrable_on_all_intervals_UNIV*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$
assumes $intf: \bigwedge a\ b. f\ integrable_on\ cbox\ a\ b$
and $normf: \bigwedge x. norm(f\ x) \leq g\ x$ **and** $g: g\ integrable_on\ UNIV$
shows $f\ integrable_on\ UNIV$
proof –
have $intg: (\forall a\ b. g\ integrable_on\ cbox\ a\ b)$
and $gle_e: \forall e>0. \exists B>0. \forall a\ b\ c\ d.$
 $ball\ 0\ B \subseteq cbox\ a\ b \wedge cbox\ a\ b \subseteq cbox\ c\ d \longrightarrow$
 $|integral\ (cbox\ a\ b)\ g - integral\ (cbox\ c\ d)\ g|$
 $< e$
using g
by (*auto simp: integrable_alt_subset [of _ UNIV] intf*)
have $le: norm\ (integral\ (cbox\ a\ b)\ f - integral\ (cbox\ c\ d)\ f) \leq |integral\ (cbox\ a\ b)\ g - integral\ (cbox\ c\ d)\ g|$
if $cbox\ a\ b \subseteq cbox\ c\ d$ **for** $a\ b\ c\ d$
proof –
have $norm\ (integral\ (cbox\ a\ b)\ f - integral\ (cbox\ c\ d)\ f) = norm\ (integral\ (cbox\ c\ d - cbox\ a\ b)\ f)$
using $intf$ **that** **by** (*simp add: norm_minus_commute integral_setdiff*)
also **have** $\dots \leq integral\ (cbox\ c\ d - cbox\ a\ b)\ g$
proof (*rule integral_norm_bound_integral [OF _ _ normf]*)
show $f\ integrable_on\ cbox\ c\ d - cbox\ a\ b$ $g\ integrable_on\ cbox\ c\ d - cbox\ a\ b$
by (*meson integrable_integral integrable_setdiff intf intg negligible_setdiff that*)+
qed
also **have** $\dots = integral\ (cbox\ c\ d)\ g - integral\ (cbox\ a\ b)\ g$
using $intg$ **that** **by** (*simp add: integral_setdiff*)
also **have** $\dots \leq |integral\ (cbox\ a\ b)\ g - integral\ (cbox\ c\ d)\ g|$
by *simp*
finally **show** *?thesis* .
qed
show *?thesis*
using gle_e
apply (*simp add: integrable_alt_subset [of _ UNIV] intf*)
apply (*erule imp_forward all_forward ex_forward asm_rl*)+
by (*meson not_less order_trans le*)
qed

lemma *integrable_on_all_intervals_integrable_bound*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::banach$
assumes $intf: \bigwedge a\ b. (\lambda x. if\ x \in S\ then\ f\ x\ else\ 0)\ integrable_on\ cbox\ a\ b$
and $normf: \bigwedge x. x \in S \implies norm(f\ x) \leq g\ x$ **and** $g: g\ integrable_on\ S$
shows $f\ integrable_on\ S$
using *integrable_on_all_intervals_UNIV [OF intf, of ($\lambda x. if\ x \in S\ then\ g\ x\ else\ 0$)]*
by (*simp add: g_integrable_restrict_UNIV normf*)

lemma *measurable_bounded_lemma*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes f: f  $\in$  borel_measurable_lebesgue and g: g integrable_on_cbox a b
and normf:  $\bigwedge x. x \in \text{cbox } a \ b \implies \text{norm}(f\ x) \leq g\ x$ 
shows f integrable_on_cbox a b
proof -
  have g absolutely_integrable_on_cbox a b
  by (metis (full_types) add_increasing_g_le_add_same_cancel1 nonnegative_absolutely_integrable_1
norm_ge_zero normf)
  then have integrable (lebesgue_on (cbox a b)) g
  by (simp add: integrable_restrict_space set_integrable_def)
  then have integrable (lebesgue_on (cbox a b)) f
  proof (rule Bochner_Integration.integrable_bound)
    show AE x in lebesgue_on (cbox a b). norm (f x)  $\leq$  norm (g x)
    by (rule AE_I2) (auto intro: normf order_trans)
  qed (simp add: f_measurable_restrict_space1)
  then show ?thesis
  by (simp add: integrable_on_lebesgue_on)
qed

```

proposition *measurable_bounded_by_integrable_imp_integrable*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes f: f  $\in$  borel_measurable (lebesgue_on S) and g: g integrable_on S
and normf:  $\bigwedge x. x \in S \implies \text{norm}(f\ x) \leq g\ x$  and S: S  $\in$  sets_lebesgue
shows f integrable_on S
proof (rule integrable_on_all_intervals_integrable_bound [OF _ normf g])
show ( $\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0$ ) integrable_on_cbox a b for a b
proof (rule measurable_bounded_lemma)
  show ( $\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0$ )  $\in$  borel_measurable_lebesgue
  by (simp add: S_borel_measurable_if_f)
  show ( $\lambda x. \text{if } x \in S \text{ then } g\ x \text{ else } 0$ ) integrable_on_cbox a b
  by (simp add: g_integrable_altD(1))
  show norm (if x  $\in$  S then f x else 0)  $\leq$  (if x  $\in$  S then g x else 0) for x
  using normf by simp
qed
qed

```

lemma *measurable_bounded_by_integrable_imp_lebesgue_integrable*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes f: f  $\in$  borel_measurable (lebesgue_on S) and g: integrable (lebesgue_on
S) g
and normf:  $\bigwedge x. x \in S \implies \text{norm}(f\ x) \leq g\ x$  and S: S  $\in$  sets_lebesgue
shows integrable (lebesgue_on S) f
proof -
  have f absolutely_integrable_on S
  by (metis (no_types) S_absolutely_integrable_integrable_bound f g integrable_on_lebesgue_on
measurable_bounded_by_integrable_imp_integrable normf)
  then show ?thesis
  by (simp add: S_integrable_restrict_space set_integrable_def)

```

3010

qed

lemma *measurable_bounded_by_integrable_imp_integrable_real*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow \text{real}$
assumes $f \in \text{borel_measurable} (\text{lebesgue_on } S)$ $g \text{ integrable_on } S \wedge x. x \in S$
 $\implies \text{abs}(f\ x) \leq g\ x \ S \in \text{sets lebesgue}$
shows $f \text{ integrable_on } S$
using *measurable_bounded_by_integrable_imp_integrable* [of $f\ S\ g$] *assms* **by**
simp

9.8.27 Relation between Borel measurability and integrability.

lemma *integrable_imp_measurable_weak*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $S \in \text{sets lebesgue}$ $f \text{ integrable_on } S$
shows $f \in \text{borel_measurable} (\text{lebesgue_on } S)$
by (*metis* (*mono_tags*, *lifting*) *assms* *has_integral_implies_lebesgue_measurable*
borel_measurable_restrict_space_iff_integrable_on_def *sets.Int_space_eq2*)

lemma *integrable_imp_measurable*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $f \text{ integrable_on } S$
shows $f \in \text{borel_measurable} (\text{lebesgue_on } S)$

proof –

have ($\text{UNIV}::'a \text{ set}$) $\in \text{sets lborel}$
by *simp*
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *assms* *borel_measurable_if_D_integrable_imp_measurable_weak*
integrable_restrict_UNIV_lebesgue_on_UNIV_eq_sets_lebesgue_on_refl)
qed

lemma *integrable_iff_integrable_on*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $S \in \text{sets lebesgue}$ $(\int^+ x. \text{ennreal} (\text{norm} (f\ x)) \ \partial \text{lebesgue_on } S) < \infty$
shows $\text{integrable} (\text{lebesgue_on } S) f \longleftrightarrow f \text{ integrable_on } S$
using *assms* *integrable_iff_bounded_integrable_imp_measurable_integrable_on_lebesgue_on*
by *blast*

lemma *absolutely_integrable_measurable*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $S \in \text{sets lebesgue}$
shows $f \text{ absolutely_integrable_on } S \longleftrightarrow f \in \text{borel_measurable} (\text{lebesgue_on } S)$
 $\wedge \text{integrable} (\text{lebesgue_on } S) (\text{norm} \circ f)$
(is *?lhs* = *?rhs*)

proof

assume $L: ?lhs$
then have $f \in \text{borel_measurable} (\text{lebesgue_on } S)$
by (*simp* *add*: *absolutely_integrable_on_def_integrable_imp_measurable*)

```

then show ?rhs
  using assms set_integrable_norm [of lebesgue S f] L
  by (simp add: integrable_restrict_space set_integrable_def)
next
  assume ?rhs then show ?lhs
    using assms integrable_on_lebesgue_on
    by (metis absolutely_integrable_integrable_bound comp_def eq_iff measurable_bounded_by_integrable_imp_integrable)
qed

lemma absolutely_integrable_measurable_real:
  fixes  $f :: 'a::euclidean\_space \Rightarrow \text{real}$ 
  assumes  $S \in \text{sets lebesgue}$ 
  shows  $f \text{ absolutely\_integrable\_on } S \iff$ 
     $f \in \text{borel\_measurable (lebesgue\_on } S) \wedge \text{integrable (lebesgue\_on } S) (\lambda x. |f$ 
 $x|)$ 
  by (simp add: absolutely_integrable_measurable assms o_def)

lemma absolutely_integrable_measurable_real':
  fixes  $f :: 'a::euclidean\_space \Rightarrow \text{real}$ 
  assumes  $S \in \text{sets lebesgue}$ 
  shows  $f \text{ absolutely\_integrable\_on } S \iff f \in \text{borel\_measurable (lebesgue\_on } S)$ 
 $\wedge (\lambda x. |f x|) \text{ integrable\_on } S$ 
  by (metis abs_absolutely_integrableI_1 absolutely_integrable_measurable_real
assms
  measurable_bounded_by_integrable_imp_integrable order_refl real_norm_def
set_integrable_abs set_lebesgue_integral_eq_integral(1))

lemma absolutely_integrable_imp_borel_measurable:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \text{ absolutely\_integrable\_on } S$   $S \in \text{sets lebesgue}$ 
  shows  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
  using absolutely_integrable_measurable assms by blast

lemma measurable_bounded_by_integrable_imp_absolutely_integrable:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $f \in \text{borel\_measurable (lebesgue\_on } S)$   $S \in \text{sets lebesgue}$ 
  and  $g \text{ integrable\_on } S$  and  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq (g x)$ 
  shows  $f \text{ absolutely\_integrable\_on } S$ 
  using assms absolutely_integrable_integrable_bound measurable_bounded_by_integrable_imp_integrable
by blast

proposition negligible_differentiable_vimage:
  fixes  $f :: 'a \Rightarrow 'a::euclidean\_space$ 
  assumes negligible T
  and  $f': \bigwedge x. x \in S \implies \text{inj}(f' x)$ 
  and  $\text{derf}: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
  shows negligible \{x \in S. f x \in T\}
proof –

```

```

define  $U$  where
   $U \equiv \lambda n::nat. \{x \in S. \forall y. y \in S \wedge norm(y - x) < 1/n$ 
     $\longrightarrow norm(y - x) \leq n * norm(f y - f x)\}$ 
have negligible  $\{x \in U n. f x \in T\}$  if  $n > 0$  for  $n$ 
proof (subst locally_negligible_alt, clarify)
  fix  $a$ 
  assume  $a: a \in U n$  and  $fa: f a \in T$ 
  define  $V$  where  $V \equiv \{x. x \in U n \wedge f x \in T\} \cap ball a (1 / n / 2)$ 
  show  $\exists V. openin (top\_of\_set \{x \in U n. f x \in T\}) V \wedge a \in V \wedge negligible V$ 
  proof (intro exI conjI)
    have  $noxy: norm(x - y) \leq n * norm(f x - f y)$  if  $x \in V y \in V$  for  $x y$ 
      using that unfolding U_def V_def mem_Collect_eq Int_iff mem_ball
dist_norm
      by (meson norm_triangle_half_r)
    then have inj_on  $f V$ 
      by (force simp: inj_on_def)
    then obtain  $g$  where  $g: \bigwedge x. x \in V \implies g(f x) = x$ 
      by (metis inv_into_f_f)
    have  $\exists T' B. open T' \wedge f x \in T' \wedge$ 
       $(\forall y \in f' V \cap T \cap T'. norm(g y - g(f x)) \leq B * norm(y - f x))$ 
      if  $f x \in T x \in V$  for  $x$ 
      using that noxy
      by (rule_tac x = ball(f x) 1 in exI (force simp: g))
    then have negligible  $(g' (f' V \cap T))$ 
      by (force simp: <negligible T> negligible_Int intro!: negligible_locally_Lipschitz_image)
    moreover have  $V \subseteq g' (f' V \cap T)$ 
      by (force simp: g_image_iff V_def)
    ultimately show negligible  $V$ 
      by (rule negligible_subset)
    qed (use a fa V_def that in auto)
qed
with negligible_countable_Union have negligible  $(\bigcup n \in \{0<..\}. \{x. x \in U n \wedge$ 
 $f x \in T\})$ 
  by auto
moreover have  $\{x \in S. f x \in T\} \subseteq (\bigcup n \in \{0<..\}. \{x. x \in U n \wedge f x \in T\})$ 
proof clarsimp
  fix  $x$ 
  assume  $x \in S$  and  $f x \in T$ 
  then obtain inj: inj(f' x) and der: (f has_derivative f' x) (at x within S)
    using assms by metis
  moreover have linear(f' x)
    and eps:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \exists \delta > 0. \forall y \in S. norm(y - x) < \delta \longrightarrow$ 
       $norm(f y - f x - f' x (y - x)) \leq \varepsilon * norm(y - x)$ 
    using der by (auto simp: has_derivative_within_alt linear_linear)
  ultimately obtain  $g$  where linear  $g$  and  $g \circ f' x = id$ 
    using linear_injective_left_inverse by metis
  then obtain  $B$  where  $B > 0$  and  $B: \bigwedge z. B * norm z \leq norm(f' x z)$ 
    using linear_invertible_bounded_below_pos <linear(f' x)> by blast
  then obtain  $i$  where  $i \neq 0$  and  $i: 1 / real i < B$ 

```



```

    by (metis (full_types) inverse_eq_divide real_arch_invD)
  then obtain  $\delta$  where  $\delta > 0$ 
    and  $\delta: \bigwedge y. \llbracket y \in S; \text{norm } (y - x) < \delta \rrbracket \implies$ 
       $\text{norm } (f y - f x - f' x (y - x)) \leq (B - 1 / \text{real } i) * \text{norm } (y - x)$ 
    using eps [of  $B - 1/i$ ] by auto
  then obtain  $j$  where  $j \neq 0$  and  $j: \text{inverse } (\text{real } j) < \delta$ 
    using real_arch_inverse by blast
  have  $\text{norm } (y - x) / (\text{max } i j) \leq \text{norm } (f y - f x)$ 
    if  $y \in S$  and less:  $\text{norm } (y - x) < 1 / (\text{max } i j)$  for  $y$ 
  proof -
    have  $1 / \text{real } (\text{max } i j) < \delta$ 
      using  $j \langle j \neq 0 \rangle \langle 0 < \delta \rangle$ 
    by (auto simp: field_split_simps max_mult_distrib_left of_nat_max)
    then have  $\text{norm } (y - x) < \delta$ 
      using less by linarith
    with  $\delta \langle y \in S \rangle$  have le:  $\text{norm } (f y - f x - f' x (y - x)) \leq B * \text{norm } (y - x)$ 
  -  $\text{norm } (y - x) / i$ 
    by (auto simp: algebra_simps)
    have  $\text{norm } (y - x) / \text{real } (\text{max } i j) \leq \text{norm } (y - x) / \text{real } i$ 
      using  $\langle i \neq 0 \rangle \langle j \neq 0 \rangle$  by (simp add: field_split_simps max_mult_distrib_left
of_nat_max less_max_iff_disj)
    also have  $\dots \leq \text{norm } (f y - f x)$ 
      using  $B$  [of  $y - x$ ] le norm_triangle_ineq3 [of  $f y - f x f' x (y - x)$ ]
      by linarith
    finally show ?thesis .
  qed
  with  $\langle x \in S \rangle \langle i \neq 0 \rangle \langle j \neq 0 \rangle$  show  $\exists n \in \{0 < ..\}. x \in U n$ 
    by (rule_tac  $x = \text{max } i j$  in bexI) (auto simp: field_simps U_def less_max_iff_disj)
  qed
  ultimately show ?thesis
    by (rule negligible_subset)
  qed

```

lemma absolutely_integrable_Un:

```

  fixes  $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$ 
  assumes  $S: f \text{ absolutely\_integrable\_on } S$  and  $T: f \text{ absolutely\_integrable\_on } T$ 
  shows  $f \text{ absolutely\_integrable\_on } (S \cup T)$ 
  proof -
    have [simp]:  $\{x. (\text{if } x \in A \text{ then } f x \text{ else } 0) \neq 0\} = \{x \in A. f x \neq 0\}$  for  $A$ 
      by auto
    let ?ST =  $\{x \in S. f x \neq 0\} \cap \{x \in T. f x \neq 0\}$ 
    have ?ST  $\in$  sets lebesgue
  proof (rule Sigma_Algebra.sets.Int)
    have  $f \text{ integrable\_on } S$ 
      using  $S \text{ absolutely\_integrable\_on\_def}$  by blast
    then have  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ integrable\_on UNIV}$ 
      by (simp add: integrable_restrict_UNIV)
    then have borel:  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \in \text{borel\_measurable } (\text{lebesgue\_on UNIV})$ 

```

```

    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then show  $\{x \in S. f x \neq 0\} \in \text{sets lebesgue}$ 
    unfolding borel_measurable_vimage_open
    by (rule allE [where  $x = -\{0\}$ ]) auto
next
  have  $f \text{ integrable\_on } T$ 
    using  $T \text{ absolutely\_integrable\_on\_def}$  by blast
  then have  $(\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ integrable\_on UNIV}$ 
    by (simp add: integrable_restrict_UNIV)
  then have borel:  $(\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \in \text{borel\_measurable (lebesgue\_on UNIV)}$ 
    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then show  $\{x \in T. f x \neq 0\} \in \text{sets lebesgue}$ 
    unfolding borel_measurable_vimage_open
    by (rule allE [where  $x = -\{0\}$ ]) auto
qed
then have  $f \text{ absolutely\_integrable\_on } ?ST$ 
  by (rule set_integrable_subset [OF S]) auto
then have Int:  $(\lambda x. \text{if } x \in ?ST \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
  using absolutely_integrable_restrict_UNIV by blast
have  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
   $(\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
  using S T absolutely_integrable_restrict_UNIV by blast+
then have  $(\lambda x. (\text{if } x \in S \text{ then } f x \text{ else } 0) + (\text{if } x \in T \text{ then } f x \text{ else } 0)) \text{ absolutely\_integrable\_on UNIV}$ 
  by (rule set_integral_add)
then have  $(\lambda x. ((\text{if } x \in S \text{ then } f x \text{ else } 0) + (\text{if } x \in T \text{ then } f x \text{ else } 0)) - (\text{if } x \in ?ST \text{ then } f x \text{ else } 0)) \text{ absolutely\_integrable\_on UNIV}$ 
  using Int by (rule set_integral_diff)
then have  $(\lambda x. \text{if } x \in S \cup T \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
  by (rule absolutely_integrable_spike) (auto intro: empty_imp_negligible)
then show ?thesis
  unfolding absolutely_integrable_restrict_UNIV .
qed

```

lemma *absolutely_integrable_on_combine:*

```

fixes  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
assumes  $f \text{ absolutely\_integrable\_on } \{a..c\}$ 
  and  $f \text{ absolutely\_integrable\_on } \{c..b\}$ 
  and  $a \leq c$ 
  and  $c \leq b$ 
shows  $f \text{ absolutely\_integrable\_on } \{a..b\}$ 
by (metis absolutely_integrable_Un assms ivl_disj_un_two_touch(4))

```

lemma *uniform_limit_set_lebesgue_integral_at_top:*

```

fixes  $f :: 'a \Rightarrow \text{real} \Rightarrow 'b::\{\text{banach, second\_countable\_topology}\}$ 
  and  $g :: \text{real} \Rightarrow \text{real}$ 
assumes bound:  $\bigwedge x y. x \in A \implies y \geq a \implies \text{norm } (f x y) \leq g y$ 
assumes integrable:  $\text{set\_integrable } M \{a.. \} g$ 

```

```

assumes measurable:  $\bigwedge x. x \in A \implies \text{set\_borel\_measurable } M \{a..\} (f x)$ 
assumes sets borel  $\subseteq$  sets  $M$ 
shows uniform_limit  $A (\lambda b x. \text{LINT } y:\{a..b\}|M. f x y) (\lambda x. \text{LINT } y:\{a..\}|M. f x y)$  at_top
proof (cases  $A = \{\}$ )
  case False
  then obtain  $x$  where  $x: x \in A$  by auto
  have  $g\_nonneg: g y \geq 0$  if  $y \geq a$  for  $y$ 
  proof -
    have  $0 \leq \text{norm } (f x y)$  by simp
    also have  $\dots \leq g y$  using bound[OF x that] by simp
    finally show ?thesis .
  qed

have integrable':  $\text{set\_integrable } M \{a..\} (\lambda y. f x y)$  if  $x \in A$  for  $x$ 
  unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound)
    show  $\text{integrable } M (\lambda x. \text{indicator } \{a..\} x * g x)$ 
    using integrable by (simp add: set_integrable_def)
    show  $(\lambda y. \text{indicat\_real } \{a..\} y *_R f x y) \in \text{borel\_measurable } M$  using measurable[OF that]
    by (simp add: set_borel_measurable_def)
    show  $\text{AE } y \text{ in } M. \text{norm } (\text{indicat\_real } \{a..\} y *_R f x y) \leq \text{norm } (\text{indicat\_real } \{a..\} y * g y)$ 
    using bound[OF that] by (intro AE_I2) (auto simp: indicator_def g_nonneg)
  qed

show ?thesis
proof (rule uniform_limitI)
  fix  $e :: \text{real}$  assume  $e: e > 0$ 
  have sets [intro]:  $A \in \text{sets } M$  if  $A \in \text{sets borel}$  for  $A$ 
  using that assms by blast

  have  $(\lambda b. \text{LINT } y:\{a..b\}|M. g y) \longrightarrow (\text{LINT } y:\{a..\}|M. g y)$  at_top
  by (intro tendsto_set_lebesgue_integral_at_top assms sets) auto
  with  $e$  obtain  $b0 :: \text{real}$  where  $b0: \forall b \geq b0. |(\text{LINT } y:\{a..\}|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| < e$ 
  by (auto simp: tendsto_iff eventually_at_top_linorder dist_real_def abs_minus_commute)
  define  $b$  where  $b = \max a b0$ 
  have  $a \leq b$  by (simp add: b_def)
  from  $b0$  have  $|(\text{LINT } y:\{a..\}|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| < e$ 
  by (auto simp: b_def)
  also have  $\{a..\} = \{a..b\} \cup \{b<..\}$  by (auto simp: b_def)
  also have  $|(\text{LINT } y:\dots|M. g y) - (\text{LINT } y:\{a..b\}|M. g y)| = |(\text{LINT } y:\{b<..\}|M. g y)|$ 
  using  $\langle a \leq b \rangle$  by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF integrable])
  also have  $(\text{LINT } y:\{b<..\}|M. g y) \geq 0$ 
  using  $g\_nonneg \langle a \leq b \rangle$  unfolding set_lebesgue_integral_def

```

by (intro Bochner_Integration.integral_nonneg) (auto simp: indicator_def)
 hence $|(LINT y:\{b<..\}|M. g y)| = (LINT y:\{b<..\}|M. g y)$ by simp
 finally have less: $(LINT y:\{b<..\}|M. g y) < e$.

have eventually $(\lambda b. b \geq b0)$ at_top by (rule eventually_ge_at_top)
 moreover have eventually $(\lambda b. b \geq a)$ at_top by (rule eventually_ge_at_top)
 ultimately show eventually $(\lambda b. \forall x \in A.$
 $dist (LINT y:\{a..b}|M. f x y) (LINT y:\{a..}|M. f x y) < e)$

at_top
 proof eventually_elim
 case (elim b)
 show ?case
 proof
 fix x assume x: $x \in A$
 have $dist (LINT y:\{a..b}|M. f x y) (LINT y:\{a..}|M. f x y) =$
 $norm ((LINT y:\{a..}|M. f x y) - (LINT y:\{a..b}|M. f x y))$
 by (simp add: dist_norm norm_minus_commute)
 also have $\{a..} = \{a..b\} \cup \{b<..\}$ using elim by auto
 also have $(LINT y:\dots|M. f x y) - (LINT y:\{a..b}|M. f x y) = (LINT$
 $y:\{b<..\}|M. f x y)$
 using elim x
 by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF inte-
 grable'])
 also have $norm \dots \leq (LINT y:\{b<..\}|M. norm (f x y))$ using elim x
 by (intro set_integral_norm_bound set_integrable_subset[OF integrable'])
 auto
 also have $\dots \leq (LINT y:\{b<..\}|M. g y)$ using elim x bound g_nonneg
 by (intro set_integral_mono set_integrable_norm set_integrable_subset[OF
 integrable']
 $set_integrable_subset[OF\ integrable])\ auto$
 also have $(LINT y:\{b<..\}|M. g y) \geq 0$
 using g_nonneg $\langle a \leq b \rangle$ unfolding set_lebesgue_integral_def
 by (intro Bochner_Integration.integral_nonneg) (auto simp: indicator_def)
 hence $(LINT y:\{b<..\}|M. g y) = |(LINT y:\{b<..\}|M. g y)|$ by simp
 also have $\dots = |(LINT y:\{a..b\} \cup \{b<..\}|M. g y) - (LINT y:\{a..b\}|M. g$
 $y)|$
 using elim by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF
 integrable'])
 also have $\{a..b\} \cup \{b<..\} = \{a..}$ using elim by auto
 also have $|(LINT y:\{a..}|M. g y) - (LINT y:\{a..b\}|M. g y)| < e$
 using b0 elim by blast
 finally show $dist (LINT y:\{a..b}|M. f x y) (LINT y:\{a..}|M. f x y) < e$.
 qed
 qed
 qed
 qed auto

Differentiability of inverse function (most basic form)**proposition** *has_derivative_inverse_within:*fixes $f :: 'a::\text{real_normed_vector} \Rightarrow 'b::\text{euclidean_space}$ assumes $\text{der_f}: (f \text{ has_derivative } f') \text{ (at } a \text{ within } S)$ and $\text{cont_g}: \text{continuous (at } (f\ a) \text{ within } f^{-1}\ S) \ g$ and $a \in S$ linear g' and $\text{id}: g' \circ f' = \text{id}$ and $\text{gf}: \bigwedge x. x \in S \implies g(f\ x) = x$ shows $(g \text{ has_derivative } g') \text{ (at } (f\ a) \text{ within } f^{-1}\ S)$ **proof** –have $[\text{simp}]: g'(f'\ x) = x$ for x by $(\text{simp add: local.id pointfree_idE})$ have *bounded_linear* f' and $f': \bigwedge e. e > 0 \implies \exists d > 0. \forall y \in S. \text{norm } (y - a) < d \implies$
 $\text{norm } (f\ y - f\ a - f'(y - a)) \leq e * \text{norm } (y - a)$ using der_f by $(\text{auto simp: has_derivative_within_alt})$ **obtain** C where $C > 0$ and $C: \bigwedge x. \text{norm } (g'\ x) \leq C * \text{norm } x$ using *linear_bounded_pos* $[OF \langle \text{linear } g' \rangle]$ by *metis***obtain** $B\ k$ where $B > 0\ k > 0$ and $Bk: \bigwedge x. [x \in S; \text{norm}(f\ x - f\ a) < k] \implies \text{norm}(x - a) \leq B * \text{norm}(f\ x - f\ a)$ **proof** –**obtain** B where $B > 0$ and $B: \bigwedge x. B * \text{norm } x \leq \text{norm } (f'\ x)$ using *linear_inj_bounded_below_pos* $[of\ f'] \langle \text{linear } g' \rangle \text{ id } \text{der_f has_derivative_linear}$
linear_invertible_bounded_below_pos by *blast***then obtain** d where $d > 0$ and $d: \bigwedge y. [y \in S; \text{norm } (y - a) < d] \implies$
 $\text{norm } (f\ y - f\ a - f'(y - a)) \leq B / 2 * \text{norm } (y - a)$ using f' $[of\ B/2]$ by *auto***then obtain** e where $e > 0$ and $e: \bigwedge x. [x \in S; \text{norm } (f\ x - f\ a) < e] \implies \text{norm } (g(f\ x) - g(f\ a)) < d$ using *cont_g* by $(\text{auto simp: continuous_within_eps_delta_dist_norm})$ **show** *thesis***proof****show** $2/B > 0$ using $\langle B > 0 \rangle$ by *simp***show** $\text{norm } (x - a) \leq 2 / B * \text{norm } (f\ x - f\ a)$ **if** $x \in S$ $\text{norm } (f\ x - f\ a) < e$ **for** x **proof** –**have** $xa: \text{norm } (x - a) < d$ using e $[OF\ \text{that}]$ gf by $(\text{simp add: } \langle a \in S \rangle \text{ that})$ **have** $*$: $[\text{norm}(y - f') \leq B / 2 * \text{norm } x; B * \text{norm } x \leq \text{norm } f']$ $\implies \text{norm } y \geq B / 2 * \text{norm } x$ **for** $y\ f'::'b$ and $x::'a$ using *norm_triangle_ineq3* $[of\ y\ f']$ by *linarith***show** *?thesis*using $*$ $[OF\ d\ [OF\ \langle x \in S \rangle\ xa]\ B]\ \langle B > 0 \rangle$ by $(\text{simp add: field_simps})$ **qed****qed** $(\text{use } \langle e > 0 \rangle \text{ in auto})$ **qed****show** *?thesis*

```

unfolding has_derivative_within_alt
proof (intro conjI impI allI)
  show bounded_linear g'
    using ⟨linear g'⟩ by (simp add: linear_linear)
next
fix e :: real
assume e > 0
then obtain d where d > 0
  and d:  $\bigwedge y. \llbracket y \in S; \text{norm } (y - a) < d \rrbracket \implies$ 
     $\text{norm } (f y - f a - f' (y - a)) \leq e / (B * C) * \text{norm } (y - a)$ 
  using f' [of e / (B * C)] ⟨B > 0⟩ ⟨C > 0⟩ by auto
have  $\text{norm } (x - a - g' (f x - f a)) \leq e * \text{norm } (f x - f a)$ 
  if  $x \in S$  and lt_k:  $\text{norm } (f x - f a) < k$  and lt_dB:  $\text{norm } (f x - f a) < d/B$ 
for x
proof -
  have  $\text{norm } (x - a) \leq B * \text{norm } (f x - f a)$ 
    using Bk lt_k ⟨x ∈ S⟩ by blast
  also have ... < d
    by (metis ⟨0 < B⟩ lt_dB mult.commute pos_less_divide_eq)
  finally have lt_d:  $\text{norm } (x - a) < d$  .
  have  $\text{norm } (x - a - g' (f x - f a)) \leq \text{norm } (g' (f x - f a - (f' (x - a))))$ 
    by (simp add: linear_diff [OF ⟨linear g'⟩] norm_minus_commute)
  also have ...  $\leq C * \text{norm } (f x - f a - f' (x - a))$ 
    using C by blast
  also have ...  $\leq e * \text{norm } (f x - f a)$ 
proof -
  have  $\text{norm } (f x - f a - f' (x - a)) \leq e / (B * C) * \text{norm } (x - a)$ 
    using d [OF ⟨x ∈ S⟩ lt_d] .
  also have ...  $\leq (\text{norm } (f x - f a) * e) / C$ 
    using ⟨B > 0⟩ ⟨C > 0⟩ ⟨e > 0⟩ by (simp add: field_simps Bk lt_k ⟨x ∈
S⟩)
  finally show ?thesis
    using ⟨C > 0⟩ by (simp add: field_simps)
qed
finally show ?thesis .
qed
with ⟨k > 0⟩ ⟨B > 0⟩ ⟨d > 0⟩ ⟨a ∈ S⟩
show  $\exists d > 0. \forall y \in f^{-1} S.$ 
   $\text{norm } (y - f a) < d \implies$ 
   $\text{norm } (g y - g (f a) - g' (y - f a)) \leq e * \text{norm } (y - f a)$ 
  by (rule_tac x=min k (d / B) in exI) (auto simp: gf)
qed
qed
end

```

9.9 Harmonic Numbers

theory *Harmonic_Numbers*

```

imports
  Complex_Transcendental
  Summation_Tests
begin

```

The definition of the Harmonic Numbers and the Euler-Mascheroni constant. Also provides a reasonably accurate approximation of $\ln 2$ and the Euler-Mascheroni constant.

9.9.1 The Harmonic numbers

```

definition harm :: nat  $\Rightarrow$  'a :: real_normed_field where
  harm n = ( $\sum$  k=1..n. inverse (of_nat k))

```

```

lemma harm_altdef: harm n = ( $\sum$  k<n. inverse (of_nat (Suc k)))
unfolding harm_def by (induction n) simp_all

```

```

lemma harm_Suc: harm (Suc n) = harm n + inverse (of_nat (Suc n))
by (simp add: harm_def)

```

```

lemma harm_nonneg: harm n  $\geq$  (0 :: 'a :: {real_normed_field, linordered_field})
unfolding harm_def by (intro sum_nonneg) simp_all

```

```

lemma harm_pos: n > 0  $\implies$  harm n > (0 :: 'a :: {real_normed_field, linordered_field})
unfolding harm_def by (intro sum_pos) simp_all

```

```

lemma harm_mono: m  $\leq$  n  $\implies$  harm m  $\leq$  (harm n :: 'a :: {real_normed_field, linordered_field})
by (simp add: harm_def sum_mono2)

```

```

lemma of_real_harm: of_real (harm n) = harm n
unfolding harm_def by simp

```

```

lemma abs_harm [simp]: (abs (harm n) :: real) = harm n
using harm_nonneg[of n] by (rule abs_of_nonneg)

```

```

lemma norm_harm: norm (harm n) = harm n
by (subst of_real_harm [symmetric]) (simp add: harm_nonneg)

```

```

lemma harm_expand:
  harm 0 = 0
  harm (Suc 0) = 1
  harm (numeral n) = harm (pred_numeral n) + inverse (numeral n)

```

```

proof -
  have numeral n = Suc (pred_numeral n) by simp
  also have harm ... = harm (pred_numeral n) + inverse (numeral n)
    by (subst harm_Suc, subst numeral_eq_Suc[symmetric]) simp
  finally show harm (numeral n) = harm (pred_numeral n) + inverse (numeral
n) .
qed (simp_all add: harm_def)

```

theorem *not_convergent_harm*: \neg convergent (harm :: nat \Rightarrow 'a :: real_normed_field)

proof –

have convergent (λn . norm (harm n :: 'a)) \longleftrightarrow
 convergent (harm :: nat \Rightarrow real) **by** (simp add: norm_harm)
also have ... \longleftrightarrow convergent (λn . $\sum k=\text{Suc } 0.. \text{Suc } n$. inverse (of_nat k) :: real)
 unfolding harm_def[abs_def] **by** (subst convergent_Suc_iff) simp_all
also have ... \longleftrightarrow convergent (λn . $\sum k \leq n$. inverse (of_nat (Suc k)) :: real)
 by (subst sum.shift_bounds_cl_Suc_ivl) (simp add: atLeast0AtMost)
also have ... \longleftrightarrow summable (λn . inverse (of_nat n) :: real)
 by (subst summable_Suc_iff [symmetric]) (simp add: summable_iff_convergent')
also have \neg ... **by** (rule not_summable_harmonic)
finally show ?thesis **by** (blast dest: convergent_norm)

qed

lemma harm_pos_iff [simp]: harm n > (0 :: 'a :: {real_normed_field, linordered_field})
 \longleftrightarrow n > 0

by (rule iffI, cases n, simp add: harm_expand, simp, rule harm_pos)

lemma ln_diff_le_inverse:

assumes x \geq (1 :: real)

shows ln (x + 1) – ln x < 1 / x

proof –

from assms **have** $\exists z > x$. z < x + 1 \wedge ln (x + 1) – ln x = (x + 1 – x) *
 inverse z

by (intro MVT2) (auto intro!: derivative_eq_intros simp: field_simps)

then obtain z **where** z > x z < x + 1 ln (x + 1) – ln x = inverse z **by** auto

have ln (x + 1) – ln x = inverse z **by** fact

also from z(1,2) assms **have** ... < 1 / x **by** (simp add: field_simps)

finally show ?thesis .

qed

lemma ln_le_harm: ln (real n + 1) \leq (harm n :: real)

proof (induction n)

fix n **assume** IH: ln (real n + 1) \leq harm n

have ln (real (Suc n) + 1) = ln (real n + 1) + (ln (real n + 2) – ln (real n + 1)) **by** simp

also have (ln (real n + 2) – ln (real n + 1)) \leq 1 / real (Suc n)

using ln_diff_le_inverse[of real n + 1] **by** (simp add: add_ac)

also note IH

also have harm n + 1 / real (Suc n) = harm (Suc n) **by** (simp add: harm_Suc field_simps)

finally show ln (real (Suc n) + 1) \leq harm (Suc n) **by** – simp

qed (simp_all add: harm_def)

lemma harm_at_top: filterlim (harm :: nat \Rightarrow real) at_top sequentially

proof (rule filterlim_at_top_mono)

show eventually (λn . harm n \geq ln (real (Suc n))) at_top

using ln_le_harm **by** (intro always_eventually_allI) (simp_all add: add_ac)


```

show filterlim ( $\lambda n. \ln (\text{real } (\text{Suc } n))$ ) at_top sequentially
by (intro filterlim_compose[OF ln_at_top] filterlim_compose[OF filterlim_real_sequentially]
      filterlim_Suc)
qed

```

9.9.2 The Euler-Mascheroni constant

The limit of the difference between the partial harmonic sum and the natural logarithm (approximately 0.577216). This value occurs e.g. in the definition of the Gamma function.

definition *euler_mascheroni* :: 'a :: *real_normed_algebra_1* **where**
euler_mascheroni = *of_real* (*lim* ($\lambda n. \text{harm } n - \ln (\text{of_nat } n)$))

lemma *of_real_euler_mascheroni* [*simp*]: *of_real euler_mascheroni* = *euler_mascheroni*
by (*simp add: euler_mascheroni_def*)

lemma *harm_ge_ln*: *harm* *n* $\geq \ln (\text{real } n + 1)$

proof –

```

have  $\ln (n + 1) = (\sum j < n. \ln (\text{real } (\text{Suc } j + 1)) - \ln (\text{real } (j + 1)))$ 
by (subst sum_lessThan_telescope) auto
also have  $\dots \leq (\sum j < n. 1 / (\text{Suc } j))$ 
proof (intro sum_mono, clarify)
  fix j assume j: j < n
  have  $\exists \xi. \xi > \text{real } j + 1 \wedge \xi < \text{real } j + 2 \wedge$ 
     $\ln (\text{real } j + 2) - \ln (\text{real } j + 1) = (\text{real } j + 2 - (\text{real } j + 1)) * (1 / \xi)$ 
by (intro MVT2) (auto intro!: derivative_eq_intros)
  then obtain  $\xi :: \text{real}$ 
    where  $\xi \in \{\text{real } j + 1.. \text{real } j + 2\} \wedge \ln (\text{real } j + 2) - \ln (\text{real } j + 1) = 1$ 
    /  $\xi$ 
    by auto
  note  $\xi(2)$ 
  also have  $1 / \xi \leq 1 / (\text{Suc } j)$ 
    using  $\xi(1)$  by (auto simp: field_simps)
  finally show  $\ln (\text{real } (\text{Suc } j + 1)) - \ln (\text{real } (j + 1)) \leq 1 / (\text{Suc } j)$ 
    by (simp add: add_ac)
qed
also have  $\dots = \text{harm } n$ 
by (simp add: harm_altdef field_simps)
finally show thesis by (simp add: add_ac)
qed

```

lemma *decseq_harm_diff_ln*: *decseq* ($\lambda n. \text{harm } (\text{Suc } n) - \ln (\text{Suc } n)$)

proof (*rule decseq_SucI*)

```

fix m :: nat
define n where n = Suc m
have n > 0 by (simp add: n_def)
have convex_on {0<..} ( $\lambda x :: \text{real}. -\ln x$ )
by (rule convex_on_realI[where  $f' = \lambda x. -1/x$ ])
    (auto intro!: derivative_eq_intros simp: field_simps)

```

3022

hence $(-1 / (n + 1)) * (real\ n - real\ (n + 1)) \leq (-\ln\ (real\ n)) - (-\ln\ (real\ (n + 1)))$
using $\langle n > 0 \rangle$ **by** $(intro\ convex_on_imp_above_tangent[\mathbf{where}\ A = \{0 < ..\}])$
 $(auto\ intro!: derivative_eq_intros\ simp: interior_open)$
thus $harm\ (Suc\ n) - \ln\ (Suc\ n) \leq harm\ n - \ln\ n$
by $(auto\ simp: harm_Suc\ field_simps)$
qed

lemma *euler_mascheroni_sequence_nonneg*:

assumes $n > 0$

shows $harm\ n - \ln\ (real\ n) \geq (0 :: real)$

proof –

have $\ln\ (real\ n) \leq \ln\ (real\ n + 1)$

using *assms* **by** *simp*

also have $\dots \leq harm\ n$

by $(rule\ harm_ge_ln)$

finally show *?thesis* **by** *simp*

qed

lemma *euler_mascheroni_convergent*: *convergent* $(\lambda n. harm\ n - \ln\ n)$

proof –

have $harm\ (Suc\ n) - \ln\ (real\ (Suc\ n)) \geq 0$ **for** $n :: nat$

using *euler_mascheroni_sequence_nonneg* $[of\ Suc\ n]$ **by** *simp*

hence *convergent* $(\lambda n. harm\ (Suc\ n) - \ln\ (Suc\ n))$

by $(intro\ Bseq_monoseq_convergent\ decseq_bounded[of_ 0]\ decseq_harm_diff_ln\ decseq_imp_monoseq)$

auto

thus *?thesis*

by $(subst\ (asm)\ convergent_Suc_iff)$

qed

lemma *euler_mascheroni_sequence_decreasing*:

$m > 0 \implies m \leq n \implies harm\ n - \ln\ (of_nat\ n) \leq harm\ m - \ln\ (of_nat\ m :: real)$

using *decseqD* $[OF\ decseq_harm_diff_ln, of\ m - 1\ n - 1]$ **by** *simp*

lemma *euler_mascheroni_LIMSEQ*:

$(\lambda n. harm\ n - \ln\ (of_nat\ n) :: real) \longrightarrow euler_mascheroni$

unfolding *euler_mascheroni_def*

by $(simp\ add: convergent_LIMSEQ_iff\ [symmetric]\ euler_mascheroni_convergent)$

lemma *euler_mascheroni_LIMSEQ_of_real*:

$(\lambda n. of_real\ (harm\ n - \ln\ (of_nat\ n))) \longrightarrow$

$(euler_mascheroni :: 'a :: \{real_normed_algebra_1, topological_space\})$

proof –

have $(\lambda n. of_real\ (harm\ n - \ln\ (of_nat\ n))) \longrightarrow (of_real\ (euler_mascheroni) :: 'a)$

by $(intro\ tendsto_of_real\ euler_mascheroni_LIMSEQ)$

thus *?thesis* **by** *simp*

qed

lemma *euler_mascheroni_sum_real*:

$(\lambda n. \text{inverse} (\text{of_nat } (n+1)) + \ln (\text{of_nat } (n+1)) - \ln (\text{of_nat } (n+2))) :: \text{real}$
 $\text{sums euler_mascheroni}$

using *sums_add*[*OF telescope_sums*[*OF LIMSEQ_Suc*[*OF euler_mascheroni_LIMSEQ*]]
 telescope_sums ^[*OF LIMSEQ_inverse_real_of_nat*]]

by (*simp_all add: harm_def algebra_simps*)

lemma *euler_mascheroni_sum*:

$(\lambda n. \text{inverse} (\text{of_nat } (n+1)) + \text{of_real} (\ln (\text{of_nat } (n+1))) - \text{of_real} (\ln (\text{of_nat } (n+2))))$

$\text{sums} (\text{euler_mascheroni} :: 'a :: \{\text{banach, real_normed_field}\})$

proof –

have $(\lambda n. \text{of_real} (\text{inverse} (\text{of_nat } (n+1)) + \ln (\text{of_nat } (n+1)) - \ln (\text{of_nat } (n+2))))$

$\text{sums} (\text{of_real euler_mascheroni} :: 'a :: \{\text{banach, real_normed_field}\})$

by (*subst sums_of_real_iff*) (*rule euler_mascheroni_sum_real*)

thus *?thesis by simp*

qed

theorem *alternating_harmonic_series_sums*: $(\lambda k. (-1)^k / \text{real_of_nat} (\text{Suc } k)) \text{sums } \ln 2$

proof –

let *?f* = $\lambda n. \text{harm } n - \ln (\text{real_of_nat } n)$

let *?g* = $\lambda n. \text{if even } n \text{ then } 0 \text{ else } (2::\text{real})$

let *?em* = $\lambda n. \text{harm } n - \ln (\text{real_of_nat } n)$

have *eventually* $(\lambda n. ?em (2*n) - ?em n + \ln 2 = (\sum k < 2*n. (-1)^k / \text{real_of_nat} (\text{Suc } k))) \text{at_top}$

using *eventually_gt_at_top*[*of 0::nat*]

proof *eventually_elim*

fix *n* :: *nat* **assume** *n*: *n* > 0

have $(\sum k < 2*n. (-1)^k / \text{real_of_nat} (\text{Suc } k)) =$

$(\sum k < 2*n. ((-1)^k + ?g k) / \text{of_nat} (\text{Suc } k)) - (\sum k < 2*n. ?g k / \text{of_nat} (\text{Suc } k))$

by (*simp add: sum.distrib algebra_simps divide_inverse*)

also have $(\sum k < 2*n. ((-1)^k + ?g k) / \text{real_of_nat} (\text{Suc } k)) = \text{harm } (2*n)$

unfolding *harm_altdef* **by** (*intro sum.cong*) (*auto simp: field_simps*)

also have $(\sum k < 2*n. ?g k / \text{real_of_nat} (\text{Suc } k)) = (\sum k | k < 2*n \wedge \text{odd } k. ?g k / \text{of_nat} (\text{Suc } k))$

by (*intro sum.mono_neutral_right*) *auto*

also have $\dots = (\sum k | k < 2*n \wedge \text{odd } k. 2 / (\text{real_of_nat} (\text{Suc } k)))$

by (*intro sum.cong*) *auto*

also have $(\sum k | k < 2*n \wedge \text{odd } k. 2 / (\text{real_of_nat} (\text{Suc } k))) = \text{harm } n$

unfolding *harm_altdef*

by (*intro sum.reindex_cong*[*of* $\lambda n. 2*n+1$]) (*auto simp: inj_on_def field_simps elim!: oddE*)

also have $\text{harm } (2*n) - \text{harm } n = ?em (2*n) - ?em n + \ln 2$ **using** *n*

by (*simp_all add: algebra_simps ln_mult*)

finally show $?em (2*n) - ?em n + \ln 2 = (\sum k < 2*n. (-1)^k / \text{real_of_nat } (Suc\ k)) ..$

qed

moreover have $(\lambda n. ?em (2*n) - ?em n + \ln (2::\text{real}))$

$\longrightarrow \text{euler_mascheroni} - \text{euler_mascheroni} + \ln 2$

by $(\text{intro tendsto_intros euler_mascheroni_LIMSEQ filterlim_compose}[OF \text{euler_mascheroni_LIMSEQ}]$

$\text{filterlim_subseq}) (\text{auto simp: strict_mono_def})$

hence $(\lambda n. ?em (2*n) - ?em n + \ln (2::\text{real})) \longrightarrow \ln 2$ **by** simp

ultimately have $(\lambda n. (\sum k < 2*n. (-1)^k / \text{real_of_nat } (Suc\ k))) \longrightarrow \ln 2$

by $(\text{blast intro: Lim_transform_eventually})$

moreover have $\text{summable } (\lambda k. (-1)^k * \text{inverse } (\text{real_of_nat } (Suc\ k)))$

using $\text{LIMSEQ_inverse_real_of_nat}$

by $(\text{intro summable_Leibniz}(1) \text{decseq_imp_monoseq decseq_SucI}) \text{simp_all}$

hence $A: (\lambda n. \sum k < n. (-1)^k / \text{real_of_nat } (Suc\ k)) \longrightarrow (\sum k. (-1)^k / \text{real_of_nat } (Suc\ k))$

by $(\text{simp add: summable_sums_iff_divide_inverse_sums_def})$

from $\text{filterlim_compose}[OF \text{this filterlim_subseq}[of (*) (2::\text{nat})]]$

have $(\lambda n. \sum k < 2*n. (-1)^k / \text{real_of_nat } (Suc\ k)) \longrightarrow (\sum k. (-1)^k / \text{real_of_nat } (Suc\ k))$

by $(\text{simp add: strict_mono_def})$

ultimately have $(\sum k. (-1)^k / \text{real_of_nat } (Suc\ k)) = \ln 2$ **by** $(\text{intro LIMSEQ_unique})$

with A **show** $?thesis$ **by** $(\text{simp add: sums_def})$

qed

lemma $\text{alternating_harmonic_series_sums'}$:

$(\lambda k. \text{inverse } (\text{real_of_nat } (2*k+1)) - \text{inverse } (\text{real_of_nat } (2*k+2))) \text{sums } \ln 2$

unfolding sums_def

proof $(\text{rule Lim_transform_eventually})$

show $(\lambda n. \sum k < 2*n. (-1)^k / (\text{real_of_nat } (Suc\ k))) \longrightarrow \ln 2$

using $\text{alternating_harmonic_series_sums_unfolding sums_def}$

by $(\text{rule filterlim_compose}) (\text{rule mult_nat_left_at_top, simp})$

show $\text{eventually } (\lambda n. (\sum k < 2*n. (-1)^k / (\text{real_of_nat } (Suc\ k))) = (\sum k < n. \text{inverse } (\text{real_of_nat } (2*k+1)) - \text{inverse } (\text{real_of_nat } (2*k+2)))) \text{sequentially}$

proof $(\text{intro always_eventually_allI})$

fix $n :: \text{nat}$

show $(\sum k < 2*n. (-1)^k / (\text{real_of_nat } (Suc\ k))) = (\sum k < n. \text{inverse } (\text{real_of_nat } (2*k+1)) - \text{inverse } (\text{real_of_nat } (2*k+2)))$

by $(\text{induction } n) (\text{simp_all add: inverse_eq_divide})$

qed

qed

9.9.3 Bounds on the Euler-Mascheroni constant

lemma *ln_inverse_approx_le:*

assumes $(x::real) > 0$ $a > 0$

shows $\ln(x + a) - \ln x \leq a * (\text{inverse } x + \text{inverse } (x + a)) / 2$ (**is** $_ \leq ?A$)

proof –

define f' **where** $f' = (\text{inverse } (x + a) - \text{inverse } x) / a$

let $?f = \lambda t. (t - x) * f' + \text{inverse } x$

let $?F = \lambda t. (t - x)^2 * f' / 2 + t * \text{inverse } x$

have *deriv*: $\exists D. ((\lambda x. ?F x - \ln x)$ *has_field_derivative* $D)$ (*at* ξ) $\wedge D \geq 0$

if $\xi \geq x$ $\xi \leq x + a$ **for** ξ

proof –

from *that assms* **have** $t: 0 \leq (\xi - x) / a$ $(\xi - x) / a \leq 1$ **by** *simp_all*

have *inverse* $\xi = \text{inverse } ((1 - (\xi - x) / a) * x + ((\xi - x) / a) * (x + a))$ (**is** $_ = ?A$)

using *assms* **by** (*simp add: field_simps*)

also from *assms* **have** *convex_on* $\{x..x+a\}$ *inverse* **by** (*intro convex_on_inverse*)

auto

from *convex_onD_Icc[OF this _ t]* *assms*

have $?A \leq (1 - (\xi - x) / a) * \text{inverse } x + (\xi - x) / a * \text{inverse } (x + a)$

by *simp*

also have $\dots = (\xi - x) * f' + \text{inverse } x$ **using** *assms*

by (*simp add: f'_def divide_simps*) (*simp add: field_simps*)

finally have $?f \xi - 1 / \xi \geq 0$ **by** (*simp add: field_simps*)

moreover have $((\lambda x. ?F x - \ln x)$ *has_field_derivative* $?f \xi - 1 / \xi)$ (*at* ξ)

using *that assms* **by** (*auto intro!: derivative_eq_intros simp: field_simps*)

ultimately show *?thesis* **by** *blast*

qed

have $?F x - \ln x \leq ?F (x + a) - \ln (x + a)$

by (*rule DERIV_nonneg_imp_nondecreasing[of x x + a, OF _ deriv]*) (*use assms in auto*)

thus *?thesis*

using *assms* **by** (*simp add: f'_def divide_simps*) (*simp add: algebra_simps power2_eq_square*)?

qed

lemma *ln_inverse_approx_ge:*

assumes $(x::real) > 0$ $x < y$

shows $\ln y - \ln x \geq 2 * (y - x) / (x + y)$ (**is** $_ \geq ?A$)

proof –

define m **where** $m = (x + y) / 2$

define f' **where** $f' = -\text{inverse } (m^2)$

from *assms* **have** $m: m > 0$ **by** (*simp add: m_def*)

let $?F = \lambda t. (t - m)^2 * f' / 2 + t / m$

let $?f = \lambda t. (t - m) * f' + \text{inverse } m$

have *deriv*: $\exists D. ((\lambda x. \ln x - ?F x)$ *has_field_derivative* $D)$ (*at* ξ) $\wedge D \geq 0$

if $\xi \geq x$ $\xi \leq y$ **for** ξ

proof –

from that *assms* **have** $\text{inverse } \xi - \text{inverse } m \geq f' * (\xi - m)$
by (*intro convex_on_imp_above_tangent*[of {0<..}] *convex_on_inverse*)
(auto simp: m_def interior_open f'_def power2_eq_square intro!: derivative_eq_intros)
hence $1 / \xi - ?f \xi \geq 0$ **by** (*simp add: field_simps f'_def*)
moreover have $((\lambda x. \ln x - ?F x)$ *has_field_derivative* $1 / \xi - ?f \xi$) (*at* ξ)
using that *assms* m **by** (*auto intro!: derivative_eq_intros simp: field_simps*)
ultimately show *?thesis* **by** *blast*
qed
have $\ln x - ?F x \leq \ln y - ?F y$
by (*rule DERIV_nonneg_imp_nondecreasing*[of x y , $OF _ deriv$]) (*use assms in auto*)
hence $\ln y - \ln x \geq ?F y - ?F x$
by (*simp add: algebra_simps*)
also have $?F y - ?F x = ?A$
using *assms* **by** (*simp add: f'_def m_def divide_simps*) (*simp add: algebra_simps power2_eq_square*)
finally show *?thesis* .
qed

lemma *euler_mascheroni_lower*:

$\text{euler_mascheroni} \geq \text{harm } (\text{Suc } n) - \ln (\text{real_of_nat } (n + 2)) + 1 / \text{real_of_nat } (2 * (n + 2))$

and *euler_mascheroni_upper*:

$\text{euler_mascheroni} \leq \text{harm } (\text{Suc } n) - \ln (\text{real_of_nat } (n + 2)) + 1 / \text{real_of_nat } (2 * (n + 1))$

proof –

define $D :: _ \Rightarrow \text{real}$

where $D n = \text{inverse } (\text{of_nat } (n+1)) + \ln (\text{of_nat } (n+1)) - \ln (\text{of_nat } (n+2))$ **for** n

let $?g = \lambda n. \ln (\text{of_nat } (n+2)) - \ln (\text{of_nat } (n+1)) - \text{inverse } (\text{of_nat } (n+1))$
 $:: \text{real}$

define *inv* **where** [*abs_def*]: $\text{inv } n = \text{inverse } (\text{real_of_nat } n)$ **for** n

fix $n :: \text{nat}$

note *summable* = *sums_summable*[*OF euler_mascheroni_sum_real, folded D_def*]

have *sums*: $(\lambda k. (\text{inv } (\text{Suc } (k + (n+1)))) - \text{inv } (\text{Suc } (\text{Suc } k + (n+1)))) / 2$ *sums*
 $((\text{inv } (\text{Suc } (0 + (n+1)))) - 0) / 2$

unfolding *inv_def*

by (*intro sums_divide_telescope_sums' LIMSEQ_ignore_initial_segment LIMSEQ_inverse_real_of_nat*)

have *sums'*: $(\lambda k. (\text{inv } (\text{Suc } (k + n))) - \text{inv } (\text{Suc } (\text{Suc } k + n))) / 2$ *sums* $((\text{inv } (\text{Suc } (0 + n))) - 0) / 2$

unfolding *inv_def*

by (*intro sums_divide_telescope_sums' LIMSEQ_ignore_initial_segment LIMSEQ_inverse_real_of_nat*)

from *euler_mascheroni_sum_real* **have** $\text{euler_mascheroni} = (\sum k. D k)$

by (*simp add: sums_iff D_def*)

also have $\dots = (\sum k. D (k + \text{Suc } n)) + (\sum k \leq n. D k)$

by (*subst suminf_split_initial_segment*[*OF summable, of Suc n*],

```

      subst lessThan_Suc_atMost) simp
  finally have sum:  $(\sum k \leq n. D k) - \text{euler\_mascheroni} = -(\sum k. D (k + \text{Suc } n))$ 
  by simp

  note sum
  also have ...  $\leq -(\sum k. (\text{inv } (k + \text{Suc } n + 1) - \text{inv } (k + \text{Suc } n + 2)) / 2)$ 
  proof (intro le_imp_neg_le suminf_le allI summable_ignore_initial_segment[OF
  summable])
    fix k' :: nat
    define k where k = k' + Suc n
    hence k: k > 0 by (simp add: k_def)
    have real_of_nat (k+1) > 0 by (simp add: k_def)
    with ln_inverse_approx_le[OF this zero_less_one]
      have ln (of_nat k + 2) - ln (of_nat k + 1)  $\leq (\text{inv } (k+1) + \text{inv } (k+2)) / 2$ 
      by (simp add: inv_def add_ac)
    hence  $(\text{inv } (k+1) - \text{inv } (k+2)) / 2 \leq \text{inv } (k+1) + \ln (\text{of\_nat } (k+1)) - \ln$ 
    (of_nat (k+2))
      by (simp add: field_simps)
    also have ... = D k unfolding D_def inv_def ..
    finally show D (k' + Suc n)  $\geq (\text{inv } (k' + \text{Suc } n + 1) - \text{inv } (k' + \text{Suc } n +$ 
    2)) / 2
      by (simp add: k_def)
    from sums_summable[OF sums]
      show summable  $(\lambda k. (\text{inv } (k + \text{Suc } n + 1) - \text{inv } (k + \text{Suc } n + 2)) / 2)$  by
  simp
  qed
  also from sums have ... =  $-\text{inv } (n+2) / 2$  by (simp add: sums_iff)
  finally have euler_mascheroni  $\geq (\sum k \leq n. D k) + 1 / (\text{of\_nat } (2 * (n+2)))$ 
    by (simp add: inv_def field_simps)
  also have  $(\sum k \leq n. D k) = \text{harm } (\text{Suc } n) - (\sum k \leq n. \ln (\text{real\_of\_nat } (\text{Suc } k+1))$ 
  -  $\ln (\text{of\_nat } (k+1)))$ 
    unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add:
  sum.distrib sum_subtractf)
  also have  $(\sum k \leq n. \ln (\text{real\_of\_nat } (\text{Suc } k+1)) - \ln (\text{of\_nat } (k+1))) = \ln$ 
  (of_nat (n+2))
    by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
  finally show euler_mascheroni  $\geq \text{harm } (\text{Suc } n) - \ln (\text{real\_of\_nat } (n + 2)) +$ 
  1/real_of_nat (2 * (n + 2))
    by simp

  note sum
  also have  $-(\sum k. D (k + \text{Suc } n)) \geq -(\sum k. (\text{inv } (\text{Suc } (k + n)) - \text{inv } (\text{Suc } (\text{Suc}$ 
  k + n))) / 2)
  proof (intro le_imp_neg_le suminf_le allI summable_ignore_initial_segment[OF
  summable])
    fix k' :: nat
    define k where k = k' + Suc n
    hence k: k > 0 by (simp add: k_def)
    have real_of_nat (k+1) > 0 by (simp add: k_def)

```

```

from ln_inverse_approx_ge[of of_nat k + 1 of_nat k + 2]
  have  $2 / (2 * \text{real\_of\_nat } k + 3) \leq \ln (\text{of\_nat } (k+2)) - \ln (\text{real\_of\_nat } (k+1))$ 
  by (simp add: add_ac)
  hence  $D k \leq 1 / \text{real\_of\_nat } (k+1) - 2 / (2 * \text{real\_of\_nat } k + 3)$ 
  by (simp add: D_def inverse_eq_divide inv_def)
  also have  $\dots = \text{inv } ((k+1)*(2*k+3))$  unfolding inv_def by (simp add: field_simps)
  also have  $\dots \leq \text{inv } (2*k*(k+1))$  unfolding inv_def using k
  by (intro le_imp_inverse_le)
  (simp add: algebra_simps, simp del: of_nat_add)
  also have  $\dots = (\text{inv } k - \text{inv } (k+1))/2$  unfolding inv_def using k
  by (simp add: divide_simps del: of_nat_mult) (simp add: algebra_simps)
  finally show  $D k \leq (\text{inv } (\text{Suc } (k' + n)) - \text{inv } (\text{Suc } (\text{Suc } k' + n)))/2$  unfolding
k_def by simp
  next
  from sums_summable[OF sums']
    show summable  $(\lambda k. (\text{inv } (\text{Suc } (k + n)) - \text{inv } (\text{Suc } (\text{Suc } k + n)))/2)$  by
simp
  qed
  also from sums' have  $(\sum k. (\text{inv } (\text{Suc } (k + n)) - \text{inv } (\text{Suc } (\text{Suc } k + n)))/2)$ 
 $= \text{inv } (n+1)/2$ 
  by (simp add: sums_iff)
  finally have euler_mascheroni  $\leq (\sum k \leq n. D k) + 1 / \text{of\_nat } (2 * (n+1))$ 
  by (simp add: inv_def field_simps)
  also have  $(\sum k \leq n. D k) = \text{harm } (\text{Suc } n) - (\sum k \leq n. \ln (\text{real\_of\_nat } (\text{Suc } k+1)) - \ln (\text{of\_nat } (k+1)))$ 
  unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add: sum.distrib sum_subtractf)
  also have  $(\sum k \leq n. \ln (\text{real\_of\_nat } (\text{Suc } k+1)) - \ln (\text{of\_nat } (k+1))) = \ln (\text{of\_nat } (n+2))$ 
  by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
  finally show euler_mascheroni  $\leq \text{harm } (\text{Suc } n) - \ln (\text{real\_of\_nat } (n + 2)) + 1/\text{real\_of\_nat } (2 * (n + 1))$ 
  by simp
qed

```

```

lemma euler_mascheroni_pos: euler_mascheroni  $> (0::\text{real})$ 
  using euler_mascheroni_lower[of 0] ln_2_less_1 by (simp add: harm_def)

```

```

context
begin

```

```

private lemma ln_approx_aux:
  fixes n :: nat and x :: real
  defines y  $\equiv (x-1)/(x+1)$ 
  assumes x:  $x > 0 \ x \neq 1$ 
  shows inverse  $(2*y^{2*n+1}) * (\ln x - (\sum k < n. 2*y^{2*k+1} / \text{of\_nat } (2*k+1)))$ 
 $\in$ 

```



```

      {0..(1 / (1 - y2) / of_nat (2*n+1))}
proof -
  from x have norm_y: norm y < 1 unfolding y_def by simp
  from power_strict_mono[OF this, of 2] have norm_y': norm y2 < 1 by simp

  let ?f = λk. 2 * y^(2*k+1) / of_nat (2*k+1)
  note sums = ln_series_quadratic[OF x(1)]
  define c where c = inverse (2*y^(2*n+1))
  let ?d = c * (ln x - (∑ k<n. ?f k))
  have ∧k. y2 k / of_nat (2*(k+n)+1) ≤ y2 k / of_nat (2*n+1)
  by (intro divide_left_mono mult_right_mono mult_pos_pos zero_le_power[of
y2]) simp_all
  moreover {
    have (λk. ?f (k + n)) sums (ln x - (∑ k<n. ?f k))
    using sums_split_initial_segment[OF sums] by (simp add: y_def)
    hence (λk. c * ?f (k + n)) sums ?d by (rule sums_mult)
    also have (λk. c * (2*y^(2*(k+n)+1) / of_nat (2*(k+n)+1))) =
      (λk. (c * (2*y^(2*n+1))) * ((y2)k / of_nat (2*(k+n)+1)))
    by (simp only: ring_distrib power_add power_mult) (simp add: mult_ac)
    also from x have c * (2*y^(2*n+1)) = 1 by (simp add: c_def y_def)
    finally have (λk. (y2)k / of_nat (2*(k+n)+1)) sums ?d by simp
  } note sums' = this
  moreover from norm_y' have (λk. (y2)k / of_nat (2*n+1)) sums (1 / (1
- y2) / of_nat (2*n+1))
  by (intro sums_divide geometric_sums) (simp_all add: norm_power)
  ultimately have ?d ≤ (1 / (1 - y2) / of_nat (2*n+1)) by (rule sums_le)
  moreover have c * (ln x - (∑ k<n. 2 * y^(2 * k + 1) / real_of_nat (2 * k
+ 1))) ≥ 0
  by (intro sums_le[OF _ sums_zero sums']) simp_all
  ultimately show ?thesis unfolding c_def by simp
qed

```

lemma

```

  fixes n :: nat and x :: real
  defines y ≡ (x-1)/(x+1)
  defines approx ≡ (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1))
  defines d ≡ y^(2*n+1) / (1 - y2) / of_nat (2*n+1)
  assumes x: x > 1
  shows ln_approx_bounds: ln x ∈ {approx..approx + 2*d}
  and ln_approx_abs: abs (ln x - (approx + d)) ≤ d
proof -
  define c where c = 2*y^(2*n+1)
  from x have c_pos: c > 0 unfolding c_def y_def
  by (intro mult_pos_pos zero_less_power) simp_all
  have A: inverse c * (ln x - (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1))) ∈
    {0.. (1 / (1 - y2) / of_nat (2*n+1))} using assms unfolding
y_def c_def
  by (intro ln_approx_aux) simp_all
  hence inverse c * (ln x - (∑ k<n. 2*y^(2*k+1)/of_nat (2*k+1))) ≤ (1 /

```

```

(1-y^2) / of_nat (2*n+1))
  by simp
  hence (ln x - (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1))) / c ≤ (1 / (1 -
y^2) / of_nat (2*n+1))
  by (auto simp add: field_split_simps)
  with c_pos have ln x ≤ c / (1 - y^2) / of_nat (2*n+1) + approx
  by (subst (asm) pos_divide_le_eq) (simp_all add: mult_ac approx_def)
  moreover {
    from A c_pos have 0 ≤ c * (inverse c * (ln x - (∑ k<n. 2*y^(2*k+1) /
of_nat (2*k+1))))
    by (intro mult_nonneg_nonneg[of c]) simp_all
    also have ... = (c * inverse c) * (ln x - (∑ k<n. 2*y^(2*k+1) / of_nat
(2*k+1)))
    by (simp add: mult_ac)
    also from c_pos have c * inverse c = 1 by simp
    finally have ln x ≥ approx by (simp add: approx_def)
  }
  ultimately show ln x ∈ {approx..approx + 2*d} by (simp add: c_def d_def)
  thus abs (ln x - (approx + d)) ≤ d by auto
qed

end

```

lemma *euler_mascheroni_bounds*:

```

  fixes n :: nat assumes n ≥ 1 defines t ≡ harm n - ln (of_nat (Suc n)) :: real
  shows euler_mascheroni ∈ {t + inverse (of_nat (2*(n+1)))..t + inverse (of_nat
(2*n))}
  using assms euler_mascheroni_upper[of n-1] euler_mascheroni_lower[of n-1]
  unfolding t_def by (cases n) (simp_all add: harm_Suc t_def inverse_eq_divide)

```

lemma *euler_mascheroni_bounds'*:

```

  fixes n :: nat assumes n ≥ 1 ln (real_of_nat (Suc n)) ∈ {l<..}
  shows euler_mascheroni ∈
    {harm n - u + inverse (of_nat (2*(n+1)))<..harm n - l + inverse
(of_nat (2*n))}
  using euler_mascheroni_bounds[OF assms(1)] assms(2) by auto

```

Approximation of $\ln (2::'a)$. The lower bound is accurate to about 0.03; the upper bound is accurate to about 0.0015.

```

lemma ln2_ge_two_thirds: 2/3 ≤ ln (2::real)
  and ln2_le_25_over_36: ln (2::real) ≤ 25/36
  using ln_approx_bounds[of 2 1, simplified, simplified eval_nat_numeral, sim-
plified] by simp_all

```

Approximation of the Euler-Mascheroni constant. The lower bound is accurate to about 0.0015; the upper bound is accurate to about 0.015.

```

lemma euler_mascheroni_gt_19_over_33: (euler_mascheroni :: real) > 19/33
(is ?th1)

```

```

  and euler_mascheroni_less_13_over_22: (euler_mascheroni :: real) < 13/22
(is ?th2)
proof -
  have ln (real (Suc 7)) = 3 * ln 2 by (simp add: ln_powr [symmetric])
  also from ln_approx_bounds[of 2 3] have ... ∈ {3*307/443 <..< 3*4615/6658}
  by (simp add: eval_nat_numeral)
  finally have ln (real (Suc 7)) ∈ ... .
  from euler_mascheroni_bounds'[OF _ this] have ?th1 ∧ ?th2 by (simp_all
add: harm_expand)
  thus ?th1 ?th2 by blast+
qed

end

```

9.10 The Gamma Function

```

theory Gamma_Function
imports
  Equivalence_Lebesgue_Henstock_Integration
  Summation_Tests
  Harmonic_Numbers
  HOL-Library.Nonpos_Ints
  HOL-Library.Periodic_Fun
begin

```

Several equivalent definitions of the Gamma function and its most important properties. Also contains the definition and some properties of the log-Gamma function and the Digamma function and the other Polygamma functions.

Based on the Gamma function, we also prove the Weierstraß product form of the sin function and, based on this, the solution of the Basel problem (the sum over all $1 / \text{real } (n^2)$).

```

lemma pochhammer_eq_0_imp_nonpos_Int:
  pochhammer (x::'a::field_char_0) n = 0 ⇒ x ∈ ℤ≤0
  by (auto simp: pochhammer_eq_0_iff)

```

```

lemma closed_nonpos_Ints [simp]: closed (ℤ≤0 :: 'a :: real_normed_algebra_1
set)

```

```

proof -
  have ℤ≤0 = (of_int ' {n. n ≤ 0} :: 'a set)
  by (auto elim!: nonpos_Ints_cases intro!: nonpos_Ints_of_int)
  also have closed ... by (rule closed_of_int_image)
  finally show ?thesis .
qed

```

```

lemma plus_one_in_nonpos_Ints_imp: z + 1 ∈ ℤ≤0 ⇒ z ∈ ℤ≤0
  using nonpos_Ints_diff_Nats[of z+1 1] by simp_all

```

lemma *of_int_in_nonpos_Ints_iff*:
 $(\text{of_int } n :: 'a :: \text{ring_char_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n \leq 0$
by (*auto simp: nonpos_Ints_def*)

lemma *one_plus_of_int_in_nonpos_Ints_iff*:
 $(1 + \text{of_int } n :: 'a :: \text{ring_char_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n \leq -1$
proof –
have $1 + \text{of_int } n = (\text{of_int } (n + 1) :: 'a)$ **by** *simp*
also have $\dots \in \mathbb{Z}_{\leq 0} \longleftrightarrow n + 1 \leq 0$ **by** (*subst of_int_in_nonpos_Ints_iff*)
simp_all
also have $\dots \longleftrightarrow n \leq -1$ **by** *presburger*
finally show *?thesis* .
qed

lemma *one_minus_of_nat_in_nonpos_Ints_iff*:
 $(1 - \text{of_nat } n :: 'a :: \text{ring_char_0}) \in \mathbb{Z}_{\leq 0} \longleftrightarrow n > 0$
proof –
have $(1 - \text{of_nat } n :: 'a) = \text{of_int } (1 - \text{int } n)$ **by** *simp*
also have $\dots \in \mathbb{Z}_{\leq 0} \longleftrightarrow n > 0$ **by** (*subst of_int_in_nonpos_Ints_iff*) *presburger*
finally show *?thesis* .
qed

lemma *fraction_not_in_ints*:
assumes $\neg(n \text{ dvd } m) \ n \neq 0$
shows $\text{of_int } m / \text{of_int } n \notin (\mathbb{Z} :: 'a :: \{\text{division_ring, ring_char_0}\} \text{ set})$
proof
assume $\text{of_int } m / (\text{of_int } n :: 'a) \in \mathbb{Z}$
then obtain *k* **where** $\text{of_int } m / \text{of_int } n = (\text{of_int } k :: 'a)$ **by** (*elim Ints_cases*)
with assms have $\text{of_int } m = (\text{of_int } (k * n) :: 'a)$ **by** (*auto simp add: field_split_simps*)
hence $m = k * n$ **by** (*subst (asm) of_int_eq_iff*)
hence $n \text{ dvd } m$ **by** *simp*
with assms(1) show *False* **by** *contradiction*
qed

lemma *fraction_not_in_nats*:
assumes $\neg n \text{ dvd } m \ n \neq 0$
shows $\text{of_int } m / \text{of_int } n \notin (\mathbb{N} :: 'a :: \{\text{division_ring, ring_char_0}\} \text{ set})$
proof
assume $\text{of_int } m / \text{of_int } n \in (\mathbb{N} :: 'a \text{ set})$
also note *Nats_subset_Ints*
finally have $\text{of_int } m / \text{of_int } n \in (\mathbb{Z} :: 'a \text{ set})$.
moreover have $\text{of_int } m / \text{of_int } n \notin (\mathbb{Z} :: 'a \text{ set})$
using assms by (*intro fraction_not_in_ints*)
ultimately show *False* **by** *contradiction*
qed

lemma *not_in_Ints_imp_not_in_nonpos_Ints*: $z \notin \mathbb{Z} \implies z \notin \mathbb{Z}_{\leq 0}$
by (*auto simp: Ints_def nonpos_Ints_def*)

lemma *double_in_nonpos_Ints_imp*:
assumes $2 * (z :: 'a :: \text{field_char_0}) \in \mathbb{Z}_{\leq 0}$
shows $z \in \mathbb{Z}_{\leq 0} \vee z + 1/2 \in \mathbb{Z}_{\leq 0}$
proof –
from *assms* **obtain** k **where** $k: 2 * z = - \text{of_nat } k$ **by** (*elim nonpos_Ints_cases'*)
thus *?thesis* **by** (*cases even k*) (*auto elim!: evenE oddE simp: field_simps*)
qed

lemma *sin_series*: $(\lambda n. ((-1)^{\hat{n}} / \text{fact } (2*n+1)) *_{\mathbb{R}} z^{\wedge}(2*n+1)) \text{ sums } \sin z$
proof –
from *sin_converges*[*of z*] **have** $(\lambda n. \text{sin_coeff } n *_{\mathbb{R}} z^{\wedge}n) \text{ sums } \sin z$.
also have $(\lambda n. \text{sin_coeff } n *_{\mathbb{R}} z^{\wedge}n) \text{ sums } \sin z \longleftrightarrow$
 $(\lambda n. ((-1)^{\hat{n}} / \text{fact } (2*n+1)) *_{\mathbb{R}} z^{\wedge}(2*n+1)) \text{ sums } \sin z$
by (*subst sums_mono_reindex*[*of* $\lambda n. 2*n+1$, *symmetric*])
(*auto simp: sin_coeff_def strict_mono_def ac_simps elim!: oddE*)
finally show *?thesis* .
qed

lemma *cos_series*: $(\lambda n. ((-1)^{\hat{n}} / \text{fact } (2*n)) *_{\mathbb{R}} z^{\wedge}(2*n)) \text{ sums } \cos z$
proof –
from *cos_converges*[*of z*] **have** $(\lambda n. \text{cos_coeff } n *_{\mathbb{R}} z^{\wedge}n) \text{ sums } \cos z$.
also have $(\lambda n. \text{cos_coeff } n *_{\mathbb{R}} z^{\wedge}n) \text{ sums } \cos z \longleftrightarrow$
 $(\lambda n. ((-1)^{\hat{n}} / \text{fact } (2*n)) *_{\mathbb{R}} z^{\wedge}(2*n)) \text{ sums } \cos z$
by (*subst sums_mono_reindex*[*of* $\lambda n. 2*n$, *symmetric*])
(*auto simp: cos_coeff_def strict_mono_def ac_simps elim!: evenE*)
finally show *?thesis* .
qed

lemma *sin_z_over_z_series*:
fixes $z :: 'a :: \{\text{real_normed_field}, \text{banach}\}$
assumes $z \neq 0$
shows $(\lambda n. (-1)^{\hat{n}} / \text{fact } (2*n+1) * z^{\wedge}(2*n)) \text{ sums } (\sin z / z)$
proof –
from *sin_series*[*of z*] **have** $(\lambda n. z * ((-1)^{\hat{n}} / \text{fact } (2*n+1)) * z^{\wedge}(2*n)) \text{ sums } \sin z$
by (*simp add: field_simps scaleR_conv_of_real*)
from *sums_mult*[*OF this, of inverse z*] **and** *assms* **show** *?thesis*
by (*simp add: field_simps*)
qed

lemma *sin_z_over_z_series'*:
fixes $z :: 'a :: \{\text{real_normed_field}, \text{banach}\}$
assumes $z \neq 0$
shows $(\lambda n. \text{sin_coeff } (n+1) *_{\mathbb{R}} z^{\wedge}n) \text{ sums } (\sin z / z)$
proof –
from *sums_split_initial_segment*[*OF sin_converges*[*of z*], *of 1*]
have $(\lambda n. z * (\text{sin_coeff } (n+1) *_{\mathbb{R}} z^{\wedge}n)) \text{ sums } \sin z$ **by** *simp*

from *sums_mult*[*OF this, of inverse z*] *assms* **show** *?thesis* **by** (*simp add: field_simps*)
qed

lemma *has_field_derivative_sin_z_over_z*:

fixes $A :: 'a :: \{\text{real_normed_field}, \text{banach}\}$ *set*

shows $((\lambda z. \text{if } z = 0 \text{ then } 1 \text{ else } \sin z / z) \text{ has_field_derivative } 0)$ (at 0 within A)

(**is** $(?f \text{ has_field_derivative } ?f')$ $_)$

proof (*rule has_field_derivative_at_within*)

have $((\lambda z :: 'a. \sum n. \text{of_real } (\sin_coeff (n+1)) * z^{\wedge} n)$

$\text{has_field_derivative } (\sum n. \text{diffs } (\lambda n. \text{of_real } (\sin_coeff (n+1)))) n * 0^{\wedge} n)$ (at 0)

proof (*rule termdiffs_strong*)

from *summable_ignore_initial_segment*[*OF sums_summable*[*OF sin_converges*[*of 1 :: 'a*]], *of 1*]

show *summable* $(\lambda n. \text{of_real } (\sin_coeff (n+1)) * (1 :: 'a)^{\wedge} n)$ **by** (*simp add: of_real_def*)

qed *simp*

also have $(\lambda z :: 'a. \sum n. \text{of_real } (\sin_coeff (n+1)) * z^{\wedge} n) = ?f$

proof

fix z

show $(\sum n. \text{of_real } (\sin_coeff (n+1)) * z^{\wedge} n) = ?f z$

by (*cases z = 0*) (*insert sin_z_over_z_series'*[*of z*],

simp_all add: scaleR_conv_of_real sums_iff sin_coeff_def)

qed

also have $(\sum n. \text{diffs } (\lambda n. \text{of_real } (\sin_coeff (n + 1)))) n * (0 :: 'a)^{\wedge} n = \text{diffs } (\lambda n. \text{of_real } (\sin_coeff (Suc n))) 0$ **by** *simp*

also have $\dots = 0$ **by** (*simp add: sin_coeff_def diffs_def*)

finally show $((\lambda z :: 'a. \text{if } z = 0 \text{ then } 1 \text{ else } \sin z / z) \text{ has_field_derivative } 0)$ (at 0).

qed

lemma *round_Re_minimises_norm*:

norm $((z :: \text{complex}) - \text{of_int } m) \geq \text{norm } (z - \text{of_int } (\text{round } (\text{Re } z)))$

proof $_$

let $?n = \text{round } (\text{Re } z)$

have *norm* $(z - \text{of_int } ?n) = \text{sqrt } ((\text{Re } z - \text{of_int } ?n)^2 + (\text{Im } z)^2)$

by (*simp add: cmod_def*)

also have $|\text{Re } z - \text{of_int } ?n| \leq |\text{Re } z - \text{of_int } m|$ **by** (*rule round_diff_minimal*)

hence *sqrt* $((\text{Re } z - \text{of_int } ?n)^2 + (\text{Im } z)^2) \leq \text{sqrt } ((\text{Re } z - \text{of_int } m)^2 + (\text{Im } z)^2)$

by (*intro real_sqrt_le_mono add_mono*) (*simp_all add: abs_le_square_iff*)

also have $\dots = \text{norm } (z - \text{of_int } m)$ **by** (*simp add: cmod_def*)

finally show *?thesis* .

qed

lemma *Re_pos_in_ball*:

assumes $\text{Re } z > 0$ $t \in \text{ball } z (\text{Re } z / 2)$

```

  shows  $Re\ t > 0$ 
proof -
  have  $Re\ (z - t) \leq norm\ (z - t)$  by (rule complex_Re_le_cmod)
  also from assms have  $\dots < Re\ z / 2$  by (simp add: dist_complex_def)
  finally show  $Re\ t > 0$  using assms by simp
qed

```

```

lemma no_nonpos_Int_in_ball_complex:
  assumes  $Re\ z > 0\ t \in ball\ z\ (Re\ z / 2)$ 
  shows  $t \notin \mathbb{Z}_{\leq 0}$ 
  using Re_pos_in_ball[OF assms] by (force elim!: nonpos_Ints_cases)

```

```

lemma no_nonpos_Int_in_ball:
  assumes  $t \in ball\ z\ (dist\ z\ (round\ (Re\ z)))$ 
  shows  $t \notin \mathbb{Z}_{\leq 0}$ 
proof
  assume  $t \in \mathbb{Z}_{\leq 0}$ 
  then obtain  $n$  where  $t = of\_int\ n$  by (auto elim!: nonpos_Ints_cases)
  have  $dist\ z\ (of\_int\ n) \leq dist\ z\ t + dist\ t\ (of\_int\ n)$  by (rule dist_triangle)
  also from assms have  $dist\ z\ t < dist\ z\ (round\ (Re\ z))$  by simp
  also have  $\dots \leq dist\ z\ (of\_int\ n)$ 
  using round_Re_minimises_norm[of  $z$ ] by (simp add: dist_complex_def)
  finally have  $dist\ t\ (of\_int\ n) > 0$  by simp
  with  $\langle t = of\_int\ n \rangle$  show False by simp
qed

```

```

lemma no_nonpos_Int_in_ball':
  assumes  $(z :: 'a :: \{euclidean\_space, real\_normed\_algebra\_1\}) \notin \mathbb{Z}_{\leq 0}$ 
  obtains  $d$  where  $d > 0 \wedge t \in ball\ z\ d \implies t \notin \mathbb{Z}_{\leq 0}$ 
proof (rule that)
  from assms show  $setdist\ \{z\}\ \mathbb{Z}_{\leq 0} > 0$  by (subst setdist_gt_0_compact_closed)
auto
next
  fix  $t$  assume  $t \in ball\ z\ (setdist\ \{z\}\ \mathbb{Z}_{\leq 0})$ 
  thus  $t \notin \mathbb{Z}_{\leq 0}$  using  $setdist\_le\_dist$ [of  $z\ \{z\}\ t\ \mathbb{Z}_{\leq 0}$ ] by force
qed

```

```

lemma no_nonpos_Real_in_ball:
  assumes  $z: z \notin \mathbb{R}_{\leq 0}$  and  $t: t \in ball\ z\ (if\ Im\ z = 0\ then\ Re\ z / 2\ else\ abs\ (Im\ z) / 2)$ 
  shows  $t \notin \mathbb{R}_{\leq 0}$ 
using  $z$ 
proof (cases  $Im\ z = 0$ )
  assume  $A: Im\ z = 0$ 
  with  $z$  have  $Re\ z > 0$  by (force simp add: complex_nonpos_Reals_iff)
  with  $t\ A$  Re_pos_in_ball[of  $z\ t$ ] show ?thesis by (force simp add: complex_nonpos_Reals_iff)
next
  assume  $A: Im\ z \neq 0$ 
  have  $abs\ (Im\ z) - abs\ (Im\ t) \leq abs\ (Im\ z - Im\ t)$  by linarith

```

also have $\dots = \text{abs } (\text{Im } (z - t))$ **by** *simp*
also have $\dots \leq \text{norm } (z - t)$ **by** (*rule abs_Im_le_cmod*)
also from $A \ t$ **have** $\dots \leq \text{abs } (\text{Im } z) / 2$ **by** (*simp add: dist_complex_def*)
finally have $\text{abs } (\text{Im } t) > 0$ **using** A **by** *simp*
thus *?thesis* **by** (*force simp add: complex_nonpos_Reals_iff*)
qed

9.10.1 The Euler form and the logarithmic Gamma function

We define the Gamma function by first defining its multiplicative inverse *rGamma*. This is more convenient because *rGamma* is entire, which makes proofs of its properties more convenient because one does not have to watch out for discontinuities. (e.g. *rGamma* fulfils $rGamma \ z = z * rGamma \ (z + 1)$ everywhere, whereas the Γ function does not fulfil the analogous equation on the non-positive integers)

We define the Γ function (resp. its reciprocal) in the Euler form. This form has the advantage that it is a relatively simple limit that converges everywhere. The limit at the poles is 0 (due to division by 0). The functional equation $\Gamma(z + 1) = z * \Gamma \ z$ follows immediately from the definition.

definition *Gamma_series* :: ('a :: {banach,real_normed_field}) \Rightarrow nat \Rightarrow 'a **where**
Gamma_series $z \ n = \text{fact } n * \exp (z * \text{of_real } (\ln (\text{of_nat } n))) / \text{pochhammer } z \ (n+1)$

definition *Gamma_series'* :: ('a :: {banach,real_normed_field}) \Rightarrow nat \Rightarrow 'a **where**
Gamma_series' $z \ n = \text{fact } (n - 1) * \exp (z * \text{of_real } (\ln (\text{of_nat } n))) / \text{pochhammer } z \ n$

definition *rGamma_series* :: ('a :: {banach,real_normed_field}) \Rightarrow nat \Rightarrow 'a **where**
rGamma_series $z \ n = \text{pochhammer } z \ (n+1) / (\text{fact } n * \exp (z * \text{of_real } (\ln (\text{of_nat } n))))$

lemma *Gamma_series_altdef*: *Gamma_series* $z \ n = \text{inverse } (rGamma_series \ z \ n)$
and *rGamma_series_altdef*: *rGamma_series* $z \ n = \text{inverse } (Gamma_series \ z \ n)$
unfolding *Gamma_series_def* *rGamma_series_def* **by** *simp_all*

lemma *rGamma_series_minus_of_nat*:
eventually ($\lambda n. rGamma_series \ (- \ \text{of_nat } k) \ n = 0$) *sequentially*
using *eventually_ge_at_top*[*of k*]
by *eventually_elim* (*auto simp: rGamma_series_def pochhammer_of_nat_eq_0_iff*)

lemma *Gamma_series_minus_of_nat*:
eventually ($\lambda n. Gamma_series \ (- \ \text{of_nat } k) \ n = 0$) *sequentially*


```

using eventually_ge_at_top[of k]
by eventually_elim (auto simp: Gamma_series_def pochhammer_of_nat_eq_0_iff)

lemma Gamma_series'_minus_of_nat:
  eventually ( $\lambda n.$  Gamma_series' (- of_nat k) n = 0) sequentially
using eventually_gt_at_top[of k]
by eventually_elim (auto simp: Gamma_series'_def pochhammer_of_nat_eq_0_iff)

lemma rGamma_series_nonpos_Ints_LIMSEQ:  $z \in \mathbf{Z}_{\leq 0} \implies rGamma\_series\ z$ 
 $\longrightarrow 0$ 
by (elim nonpos_Ints_cases', hypsubst, subst tendsto_cong, rule rGamma_series_minus_of_nat,
simp)

lemma Gamma_series_nonpos_Ints_LIMSEQ:  $z \in \mathbf{Z}_{\leq 0} \implies Gamma\_series\ z$ 
 $\longrightarrow 0$ 
by (elim nonpos_Ints_cases', hypsubst, subst tendsto_cong, rule Gamma_series_minus_of_nat,
simp)

lemma Gamma_series'_nonpos_Ints_LIMSEQ:  $z \in \mathbf{Z}_{\leq 0} \implies Gamma\_series'\ z$ 
 $\longrightarrow 0$ 
by (elim nonpos_Ints_cases', hypsubst, subst tendsto_cong, rule Gamma_series'_minus_of_nat,
simp)

lemma Gamma_series_Gamma_series':
  assumes  $z: z \notin \mathbf{Z}_{\leq 0}$ 
  shows ( $\lambda n.$  Gamma_series' z n / Gamma_series z n)  $\longrightarrow 1$ 
proof (rule Lim_transform_eventually)
  from eventually_gt_at_top[of 0::nat]
  show eventually ( $\lambda n.$  z / of_nat n + 1 = Gamma_series' z n / Gamma_series
z n) sequentially
  proof eventually_elim
    fix n :: nat assume n: n > 0
    from n z have Gamma_series' z n / Gamma_series z n = (z + of_nat n) /
of_nat n
    by (cases n, simp)
    (auto simp add: Gamma_series_def Gamma_series'_def pochhammer_rec'
dest: pochhammer_eq_0_imp_nonpos_Int plus_of_nat_eq_0_imp)
    also from n have ... = z / of_nat n + 1 by (simp add: field_split_simps)
    finally show z / of_nat n + 1 = Gamma_series' z n / Gamma_series z n ..
  qed
have ( $\lambda x.$  z / of_nat x)  $\longrightarrow 0$ 
by (rule tendsto_norm_zero_cancel)
  (insert tendsto_mult[OF tendsto_const[of norm z] lim_inverse_n],
simp add: norm_divide_inverse_eq_divide)
from tendsto_add[OF this tendsto_const[of 1]] show ( $\lambda n.$  z / of_nat n + 1)
 $\longrightarrow 1$  by simp
qed

```

We now show that the series that defines the Γ function in the Euler form

converges and that the function defined by it is continuous on the complex halfspace with positive real part.

We do this by showing that the logarithm of the Euler series is continuous and converges locally uniformly, which means that the log-Gamma function defined by its limit is also continuous.

This will later allow us to lift holomorphicity and continuity from the log-Gamma function to the inverse of the Gamma function, and from that to the Gamma function itself.

definition $\text{ln_Gamma_series} :: ('a :: \{\text{banach, real_normed_field, ln}\}) \Rightarrow \text{nat} \Rightarrow 'a$ **where**

$\text{ln_Gamma_series } z \ n = z * \text{ln } (\text{of_nat } n) - \text{ln } z - (\sum k=1..n. \text{ln } (z / \text{of_nat } k + 1))$

definition $\text{ln_Gamma_series}' :: ('a :: \{\text{banach, real_normed_field, ln}\}) \Rightarrow \text{nat} \Rightarrow 'a$ **where**

$\text{ln_Gamma_series}' z \ n =$
 $- \text{euler_mascheroni} * z - \text{ln } z + (\sum k=1..n. z / \text{of_nat } n - \text{ln } (z / \text{of_nat } k + 1))$

definition $\text{ln_Gamma} :: ('a :: \{\text{banach, real_normed_field, ln}\}) \Rightarrow 'a$ **where**

$\text{ln_Gamma } z = \text{lim } (\text{ln_Gamma_series } z)$

We now show that the log-Gamma series converges locally uniformly for all complex numbers except the non-positive integers. We do this by proving that the series is locally Cauchy.

context

begin

private lemma $\text{ln_Gamma_series_complex_converges_aux}:$

fixes $z :: \text{complex}$ **and** $k :: \text{nat}$

assumes $z: z \neq 0$ **and** $k: \text{of_nat } k \geq 2 * \text{norm } z$ $k \geq 2$

shows $\text{norm } (z * \text{ln } (1 - 1/\text{of_nat } k) + \text{ln } (z/\text{of_nat } k + 1)) \leq 2 * (\text{norm } z + \text{norm } z^2) / \text{of_nat } k^2$

proof -

let $?k = \text{of_nat } k :: \text{complex}$ **and** $?z = \text{norm } z$

have $z * \text{ln } (1 - 1/?k) + \text{ln } (z/?k + 1) = z * (\text{ln } (1 - 1/?k :: \text{complex}) + 1/?k)$
 $+ (\text{ln } (1 + z/?k) - z/?k)$

by (*simp add: algebra_simps*)

also have $\text{norm } \dots \leq ?z * \text{norm } (\text{ln } (1 - 1/?k) + 1/?k :: \text{complex}) + \text{norm } (\text{ln } (1 + z/?k) - z/?k)$

by (*subst norm_mult [symmetric], rule norm_triangle_ineq*)

also have $\text{norm } (\text{Ln } (1 - 1/?k) - (-1/?k)) \leq (\text{norm } (-1/?k))^2 / (1 - \text{norm } (-1/?k))$

using k **by** (*intro Ln_approx_linear*) (*simp add: norm_divide*)

hence $?z * \text{norm } (\text{ln } (1 - 1/?k) + 1/?k) \leq ?z * ((\text{norm } (1/?k))^2 / (1 - \text{norm } (1/?k)))$

by (*intro mult_left_mono*) *simp_all*

```

also have ... ≤ (?z * (of_nat k / (of_nat k - 1))) / of_nat k^2 using k
  by (simp add: field_simps power2_eq_square norm_divide)
also have ... ≤ (?z * 2) / of_nat k^2 using k
  by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
also have norm (ln (1+z/?k) - z/?k) ≤ norm (z/?k)^2 / (1 - norm (z/?k))
using k
  by (intro Ln_approx_linear) (simp add: norm_divide)
hence norm (ln (1+z/?k) - z/?k) ≤ ?z^2 / of_nat k^2 / (1 - ?z / of_nat k)
  by (simp add: field_simps norm_divide)
also have ... ≤ (?z^2 * (of_nat k / (of_nat k - ?z))) / of_nat k^2 using k
  by (simp add: field_simps power2_eq_square)
also have ... ≤ (?z^2 * 2) / of_nat k^2 using k
  by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
also note add_divide_distrib [symmetric]
finally show ?thesis by (simp only: distrib_left mult.commute)
qed

```

lemma *ln_Gamma_series_complex_converges*:

```

assumes z: z ∉ ℤ≤0
assumes d: d > 0 ∧ n. n ∈ ℤ≤0 ⇒ norm (z - of_int n) > d
shows uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z n :: complex)
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI')
  fix e :: real assume e: e > 0
  define e'' where e'' = (SUP t∈ball z d. norm t + norm t^2)
  define e' where e' = e / (2*e'')
  have bounded ((λt. norm t + norm t^2) ' cball z d)
    by (intro compact_imp_bounded compact_continuous_image) (auto intro!: continuous_intros)
  hence bounded ((λt. norm t + norm t^2) ' ball z d) by (rule bounded_subset)
  auto
  hence bdd: bdd_above ((λt. norm t + norm t^2) ' ball z d) by (rule bounded_imp_bdd_above)

  with z d(1) d(2)[of -1] have e''_pos: e'' > 0 unfolding e''_def
    by (subst less_cSUP_iff) (auto intro!: add_pos_nonneg bexI[of _ z])
  have e'': norm t + norm t^2 ≤ e'' if t ∈ ball z d for t unfolding e''_def using
  that
    by (rule cSUP_upper[OF _ bdd])
  from e z e''_pos have e': e' > 0 unfolding e'_def
    by (intro divide_pos_pos mult_pos_pos add_pos_pos) (simp_all add: field_simps)

  have summable (λk. inverse ((real_of_nat k)^2))
    by (rule inverse_power_summable) simp
  from summable_partial_sum_bound[OF this e']
  obtain M where M: ∧m n. M ≤ m ⇒ norm (∑ k = m..n. inverse ((real k)^2))
  < e'
    by auto

  define N where N = max 2 (max (nat ⌈2 * (norm z + d)⌉) M)

```

```

{
  from d have  $\lceil 2 * (cmod z + d) \rceil \geq \lceil 0::real \rceil$ 
    by (intro ceiling_mono mult_nonneg_nonneg add_nonneg_nonneg) simp_all
  hence  $2 * (norm z + d) \leq of\_nat (nat \lceil 2 * (norm z + d) \rceil)$  unfolding N_def
    by (simp_all)
  also have  $\dots \leq of\_nat N$  unfolding N_def
    by (subst of_nat_le_iff) (rule max.coboundedI2, rule max.cobounded1)
  finally have  $of\_nat N \geq 2 * (norm z + d)$  .
  moreover have  $N \geq 2 N \geq M$  unfolding N_def by simp_all
  moreover have  $(\sum_{k=m..n}. 1/(of\_nat k)^2) < e'$  if  $m \geq N$  for  $m n$ 
    using M[OF order.trans[OF  $\langle N \geq M \rangle$  that]] unfolding real_norm_def
    by (subst (asm) abs_of_nonneg) (auto intro: sum_nonneg simp: field_split_simps)
  moreover note calculation
} note N = this

show  $\exists M. \forall t \in ball z d. \forall m \geq M. \forall n > m. dist (ln\_Gamma\_series t m) (ln\_Gamma\_series t n) < e$ 
  unfolding dist_complex_def
  proof (intro exI[of _ N] ballI allI impI)
    fix t m n assume t:  $t \in ball z d$  and mn:  $m \geq N n > m$ 
    from d(2)[of 0] t have  $0 < dist z 0 - dist z t$  by (simp add: field_simps dist_complex_def)
    also have  $dist z 0 - dist z t \leq dist 0 t$  using dist_triangle[of 0 z t]
      by (simp add: dist_commute)
    finally have t_nz:  $t \neq 0$  by auto

    have  $norm t \leq norm z + norm (t - z)$  by (rule norm_triangle_sub)
    also from t have  $norm (t - z) < d$  by (simp add: dist_complex_def norm_minus_commute)
    also have  $2 * (norm z + d) \leq of\_nat N$  by (rule N)
    also have  $N \leq m$  by (rule mn)
    finally have norm_t:  $2 * norm t < of\_nat m$  by simp

    have  $ln\_Gamma\_series t m - ln\_Gamma\_series t n =$ 
       $(-(t * Ln (of\_nat n)) - (-(t * Ln (of\_nat m)))) +$ 
       $((\sum_{k=1..n}. Ln (t / of\_nat k + 1)) - (\sum_{k=1..m}. Ln (t / of\_nat k + 1)))$ 
    by (simp add: ln_Gamma_series_def algebra_simps)
    also have  $(\sum_{k=1..n}. Ln (t / of\_nat k + 1)) - (\sum_{k=1..m}. Ln (t / of\_nat k + 1)) =$ 
       $(\sum_{k \in \{1..n\} - \{1..m\}}. Ln (t / of\_nat k + 1))$  using mn
    by (simp add: sum_diff)
    also from mn have  $\{1..n\} - \{1..m\} = \{Suc m..n\}$  by fastforce
    also have  $-(t * Ln (of\_nat n)) - (-(t * Ln (of\_nat m))) =$ 
       $(\sum_{k = Suc m..n}. t * Ln (of\_nat (k - 1)) - t * Ln (of\_nat k))$ 
  using mn
  by (subst sum_telescope'' [symmetric]) simp_all
  also have  $\dots = (\sum_{k = Suc m..n}. t * Ln (of\_nat (k - 1) / of\_nat k))$  using mn N
  by (intro sum_cong_Suc)

```

```

    (simp_all del: of_nat_Suc add: field_simps Ln_of_nat Ln_of_nat_over_of_nat)
  also have of_nat (k - 1) / of_nat k = 1 - 1 / (of_nat k :: complex) if k ∈
  {Suc m..n} for k
    using that of_nat_eq_0_iff[of Suc i for i] by (cases k) (simp_all add:
  field_split_simps)
  hence (∑ k = Suc m..n. t * Ln (of_nat (k - 1) / of_nat k)) =
    (∑ k = Suc m..n. t * Ln (1 - 1 / of_nat k)) using mn N
  by (intro sum.cong) simp_all
  also note sum.distrib [symmetric]
  also have norm (∑ k=Suc m..n. t * Ln (1 - 1/of_nat k) + Ln (t/of_nat k
  + 1)) ≤
    (∑ k=Suc m..n. 2 * (norm t + (norm t)2) / (real_of_nat k)2) using t_nz
  N(2) mn norm_t
  by (intro order.trans[OF norm_sum sum_mono[OF ln_Gamma_series_complex_converges_aux]])
  simp_all
  also have ... ≤ 2 * (norm t + norm t2) * (∑ k=Suc m..n. 1 / (of_nat k)2)
  by (simp add: sum_distrib_left)
  also have ... < 2 * (norm t + norm t2) * e' using mn z t_nz
  by (intro mult_strict_left_mono N mult_pos_pos add_pos_pos) simp_all
  also from e''_pos have ... = e * ((cmod t + (cmod t)2) / e'')
  by (simp add: e'_def field_simps power2_eq_square)
  also from e''[OF t] e''_pos e
  have ... ≤ e * 1 by (intro mult_left_mono) (simp_all add: field_simps)
  finally show norm (ln_Gamma_series t m - ln_Gamma_series t n) < e by
  simp
  qed
  qed
end

lemma ln_Gamma_series_complex_converges':
  assumes z: (z :: complex) ∉ ℤ≤0
  shows ∃ d>0. uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z
  n)
  proof -
    define d' where d' = Re z
    define d where d = (if d' > 0 then d' / 2 else norm (z - of_int (round d')) /
    2)
    have of_int (round d') ∈ ℤ≤0 if d' ≤ 0 using that
    by (intro nonpos_Ints_of_int) (simp_all add: round_def)
    with assms have d_pos: d > 0 unfolding d_def by (force simp: not_less)

    have d < cmod (z - of_int n) if n ∈ ℤ≤0 for n
  proof (cases Re z > 0)
    case True
    from nonpos_Ints_nonpos[OF that] have n: n ≤ 0 by simp
    from True have d = Re z/2 by (simp add: d_def d'_def)
    also from n True have ... < Re (z - of_int n) by simp
    also have ... ≤ norm (z - of_int n) by (rule complex_Re_le_cmod)
  
```

```

    finally show ?thesis .
  next
  case False
  with assms nonpos_Ints_of_int[of round (Re z)]
    have z ≠ of_int (round d') by (auto simp: not_less)
  with False have d < norm (z - of_int (round d')) by (simp add: d_def
d'_def)
  also have ... ≤ norm (z - of_int n) unfolding d'_def by (rule round_Re_minimises_norm)
  finally show ?thesis .
qed
hence conv: uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z n)
  by (intro ln_Gamma_series_complex_converges d_pos z) simp_all
from d_pos conv show ?thesis by blast
qed

```

```

lemma ln_Gamma_series_complex_converges'': (z :: complex) ∉ ℤ≤0 ⇒ con-
vergent (ln_Gamma_series z)
  by (drule ln_Gamma_series_complex_converges') (auto intro: uniformly_convergent_imp_convergent)

```

```

theorem ln_Gamma_complex_LIMSEQ: (z :: complex) ∉ ℤ≤0 ⇒ ln_Gamma_series
z → ln_Gamma z
  using ln_Gamma_series_complex_converges'' by (simp add: convergent_LIMSEQ_iff
ln_Gamma_def)

```

```

lemma exp_ln_Gamma_series_complex:
  assumes n > 0 z ∉ ℤ≤0
  shows exp (ln_Gamma_series z n :: complex) = Gamma_series z n
proof -
  from assms obtain m where m: n = Suc m by (cases n) blast
  from assms have z ≠ 0 by (intro notI) auto
  with assms have exp (ln_Gamma_series z n) =
    (of_nat n) powr z / (z * (∏ k=1..n. exp (Ln (z / of_nat k + 1))))
  unfolding ln_Gamma_series_def powr_def by (simp add: exp_diff exp_sum)
  also from assms have (∏ k=1..n. exp (Ln (z / of_nat k + 1))) = (∏ k=1..n.
z / of_nat k + 1)
  by (intro prod.cong[OF refl], subst exp_Ln) (auto simp: field_simps plus_of_nat_eq_0_imp)
  also have ... = (∏ k=1..n. z + k) / fact n
  by (simp add: fact_prod)
  (subst prod_dividef [symmetric], simp_all add: field_simps)
  also from m have z * ... = (∏ k=0..n. z + k) / fact n
  by (simp add: prod.atLeast0_atMost_Suc_shift prod.atLeast_Suc_atMost_Suc_shift
del: prod.cl_ivl_Suc)
  also have (∏ k=0..n. z + k) = pochhammer z (Suc n)
  unfolding pochhammer_prod
  by (simp add: prod.atLeast0_atMost_Suc atLeastLessThanSuc_atLeastAtMost)
  also have of_nat n powr z / (pochhammer z (Suc n) / fact n) = Gamma_series
z n
  unfolding Gamma_series_def using assms by (simp add: field_split_simps
powr_def)

```

finally show ?thesis .
qed

lemma *ln_Gamma_series'_aux*:

assumes $(z::\text{complex}) \notin \mathbb{Z}_{\leq 0}$

shows $(\lambda k. z / \text{of_nat} (\text{Suc } k) - \ln (1 + z / \text{of_nat} (\text{Suc } k))) \text{ sums}$
 $(\ln_Gamma_z + \text{euler_mascheroni} * z + \ln z)$ (is ?f sums ?s)

unfolding *sums_def*

proof (rule *Lim_transform*)

show $(\lambda n. \ln_Gamma_series\ z\ n + \text{of_real} (\text{harm } n - \ln (\text{of_nat } n)) * z + \ln z) \longrightarrow ?s$

(is ?g \longrightarrow $_$)

by (intro *tendsto_intros ln_Gamma_complex LIMSEQ euler_mascheroni LIMSEQ_of_real assms*)

have *A*: eventually $(\lambda n. (\sum k < n. ?f\ k) - ?g\ n = 0)$ sequentially

using *eventually_gt_at_top[of 0::nat]*

proof *eventually_elim*

fix *n* :: nat assume *n*: $n > 0$

have $(\sum k < n. ?f\ k) = (\sum k=1..n. z / \text{of_nat } k - \ln (1 + z / \text{of_nat } k))$

by (subst *atLeast0LessThan [symmetric]*, *subst sum.shift_bounds_Suc_ivl [symmetric]*,

subst atLeastLessThanSuc_atLeastAtMost) *simp_all*

also have $\dots = z * \text{of_real} (\text{harm } n) - (\sum k=1..n. \ln (1 + z / \text{of_nat } k))$

by (*simp add: harm_def sum_subtractf sum_distrib_left divide_inverse*)

also from *n* have $\dots - ?g\ n = 0$

by (*simp add: ln_Gamma_series_def sum_subtractf algebra_simps*)

finally show $(\sum k < n. ?f\ k) - ?g\ n = 0$.

qed

show $(\lambda n. (\sum k < n. ?f\ k) - ?g\ n) \longrightarrow 0$ by (*subst tendsto_cong[OF A]*)
simp_all

qed

lemma *uniformly_summable_deriv_ln_Gamma*:

assumes *z*: $(z :: 'a :: \{\text{real_normed_field, banach}\}) \neq 0$ and *d*: $d > 0\ d \leq \text{norm } z / 2$

shows *uniformly_convergent_on* (ball *z* *d*)

$(\lambda k\ z. \sum i < k. \text{inverse} (\text{of_nat} (\text{Suc } i)) - \text{inverse} (z + \text{of_nat} (\text{Suc } i)))$

(is *uniformly_convergent_on* $_$ $(\lambda k\ z. \sum i < k. ?f\ i\ z)$)

proof (rule *Weierstrass_m_test'_ev*)

{

fix *t* assume *t*: $t \in \text{ball } z\ d$

have $\text{norm } z = \text{norm} (t + (z - t))$ by *simp*

have $\text{norm} (t + (z - t)) \leq \text{norm } t + \text{norm} (z - t)$ by (rule *norm_triangle_ineq*)

also from *t d* have $\text{norm} (z - t) < \text{norm } z / 2$ by (*simp add: dist_norm*)

finally have *A*: $\text{norm } t > \text{norm } z / 2$ using *z* by (*simp add: field_simps*)

```

    have norm t = norm (z + (t - z)) by simp
    also have ... ≤ norm z + norm (t - z) by (rule norm_triangle_ineq)
    also from t d have norm (t - z) ≤ norm z / 2 by (simp add: dist_norm
norm_minus_commute)
    also from z have ... < norm z by simp
    finally have B: norm t < 2 * norm z by simp
    note A B
  } note ball = this

show eventually (λn. ∀ t ∈ ball z d. norm (?f n t) ≤ 4 * norm z * inverse (of_nat
(Suc n)^2)) sequentially
  using eventually_gt_at_top apply eventually_elim
proof safe
  fix t :: 'a assume t: t ∈ ball z d
  from z ball[OF t] have t_nz: t ≠ 0 by auto
  fix n :: nat assume n: n > nat ⌈4 * norm z⌉
  from ball[OF t] t_nz have 4 * norm z > 2 * norm t by simp
  also from n have ... < of_nat n by linarith
  finally have n: of_nat n > 2 * norm t .
  hence of_nat n > norm t by simp
  hence t': t ≠ -of_nat (Suc n) by (intro notI) (simp del: of_nat_Suc)

  with t_nz have ?f n t = 1 / (of_nat (Suc n) * (1 + of_nat (Suc n)/t))
  by (simp add: field_split_simps eq_neg_iff_add_eq_0 del: of_nat_Suc)
  also have norm ... = inverse (of_nat (Suc n)) * inverse (norm (of_nat (Suc
n)/t + 1))
  by (simp add: norm_divide norm_mult field_split_simps del: of_nat_Suc)
  also {
    from z t_nz ball[OF t] have of_nat (Suc n) / (4 * norm z) ≤ of_nat (Suc
n) / (2 * norm t)
    by (intro divide_left_mono mult_pos_pos) simp_all
    also have ... < norm (of_nat (Suc n) / t) - norm (1 :: 'a)
    using t_nz n by (simp add: field_simps norm_divide del: of_nat_Suc)
    also have ... ≤ norm (of_nat (Suc n)/t + 1) by (rule norm_diff_ineq)
    finally have inverse (norm (of_nat (Suc n)/t + 1)) ≤ 4 * norm z / of_nat
(Suc n)
    using z by (simp add: field_split_simps norm_divide mult_ac del: of_nat_Suc)
  }
  also have inverse (real_of_nat (Suc n)) * (4 * norm z / real_of_nat (Suc
n)) =
    4 * norm z * inverse (of_nat (Suc n)^2)
    by (simp add: field_split_simps power2_eq_square del: of_nat_Suc)
  finally show norm (?f n t) ≤ 4 * norm z * inverse (of_nat (Suc n)^2)
  by (simp del: of_nat_Suc)
qed
next
show summable (λn. 4 * norm z * inverse ((of_nat (Suc n))^2))
  by (subst summable_Suc_iff) (simp add: summable_mult inverse_power_summable)
qed

```


9.10.2 The Polygamma functions

lemma *summable_deriv_ln_Gamma*:

$z \neq 0 \Rightarrow 'a :: \{\text{real_normed_field}, \text{banach}\} \Rightarrow$
 $\text{summable } (\lambda n. \text{inverse } (\text{of_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of_nat } (\text{Suc } n)))$
unfolding *summable_iff_convergent*
by (*rule uniformly_convergent_imp_convergent*,
rule uniformly_summable_deriv_ln_Gamma[*of z norm z/2*]) *simp_all*

definition *Polygamma* :: $\text{nat} \Rightarrow ('a :: \{\text{real_normed_field}, \text{banach}\}) \Rightarrow 'a$ **where**

Polygamma $n z = (\text{if } n = 0 \text{ then}$
 $(\sum k. \text{inverse } (\text{of_nat } (\text{Suc } k)) - \text{inverse } (z + \text{of_nat } k)) - \text{euler_mascheroni}$
else
 $(-1)^{\wedge \text{Suc } n} * \text{fact } n * (\sum k. \text{inverse } ((z + \text{of_nat } k)^{\wedge \text{Suc } n}))$

abbreviation *Digamma* :: $('a :: \{\text{real_normed_field}, \text{banach}\}) \Rightarrow 'a$ **where**

Digamma $\equiv \text{Polygamma } 0$

lemma *Digamma_def*:

Digamma $z = (\sum k. \text{inverse } (\text{of_nat } (\text{Suc } k)) - \text{inverse } (z + \text{of_nat } k)) -$
euler_mascheroni
by (*simp add: Polygamma_def*)

lemma *summable_Digamma*:

assumes $z \neq 0$
shows $\text{summable } (\lambda n. \text{inverse } (\text{of_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of_nat } n))$
proof –
have *sums*: $(\lambda n. \text{inverse } (z + \text{of_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of_nat } n)) \text{ sums}$
 $(0 - \text{inverse } (z + \text{of_nat } 0))$
by (*intro telescope_sums filterlim_compose*[*OF tendsto_inverse_0*]
tendsto_add_filterlim_at_infinity[*OF tendsto_const*] *tendsto_of_nat*)
from *summable_add*[*OF summable_deriv_ln_Gamma*[*OF assms*] *sums_summable*[*OF*
sums]]
show $\text{summable } (\lambda n. \text{inverse } (\text{of_nat } (\text{Suc } n)) - \text{inverse } (z + \text{of_nat } n))$ **by**
simp
qed

lemma *summable_offset*:

assumes $\text{summable } (\lambda n. f (n + k))$
shows $\text{summable } f$
proof –
from *assms* **have** *convergent* $(\lambda m. \sum n < m. f (n + k))$
using *summable_iff_convergent* **by** *blast*
hence *convergent* $(\lambda m. (\sum n < k. f n) + (\sum n < m. f (n + k)))$
by (*intro convergent_add convergent_const*)
also have $(\lambda m. (\sum n < k. f n) + (\sum n < m. f (n + k))) = (\lambda m. \sum n < m+k. f n)$
proof
fix $m :: \text{nat}$
have $\{.. < m+k\} = \{.. < k\} \cup \{k.. < m+k\}$ **by** *auto*

```

also have ( $\sum n \in \dots f n$ ) = ( $\sum n < k. f n$ ) + ( $\sum n = k..< m+k. f n$ )
by (rule sum.union_disjoint) auto
also have ( $\sum n = k..< m+k. f n$ ) = ( $\sum n = 0..< m+k-k. f (n + k)$ )
using sum.shift_bounds_nat_ivl [of f 0 k m] by simp
finally show ( $\sum n < k. f n$ ) + ( $\sum n < m. f (n + k)$ ) = ( $\sum n < m+k. f n$ ) by
(simp add: atLeast0LessThan)
qed
finally have ( $\lambda a. \text{sum } f \{..<a\}$ )  $\longrightarrow$  lim ( $\lambda m. \text{sum } f \{..<m+k\}$ )
by (auto simp: convergent_LIMSEQ_iff dest: LIMSEQ_offset)
thus ?thesis by (auto simp: summable_iff_convergent convergent_def)
qed

```

lemma *Polygamma_converges*:

```

fixes  $z :: 'a :: \{\text{real\_normed\_field}, \text{banach}\}$ 
assumes  $z: z \neq 0$  and  $n: n \geq 2$ 
shows uniformly_convergent_on (ball z d) ( $\lambda k z. \sum i < k. \text{inverse} ((z + \text{of\_nat } i)^{\wedge} n)$ )
proof (rule Weierstrass_m_test'_ev)
define  $e$  where  $e = (1 + d / \text{norm } z)$ 
define  $m$  where  $m = \text{nat } \lceil \text{norm } z * e \rceil$ 
{
fix  $t$  assume  $t: t \in \text{ball } z d$ 
have  $\text{norm } t = \text{norm } (z + (t - z))$  by simp
also have  $\dots \leq \text{norm } z + \text{norm } (t - z)$  by (rule norm_triangle_ineq)
also from  $t$  have  $\text{norm } (t - z) < d$  by (simp add: dist_norm norm_minus_commute)
finally have  $\text{norm } t < \text{norm } z * e$  using  $z$  by (simp add: divide_simps e_def)
} note  $\text{ball} = \text{this}$ 

```

```

show eventually ( $\lambda k. \forall t \in \text{ball } z d. \text{norm} (\text{inverse} ((t + \text{of\_nat } k)^{\wedge} n)) \leq$ 
 $\text{inverse} (\text{of\_nat } (k - m)^{\wedge} n)$ ) sequentially

```

```

using eventually_gt_at_top[of m] apply eventually_elim

```

```

proof (intro ballI)

```

```

fix  $k :: \text{nat}$  and  $t :: 'a$  assume  $k: k > m$  and  $t: t \in \text{ball } z d$ 

```

```

from  $k$  have  $\text{real\_of\_nat } (k - m) = \text{of\_nat } k - \text{of\_nat } m$  by (simp add: of_nat_diff)

```

```

also have  $\dots \leq \text{norm} (\text{of\_nat } k :: 'a) - \text{norm } z * e$ 

```

```

unfolding  $m\_def$  by (subst norm_of_nat) linarith

```

```

also from  $\text{ball}[OF t]$  have  $\dots \leq \text{norm} (\text{of\_nat } k :: 'a) - \text{norm } t$  by simp

```

```

also have  $\dots \leq \text{norm} (\text{of\_nat } k + t)$  by (rule norm_diff_ineq)

```

```

finally have  $\text{inverse} ((\text{norm } (t + \text{of\_nat } k))^{\wedge} n) \leq \text{inverse} (\text{real\_of\_nat } (k - m)^{\wedge} n)$  using  $k n$ 

```

```

by (intro le_imp_inverse_le power_mono) (simp_all add: add_ac del: of_nat_Suc)

```

```

thus  $\text{norm} (\text{inverse} ((t + \text{of\_nat } k)^{\wedge} n)) \leq \text{inverse} (\text{of\_nat } (k - m)^{\wedge} n)$ 

```

```

by (simp add: norm_inverse norm_power power_inverse)

```

```

qed

```

```

have summable ( $\lambda k. \text{inverse} ((\text{real\_of\_nat } k)^{\wedge} n)$ )

```

```

using inverse_power_summable[of n]  $n$  by simp

```

hence *summable* ($\lambda k. \text{inverse } ((\text{real_of_nat } (k + m - m))^{\wedge} n)$) **by** *simp*
thus *summable* ($\lambda k. \text{inverse } ((\text{real_of_nat } (k - m))^{\wedge} n)$) **by** (*rule summable_offset*)
qed

lemma *Polygamma_converges'*:
fixes $z :: 'a :: \{\text{real_normed_field}, \text{banach}\}$
assumes $z: z \neq 0$ **and** $n: n \geq 2$
shows *summable* ($\lambda k. \text{inverse } ((z + \text{of_nat } k)^{\wedge} n)$)
using *uniformly_convergent_imp_convergent*[*OF Polygamma_converges*[*OF assms*,
of 1], *of z*]
by (*simp add: summable_iff_convergent*)

theorem *Digamma_LIMSEQ*:
fixes $z :: 'a :: \{\text{banach}, \text{real_normed_field}\}$
assumes $z: z \neq 0$
shows $(\lambda m. \text{of_real } (\ln (\text{real } m)) - (\sum n < m. \text{inverse } (z + \text{of_nat } n))) \longrightarrow$
Digamma z
proof –
have $(\lambda n. \text{of_real } (\ln (\text{real } n / (\text{real } (\text{Suc } n)))) \longrightarrow (\text{of_real } (\ln 1) :: 'a)$
by (*intro tendsto_intros LIMSEQ_n_over_Suc_n simp_all*)
hence $(\lambda n. \text{of_real } (\ln (\text{real } n / (\text{real } n + 1)))) \longrightarrow (0 :: 'a)$ **by** (*simp add:*
add_ac)
hence *lim*: $(\lambda n. \text{of_real } (\ln (\text{real } n)) - \text{of_real } (\ln (\text{real } n + 1))) \longrightarrow (0 :: 'a)$
proof (*rule Lim_transform_eventually*)
show *eventually* $(\lambda n. \text{of_real } (\ln (\text{real } n / (\text{real } n + 1))) =$
 $\text{of_real } (\ln (\text{real } n)) - (\text{of_real } (\ln (\text{real } n + 1)) :: 'a))$ *at_top*
using *eventually_gt_at_top*[*of 0::nat*] **by** *eventually_elim* (*simp add: ln_div*)
qed

from *summable_Digamma*[*OF z*]
have $(\lambda n. \text{inverse } (\text{of_nat } (n+1)) - \text{inverse } (z + \text{of_nat } n))$
 $\text{sums } (\text{Digamma } z + \text{euler_mascheroni})$
by (*simp add: Digamma_def summable_sums*)
from *sums_diff*[*OF this euler_mascheroni_sum*]
have $(\lambda n. \text{of_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of_real } (\ln (\text{real } n + 1)) - \text{inverse}$
 $(z + \text{of_nat } n))$
 $\text{sums } \text{Digamma } z$ **by** (*simp add: add_ac*)
hence $(\lambda m. (\sum n < m. \text{of_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of_real } (\ln (\text{real } n +$
 $1)))) -$
 $(\sum n < m. \text{inverse } (z + \text{of_nat } n))) \longrightarrow \text{Digamma } z$
by (*simp add: sums_def sum_subtractf*)
also have $(\lambda m. (\sum n < m. \text{of_real } (\ln (\text{real } (\text{Suc } n) + 1)) - \text{of_real } (\ln (\text{real } n$
 $+ 1)))) =$
 $(\lambda m. \text{of_real } (\ln (m + 1)) :: 'a)$
by (*subst sum_lessThan_telescope simp_all*)
finally show *?thesis* **by** (*rule Lim_transform*) (*insert lim, simp*)
qed

theorem *Polygamma_LIMSEQ*:

```

fixes z :: 'a :: {banach,real_normed_field}
assumes z ≠ 0 and n > 0
shows (λk. inverse ((z + of_nat k) ^ Suc n)) sums ((-1) ^ Suc n * Polygamma
n z / fact n)
using Polygamma_converges'[OF assms(1), of Suc n] assms(2)
by (simp add: sums_iff Polygamma_def)

```

theorem has_field_derivative_ln_Gamma_complex [derivative_intros]:

```

fixes z :: complex
assumes z: z ∉ ℝ≤0
shows (ln_Gamma has_field_derivative Digamma z) (at z)
proof -
have not_nonpos_Int [simp]: t ∉ ℤ≤0 if Re t > 0 for t
using that by (auto elim!: nonpos_Ints_cases')
from z have z': z ∉ ℤ≤0 and z'': z ≠ 0 using nonpos_Ints_subset_nonpos_Reals
nonpos_Reals_zero_I
by blast+
let ?f' = λz k. inverse (of_nat (Suc k)) - inverse (z + of_nat (Suc k))
let ?f = λz k. z / of_nat (Suc k) - ln (1 + z / of_nat (Suc k)) and ?F' = λz.
∑ n. ?f' z n
define d where d = min (norm z / 2) (if Im z = 0 then Re z / 2 else abs (Im z)
/ 2)
from z have d: d > 0 norm z / 2 ≥ d by (auto simp add: complex_nonpos_Reals_iff
d_def)
have ball: Im t = 0 → Re t > 0 if dist z t < d for t
using no_nonpos_Real_in_ball[OF z, of t] that unfolding d_def by (force
simp add: complex_nonpos_Reals_iff)
have sums: (λn. inverse (z + of_nat (Suc n)) - inverse (z + of_nat n)) sums
(0 - inverse (z + of_nat 0))
by (intro telescope_sums filterlim_compose[OF tendsto_inverse_0]
tendsto_add_filterlim_at_infinity[OF tendsto_const] tendsto_of_nat)

have ((λz. ∑ n. ?f z n) has_field_derivative ?F' z) (at z)
using d z ln_Gamma_series'_aux[OF z']
apply (intro has_field_derivative_series'(2)[of ball z d _ z] uniformly_summable_deriv_ln_Gamma)
apply (auto intro!: derivative_eq_intros add_pos_pos mult_pos_pos dest!: ball
simp: field_simps sums_iff nonpos_Reals_divide_of_nat_iff
simp del: of_nat_Suc)
apply (auto simp add: complex_nonpos_Reals_iff)
done
with z have ((λz. (∑ k. ?f z k) - euler_mascheroni * z - Ln z) has_field_derivative
?F' z - euler_mascheroni - inverse z) (at z)
by (force intro!: derivative_eq_intros simp: Digamma_def)
also have ?F' z - euler_mascheroni - inverse z = (?F' z + -inverse z) -
euler_mascheroni by simp
also from sums have -inverse z = (∑ n. inverse (z + of_nat (Suc n)) -
inverse (z + of_nat n))
by (simp add: sums_iff)
also from sums summable_deriv_ln_Gamma[OF z']

```

have $?F' z + \dots = (\sum n. \text{inverse} (\text{of_nat} (\text{Suc } n)) - \text{inverse} (z + \text{of_nat } n))$
by (*subst suminf_add*) (*simp_all add: add_ac sums_iff*)
also have $\dots - \text{euler_mascheroni} = \text{Digamma } z$ **by** (*simp add: Digamma_def*)
finally have $((\lambda z. (\sum k. ?f z k) - \text{euler_mascheroni} * z - \text{Ln } z)$
 $\quad \text{has_field_derivative } \text{Digamma } z) (\text{at } z) .$
moreover from *eventually_nhds_ball[OF d(1), of z]*
have *eventually* $(\lambda z. \text{ln_Gamma } z = (\sum k. ?f z k) - \text{euler_mascheroni} * z - \text{Ln } z)$ (*nhds z*)
proof *eventually_elim*
fix *t* **assume** $t \in \text{ball } z \text{ } d$
hence $t \notin \mathbb{Z}_{\leq 0}$ **by** (*auto dest!: ball_elim!: nonpos_Ints_cases*)
from *ln_Gamma_series'_aux[OF this]*
show $\text{ln_Gamma } t = (\sum k. ?f t k) - \text{euler_mascheroni} * t - \text{Ln } t$ **by** (*simp add: sums_iff*)
qed
ultimately show *?thesis* **by** (*subst DERIV_cong_ev[OF refl _ refl]*)
qed

declare *has_field_derivative_ln_Gamma_complex*[*THEN DERIV_chain2, derivative_intros*]

lemma *Digamma_1* [*simp*]: $\text{Digamma } (1 :: 'a :: \{\text{real_normed_field}, \text{banach}\}) = - \text{euler_mascheroni}$
by (*simp add: Digamma_def*)

lemma *Digamma_plus1*:

assumes $z \neq 0$
shows $\text{Digamma } (z+1) = \text{Digamma } z + 1/z$
proof –
have *sums*: $(\lambda k. \text{inverse} (z + \text{of_nat } k) - \text{inverse} (z + \text{of_nat} (\text{Suc } k)))$
 $\quad \text{sums } (\text{inverse} (z + \text{of_nat } 0) - 0)$
by (*intro telescope_sums'*[*OF filterlim_compose*[*OF tendsto_inverse_0*]]
 $\quad \text{tendsto_add_filterlim_at_infinity$ [*OF tendsto_const*] *tendsto_of_nat*)
have $\text{Digamma } (z+1) = (\sum k. \text{inverse} (\text{of_nat} (\text{Suc } k)) - \text{inverse} (z + \text{of_nat} (\text{Suc } k))) -$
 $\quad \text{euler_mascheroni}$ (*is _ = suminf ?f - _*) **by** (*simp add: Digamma_def add_ac*)
also have $\text{suminf } ?f = (\sum k. \text{inverse} (\text{of_nat} (\text{Suc } k)) - \text{inverse} (z + \text{of_nat } k)) +$
 $\quad (\sum k. \text{inverse} (z + \text{of_nat } k) - \text{inverse} (z + \text{of_nat} (\text{Suc } k)))$
using *summable_Digamma*[*OF assms*] *sums* **by** (*subst suminf_add*) (*simp_all add: add_ac sums_iff*)
also have $(\sum k. \text{inverse} (z + \text{of_nat } k) - \text{inverse} (z + \text{of_nat} (\text{Suc } k))) = 1/z$
using *sums* **by** (*simp add: sums_iff inverse_eq_divide*)
finally show *?thesis* **by** (*simp add: Digamma_def*[*of z*])
qed

theorem *Polygamma_plus1*:

assumes $z \neq 0$

shows $\text{Polygamma } n (z + 1) = \text{Polygamma } n z + (-1)^{\wedge n} * \text{fact } n / (z \wedge \text{Suc } n)$

proof (*cases* $n = 0$)

assume $n: n \neq 0$

let $?f = \lambda k. \text{inverse } ((z + \text{of_nat } k) \wedge \text{Suc } n)$

have $\text{Polygamma } n (z + 1) = (-1)^{\wedge \text{Suc } n} * \text{fact } n * (\sum k. ?f (k+1))$

using n **by** (*simp add: Polygamma_def add_ac*)

also have $(\sum k. ?f (k+1)) + (\sum k < 1. ?f k) = (\sum k. ?f k)$

using *Polygamma_converges'*[*OF assms, of Suc n*] n

by (*subst suminf_split_initial_segment [symmetric]*) *simp_all*

hence $(\sum k. ?f (k+1)) = (\sum k. ?f k) - \text{inverse } (z \wedge \text{Suc } n)$ **by** (*simp add: algebra_simps*)

also have $(-1)^{\wedge \text{Suc } n} * \text{fact } n * ((\sum k. ?f k) - \text{inverse } (z \wedge \text{Suc } n)) = \text{Polygamma } n z + (-1)^{\wedge n} * \text{fact } n / (z \wedge \text{Suc } n)$ **using** n

by (*simp add: inverse_eq_divide algebra_simps Polygamma_def*)

finally show *?thesis* .

qed (*insert assms, simp add: Digamma_plus1 inverse_eq_divide*)

theorem *Digamma_of_nat*:

$\text{Digamma } (\text{of_nat } (\text{Suc } n)) :: 'a :: \{\text{real_normed_field, banach}\} = \text{harm } n - \text{euler_mascheroni}$

proof (*induction n*)

case (*Suc n*)

have $\text{Digamma } (\text{of_nat } (\text{Suc } (\text{Suc } n))) :: 'a = \text{Digamma } (\text{of_nat } (\text{Suc } n) + 1)$

by *simp*

also have $\dots = \text{Digamma } (\text{of_nat } (\text{Suc } n)) + \text{inverse } (\text{of_nat } (\text{Suc } n))$

by (*subst Digamma_plus1*) (*simp_all add: inverse_eq_divide del: of_nat_Suc*)

also have $\text{Digamma } (\text{of_nat } (\text{Suc } n)) :: 'a = \text{harm } n - \text{euler_mascheroni}$ **by** (*rule Suc*)

also have $\dots + \text{inverse } (\text{of_nat } (\text{Suc } n)) = \text{harm } (\text{Suc } n) - \text{euler_mascheroni}$

by (*simp add: harm_Suc*)

finally show *?case* .

qed (*simp add: harm_def*)

lemma *Digamma_natural*: $\text{Digamma } (\text{numeral } n) = \text{harm } (\text{pred_numeral } n) - \text{euler_mascheroni}$

by (*subst of_nat_natural [symmetric]*, *subst numeral_eq_Suc*, *subst Digamma_of_nat*) (*rule refl*)

lemma *Polygamma_of_real*: $x \neq 0 \implies \text{Polygamma } n (\text{of_real } x) = \text{of_real } (\text{Polygamma } n x)$

unfolding *Polygamma_def* **using** *summable_Digamma*[*of x*] *Polygamma_converges'*[*of x Suc n*]

by (*simp_all add: suminf_of_real*)

lemma *Polygamma_Real*: $z \in \mathbb{R} \implies z \neq 0 \implies \text{Polygamma } n z \in \mathbb{R}$

by (*elim Reals_cases*, *hypsubst*, *subst Polygamma_of_real*) *simp_all*

```

lemma Digamma_half_integer:
  Digamma (of_nat n + 1/2 :: 'a :: {real_normed_field,banach}) =
    (∑ k<n. 2 / (of_nat (2*k+1))) - euler_mascheroni - of_real (2 * ln 2)
proof (induction n)
  case 0
  have Digamma (1/2 :: 'a) = of_real (Digamma (1/2)) by (simp add: Polygamma_of_real
[symmetric])
  also have Digamma (1/2::real) =
    (∑ k. inverse (of_nat (Suc k)) - inverse (of_nat k + 1/2)) -
euler_mascheroni
  by (simp add: Digamma_def add_ac)
  also have (∑ k. inverse (of_nat (Suc k) :: real) - inverse (of_nat k + 1/2)) =
    (∑ k. inverse (1/2) * (inverse (2 * of_nat (Suc k)) - inverse (2 *
of_nat k + 1)))
  by (simp_all add: add_ac inverse_mult_distrib[symmetric] ring_distrib del:
inverse_divide)
  also have ... = - 2 * ln 2 using sums_minus[OF alternating_harmonic_series_sums]
  by (subst suminf_mult) (simp_all add: algebra_simps sums_iff)
  finally show ?case by simp
next
  case (Suc n)
  have nz: 2 * of_nat n + (1:: 'a) ≠ 0
  using of_nat_neq_0[of 2*n] by (simp only: of_nat_Suc) (simp add: add_ac)
  hence nz': of_nat n + (1/2::'a) ≠ 0 by (simp add: field_simps)
  have Digamma (of_nat (Suc n) + 1/2 :: 'a) = Digamma (of_nat n + 1/2 +
1) by simp
  also from nz' have ... = Digamma (of_nat n + 1/2) + 1 / (of_nat n + 1/2)
  by (rule Digamma_plus1)
  also from nz nz' have 1 / (of_nat n + 1/2 :: 'a) = 2 / (2 * of_nat n + 1)
  by (subst divide_eq_eq) simp_all
  also note Suc
  finally show ?case by (simp add: add_ac)
qed

lemma Digamma_one_half: Digamma (1/2) = - euler_mascheroni - of_real
(2 * ln 2)
  using Digamma_half_integer[of 0] by simp

lemma Digamma_real_three_halves_pos: Digamma (3/2 :: real) > 0
proof -
  have -Digamma (3/2 :: real) = -Digamma (of_nat 1 + 1/2) by simp
  also have ... = 2 * ln 2 + euler_mascheroni - 2 by (subst Digamma_half_integer)
simp
  also note euler_mascheroni_less_13_over_22
  also note ln2_le_25_over_36
  finally show ?thesis by simp
qed

```

```

theorem has_field_derivative_Polygamma [derivative_intros]:
  fixes z :: 'a :: {real_normed_field,euclidean_space}
  assumes z: z  $\notin$   $\mathbb{Z}_{\leq 0}$ 
  shows (Polygamma n has_field_derivative Polygamma (Suc n) z) (at z within
A)
proof (rule has_field_derivative_at_within, cases n = 0)
  assume n: n = 0
  let ?f =  $\lambda k z.$  inverse (of_nat (Suc k)) - inverse (z + of_nat k)
  let ?F =  $\lambda z.$   $\sum k.$  ?f k z and ?f' =  $\lambda k z.$  inverse ((z + of_nat k)2)
  from no_nonpos_Int_in_ball'[OF z] obtain d where d: 0 < d  $\wedge$  t. t  $\in$  ball z
d  $\implies$  t  $\notin$   $\mathbb{Z}_{\leq 0}$ 
  by auto
  from z have summable: summable ( $\lambda k.$  inverse (of_nat (Suc k)) - inverse (z +
of_nat k))
  by (intro summable_Digamma) force
  from z have conv: uniformly_convergent_on (ball z d) ( $\lambda k z.$   $\sum i < k.$  inverse
((z + of_nat i)2))
  by (intro Polygamma_converges) auto
  with d have summable ( $\lambda k.$  inverse ((z + of_nat k)2)) unfolding summable_iff_convergent
by (auto dest!: uniformly_convergent_imp_convergent simp: summable_iff_convergent
)

  have (?F has_field_derivative ( $\sum k.$  ?f' k z)) (at z)
proof (rule has_field_derivative_series'[of ball z d _ _ z])
  fix k :: nat and t :: 'a assume t: t  $\in$  ball z d
  from t d(2)[of t] show (( $\lambda z.$  ?f k z) has_field_derivative ?f' k t) (at t within
ball z d)
  by (auto intro!: derivative_eq_intros simp: power2_eq_square simp del:
of_nat_Suc
dest!: plus_of_nat_eq_0_imp_elim!: nonpos_Ints_cases)
qed (insert d(1) summable conv, (assumption|simp)+)
  with z show (Polygamma n has_field_derivative Polygamma (Suc n) z) (at z)
unfolding Digamma_def [abs_def] Polygamma_def [abs_def] using n
by (force simp: power2_eq_square intro!: derivative_eq_intros)
next
assume n: n  $\neq$  0
from z have z': z  $\neq$  0 by auto
from no_nonpos_Int_in_ball'[OF z] obtain d where d: 0 < d  $\wedge$  t. t  $\in$  ball z
d  $\implies$  t  $\notin$   $\mathbb{Z}_{\leq 0}$ 
  by auto
  define n' where n' = Suc n
from n have n': n'  $\geq$  2 by (simp add: n'_def)
have (( $\lambda z.$   $\sum k.$  inverse ((z + of_nat k) ^ n') has_field_derivative
( $\sum k.$  - of_nat n' * inverse ((z + of_nat k) ^ (n'+1)))) (at z)
proof (rule has_field_derivative_series'[of ball z d _ _ z])
  fix k :: nat and t :: 'a assume t: t  $\in$  ball z d
  with d have t': t  $\notin$   $\mathbb{Z}_{\leq 0}$  t  $\neq$  0 by auto
  show (( $\lambda a.$  inverse ((a + of_nat k) ^ n')) has_field_derivative

```



```

      - of_nat n' * inverse ((t + of_nat k) ^ (n'+1)) (at t within ball z
d) using t'
    by (fastforce intro!: derivative_eq_intros simp: divide_simps power_diff dest:
plus_of_nat_eq_0_imp)
  next
    have uniformly_convergent_on (ball z d)
      (λk z. (- of_nat n' :: 'a) * (∑ i<k. inverse ((z + of_nat i) ^ (n'+1))))
      using z' n by (intro uniformly_convergent_mult Polygamma_converges)
(simp_all add: n'_def)
    thus uniformly_convergent_on (ball z d)
      (λk z. ∑ i<k. - of_nat n' * inverse ((z + of_nat i :: 'a) ^ (n'+1)))
      by (subst (asm) sum_distrib_left) simp
  qed (insert Polygamma_converges'[OF z' n'] d, simp_all)
  also have (∑ k. - of_nat n' * inverse ((z + of_nat k) ^ (n' + 1))) =
    (- of_nat n') * (∑ k. inverse ((z + of_nat k) ^ (n' + 1)))
    using Polygamma_converges'[OF z', of n'+1] n' by (subst suminf_mult)
simp_all
  finally have ((λz. ∑ k. inverse ((z + of_nat k) ^ n')) has_field_derivative
    - of_nat n' * (∑ k. inverse ((z + of_nat k) ^ (n' + 1)))) (at z) .
  from DERIV_cmult[OF this, of (-1) ^ Suc n * fact n :: 'a]
  show (Polygamma n has_field_derivative Polygamma (Suc n) z) (at z)
    unfolding n'_def Polygamma_def[abs_def] using n by (simp add: alge-
bra_simps)
  qed

declare has_field_derivative_Polygamma[THEN DERIV_chain2, derivative_intros]

lemma isCont_Polygamma [continuous_intros]:
  fixes f :: _ ⇒ 'a :: {real_normed_field,euclidean_space}
  shows isCont f z ⇒ f z ∉ ℤ<0 ⇒ isCont (λx. Polygamma n (f x)) z
  by (rule isCont_o2[OF _ DERIV_isCont[OF has_field_derivative_Polygamma]])

lemma continuous_on_Polygamma:
  A ∩ ℤ<0 = {} ⇒ continuous_on A (Polygamma n :: _ ⇒ 'a :: {real_normed_field,euclidean_space})
  by (intro continuous_at_imp_continuous_on isCont_Polygamma[OF continu-
ous_ident] ballI) blast

lemma isCont_ln_Gamma_complex [continuous_intros]:
  fixes f :: 'a::t2_space ⇒ complex
  shows isCont f z ⇒ f z ∉ ℝ<0 ⇒ isCont (λz. ln_Gamma (f z)) z
  by (rule isCont_o2[OF _ DERIV_isCont[OF has_field_derivative_ln_Gamma_complex]])

lemma continuous_on_ln_Gamma_complex [continuous_intros]:
  fixes A :: complex set
  shows A ∩ ℝ<0 = {} ⇒ continuous_on A ln_Gamma
  by (intro continuous_at_imp_continuous_on ballI isCont_ln_Gamma_complex[OF
continuous_ident])
    fastforce

```

lemma *deriv_Polygamma*:

assumes $z \notin \mathbb{Z}_{\leq 0}$
shows $\text{deriv } (\text{Polygamma } m) z =$
 $\text{Polygamma } (\text{Suc } m) (z :: 'a :: \{\text{real_normed_field, euclidean_space}\})$
by (*intro DERIV_imp_deriv has_field_derivative_Polygamma assms*)
thm *has_field_derivative_Polygamma*

lemma *higher_deriv_Polygamma*:

assumes $z \notin \mathbb{Z}_{\leq 0}$
shows $(\text{deriv } \widetilde{\sim} n) (\text{Polygamma } m) z =$
 $\text{Polygamma } (m + n) (z :: 'a :: \{\text{real_normed_field, euclidean_space}\})$

proof –

have *eventually* $(\lambda u. (\text{deriv } \widetilde{\sim} n) (\text{Polygamma } m) u = \text{Polygamma } (m + n) u)$
(nhds z)

proof (*induction n*)

case (*Suc n*)

from *Suc.IH* **have** *eventually* $(\lambda z. \text{eventually } (\lambda u. (\text{deriv } \widetilde{\sim} n) (\text{Polygamma } m) u = \text{Polygamma } (m + n) u) (\text{nhds } z)) (\text{nhds } z)$

by (*simp add: eventually_eventually*)

hence *eventually* $(\lambda z. \text{deriv } ((\text{deriv } \widetilde{\sim} n) (\text{Polygamma } m)) z =$
 $\text{deriv } (\text{Polygamma } (m + n)) z) (\text{nhds } z)$

by *eventually_elim* (*intro deriv_cong_ev refl*)

moreover **have** *eventually* $(\lambda z. z \in \text{UNIV} - \mathbb{Z}_{\leq 0}) (\text{nhds } z)$ **using** *assms*

by (*intro eventually_nhds_in_open open_Diff open_UNIV*) *auto*

ultimately show *?thesis* **by** *eventually_elim* (*simp_all add: deriv_Polygamma*)

qed *simp_all*

thus *?thesis* **by** (*rule eventually_nhds_x_imp_x*)

qed

lemma *deriv_ln_Gamma_complex*:

assumes $z \notin \mathbb{R}_{\leq 0}$
shows $\text{deriv } \ln_Gamma z = \text{Digamma } (z :: \text{complex})$
by (*intro DERIV_imp_deriv has_field_derivative_ln_Gamma_complex assms*)

lemma *higher_deriv_ln_Gamma_complex*:

assumes $(x :: \text{complex}) \notin \mathbb{R}_{\leq 0}$
shows $(\text{deriv } \widetilde{\sim} j) \ln_Gamma x = (\text{if } j = 0 \text{ then } \ln_Gamma x \text{ else } \text{Polygamma } (j - 1) x)$

proof (*cases j*)

case (*Suc j'*)

have $(\text{deriv } \widetilde{\sim} j') (\text{deriv } \ln_Gamma) x = (\text{deriv } \widetilde{\sim} j') \text{Digamma } x$

using *eventually_nhds_in_open*[*of UNIV - ℝ_{≤0} x*] *assms*

by (*intro higher_deriv_cong_ev refl*)

(*auto elim!: eventually_mono simp: open_Diff deriv_ln_Gamma_complex*)

also **have** $\dots = \text{Polygamma } j' x$ **using** *assms*

by (*subst higher_deriv_Polygamma*)

(*auto elim!: nonpos_Ints_cases simp: complex_nonpos_Reals_iff*)

finally show *?thesis* **using** *Suc* **by** (*simp del: funpow.simps add: funpow_Suc_right*)

qed simp_all

We define a type class that captures all the fundamental properties of the inverse of the Gamma function and defines the Gamma function upon that. This allows us to instantiate the type class both for the reals and for the complex numbers with a minimal amount of proof duplication.

```

class Gamma = real_normed_field + complete_space +
  fixes rGamma :: 'a ⇒ 'a
  assumes rGamma_eq_zero_iff_aux: rGamma z = 0 ⟷ (∃ n. z = - of_nat
n)
  assumes differentiable_rGamma_aux1:
    (∧ n. z ≠ - of_nat n) ⇒
      let d = (THE d. (λ n. ∑ k < n. inverse (of_nat (Suc k)) - inverse (z + of_nat
k))
        ⟶ d) - scaleR euler_mascheroni 1
      in filterlim (λ y. (rGamma y - rGamma z + rGamma z * d * (y - z)) /_R
        norm (y - z)) (nhds 0) (at z)
  assumes differentiable_rGamma_aux2:
    let z = - of_nat n
      in filterlim (λ y. (rGamma y - rGamma z - (-1) ^ n * (prod of_nat {1..n})
* (y - z)) /_R
        norm (y - z)) (nhds 0) (at z)
  assumes rGamma_series_aux: (∧ n. z ≠ - of_nat n) ⇒
    let fact' = (λ n. prod of_nat {1..n});
      exp = (λ x. THE e. (λ n. ∑ k < n. x ^ k /_R fact k) ⟶ e);
      pochhammer' = (λ a n. (∏ n = 0..n. a + of_nat n))
      in filterlim (λ n. pochhammer' z n / (fact' n * exp (z * (ln (of_nat n)
*_R 1))))
        (nhds (rGamma z)) sequentially
begin
subclass banach ..
end

```

definition Gamma z = inverse (rGamma z)

9.10.3 Basic properties

lemma Gamma_nonpos_Int: $z \in \mathbb{Z}_{\leq 0} \implies \text{Gamma } z = 0$
and rGamma_nonpos_Int: $z \in \mathbb{Z}_{\leq 0} \implies \text{rGamma } z = 0$
using rGamma_eq_zero_iff_aux[of z] **unfolding** Gamma_def **by** (auto elim!: nonpos_Ints_cases')

lemma Gamma_nonzero: $z \notin \mathbb{Z}_{\leq 0} \implies \text{Gamma } z \neq 0$
and rGamma_nonzero: $z \notin \mathbb{Z}_{\leq 0} \implies \text{rGamma } z \neq 0$
using rGamma_eq_zero_iff_aux[of z] **unfolding** Gamma_def **by** (auto elim!: nonpos_Ints_cases')

lemma Gamma_eq_zero_iff: $\text{Gamma } z = 0 \iff z \in \mathbb{Z}_{\leq 0}$
and rGamma_eq_zero_iff: $\text{rGamma } z = 0 \iff z \in \mathbb{Z}_{\leq 0}$

using $rGamma_eq_zero_iff_aux$ [of z] **unfolding** $Gamma_def$ **by** (*auto elim!*: *nonpos_Ints_cases'*)

lemma $rGamma_inverse_Gamma$: $rGamma\ z = inverse\ (Gamma\ z)$
unfolding $Gamma_def$ **by** *simp*

lemma $rGamma_series_LIMSEQ$ [*tendsto_intros*]:

$rGamma_series\ z \longrightarrow rGamma\ z$

proof (*cases* $z \in \mathbf{Z}_{\leq 0}$)

case *False*

hence $z \neq -\ of_nat\ n$ **for** n **by** *auto*

from $rGamma_series_aux$ [*OF this*] **show** *?thesis*

by (*simp add*: $rGamma_series_def$ [*abs_def*] *fact_prod pochhammer_Suc_prod*
 $exp_def\ of_real_def$ [*symmetric*] *suminf_def sums_def*[*abs_def*])

atLeast0AtMost)

qed (*insert* $rGamma_eq_zero_iff$ [of z], *simp_all add*: $rGamma_series_nonpos_Ints_LIMSEQ$)

theorem $Gamma_series_LIMSEQ$ [*tendsto_intros*]:

$Gamma_series\ z \longrightarrow Gamma\ z$

proof (*cases* $z \in \mathbf{Z}_{\leq 0}$)

case *False*

hence $(\lambda n. inverse\ (rGamma_series\ z\ n)) \longrightarrow inverse\ (rGamma\ z)$

by (*intro tendsto_intros*) (*simp_all add*: $rGamma_eq_zero_iff$)

also have $(\lambda n. inverse\ (rGamma_series\ z\ n)) = Gamma_series\ z$

by (*simp add*: $rGamma_series_def\ Gamma_series_def$ [*abs_def*])

finally show *?thesis* **by** (*simp add*: $Gamma_def$)

qed (*insert* $Gamma_eq_zero_iff$ [of z], *simp_all add*: $Gamma_series_nonpos_Ints_LIMSEQ$)

lemma $Gamma_altdef$: $Gamma\ z = lim\ (Gamma_series\ z)$

using $Gamma_series_LIMSEQ$ [of z] **by** (*simp add*: *limI*)

lemma $rGamma_1$ [*simp*]: $rGamma\ 1 = 1$

proof –

have A : *eventually* $(\lambda n. rGamma_series\ 1\ n = of_nat\ (Suc\ n) / of_nat\ n)$
sequentially

using *eventually_gt_at_top*[of $0::nat$]

by (*force elim!*: *eventually_mono simp*: $rGamma_series_def\ exp_of_real\ pochhammer_fact$

$field_split_simps\ pochhammer_rec'$ *dest!*: $pochhammer_eq_0_imp_nonpos_Int$)

have $rGamma_series\ 1 \longrightarrow 1$ **by** (*subst tendsto_cong*[*OF A*]) (*rule LIMSEQ_Suc_n_over_n*)

moreover have $rGamma_series\ 1 \longrightarrow rGamma\ 1$ **by** (*rule tendsto_intros*)

ultimately show *?thesis* **by** (*intro LIMSEQ_unique*)

qed

lemma $rGamma_plus1$: $z * rGamma\ (z + 1) = rGamma\ z$

proof –

let $?f = \lambda n. (z + 1) * inverse\ (of_nat\ n) + 1$

have *eventually* $(\lambda n. ?f\ n * rGamma_series\ z\ n = z * rGamma_series\ (z + 1))$

n) *sequentially*
using *eventually_gt_at_top*[*of 0::nat*]
proof *eventually_elim*
fix *n :: nat* **assume** *n: n > 0*
hence $z * rGamma_series (z + 1) n = inverse (of_nat n) * pochhammer z (Suc (Suc n)) / (fact n * exp (z * of_real (ln (of_nat n))))$
by (*subst pochhammer_rec*) (*simp add: rGamma_series_def field_simps exp_add exp_of_real*)
also from *n* **have** $\dots = ?f n * rGamma_series z n$
by (*subst pochhammer_rec'*) (*simp_all add: field_split_simps rGamma_series_def*)
finally show $?f n * rGamma_series z n = z * rGamma_series (z + 1) n ..$
qed
moreover have $(\lambda n. ?f n * rGamma_series z n) \longrightarrow ((z+1) * 0 + 1) * rGamma z$
by (*intro tendsto_intros lim_inverse_n*)
hence $(\lambda n. ?f n * rGamma_series z n) \longrightarrow rGamma z$ **by** *simp*
ultimately have $(\lambda n. z * rGamma_series (z + 1) n) \longrightarrow rGamma z$
by (*blast intro: Lim_transform_eventually*)
moreover have $(\lambda n. z * rGamma_series (z + 1) n) \longrightarrow z * rGamma (z + 1)$
by (*intro tendsto_intros*)
ultimately show $z * rGamma (z + 1) = rGamma z$ **using** *LIMSEQ_unique*
by *blast*
qed

lemma *pochhammer_rGamma*: $rGamma z = pochhammer z n * rGamma (z + of_nat n)$
proof (*induction n arbitrary: z*)
case (*Suc n z*)
have $rGamma z = pochhammer z n * rGamma (z + of_nat n)$ **by** (*rule Suc.IH*)
also note *rGamma_plus1 [symmetric]*
finally show *case* **by** (*simp add: add_ac pochhammer_rec'*)
qed *simp_all*

theorem *Gamma_plus1*: $z \notin \mathbb{Z}_{\leq 0} \implies Gamma (z + 1) = z * Gamma z$
using *rGamma_plus1*[*of z*] **by** (*simp add: rGamma_inverse_Gamma field_simps Gamma_eq_zero_iff*)

theorem *pochhammer_Gamma*: $z \notin \mathbb{Z}_{\leq 0} \implies pochhammer z n = Gamma (z + of_nat n) / Gamma z$
using *pochhammer_rGamma*[*of z*]
by (*simp add: rGamma_inverse_Gamma Gamma_eq_zero_iff field_simps*)

lemma *Gamma_0* [*simp*]: $Gamma 0 = 0$
and *rGamma_0* [*simp*]: $rGamma 0 = 0$
and *Gamma_neg_1* [*simp*]: $Gamma (- 1) = 0$
and *rGamma_neg_1* [*simp*]: $rGamma (- 1) = 0$

and Gamma_neg_numeral [simp]: $\text{Gamma} (- \text{numeral } n) = 0$
and $\text{rGamma_neg_numeral}$ [simp]: $\text{rGamma} (- \text{numeral } n) = 0$
and Gamma_neg_of_nat [simp]: $\text{Gamma} (- \text{of_nat } m) = 0$
and rGamma_neg_of_nat [simp]: $\text{rGamma} (- \text{of_nat } m) = 0$
by (simp_all add: $\text{rGamma_eq_zero_iff}$ Gamma_eq_zero_iff)

lemma Gamma_1 [simp]: $\text{Gamma } 1 = 1$ **unfolding** Gamma_def **by** simp

theorem Gamma_fact : $\text{Gamma} (1 + \text{of_nat } n) = \text{fact } n$
by (simp add: pochhammer_fact pochhammer_Gamma of_nat_in_nonpos_Ints_iff flip: of_nat_Suc)

lemma Gamma_numeral : $\text{Gamma} (\text{numeral } n) = \text{fact} (\text{pred_numeral } n)$
by (subst of_nat_numeral[symmetric], subst numeral_eq_Suc, subst of_nat_Suc, subst Gamma_fact) (rule refl)

lemma Gamma_of_int : $\text{Gamma} (\text{of_int } n) = (\text{if } n > 0 \text{ then } \text{fact} (\text{nat } (n - 1)) \text{ else } 0)$

proof (cases $n > 0$)

case True

hence $\text{Gamma} (\text{of_int } n) = \text{Gamma} (\text{of_nat } (\text{Suc } (\text{nat } (n - 1))))$ **by** (subst of_nat_Suc) simp_all

with True **show** ?thesis **by** (subst (asm) of_nat_Suc, subst (asm) Gamma_fact) simp

qed (simp_all add: Gamma_eq_zero_iff nonpos_Ints_of_int)

lemma rGamma_of_int : $\text{rGamma} (\text{of_int } n) = (\text{if } n > 0 \text{ then } \text{inverse} (\text{fact} (\text{nat } (n - 1))) \text{ else } 0)$

by (simp add: Gamma_of_int $\text{rGamma_inverse_Gamma}$)

lemma Gamma_seriesI :

assumes $(\lambda n. g \ n / \text{Gamma_series } z \ n) \longrightarrow 1$

shows $g \longrightarrow \text{Gamma } z$

proof (rule Lim_transform_eventually)

have $1/2 > (0::\text{real})$ **by** simp

from tendstoD[OF assms, OF this]

show eventually $(\lambda n. g \ n / \text{Gamma_series } z \ n * \text{Gamma_series } z \ n = g \ n)$ sequentially

by (force elim!: eventually_mono simp: dist_real_def)

from assms **have** $(\lambda n. g \ n / \text{Gamma_series } z \ n * \text{Gamma_series } z \ n) \longrightarrow 1 * \text{Gamma } z$

by (intro tendsto_intros)

thus $(\lambda n. g \ n / \text{Gamma_series } z \ n * \text{Gamma_series } z \ n) \longrightarrow \text{Gamma } z$ **by** simp

qed

lemma $\text{Gamma_seriesI}'$:

assumes $f \longrightarrow \text{rGamma } z$

assumes $(\lambda n. g \ n * f \ n) \longrightarrow 1$

```

  assumes  $z \notin \mathbb{Z}_{\leq 0}$ 
  shows  $g \longrightarrow \text{Gamma } z$ 
proof (rule Lim_transform_eventually)
  have  $1/2 > (0::\text{real})$  by simp
  from tendstoD[OF assms(2), OF this] show eventually  $(\lambda n. g \ n * f \ n / f \ n = g \ n)$  sequentially
  by (force elim!: eventually_mono simp: dist_real_def)
  from assms have  $(\lambda n. g \ n * f \ n / f \ n) \longrightarrow 1 / r\text{Gamma } z$ 
  by (intro tendsto_divide assms) (simp_all add: rGamma_eq_zero_iff)
  thus  $(\lambda n. g \ n * f \ n / f \ n) \longrightarrow \text{Gamma } z$  by (simp add: Gamma_def divide_inverse)
qed

```

```

lemma Gamma_series'_LIMSEQ:  $\text{Gamma\_series}' \ z \longrightarrow \text{Gamma } z$ 
  by (cases  $z \in \mathbb{Z}_{\leq 0}$ ) (simp_all add: Gamma_nonpos_Int Gamma_seriesI[OF Gamma_series_Gamma_series'] Gamma_series'_nonpos_Ints_LIMSEQ[of z])

```

9.10.4 Differentiability

lemma has_field_derivative_rGamma_no_nonpos_int:

```

  assumes  $z \notin \mathbb{Z}_{\leq 0}$ 
  shows  $(r\text{Gamma } \text{has\_field\_derivative } -r\text{Gamma } z * \text{Digamma } z)$  (at  $z$  within  $A$ )
proof (rule has_field_derivative_at_within)
  from assms have  $z \neq - \text{of\_nat } n$  for  $n$  by auto
  from differentiable_rGamma_aux1[OF this]
  show  $(r\text{Gamma } \text{has\_field\_derivative } -r\text{Gamma } z * \text{Digamma } z)$  (at  $z$ )
  unfolding Digamma_def suminf_def sums_def[abs_def]
    has_field_derivative_def has_derivative_def netlimit_at
  by (simp add: Let_def bounded_linear_mult_right mult_ac of_real_def [symmetric])
qed

```

lemma has_field_derivative_rGamma_nonpos_int:

```

   $(r\text{Gamma } \text{has\_field\_derivative } (-1)^n * \text{fact } n)$  (at  $(- \text{of\_nat } n)$  within  $A$ )
  apply (rule has_field_derivative_at_within)
  using differentiable_rGamma_aux2[of  $n$ ]
  unfolding Let_def has_field_derivative_def has_derivative_def netlimit_at
  by (simp only: bounded_linear_mult_right mult_ac of_real_def [symmetric] fact_prod) simp

```

lemma has_field_derivative_rGamma [derivative_intros]:

```

   $(r\text{Gamma } \text{has\_field\_derivative } (\text{if } z \in \mathbb{Z}_{\leq 0} \text{ then } (-1)^{\text{nat } \lfloor \text{norm } z \rfloor} * \text{fact } (\text{nat } \lfloor \text{norm } z \rfloor)$ 
  else  $-r\text{Gamma } z * \text{Digamma } z)$  (at  $z$  within  $A$ )
  using has_field_derivative_rGamma_no_nonpos_int[of  $z \ A$ ]
    has_field_derivative_rGamma_nonpos_int[of  $\text{nat } \lfloor \text{norm } z \rfloor \ A$ ]
  by (auto elim!: nonpos_Ints_cases')

```

```

declare has_field_derivative_rGamma_no_nonpos_int [THEN DERIV_chain2,
derivative_intros]
declare has_field_derivative_rGamma [THEN DERIV_chain2, derivative_intros]
declare has_field_derivative_rGamma_nonpos_int [derivative_intros]
declare has_field_derivative_rGamma_no_nonpos_int [derivative_intros]
declare has_field_derivative_rGamma [derivative_intros]

```

```

theorem has_field_derivative_Gamma [derivative_intros]:
   $z \notin \mathbf{Z}_{\leq 0} \implies (\text{Gamma has\_field\_derivative Gamma } z * \text{Digamma } z)$  (at  $z$  within
  A)
  unfolding Gamma_def [abs_def]
  by (fastforce intro!: derivative_eq_intros simp: rGamma_eq_zero_iff)

```

```

declare has_field_derivative_Gamma [THEN DERIV_chain2, derivative_intros]

```

```

hide_fact rGamma_eq_zero_iff_aux differentiable_rGamma_aux1 differentiable_rGamma_aux2
differentiable_rGamma_aux2 rGamma_series_aux Gamma_class.rGamma_eq_zero_iff_aux

```

```

lemma continuous_on_rGamma [continuous_intros]: continuous_on A rGamma
  by (rule DERIV_continuous_on has_field_derivative_rGamma)+

```

```

lemma continuous_on_Gamma [continuous_intros]:  $A \cap \mathbf{Z}_{\leq 0} = \{\}$   $\implies$  contin-
uous_on A Gamma
  by (rule DERIV_continuous_on has_field_derivative_Gamma)+ blast

```

```

lemma isCont_rGamma [continuous_intros]:
  isCont f z  $\implies$  isCont ( $\lambda x.$  rGamma (f x)) z
  by (rule isCont_o2[OF DERIV_isCont[OF has_field_derivative_rGamma]])

```

```

lemma isCont_Gamma [continuous_intros]:
  isCont f z  $\implies$   $f z \notin \mathbf{Z}_{\leq 0} \implies$  isCont ( $\lambda x.$  Gamma (f x)) z
  by (rule isCont_o2[OF DERIV_isCont[OF has_field_derivative_Gamma]])

```

9.10.5 The complex Gamma function

```

instantiation complex :: Gamma
begin

```

```

definition rGamma_complex :: complex  $\Rightarrow$  complex where
  rGamma_complex z = lim (rGamma_series z)

```

```

lemma rGamma_series_complex_converges:
  convergent (rGamma_series (z :: complex)) (is ?thesis1)
  and rGamma_complex_altdef:
  rGamma z = (if z  $\in \mathbf{Z}_{\leq 0}$  then 0 else exp (-ln_Gamma z)) (is ?thesis2)

```

```

proof -
  have ?thesis1  $\wedge$  ?thesis2
  proof (cases z  $\in \mathbf{Z}_{\leq 0}$ )

```



```

case False
have rGamma_series z  $\longrightarrow$  exp ( $- \ln\_Gamma$  z)
proof (rule Lim_transform_eventually)
  from ln_Gamma_series_complex_converges'[OF False]
  obtain d where  $0 < d$  uniformly_convergent_on (ball z d) ( $\lambda n$  z. ln_Gamma_series
z n)
    by auto
    from this(1) uniformly_convergent_imp_convergent[OF this(2), of z]
    have ln_Gamma_series z  $\longrightarrow$  lim (ln_Gamma_series z) by (simp add:
convergent_LIMSEQ_iff)
    thus ( $\lambda n$ . exp ( $- \ln\_Gamma\_series$  z n))  $\longrightarrow$  exp ( $- \ln\_Gamma$  z)
      unfolding convergent_def ln_Gamma_def by (intro tendsto_exp_tendsto_minus)
    from eventually_gt_at_top[of  $0::nat$ ] exp_ln_Gamma_series_complex False
    show eventually ( $\lambda n$ . exp ( $- \ln\_Gamma\_series$  z n) = rGamma_series z
n) sequentially
      by (force elim!: eventually_mono simp: exp_minus_Gamma_series_def
rGamma_series_def)
    qed
with False show ?thesis
  by (auto simp: convergent_def rGamma_complex_def intro!: limI)
next
case True
then obtain k where z =  $-$  of_nat k by (erule nonpos_Ints_cases')
also have rGamma_series ...  $\longrightarrow$   $0$ 
  by (subst tendsto_cong[OF rGamma_series_minus_of_nat]) (simp_all add:
convergent_const)
  finally show ?thesis using True
  by (auto simp: rGamma_complex_def convergent_def intro!: limI)
qed
thus ?thesis1 ?thesis2 by blast+
qed

```

context

begin

private lemma *rGamma_complex_plus1*: $z * rGamma$ ($z + 1$) = *rGamma* ($z ::$
complex)

proof $-$

let *?f* = λn . ($z + 1$) * *inverse* (*of_nat* *n*) + 1

have *eventually* (λn . *?f* *n* * *rGamma_series* *z* *n* = $z * rGamma_series$ ($z + 1$)
n) *sequentially*

using *eventually_gt_at_top*[*of* $0::nat$]

proof *eventually_elim*

fix *n* :: *nat* **assume** *n*: $n > 0$

hence $z * rGamma_series$ ($z + 1$) *n* = *inverse* (*of_nat* *n*) *
pochhammer *z* (*Suc* (*Suc* *n*)) / (*fact* *n* * *exp* ($z * of_real$ (\ln (*of_nat*
n))))

by (subst pochhammer_rec) (simp add: rGamma_series_def field_simps
 exp_add exp_of_real)
 also from n have ... = ?f n * rGamma_series z n
 by (subst pochhammer_rec') (simp_all add: field_split_simps rGamma_series_def
 add_ac)
 finally show ?f n * rGamma_series z n = z * rGamma_series (z + 1) n ..
 qed
 moreover have ($\lambda n. ?f n * rGamma_series z n$) $\longrightarrow ((z+1) * 0 + 1) * rGamma z$
 using rGamma_series_complex_converges
 by (intro tendsto_intros lim_inverse_n)
 (simp_all add: convergent_LIMSEQ_iff rGamma_complex_def)
 hence ($\lambda n. ?f n * rGamma_series z n$) $\longrightarrow rGamma z$ by simp
 ultimately have ($\lambda n. z * rGamma_series (z + 1) n$) $\longrightarrow rGamma z$
 by (blast intro: Lim_transform_eventually)
 moreover have ($\lambda n. z * rGamma_series (z + 1) n$) $\longrightarrow z * rGamma (z + 1)$
 1)
 using rGamma_series_complex_converges
 by (auto intro!: tendsto_mult simp: rGamma_complex_def convergent_LIMSEQ_iff)
 ultimately show $z * rGamma (z + 1) = rGamma z$ using LIMSEQ_unique
 by blast
 qed

private lemma has_field_derivative_rGamma_complex_no_nonpos_Int:
 assumes ($z :: complex$) $\notin \mathbb{Z}_{\leq 0}$
 shows ($rGamma$ has_field_derivative - $rGamma z * Digamma z$) (at z)
proof -
 have diff: ($rGamma$ has_field_derivative - $rGamma z * Digamma z$) (at z) **if**
 $Re z > 0$ **for** z
proof (subst DERIV_cong_ev[OF refl _ refl])
 from that have eventually ($\lambda t. t \in ball z (Re z/2)$) (nhds z)
 by (intro eventually_nhds_in_nhd) simp_all
 thus eventually ($\lambda t. rGamma t = exp (- ln_Gamma t)$) (nhds z)
 using no_nonpos_Int_in_ball_complex[OF that]
 by (auto elim!: eventually_mono simp: rGamma_complex_altdef)
next
 have $z \notin \mathbb{R}_{\leq 0}$ using that by (simp add: complex_nonpos_Reals_iff)
 with that show (($\lambda t. exp (- ln_Gamma t)$) has_field_derivative (- $rGamma$
 $z * Digamma z$)) (at z)
 by (force elim!: nonpos_Ints_cases intro!: derivative_eq_intros simp: rGamma_complex_altdef)
 qed

from assms **show** ($rGamma$ has_field_derivative - $rGamma z * Digamma z$)
 (at z)
proof (induction nat [1 - Re z] arbitrary: z)
 case (Suc n z)
 from Suc.premis have z: $z \neq 0$ by auto
 from Suc.hyps have $n = nat [- Re z]$ by linarith
 hence $A: n = nat [1 - Re (z + 1)]$ by simp

```

from Suc.prems have  $B: z + 1 \notin \mathbb{Z}_{\leq 0}$  by (force dest: plus_one_in_nonpos_Ints_imp)

have  $((\lambda z. z * (rGamma \circ (\lambda z. z + 1))) z)$  has_field_derivative
   $-rGamma (z + 1) * (Digamma (z + 1) * z - 1)$  (at z)
  by (rule derivative_eq_intros DERIV_chain Suc refl A B)+ (simp add:
algebra_simps)
also have  $(\lambda z. z * (rGamma \circ (\lambda z. z + 1 :: complex))) z = rGamma$ 
  by (simp add: rGamma_complex_plus1)
also from  $z$  have  $Digamma (z + 1) * z - 1 = z * Digamma z$ 
  by (subst Digamma_plus1) (simp_all add: field_simps)
also have  $-rGamma (z + 1) * (z * Digamma z) = -rGamma z * Digamma z$ 
  by (simp add: rGamma_complex_plus1[of z, symmetric])
finally show ?case .
qed (intro diff, simp)
qed

private lemma rGamma_complex_1:  $rGamma (1 :: complex) = 1$ 
proof -
  have  $A: eventually (\lambda n. rGamma\_series 1 n = of\_nat (Suc n) / of\_nat n)$ 
sequentially
    using eventually_gt_at_top[of 0::nat]
    by (force elim!: eventually_mono simp: rGamma_series_def exp_of_real pochhammer_fact
      field_split_simps pochhammer_rec' dest!: pochhammer_eq_0_imp_nonpos_Int)
  have  $rGamma\_series 1 \longrightarrow 1$  by (subst tendsto_cong[OF A]) (rule LIM-SEQ_Suc_n_over_n)
  thus  $rGamma 1 = (1 :: complex)$  unfolding rGamma_complex_def by (rule limI)
qed

private lemma has_field_derivative_rGamma_complex_nonpos_Int:
   $(rGamma \text{ has\_field\_derivative } (-1)^{\wedge} n * fact n)$  (at  $(- of\_nat n :: complex)$ )
proof (induction n)
  case 0
  have  $A: (0 :: complex) + 1 \notin \mathbb{Z}_{\leq 0}$  by simp
  have  $((\lambda z. z * (rGamma \circ (\lambda z. z + 1 :: complex))) z)$  has_field_derivative 1
    (at 0)
    by (rule derivative_eq_intros DERIV_chain refl
      has_field_derivative_rGamma_complex_no_nonpos_Int A)+ (simp
add: rGamma_complex_1)
    thus ?case by (simp add: rGamma_complex_plus1)
  next
  case (Suc n)
  hence  $A: (rGamma \text{ has\_field\_derivative } (-1)^{\wedge} n * fact n)$ 
    (at  $(- of\_nat (Suc n) + 1 :: complex)$ ) by simp
  have  $((\lambda z. z * (rGamma \circ (\lambda z. z + 1 :: complex))) z)$  has_field_derivative
     $(- 1)^{\wedge} Suc n * fact (Suc n)$  (at  $(- of\_nat (Suc n))$ )
    by (rule derivative_eq_intros refl A DERIV_chain)+
      (simp add: algebra_simps rGamma_complex_altdef)

```

3064

thus ?case by (simp add: rGamma_complex_plus1)
qed

instance proof

fix z :: complex **show** (rGamma z = 0) \longleftrightarrow (\exists n. z = - of_nat n)
by (auto simp: rGamma_complex_altdef elim!: nonpos_Ints_cases')

next

fix z :: complex **assume** \bigwedge n. z \neq - of_nat n

hence z \notin $\mathbb{Z}_{\leq 0}$ **by** (auto elim!: nonpos_Ints_cases')

from has_field_derivative_rGamma_complex_no_nonpos_Int[OF this]

show let d = (THE d. $(\lambda$ n. \sum k < n. inverse (of_nat (Suc k)) - inverse (z + of_nat k))

\longrightarrow d) - euler_mascheroni *_R 1 in (λ y. (rGamma y -

rGamma z +

$rGamma z * d * (y - z)) /_R \text{cmod } (y - z)) - z \rightarrow 0$

by (simp add: has_field_derivative_def has_derivative_def Digamma_def sums_def [abs_def]

of_real_def[symmetric] suminf_def)

next

fix n :: nat

from has_field_derivative_rGamma_complex_nonpos_Int[of n]

show let z = - of_nat n in (λ y. (rGamma y - rGamma z - $(-1)^n * \text{prod}$ of_nat {1..n} *

$(y - z)) /_R \text{cmod } (y - z)) - z \rightarrow 0$

by (simp add: has_field_derivative_def has_derivative_def fact_prod Let_def)

next

fix z :: complex

from rGamma_series_complex_converges[of z] **have** rGamma_series z \longrightarrow rGamma z

by (simp add: convergent_LIMSEQ_iff rGamma_complex_def)

thus let fact' = λ n. prod of_nat {1..n};

exp = λ x. THE e. $(\lambda$ n. \sum k < n. $x^k /_R \text{fact } k) \longrightarrow e$;

pochhammer' = λ a n. \prod n = 0..n. a + of_nat n

in $(\lambda$ n. pochhammer' z n / (fact' n * exp (z * ln (real_of_nat n) *_R 1))) \longrightarrow rGamma z

by (simp add: fact_prod pochhammer_Suc_prod rGamma_series_def [abs_def] exp_def

of_real_def [symmetric] suminf_def sums_def [abs_def] atLeast0At-

Most)

qed

end

end

lemma Gamma_complex_altdef:

Gamma z = (if z \in $\mathbb{Z}_{<0}$ then 0 else exp (ln_Gamma (z :: complex)))

unfolding Gamma_def rGamma_complex_altdef **by** (simp add: exp_minus)

```

lemma cnj_rGamma:  $\text{cnj } (r\text{Gamma } z) = r\text{Gamma } (\text{cnj } z)$ 
proof -
  have rGamma_series ( $\text{cnj } z$ ) =  $(\lambda n. \text{cnj } (r\text{Gamma\_series } z n))$ 
    by (intro ext) (simp_all add: rGamma_series_def exp_cnj)
  also have ...  $\longrightarrow \text{cnj } (r\text{Gamma } z)$  by (intro tendsto_cnj tendsto_intros)
  finally show ?thesis unfolding rGamma_complex_def by (intro sym[OF limI])
qed

lemma cnj_Gamma:  $\text{cnj } (\text{Gamma } z) = \text{Gamma } (\text{cnj } z)$ 
  unfolding Gamma_def by (simp add: cnj_rGamma)

lemma Gamma_complex_real:
   $z \in \mathbb{R} \implies \text{Gamma } z \in (\mathbb{R} :: \text{complex set})$  and rGamma_complex_real:  $z \in \mathbb{R} \implies r\text{Gamma } z \in \mathbb{R}$ 
  by (simp_all add: Reals_cnj_iff cnj_Gamma cnj_rGamma)

lemma field_differentiable_rGamma:  $r\text{Gamma}$  field_differentiable (at  $z$  within  $A$ )
  using has_field_derivative_rGamma[of  $z$ ] unfolding field_differentiable_def
by blast

lemma holomorphic_rGamma [holomorphic_intros]:  $r\text{Gamma}$  holomorphic_on  $A$ 
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_rGamma)

lemma holomorphic_rGamma' [holomorphic_intros]:
  assumes  $f$  holomorphic_on  $A$ 
  shows  $(\lambda x. r\text{Gamma } (f x))$  holomorphic_on  $A$ 
proof -
  have  $r\text{Gamma} \circ f$  holomorphic_on  $A$  using assms
    by (intro holomorphic_on_compose assms holomorphic_rGamma)
  thus ?thesis by (simp only: o_def)
qed

lemma analytic_rGamma:  $r\text{Gamma}$  analytic_on  $A$ 
  unfolding analytic_on_def by (auto intro!: exI[of _ 1] holomorphic_rGamma)

lemma field_differentiable_Gamma:  $z \notin \mathbb{Z}_{\leq 0} \implies \text{Gamma}$  field_differentiable (at  $z$  within  $A$ )
  using has_field_derivative_Gamma[of  $z$ ] unfolding field_differentiable_def by
auto

lemma holomorphic_Gamma [holomorphic_intros]:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies \text{Gamma}$ 
holomorphic_on  $A$ 
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_Gamma)

lemma holomorphic_Gamma' [holomorphic_intros]:
  assumes  $f$  holomorphic_on  $A$  and  $\bigwedge x. x \in A \implies f x \notin \mathbb{Z}_{\leq 0}$ 
  shows  $(\lambda x. \text{Gamma } (f x))$  holomorphic_on  $A$ 
proof -

```

```

have  $\Gamma \circ f$  holomorphic_on  $A$  using assms
  by (intro holomorphic_on_compose assms holomorphic_Γ) auto
  thus ?thesis by (simp only: o_def)
qed

```

```

lemma analytic_Γ:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies$  Γ analytic_on  $A$ 
  by (rule analytic_on_subset[of _ UNIV -  $\mathbb{Z}_{\leq 0}$ ], subst analytic_on_open)
  (auto intro!: holomorphic_Γ)

```

```

lemma field_differentiable_ln_Γ_complex:
   $z \notin \mathbb{R}_{\leq 0} \implies$  ln_Γ field_differentiable (at  $(z::\text{complex})$  within  $A$ )
  by (rule field_differentiable_within_subset[of _ UNIV])
  (force simp: field_differentiable_def intro!: derivative_intros)+

```

```

lemma holomorphic_ln_Γ [holomorphic_intros]:  $A \cap \mathbb{R}_{\leq 0} = \{\}$   $\implies$  ln_Γ
holomorphic_on  $A$ 
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_ln_Γ_complex)

```

```

lemma analytic_ln_Γ:  $A \cap \mathbb{R}_{\leq 0} = \{\}$   $\implies$  ln_Γ analytic_on  $A$ 
  by (rule analytic_on_subset[of _ UNIV -  $\mathbb{R}_{\leq 0}$ ], subst analytic_on_open)
  (auto intro!: holomorphic_ln_Γ)

```

```

lemma has_field_derivative_rΓ_complex' [derivative_intros]:
  (rΓ has_field_derivative (if  $z \in \mathbb{Z}_{\leq 0}$  then  $(-1)^{\sim}(\text{nat } [-\text{Re } z]) * \text{fact } (\text{nat } [-\text{Re } z])$  else
     $-r\Gamma z * \text{Digamma } z$ ) (at  $z$  within  $A$ )
  using has_field_derivative_rΓ[of  $z$ ] by (auto elim!: nonpos_Ints_cases')

```

```

declare has_field_derivative_rΓ_complex' [THEN DERIV_chain2, derivative_intros]

```

```

lemma field_differentiable_Polygamma:
  fixes  $z :: \text{complex}$ 
  shows
   $z \notin \mathbb{Z}_{\leq 0} \implies$  Polygamma  $n$  field_differentiable (at  $z$  within  $A$ )
  using has_field_derivative_Polygamma[of  $z n$ ] unfolding field_differentiable_def
by auto

```

```

lemma holomorphic_on_Polygamma [holomorphic_intros]:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies$ 
Polygamma  $n$  holomorphic_on  $A$ 
  unfolding holomorphic_on_def by (auto intro!: field_differentiable_Polygamma)

```

```

lemma analytic_on_Polygamma:  $A \cap \mathbb{Z}_{\leq 0} = \{\}$   $\implies$  Polygamma  $n$  analytic_on
 $A$ 
  by (rule analytic_on_subset[of _ UNIV -  $\mathbb{Z}_{\leq 0}$ ], subst analytic_on_open)
  (auto intro!: holomorphic_on_Polygamma)

```

lemma *analytic_on_rGamma* [*analytic_intros*]: f *analytic_on* $A \implies (\lambda w. rGamma (f w))$ *analytic_on* A
using *analytic_on_compose*[*OF _ analytic_rGamma, of f A*] **by** (*simp add: o_def*)

lemma *analytic_on_ln_Gamma* [*analytic_intros*]:
 f *analytic_on* $A \implies (\bigwedge z. z \in A \implies f z \notin \mathbb{R}_{\leq 0}) \implies (\lambda w. ln_Gamma (f w))$ *analytic_on* A
by (*rule analytic_on_compose*[*OF _ analytic_ln_Gamma, unfolded o_def*])
(*auto simp: o_def*)

lemma *Polygamma_plus_of_nat*:
assumes $\forall k < m. z \neq -of_nat\ k$
shows $Polygamma\ n\ (z + of_nat\ m) =$
 $Polygamma\ n\ z + (-1)^n * fact\ n * (\sum_{k < m}. 1 / (z + of_nat\ k)) ^{Suc\ n}$
using *assms*
proof (*induction m*)
case (*Suc m*)
have $Polygamma\ n\ (z + of_nat\ (Suc\ m)) = Polygamma\ n\ (z + of_nat\ m + 1)$
by (*simp add: add_ac*)
also have $\dots = Polygamma\ n\ (z + of_nat\ m) + (-1)^n * fact\ n * (1 / ((z + of_nat\ m) ^{Suc\ n}))$
using *Suc.prem*s **by** (*subst Polygamma_plus1*) (*auto simp: add_eq_0_iff2*)
also have $Polygamma\ n\ (z + of_nat\ m) =$
 $Polygamma\ n\ z + (-1)^n * (\sum_{k < m}. 1 / (z + of_nat\ k)) ^{Suc\ n}$
 $* fact\ n$
using *Suc.prem*s **by** (*subst Suc.IH*) *auto*
finally show ?*case*
by (*simp add: algebra_simps*)
qed *auto*

lemma *tendsto_Gamma* [*tendsto_intros*]:
assumes $(f \longrightarrow c)$ $F\ c \notin \mathbb{Z}_{<0}$
shows $((\lambda z. Gamma (f z)) \longrightarrow Gamma\ c)$ F
by (*intro isCont_tendsto_compose*[*OF _ assms(1)*] *continuous_intros assms*)

lemma *tendsto_Polygamma* [*tendsto_intros*]:
fixes $f :: _ \Rightarrow 'a :: \{real_normed_field, euclidean_space\}$
assumes $(f \longrightarrow c)$ $F\ c \notin \mathbb{Z}_{<0}$
shows $((\lambda z. Polygamma\ n (f z)) \longrightarrow Polygamma\ n\ c)$ F
by (*intro isCont_tendsto_compose*[*OF _ assms(1)*] *continuous_intros assms*)

lemma *analytic_on_Gamma'* [*analytic_intros*]:
assumes f *analytic_on* $A \forall x \in A. f\ x \notin \mathbb{Z}_{<0}$
shows $(\lambda z. Gamma (f z))$ *analytic_on* A
using *analytic_on_compose_gen*[*OF assms(1) analytic_Gamma* [*of f 'A*]] *assms(2)*

by (auto simp: o_def)

lemma *analytic_on_Polygamma'* [analytic_intros]:
assumes f analytic_on $A \ \forall x \in A. f \ x \notin \mathbb{Z}_{\leq 0}$
shows $(\lambda z. \text{Polygamma } n \ (f \ z))$ analytic_on A
using analytic_on_compose_gen[OF assms(1) analytic_on_Polygamma[of $f \ 'A \ n$]] assms(2)
by (auto simp: o_def)

9.10.6 The real Gamma function

lemma *rGamma_series_real*:
eventually $(\lambda n. \text{rGamma_series } x \ n = \text{Re } (\text{rGamma_series } (\text{of_real } x) \ n))$ sequentially
using eventually_gt_at_top[of $0 :: \text{nat}$]
proof eventually_elim
fix $n :: \text{nat}$ **assume** $n > 0$
have $\text{Re } (\text{rGamma_series } (\text{of_real } x) \ n) =$
 $\text{Re } (\text{of_real } (\text{pochhammer } x \ (\text{Suc } n)) / (\text{fact } n * \exp (\text{of_real } (x * \ln (\text{real_of_nat } n))))))$
using n **by** (simp add: rGamma_series_def powr_def pochhammer_of_real)
also from n **have** $\dots = \text{Re } (\text{of_real } ((\text{pochhammer } x \ (\text{Suc } n)) /$
 $(\text{fact } n * (\exp (x * \ln (\text{real_of_nat } n))))))$
by (subst exp_of_real) simp
also from n **have** $\dots = \text{rGamma_series } x \ n$
by (subst Re_complex_of_real) (simp add: rGamma_series_def powr_def)
finally show $\text{rGamma_series } x \ n = \text{Re } (\text{rGamma_series } (\text{of_real } x) \ n) ..$
qed

instantiation *real* :: *Gamma*

begin

definition *rGamma_real* $x = \text{Re } (\text{rGamma } (\text{of_real } x :: \text{complex}))$

instance proof

fix $x :: \text{real}$

have $\text{rGamma } x = \text{Re } (\text{rGamma } (\text{of_real } x))$ **by** (simp add: rGamma_real_def)

also have $\text{of_real } \dots = \text{rGamma } (\text{of_real } x :: \text{complex})$

by (intro of_real_Re rGamma_complex_real) simp_all

also have $\dots = 0 \iff x \in \mathbb{Z}_{\leq 0}$ **by** (simp add: rGamma_eq_zero_iff of_real_in_nonpos_Ints_iff)

also have $\dots \iff (\exists n. x = - \text{of_nat } n)$ **by** (auto elim!: nonpos_Ints_cases')

finally show $(\text{rGamma } x) = 0 \iff (\exists n. x = - \text{real_of_nat } n)$ **by** simp

next

fix $x :: \text{real}$ **assume** $\bigwedge n. x \neq - \text{of_nat } n$

hence $x: \text{complex_of_real } x \notin \mathbb{Z}_{\leq 0}$

by (subst of_real_in_nonpos_Ints_iff) (auto elim!: nonpos_Ints_cases')

then have $x \neq 0$ **by** auto

with x **have** $(\text{rGamma has_field_derivative } - \text{rGamma } x * \text{Digamma } x)$ (at x)

by (fastforce intro!: derivative_eq_intros has_vector_derivative_real_field)


```

      simp: Polygamma_of_real rGamma_real_def [abs_def])
  thus let d = (THE d. ( $\lambda n. \sum k < n. \text{inverse} (\text{of\_nat} (\text{Suc } k)) - \text{inverse} (x + \text{of\_nat } k)$ )
     $\longrightarrow d) - \text{euler\_mascheroni} *_{\mathbb{R}} 1$  in ( $\lambda y. (\text{rGamma } y - \text{rGamma } x + \text{rGamma } x * d * (y - x)) /_{\mathbb{R}} \text{norm } (y - x) - x \rightarrow 0$ )
    by (simp add: has_field_derivative_def has_derivative_def Digamma_def sums_def [abs_def]
      of_real_def[symmetric] suminf_def)
next
  fix n :: nat
  have (rGamma has_field_derivative  $(-1)^n * \text{fact } n$ ) (at  $(- \text{of\_nat } n :: \text{real})$ )
    by (fastforce intro!: derivative_eq_intros has_vector_derivative_real_field
      simp: Polygamma_of_real rGamma_real_def [abs_def])
  thus let x =  $- \text{of\_nat } n$  in ( $\lambda y. (\text{rGamma } y - \text{rGamma } x - (-1)^n * \text{prod of\_nat } \{1..n\} * (y - x)) /_{\mathbb{R}} \text{norm } (y - x) - x :: \text{real} \rightarrow 0$ )
    by (simp add: has_field_derivative_def has_derivative_def fact_prod Let_def)
next
  fix x :: real
  have rGamma_series x  $\longrightarrow \text{rGamma } x$ 
  proof (rule Lim_transform_eventually)
    show ( $\lambda n. \text{Re} (\text{rGamma\_series} (\text{of\_real } x) n)$ )  $\longrightarrow \text{rGamma } x$  unfolding
    rGamma_real_def
      by (intro tendsto_intros)
    qed (insert rGamma_series_real, simp add: eq_commute)
  thus let fact' =  $\lambda n. \text{prod of\_nat } \{1..n\}$ ;
    exp =  $\lambda x. \text{THE } e. (\lambda n. \sum k < n. x^k /_{\mathbb{R}} \text{fact } k) \longrightarrow e$ ;
    pochhammer' =  $\lambda a n. \prod n = 0..n. a + \text{of\_nat } n$ 
    in ( $\lambda n. \text{pochhammer}' x n / (\text{fact}' n * \text{exp } (x * \ln (\text{real\_of\_nat } n) *_{\mathbb{R}} 1))$ )
 $\longrightarrow \text{rGamma } x$ 
    by (simp add: fact_prod pochhammer_Suc_prod rGamma_series_def [abs_def]
      exp_def
        of_real_def [symmetric] suminf_def sums_def [abs_def] atLeast0AtMost)
  qed
end

```

lemma rGamma_complex_of_real: $\text{rGamma} (\text{complex_of_real } x) = \text{complex_of_real} (\text{rGamma } x)$

unfolding rGamma_real_def **using** rGamma_complex_real **by** simp

lemma Gamma_complex_of_real: $\text{Gamma} (\text{complex_of_real } x) = \text{complex_of_real} (\text{Gamma } x)$

unfolding Gamma_def **by** (simp add: rGamma_complex_of_real)

lemma rGamma_real_altdef: $\text{rGamma } x = \text{lim} (\text{rGamma_series } (x :: \text{real}))$

by (rule sym, rule limI, rule tendsto_intros)

lemma *Gamma_real_altdef1*: $\Gamma x = \lim (\Gamma_series (x :: real))$
 by (rule sym, rule limI, rule tendsto_intros)

lemma *Gamma_real_altdef2*: $\Gamma x = \text{Re} (\Gamma (\text{of_real } x))$
 using *rGamma_complex_real*[OF *Reals_of_real*[of *x*]]
 by (elim *Reals_cases*)
 (simp only: *Gamma_def* *rGamma_real_def* *of_real_inverse*[symmetric] *Re_complex_of_real*)

lemma *ln_Gamma_series_complex_of_real*:
 $x > 0 \implies n > 0 \implies \ln_Gamma_series (\text{complex_of_real } x) n = \text{of_real}$
 $(\ln_Gamma_series x n)$

proof –

assume *xn*: $x > 0 \ n > 0$

have $\ln (\text{complex_of_real } x / \text{of_nat } k + 1) = \text{of_real} (\ln (x / \text{of_nat } k + 1))$

if $k \geq 1$ for *k*

using that *xn* by (subst *Ln_of_real* [symmetric]) (auto intro!: *add_nonneg_pos* *simp*: *field_simps*)

with *xn* show ?thesis by (simp add: *ln_Gamma_series_def* *Ln_of_real*)

qed

lemma *ln_Gamma_real_converges*:

assumes $(x :: real) > 0$

shows convergent $(\ln_Gamma_series x)$

proof –

have $(\lambda n. \ln_Gamma_series (\text{complex_of_real } x) n) \longrightarrow \ln_Gamma (\text{of_real } x)$ using *assms*

by (intro *ln_Gamma_complex_LIMSEQ*) (auto simp: *of_real_in_nonpos_Ints_iff*)
 moreover from *eventually_gt_at_top*[of $0 :: nat$]

have eventually $(\lambda n. \text{complex_of_real} (\ln_Gamma_series x n) = \ln_Gamma_series (\text{complex_of_real } x) n)$ sequentially

by eventually_elim (simp add: *ln_Gamma_series_complex_of_real* *assms*)

ultimately have $(\lambda n. \text{complex_of_real} (\ln_Gamma_series x n)) \longrightarrow \ln_Gamma (\text{of_real } x)$

by (subst *tendsto_cong*) *assumption*+

from *tendsto_Re*[OF *this*] show ?thesis by (auto simp: *convergent_def*)

qed

lemma *ln_Gamma_real_LIMSEQ*: $(x :: real) > 0 \implies \ln_Gamma_series x \longrightarrow \ln_Gamma x$

using *ln_Gamma_real_converges*[of *x*] **unfolding** *ln_Gamma_def* by (simp add: *convergent_LIMSEQ_iff*)

lemma *ln_Gamma_complex_of_real*: $x > 0 \implies \ln_Gamma (\text{complex_of_real } x) = \text{of_real} (\ln_Gamma x)$

proof (unfold *ln_Gamma_def*, rule *limI*, rule *Lim_transform_eventually*)

assume $x > 0$

show eventually $(\lambda n. \text{of_real} (\ln_Gamma_series x n) =$

```

      ln_Gamma_series (complex_of_real x) n sequentially
    using eventually_gt_at_top[of 0::nat]
    by eventually_elim (simp add: ln_Gamma_series_complex_of_real x)
qed (intro tendsto_of_real, insert ln_Gamma_real_LIMSEQ[of x], simp add:
ln_Gamma_def)

lemma Gamma_real_pos_exp:  $x > (0 :: real) \implies \Gamma x = \exp (\ln \Gamma x)$ 
  by (auto simp: Gamma_real_altdef2 Gamma_complex_altdef_of_real_in_nonpos_Ints_iff
      ln_Gamma_complex_of_real_exp_of_real)

lemma ln_Gamma_real_pos:  $x > 0 \implies \ln \Gamma x = \ln (\Gamma x :: real)$ 
  unfolding Gamma_real_pos_exp by simp

lemma ln_Gamma_complex_conv_fact:  $n > 0 \implies \ln \Gamma (\text{of\_nat } n :: \text{complex}) = \ln (\text{fact } (n - 1))$ 
  using ln_Gamma_complex_of_real[of real n] Gamma_fact[of n - 1, where 'a = real]
  by (simp add: ln_Gamma_real_pos of_nat_diff Ln_of_real [symmetric])

lemma ln_Gamma_real_conv_fact:  $n > 0 \implies \ln \Gamma (\text{real } n) = \ln (\text{fact } (n - 1))$ 
  using Gamma_fact[of n - 1, where 'a = real]
  by (simp add: ln_Gamma_real_pos of_nat_diff Ln_of_real [symmetric])

lemma Gamma_real_pos [simp, intro]:  $x > (0 :: real) \implies \Gamma x > 0$ 
  by (simp add: Gamma_real_pos_exp)

lemma Gamma_real_nonneg [simp, intro]:  $x > (0 :: real) \implies \Gamma x \geq 0$ 
  by (simp add: Gamma_real_pos_exp)

lemma has_field_derivative_ln_Gamma_real [derivative_intros]:
  assumes  $x > (0 :: real)$ 
  shows  $(\ln \Gamma \text{ has\_field\_derivative } \text{Digamma } x) \text{ (at } x)$ 
proof (subst DERIV_cong_ev[OF refl _ refl])
  from assms show  $((\text{Re} \circ \ln \Gamma \circ \text{complex\_of\_real}) \text{ has\_field\_derivative } \text{Digamma } x) \text{ (at } x)$ 
  by (auto intro!: derivative_eq_intros has_vector_derivative_real_field
      simp: Polygamma_of_real o_def)
  from eventually_nhds_in_nhd[of x {0<..}] assms
  show eventually  $(\lambda y. \ln \Gamma y = (\text{Re} \circ \ln \Gamma \circ \text{of\_real}) y) \text{ (nhds } x)$ 
  by (auto elim!: eventually_mono simp: ln_Gamma_complex_of_real interior_open)
qed

lemma field_differentiable_ln_Gamma_real:
 $x > 0 \implies \ln \Gamma \text{ field\_differentiable (at } (x :: real) \text{ within } A)$ 
  by (rule field_differentiable_within_subset[of _ UNIV])
  (auto simp: field_differentiable_def intro!: derivative_intros)+

```

declare *has_field_derivative_ln_Gamma_real* [THEN DERIV_chain2, derivative_intros]

lemma *deriv_ln_Gamma_real*:

assumes $z > 0$

shows $\text{deriv } \ln_Gamma\ z = \text{Digamma } (z :: \text{real})$

by (intro DERIV_imp_deriv *has_field_derivative_ln_Gamma_real* *assms*)

lemma *higher_deriv_ln_Gamma_real*:

assumes $(x :: \text{real}) > 0$

shows $(\text{deriv } \overset{\sim}{\sim} j) \ln_Gamma\ x = (\text{if } j = 0 \text{ then } \ln_Gamma\ x \text{ else } \text{Polygamma } (j - 1) x)$

proof (cases j)

case (Suc j')

have $(\text{deriv } \overset{\sim}{\sim} j') (\text{deriv } \ln_Gamma) x = (\text{deriv } \overset{\sim}{\sim} j') \text{Digamma } x$

using *eventually_nhds_in_open*[of $\{0 < ..\}$ x] *assms*

by (intro *higher_deriv_cong_ev_refl*)

(*auto elim!*: *eventually_mono simp: open_Diff deriv_ln_Gamma_real*)

also have $\dots = \text{Polygamma } j' x$ **using** *assms*

by (*subst higher_deriv_Polygamma*)

(*auto elim!*: *nonpos_Ints_cases simp: complex_nonpos_Reals_iff*)

finally show *?thesis* **using** *Suc* **by** (*simp del: funpow.simps add: funpow_Suc_right*)

qed *simp_all*

lemma *higher_deriv_ln_Gamma_complex_of_real*:

assumes $(x :: \text{real}) > 0$

shows $(\text{deriv } \overset{\sim}{\sim} j) \ln_Gamma\ (\text{complex_of_real } x) = \text{of_real } ((\text{deriv } \overset{\sim}{\sim} j) \ln_Gamma\ x)$

using *assms*

by (*auto simp: higher_deriv_ln_Gamma_real higher_deriv_ln_Gamma_complex ln_Gamma_complex_of_real Polygamma_of_real*)

lemma *has_field_derivative_rGamma_real'* [*derivative_intros*]:

$(\text{rGamma } \text{has_field_derivative } (\text{if } x \in \mathbb{Z}_{\leq 0} \text{ then } (-1)^{\text{nat } [-x]} * \text{fact } (\text{nat } [-x]) \text{ else } -\text{rGamma } x * \text{Digamma } x)) (\text{at } x \text{ within } A)$

using *has_field_derivative_rGamma*[of x] **by** (*force elim!*: *nonpos_Ints_cases'*)

declare *has_field_derivative_rGamma_real'* [THEN DERIV_chain2, derivative_intros]

lemma *Polygamma_real_odd_pos*:

assumes $(x :: \text{real}) \notin \mathbb{Z}_{\leq 0}$ *odd* n

shows $\text{Polygamma } n\ x > 0$

proof –

from *assms* **have** $x \neq 0$ **by** *auto*

with *assms* **show** *?thesis*

unfolding *Polygamma_def* **using** *Polygamma_converges'*[of x *Suc* n]

by (*auto simp: zero_less_power_eq simp del: power_Suc*

dest: plus_of_nat_eq_0_imp intro!: *mult_pos_pos suminf_pos*)

qed

lemma *Polygamma_real_even_neg*:

assumes $(x::real) > 0$ $n > 0$ *even* n

shows $Polygamma\ n\ x < 0$

using *assms* **unfolding** *Polygamma_def* **using** *Polygamma_converges'*[of x *Suc* n]

by (*auto intro!*: *mult_pos_pos suminf_pos*)

lemma *Polygamma_real_strict_mono*:

assumes $x > 0$ $x < (y::real)$ *even* n

shows $Polygamma\ n\ x < Polygamma\ n\ y$

proof –

have $\exists \xi. x < \xi \wedge \xi < y \wedge Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$

using *assms* **by** (*intro MVT2 derivative_intros impI allI*) (*auto elim!*: *nonpos_Ints_cases*)

then obtain ξ

where $\xi: x < \xi \wedge \xi < y$

and *Polygamma*: $Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$

by *auto*

note *Polygamma*

also from ξ *assms* **have** $(y - x) * Polygamma\ (Suc\ n)\ \xi > 0$

by (*intro mult_pos_pos Polygamma_real_odd_pos*) (*auto elim!*: *nonpos_Ints_cases*)

finally show *?thesis* **by** *simp*

qed

lemma *Polygamma_real_strict_antimono*:

assumes $x > 0$ $x < (y::real)$ *odd* n

shows $Polygamma\ n\ x > Polygamma\ n\ y$

proof –

have $\exists \xi. x < \xi \wedge \xi < y \wedge Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$

using *assms* **by** (*intro MVT2 derivative_intros impI allI*) (*auto elim!*: *nonpos_Ints_cases*)

then obtain ξ

where $\xi: x < \xi \wedge \xi < y$

and *Polygamma*: $Polygamma\ n\ y - Polygamma\ n\ x = (y - x) * Polygamma\ (Suc\ n)\ \xi$

by *auto*

note *Polygamma*

also from ξ *assms* **have** $(y - x) * Polygamma\ (Suc\ n)\ \xi < 0$

by (*intro mult_pos_neg Polygamma_real_even_neg*) *simp_all*

finally show *?thesis* **by** *simp*

qed

lemma *Polygamma_real_mono*:

assumes $x > 0$ $x \leq (y::real)$ *even* n

3074

shows $Polygamma\ n\ x \leq Polygamma\ n\ y$
using $Polygamma_real_strict_mono[OF\ assms(1)\ _ \ assms(3),\ of\ y]\ assms(2)$
by $(cases\ x = y)\ simp_all$

lemma $Digamma_real_strict_mono: (0::real) < x \implies x < y \implies Digamma\ x < Digamma\ y$
by $(rule\ Polygamma_real_strict_mono)\ simp_all$

lemma $Digamma_real_mono: (0::real) < x \implies x \leq y \implies Digamma\ x \leq Digamma\ y$
by $(rule\ Polygamma_real_mono)\ simp_all$

lemma $Digamma_real_ge_three_halves_pos:$
assumes $x \geq 3/2$
shows $Digamma\ (x :: real) > 0$
proof –
have $0 < Digamma\ (3/2 :: real)$ **by** $(fact\ Digamma_real_three_halves_pos)$
also from $assms$ **have** $\dots \leq Digamma\ x$ **by** $(intro\ Polygamma_real_mono)\ simp_all$
finally show $?thesis$.
qed

lemma $ln_Gamma_real_strict_mono:$
assumes $x \geq 3/2\ x < y$
shows $ln_Gamma\ (x :: real) < ln_Gamma\ y$
proof –
have $\exists \xi. x < \xi \wedge \xi < y \wedge ln_Gamma\ y - ln_Gamma\ x = (y - x) * Digamma\ \xi$
using $assms$ **by** $(intro\ MVT2\ derivative_intros\ impI\ allI)\ (auto\ elim!: non-pos_Ints_cases)$
then obtain ξ **where** $\xi: x < \xi \wedge \xi < y$
and $ln_Gamma: ln_Gamma\ y - ln_Gamma\ x = (y - x) * Digamma\ \xi$
by $auto$
note ln_Gamma
also from ξ $assms$ **have** $(y - x) * Digamma\ \xi > 0$
by $(intro\ mult_pos_pos\ Digamma_real_ge_three_halves_pos)\ simp_all$
finally show $?thesis$ **by** $simp$
qed

lemma $Gamma_real_strict_mono:$
assumes $x \geq 3/2\ x < y$
shows $Gamma\ (x :: real) < Gamma\ y$
proof –
from $Gamma_real_pos_exp[of\ x]$ $assms$ **have** $Gamma\ x = exp\ (ln_Gamma\ x)$
by $simp$
also have $\dots < exp\ (ln_Gamma\ y)$ **by** $(intro\ exp_less_mono\ ln_Gamma_real_strict_mono\ assms)$
also from $Gamma_real_pos_exp[of\ y]$ $assms$ **have** $\dots = Gamma\ y$ **by** $simp$
finally show $?thesis$.

qed

theorem *log_convex_Gamma_real*: *convex_on* {0<..} (ln ∘ Gamma :: real ⇒ real)
by (rule *convex_on_realI*[of __ *Digamma*])
 (auto intro!: *derivative_eq_intros Polygamma_real_mono Gamma_real_pos*
simp: o_def Gamma_eq_zero_iff elim!: nonpos_Ints_cases')

9.10.7 The uniqueness of the real Gamma function

The following is a proof of the Bohr–Mollerup theorem, which states that any log-convex function G on the positive reals that fulfils $G(1) = 1$ and satisfies the functional equation $G(x + 1) = x G(x)$ must be equal to the Gamma function. In principle, if G is a holomorphic complex function, one could then extend this from the positive reals to the entire complex plane (minus the non-positive integers, where the Gamma function is not defined).

context

fixes $G :: real \Rightarrow real$
assumes G_1 : $G\ 1 = 1$
assumes G_plus1 : $x > 0 \implies G\ (x + 1) = x * G\ x$
assumes G_pos : $x > 0 \implies G\ x > 0$
assumes *log_convex_G*: *convex_on* {0<..} (ln ∘ G)
begin

private lemma *G_fact*: $G\ (of_nat\ n + 1) = fact\ n$
using G_plus1 [of real $n + 1$ for n]
by (*induction* n) (*simp_all* add: $G_1\ G_plus1$)

private definition $S :: real \Rightarrow real \Rightarrow real$ **where**
 $S\ x\ y = (ln\ (G\ y) - ln\ (G\ x)) / (y - x)$

private lemma *S_eq*:
 $n \geq 2 \implies S\ (of_nat\ n)\ (of_nat\ n + x) = (ln\ (G\ (real\ n + x)) - ln\ (fact\ (n - 1))) / x$
by (*subst* G_fact [*symmetric*]) (*simp* add: $S_def\ add_ac\ of_nat_diff$)

private lemma *G_lower*:
assumes x : $x > 0$ **and** n : $n \geq 1$
shows $\Gamma_series\ x\ n \leq G\ x$

proof –

have $(ln \circ G)\ (real\ (Suc\ n)) \leq ((ln \circ G)\ (real\ (Suc\ n) + x) - (ln \circ G)\ (real\ (Suc\ n) - 1)) / (real\ (Suc\ n) + x - (real\ (Suc\ n) - 1)) * (real\ (Suc\ n) - (real\ (Suc\ n) - 1)) + (ln \circ G)\ (real\ (Suc\ n) - 1)$
using $x\ n$ **by** (*intro* *convex_onD_Icc'* *convex_on_subset*[OF *log_convex_G*])
auto
hence $S\ (of_nat\ n)\ (of_nat\ (Suc\ n)) \leq S\ (of_nat\ (Suc\ n))\ (of_nat\ (Suc\ n) + x)$
unfolding S_def **using** x **by** (*simp* add: *field_simps*)

also have S (of_nat n) (of_nat (Suc n)) = \ln (fact n) - \ln (fact ($n-1$))
unfolding S_def **using** n
by (subst (1 2) G_fact [symmetric]) (simp_all add: add_ac of_nat_diff)
also have ... = \ln (fact n / fact ($n-1$)) **by** (subst ln_div) simp_all
also from n **have** fact n / fact ($n-1$) = n **by** (cases n) simp_all
finally have $x * \ln$ (real n) + \ln (fact n) \leq \ln (G (real (Suc n) + x))
using x n **by** (subst (asm) S_eq) (simp_all add: field_simps)
also have $x * \ln$ (real n) + \ln (fact n) = \ln (exp ($x * \ln$ (real n)) * fact n)
using x **by** (simp add: ln_mult)
finally have exp ($x * \ln$ (real n)) * fact n \leq G (real (Suc n) + x) **using** x
by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)
also have G (real (Suc n) + x) = pochhammer x (Suc n) * G x
using G_plus1 [of real (Suc n) + x for n] G_plus1 [of x] x
by (induction n) (simp_all add: pochhammer_Suc add_ac)
finally show Gamma_series x n \leq G x
using x **by** (simp add: field_simps pochhammer_pos Gamma_series_def)

qed

private lemma G_upper :

assumes x : $x > 0$ $x \leq 1$ **and** n : $n \geq 2$

shows G $x \leq$ Gamma_series x $n * (1 + x / \text{real } n)$

proof -

have ($\ln \circ G$) (real $n + x$) \leq (($\ln \circ G$) (real $n + 1$) -
($\ln \circ G$) (real n)) / (real $n + 1$ - (real n)) *
(real $n + x$) - real n) + ($\ln \circ G$) (real n)

using x n **by** (intro convex_onD_Icc' convex_on_subset[OF log_convex_G])

auto

hence S (of_nat n) (of_nat $n + x$) \leq S (of_nat n) (of_nat $n + 1$)

unfolding S_def **using** x **by** (simp add: field_simps)

also from n **have** S (of_nat n) (of_nat $n + 1$) = \ln (fact n) - \ln (fact ($n-1$))

by (subst (1 2) G_fact [symmetric]) (simp add: S_def add_ac of_nat_diff)

also have ... = \ln (fact n / (fact ($n-1$))) **using** n **by** (subst ln_div) simp_all

also from n **have** fact n / fact ($n-1$) = n **by** (cases n) simp_all

finally have \ln (G (real $n + x$)) \leq $x * \ln$ (real n) + \ln (fact ($n-1$))

using x n **by** (subst (asm) S_eq) (simp_all add: field_simps)

also have ... = \ln (exp ($x * \ln$ (real n)) * fact ($n-1$)) **using** x

by (simp add: ln_mult)

finally have G (real $n + x$) \leq exp ($x * \ln$ (real n)) * fact ($n-1$) **using** x

by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)

also have G (real $n + x$) = pochhammer x $n * G$ x

using G_plus1 [of real $n + x$ for n] x

by (induction n) (simp_all add: pochhammer_Suc add_ac)

finally have G $x \leq$ exp ($x * \ln$ (real n)) * fact ($n-1$) / pochhammer x n

using x **by** (simp add: field_simps pochhammer_pos)

also from n **have** fact ($n-1$) = fact n / n **by** (cases n) simp_all

also have exp ($x * \ln$ (real n)) * ... / pochhammer x n =

Gamma_series x $n * (1 + x / \text{real } n)$ **using** n x

by (simp add: Gamma_series_def divide_simps pochhammer_Suc)

finally show ?thesis .

qed

private lemma *G_eq_Gamma_aux*:

assumes $x: x > 0 \ x \leq 1$

shows $G \ x = \text{Gamma } x$

proof (rule *antisym*)

show $G \ x \geq \text{Gamma } x$

proof (rule *tendsto_upperbound*)

from *G_lower[of x]* **show** *eventually* $(\lambda n. \text{Gamma_series } x \ n \leq G \ x)$ *sequentially*

using x **by** (auto *intro: eventually_mono[OF eventually_ge_at_top[of 1::nat]]*)

qed (*simp_all add: Gamma_series_LIMSEQ*)

next

show $G \ x \leq \text{Gamma } x$

proof (rule *tendsto_lowerbound*)

have $(\lambda n. \text{Gamma_series } x \ n * (1 + x / \text{real } n)) \longrightarrow \text{Gamma } x * (1 + 0)$

by (rule *tendsto_intros real_tendsto_divide_at_top*)

Gamma_series_LIMSEQ filterlim_real_sequentially)+

thus $(\lambda n. \text{Gamma_series } x \ n * (1 + x / \text{real } n)) \longrightarrow \text{Gamma } x$ **by** *simp*

next

from *G_upper[of x]* **show** *eventually* $(\lambda n. \text{Gamma_series } x \ n * (1 + x / \text{real } n) \geq G \ x)$ *sequentially*

using x **by** (auto *intro: eventually_mono[OF eventually_ge_at_top[of 2::nat]]*)

qed *simp_all*

qed

theorem *Gamma_pos_real_unique*:

assumes $x: x > 0$

shows $G \ x = \text{Gamma } x$

proof -

have *G_eq*: $G \ (\text{real } n + x) = \text{Gamma} \ (\text{real } n + x)$ **if** $x \in \{0 <..1\}$ **for** $n \ x$ **using** *that*

proof (*induction n*)

case (*Suc n*)

from *Suc* **have** $x + \text{real } n > 0$ **by** *simp*

hence $x + \text{real } n \notin \mathbb{Z}_{\leq 0}$ **by** *auto*

with *Suc* **show** *?case* **using** *G_plus1[of real n + x]* *Gamma_plus1[of real n + x]*

by (auto *simp: add_ac*)

qed (*simp_all add: G_eq_Gamma_aux*)

show *?thesis*

proof (*cases frac x = 0*)

case *True*

hence $x = \text{of_int} \ (\text{floor } x)$ **by** (*simp add: frac_def*)

with x **have** *x_eq*: $x = \text{of_nat} \ (\text{nat} \ (\text{floor } x) - 1) + 1$ **by** *simp*

show *?thesis* **by** (*subst (1 2) x_eq, rule G_eq*) *simp_all*

next

```

    case False
    from assms have x_eq: x = of_nat (nat (floor x)) + frac x
    by (simp add: frac_def)
    have frac_le_1: frac x ≤ 1 unfolding frac_def by linarith
    show ?thesis
    by (subst (1 2) x_eq, rule G_eq, insert False frac_le_1) simp_all
  qed
qed
end

```

9.10.8 The Beta function

definition *Beta* where $Beta\ a\ b = Gamma\ a * Gamma\ b / Gamma\ (a + b)$

lemma *Beta_altdef*: $Beta\ a\ b = Gamma\ a * Gamma\ b * rGamma\ (a + b)$
 by (simp add: inverse_eq_divide Beta_def Gamma_def)

lemma *Beta_commute*: $Beta\ a\ b = Beta\ b\ a$
 unfolding Beta_def by (simp add: ac_simps)

lemma *has_field_derivative_Beta1* [derivative_intros]:
 assumes $x \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
 shows $((\lambda x. Beta\ x\ y) \text{ has_field_derivative } (Beta\ x\ y * (Digamma\ x - Digamma\ (x + y))))$
 (at x within A) unfolding Beta_altdef
 by (rule DERIV_cong, (rule derivative_intros assms)+) (simp add: algebra_simps)

lemma *Beta_pole1*: $x \in \mathbb{Z}_{\leq 0} \implies Beta\ x\ y = 0$
 by (auto simp add: Beta_def elim!: nonpos_Ints_cases')

lemma *Beta_pole2*: $y \in \mathbb{Z}_{\leq 0} \implies Beta\ x\ y = 0$
 by (auto simp add: Beta_def elim!: nonpos_Ints_cases')

lemma *Beta_zero*: $x + y \in \mathbb{Z}_{\leq 0} \implies Beta\ x\ y = 0$
 by (auto simp add: Beta_def elim!: nonpos_Ints_cases')

lemma *has_field_derivative_Beta2* [derivative_intros]:
 assumes $y \notin \mathbb{Z}_{\leq 0}$ $x + y \notin \mathbb{Z}_{\leq 0}$
 shows $((\lambda y. Beta\ x\ y) \text{ has_field_derivative } (Beta\ x\ y * (Digamma\ y - Digamma\ (x + y))))$
 (at y within A)
 using has_field_derivative_Beta1 [of $y\ x\ A$] assms by (simp add: Beta_commute add_ac)

theorem *Beta_plus1_plus1*:
 assumes $x \notin \mathbb{Z}_{\leq 0}$ $y \notin \mathbb{Z}_{\leq 0}$
 shows $Beta\ (x + 1)\ y + Beta\ x\ (y + 1) = Beta\ x\ y$
proof –

```

have Beta (x + 1) y + Beta x (y + 1) =
  (Gamma (x + 1) * Gamma y + Gamma x * Gamma (y + 1)) * rGamma
  ((x + y) + 1)
  by (simp add: Beta_altdef add_divide_distrib algebra_simps)
also have ... = (Gamma x * Gamma y) * ((x + y) * rGamma ((x + y) + 1))
  by (subst assms[THEN Gamma_plus1]) + (simp add: algebra_simps)
also from assms have ... = Beta x y unfolding Beta_altdef by (subst rGamma_plus1)
simp
finally show ?thesis .
qed

```

theorem Beta_plus1_left:

```

assumes x  $\notin$   $\mathbb{Z}_{\leq 0}$ 
shows (x + y) * Beta (x + 1) y = x * Beta x y
proof -
  have (x + y) * Beta (x + 1) y = Gamma (x + 1) * Gamma y * ((x + y) *
  rGamma ((x + y) + 1))
  unfolding Beta_altdef by (simp only: ac_simps)
  also have ... = x * Beta x y unfolding Beta_altdef
  by (subst assms[THEN Gamma_plus1] rGamma_plus1) + (simp only: ac_simps)
  finally show ?thesis .
qed

```

theorem Beta_plus1_right:

```

assumes y  $\notin$   $\mathbb{Z}_{\leq 0}$ 
shows (x + y) * Beta x (y + 1) = y * Beta x y
using Beta_plus1_left[of y x] assms by (simp_all add: Beta_commute add.commute)

```

lemma Gamma_Gamma_Beta:

```

assumes x + y  $\notin$   $\mathbb{Z}_{\leq 0}$ 
shows Gamma x * Gamma y = Beta x y * Gamma (x + y)
unfolding Beta_altdef using assms Gamma_eq_zero_iff[of x+y]
by (simp add: rGamma_inverse_Gamma)

```

9.10.9 Legendre duplication theorem

context

begin

private lemma Gamma_legendre_duplication_aux:

```

fixes z :: 'a :: Gamma
assumes z  $\notin$   $\mathbb{Z}_{\leq 0}$  z + 1/2  $\notin$   $\mathbb{Z}_{\leq 0}$ 
shows Gamma z * Gamma (z + 1/2) = exp ((1 - 2*z) * of_real (ln 2)) *
  Gamma (1/2) * Gamma (2*z)
proof -
  let ?powr =  $\lambda b$  a. exp (a * of_real (ln (of_nat b)))
  let ?h =  $\lambda n$ . (fact (n-1))2 / fact (2*n-1) * of_nat (2~(2*n)) *
    exp (1/2 * of_real (ln (real_of_nat n)))
  {

```

```

fix z :: 'a assume z: z  $\notin \mathbb{Z}_{\leq 0}$  z + 1/2  $\notin \mathbb{Z}_{\leq 0}$ 
let ?g =  $\lambda n. ?powr\ 2\ (2*z) * \text{Gamma\_series}'\ z\ n * \text{Gamma\_series}'\ (z + 1/2)$ 
n /
       $\text{Gamma\_series}'\ (2*z)\ (2*n)$ 
have eventually ( $\lambda n. ?g\ n = ?h\ n$ ) sequentially using eventually_gt_at_top
proof eventually_elim
  fix n :: nat assume n: n > 0
  let ?f = fact (n - 1) :: 'a and ?f' = fact (2*n - 1) :: 'a
  have A: exp t * exp t = exp (2*t :: 'a) for t by (subst exp_add [symmetric])
simp
  have A:  $\text{Gamma\_series}'\ z\ n * \text{Gamma\_series}'\ (z + 1/2)\ n = ?f^2 * ?powr$ 
n (2*z + 1/2) /
      ( $\text{pochhammer}\ z\ n * \text{pochhammer}\ (z + 1/2)\ n$ )
  by (simp add: Gamma_series'_def exp_add ring_distrib power2_eq_square
A mult_ac)
  have B:  $\text{Gamma\_series}'\ (2*z)\ (2*n) =$ 
      ?f' * ?powr 2 (2*z) * ?powr n (2*z) /
      ( $\text{of\_nat}\ (2^{(2*n)}) * \text{pochhammer}\ z\ n * \text{pochhammer}\ (z+1/2)$ 
n) using n
  by (simp add: Gamma_series'_def ln_mult exp_add ring_distrib pochhammer_double)
  from z have pochhammer z n  $\neq 0$  by (auto dest: pochhammer_eq_0_imp_nonpos_Int)
  moreover from z have pochhammer (z + 1/2) n  $\neq 0$  by (auto dest:
pochhammer_eq_0_imp_nonpos_Int)
  ultimately have ?powr 2 (2*z) * ( $\text{Gamma\_series}'\ z\ n * \text{Gamma\_series}'\ (z$ 
+ 1/2) n) /  $\text{Gamma\_series}'\ (2*z)\ (2*n) =$ 
      ?f^2 / ?f' *  $\text{of\_nat}\ (2^{(2*n)}) * (?powr\ n\ ((4*z + 1)/2) * ?powr\ n\ (-2*z))$ 
  using n unfolding A B by (simp add: field_split_simps exp_minus)
  also have ?powr n ((4*z + 1)/2) * ?powr n (-2*z) = ?powr n (1/2)
  by (simp add: algebra_simps exp_add [symmetric] add_divide_distrib)
  finally show ?g n = ?h n by (simp only: mult_ac)
qed

moreover from z double_in_nonpos_Ints_imp[of z] have 2 * z  $\notin \mathbb{Z}_{\leq 0}$  by
auto
hence ?g  $\longrightarrow ?powr\ 2\ (2*z) * \text{Gamma}\ z * \text{Gamma}\ (z+1/2) / \text{Gamma}$ 
(2*z)
using LIMSEQ_subseq_LIMSEQ[OF Gamma_series'_LIMSEQ, of (*) 2 2*z]
by (intro tendsto_intros Gamma_series'_LIMSEQ)
      (simp_all add: o_def strict_mono_def Gamma_eq_zero_iff)
ultimately have ?h  $\longrightarrow ?powr\ 2\ (2*z) * \text{Gamma}\ z * \text{Gamma}\ (z+1/2) /$ 
Gamma (2*z)
by (blast intro: Lim_transform_eventually)
} note lim = this

from assms double_in_nonpos_Ints_imp[of z] have z': 2 * z  $\notin \mathbb{Z}_{\leq 0}$  by auto
from fraction_not_in_ints[of 2 1] have (1/2 :: 'a)  $\notin \mathbb{Z}_{\leq 0}$ 
by (intro not_in_Ints_imp_not_in_nonpos_Ints) simp_all
with lim[of 1/2 :: 'a] have ?h  $\longrightarrow 2 * \text{Gamma}\ (1/2 :: 'a)$  by (simp add:

```

```

exp_of_real)
  from LIMSEQ_unique[OF this lim[OF assms]] z' show ?thesis
  by (simp add: field_split_simps Gamma_eq_zero_iff ring_distrib exp_diff
exp_of_real)
qed

```

The following lemma is somewhat annoying. With a little bit of complex analysis (Cauchy's integral theorem, to be exact), this would be completely trivial. However, we want to avoid depending on the complex analysis session at this point, so we prove it the hard way.

private lemma *Gamma_reflection_aux*:

```

defines h ≡ λz::complex. if z ∈ ℤ then 0 else
  (of_real pi * cot (of_real pi*z) + Digamma z - Digamma (1 - z))
defines a ≡ complex_of_real pi
obtains h' where continuous_on UNIV h' ∧ z. (h has_field_derivative (h' z))
(at z)
proof -
  define f where f n = a * of_real (cos_coeff (n+1) - sin_coeff (n+2)) for n
  define F where F z = (if z = 0 then 0 else (cos (a*z) - sin (a*z)/(a*z)) / z)
for z
  define g where g n = complex_of_real (sin_coeff (n+1)) for n
  define G where G z = (if z = 0 then 1 else sin (a*z)/(a*z)) for z
  have a_nz: a ≠ 0 unfolding a_def by simp

  have (λn. f n * (a*z)^(n)) sums (F z) ∧ (λn. g n * (a*z)^(n)) sums (G z)
  if abs (Re z) < 1 for z
  proof (cases z = 0; rule conjI)
    assume z ≠ 0
    note z = this that

    from z have sin_nz: sin (a*z) ≠ 0 unfolding a_def by (auto simp: sin_eq_0)
    have (λn. of_real (sin_coeff n) * (a*z)^(n)) sums (sin (a*z)) using sin_converges[of
a*z]
      by (simp add: scaleR_conv_of_real)
    from sums_split_initial_segment[OF this, of 1]
      have (λn. (a*z) * of_real (sin_coeff (n+1)) * (a*z)^(n)) sums (sin (a*z)) by
(simp add: mult_ac)
    from sums_mult[OF this, of inverse (a*z)] z a_nz
      have A: (λn. g n * (a*z)^(n)) sums (sin (a*z)/(a*z))
      by (simp add: field_simps g_def)
    with z show (λn. g n * (a*z)^(n)) sums (G z) by (simp add: G_def)
    from A z a_nz sin_nz have g_nz: (∑ n. g n * (a*z)^(n)) ≠ 0 by (simp add:
sums_iff g_def)

    have [simp]: sin_coeff (Suc 0) = 1 by (simp add: sin_coeff_def)
    from sums_split_initial_segment[OF sums_diff[OF cos_converges[of a*z] A],
of 1]
      have (λn. z * f n * (a*z)^(n)) sums (cos (a*z) - sin (a*z) / (a*z))
      by (simp add: mult_ac scaleR_conv_of_real ring_distrib f_def g_def)

```

```

from sums_mult[OF this, of inverse z] z assms
  show ( $\lambda n. f\ n * (a * z)^{\wedge} n$ ) sums (F z) by (simp add: divide_simps mult_ac
f_def F_def)
next
  assume z: z = 0
  have ( $\lambda n. f\ n * (a * z)^{\wedge} n$ ) sums f 0 using power_sums_zero[of f] z by simp
  with z show ( $\lambda n. f\ n * (a * z)^{\wedge} n$ ) sums (F z)
    by (simp add: f_def F_def sin_coeff_def cos_coeff_def)
  have ( $\lambda n. g\ n * (a * z)^{\wedge} n$ ) sums g 0 using power_sums_zero[of g] z by
simp
  with z show ( $\lambda n. g\ n * (a * z)^{\wedge} n$ ) sums (G z)
    by (simp add: g_def G_def sin_coeff_def cos_coeff_def)
qed
note sums = conjunct1[OF this] conjunct2[OF this]

define h2 where [abs_def]:
   $h2\ z = (\sum n. f\ n * (a * z)^{\wedge} n) / (\sum n. g\ n * (a * z)^{\wedge} n) + Digamma\ (1 + z) -$ 
Digamma (1 - z) for z
  define POWSER where [abs_def]: POWSER f z = ( $\sum n. f\ n * (z^{\wedge} n :: complex)$ )
for f z
  define POWSER' where [abs_def]: POWSER' f z = ( $\sum n. diff\ f\ n * (z^{\wedge} n)$ )
for f and z :: complex
  define h2' where [abs_def]:
   $h2'\ z = a * (POWSER\ g\ (a * z) * POWSER'\ f\ (a * z) - POWSER\ f\ (a * z) *$ 
POWSER' g (a * z)) /
  ( $(POWSER\ g\ (a * z))^{\wedge} 2 + Polygamma\ 1\ (1 + z) + Polygamma\ 1\ (1 - z)$ ) for
z

have h_eq: h t = h2 t if abs (Re t) < 1 for t
proof -
  from that have t: t ∈  $\mathbb{Z}$   $\longleftrightarrow$  t = 0 by (auto elim!: Ints_cases)
  hence h t = a * cot (a * t) - 1 / t + Digamma (1 + t) - Digamma (1 - t)
    unfolding h_def using Digamma_plus1[of t] by (force simp: field_simps
a_def)
  also have a * cot (a * t) - 1 / t = (F t) / (G t)
    using t by (auto simp add: divide_simps sin_eq_0 cot_def a_def F_def
G_def)
  also have ... = ( $\sum n. f\ n * (a * t)^{\wedge} n$ ) / ( $\sum n. g\ n * (a * t)^{\wedge} n$ )
    using sums[of t] that by (simp add: sums_iff)
  finally show h t = h2 t by (simp only: h2_def)
qed

let ?A = {z. abs (Re z) < 1}
have open ({z. Re z < 1} ∩ {z. Re z > -1})
  using open_halfspace_Re_gt open_halfspace_Re_lt by auto
also have ({z. Re z < 1} ∩ {z. Re z > -1}) = {z. abs (Re z) < 1} by auto
finally have open_A: open ?A .
hence [simp]: interior ?A = ?A by (simp add: interior_open)

```

```

have summable_f: summable ( $\lambda n. f\ n * z^{\wedge}n$ ) for z
  by (rule power_inside, rule sums_summable, rule sums[of i * of_real (norm
z + 1) / a])
  (simp_all add: norm_mult a_def del: of_real_add)
have summable_g: summable ( $\lambda n. g\ n * z^{\wedge}n$ ) for z
  by (rule power_inside, rule sums_summable, rule sums[of i * of_real (norm
z + 1) / a])
  (simp_all add: norm_mult a_def del: of_real_add)
have summable_fg': summable ( $\lambda n. \text{diffs}\ f\ n * z^{\wedge}n$ ) summable ( $\lambda n. \text{diffs}\ g\ n *
z^{\wedge}n$ ) for z
  by (intro termdiff_converges_all summable_f summable_g)+
have (POWSER f has_field_derivative (POWSER' f z)) (at z)
  (POWSER g has_field_derivative (POWSER' g z)) (at z) for z
  unfolding POWSER_def POWSER'_def
  by (intro termdiffs_strong_converges_everywhere summable_f summable_g)+
note derivs = this[THEN DERIV_chain2[OF _ DERIV_cmult[OF DERIV_ident]],
unfolded POWSER_def]
have isCont (POWSER f) z isCont (POWSER g) z isCont (POWSER' f) z
isCont (POWSER' g) z
  for z unfolding POWSER_def POWSER'_def
  by (intro isCont_power_converges_everywhere summable_f summable_g summable_fg')+
note cont = this[THEN isCont_o2[rotated], unfolded POWSER_def POWSER'_def]

{
  fix z :: complex assume z: abs (Re z) < 1
  define d where d = i * of_real (norm z + 1)
  have d: abs (Re d) < 1 norm z < norm d by (simp_all add: d_def norm_mult
del: of_real_add)
  have eventually ( $\lambda z. h\ z = h2\ z$ ) (nhds z)
    using eventually_nhds_in_nhd[of z ?A] using h_eq z
    by (auto elim!: eventually_mono)

  moreover from sums(2)[OF z] z have nz: ( $\sum n. g\ n * (a * z)^{\wedge}n$ )  $\neq 0$ 
    unfolding G_def by (auto simp: sums_iff sin_eq_0 a_def)
  have A:  $z \in \mathbb{Z} \longleftrightarrow z = 0$  using z by (auto elim!: Ints_cases)
  have no_int:  $1 + z \in \mathbb{Z} \longleftrightarrow z = 0$  using z Ints_diff[of 1+z 1] A
    by (auto elim!: nonpos_Ints_cases)
  have no_int':  $1 - z \in \mathbb{Z} \longleftrightarrow z = 0$  using z Ints_diff[of 1 1-z] A
    by (auto elim!: nonpos_Ints_cases)
  from no_int no_int' have no_int:  $1 - z \notin \mathbb{Z}_{\leq 0} \wedge 1 + z \notin \mathbb{Z}_{\leq 0}$  by auto
  have (h2 has_field_derivative h2' z) (at z) unfolding h2_def
  by (rule DERIV_cong, (rule derivative_intros refl derivs[unfolded POWSER_def]
nz no_int)+)
    (auto simp: h2'_def POWSER_def field_simps power2_eq_square)
  ultimately have deriv: (h has_field_derivative h2' z) (at z)
    by (subst DERIV_cong_ev[OF refl _ refl])

  from sums(2)[OF z] z have ( $\sum n. g\ n * (a * z)^{\wedge}n$ )  $\neq 0$ 
    unfolding G_def by (auto simp: sums_iff a_def sin_eq_0)

```

hence $isCont\ h2'\ z$ **using** $no_int\ unfolding\ h2'_def[abs_def]\ POWSER_def\ POWSER'_def$
by ($intro\ continuous_intros\ cont$
 $continuous_on_compose2[OF\ _continuous_on_Polygamma[of\ \{z.\ Re$
 $z > 0\}]]$) $auto$
note $deriv\ and\ this$
} note $A = this$

interpret $h: periodic_fun_simple'\ h$

proof

fix $z :: complex$

show $h\ (z + 1) = h\ z$

proof ($cases\ z \in \mathbb{Z}$)

assume $z: z \notin \mathbb{Z}$

hence $A: z + 1 \notin \mathbb{Z}\ z \neq 0$ **using** $Ints_diff[of\ z+1\ 1]$ **by** $auto$

hence $Digamma\ (z + 1) - Digamma\ (-z) = Digamma\ z - Digamma\ (-z + 1)$

by ($subst\ (1\ 2)\ Digamma_plus1$) $simp_all$

with A **show** $h\ (z + 1) = h\ z$

by ($simp\ add: h_def\ sin_plus_pi\ cos_plus_pi\ ring_distrib\ cot_def$)

qed ($simp\ add: h_def$)

qed

have $h2'_eq: h2'\ (z - 1) = h2'\ z$ **if** $z: Re\ z > 0\ Re\ z < 1$ **for** z

proof -

have ($(\lambda z. h\ (z - 1))\ has_field_derivative\ h2'\ (z - 1)$) ($at\ z$)

by ($rule\ DERIV_cong, rule\ DERIV_chain'[OF\ _A(1)]$)

($insert\ z, auto\ intro!: derivative_eq_intros$)

hence ($h\ has_field_derivative\ h2'\ (z - 1)$) ($at\ z$) **by** ($subst\ (asm)\ h.minus_1$)

moreover from z **have** ($h\ has_field_derivative\ h2'\ z$) ($at\ z$) **by** ($intro\ A$)
 $simp_all$

ultimately show $h2'\ (z - 1) = h2'\ z$ **by** ($rule\ DERIV_unique$)

qed

define $h2''$ **where** $h2''\ z = h2'\ (z - of_int\ \lfloor Re\ z \rfloor)$ **for** z

have $deriv: (h\ has_field_derivative\ h2''\ z)$ ($at\ z$) **for** z

proof -

fix $z :: complex$

have $B: \lfloor Re\ z - real_of_int\ \lfloor Re\ z \rfloor \rfloor < 1$ **by** $linarith$

have ($(\lambda t. h\ (t - of_int\ \lfloor Re\ z \rfloor))\ has_field_derivative\ h2''\ z$) ($at\ z$)

unfolding $h2''_def$ **by** ($rule\ DERIV_cong, rule\ DERIV_chain'[OF\ _A(1)]$)

($insert\ B, auto\ intro!: derivative_intros$)

thus ($h\ has_field_derivative\ h2''\ z$) ($at\ z$) **by** ($simp\ add: h.minus_of_int$)

qed

have $cont: continuous_on\ UNIV\ h2''$

proof ($intro\ continuous_at_imp_continuous_on\ ballI$)

fix $z :: complex$

define r **where** $r = \lfloor Re\ z \rfloor$


```

define A where A = {t. of_int r - 1 < Re t ∧ Re t < of_int r + 1}
have continuous_on A (λt. h2' (t - of_int r)) unfolding A_def
  by (intro continuous_at_imp_continuous_on isCont_o2[OF _ A(2)] ballI
continuous_intros)
  (simp_all add: abs_real_def)
moreover have h2'' t = h2' (t - of_int r) if t: t ∈ A for t
proof (cases Re t ≥ of_int r)
  case True
    from t have of_int r - 1 < Re t Re t < of_int r + 1 by (simp_all add:
A_def)
    with True have [Re t] = [Re z] unfolding r_def by linarith
    thus ?thesis by (auto simp: r_def h2''_def)
  next
    case False
    from t have t: of_int r - 1 < Re t Re t < of_int r + 1 by (simp_all add:
A_def)
    with False have t': [Re t] = [Re z] - 1 unfolding r_def by linarith
    moreover from t False have h2' (t - of_int r + 1 - 1) = h2' (t - of_int
r + 1)
      by (intro h2'_eq) simp_all
    ultimately show ?thesis by (auto simp: r_def h2''_def algebra_simps t')
  qed
ultimately have continuous_on A h2'' by (subst continuous_on_cong[OF
refl])
moreover {
  have open ({t. of_int r - 1 < Re t} ∩ {t. of_int r + 1 > Re t})
    by (intro open_Int open_halfspace_Re_gt open_halfspace_Re_lt)
  also have {t. of_int r - 1 < Re t} ∩ {t. of_int r + 1 > Re t} = A
    unfolding A_def by blast
  finally have open A .
}
ultimately have C: isCont h2'' t if t ∈ A for t using that
  by (subst (asm) continuous_on_eq_continuous_at) auto
have of_int r - 1 < Re z Re z < of_int r + 1 unfolding r_def by linarith+
thus isCont h2'' z by (intro C) (simp_all add: A_def)
qed

from that[OF cont_deriv] show ?thesis .
qed

```

lemma Gamma_reflection_complex:

fixes z :: complex

shows Gamma z * Gamma (1 - z) = of_real pi / sin (of_real pi * z)

proof -

let ?g = λz::complex. Gamma z * Gamma (1 - z) * sin (of_real pi * z)

define g **where** [abs_def]: g z = (if z ∈ ℤ then of_real pi else ?g z) **for** z :: complex

let ?h = λz::complex. (of_real pi * cot (of_real pi*z) + Digamma z - Digamma (1 - z))

```

define h where [abs_def]:  $h\ z = (if\ z \in \mathbb{Z}\ then\ 0\ else\ ?h\ z)$  for  $z :: complex$ 

— g is periodic with period 1.
interpret g: periodic_fun_simple' g
proof
  fix  $z :: complex$ 
  show  $g\ (z + 1) = g\ z$ 
  proof (cases  $z \in \mathbb{Z}$ )
    case False
      hence  $z * g\ z = z * Beta\ z\ (-z + 1) * sin\ (of\_real\ pi * z)$  by (simp add:
g_def Beta_def)
      also have  $z * Beta\ z\ (-z + 1) = (z + 1 + -z) * Beta\ (z + 1)\ (-z + 1)$ 
        using False Ints_diff[of 1 1 -z] nonpos_Ints_subset_Ints
        by (subst Beta_plus1_left [symmetric]) auto
      also have  $\dots * sin\ (of\_real\ pi * z) = z * (Beta\ (z + 1)\ (-z) * sin\ (of\_real\ pi * (z + 1)))$ 
        using False Ints_diff[of z+1 1] Ints_minus[of -z] nonpos_Ints_subset_Ints
        by (subst Beta_plus1_right) (auto simp: ring_distrib sin_plus_pi)
      also from False have  $Beta\ (z + 1)\ (-z) * sin\ (of\_real\ pi * (z + 1)) = g\ (z + 1)$ 
        using Ints_diff[of z+1 1] by (auto simp: g_def Beta_def)
      finally show  $g\ (z + 1) = g\ z$  using False by (subst (asm) mult_left_cancel)
auto
    qed (simp add: g_def)
  qed

— g is entire.
have g_g': (g has_field_derivative ( $h\ z * g\ z$ )) (at z) for  $z :: complex$ 
proof (cases  $z \in \mathbb{Z}$ )
  let  $?h' = \lambda z. Beta\ z\ (1 - z) * ((Digamma\ z - Digamma\ (1 - z)) * sin\ (z * of\_real\ pi) +$ 
     $of\_real\ pi * cos\ (z * of\_real\ pi))$ 
    case False
      from False have eventually ( $\lambda t. t \in UNIV - \mathbb{Z}$ ) (nhds  $z$ )
        by (intro eventually_nhds_in_open) (auto simp: open_Diff)
      hence eventually ( $\lambda t. g\ t = ?h' t$ ) (nhds  $z$ ) by eventually_elim (simp add:
g_def)
      moreover {
        from False Ints_diff[of 1 1-z] have  $1 - z \notin \mathbb{Z}$  by auto
        hence ( $?g\ has\_field\_derivative\ ?h'\ z$ ) (at z) using nonpos_Ints_subset_Ints
          by (auto intro!: derivative_eq_intros simp: algebra_simps Beta_def)
        also from False have  $sin\ (of\_real\ pi * z) \neq 0$  by (subst sin_eq_0) auto
        hence  $?h'\ z = h\ z * g\ z$ 
          using False unfolding g_def h_def cot_def by (simp add: field_simps Beta_def)
        finally have ( $?g\ has\_field\_derivative\ (h\ z * g\ z)$ ) (at z) .
      }
    ultimately show thesis by (subst DERIV_cong_ev[OF refl _ refl])
  next

```

```

case True
then obtain n where z: z = of_int n by (auto elim!: Ints_cases)
let ?t = (λz::complex. if z = 0 then 1 else sin z / z) ∘ (λz. of_real pi * z)
have deriv_0: (g has_field_derivative 0) (at 0)
proof (subst DERIV_cong_ev[OF refl _ refl])
  show eventually (λz. g z = of_real pi * Gamma (1 + z) * Gamma (1 - z)
* ?t z) (nhds 0)
  using eventually_nhds_ball[OF zero_less_one, of 0::complex]
proof eventually_elim
  fix z :: complex assume z: z ∈ ball 0 1
  show g z = of_real pi * Gamma (1 + z) * Gamma (1 - z) * ?t z
  proof (cases z = 0)
    assume z': z ≠ 0
    with z have z'': z ∉ ℤ<0 z ∉ ℤ by (auto elim!: Ints_cases)
    from Gamma_plus1[OF this(1)] have Gamma z = Gamma (z + 1) / z
by simp
  with z'' z' show ?thesis by (simp add: g_def ac_simps)
  qed (simp add: g_def)
  qed
  have (?t has_field_derivative (0 * of_real pi)) (at 0)
  using has_field_derivative_sin_z_over_z[of UNIV :: complex set]
  by (intro DERIV_chain) simp_all
  thus ((λz. of_real pi * Gamma (1 + z) * Gamma (1 - z) * ?t z) has_field_derivative
0) (at 0)
  by (auto intro!: derivative_eq_intros simp: o_def)
  qed

  have ((g ∘ (λx. x - of_int n)) has_field_derivative 0 * 1) (at (of_int n))
  using deriv_0 by (intro DERIV_chain) (auto intro!: derivative_eq_intros)
  also have g ∘ (λx. x - of_int n) = g by (intro ext) (simp add: g.minus_of_int)
  finally show (g has_field_derivative (h z * g z)) (at z) by (simp add: z h_def)
  qed

  have g_eq: g (z/2) * g ((z+1)/2) = Gamma (1/2)2 * g z if Re z > -1 Re z
< 2 for z
  proof (cases z ∈ ℤ)
    case True
    with that have z = 0 ∨ z = 1 by (force elim!: Ints_cases)
    moreover have g 0 * g (1/2) = Gamma (1/2)2 * g 0
    using fraction_not_in_ints[where 'a = complex, of 2 1] by (simp add: g_def
power2_eq_square)
    moreover have g (1/2) * g 1 = Gamma (1/2)2 * g 1
    using fraction_not_in_ints[where 'a = complex, of 2 1]
    by (simp add: g_def power2_eq_square Beta_def algebra_simps)
    ultimately show ?thesis by force
  next
  case False
  hence z: z/2 ∉ ℤ (z+1)/2 ∉ ℤ using Ints_diff[of z+1 1] by (auto elim!:
Ints_cases)

```

hence z' : $z/2 \notin \mathbb{Z}_{\leq 0}$ $(z+1)/2 \notin \mathbb{Z}_{\leq 0}$ by (auto elim!: nonpos_Ints_cases)
 from z have $1-z/2 \notin \mathbb{Z}$ $1-((z+1)/2) \notin \mathbb{Z}$
 using Ints_diff[of 1 1-z/2] Ints_diff[of 1 1-((z+1)/2)] by auto
 hence z'' : $1-z/2 \notin \mathbb{Z}_{\leq 0}$ $1-((z+1)/2) \notin \mathbb{Z}_{\leq 0}$ by (auto elim!: nonpos_Ints_cases)
 from z have $g(z/2) * g((z+1)/2) =$
 $(\text{Gamma}(z/2) * \text{Gamma}((z+1)/2)) * (\text{Gamma}(1-z/2) * \text{Gamma}(1-((z+1)/2)))$
 *
 $(\sin(\text{of_real } \pi * z/2) * \sin(\text{of_real } \pi * (z+1)/2))$
 by (simp add: g_def)
 also from z' Gamma_legendre_duplication_aux[of z/2]
 have $\text{Gamma}(z/2) * \text{Gamma}((z+1)/2) = \exp((1-z) * \text{of_real}(\ln 2)) * \text{Gamma}(1/2) * \text{Gamma } z$
 by (simp add: add_divide_distrib)
 also from z'' Gamma_legendre_duplication_aux[of 1-(z+1)/2]
 have $\text{Gamma}(1-z/2) * \text{Gamma}(1-(z+1)/2) =$
 $\text{Gamma}(1-z) * \text{Gamma}(1/2) * \exp(z * \text{of_real}(\ln 2))$
 by (simp add: add_divide_distrib ac_simps)
 finally have $g(z/2) * g((z+1)/2) = \text{Gamma}(1/2)^2 * (\text{Gamma } z * \text{Gamma}(1-z) *$
 $(2 * (\sin(\text{of_real } \pi * z/2) * \sin(\text{of_real } \pi * (z+1)/2))))$
 by (simp add: add_ac power2_eq_square exp_add ring_distrib exp_diff exp_of_real)
 also have $\sin(\text{of_real } \pi * (z+1)/2) = \cos(\text{of_real } \pi * z/2)$
 using cos_sin_eq[of - of_real pi * z/2, symmetric]
 by (simp add: ring_distrib add_divide_distrib ac_simps)
 also have $2 * (\sin(\text{of_real } \pi * z/2) * \cos(\text{of_real } \pi * z/2)) = \sin(\text{of_real } \pi * z)$
 * z
 by (subst sin_times_cos) (simp add: field_simps)
 also have $\text{Gamma } z * \text{Gamma}(1-z) * \sin(\text{complex_of_real } \pi * z) = g z$
 using ⟨ $z \notin \mathbb{Z}$ ⟩ by (simp add: g_def)
 finally show ?thesis .
 qed
 have g_eq: $g(z/2) * g((z+1)/2) = \text{Gamma}(1/2)^2 * g z$ for z
 proof -
 define r where $r = \lfloor \text{Re } z / 2 \rfloor$
 have $\text{Gamma}(1/2)^2 * g z = \text{Gamma}(1/2)^2 * g(z - \text{of_int}(2*r))$ by
 (simp only: g.minus_of_int)
 also have $\text{of_int}(2*r) = 2 * \text{of_int } r$ by simp
 also have $\text{Re } z - 2 * \text{of_int } r > -1$ $\text{Re } z - 2 * \text{of_int } r < 2$ unfolding
 r_def by linarith+
 hence $\text{Gamma}(1/2)^2 * g(z - 2 * \text{of_int } r) =$
 $g((z - 2 * \text{of_int } r)/2) * g((z - 2 * \text{of_int } r + 1)/2)$
 unfolding r_def by (intro g_eq[symmetric]) simp_all
 also have $(z - 2 * \text{of_int } r) / 2 = z/2 - \text{of_int } r$ by simp
 also have $g \dots = g(z/2)$ by (rule g.minus_of_int)
 also have $(z - 2 * \text{of_int } r + 1) / 2 = (z + 1)/2 - \text{of_int } r$ by simp
 also have $g \dots = g((z+1)/2)$ by (rule g.minus_of_int)
 finally show ?thesis ..
 qed

```

have g_nz [simp]:  $g\ z \neq 0$  for  $z :: \text{complex}$ 
unfolding g_def using Ints_diff[of 1 1 - z]
  by (auto simp: Gamma_eq_zero_iff sin_eq_0 dest!: nonpos_Ints_Int)

have h_eq:  $h\ z = (h\ (z/2) + h\ ((z+1)/2)) / 2$  for  $z$ 
proof -
  have (( $\lambda t. g\ (t/2) * g\ ((t+1)/2)$ ) has_field_derivative
    ( $g\ (z/2) * g\ ((z+1)/2)$ ) * (( $h\ (z/2) + h\ ((z+1)/2)$ ) / 2)) (at z)
  by (auto intro!: derivative_eq_intros g_g'[THEN DERIV_chain2] simp:
field_simps)
  hence (( $\lambda t. \text{Gamma}\ (1/2)^2 * g\ t$ ) has_field_derivative
     $\text{Gamma}\ (1/2)^2 * g\ z * ((h\ (z/2) + h\ ((z+1)/2)) / 2)$ ) (at z)
  by (subst (1 2) g_eq[symmetric]) simp
  from DERIV_cmult[OF this, of inverse (( $\text{Gamma}\ (1/2)$ )2)]
  have ( $g$  has_field_derivative ( $g\ z * ((h\ (z/2) + h\ ((z+1)/2))/2$ )) (at z)
  using fraction_not_in_ints[where 'a = complex, of 2 1]
  by (simp add: divide_simps Gamma_eq_zero_iff not_in_Ints_imp_not_in_nonpos_Ints)
  moreover have ( $g$  has_field_derivative ( $g\ z * h\ z$ )) (at z)
  using g_g'[of z] by (simp add: ac_simps)
  ultimately have  $g\ z * h\ z = g\ z * ((h\ (z/2) + h\ ((z+1)/2))/2$ 
  by (intro DERIV_unique)
  thus  $h\ z = (h\ (z/2) + h\ ((z+1)/2)) / 2$  by simp
qed

obtain h' where h'_cont: continuous_on UNIV h' and
  h_h':  $\bigwedge z. (h$  has_field_derivative h' z) (at z)
  unfolding h_def by (erule Gamma_reflection_aux)

have h'_eq:  $h'\ z = (h'\ (z/2) + h'\ ((z+1)/2)) / 4$  for  $z$ 
proof -
  have (( $\lambda t. (h\ (t/2) + h\ ((t+1)/2)) / 2$ ) has_field_derivative
    (( $h'\ (z/2) + h'\ ((z+1)/2)$ ) / 4)) (at z)
  by (fastforce intro!: derivative_eq_intros h_h'[THEN DERIV_chain2])
  hence ( $h$  has_field_derivative (( $h'\ (z/2) + h'\ ((z+1)/2)$ )/4)) (at z)
  by (subst (asm) h_eq[symmetric])
  from h_h' and this show  $h'\ z = (h'\ (z/2) + h'\ ((z+1)/2)) / 4$  by (rule
DERIV_unique)
qed

have h'_zero:  $h'\ z = 0$  for  $z$ 
proof -
  define m where  $m = \max 1 |Re\ z|$ 
  define B where  $B = \{t. \text{abs}\ (Re\ t) \leq m \wedge \text{abs}\ (Im\ t) \leq \text{abs}\ (Im\ z)\}$ 
  have closed ( $\{t. Re\ t \geq -m\} \cap \{t. Re\ t \leq m\} \cap$ 
     $\{t. Im\ t \geq -|Im\ z|\} \cap \{t. Im\ t \leq |Im\ z|\}$ )
  (is closed ?B) by (intro closed_Int closed_halfspace_Re_ge closed_halfspace_Re_le
    closed_halfspace_Im_ge closed_halfspace_Im_le)
  also have ?B = B unfolding B_def by fastforce

```

```

finally have closed B .
moreover have bounded B unfolding bounded_iff
proof (intro ballI exI)
  fix t assume t: t ∈ B
  have norm t ≤ |Re t| + |Im t| by (rule cmod_le)
  also from t have |Re t| ≤ m unfolding B_def by blast
  also from t have |Im t| ≤ |Im z| unfolding B_def by blast
  finally show norm t ≤ m + |Im z| by - simp
qed
ultimately have compact: compact B by (subst compact_eq_bounded_closed)
blast

define M where M = (SUP z∈B. norm (h' z))
have compact (h' ` B)
  by (intro compact_continuous_image continuous_on_subset[OF h'_cont]
compact) blast+
  hence bdd: bdd_above ((λz. norm (h' z)) ` B)
  using bdd_above_norm[of h' ` B] by (simp add: image_comp o_def com-
pact_imp_bounded)
  have norm (h' z) ≤ M unfolding M_def by (intro cSUP_upper bdd) (simp_all
add: B_def m_def)
  also have M ≤ M/2
  proof (subst M_def, subst cSUP_le_iff)
    have z ∈ B unfolding B_def m_def by simp
    thus B ≠ {} by auto
  next
  show ∀ z∈B. norm (h' z) ≤ M/2
  proof
    fix t :: complex assume t: t ∈ B
    from h'_eq[of t] t have h' t = (h' (t/2) + h' ((t+1)/2)) / 4 by (simp)
    also have norm ... = norm (h' (t/2) + h' ((t+1)/2)) / 4 by simp
    also have norm (h' (t/2) + h' ((t+1)/2)) ≤ norm (h' (t/2)) + norm (h'
((t+1)/2))
      by (rule norm_triangle_ineq)
    also from t have abs (Re ((t + 1)/2)) ≤ m unfolding m_def B_def by
auto
    with t have t/2 ∈ B (t+1)/2 ∈ B unfolding B_def by auto
    hence norm (h' (t/2)) + norm (h' ((t+1)/2)) ≤ M + M unfolding M_def
      by (intro add_mono cSUP_upper bdd) (auto simp: B_def)
    also have (M + M) / 4 = M / 2 by simp
    finally show norm (h' t) ≤ M/2 by - simp_all
  qed
qed (insert bdd, auto)
hence M ≤ 0 by simp
finally show h' z = 0 by simp
qed
have h_h'_2: (h has_field_derivative 0) (at z) for z
  using h_h'[of z] h'_zero[of z] by simp

```

```

have g_real:  $g z \in \mathbb{R}$  if  $z \in \mathbb{R}$  for  $z$ 
  unfolding g_def using that by (auto intro!: Reals_mult Gamma_complex_real)
have h_real:  $h z \in \mathbb{R}$  if  $z \in \mathbb{R}$  for  $z$ 
  unfolding h_def using that by (auto intro!: Reals_mult Reals_add Reals_diff
Polygamma_Real)
have g_nz:  $g z \neq 0$  for  $z$  unfolding g_def using Ints_diff[of 1 1-z]
  by (auto simp: Gamma_eq_zero_iff sin_eq_0)

from h'_zero h'_2 have  $\exists c. \forall z \in UNIV. h z = c$ 
  by (intro has_field_derivative_zero_constant) (simp_all add: dist_0_norm)
then obtain c where  $c: \bigwedge z. h z = c$  by auto
have  $\exists u. u \in \text{closed\_segment } 0\ 1 \wedge \text{Re } (g\ 1) - \text{Re } (g\ 0) = \text{Re } (h\ u * g\ u * (1 - 0))$ 
  by (intro complex_mvt_line g_g')
then obtain u where  $u: u \in \text{closed\_segment } 0\ 1 \wedge \text{Re } (g\ 1) - \text{Re } (g\ 0) = \text{Re } (h\ u * g\ u)$ 
  by auto
from u(1) have u':  $u \in \mathbb{R}$  unfolding closed_segment_def
  by (auto simp: scaleR_conv_of_real)
from u' g_real[of u] g_nz[of u] have  $\text{Re } (g\ u) \neq 0$  by (auto elim!: Reals_cases)
with u(2) c[of u] g_real[of u] g_nz[of u] u'
  have  $\text{Re } c = 0$  by (simp add: complex_is_Real_iff g.of_1)
with h_real[of 0] c[of 0] have  $c = 0$  by (auto elim!: Reals_cases)
with c have A:  $h z * g z = 0$  for  $z$  by simp
hence (g has_field_derivative 0) (at z) for z using g_g'[of z] by simp
hence  $\exists c'. \forall z \in UNIV. g z = c'$  by (intro has_field_derivative_zero_constant)
simp_all
then obtain c' where  $c: \bigwedge z. g z = c'$  by (force)
from this[of 0] have  $c' = \pi$  unfolding g_def by simp
with c have  $g z = \pi$  by simp

show ?thesis
proof (cases  $z \in \mathbb{Z}$ )
case False
  with  $\langle g z = \pi \rangle$  show ?thesis by (auto simp: g_def divide_simps)
next
case True
  then obtain n where  $z = \text{of\_int } n$  by (elim Ints_cases)
  with  $\text{sin\_eq\_0}[of \text{of\_real } \pi * z]$  have  $\text{sin } (\text{of\_real } \pi * z) = 0$  by force
  moreover have  $\text{of\_int } (1 - n) \in \mathbb{Z}_{\leq 0}$  if  $n > 0$  using that by (intro nonpos_Ints_of_int) simp
  ultimately show ?thesis using n
    by (cases  $n \leq 0$ ) (auto simp: Gamma_eq_zero_iff nonpos_Ints_of_int)
qed
qed

lemma rGamma_reflection_complex:
   $rGamma\ z * rGamma\ (1 - z :: \text{complex}) = \text{sin } (\text{of\_real } \pi * z) / \text{of\_real } \pi$ 
  using Gamma_reflection_complex[of z]

```

3092

by (simp add: Gamma_def field_split_simps split: if_split_asm)

lemma *rGamma_reflection_complex'*:

$rGamma\ z * rGamma\ (-z :: complex) = -z * \sin\ (of_real\ pi * z) / of_real\ pi$

proof –

have $rGamma\ z * rGamma\ (-z) = -z * (rGamma\ z * rGamma\ (1 - z))$

using *rGamma_plus1*[of $-z$, *symmetric*] **by** *simp*

also have $rGamma\ z * rGamma\ (1 - z) = \sin\ (of_real\ pi * z) / of_real\ pi$

by (rule *rGamma_reflection_complex*)

finally show *?thesis* **by** *simp*

qed

lemma *Gamma_reflection_complex'*:

$Gamma\ z * Gamma\ (-z :: complex) = - of_real\ pi / (z * \sin\ (of_real\ pi * z))$

using *rGamma_reflection_complex'*[of z] **by** (force simp add: *Gamma_def field_split_simps*)

lemma *Gamma_one_half_real*: $Gamma\ (1/2 :: real) = \sqrt{pi}$

proof –

from *Gamma_reflection_complex*[of $1/2$] *fraction_not_in_ints*[where $'a = complex, of\ 2\ 1$]

have $Gamma\ (1/2 :: complex)^2 = of_real\ pi$ **by** (simp add: *power2_eq_square*)

hence $of_real\ pi = Gamma\ (complex_of_real\ (1/2))^2$ **by** *simp*

also have $\dots = of_real\ ((Gamma\ (1/2))^2)$ **by** (subst *Gamma_complex_of_real simp_all*)

finally have $Gamma\ (1/2)^2 = pi$ **by** (subst (*asm*) *of_real_eq_iff*) *simp_all*

moreover have $Gamma\ (1/2 :: real) \geq 0$ **using** *Gamma_real_pos*[of $1/2$] **by** *simp*

ultimately show *?thesis* **by** (rule *real_sqrt_unique* [*symmetric*])

qed

lemma *Gamma_one_half_complex*: $Gamma\ (1/2 :: complex) = of_real\ (\sqrt{pi})$

proof –

have $Gamma\ (1/2 :: complex) = Gamma\ (of_real\ (1/2))$ **by** *simp*

also have $\dots = of_real\ (\sqrt{pi})$ **by** (simp only: *Gamma_complex_of_real Gamma_one_half_real*)

finally show *?thesis* .

qed

theorem *Gamma_legendre_duplication*:

fixes $z :: complex$

assumes $z \notin \mathbb{Z}_{\leq 0}$ $z + 1/2 \notin \mathbb{Z}_{\leq 0}$

shows $Gamma\ z * Gamma\ (z + 1/2) =$

$exp\ ((1 - 2*z) * of_real\ (\ln\ 2)) * of_real\ (\sqrt{pi}) * Gamma\ (2*z)$

using *Gamma_legendre_duplication_aux*[*OF assms*] **by** (simp add: *Gamma_one_half_complex*)

end

9.10.10 Limits and residues

The inverse of the Gamma function has simple zeros:

lemma *rGamma_zeros*:

$(\lambda z. rGamma\ z / (z + of_nat\ n)) - (-\ of_nat\ n) \rightarrow ((-1)^n * fact\ n :: 'a :: Gamma)$

proof (*subst tendsto_cong*)

let $?f = \lambda z. pochhammer\ z\ n * rGamma\ (z + of_nat\ (Suc\ n)) :: 'a$

from *eventually_at_ball'*[*OF zero_less_one, of - of_nat n :: 'a UNIV*]

show *eventually* $(\lambda z. rGamma\ z / (z + of_nat\ n) = ?f\ z)$ (*at (- of_nat n)*)

by (*subst pochhammer_rGamma[of _ Suc n]*)

(auto elim!: eventually_mono simp: field_split_simps pochhammer_rec' eq_neg_iff_add_eq_0)

have *isCont* $?f\ (-\ of_nat\ n)$ **by** (*intro continuous_intros*)

thus $?f\ (-\ of_nat\ n) \rightarrow (-1)^n * fact\ n$ **unfolding** *isCont_def*

by (*simp add: pochhammer_same*)

qed

The simple zeros of the inverse of the Gamma function correspond to simple poles of the Gamma function, and their residues can easily be computed from the limit we have just proven:

lemma *Gamma_poles*: *filterlim Gamma at_infinity (at (- of_nat n :: 'a :: Gamma))*

proof –

from *eventually_at_ball'*[*OF zero_less_one, of - of_nat n :: 'a UNIV*]

have *eventually* $(\lambda z. rGamma\ z \neq (0 :: 'a))$ (*at (- of_nat n)*)

by (*auto elim!: eventually_mono nonpos_Ints_cases'*)

simp: rGamma_eq_zero_iff dist_of_nat dist_minus)

with *isCont_rGamma*[*of - of_nat n :: 'a, OF continuous_ident*]

have *filterlim* $(\lambda z. inverse\ (rGamma\ z) :: 'a)$ *at_infinity (at (- of_nat n))*

unfolding *isCont_def* **by** (*intro filterlim_compose[OF filterlim_inverse_at_infinity]*)

(simp_all add: filterlim_at)

moreover have $(\lambda z. inverse\ (rGamma\ z) :: 'a) = Gamma$

by (*intro ext*) (*simp add: rGamma_inverse_Gamma*)

ultimately show *?thesis* **by** (*simp only:*)

qed

lemma *Gamma_residues*:

$(\lambda z. Gamma\ z * (z + of_nat\ n)) - (-\ of_nat\ n) \rightarrow ((-1)^n / fact\ n :: 'a :: Gamma)$

proof (*subst tendsto_cong*)

let $?c = (-1)^n / fact\ n :: 'a$

from *eventually_at_ball'*[*OF zero_less_one, of - of_nat n :: 'a UNIV*]

show *eventually* $(\lambda z. Gamma\ z * (z + of_nat\ n) = inverse\ (rGamma\ z / (z + of_nat\ n)))$

(at (- of_nat n)))

by (*auto elim!: eventually_mono simp: field_split_simps rGamma_inverse_Gamma*)

have $(\lambda z. inverse\ (rGamma\ z / (z + of_nat\ n))) - (-\ of_nat\ n) \rightarrow$

$inverse\ ((-1)^n * fact\ n :: 'a)$

by (*intro tendsto_intros rGamma_zeros*) *simp_all*

3094

also have $inverse ((- 1) ^ n * fact n) = ?c$
by (*simp_all add: field_simps flip: power_mult_distrib*)
finally show $(\lambda z. inverse (rGamma z / (z + of_nat n))) - (- of_nat n) \rightarrow ?c$
.
qed

9.10.11 Alternative definitions

Variant of the Euler form

definition *Gamma_series_euler'* **where**

Gamma_series_euler' z $n =$
 $inverse z * (\prod k=1..n. exp (z * of_real (ln (1 + inverse (of_nat k)))) / (1 +$
 $z / of_nat k))$

context

begin

private lemma *Gamma_euler'_aux1*:

fixes $z :: 'a :: \{real_normed_field, banach\}$

assumes $n: n > 0$

shows $exp (z * of_real (ln (of_nat n + 1))) = (\prod k=1..n. exp (z * of_real (ln$
 $(1 + 1 / of_nat k))))$

proof -

have $(\prod k=1..n. exp (z * of_real (ln (1 + 1 / of_nat k)))) =$
 $exp (z * of_real (\sum k = 1..n. ln (1 + 1 / real_of_nat k)))$

by (*subst exp_sum [symmetric]*) (*simp_all add: sum_distrib_left*)

also have $(\sum k=1..n. ln (1 + 1 / of_nat k) :: real) = ln (\prod k=1..n. 1 + 1 /$
 $real_of_nat k)$

by (*subst ln_prod [symmetric]*) (*auto intro!: add_pos_nonneg*)

also have $(\prod k=1..n. 1 + 1 / of_nat k :: real) = (\prod k=1..n. (of_nat k + 1)$
 $/ of_nat k)$

by (*intro prod.cong*) (*simp_all add: field_split_simps*)

also have $(\prod k=1..n. (of_nat k + 1) / of_nat k :: real) = of_nat n + 1$

by (*induction n*) (*simp_all add: prod_nat_ivl_Suc' field_split_simps*)

finally show *?thesis ..*

qed

theorem *Gamma_series_euler'*:

assumes $z: (z :: 'a :: Gamma) \notin \mathbb{Z}_{\leq 0}$

shows $(\lambda n. Gamma_series_euler' z n) \longrightarrow Gamma z$

proof (*rule Gamma_seriesI, rule Lim_transform_eventually*)

let $?f = \lambda n. fact n * exp (z * of_real (ln (of_nat n + 1))) / pochhammer z (n$
 $+ 1)$

let $?r = \lambda n. ?f n / Gamma_series z n$

let $?r' = \lambda n. exp (z * of_real (ln (of_nat (Suc n) / of_nat n)))$

from z **have** $z' : z \neq 0$ **by** *auto*

have *eventually* $(\lambda n. ?r' n = ?r n)$ *sequentially*

using z **by** (*auto simp: field_split_simps Gamma_series_def ring_distrib*
exp_diff ln_div)

```

      intro: eventually_mono eventually_gt_at_top[of 0::nat] dest:
pochhammer_eq_0_imp_nonpos_Int)
moreover have ?r'  $\longrightarrow$  exp (z * of_real (ln 1))
  by (intro tendsto_intros LIMSEQ_Suc_n_over_n) simp_all
ultimately show ?r  $\longrightarrow$  1 by (force intro: Lim_transform_eventually)

from eventually_gt_at_top[of 0::nat]
  show eventually ( $\lambda n. ?r n = \text{Gamma\_series\_euler}' z n / \text{Gamma\_series } z n$ )
sequentially
proof eventually_elim
  fix n :: nat assume n: n > 0
  from n z' have Gamma_series_euler' z n =
    exp (z * of_real (ln (of_nat n + 1))) / (z * ( $\prod_{k=1..n} (1 + z / \text{of\_nat } k)$ ))
  by (subst Gamma_euler'_aux1)
    (simp_all add: Gamma_series_euler'_def prod.distrib
      prod_inverse[symmetric] divide_inverse)
  also have ( $\prod_{k=1..n} (1 + z / \text{of\_nat } k)$ ) = pochhammer (z + 1) n / fact n
  proof (cases n)
  case (Suc n')
  then show ?thesis
    unfolding pochhammer_prod fact_prod
  by (simp add: atLeastLessThanSuc_atLeastAtMost field_simps prod_dividef

      prod.atLeast_Suc_atMost_Suc_shift del: prod.cl_ivl_Suc)
  qed auto
  also have z * ... = pochhammer z (Suc n) / fact n by (simp add: pochhammer_rec)
  finally show ?r n = Gamma_series_euler' z n / Gamma_series z n by simp
qed
qed
end

```

Weierstrass form

definition Gamma_series_Weierstrass :: 'a :: {banach,real_normed_field} \Rightarrow nat \Rightarrow 'a **where**

Gamma_series_Weierstrass z n =
 exp (-euler_mascheroni * z) / z * ($\prod_{k=1..n} \text{exp } (z / \text{of_nat } k) / (1 + z / \text{of_nat } k)$)

definition

rGamma_series_Weierstrass :: 'a :: {banach,real_normed_field} \Rightarrow nat \Rightarrow 'a **where**

rGamma_series_Weierstrass z n =
 exp (euler_mascheroni * z) * z * ($\prod_{k=1..n} (1 + z / \text{of_nat } k) * \text{exp } (-z / \text{of_nat } k)$)

lemma Gamma_series_Weierstrass_nonpos_Ints:

eventually ($\lambda k. \text{Gamma_series_Weierstrass } (- \text{ of_nat } n) k = 0$) *sequentially*
using *eventually_ge_at_top*[of n] **by** *eventually_elim* (*auto simp: Gamma_series_Weierstrass_def*)

lemma *rGamma_series_Weierstrass_nonpos_Int*:

eventually ($\lambda k. \text{rGamma_series_Weierstrass } (- \text{ of_nat } n) k = 0$) *sequentially*
using *eventually_ge_at_top*[of n] **by** *eventually_elim* (*auto simp: rGamma_series_Weierstrass_def*)

theorem *Gamma_Weierstrass_complex*: *Gamma_series_Weierstrass* $z \longrightarrow$
Gamma ($z :: \text{complex}$)

proof (*cases* $z \in \mathbb{Z}_{\leq 0}$)

case *True*

then obtain n **where** $z = - \text{ of_nat } n$ **by** (*elim nonpos_Ints_cases'*)

also from *True* **have** *Gamma_series_Weierstrass* ... \longrightarrow *Gamma* z

by (*simp add: tendsto_cong[OF Gamma_series_Weierstrass_nonpos_Int]*
Gamma_nonpos_Int)

finally show *?thesis* .

next

case *False*

hence $z: z \neq 0$ **by** *auto*

let $?f = (\lambda x. \prod x = \text{Suc } 0..x. \text{exp } (z / \text{of_nat } x) / (1 + z / \text{of_nat } x))$

have $A: \text{exp } (\ln (1 + z / \text{of_nat } n)) = (1 + z / \text{of_nat } n)$ **if** $n \geq 1$ **for** $n :: \text{nat}$

using *False that* **by** (*subst exp_Ln*) (*auto simp: field_simps dest!: plus_of_nat_eq_0_imp*)

have ($\lambda n. \sum k=1..n. z / \text{of_nat } k - \ln (1 + z / \text{of_nat } k)$) \longrightarrow *ln_Gamma*
 $z + \text{euler_mascheroni} * z + \ln z$

using *ln_Gamma_series'_aux*[OF *False*]

by (*simp only: atLeastLessThanSuc_atLeastAtMost [symmetric] One_nat_def*
sum.shift_bounds_Suc_ivl sums_def atLeast0LessThan)

from *tendsto_exp*[OF *this*] *False* z **have** $?f \longrightarrow z * \text{exp } (\text{euler_mascheroni}$
 $* z) * \text{Gamma } z$

by (*simp add: exp_add exp_sum exp_diff mult_ac Gamma_complex_altdef A*)

from *tendsto_mult*[OF *tendsto_const*[of $\text{exp } (-\text{euler_mascheroni} * z) / z$] *this*]
 z

show *Gamma_series_Weierstrass* $z \longrightarrow$ *Gamma* z

by (*simp add: exp_minus field_split_simps Gamma_series_Weierstrass_def*
[abs_def])

qed

lemma *tendsto_complex_of_real_iff*: ($(\lambda x. \text{complex_of_real } (f x)) \longrightarrow \text{of_real}$
 c) $F = (f \longrightarrow c)$ F

by (*rule tendsto_of_real_iff*)

lemma *Gamma_Weierstrass_real*: *Gamma_series_Weierstrass* $x \longrightarrow$ *Gamma*
 $(x :: \text{real})$

using *Gamma_Weierstrass_complex*[of $\text{of_real } x$] **unfolding** *Gamma_series_Weierstrass_def*[*abs_def*]

by (*subst tendsto_complex_of_real_iff [symmetric]*)

(*simp_all add: exp_of_real [symmetric] Gamma_complex_of_real*)

lemma *rGamma_Weierstrass_complex*: *rGamma_series_Weierstrass* $z \longrightarrow$
rGamma ($z :: \text{complex}$)

```

proof (cases z ∈ ℤ≤0)
  case True
    then obtain n where z = - of_nat n by (elim nonpos_Ints_cases')
    also from True have rGamma_series_Weierstrass ... ⟶ rGamma z
      by (simp add: tendsto_cong[OF rGamma_series_Weierstrass_nonpos_Ints]
rGamma_nonpos_Int)
    finally show ?thesis .
  next
    case False
    have rGamma_series_Weierstrass z = (λn. inverse (Gamma_series_Weierstrass
z n))
    by (simp add: rGamma_series_Weierstrass_def[abs_def] Gamma_series_Weierstrass_def
exp_minus divide_inverse prod_inverse[symmetric] mult_ac)
    also from False have ... ⟶ inverse (Gamma z)
    by (intro tendsto_intros Gamma_Weierstrass_complex) (simp add: Gamma_eq_zero_iff)
    finally show ?thesis by (simp add: Gamma_def)
qed

```

Binomial coefficient form

lemma Gamma_gbinomial:

(λn. ((z + of_nat n) gchoose n) * exp (-z * of_real (ln (of_nat n)))) ⟶
rGamma (z+1)

proof (cases z = 0)

case False

show ?thesis

proof (rule Lim_transform_eventually)

let ?powr = λa b. exp (b * of_real (ln (of_nat a)))

show eventually (λn. rGamma_series z n / z =
((z + of_nat n) gchoose n) * ?powr n (-z)) sequentially

proof (intro always_eventually allI)

fix n :: nat

from False **have** ((z + of_nat n) gchoose n) = pochhammer z (Suc n) / z /
fact n

by (simp add: gbinomial_pochhammer' pochhammer_rec)

also have pochhammer z (Suc n) / z / fact n * ?powr n (-z) = rGamma_series
z n / z

by (simp add: rGamma_series_def field_split_simps exp_minus)

finally show rGamma_series z n / z = ((z + of_nat n) gchoose n) * ?powr
n (-z) ..

qed

from False **have** (λn. rGamma_series z n / z) ⟶ rGamma z / z **by** (intro
tendsto_intros)

also from False **have** rGamma z / z = rGamma (z + 1) **using** rGamma_plus1[of
z]

by (simp add: field_simps)

finally show (λn. rGamma_series z n / z) ⟶ rGamma (z+1) .

qed

qed (*simp_all* add: *binomial_gbinomial* [*symmetric*])

lemma *gbinomial_minus'*: $(a + \text{of_nat } b) \text{ gchoose } b = (-1)^b * (- (a + 1) \text{ gchoose } b)$

by (*subst gbinomial_minus*) (*simp* add: *power_mult_distrib* [*symmetric*])

lemma *gbinomial_asymptotic*:

fixes $z :: 'a :: \text{Gamma}$

shows $(\lambda n. (z \text{ gchoose } n) / ((-1)^n / \exp((z+1) * \text{of_real } (\ln (\text{real } n))))$
 \longrightarrow

$\text{inverse } (\text{Gamma } (-z))$

unfolding *rGamma_inverse_Gamma* [*symmetric*] **using** *Gamma_gbinomial*[*of -z-1*]

by (*subst (asm) gbinomial_minus'*)

(*simp* add: *add_ac mult_ac divide_inverse power_inverse* [*symmetric*])

lemma *fact_binomial_limit*:

$(\lambda n. \text{of_nat } ((k + n) \text{ choose } n) / \text{of_nat } (n^k) :: 'a :: \text{Gamma}) \longrightarrow 1 / \text{fact } k$

proof (*rule Lim_transform_eventually*)

have $(\lambda n. \text{of_nat } ((k + n) \text{ choose } n) / \text{of_real } (\exp(\text{of_nat } k * \ln(\text{real_of_nat } n))))$
 $\longrightarrow 1 / \text{Gamma } (\text{of_nat } (\text{Suc } k) :: 'a)$ (**is** $?f \longrightarrow _$)

using *Gamma_gbinomial*[*of of_nat k :: 'a*]

by (*simp* add: *binomial_gbinomial Gamma_def field_split_simps exp_of_real* [*symmetric*] *exp_minus*)

also have $\text{Gamma } (\text{of_nat } (\text{Suc } k)) = \text{fact } k$ **by** (*simp* add: *Gamma_fact*)

finally show $?f \longrightarrow 1 / \text{fact } k$.

show eventually $(\lambda n. ?f n = \text{of_nat } ((k + n) \text{ choose } n) / \text{of_nat } (n^k))$
sequentially

using *eventually_gt_at_top*[*of 0::nat*]

proof *eventually_elim*

fix $n :: \text{nat}$ **assume** $n > 0$

from n **have** $\exp(\text{real_of_nat } k * \ln(\text{real_of_nat } n)) = \text{real_of_nat } (n^k)$

by (*simp* add: *exp_of_nat_mult*)

thus $?f n = \text{of_nat } ((k + n) \text{ choose } n) / \text{of_nat } (n^k)$ **by** *simp*

qed

qed

lemma *binomial_asymptotic'*:

$(\lambda n. \text{of_nat } ((k + n) \text{ choose } n) / (\text{of_nat } (n^k) / \text{fact } k) :: 'a :: \text{Gamma})$
 $\longrightarrow 1$

using *tendsto_mult*[*OF fact_binomial_limit*[*of k*] *tendsto_const*[*of fact k :: 'a*]]
by *simp*

lemma *gbinomial_Beta*:

assumes $z + 1 \notin \mathbb{Z}_{\leq 0}$

shows $((z :: 'a :: \text{Gamma}) \text{ gchoose } n) = \text{inverse } ((z + 1) * \text{Beta } (z - \text{of_nat } n$

```

+ 1) (of_nat n + 1))
using assms
proof (induction n arbitrary: z)
  case 0
  hence  $z + 2 \notin \mathbb{Z}_{\leq 0}$ 
  using plus_one_in_nonpos_Ints_imp[of z+1] by (auto simp: add commute)
  with 0 show ?case
  by (auto simp: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric]
add commute)
next
  case (Suc n z)
  show ?case
  proof (cases  $z \in \mathbb{Z}_{\leq 0}$ )
    case True
    with Suc.prem1 have  $z = 0$ 
    by (auto elim!: nonpos_Ints_cases simp: algebra_simps one_plus_of_int_in_nonpos_Ints_iff)
    show ?thesis
    proof (cases  $n = 0$ )
      case True
      with ‹ $z = 0$ › show ?thesis
      by (simp add: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric])
    next
      case False
      with ‹ $z = 0$ › show ?thesis
      by (simp_all add: Beta_pole1 one_minus_of_nat_in_nonpos_Ints_iff)
    qed
  next
  case False
  have (z gchoose (Suc n)) = ((z - 1 + 1) gchoose (Suc n)) by simp
  also have ... = (z - 1 gchoose n) * ((z - 1) + 1) / of_nat (Suc n)
  by (subst gbinomial_factors) (simp add: field_simps)
  also from False have ... = inverse (of_nat (Suc n) * Beta (z - of_nat n)
(of_nat (Suc n)))
  (is _ = inverse ?x) by (subst Suc.IH) (simp_all add: field_simps Beta_pole1)
  also have of_nat (Suc n)  $\notin (\mathbb{Z}_{\leq 0} :: 'a \text{ set})$  by (subst of_nat_in_nonpos_Ints_iff)
simp_all
  hence ?x = (z + 1) * Beta (z - of_nat (Suc n) + 1) (of_nat (Suc n) + 1)
  by (subst Beta_plus1_right [symmetric]) simp_all
  finally show ?thesis .
qed
qed

theorem gbinomial_Gamma:
  assumes  $z + 1 \notin \mathbb{Z}_{\leq 0}$ 
  shows (z gchoose n) = Gamma (z + 1) / (fact n * Gamma (z - of_nat n +
1))
proof -
  have (z gchoose n) = Gamma (z + 2) / (z + 1) / (fact n * Gamma (z - of_nat
n + 1))

```

3100

by (subst gbinomial_Beta[OF assms]) (simp_all add: Beta_def Gamma_fact
[symmetric] add_ac)
also from assms have $\Gamma(z + 2) / (z + 1) = \Gamma(z + 1)$
using Gamma_plus1[of z+1] by (auto simp add: field_split_simps)
finally show ?thesis .
qed

Integral form

lemma integrable_on_pwr_from_0':
assumes $a > (-1::real)$ and $c \geq 0$
shows $(\lambda x. x^{\text{powr } a})$ integrable_on $\{0 <..c\}$
proof -
from c have *: $\{0 <..c\} - \{0..c\} = \{\}$ $\{0..c\} - \{0 <..c\} = \{0\}$ by auto
show ?thesis
by (rule integrable_spike_set [OF integrable_on_pwr_from_0 [OF a c]]) (simp_all
add: *)
qed

lemma absolutely_integrable_Gamma_integral:
assumes $\text{Re } z > 0$ $a > 0$
shows $(\lambda t. \text{complex_of_real } t^{\text{powr } (z - 1)} / \text{of_real } (\exp(a * t)))$
absolutely_integrable_on $\{0 <.. \}$ (is ?f absolutely_integrable_on _)
proof -
have $((\lambda x. (\text{Re } z - 1) * (\ln x / x)) \longrightarrow (\text{Re } z - 1) * 0)$ at_top
by (intro tendsto_intros ln_x_over_x_tendsto_0)
hence $((\lambda x. ((\text{Re } z - 1) * \ln x) / x) \longrightarrow 0)$ at_top by simp
from order_tendstoD(2)[OF this, of a/2] and $\langle a > 0 \rangle$
have eventually $(\lambda x. (\text{Re } z - 1) * \ln x / x < a/2)$ at_top by simp
from eventually_conj[OF this eventually_gt_at_top[of 0]]
obtain $x0$ where $\forall x \geq x0. (\text{Re } z - 1) * \ln x / x < a/2 \wedge x > 0$
by (auto simp: eventually_at_top_linorder)
hence $x0 > 0$ by simp
have $x^{\text{powr } (\text{Re } z - 1)} / \exp(a * x) < \exp(-(a/2) * x)$ if $x \geq x0$ for x
proof -
from that and $\langle \forall x \geq x0. _ \rangle$ have $x: (\text{Re } z - 1) * \ln x / x < a / 2$ $x > 0$ by
auto
have $x^{\text{powr } (\text{Re } z - 1)} = \exp((\text{Re } z - 1) * \ln x)$
using $\langle x > 0 \rangle$ by (simp add: powr_def)
also from x have $(\text{Re } z - 1) * \ln x < (a * x) / 2$ by (simp add: field_simps)
finally show ?thesis by (simp add: field_simps exp_add [symmetric])
qed
note $x0 = \langle x0 > 0 \rangle$ this

have ?f absolutely_integrable_on $(\{0 <..x0\} \cup \{x0.. \})$
proof (rule set_integrable_Un)
show ?f absolutely_integrable_on $\{0 <..x0\}$
unfolding set_integrable_def
proof (rule Bochner_Integration.integrable_bound [OF _ _ AE_I2])


```

show integrable lebesgue ( $\lambda x. \text{indicat\_real } \{0 <..x0\} x *_R x \text{ powr } (\text{Re } z - 1)$ )

  using x0(1) assms
  by (intro nonnegative_absolutely_integrable_1 [unfolded set_integrable_def]
integrable_on_powr_from_0') auto
  show ( $\lambda x. \text{indicat\_real } \{0 <..x0\} x *_R (x \text{ powr } (z - 1) / \text{exp } (a * x))$ )  $\in$ 
borel_measurable lebesgue
  by (intro measurable_completion)
  (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
fix x :: real
have x powr (Re z - 1) / exp (a * x)  $\leq$  x powr (Re z - 1) / 1 if x  $\geq$  0
  using that assms by (intro divide_left_mono) auto
thus norm (indicator {0 <..x0} x *_R ?f x)  $\leq$ 
norm (indicator {0 <..x0} x *_R x powr (Re z - 1))
  by (simp_all add: norm_divide norm_powr_real_powr indicator_def)
qed
next
show ?f absolutely_integrable_on {x0..}
  unfolding set_integrable_def
proof (rule Bochner_Integration.integrable_bound [OF __ AE_I2])
  show integrable lebesgue ( $\lambda x. \text{indicat\_real } \{x0..\} x *_R \text{exp } (- (a / 2) * x)$ )
using assms
  by (intro nonnegative_absolutely_integrable_1 [unfolded set_integrable_def]
integrable_on_exp_minus_to_infinity) auto
  show ( $\lambda x. \text{indicat\_real } \{x0..\} x *_R (x \text{ powr } (z - 1) / \text{exp } (a * x))$ )  $\in$ 
borel_measurable lebesgue using x0(1)
  by (intro measurable_completion)
  (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
fix x :: real
show norm (indicator {x0..} x *_R ?f x)  $\leq$ 
norm (indicator {x0..} x *_R exp (-(a/2) * x)) using x0
  by (auto simp: norm_divide norm_powr_real_powr indicator_def less_imp_le)
qed
qed auto
also have {0 <..x0}  $\cup$  {x0..} = {0 <..} using x0(1) by auto
finally show ?thesis .
qed

```

lemma integrable_Gamma_integral_bound:

```

fixes a c :: real
assumes a: a > -1 and c: c  $\geq$  0
defines f  $\equiv$   $\lambda x. \text{if } x \in \{0..c\} \text{ then } x \text{ powr } a \text{ else } \text{exp } (-x/2)$ 
shows f integrable_on {0..}
proof -
  have f integrable_on {0..c}
  by (rule integrable_spike_finite[of {}], OF __ integrable_on_powr_from_0[of
a c])
  (insert a c, simp_all add: f_def)
moreover have A: ( $\lambda x. \text{exp } (-x/2)$ ) integrable_on {c..}

```

```

    using integrable_on_exp_minus_to_infinity[of 1/2] by simp
  have f_integrable_on {c..}
    by (rule integrable_spike_finite[of {c}, OF _ _ A]) (simp_all add: f_def)
  ultimately show f_integrable_on {0..}
    by (rule integrable_Un) (insert c, auto simp: max_def)
qed

theorem Gamma_integral_complex:
  assumes z: Re z > 0
  shows (( $\lambda t.$  of_real t powr (z - 1) / of_real (exp t)) has_integral Gamma z)
  {0..}
proof -
  have A: (( $\lambda t.$  (of_real t) powr (z - 1) * of_real ((1 - t) ^ n))
    has_integral (fact n / pochhammer z (n+1))) {0..1}
  if Re z > 0 for n z using that
proof (induction n arbitrary: z)
  case 0
  have (( $\lambda t.$  complex_of_real t powr (z - 1)) has_integral
    (of_real 1 powr z / z - of_real 0 powr z / z)) {0..1} using 0
  by (intro fundamental_theorem_of_calculus_interior)
  (auto intro!: continuous_intros derivative_eq_intros has_vector_derivative_real_field)
  thus ?case by simp
next
  case (Suc n)
  let ?f =  $\lambda t.$  complex_of_real t powr z / z
  let ?f' =  $\lambda t.$  complex_of_real t powr (z - 1)
  let ?g =  $\lambda t.$  (1 - complex_of_real t) ^ Suc n
  let ?g' =  $\lambda t.$  - ((1 - complex_of_real t) ^ n) * of_nat (Suc n)
  have (( $\lambda t.$  ?f' t * ?g t) has_integral
    (of_nat (Suc n) * fact n / pochhammer z (n+2))) {0..1}
  (is _ has_integral ?I) _
  proof (rule integration_by_parts_interior[where f' = ?f' and g = ?g])
    from Suc.prem show continuous_on {0..1} ?f continuous_on {0..1} ?g
    by (auto intro!: continuous_intros)
  next
  fix t :: real assume t: t  $\in$  {0 <.. $<$ 1}
  show (?f has_vector_derivative ?f' t) (at t) using t Suc.prem
    by (auto intro!: derivative_eq_intros has_vector_derivative_real_field)
  show (?g has_vector_derivative ?g' t) (at t)
    by (rule has_vector_derivative_real_field derivative_eq_intros refl)+
  simp_all
  next
  from Suc.prem have [simp]: z  $\neq$  0 by auto
  from Suc.prem have A: Re (z + of_nat n) > 0 for n by simp
  have [simp]: z + of_nat n  $\neq$  0 z + 1 + of_nat n  $\neq$  0 for n
  using A[of n] A[of Suc n] by (auto simp add: add_assoc simp del: plus_complex.sel)
  have (( $\lambda x.$  of_real x powr z * of_real ((1 - x) ^ n) * (- of_nat (Suc n) /
z)) has_integral
    fact n / pochhammer (z+1) (n+1) * (- of_nat (Suc n) / z)) {0..1}

```

```

    (is (?A has_integral ?B) _)
  using Suc.IH[of z+1] Suc.prem by (intro has_integral_mult_left) (simp_all
add: add_ac pochhammer_rec)
  also have ?A = ( $\lambda t. ?f t * ?g' t$ ) by (intro ext) (simp_all add: field_simps)
  also have ?B = - (of_nat (Suc n) * fact n / pochhammer z (n+2))
    by (simp add: field_split_simps pochhammer_rec
prod.shift_bounds_cl_Suc_ivl del: of_nat_Suc)
  finally show (( $\lambda t. ?f t * ?g' t$ ) has_integral (?f 1 * ?g 1 - ?f 0 * ?g 0 -
?I)) {0..1}
    by simp
  qed (simp_all add: bounded_bilinear_mult)
  thus ?case by simp
qed

have B: (( $\lambda t. \text{if } t \in \{0..of\_nat\ n\} \text{ then } of\_real\ t\ \text{powr}\ (z - 1) * (1 - of\_real\ t / of\_nat\ n) \wedge^n \text{ else } 0$ )
has_integral (of_nat n powr z * fact n / pochhammer z (n+1))) {0..}
for n
proof (cases n > 0)
case [simp]: True
hence [simp]: n  $\neq$  0 by auto
with has_integral_affinity01[OF A[OF z, of n], of inverse (of_nat n) 0]
have (( $\lambda x. (of\_nat\ n - of\_real\ x) \wedge^n * (of\_real\ x / of\_nat\ n) \text{ powr}\ (z - 1) / of\_nat\ n \wedge^n$ )
has_integral fact n * of_nat n / pochhammer z (n+1)) (( $\lambda x. \text{real } n * x$ ) {0..1})
(is (?f has_integral ?I) ?ivl) by (simp add: field_simps scaleR_conv_of_real)
also from True have (( $\lambda x. \text{real } n * x$ ) {0..1}) = {0..real n}
by (subst image_mult_atLeastAtMost) simp_all
also have ?f = ( $\lambda x. (of\_real\ x / of\_nat\ n) \text{ powr}\ (z - 1) * (1 - of\_real\ x / of\_nat\ n) \wedge^n$ )
using True by (intro ext) (simp add: field_simps)
finally have (( $\lambda x. (of\_real\ x / of\_nat\ n) \text{ powr}\ (z - 1) * (1 - of\_real\ x / of\_nat\ n) \wedge^n$ )
has_integral ?I) {0..real n} (is ?P) .
also have ?P  $\longleftrightarrow$  (( $\lambda x. \text{exp } ((z - 1) * of\_real (ln (x / of\_nat n))) * (1 - of\_real\ x / of\_nat\ n) \wedge^n$ )
has_integral ?I) {0..real n}
by (intro has_integral_spike_finite_eq[of {0}]) (auto simp: powr_def Ln_of_real [symmetric])
also have ...  $\longleftrightarrow$  (( $\lambda x. \text{exp } ((z - 1) * of\_real (ln x - ln (of\_nat n))) * (1 - of\_real\ x / of\_nat\ n) \wedge^n$ )
has_integral ?I) {0..real n}
by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: ln_div)
finally have ... .
note B = has_integral_mult_right[OF this, of exp ((z - 1) * ln (of_nat n))]
have (( $\lambda x. \text{exp } ((z - 1) * of\_real (ln x)) * (1 - of\_real\ x / of\_nat\ n) \wedge^n$ )
has_integral (?I * exp ((z - 1) * ln (of_nat n)))) {0..real n} (is ?P)
by (insert B, subst (asm) mult.assoc [symmetric], subst (asm) exp_add

```

```

[symmetric]
  (simp add: algebra_simps)
  also have ?P  $\longleftrightarrow$   $((\lambda x. \text{of\_real } x \text{ powr } (z - 1) * (1 - \text{of\_real } x / \text{of\_nat } n) ^ n)$ 
    has_integral (?I * exp ((z - 1) * ln (of_nat n)))) {0..real n}
  by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: powr_def
Ln_of_real)
  also have fact n * of_nat n / pochhammer z (n+1) * exp ((z - 1) * Ln
(of_nat n)) =
    (of_nat n powr z * fact n / pochhammer z (n+1))
  by (auto simp add: powr_def algebra_simps exp_diff exp_of_real)
  finally show ?thesis by (subst has_integral_restrict) simp_all
next
case False
thus ?thesis by (subst has_integral_restrict) (simp_all add: has_integral_refl)
qed

have eventually  $(\lambda n. \text{Gamma\_series } z \text{ n} =$ 
  of_nat n powr z * fact n / pochhammer z (n+1)) sequentially
using eventually_gt_at_top[of 0::nat]
by eventually_elim (simp add: powr_def algebra_simps Gamma_series_def)
from this and Gamma_series_LIMSEQ[of z]
have C:  $(\lambda k. \text{of\_nat } k \text{ powr } z * \text{fact } k / \text{pochhammer } z (k+1)) \longrightarrow \text{Gamma}$ 
 $z$ 
  by (blast intro: Lim_transform_eventually)
{
  fix x :: real assume x:  $x \geq 0$ 
  have lim_exp:  $(\lambda k. (1 - x / \text{real } k) ^ k) \longrightarrow \text{exp } (-x)$ 
  using tendsto_exp_limit_sequentially[of -x] by simp
  have  $(\lambda k. \text{of\_real } x \text{ powr } (z - 1) * \text{of\_real } ((1 - x / \text{of\_nat } k) ^ k))$ 
 $\longrightarrow \text{of\_real } x \text{ powr } (z - 1) * \text{of\_real } (\text{exp } (-x))$  (is ?P)
  by (intro tendsto_intros lim_exp)
  also from eventually_gt_at_top[of nat [x]]
  have eventually  $(\lambda k. \text{of\_nat } k > x)$  sequentially by eventually_elim linarith
  hence ?P  $\longleftrightarrow$   $(\lambda k. \text{if } x \leq \text{of\_nat } k \text{ then}$ 
    of_real x powr (z - 1) * of_real  $((1 - x / \text{of\_nat } k) ^ k)$  else 0)
 $\longrightarrow \text{of\_real } x \text{ powr } (z - 1) * \text{of\_real } (\text{exp } (-x))$ 
  by (intro tendsto_cong) (auto elim!: eventually_mono)
  finally have ... .
}
hence D:  $\forall x \in \{0..\}. (\lambda k. \text{if } x \in \{0..\text{real } k\} \text{ then}$ 
  of_real x powr (z - 1) *  $(1 - \text{of\_real } x / \text{of\_nat } k) ^ k$  else 0)
 $\longrightarrow \text{of\_real } x \text{ powr } (z - 1) / \text{of\_real } (\text{exp } x)$ 
  by (simp add: exp_minus_field_simps cong: if_cong)

have  $((\lambda x. (\text{Re } z - 1) * (\ln x / x)) \longrightarrow (\text{Re } z - 1) * 0)$  at_top
  by (intro tendsto_intros ln_x_over_x_tendsto_0)
hence  $((\lambda x. ((\text{Re } z - 1) * \ln x) / x) \longrightarrow 0)$  at_top by simp
from order_tendstoD(2)[OF this, of 1/2]

```

```

  have eventually ( $\lambda x. (\operatorname{Re} z - 1) * \ln x / x < 1/2$ ) at_top by simp
  from eventually_conj[OF this eventually_gt_at_top[of 0]]
  obtain x0 where  $\forall x \geq x0. (\operatorname{Re} z - 1) * \ln x / x < 1/2 \wedge x > 0$ 
  by (auto simp: eventually_at_top_linorder)
  hence x0:  $x0 > 0 \wedge x. x \geq x0 \implies (\operatorname{Re} z - 1) * \ln x < x / 2$  by auto

  define h where  $h = (\lambda x. \text{if } x \in \{0..x0\} \text{ then } x \text{ powr } (\operatorname{Re} z - 1) \text{ else } \exp(-x/2))$ 
  have le_h:  $x \text{ powr } (\operatorname{Re} z - 1) * \exp(-x) \leq h x$  if  $x: x \geq 0$  for  $x$ 
  proof (cases  $x > x0$ )
    case True
      from True x0(1) have  $x \text{ powr } (\operatorname{Re} z - 1) * \exp(-x) = \exp((\operatorname{Re} z - 1) * \ln x - x)$ 
      by (simp add: powr_def exp_diff exp_minus field_simps exp_add)
      also from x0(2)[of  $x$ ] True have  $\dots < \exp(-x/2)$ 
      by (simp add: field_simps)
      finally show ?thesis using True by (auto simp add: h_def)
    next
      case False
      from  $x$  have  $x \text{ powr } (\operatorname{Re} z - 1) * \exp(-x) \leq x \text{ powr } (\operatorname{Re} z - 1) * 1$ 
      by (intro mult_left_mono) simp_all
      with False show ?thesis by (auto simp add: h_def)
  qed

  have E:  $\forall x \in \{0..\}. \operatorname{cmod}(\text{if } x \in \{0..\operatorname{real} k\} \text{ then } \operatorname{of\_real} x \text{ powr } (z - 1) * (1 - \operatorname{complex\_of\_real} x / \operatorname{of\_nat} k) ^ k \text{ else } 0) \leq h x$ 
  (is  $\forall x \in \dots. ?f x \leq \_$ ) for  $k$ 
  proof safe
    fix  $x :: \operatorname{real}$  assume  $x: x \geq 0$ 
    {
      fix  $x :: \operatorname{real}$  and  $n :: \operatorname{nat}$  assume  $x: x \leq \operatorname{of\_nat} n$ 
      have  $(1 - \operatorname{complex\_of\_real} x / \operatorname{of\_nat} n) = \operatorname{complex\_of\_real}((1 - x / \operatorname{of\_nat} n))$  by simp
      also have  $\operatorname{norm} \dots = |(1 - x / \operatorname{real} n)|$  by (subst norm_of_real) (rule refl)
      also from  $x$  have  $\dots = (1 - x / \operatorname{real} n)$  by (intro abs_of_nonneg) (simp_all add: field_split_simps)
      finally have  $\operatorname{cmod}(1 - \operatorname{complex\_of\_real} x / \operatorname{of\_nat} n) = 1 - x / \operatorname{real} n$ .
    } note D = this
    from D[of  $x k$ ]  $x$ 
    have  $?f x \leq (\text{if } \operatorname{of\_nat} k \geq x \wedge k > 0 \text{ then } x \text{ powr } (\operatorname{Re} z - 1) * (1 - x / \operatorname{real} k) ^ k \text{ else } 0)$ 
    by (auto simp: norm_mult norm_powr_real_powr norm_power intro!: mult_nonneg_nonneg)
    also have  $\dots \leq x \text{ powr } (\operatorname{Re} z - 1) * \exp(-x)$ 
    by (auto intro!: mult_left_mono exp_ge_one_minus_x_over_n_power_n)
    also from  $x$  have  $\dots \leq h x$  by (rule le_h)
    finally show  $?f x \leq h x$ .
  qed

  have F:  $h$  integrable_on  $\{0..\}$  unfolding h_def
  by (rule integrable_Gamma_integral_bound) (insert assms x0(1), simp_all)

```

3106

show *?thesis*
by (rule *has_integral_dominated_convergence*[OF B F E D C])
qed

lemma *Gamma_integral_real*:

assumes $x: x > (0 :: \text{real})$

shows $((\lambda t. t \text{ powr } (x - 1) / \text{exp } t) \text{ has_integral } \text{Gamma } x) \{0..\}$

proof -

have $A: ((\lambda t. \text{complex_of_real } t \text{ powr } (\text{complex_of_real } x - 1) /$
 $\text{complex_of_real } (\text{exp } t)) \text{ has_integral } \text{complex_of_real } (\text{Gamma } x)) \{0..\}$

using *Gamma_integral_complex*[of x] **assms** **by** (*simp_all* add: *Gamma_complex_of_real*
powr_of_real)

have $((\lambda t. \text{complex_of_real } (t \text{ powr } (x - 1) / \text{exp } t)) \text{ has_integral } \text{of_real}$
 $(\text{Gamma } x)) \{0..\}$

by (rule *has_integral_eq*[OF A]) (*simp_all* add: *powr_of_real* [*symmetric*])

from *has_integral_linear*[OF *this* *bounded_linear_Re*] **show** *?thesis* **by** (*simp*
add: o_def)

qed

lemma *absolutely_integrable_Gamma_integral'*:

assumes $\text{Re } z > 0$

shows $(\lambda t. \text{complex_of_real } t \text{ powr } (z - 1) / \text{of_real } (\text{exp } t)) \text{ absolutely_integrable_on}$
 $\{0<..\}$

using *absolutely_integrable_Gamma_integral* [OF *assms* *zero_less_one*] **by** *simp*

lemma *Gamma_integral_complex'*:

assumes $z: \text{Re } z > 0$

shows $((\lambda t. \text{of_real } t \text{ powr } (z - 1) / \text{of_real } (\text{exp } t)) \text{ has_integral } \text{Gamma } z)$
 $\{0<..\}$

proof -

have $((\lambda t. \text{of_real } t \text{ powr } (z - 1) / \text{of_real } (\text{exp } t)) \text{ has_integral } \text{Gamma } z)$
 $\{0..\}$

by (rule *Gamma_integral_complex*) *fact+*

hence $((\lambda t. \text{if } t \in \{0<..\} \text{ then } \text{of_real } t \text{ powr } (z - 1) / \text{of_real } (\text{exp } t) \text{ else } 0)$
 $\text{has_integral } \text{Gamma } z) \{0..\}$

by (rule *has_integral_spike* [of $\{0\}$, *rotated 2*]) *auto*

also have *?this = ?thesis*

by (*subst* *has_integral_restrict*) *auto*

finally show *?thesis* .

qed

lemma *Gamma_conv_nn_integral_real*:

assumes $s > (0 :: \text{real})$

shows $\text{Gamma } s = \text{nn_integral } \text{l borel } (\lambda t. \text{ennreal } (\text{indicator } \{0..\} t * t \text{ powr}$
 $(s - 1) / \text{exp } t))$

using *nn_integral_has_integral_lebesgue*[OF $\text{Gamma_integral_real}$ [OF *assms*]]
by *simp*

lemma *integrable_Beta*:

```

assumes  $a > 0 \ b > (0::real)$ 
shows  $set\_integrable \ lborel \ \{0..1\} \ (\lambda t. t \ powr \ (a - 1) * (1 - t) \ powr \ (b - 1))$ 
proof -
  define  $C$  where  $C = \max 1 \ ((1/2) \ powr \ (b - 1))$ 
  define  $D$  where  $D = \max 1 \ ((1/2) \ powr \ (a - 1))$ 
  have  $C: (1 - x) \ powr \ (b - 1) \leq C$  if  $x \in \{0..1/2\}$  for  $x$ 
  proof (cases  $b < 1$ )
    case  $False$ 
      with that have  $(1 - x) \ powr \ (b - 1) \leq (1 \ powr \ (b - 1))$  by (intro powr_mono2) auto
      thus ?thesis by (auto simp:  $C\_def$ )
    qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2' powr_mono2 C_def)
    have  $D: x \ powr \ (a - 1) \leq D$  if  $x \in \{1/2..1\}$  for  $x$ 
    proof (cases  $a < 1$ )
      case  $False$ 
        with that have  $x \ powr \ (a - 1) \leq (1 \ powr \ (a - 1))$  by (intro powr_mono2) auto
        thus ?thesis by (auto simp:  $D\_def$ )
      next
        case  $True$ 
        qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2' powr_mono2 D_def)
        have [simp]:  $C \geq 0 \ D \geq 0$  by (simp_all add:  $C\_def \ D\_def$ )

  have  $I1: set\_integrable \ lborel \ \{0..1/2\} \ (\lambda t. t \ powr \ (a - 1) * (1 - t) \ powr \ (b - 1))$ 
  unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound[OF __ AE_I2])
    have  $(\lambda t. t \ powr \ (a - 1)) \ integrable\_on \ \{0..1/2\}$ 
      by (rule integrable_on_powr_from_0) (use assms in auto)
    hence  $(\lambda t. t \ powr \ (a - 1)) \ absolutely\_integrable\_on \ \{0..1/2\}$ 
      by (subst absolutely_integrable_on_iff_nonneg) auto
    from integrable_mult_right[OF this [unfolded set_integrable_def], of C]
    show  $integrable \ lborel \ (\lambda x. \ indicat\_real \ \{0..1/2\} \ x *_{\mathbb{R}} \ (C * x \ powr \ (a - 1)))$ 
      by (subst (asm) integrable_completion) (auto simp: mult_ac)
  next
    fix  $x :: real$ 
    have  $x \ powr \ (a - 1) * (1 - x) \ powr \ (b - 1) \leq x \ powr \ (a - 1) * C$  if  $x \in \{0..1/2\}$ 
      using that by (intro mult_left_mono powr_mono2 C) auto
    thus  $norm \ (indicator \ \{0..1/2\} \ x *_{\mathbb{R}} \ (x \ powr \ (a - 1) * (1 - x) \ powr \ (b - 1))) \leq$ 
       $norm \ (indicator \ \{0..1/2\} \ x *_{\mathbb{R}} \ (C * x \ powr \ (a - 1)))$ 
      by (auto simp: indicator_def abs_mult mult_ac)
    qed (auto intro!: AE_I2 simp: indicator_def)

  have  $I2: set\_integrable \ lborel \ \{1/2..1\} \ (\lambda t. t \ powr \ (a - 1) * (1 - t) \ powr \ (b - 1))$ 

```

```

- 1))
  unfolding set_integrable_def
proof (rule Bochner_Integration.integrable_bound[OF _ _ AE_I2])
  have ( $\lambda t. t \text{ powr } (b - 1)$ ) integrable_on {0..1/2}
  by (rule integrable_on_powr_from_0) (use assms in auto)
  hence ( $\lambda t. t \text{ powr } (b - 1)$ ) integrable_on (cbox 0 (1/2)) by simp
  from integrable_affinity[OF this, of -1 1]
  have ( $\lambda t. (1 - t) \text{ powr } (b - 1)$ ) integrable_on {1/2..1} by simp
  hence ( $\lambda t. (1 - t) \text{ powr } (b - 1)$ ) absolutely_integrable_on {1/2..1}
  by (subst absolutely_integrable_on_iff_nonneg) auto
  from integrable_mult_right[OF this [unfolded set_integrable_def], of D]
  show integrable lborel ( $\lambda x. \text{indicator\_real } \{1/2..1\} x *_R (D * (1 - x) \text{ powr } (b - 1))$ )
- 1)))
  by (subst (asm) integrable_completion) (auto simp: mult_ac)
next
fix x :: real
have  $x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1) \leq D * (1 - x) \text{ powr } (b - 1)$  if
 $x \in \{1/2..1\}$ 
  using that by (intro mult_right_mono powr_mono2 D) auto
  thus norm (indicator {1/2..1} x *_R (x powr (a - 1) * (1 - x) powr (b - 1))) ≤
    norm (indicator {1/2..1} x *_R (D * (1 - x) powr (b - 1)))
  by (auto simp: indicator_def abs_mult mult_ac)
qed (auto intro!: AE_I2 simp: indicator_def)

have set_integrable lborel ({0..1/2} ∪ {1/2..1}) ( $\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1)$ )
  by (intro set_integrable_Un I1 I2) auto
also have {0..1/2} ∪ {1/2..1} = {0..(1::real)} by auto
finally show ?thesis .
qed

```

lemma integrable_Beta':

```

assumes  $a > 0$   $b > (0::real)$ 
shows ( $\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1)$ ) integrable_on {0..1}
using integrable_Beta[OF assms] by (rule set_borel_integral_eq_integral)

```

theorem has_integral_Beta_real:

```

assumes  $a > 0$  and  $b > (0 :: real)$ 
shows (( $\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1)$ ) has_integral Beta a b)
{0..1}
proof -

```

```

  define B where  $B = \text{integral } \{0..1\} (\lambda x. x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1))$ 

```

```

  have [simp]:  $B \geq 0$  unfolding B_def using a b

```

```

  by (intro integral_nonneg integrable_Beta') auto

```

```

  from a b have ennreal (Gamma a * Gamma b) =

```

```

    ( $\int^+ t. \text{ennreal } (\text{indicator } \{0..1\} t * t \text{ powr } (a - 1) / \exp t) \partial \text{lborel}$ ) *
    ( $\int^+ t. \text{ennreal } (\text{indicator } \{0..1\} t * t \text{ powr } (b - 1) / \exp t) \partial \text{lborel}$ )

```



```

  by (subst ennreal_mult') (simp_all add: Gamma_conv_nn_integral_real)
  also have ... = ( $\int^{+t} \int^{+u} \text{ennreal} (\text{indicator } \{0..\} t * t \text{ powr } (a - 1) / \exp t) * \text{ennreal} (\text{indicator } \{0..\} u * u \text{ powr } (b - 1) / \exp u) \partial \text{lborel} \partial \text{lborel}$ )
  by (simp add: nn_integral_cmult nn_integral_multc)
  also have ... = ( $\int^{+t} \int^{+u} \text{ennreal} (\text{indicator } (\{0..\} \times \{0..\}) (t,u) * t \text{ powr } (a - 1) * u \text{ powr } (b - 1) / \exp (t + u)) \partial \text{lborel} \partial \text{lborel}$ )
  by (intro nn_integral_cong)
  (auto simp: indicator_def divide_ennreal ennreal_mult' [symmetric] exp_add)
  also have ... = ( $\int^{+t} \int^{+u} \text{ennreal} (\text{indicator } (\{0..\} \times \{t..\}) (t,u) * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1) / \exp u) \partial \text{lborel} \partial \text{lborel}$ )
  proof (rule nn_integral_cong, goal_cases)
  case (1 t)
  have ( $\int^{+u} \text{ennreal} (\text{indicator } (\{0..\} \times \{0..\}) (t,u) * t \text{ powr } (a - 1) * u \text{ powr } (b - 1) / \exp (t + u)) \partial \text{distr lborel borel } ((+(-t)))) =$ 
    ( $\int^{+u} \text{ennreal} (\text{indicator } (\{0..\} \times \{t..\}) (t,u) * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1) / \exp u) \partial \text{lborel}$ )
  by (subst nn_integral_distr) (auto intro!: nn_integral_cong simp: indicator_def)
  thus ?case by (subst (asm) lborel_distr_plus)
  qed
  also have ... = ( $\int^{+u} \int^{+t} \text{ennreal} (\text{indicator } (\{0..\} \times \{t..\}) (t,u) * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1) / \exp u) \partial \text{lborel} \partial \text{lborel}$ )
  by (subst lborel_pair.Fubini')
  (auto simp: case_prod_unfold indicator_def cong: measurable_cong_sets)
  also have ... = ( $\int^{+u} \int^{+t} \text{ennreal} (\text{indicator } \{0..u\} t * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1)) * \text{ennreal} (\text{indicator } \{0..\} u / \exp u) \partial \text{lborel} \partial \text{lborel}$ )
  by (intro nn_integral_cong) (auto simp: indicator_def ennreal_mult' [symmetric])
  also have ... = ( $\int^{+u} (\int^{+t} \text{ennreal} (\text{indicator } \{0..u\} t * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1)) \partial \text{lborel}) * \text{ennreal} (\text{indicator } \{0..\} u / \exp u) \partial \text{lborel}$ )
  by (subst nn_integral_multc [symmetric]) auto
  also have ... = ( $\int^{+u} (\int^{+t} \text{ennreal} (\text{indicator } \{0..u\} t * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1)) \partial \text{lborel}) * \text{ennreal} (\text{indicator } \{0<..\} u / \exp u) \partial \text{lborel}$ )
  by (intro nn_integral_cong_AE eventually_mono[OF AE_lborel_singleton[of 0]])
  (auto simp: indicator_def)
  also have ... = ( $\int^{+u} \text{ennreal } B * \text{ennreal} (\text{indicator } \{0..\} u / \exp u * u \text{ powr } (a + b - 1)) \partial \text{lborel}$ )
  proof (intro nn_integral_cong, goal_cases)
  case (1 u)
  show ?case

```

```

proof (cases u > 0)
  case True
    have ( $\int^{+t}. \text{ennreal } (\text{indicator } \{0..u\} t * t \text{ powr } (a - 1) * (u - t) \text{ powr } (b - 1)) \partial \text{lborel}$ ) =
      ( $\int^{+t}. \text{ennreal } (\text{indicator } \{0..1\} t * (u * t) \text{ powr } (a - 1) * (u - u * t) \text{ powr } (b - 1))$ )
       $\partial \text{distr lborel borel } ((*) (1 / u))$  (is _ = nn_integral _ ?f)
    using True
    by (subst nn_integral_distr) (auto simp: indicator_def field_simps intro!: nn_integral_cong)
    also have  $\text{distr lborel borel } ((*) (1 / u)) = \text{density lborel } (\lambda_. u)$ 
    using  $\langle u > 0 \rangle$  by (subst lborel_distr_mult) auto
    also have nn_integral ... ?f = ( $\int^{+x}. \text{ennreal } (\text{indicator } \{0..1\} x * (u * (u * x) \text{ powr } (a - 1) * (u * (1 - x)) \text{ powr } (b - 1))) \partial \text{lborel}$ ) using
 $\langle u > 0 \rangle$ 
    by (subst nn_integral_density) (auto simp: ennreal_mult' [symmetric] algebra_simps)
    also have ... = ( $\int^{+x}. \text{ennreal } (u \text{ powr } (a + b - 1)) * \text{ennreal } (\text{indicator } \{0..1\} x * x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1)) \partial \text{lborel}$ ) using  $\langle u > 0 \rangle$  a b
    by (intro nn_integral_cong)
      (auto simp: indicator_def powr_mult powr_add powr_diff mult_ac ennreal_mult' [symmetric])
    also have ... = ennreal (u powr (a + b - 1)) *
      ( $\int^{+x}. \text{ennreal } (\text{indicator } \{0..1\} x * x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1)) \partial \text{lborel}$ )
    by (subst nn_integral_cmult) auto
    also have ( $(\lambda x. x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1)) \text{ has\_integral } \text{integral } \{0..1\} (\lambda x. x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1)))$ )
    {0..1}
    using a b by (intro integrable_integral integrable_Beta)
    from nn_integral_has_integral_lebesgue[OF _ this] a b
    have ( $\int^{+x}. \text{ennreal } (\text{indicator } \{0..1\} x * x \text{ powr } (a - 1) * (1 - x) \text{ powr } (b - 1)) \partial \text{lborel}$ ) = B by (simp add: mult_ac B_def)
    finally show ?thesis using  $\langle u > 0 \rangle$  by (simp add: ennreal_mult' [symmetric] mult_ac)
  qed auto
qed
also have ... = ennreal B * ennreal (Gamma (a + b))
using a b by (subst nn_integral_cmult) (auto simp: Gamma_conv_nn_integral_real)
also have ... = ennreal (B * Gamma (a + b))
by (subst (1 2) mult.commute, intro ennreal_mult' [symmetric]) (use a b in auto)
finally have B = Beta a b using a b Gamma_real_pos[of a + b]
by (subst (asm) ennreal_inj) (auto simp: field_simps Beta_def Gamma_eq_zero_iff)
moreover have ( $\lambda t. t \text{ powr } (a - 1) * (1 - t) \text{ powr } (b - 1)$ ) integrable_on {0..1}

```

by (intro integrable_Beta' a b)
ultimately show ?thesis by (simp add: has_integral_iff B_def)
qed

9.10.12 The Weierstraß product formula for the sine

theorem *sin_product_formula_complex*:

fixes $z :: \text{complex}$
shows $(\lambda n. \text{of_real } \pi * z * (\prod_{k=1..n}. 1 - z^2 / \text{of_nat } k^2)) \longrightarrow \text{sin}(\text{of_real } \pi * z)$

proof –

let $?f = \text{rGamma_series_Weierstrass}$

have $(\lambda n. (- \text{of_real } \pi * \text{inverse } z) * (?f z n * ?f (-z) n)) \longrightarrow (- \text{of_real } \pi * \text{inverse } z) * (\text{rGamma } z * \text{rGamma } (-z))$

by (intro tendsto_intros rGamma_Weierstrass_complex)

also have $(\lambda n. (- \text{of_real } \pi * \text{inverse } z) * (?f z n * ?f (-z) n)) = (\lambda n. \text{of_real } \pi * z * (\prod_{k=1..n}. 1 - z^2 / \text{of_nat } k^2))$

proof

fix $n :: \text{nat}$

have $(- \text{of_real } \pi * \text{inverse } z) * (?f z n * ?f (-z) n) = \text{of_real } \pi * z * (\prod_{k=1..n}. (\text{of_nat } k - z) * (\text{of_nat } k + z) / \text{of_nat } k^2)$

by (simp add: rGamma_series_Weierstrass_def mult_ac exp_minus divide_simps prod.distrib[symmetric] power2_eq_square)

also have $(\prod_{k=1..n}. (\text{of_nat } k - z) * (\text{of_nat } k + z) / \text{of_nat } k^2) = (\prod_{k=1..n}. 1 - z^2 / \text{of_nat } k^2)$

by (intro prod.cong) (simp_all add: power2_eq_square field_simps)

finally show $(- \text{of_real } \pi * \text{inverse } z) * (?f z n * ?f (-z) n) = \text{of_real } \pi * z * \dots$

by (simp add: field_split_simps)

qed

also have $(- \text{of_real } \pi * \text{inverse } z) * (\text{rGamma } z * \text{rGamma } (-z)) = \text{sin}(\text{of_real } \pi * z)$

by (subst rGamma_reflection_complex') (simp add: field_split_simps)

finally show ?thesis .

qed

lemma *sin_product_formula_real*:

$(\lambda n. \pi * (x :: \text{real}) * (\prod_{k=1..n}. 1 - x^2 / \text{of_nat } k^2)) \longrightarrow \text{sin}(\pi * x)$

proof –

from *sin_product_formula_complex*[of *of_real x*]

have $(\lambda n. \text{of_real } \pi * \text{of_real } x * (\prod_{k=1..n}. 1 - (\text{of_real } x)^2 / (\text{of_nat } k)^2)) \longrightarrow \text{sin}(\text{of_real } \pi * \text{of_real } x :: \text{complex})$ (is ?f \longrightarrow ?y) .

also have ?f = $(\lambda n. \text{of_real } (\pi * x * (\prod_{k=1..n}. 1 - x^2 / (\text{of_nat } k^2))))$

by *simp*

also have ?y = $\text{of_real}(\text{sin}(\pi * x))$ **by** (*simp only*: *sin_of_real* [*symmetric*] *of_real_mult*)

finally show ?thesis **by** (*subst* (*asm*) *tendsto_of_real_iff*)

3112

qed

lemma *sin_product_formula_real'*:

assumes $x \neq (0::\text{real})$

shows $(\lambda n. (\prod_{k=1..n} 1 - x^2 / \text{of_nat } k^2)) \longrightarrow \sin(\pi * x) / (\pi * x)$

using *tendsto_divide[OF sin_product_formula_real[of x] tendsto_const[of pi * x]] assms*

by *simp*

theorem *wallis*: $(\lambda n. \prod_{k=1..n} (4 * \text{real } k^2) / (4 * \text{real } k^2 - 1)) \longrightarrow \pi / 2$

proof –

from *tendsto_inverse[OF tendsto_mult[OF*

sin_product_formula_real[of 1/2] tendsto_const[of 2/pi]]

have $(\lambda n. (\prod_{k=1..n} \text{inverse}(1 - (1/2)^2 / (\text{real } k^2)))) \longrightarrow \pi/2$

by (*simp add: prod_inversef [symmetric]*)

also have $(\lambda n. (\prod_{k=1..n} \text{inverse}(1 - (1/2)^2 / (\text{real } k^2)))) =$

$(\lambda n. (\prod_{k=1..n} (4 * \text{real } k^2) / (4 * \text{real } k^2 - 1)))$

by (*intro ext prod.cong refl (simp add: field_split_simps)*)

finally show *?thesis* .

qed

9.10.13 The Solution to the Basel problem

theorem *inverse_squares_sums*: $(\lambda n. 1 / (n + 1)^2) \text{ sums } (\pi^2 / 6)$

proof –

define *P* **where** $P \ x \ n = (\prod_{k=1..n} 1 - x^2 / \text{of_nat } k^2)$ **for** $x :: \text{real}$ **and** n

define *K* **where** $K = (\sum n. \text{inverse}(\text{real_of_nat } (\text{Suc } n))^2)$

define *f* **where** [*abs_def*]: $f \ x = (\sum n. P \ x \ n / \text{of_nat } (\text{Suc } n)^2)$ **for** x

define *g* **where** [*abs_def*]: $g \ x = (1 - \sin(\pi * x) / (\pi * x))$ **for** x

have *sums*: $(\lambda n. P \ x \ n / \text{of_nat } (\text{Suc } n)^2) \text{ sums } (\text{if } x = 0 \text{ then } K \text{ else } g \ x / x^2)$ **for** x

proof (*cases x = 0*)

assume $x: x = 0$

have *summable* $(\lambda n. \text{inverse}((\text{real_of_nat } (\text{Suc } n))^2))$

using *inverse_power_summable[of 2]* **by** (*subst summable_Suc_iff*) *simp*

thus *?thesis* **by** (*simp add: x_g_def P_def K_def inverse_eq_divide power_divide summable_sums*)

next

assume $x: x \neq 0$

have $(\lambda n. P \ x \ n - P \ x \ (\text{Suc } n)) \text{ sums } (P \ x \ 0 - \sin(\pi * x) / (\pi * x))$

unfolding *P_def* **using** x **by** (*intro telescope_sums' sin_product_formula_real'*)

also have $(\lambda n. P \ x \ n - P \ x \ (\text{Suc } n)) = (\lambda n. (x^2 / \text{of_nat } (\text{Suc } n)^2) * P \ x \ n)$

unfolding *P_def* **by** (*simp add: prod.nat_ivl_Suc' algebra_simps*)

also have $P \ x \ 0 = 1$ **by** (*simp add: P_def*)

finally have $(\lambda n. x^2 / (\text{of_nat } (\text{Suc } n))^2 * P \ x \ n) \text{ sums } (1 - \sin(\pi * x) / (\pi * x))$.

```

    from sums_divide[OF this, of  $x^2$ ]  $x$  show ?thesis unfolding  $g\_def$  by simp
qed

have continuous_on (ball 0 1)  $f$ 
proof (rule uniform_limit_theorem; (intro always_eventually_allI)?)
  show uniform_limit (ball 0 1) ( $\lambda n x. \sum_{k < n}. P\ x\ k / of\_nat\ (Suc\ k)^2$ )  $f$ 
    sequentially
  proof (unfold  $f\_def$ , rule Weierstrass_m_test)
    fix  $n :: nat$  and  $x :: real$  assume  $x : x \in ball\ 0\ 1$ 
    {
      fix  $k :: nat$  assume  $k : k \geq 1$ 
      from  $x$  have  $x^2 < 1$  by (auto simp: abs_square_less_1)
      also from  $k$  have  $\dots \leq of\_nat\ k^2$  by simp
      finally have  $(1 - x^2 / of\_nat\ k^2) \in \{0..1\}$  using  $k$ 
        by (simp_all add: field_simps del: of_nat_Suc)
    }
    hence  $(\prod_{k=1..n}. abs\ (1 - x^2 / of\_nat\ k^2)) \leq (\prod_{k=1..n}. 1)$  by (intro
prod_mono) simp
    thus  $norm\ (P\ x\ n / (of\_nat\ (Suc\ n)^2)) \leq 1 / of\_nat\ (Suc\ n)^2$ 
      unfolding  $P\_def$  by (simp add: field_simps abs_prod del: of_nat_Suc)
    qed (subst summable_Suc_iff, insert inverse_power_summable[of 2], simp
add: inverse_eq_divide)
    qed (auto simp:  $P\_def$  intro!: continuous_intros)
    hence isCont  $f\ 0$  by (subst (asm) continuous_on_eq_continuous_at) simp_all
    hence  $(f - 0 \rightarrow f\ 0)$  by (simp add: isCont_def)
    also have  $f\ 0 = K$  unfolding  $f\_def\ P\_def\ K\_def$  by (simp add: inverse_eq_divide
power_divide)
    finally have  $f - 0 \rightarrow K$  .

moreover have  $f - 0 \rightarrow pi^2 / 6$ 
proof (rule Lim_transform_eventually)
  define  $f'$  where [abs_def]:  $f'\ x = (\sum n. - sin\_coeff\ (n+3) * pi^{(n+2)} * x^n)$  for  $x$ 
  have eventually  $(\lambda x. x \neq (0::real))$  (at 0)
    by (auto simp add: eventually_at intro!: exI[of _ 1])
  thus eventually  $(\lambda x. f'\ x = f\ x)$  (at 0)
  proof eventually_elim
    fix  $x :: real$  assume  $x : x \neq 0$ 
    have  $sin\_coeff\ 1 = (1 :: real)$   $sin\_coeff\ 2 = (0::real)$  by (simp_all add:
sin_coeff_def)
    with sums_split_initial_segment[OF sums_minus[OF sin_converges], of 3
 $pi*x$ ]
    have  $(\lambda n. - (sin\_coeff\ (n+3) * (pi*x)^{(n+3)}))\ sums\ (pi * x - sin\ (pi*x))$ 
      by (simp add: eval_nat_numeral)
    from sums_divide[OF this, of  $x^3 * pi$ ]  $x$ 
    have  $(\lambda n. - (sin\_coeff\ (n+3) * pi^{(n+2)} * x^n))\ sums\ ((1 - sin\ (pi*x)) / (pi*x)) / x^2$ 
      by (simp add: field_split_simps eval_nat_numeral)
    with  $x$  have  $(\lambda n. - (sin\_coeff\ (n+3) * pi^{(n+2)} * x^n))\ sums\ (g\ x / x^2)$ 

```

by (simp add: g_def)
 hence $f' x = g x / x^2$ by (simp add: sums_iff f'_def)
 also have $\dots = f x$ using sums[of x] x by (simp add: sums_iff g_def f'_def)
 finally show $f' x = f x$.
 qed

have isCont f' 0 unfolding f'_def
 proof (intro isCont_powser_converges_everywhere)
 fix x :: real show summable ($\lambda n. -\sin_coeff (n+3) * \pi^{n+2} * x^n$)
 proof (cases x = 0)
 assume x: $x \neq 0$
 from summable_divide[OF sums_summable[OF sums_split_initial_segment[OF
 sin_converges[of pi*x], of 3], of -pi*x^3] x
 show ?thesis by (simp add: field_split_simps eval_nat_numeral)
 qed (simp only: summable_0_powser)
 qed
 hence $f' - 0 \rightarrow f' 0$ by (simp add: isCont_def)
 also have $f' 0 = \pi * \pi / \text{fact } 3$ unfolding f'_def
 by (subst powser_zero) (simp add: sin_coeff_def)
 finally show $f' - 0 \rightarrow \pi^2 / 6$ by (simp add: eval_nat_numeral)
 qed

ultimately have $K = \pi^2 / 6$ by (rule LIM_unique)
 moreover from inverse_power_summable[of 2]
 have summable ($\lambda n. (\text{inverse } (\text{real_of_nat } (\text{Suc } n)))^2$)
 by (subst summable_Suc_iff) (simp add: power_inverse)
 ultimately show ?thesis unfolding K_def
 by (auto simp add: sums_iff power_divide inverse_eq_divide)
 qed

end

theory Interval_Integral

imports Equivalence_Lebesgue_Henstock_Integration
 begin

definition einterval a b = {x. a < ereal x \wedge ereal x < b}

lemma einterval_eq[simp]:

shows einterval_eq_Icc: einterval (ereal a) (ereal b) = {a <.. $<$ b}
 and einterval_eq_Ici: einterval (ereal a) ∞ = {a <.. ∞ }
 and einterval_eq_Iic: einterval ($-\infty$) (ereal b) = {.. $<$ b}
 and einterval_eq_UNIV: einterval ($-\infty$) ∞ = UNIV
 by (auto simp: einterval_def)

lemma einterval_same: einterval a a = {}

by (auto simp: einterval_def)

lemma *einterval_iff*: $x \in \text{einterval } a \ b \longleftrightarrow a < \text{ereal } x \wedge \text{ereal } x < b$
by (*simp add: einterval_def*)

lemma *einterval_nonempty*: $a < b \implies \exists c. c \in \text{einterval } a \ b$
by (*cases a b rule: ereal2_cases, auto simp: einterval_def intro!: dense_gt_ex lt_ex*)

lemma *open_einterval[simp]*: *open* (*einterval* $a \ b$)
by (*cases a b rule: ereal2_cases*)
(auto simp: einterval_def intro!: open_Collect_conj open_Collect_less continuous_intros)

lemma *borel_einterval[measurable]*: *einterval* $a \ b \in \text{sets borel}$
unfolding *einterval_def* **by** *measurable*

9.10.14 Approximating a (possibly infinite) interval

lemma *filterlim_sup1*: $(\text{LIM } x \ F. f \ x \ :> \ G1) \implies (\text{LIM } x \ F. f \ x \ :> (\text{sup } G1 \ G2))$
unfolding *filterlim_def* **by** (*auto intro: le_supI1*)

lemma *ereal_incseq_approx*:
fixes $a \ b :: \text{ereal}$
assumes $a < b$
obtains $X :: \text{nat} \Rightarrow \text{real}$ **where** $\text{incseq } X \wedge i. a < X \ i \wedge i. X \ i < b \ X \longrightarrow b$
proof (*cases b*)
case *PInf*
with $\langle a < b \rangle$ **have** $a = -\infty \vee (\exists r. a = \text{ereal } r)$
by (*cases a*) *auto*
moreover **have** $(\lambda x. \text{ereal } (\text{real } (\text{Suc } x))) \longrightarrow \infty$
by (*simp add: Lim_PInfthy filterlim_sequentially_Suc*) (*metis le_SucI of_nat_Suc of_nat_mono order_trans real_arch_simple*)
moreover **have** $\bigwedge r. (\lambda x. \text{ereal } (r + \text{real } (\text{Suc } x))) \longrightarrow \infty$
by (*simp add: filterlim_sequentially_Suc Lim_PInfthy*) (*metis add commute diff_le_eq nat_ceiling_le_eq*)
ultimately show *thesis*
by (*intro that[of $\lambda i. \text{real_of_ereal } a + \text{Suc } i$]*)
(auto simp: incseq_def PInf)
next
case (*real b'*)
define d **where** $d = b' - (\text{if } a = -\infty \text{ then } b' - 1 \text{ else } \text{real_of_ereal } a)$
with $\langle a < b \rangle$ **have** $a' : 0 < d$
by (*cases a*) (*auto simp: real*)
moreover
have $\bigwedge i \ r. r < b' \implies (b' - r) * 1 < (b' - r) * \text{real } (\text{Suc } (\text{Suc } i))$
by (*intro mult_strict_left_mono*) *auto*
with $\langle a < b \rangle$ a' **have** $\bigwedge i. a < \text{ereal } (b' - d / \text{real } (\text{Suc } (\text{Suc } i)))$
by (*cases a*) (*auto simp: real_d_def field_simps*)
moreover
have $(\lambda i. b' - d / \text{real } i) \longrightarrow b'$

by (*force intro: tendsto_eq_intros tendsto_divide_0*[*OF tendsto_const*] *filterlim_sup1*
simp: at_infinity_eq_at_top_bot filterlim_real_sequentially)
then have $(\lambda i. b' - d / \text{Suc } (\text{Suc } i)) \longrightarrow b'$
by (*blast intro: dest: filterlim_sequentially_Suc* [*THEN iffD2*])
ultimately show thesis
by (*intro that*[*of* $\lambda i. b' - d / \text{Suc } (\text{Suc } i)$])
(auto simp: real_incseq_def intro!: divide_left_mono)
qed (*use* $\langle a < b \rangle$ **in** *auto*)

lemma *ereal_decseq_approx*:

fixes $a b :: \text{ereal}$

assumes $a < b$

obtains $X :: \text{nat} \Rightarrow \text{real}$ **where**

$\text{decseq } X \wedge i. a < X i \wedge i. X i < b \longrightarrow a$

proof –

have $-b < -a$ **using** $\langle a < b \rangle$ **by** *simp*

from *ereal_incseq_approx*[*OF this*] **obtain** X **where**

incseq X

$\wedge i. -b < \text{ereal } (X i)$

$\wedge i. \text{ereal } (X i) < -a$

$(\lambda x. \text{ereal } (X x)) \longrightarrow -a$

by *auto*

then show thesis

apply (*intro that*[*of* $\lambda i. -X i$])

apply (*auto simp: decseq_def incseq_def simp flip: uminus_ereal_simps*)

apply (*metis* *ereal_minus_less_minus* *ereal_uminus_uminus* *ereal_Lim_uminus*)
done

qed

proposition *einterval_Icc_approximation*:

fixes $a b :: \text{ereal}$

assumes $a < b$

obtains $u l :: \text{nat} \Rightarrow \text{real}$ **where**

einterval $a b = (\bigcup i. \{l i .. u i\})$

incseq u *decseq* $l \wedge i. l i < u i \wedge i. a < l i \wedge i. u i < b$

$l \longrightarrow a \quad u \longrightarrow b$

proof –

from *dense*[*OF* $\langle a < b \rangle$] **obtain** c **where** $a < c < b$ **by** *safe*

from *ereal_incseq_approx*[*OF* $\langle c < b \rangle$] **obtain** u **where** u :

incseq u

$\wedge i. c < \text{ereal } (u i)$

$\wedge i. \text{ereal } (u i) < b$

$(\lambda x. \text{ereal } (u x)) \longrightarrow b$

by *auto*

from *ereal_decseq_approx*[*OF* $\langle a < c \rangle$] **obtain** l **where** l :

decseq l

$\wedge i. a < \text{ereal } (l i)$

$\wedge i. \text{ereal } (l i) < c$


```

  (λx. ereal (l x)) ⟶ a
  by auto
  have einterval a b = (⋃ i. {l i .. u i})
  proof (auto simp: einterval_iff)
    fix x assume a < ereal x ereal x < b
    have eventually (λi. ereal (l i) < ereal x) sequentially
      using l(4) ⟨a < ereal x⟩ by (rule order_tendstoD)
    moreover
    have eventually (λi. ereal x < ereal (u i)) sequentially
      using u(4) ⟨ereal x < b⟩ by (rule order_tendstoD)
    ultimately have eventually (λi. l i < x ∧ x < u i) sequentially
      by eventually_elim auto
    then show ∃ i. l i ≤ x ∧ x ≤ u i
      by (auto intro: less_imp_le simp: eventually_sequentially)
  next
    fix x i assume l i ≤ x x ≤ u i
    with ⟨a < ereal (l i)⟩ ⟨ereal (u i) < b⟩
    show a < ereal x ereal x < b
      by (auto simp flip: ereal_less_eq(3))
  qed
  moreover { fix i from less_trans[OF ⟨l i < c⟩ ⟨c < u i⟩] have l i < u i by
simp }
  ultimately show thesis
    by (simp add: l that u)
  qed

```

definition *interval_lebesgue_integral* :: real measure ⇒ ereal ⇒ ereal ⇒ (real ⇒ 'a) ⇒ 'a::{banach, second_countable_topology} **where**
interval_lebesgue_integral M a b f =
 (if a ≤ b then (LINT x:einterval a b|M. f x) else - (LINT x:einterval b a|M. f x))

syntax

```

_asciï_interval_lebesgue_integral :: pptrn ⇒ real ⇒ real ⇒ real measure ⇒ real
⇒ real
((5LINT _=.._|_.) [0,60,60,61,100] 60)

```

translations

```

LINT x=a..b|M. f == CONST interval_lebesgue_integral M a b (λx. f)

```

definition *interval_lebesgue_integrable* :: real measure ⇒ ereal ⇒ ereal ⇒ (real ⇒ 'a::{banach, second_countable_topology}) ⇒ bool **where**
interval_lebesgue_integrable M a b f =
 (if a ≤ b then set_integrable M (einterval a b) f else set_integrable M (einterval b a) f)

syntax

```

_asciï_interval_lebesgue_borel_integral :: pptrn ⇒ real ⇒ real ⇒ real ⇒ real

```

((4LBINT _=... _) [0,60,60,61] 60)

translations

LBINT $x=a..b$. $f ==$ CONST *interval_lebesgue_integral* CONST *lborel a b* (λx . f)

9.10.15 Basic properties of integration over an interval

lemma *interval_lebesgue_integral_cong*:

$a \leq b \implies (\bigwedge x. x \in \text{einterval } a \ b \implies f \ x = g \ x) \implies \text{einterval } a \ b \in \text{sets } M \implies$
 $\text{interval_lebesgue_integral } M \ a \ b \ f = \text{interval_lebesgue_integral } M \ a \ b \ g$
by (*auto intro: set_lebesgue_integral_cong simp: interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_cong_AE*:

$f \in \text{borel_measurable } M \implies g \in \text{borel_measurable } M \implies$
 $a \leq b \implies \text{AE } x \in \text{einterval } a \ b \ \text{in } M. f \ x = g \ x \implies \text{einterval } a \ b \in \text{sets } M \implies$
 $\text{interval_lebesgue_integral } M \ a \ b \ f = \text{interval_lebesgue_integral } M \ a \ b \ g$
by (*auto intro: set_lebesgue_integral_cong_AE simp: interval_lebesgue_integral_def*)

lemma *interval_integrable_mirror*:

shows *interval_lebesgue_integrable lborel a b* ($\lambda x. f \ (-x)$) \longleftrightarrow
interval_lebesgue_integrable lborel (-b) (-a) f

proof –

have *: *indicator (einterval a b) (- x) = (indicator (einterval (-b) (-a)) x ::*
real)

for $a \ b :: \text{ereal}$ **and** $x :: \text{real}$

by (*cases a b rule: ereal2_cases*) (*auto simp: einterval_def split: split_indicator*)

show ?thesis

unfolding *interval_lebesgue_integrable_def*

using *lborel_integrable_real_affine_iff[symmetric, of -1 λx . indicator (einterval*
*_ _) x *_R f x 0]*

by (*simp add: * set_integrable_def*)

qed

lemma *interval_lebesgue_integral_add* [*intro, simp*]:

fixes $M \ a \ b \ f$

assumes *interval_lebesgue_integrable M a b f* *interval_lebesgue_integrable M a*
b g

shows *interval_lebesgue_integrable M a b* ($\lambda x. f \ x + g \ x$)

and *interval_lebesgue_integral M a b* ($\lambda x. f \ x + g \ x$) =

interval_lebesgue_integral M a b f + interval_lebesgue_integral M a b g

using *assms* **by** (*auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def*
field_simps)

lemma *interval_lebesgue_integral_diff* [*intro, simp*]:

fixes $M \ a \ b \ f$

assumes *interval_lebesgue_integrable M a b f*

interval_lebesgue_integrable M a b g

shows *interval_lebesgue_integrable M a b* ($\lambda x. f \ x - g \ x$) **and**

$$\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. f \ x - g \ x) =$$

$$\text{interval_lebesgue_integral } M \ a \ b \ f - \text{interval_lebesgue_integral } M \ a \ b \ g$$
using *assms* **by** (auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def field_simps)

lemma *interval_lebesgue_integrable_mult_right* [intro, simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, second_countable_topology}\}$
shows $(c \neq 0 \implies \text{interval_lebesgue_integrable } M \ a \ b \ f) \implies$
 $\text{interval_lebesgue_integrable } M \ a \ b \ (\lambda x. c * f \ x)$
by (simp add: interval_lebesgue_integrable_def)

lemma *interval_lebesgue_integrable_mult_left* [intro, simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, second_countable_topology}\}$
shows $(c \neq 0 \implies \text{interval_lebesgue_integrable } M \ a \ b \ f) \implies$
 $\text{interval_lebesgue_integrable } M \ a \ b \ (\lambda x. f \ x * c)$
by (simp add: interval_lebesgue_integrable_def)

lemma *interval_lebesgue_integrable_divide* [intro, simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, field, second_countable_topology}\}$
shows $(c \neq 0 \implies \text{interval_lebesgue_integrable } M \ a \ b \ f) \implies$
 $\text{interval_lebesgue_integrable } M \ a \ b \ (\lambda x. f \ x / c)$
by (simp add: interval_lebesgue_integrable_def)

lemma *interval_lebesgue_integral_mult_right* [simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, second_countable_topology}\}$
shows $\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. c * f \ x) =$
 $c * \text{interval_lebesgue_integral } M \ a \ b \ f$
by (simp add: interval_lebesgue_integral_def)

lemma *interval_lebesgue_integral_mult_left* [simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, second_countable_topology}\}$
shows $\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. f \ x * c) =$
 $\text{interval_lebesgue_integral } M \ a \ b \ f * c$
by (simp add: interval_lebesgue_integral_def)

lemma *interval_lebesgue_integral_divide* [simp]:
fixes $M \ a \ b \ c$ **and** $f :: \text{real} \Rightarrow 'a::\{\text{banach, real_normed_field, field, second_countable_topology}\}$
shows $\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. f \ x / c) =$
 $\text{interval_lebesgue_integral } M \ a \ b \ f / c$
by (simp add: interval_lebesgue_integral_def)

lemma *interval_lebesgue_integral_uminus*:
 $\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. - f \ x) = - \text{interval_lebesgue_integral } M$
 $a \ b \ f$
by (auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def set_lebesgue_integral_def)

lemma *interval_lebesgue_integral_of_real*:
 $\text{interval_lebesgue_integral } M \ a \ b \ (\lambda x. \text{complex_of_real } (f \ x)) =$

$of_real (interval_lebesgue_integral M a b f)$
unfolding $interval_lebesgue_integral_def$
by $(auto simp: interval_lebesgue_integral_def set_integral_complex_of_real)$

lemma $interval_lebesgue_integral_le_eq$:
fixes $a b f$
assumes $a \leq b$
shows $interval_lebesgue_integral M a b f = (LINT x : einterval a b | M. f x)$
using $assms$ **by** $(auto simp: interval_lebesgue_integral_def)$

lemma $interval_lebesgue_integral_gt_eq$:
fixes $a b f$
assumes $a > b$
shows $interval_lebesgue_integral M a b f = -(LINT x : einterval b a | M. f x)$
using $assms$ **by** $(auto simp: interval_lebesgue_integral_def less_imp_le einterval_def)$

lemma $interval_lebesgue_integral_gt_eq'$:
fixes $a b f$
assumes $a > b$
shows $interval_lebesgue_integral M a b f = - interval_lebesgue_integral M b a f$
using $assms$ **by** $(auto simp: interval_lebesgue_integral_def less_imp_le einterval_def)$

lemma $interval_integral_endpoints_same [simp]$: $(LBINT x=a..a. f x) = 0$
by $(simp add: interval_lebesgue_integral_def set_lebesgue_integral_def einterval_same)$

lemma $interval_integral_endpoints_reverse$: $(LBINT x=a..b. f x) = -(LBINT x=b..a. f x)$
by $(cases a b rule: linorder_cases) (auto simp: interval_lebesgue_integral_def set_lebesgue_integral_def einterval_same)$

lemma $interval_integrable_endpoints_reverse$:
 $interval_lebesgue_integrable lborel a b f \longleftrightarrow$
 $interval_lebesgue_integrable lborel b a f$
by $(cases a b rule: linorder_cases) (auto simp: interval_lebesgue_integrable_def einterval_same)$

lemma $interval_integral_reflect$:
 $(LBINT x=a..b. f x) = (LBINT x=-b..-a. f (-x))$
proof $(induct a b rule: linorder_wlog)$
case $(sym a b)$ **then show** $?case$
by $(auto simp: interval_lebesgue_integral_def interval_integrable_endpoints_reverse split: if_split_asm)$

next
case $(le a b)$
have $(LBINT x:\{x. - x \in einterval a b\}. f (- x)) = (LBINT x:einterval (- b)$

```

(- a). f (- x))
  unfolding interval_lebesgue_integrable_def set_lebesgue_integral_def einterval_def
  by (metis (lifting) ereal_less_uminus_reorder ereal_uminus_less_reorder indicator_simps mem_Collect_eq uminus_ereal_simps(1))
  then show ?case
    unfolding interval_lebesgue_integral_def
    by (subst set_integral_reflect) (simp add: le)
qed

```

```

lemma interval_lebesgue_integral_0_infty:
  interval_lebesgue_integrable M 0 ∞ f ↔ set_integrable M {0<..} f
  interval_lebesgue_integral M 0 ∞ f = (LINT x:{0<..}|M. f x)
  unfolding zero_ereal_def
  by (auto simp: interval_lebesgue_integral_le_eq interval_lebesgue_integrable_def)

```

```

lemma interval_integral_to_infinity_eq: (LINT x=ereal a..∞ | M. f x) = (LINT
x : {a<..} | M. f x)
  unfolding interval_lebesgue_integral_def by auto

```

```

proposition interval_integrable_to_infinity_eq: (interval_lebesgue_integrable M
a ∞ f) =
  (set_integrable M {a<..} f)
  unfolding interval_lebesgue_integrable_def by auto

```

9.10.16 Basic properties of integration over an interval wrt lebesgue measure

```

lemma interval_integral_zero [simp]:
  fixes a b :: ereal
  shows (LBINT x=a..b. 0) = 0
  unfolding interval_lebesgue_integral_def set_lebesgue_integral_def einterval_eq
  by simp

```

```

lemma interval_integral_const [intro, simp]:
  fixes a b c :: real
  shows interval_lebesgue_integrable lborel a b (λx. c) and (LBINT x=a..b. c) =
c * (b - a)
  unfolding interval_lebesgue_integral_def interval_lebesgue_integrable_def einterval_eq
  by (auto simp: less_imp_le field_simps measure_def set_integrable_def set_lebesgue_integral_def)

```

```

lemma interval_integral_cong_AE:
  assumes [measurable]: f ∈ borel_measurable borel g ∈ borel_measurable borel
  assumes AE x ∈ einterval (min a b) (max a b) in lborel. f x = g x
  shows interval_lebesgue_integral lborel a b f = interval_lebesgue_integral lborel
a b g
  using assms
  by (auto simp: interval_lebesgue_integral_def max_def min_def intro!: set_lebesgue_integral_cong_AE)

```

lemma *interval_integral_cong*:

assumes $\bigwedge x. x \in \text{einterval } (\text{min } a \ b) \ (\text{max } a \ b) \implies f \ x = g \ x$

shows $\text{interval_lebesgue_integral } \text{lborel } a \ b \ f = \text{interval_lebesgue_integral } \text{lborel } a \ b \ g$

using *assms* **by** (*simp add: interval_lebesgue_integral_def set_lebesgue_integral_cong*)

lemma *interval_lebesgue_integrable_cong_AE*:

$f \in \text{borel_measurable } \text{lborel} \implies g \in \text{borel_measurable } \text{lborel} \implies$

$\text{AE } x \in \text{einterval } (\text{min } a \ b) \ (\text{max } a \ b) \ \text{in } \text{lborel}. f \ x = g \ x \implies$

$\text{interval_lebesgue_integrable } \text{lborel } a \ b \ f = \text{interval_lebesgue_integrable } \text{lborel } a \ b \ g$

apply (*simp add: interval_lebesgue_integrable_def*)

apply (*intro conjI impI set_integrable_cong_AE*)

apply (*auto simp: min_def max_def*)

done

lemma *interval_integrable_abs_iff*:

fixes $f :: \text{real} \Rightarrow \text{real}$

shows $f \in \text{borel_measurable } \text{lborel} \implies$

$\text{interval_lebesgue_integrable } \text{lborel } a \ b \ (\lambda x. |f \ x|) = \text{interval_lebesgue_integrable } \text{lborel } a \ b \ f$

unfolding *interval_lebesgue_integrable_def*

by (*simp add: set_integrable_abs_iff'*)

lemma *interval_integral_Icc*:

fixes $a \ b :: \text{real}$

shows $a \leq b \implies (\text{LBINT } x=a..b. f \ x) = (\text{LBINT } x : \{a..b\}. f \ x)$

by (*auto intro!: set_integral_discrete_difference[where X={a, b}]*
simp add: interval_lebesgue_integral_def)

lemma *interval_integral_Icc'*:

$a \leq b \implies (\text{LBINT } x=a..b. f \ x) = (\text{LBINT } x : \{x. a \leq \text{ereal } x \wedge \text{ereal } x \leq b\}. f \ x)$

by (*auto intro!: set_integral_discrete_difference[where X={real_of_ereal a, real_of_ereal b}]*

simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ioc*:

$a \leq b \implies (\text{LBINT } x=a..b. f \ x) = (\text{LBINT } x : \{a<..b\}. f \ x)$

by (*auto intro!: set_integral_discrete_difference[where X={a, b}]*
simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ioc'*:

$a \leq b \implies (\text{LBINT } x=a..b. f \ x) = (\text{LBINT } x : \{x. a < \text{ereal } x \wedge \text{ereal } x \leq b\}. f \ x)$

by (*auto intro!: set_integral_discrete_difference[where X={real_of_ereal a, real_of_ereal b}]*)

simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ico:*

$a \leq b \implies (LBINT\ x=a..b.\ f\ x) = (LBINT\ x : \{a..<b\}.\ f\ x)$
by (*auto intro!: set_integral_discrete_difference*[**where** $X=\{a, b\}$]
simp add: interval_lebesgue_integral_def einterval_iff)

lemma *interval_integral_Ioi:*

$|a| < \infty \implies (LBINT\ x=a..\infty.\ f\ x) = (LBINT\ x : \{real_of_ereal\ a <..\}.\ f\ x)$
by (*auto simp: interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioo:*

$a \leq b \implies |a| < \infty \implies |b| < \infty \implies (LBINT\ x=a..b.\ f\ x) = (LBINT\ x : \{real_of_ereal\ a <..< real_of_ereal\ b\}.\ f\ x)$
by (*auto simp: interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_discrete_difference:*

fixes $f :: real \Rightarrow 'b::\{banach, second_countable_topology\}$ **and** $a\ b :: ereal$
assumes *countable* X
and *eq:* $\bigwedge x. a \leq b \implies a < x \implies x < b \implies x \notin X \implies f\ x = g\ x$
and *anti_eq:* $\bigwedge x. b \leq a \implies b < x \implies x < a \implies x \notin X \implies f\ x = g\ x$
assumes $\bigwedge x. x \in X \implies emeasure\ M\ \{x\} = 0 \bigwedge x. x \in X \implies \{x\} \in sets\ M$
shows $interval_lebesgue_integral\ M\ a\ b\ f = interval_lebesgue_integral\ M\ a\ b\ g$
unfolding *interval_lebesgue_integral_def set_lebesgue_integral_def*
apply (*intro if_cong refl arg_cong*[**where** $f=\lambda x. -x$] *interval_discrete_difference*[*of* X] *assms*)
apply (*auto simp: eq anti_eq einterval_iff split: split_indicator*)
done

lemma *interval_integral_sum:*

fixes $a\ b\ c :: ereal$
assumes *integrable:* $interval_lebesgue_integrable\ lborel\ (min\ a\ (min\ b\ c))\ (max\ a\ (max\ b\ c))\ f$
shows $(LBINT\ x=a..b.\ f\ x) + (LBINT\ x=b..c.\ f\ x) = (LBINT\ x=a..c.\ f\ x)$
proof –
let $?I = \lambda a\ b.\ LBINT\ x=a..b.\ f\ x$
{ **fix** $a\ b\ c :: ereal$ **assume** $interval_lebesgue_integrable\ lborel\ a\ c\ f\ a \leq b\ b \leq c$
then have *ord:* $a \leq b\ b \leq c\ a \leq c$ **and** *f':* $set_integrable\ lborel\ (einterval\ a\ c)\ f$
by (*auto simp: interval_lebesgue_integrable_def*)
then have *f:* $set_borel_measurable\ borel\ (einterval\ a\ c)\ f$
unfolding *set_integrable_def set_borel_measurable_def*
by (*drule_tac borel_measurable_integrable*) *simp*
have $(LBINT\ x:einterval\ a\ c.\ f\ x) = (LBINT\ x:einterval\ a\ b \cup einterval\ b\ c.\ f\ x)$
proof (*rule set_integral_cong_set*)
show $AE\ x\ in\ lborel.\ (x \in einterval\ a\ b \cup einterval\ b\ c) = (x \in einterval\ a\ c)$
using *AE_lborel_singleton*[*of* $real_of_ereal\ b$] *ord*
by (*cases a b c rule: ereal3_cases*) (*auto simp: einterval_iff*)

```

show set_borel_measurable lborel (einterval a c) f set_borel_measurable lborel
(einterval a b  $\cup$  einterval b c) f
  unfolding set_borel_measurable_def
  using ord by (auto simp: einterval_iff intro!: set_borel_measurable_subset[OF
f, unfolded set_borel_measurable_def])
qed
also have ... = (LBINT x:einterval a b. f x) + (LBINT x:einterval b c. f x)
  using ord
  by (intro set_integral_Un_AE) (auto intro!: set_integrable_subset[OF f]
simp: einterval_iff not_less)
  finally have ?I a b + ?I b c = ?I a c
    using ord by (simp add: interval_lebesgue_integral_def)
  } note 1 = this
  { fix a b c :: ereal assume interval_lebesgue_integrable lborel a c f a  $\leq$  b b  $\leq$  c
    from 1[OF this] have ?I b c + ?I a b = ?I a c
      by (metis add.commute)
    } note 2 = this
  have  $\exists$ :  $\bigwedge$  a b. b  $\leq$  a  $\implies$  (LBINT x=a..b. f x) = - (LBINT x=b..a. f x)
    by (rule interval_integral_endpoints_reverse)
  show ?thesis
    using integrable
    apply (cases a b b c a c rule: linorder_le_cases[case_product linorder_le_cases
linorder_cases])
    apply simp_all
    by (simp_all add: min_absorb1 min_absorb2 max_absorb1 max_absorb2 field_simps
1 2 3)
qed

```

```

lemma interval_integrable_isCont:
  fixes a b and f :: real  $\implies$  'a::{banach, second_countable_topology}
  shows ( $\bigwedge$ x. min a b  $\leq$  x  $\implies$  x  $\leq$  max a b  $\implies$  isCont f x)  $\implies$ 
  interval_lebesgue_integrable lborel a b f
proof (induct a b rule: linorder_wlog)
  case (le a b) then show ?case
    unfolding interval_lebesgue_integrable_def set_integrable_def
    by (auto simp: interval_lebesgue_integrable_def
  intro!: set_integrable_subset[unfolded set_integrable_def, OF borel_integrable_compact[of
{a .. b}]]
  continuous_at_imp_continuous_on)
qed (auto intro: interval_integrable_endpoints_reverse[THEN iffD1])

```

```

lemma interval_integrable_continuous_on:
  fixes a b :: real and f
  assumes a  $\leq$  b and continuous_on {a..b} f
  shows interval_lebesgue_integrable lborel a b f
using assms unfolding interval_lebesgue_integrable_def apply simp
  by (rule set_integrable_subset, rule borel_integrable_atLeastAtMost' [of a b],
auto)

```


lemma *interval_integral_eq_integral*:
fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$
shows $a \leq b \implies \text{set_integrable } \text{lborel } \{a..b\} f \implies \text{LBINT } x=a..b. f x = \text{integral } \{a..b\} f$
by (*subst interval_integral_Icc, simp*) (*rule set_borel_integral_eq_integral*)

lemma *interval_integral_eq_integral'*:
fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$
shows $a \leq b \implies \text{set_integrable } \text{lborel } (\text{einterval } a b) f \implies \text{LBINT } x=a..b. f x = \text{integral } (\text{einterval } a b) f$
by (*subst interval_lebesgue_integral_le_eq, simp*) (*rule set_borel_integral_eq_integral*)

9.10.17 General limit approximation arguments

proposition *interval_integral_Icc_approx_nonneg*:
fixes $a b :: \text{ereal}$
assumes $a < b$
fixes $u l :: \text{nat} \Rightarrow \text{real}$
assumes $\text{approx: } \text{einterval } a b = (\bigcup i. \{l i .. u i\})$
 $\text{incseq } u \text{ decseq } l \wedge i. l i < u i \wedge i. a < l i \wedge i. u i < b$
 $l \longrightarrow a \quad u \longrightarrow b$
fixes $f :: \text{real} \Rightarrow \text{real}$
assumes $f_integrable: \wedge i. \text{set_integrable } \text{lborel } \{l i .. u i\} f$
assumes $f_nonneg: AE x \text{ in } \text{lborel}. a < \text{ereal } x \longrightarrow \text{ereal } x < b \longrightarrow 0 \leq f x$
assumes $f_measurable: \text{set_borel_measurable } \text{lborel } (\text{einterval } a b) f$
assumes $\text{lbint_lim: } (\lambda i. \text{LBINT } x=l i.. u i. f x) \longrightarrow C$
shows
 $\text{set_integrable } \text{lborel } (\text{einterval } a b) f$
 $(\text{LBINT } x=a..b. f x) = C$

proof –
have 1 [*unfolded set_integrable_def*]: $\wedge i. \text{set_integrable } \text{lborel } \{l i .. u i\} f$ **by** (*rule f_integrable*)
have 2: $AE x \text{ in } \text{lborel}. \text{mono } (\lambda n. \text{indicator } \{l n .. u n\} x *_R f x)$
proof –
from f_nonneg **have** $AE x \text{ in } \text{lborel}. \forall i. l i \leq x \longrightarrow x \leq u i \longrightarrow 0 \leq f x$
by *eventually_elim*
 $(\text{metis approx}(5) \text{ approx}(6) \text{ dual_order.strict_trans1 } \text{ereal_less_eq}(3) \text{ le_less_trans})$
then show *?thesis*
apply *eventually_elim*
apply (*auto simp: mono_def split: split_indicator*)
apply (*metis approx}(3) \text{ decseqD } \text{order_trans}*)
apply (*metis approx}(2) \text{ incseqD } \text{order_trans}*)
done

qed
have 3: $AE x \text{ in } \text{lborel}. (\lambda i. \text{indicator } \{l i .. u i\} x *_R f x) \longrightarrow \text{indicator } (\text{einterval } a b) x *_R f x$
proof –
{ fix } x i **assume $l i \leq x \leq u i$**

```

then have eventually ( $\lambda i. l i \leq x \wedge x \leq u i$ ) sequentially
  apply (auto simp: eventually_sequentially intro!: exI[of _ i])
  apply (metis approx(3) decseqD order_trans)
  apply (metis approx(2) incseqD order_trans)
  done
then have eventually ( $\lambda i. f x * \text{indicator } \{l i..u i\} x = f x$ ) sequentially
  by eventually_elim auto }
then show ?thesis
unfolding approx(1) by (auto intro!: AE_I2 tendsto_eventually_split: split_indicator)
qed
have 4: ( $\lambda i. \int x. \text{indicator } \{l i..u i\} x *_R f x \partial \text{l borel}$ )  $\longrightarrow C$ 
  using lbint_lim by (simp add: interval_integral_Icc [unfolded set_lebesgue_integral_def]
approx less_imp_le)
have 5: ( $\lambda x. \text{indicat\_real } (einterval a b) x *_R f x \in \text{borel\_measurable } \text{l borel}$ )
  using f_measurable set_borel_measurable_def by blast
have (LBINT  $x=a..b. f x$ ) = lebesgue_integral l borel ( $\lambda x. \text{indicator } (einterval a b) x *_R f x$ )
  using assms by (simp add: interval_lebesgue_integral_def set_lebesgue_integral_def
less_imp_le)
also have ... = C
  by (rule integral_monotone_convergence [OF 1 2 3 4 5])
finally show (LBINT  $x=a..b. f x$ ) = C .
show set_integrable l borel (einterval a b) f
  unfolding set_integrable_def
  by (rule integrable_monotone_convergence[OF 1 2 3 4 5])
qed

proposition interval_integral_Icc_approx_integrable:
fixes u l :: nat  $\Rightarrow$  real and a b :: ereal
fixes f :: real  $\Rightarrow$  'a::{banach, second_countable_topology}
assumes a < b
assumes approx:  $einterval a b = (\bigcup i. \{l i..u i\})$ 
  incseq u decseq l  $\wedge i. l i < u i \wedge i. a < l i \wedge i. u i < b$ 
  l  $\longrightarrow a$  u  $\longrightarrow b$ 
assumes f_integrable: set_integrable l borel (einterval a b) f
shows ( $\lambda i. \text{LBINT } x=l i..u i. f x$ )  $\longrightarrow (\text{LBINT } x=a..b. f x)$ 
proof -
have ( $\lambda i. \text{LBINT } x:\{l i..u i\}. f x$ )  $\longrightarrow (\text{LBINT } x:einterval a b. f x)$ 
  unfolding set_lebesgue_integral_def
proof (rule integral_dominated_convergence)
show integrable l borel ( $\lambda x. \text{norm } (\text{indicator } (einterval a b) x *_R f x)$ )
  using f_integrable integrable_norm set_integrable_def by blast
show ( $\lambda x. \text{indicat\_real } (einterval a b) x *_R f x \in \text{borel\_measurable } \text{l borel}$ )
  using f_integrable by (simp add: set_integrable_def)
then show  $\wedge i. (\lambda x. \text{indicat\_real } \{l i..u i\} x *_R f x \in \text{borel\_measurable } \text{l borel}$ )
  by (rule set_borel_measurable_subset [unfolded set_borel_measurable_def])
(auto simp: approx)
show  $\wedge i. \text{AE } x \text{ in } \text{l borel}. \text{norm } (\text{indicator } \{l i..u i\} x *_R f x) \leq \text{norm } (\text{indicator } (einterval a b) x *_R f x)$ 

```

```

    by (intro AE_I2) (auto simp: approx_split: split_indicator)
  show AE x in lborel. (λi. indicator {l i..u i} x *R f x) ⟶ indicator (einterval
a b) x *R f x
  proof (intro AE_I2 tendsto_intros tendsto_eventually)
    fix x
    { fix i assume l i ≤ x x ≤ u i
      with ⟨incseq u⟩[THEN incseqD, of i] ⟨decseq l⟩[THEN decseqD, of i]
      have eventually (λi. l i ≤ x ∧ x ≤ u i) sequentially
        by (auto simp: eventually_sequentially decseq_def incseq_def intro: order_trans) }
    then show eventually (λx a. indicator {l x..u x} x = (indicator (einterval
a b) x :: real)) sequentially
      using approx_order_tendstoD(2)[OF ⟨l ⟶ a⟩, of x] order_tendstoD(1)[OF
⟨u ⟶ b⟩, of x]
      by (auto split: split_indicator)
    qed
  qed
  with ⟨a < b⟩ ⟨∧i. l i < u i⟩ show ?thesis
  by (simp add: interval_lebesgue_integral_le_eq[symmetric] interval_integral_Icc
less_imp_le)
qed

```

9.10.18 A slightly stronger Fundamental Theorem of Calculus

Three versions: first, for finite intervals, and then two versions for arbitrary intervals.

lemma *interval_integral_FTC_finite*:

```

  fixes f F :: real ⇒ 'a::euclidean_space and a b :: real
  assumes f: continuous_on {min a b..max a b} f
  assumes F: ∧x. min a b ≤ x ⟹ x ≤ max a b ⟹ (F has_vector_derivative (f
x)) (at x within
{min a b..max a b})
  shows (LBINT x=a..b. f x) = F b - F a
  proof (cases a ≤ b)
  case True
  have (LBINT x=a..b. f x) = (LBINT x. indicat_real {a..b} x *R f x)
    by (simp add: True interval_integral_Icc set_lebesgue_integral_def)
  also have ... = F b - F a
  proof (rule interval_FTC_atLeastAtMost [OF True])
    show continuous_on {a..b} f
      using True f by linarith
    show ∧x. [a ≤ x; x ≤ b] ⟹ (F has_vector_derivative f x) (at x within {a..b})
      by (metis F True max commute max_absorb1 min_def)
    qed
  finally show ?thesis .
  next
  case False

```

```

then have  $b \leq a$ 
  by simp
  have  $- \text{interval\_lebesgue\_integral } \text{lborel } (\text{ereal } b) (\text{ereal } a) f = - (\text{LBINT } x. \text{indicat\_real } \{b..a\} x *_{\mathbb{R}} f x)$ 
    by (simp add: <math>b \leq a</math> interval\_integral\_Icc set\_lebesgue\_integral\_def)
  also have  $\dots = F b - F a$ 
  proof (subst interval\_FTC\_atLeastAtMost [OF <math>b \leq a</math>])
    show continuous\_on  $\{b..a\} f$ 
      using False f by linarith
    show  $\bigwedge x. \llbracket b \leq x; x \leq a \rrbracket \implies (F \text{ has\_vector\_derivative } f x) \text{ (at } x \text{ within } \{b..a\})$ 
      by (metis F False max\_def min\_def)
  qed auto
finally show ?thesis
  by (metis interval\_integral\_endpoints\_reverse)
qed

```

```

lemma interval\_integral\_FTC\_nonneg:
  fixes  $f F :: \text{real} \Rightarrow \text{real}$  and  $a b :: \text{ereal}$ 
  assumes  $a < b$ 
  assumes  $F: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{DERIV } F x :> f x$ 
  assumes  $f: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f x$ 
  assumes  $f\_nonneg: \text{AE } x \text{ in } \text{lborel}. a < \text{ereal } x \longrightarrow \text{ereal } x < b \longrightarrow 0 \leq f x$ 
  assumes  $A: ((F \circ \text{real\_of\_ereal}) \longrightarrow A) \text{ (at\_right } a)$ 
  assumes  $B: ((F \circ \text{real\_of\_ereal}) \longrightarrow B) \text{ (at\_left } b)$ 
  shows
     $\text{set\_integrable } \text{lborel } (\text{einterval } a b) f$ 
     $(\text{LBINT } x=a..b. f x) = B - A$ 
proof -
  obtain  $u l$  where approx:
     $\text{einterval } a b = (\bigcup i. \{l i .. u i\})$ 
     $\text{incseq } u \text{ decseq } l \bigwedge i. l i < u i \bigwedge i. a < l i \bigwedge i. u i < b$ 
     $l \longrightarrow a \quad u \longrightarrow b$ 
  by (blast intro: einterval\_Icc\_approximation[OF <math>a < b</math>])
  have ales[simp]:  $\bigwedge x i. l i \leq x \implies a < \text{ereal } x$ 
  by (rule order\_less\_le\_trans, rule approx, force)
  have lessb[simp]:  $\bigwedge x i. x \leq u i \implies \text{ereal } x < b$ 
  by (rule order\_le\_less\_trans, subst ereal\_less\_eq(3), assumption, rule approx)
  have cf:  $\bigwedge i. \text{continuous\_on } \{\min(l i) (u i).. \max(l i) (u i)\} f$ 
  using approx f by (intro continuous\_at\_imp\_continuous\_on strip) auto
  have FTCi:  $\bigwedge i. (\text{LBINT } x=l i..u i. f x) = F (u i) - F (l i)$ 
  apply (intro interval\_integral\_FTC\_finite cf DERIV\_subset [OF F])
  by (smt (verit) F ales[simp] approx(4) has\_real\_derivative\_iff\_has\_vector\_derivative has\_vector\_derivative\_at\_within lessb)
  have  $1: \bigwedge i. \text{set\_integrable } \text{lborel } \{l i..u i\} f$ 
  by (meson ales lessb assms(3) atLeastAtMost\_iff borel\_integrable\_atLeastAtMost' continuous\_at\_imp\_continuous\_on)
  have  $2: \text{set\_borel\_measurable } \text{lborel } (\text{einterval } a b) f$ 

```

```

  unfolding set_borel_measurable_def
  by (auto simp del: real_scaleR_def intro!: borel_measurable_continuous_on_indicator
      simp: continuous_on_eq_continuous_at einterval_iff f)
  have ( $\lambda x. F (l x)$ )  $\longrightarrow A$ 
    using A approx unfolding tendsto_at_iff_sequentially_comp_def
    by (force elim!: allE[of  $\lambda i. \text{ereal } (l i)$ ])
  moreover have ( $\lambda x. F (u x)$ )  $\longrightarrow B$ 
    using B approx unfolding tendsto_at_iff_sequentially_comp_def
    by (force elim!: allE[of  $\lambda i. \text{ereal } (u i)$ ])
  ultimately have  $\exists: (\lambda i. \text{LBINT } x=l i..u i. f x) \longrightarrow B - A$ 
    by (simp add: FTCi tendsto_diff)
  show ( $\text{LBINT } x=a..b. f x$ ) =  $B - A$ 
    by (rule interval_integral_Icc_approx_nonneg [OF  $\langle a < b \rangle$  approx 1 f_nonneg
2 3])
  show set_integrable lborel (einterval a b) f
    by (rule interval_integral_Icc_approx_nonneg [OF  $\langle a < b \rangle$  approx 1 f_nonneg
2 3])
qed

theorem interval_integral_FTC_integrable:
  fixes f F :: real  $\Rightarrow$  'a::euclidean_space and a b :: ereal
  assumes a < b
  assumes F:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies (F \text{ has\_vector\_derivative } f x)$ 
  (at x)
  assumes f:  $\bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f x$ 
  assumes f_integrable: set_integrable lborel (einterval a b) f
  assumes A:  $((F \circ \text{real\_of\_ereal}) \longrightarrow A)$  (at_right a)
  assumes B:  $((F \circ \text{real\_of\_ereal}) \longrightarrow B)$  (at_left b)
  shows ( $\text{LBINT } x=a..b. f x$ ) =  $B - A$ 
proof -
  obtain u l where approx:
    einterval a b =  $(\bigcup i. \{l i .. u i\})$ 
    incseq u decseq l  $\bigwedge i. l i < u i \bigwedge i. a < l i \bigwedge i. u i < b$ 
    l  $\longrightarrow a$  u  $\longrightarrow b$ 
  by (blast intro: einterval_Icc_approximation[OF  $\langle a < b \rangle$ ])
  have [simp]:  $\bigwedge x i. l i \leq x \implies a < \text{ereal } x$ 
    by (rule order_less_le_trans, rule approx, force)
  have [simp]:  $\bigwedge x i. x \leq u i \implies \text{ereal } x < b$ 
    by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
  have FTCi:  $\bigwedge i. (\text{LBINT } x=l i..u i. f x) = F (u i) - F (l i)$ 
    using assms approx
  by (auto simp: less_imp_le min_def max_def
      intro: f_continuous_at_imp_continuous_on interval_integral_FTC_finite
      intro: has_vector_derivative_at_within)
  have ( $\lambda i. \text{LBINT } x=l i..u i. f x$ )  $\longrightarrow B - A$ 
    unfolding FTCi
  proof (intro tendsto_intros)
    show ( $\lambda x. F (l x)$ )  $\longrightarrow A$ 
      using A approx unfolding tendsto_at_iff_sequentially_comp_def

```

```

    by (elim allE[of _ λi. ereal (l i)], auto)
  show (λx. F (u x)) → B
    using B approx unfolding tendsto_at_iff_sequentially_comp_def
    by (elim allE[of _ λi. ereal (u i)], auto)
qed
moreover have (λi. LBINT x=l i..u i. f x) → (LBINT x=a..b. f x)
  by (rule interval_integral_Icc_approx_integrable [OF ⟨a < b⟩ approx_f_integrable])
ultimately show ?thesis
  by (elim LIMSEQ_unique)
qed

```

```

theorem interval_integral FTC2:
  fixes a b c :: real and f :: real ⇒ 'a::euclidean_space
  assumes a ≤ c c ≤ b
  and contf: continuous_on {a..b} f
  fixes x :: real
  assumes a ≤ x and x ≤ b
  shows ((λu. LBINT y=c..u. f y) has_vector_derivative (f x)) (at x within {a..b})
proof -
  let ?F = (λu. LBINT y=a..u. f y)
  have intf: set_integrable lborel {a..b} f
    by (rule borel_integrable_atLeastAtMost', rule contf)
  have ((λu. integral {a..u} f) has_vector_derivative f x) (at x within {a..b})
    using ⟨a ≤ x⟩ ⟨x ≤ b⟩
    by (auto intro: integral_has_vector_derivative_continuous_on_subset [OF
contf])
  then have ((λu. integral {a..u} f) has_vector_derivative (f x)) (at x within
{a..b})
    by simp
  then have (?F has_vector_derivative (f x)) (at x within {a..b})
    by (rule has_vector_derivative_weaken)
    (auto intro!: assms interval_integral_eq_integral[symmetric] set_integrable_subset
[OF intf])
  then have ((λx. (LBINT y=c..a. f y) + ?F x) has_vector_derivative (f x)) (at
x within {a..b})
    by (auto intro!: derivative_eq_intros)
  then show ?thesis
proof (rule has_vector_derivative_weaken)
  fix u assume u ∈ {a .. b}
  then show (LBINT y=c..a. f y) + (LBINT y=a..u. f y) = (LBINT y=c..u. f
y)
    using assms
    apply (intro interval_integral_sum)
    apply (auto simp: interval_lebesgue_integrable_def simp del: real_scaleR_def)
    by (rule set_integrable_subset [OF intf], auto simp: min_def max_def)
qed (insert assms, auto)
qed

```

proposition *einterval_antiderivative*:

```

fixes  $a\ b :: \text{ereal}$  and  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
assumes  $a < b$  and  $\text{contf}: \bigwedge x :: \text{real}. a < x \implies x < b \implies \text{isCont } f\ x$ 
shows  $\exists F. \forall x :: \text{real}. a < x \longrightarrow x < b \longrightarrow (F \text{ has\_vector\_derivative } f\ x)$  (at  $x$ )
proof -
  from einterval_nonempty [OF  $\langle a < b \rangle$ ] obtain  $c :: \text{real}$  where [simp]:  $a < c < b$ 
  by (auto simp: einterval_def)
  let  $?F = (\lambda u. \text{LBINT } y=c..u. f\ y)$ 
  show ?thesis
  proof (rule exI, clarsimp)
    fix  $x :: \text{real}$ 
    assume [simp]:  $a < x < b$ 
    have  $1: a < \min\ c\ x$  by simp
    from einterval_nonempty [OF 1] obtain  $d :: \text{real}$  where [simp]:  $a < d < c$ 
     $d < x$ 
    by (auto simp: einterval_def)
    have  $2: \max\ c\ x < b$  by simp
    from einterval_nonempty [OF 2] obtain  $e :: \text{real}$  where [simp]:  $c < e < x < e < b$ 
    by (auto simp: einterval_def)
    have ( $?F \text{ has\_vector\_derivative } f\ x$ ) (at  $x$  within  $\{d..e\}$ )
    proof (rule has_vector_derivative_within_subset [of _ _ _  $\{d..e\}$ ])
      have continuous_on  $\{d..e\}$   $f$ 
      proof (intro continuous_at_imp_continuous_on ballI contf; clarsimp)
        show  $\bigwedge x. \llbracket d \leq x; x \leq e \rrbracket \implies a < \text{ereal } x$ 
        using  $\langle a < \text{ereal } d \rangle$  ereal_less_ereal_Ex by auto
        show  $\bigwedge x. \llbracket d \leq x; x \leq e \rrbracket \implies \text{ereal } x < b$ 
        using  $\langle \text{ereal } e < b \rangle$  ereal_less_eq(3) le_less_trans by blast
      qed
    then show ( $?F \text{ has\_vector\_derivative } f\ x$ ) (at  $x$  within  $\{d..e\}$ )
    by (intro interval_integral FTC2) (use  $\langle d < c \rangle \langle c < e \rangle \langle d < x \rangle \langle x < e \rangle$ )
  in  $\langle \text{linarith} \rangle$ 
  qed auto
  then show ( $?F \text{ has\_vector\_derivative } f\ x$ ) (at  $x$ )
  by (force simp: has_vector_derivative_within_open [of _  $\{d..e\}$ ])
qed
qed

```

9.10.19 The substitution theorem

Once again, three versions: first, for finite intervals, and then two versions for arbitrary intervals.

theorem *interval_integral_substitution_finite*:

```

fixes  $a\ b :: \text{real}$  and  $f :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
assumes  $a \leq b$ 
and derivg:  $\bigwedge x. a \leq x \implies x \leq b \implies (g \text{ has\_real\_derivative } (g'\ x))$  (at  $x$  within  $\{a..b\}$ )

```

```

and contf : continuous_on (g ' {a..b}) f
and contg' : continuous_on {a..b} g'
shows (LBINT x=a..b. g' x *R f (g x)) = (LBINT y=g a..g b. f y)
proof -
  have v_derivg:  $\bigwedge x. a \leq x \implies x \leq b \implies (g \text{ has\_vector\_derivative } (g' x))$  (at x
  within {a..b})
    using derivg unfolding has_real_derivative_iff_has_vector_derivative .
  then have contg [simp]: continuous_on {a..b} g
    by (rule continuous_on_vector_derivative) auto
  have 1:  $\exists x \in \{a..b\}. u = g x$  if  $\min (g a) (g b) \leq u \leq \max (g a) (g b)$  for u
    by (cases g a ≤ g b) (use that assms IVT' [of g a u b] IVT2' [of g b u a] in
    ‹auto simp: min_def max_def›)
  obtain c d where g_im: g ' {a..b} = {c..d} and c ≤ d
    by (metis continuous_image_closed_interval contg ‹a ≤ b›)
  obtain F where derivF:
     $\bigwedge x. [a \leq x; x \leq b] \implies (F \text{ has\_vector\_derivative } (f (g x)))$  (at (g x) within
    (g ' {a..b}))
    using continuous_on_subset [OF contf] g_im
    by (metis antiderivative_continuous atLeastAtMost_iff image_subset_iff
    set_eq_subset)
  have contfg: continuous_on {a..b} ( $\lambda x. f (g x)$ )
    by (blast intro: continuous_on_compose2 contf contg)
  have continuous_on {a..b} ( $\lambda x. g' x *_{\mathbb{R}} f (g x)$ )
    by (auto intro!: continuous_on_scaleR contg' contfg)
  then have (LBINT x. indicat_real {a..b} x *R g' x *R f (g x)) = F (g b) - F
  (g a)
    using integral FTC_atLeastAtMost [OF ‹a ≤ b› vector_diff_chain_within [OF
    v_derivg derivF]]
    by force
  then have LBINT x=a..b. g' x *R f (g x) = F (g b) - F (g a)
    by (simp add: assms interval_integral_Icc set_lebesgue_integral_def)
  moreover have LBINT y=(g a)..(g b). f y = F (g b) - F (g a)
    proof (rule interval_integral FTC_finite)
    show continuous_on { $\min (g a) (g b).. \max (g a) (g b)$ } f
      by (rule continuous_on_subset [OF contf]) (auto simp: image_def 1)
    show (F has_vector_derivative f y) (at y within { $\min (g a) (g b).. \max (g a) (g b)$ })
    (g b))
      if y:  $\min (g a) (g b) \leq y \leq \max (g a) (g b)$  for y
    proof -
      obtain x where a ≤ x x ≤ b y = g x
        using 1 y by force
      then show ?thesis
        by (auto simp: image_def intro!: 1 has_vector_derivative_within_subset
        [OF derivF])
    qed
  qed
  ultimately show ?thesis by simp
qed

```


theorem *interval_integral_substitution_integrable*:

fixes $f :: \text{real} \Rightarrow 'a::\text{euclidean_space}$ **and** $a\ b\ u\ v :: \text{ereal}$
assumes $a < b$
and $\text{deriv_}g: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{DERIV } g\ x :> g'\ x$
and $\text{contf}: \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } f\ (g\ x)$
and $\text{contg}': \bigwedge x. a < \text{ereal } x \implies \text{ereal } x < b \implies \text{isCont } g'\ x$
and $g'\text{-nonneg}: \bigwedge x. a \leq \text{ereal } x \implies \text{ereal } x \leq b \implies 0 \leq g'\ x$
and $A: ((\text{ereal} \circ g \circ \text{real_of_ereal}) \longrightarrow A)$ (*at_right* a)
and $B: ((\text{ereal} \circ g \circ \text{real_of_ereal}) \longrightarrow B)$ (*at_left* b)
and $\text{integrable}: \text{set_integrable } \text{l borel } (\text{einterval } a\ b)$ $(\lambda x. g'\ x *_{\mathbb{R}} f\ (g\ x))$
and $\text{integrable2}: \text{set_integrable } \text{l borel } (\text{einterval } A\ B)$ $(\lambda x. f\ x)$
shows $(\text{LBINT } x=A..B. f\ x) = (\text{LBINT } x=a..b. g'\ x *_{\mathbb{R}} f\ (g\ x))$

proof –

obtain $u\ l$ **where** $\text{approx } [\text{simp}]$:
 $\text{einterval } a\ b = (\bigcup i. \{l\ i .. u\ i\})$
 $\text{incseq } u\ \text{decseq } l \bigwedge i. l\ i < u\ i \bigwedge i. a < l\ i \bigwedge i. u\ i < b$
 $l \longrightarrow a\ u \longrightarrow b$
by (*blast intro: einterval_Icc_approximation[OF <a < b>]*)
note $\text{less_imp_le } [\text{simp}]$
have $[\text{simp}]$: $\bigwedge x\ i. l\ i \leq x \implies a < \text{ereal } x$
by (*rule order_less_le_trans, rule approx, force*)
have $[\text{simp}]$: $\bigwedge x\ i. x \leq u\ i \implies \text{ereal } x < b$
by (*rule order_le_less_trans, subst eereal_less_eq(3), assumption, rule approx*)
then have $\text{lessb}[\text{simp}]$: $\bigwedge i. l\ i < b$
using $\text{approx}(4)\ \text{less_eq_real_def}$ **by** *blast*
have $[\text{simp}]$: $\bigwedge i. a < u\ i$
by (*rule order_less_trans, rule approx, auto, rule approx*)
have $\text{lle}[\text{simp}]$: $\bigwedge i\ j. i \leq j \implies l\ j \leq l\ i$ **by** (*rule decseqD, rule approx*)
have $[\text{simp}]$: $\bigwedge i\ j. i \leq j \implies u\ i \leq u\ j$ **by** (*rule incseqD, rule approx*)
have $g\text{-nondec } [\text{simp}]$: $g\ x \leq g\ y$ **if** $a < x\ x \leq y\ y < b$ **for** $x\ y$
proof (*rule DERIV_nonneg_imp_nondecreasing [OF <x ≤ y>], intro exI conjI*)
show $\bigwedge u. x \leq u \implies u \leq y \implies (g\ \text{has_real_derivative } g'\ u)$ (*at* u)
by (*meson deriv_g eereal_less_eq(3) le_less_trans less_le_trans that*)
show $\bigwedge u. x \leq u \implies u \leq y \implies 0 \leq g'\ u$
by (*meson assms(5) dual_order.trans le_ereal_le less_imp_le order_refl*)

that)

qed

have $A \leq B$ **and** $un: \text{einterval } A\ B = (\bigcup i. \{g(l\ i) <..<g(u\ i)\})$

proof –

have $A2: (\lambda i. g\ (l\ i)) \longrightarrow A$
using A **apply** (*auto simp: einterval_def tendsto_at_iff_sequentially_comp_def*)
by (*drule_tac x = λi. eereal (l i) in spec, auto*)
have $A3: \bigwedge i. g\ (l\ i) \geq A$
by (*intro decseq_ge, auto simp: decseq_def*)
have $B2: (\lambda i. g\ (u\ i)) \longrightarrow B$
using B **apply** (*auto simp: einterval_def tendsto_at_iff_sequentially_comp_def*)
by (*drule_tac x = λi. eereal (u i) in spec, auto*)

```

hence B3:  $\bigwedge i. g (u i) \leq B$ 
  by (intro incseq_le, auto simp: incseq_def)
have ereal (g (l 0))  $\leq$  ereal (g (u 0))
  by auto
then show  $A \leq B$ 
  by (meson A3 B3 order.trans)
{ fix x :: real
  assume  $A < x$  and  $x < B$ 
  then have eventually ( $\lambda i. \text{ereal} (g (l i)) < x \wedge x < \text{ereal} (g (u i))$ ) sequentially
    by (fast intro: eventually_conj order_tendstoD A2 B2)
    hence  $\exists i. g (l i) < x \wedge x < g (u i)$ 
    by (simp add: eventually_sequentially, auto)
  } note AB = this
show einterval A B =  $(\bigcup i. \{g(l i) <..< g(u i)\})$ 
proof
  show einterval A B  $\subseteq$   $(\bigcup i. \{g(l i) <..< g(u i)\})$ 
    by (auto simp: einterval_def AB)
  show  $(\bigcup i. \{g(l i) <..< g(u i)\}) \subseteq$  einterval A B
  proof (clarsimp simp add: einterval_def, intro conjI)
    show  $\bigwedge x i. \llbracket g (l i) < x; x < g (u i) \rrbracket \implies A < \text{ereal } x$ 
      using A3 le_ereal_less by blast
    show  $\bigwedge x i. \llbracket g (l i) < x; x < g (u i) \rrbracket \implies \text{ereal } x < B$ 
      using B3 ereal_le_less by blast
  qed
qed
qed
qed

have eq1:  $(\text{LBINT } x=l i.. u i. g' x *_R f (g x)) = (\text{LBINT } y=g (l i)..g (u i). f y)$  for i
  apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF deriv_g]])
  unfolding has_real_derivative_iff_has_vector_derivative[symmetric]
  apply (auto intro!: continuous_at_imp_continuous_on contf contg')
  done
have  $(\lambda i. \text{LBINT } x=l i..u i. g' x *_R f (g x)) \longrightarrow (\text{LBINT } x=a..b. g' x *_R f (g x))$ 
  using approx(4)  $\langle a < b \rangle$  integrable_interval_integral_Icc_approx_integrable by fastforce
hence 2:  $(\lambda i. (\text{LBINT } y=g (l i)..g (u i). f y)) \longrightarrow (\text{LBINT } x=a..b. g' x *_R f (g x))$ 
  by (simp add: eq1)
have incseq: incseq  $(\lambda i. \{g (l i) <..< g (u i)\})$ 
  apply (clarsimp simp: incseq_def, intro conjI)
  using lessb lle approx(5) g_nondec le_less_trans apply blast
  by (force intro: less_le_trans)
have  $(\lambda i. \text{set\_lebesgue\_integral lborel } \{g (l i) <..< g (u i)\} f)$ 
   $\longrightarrow \text{set\_lebesgue\_integral lborel } (\text{einterval } A B) f$ 
  unfolding un by (rule set_integral_cont_up) (use incseq integrable2 un in auto)

```

then have $(\lambda i. (LBINT\ y=g\ (l\ i)..g\ (u\ i). f\ y)) \longrightarrow (LBINT\ x = A..B. f\ x)$
by (*simp add: interval_lebesgue_integral_le_eq* $\langle A \leq B \rangle$)
thus *?thesis* **by** (*intro LIMSEQ_unique* [*OF* _ 2])
qed

theorem *interval_integral_substitution_nonneg*:

fixes $f\ g\ g' :: real \Rightarrow real$ **and** $a\ b\ u\ v :: ereal$
assumes $a < b$
and *deriv_g*: $\bigwedge x. a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow DERIV\ g\ x :> g'\ x$
and *contf*: $\bigwedge x. a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ f\ (g\ x)$
and *contg'*: $\bigwedge x. a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ g'\ x$
and *f_nonneg*: $\bigwedge x. a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow 0 \leq f\ (g\ x)$
and *g'_nonneg*: $\bigwedge x. a \leq ereal\ x \Longrightarrow ereal\ x \leq b \Longrightarrow 0 \leq g'\ x$
and *A*: $((ereal \circ g \circ real_of_ereal) \longrightarrow A)$ (*at_right* a)
and *B*: $((ereal \circ g \circ real_of_ereal) \longrightarrow B)$ (*at_left* b)
and *integrable_fg*: *set_integrable lborel* (*einterval* $a\ b$) $(\lambda x. f\ (g\ x) * g'\ x)$
shows
set_integrable lborel (*einterval* $A\ B$) f
 $(LBINT\ x=A..B. f\ x) = (LBINT\ x=a..b. (f\ (g\ x) * g'\ x))$

proof –

from *einterval_Icc_approximation*[*OF* $\langle a < b \rangle$] **obtain** $u\ l$ **where** *approx* [*simp*]:
einterval $a\ b = (\bigcup i. \{l\ i..u\ i\})$
incseq u
decseq l
 $\bigwedge i. l\ i < u\ i$
 $\bigwedge i. a < ereal\ (l\ i)$
 $\bigwedge i. ereal\ (u\ i) < b$
 $(\lambda x. ereal\ (l\ x)) \longrightarrow a$
 $(\lambda x. ereal\ (u\ x)) \longrightarrow b$ **by** *this auto*
have *ales[simp]*: $\bigwedge x\ i. l\ i \leq x \Longrightarrow a < ereal\ x$
by (*rule order_less_le_trans, rule approx, force*)
have *lessb[simp]*: $\bigwedge x\ i. x \leq u\ i \Longrightarrow ereal\ x < b$
by (*rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx*)
have *llb[simp]*: $\bigwedge i. l\ i < b$
using *lessb approx(4) less_eq_real_def* **by** *blast*
have *alu[simp]*: $\bigwedge i. a < u\ i$
by (*rule order_less_trans, rule approx, auto, rule approx*)
have [*simp*]: $\bigwedge i\ j. i \leq j \Longrightarrow l\ j \leq l\ i$ **by** (*rule decseqD, rule approx*)
have *uleu[simp]*: $\bigwedge i\ j. i \leq j \Longrightarrow u\ i \leq u\ j$ **by** (*rule incseqD, rule approx*)
have *g_nondec* [*simp*]: $g\ x \leq g\ y$ **if** $a < x \leq y < b$ **for** $x\ y$
proof (*rule DERIV_nonneg_imp_nondecreasing* [*OF* $\langle x \leq y \rangle$], *intro exI conjI*)
show $\bigwedge u. x \leq u \Longrightarrow u \leq y \Longrightarrow (g\ has_real_derivative\ g'\ u)$ (*at* u)
by (*meson deriv_g ereal_less_eq(3) le_less_trans less_le_trans that*)
show $\bigwedge u. x \leq u \Longrightarrow u \leq y \Longrightarrow 0 \leq g'\ u$
by (*meson g'_nonneg less_ereal.simps(1) less_trans not_less that*)
qed
have $A \leq B$ **and** *un*: *einterval* $A\ B = (\bigcup i. \{g(l\ i) < .. < g(u\ i)\})$

```

proof -
  have A2:  $(\lambda i. g (l i)) \longrightarrow A$ 
    using A by (force simp: einterval_def tendsto_at_iff_sequentially_comp_def
elim!: allE[where  $x = \lambda i. \text{ereal} (l i)$ ])
  hence A3:  $\bigwedge i. g (l i) \geq A$ 
    by (intro decseq_ge, auto simp: decseq_def)
  have B2:  $(\lambda i. g (u i)) \longrightarrow B$ 
    using B by (force simp: einterval_def tendsto_at_iff_sequentially_comp_def
elim!: allE[where  $x = \lambda i. \text{ereal} (u i)$ ])
  hence B3:  $\bigwedge i. g (u i) \leq B$ 
    by (intro incseq_le, auto simp: incseq_def)
  have  $\text{ereal} (g (l 0)) \leq \text{ereal} (g (u 0))$ 
    by (auto simp: less_imp_le)
  then show  $A \leq B$ 
    by (meson A3 B3 order.trans)
  { fix  $x :: \text{real}$ 
    assume  $A < x$  and  $x < B$ 
    then have eventually  $(\lambda i. \text{ereal} (g (l i)) < x \wedge x < \text{ereal} (g (u i)))$  sequentially
      by (fast intro: eventually_conj order_tendstoD A2 B2)
    hence  $\exists i. g (l i) < x \wedge x < g (u i)$ 
      by (simp add: eventually_sequentially, auto)
    } note  $AB = \text{this}$ 
  show einterval A B =  $(\bigcup i. \{g(l i) <.. < g(u i)\})$ 
proof
  show einterval A B  $\subseteq (\bigcup i. \{g (l i) <.. < g (u i)\})$ 
    by (auto simp: einterval_def AB)
  show  $(\bigcup i. \{g (l i) <.. < g (u i)\}) \subseteq \text{einterval A B}$ 
    using A3 B3 by (force simp: einterval_def intro: le_ereal_less_ereal_le_less)
qed
qed

  have eq1:  $(\text{LBINT } x=l i.. u i. (f (g x) * g' x)) = (\text{LBINT } y=g (l i)..g (u i). f y)$  for  $i$ 
proof -
  have  $(\text{LBINT } x=l i.. u i. g' x *_{\mathbb{R}} f (g x)) = (\text{LBINT } y=g (l i)..g (u i). f y)$ 
    apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF
deriv_g]])
  unfolding has_real_derivative_iff_has_vector_derivative[symmetric]
    apply (auto simp: less_imp_le intro!: continuous_at_imp_continuous_on
contf contg')
  done
  then show ?thesis
    by (simp add: ac_simps)
qed
have incseq: incseq  $(\lambda i. \{g (l i) <.. < g (u i)\})$ 
  apply (clarsimp simp: incseq_def, intro conjI)
  apply (meson llb antimono_def approx(3) approx(5) g_nondec le_less_trans)
  using alu uleu approx(6) g_nondec less_le_trans by blast
  have img:  $\exists c \geq l i. c \leq u i \wedge x = g c$  if  $g (l i) \leq x \leq g (u i)$  for  $x i$ 

```

```

proof –
  have continuous_on {l i..u i} g
  by (force intro!: DERIV_isCont deriv_g continuous_at_imp_continuous_on)
  with that show ?thesis
    using IVT' [of g] approx(4) dual_order.strict_implies_order by blast
qed
have continuous_on {g (l i)..g (u i)} f for i
  using contfimg by (force simp add: intro!: continuous_at_imp_continuous_on)
then have int_f:  $\bigwedge i. \text{set\_integrable\_lborel } \{g (l i) < .. < g (u i)\} f$ 
  by (rule set_integrable_subset [OF borel_integrable_atLeastAtMost']) (auto intro: less_imp_le)
have integrable: set_integrable lborel ( $\bigcup i. \{g (l i) < .. < g (u i)\}$ ) f
proof (intro pos_integrable_to_top incseq int_f)
  let ?l = (LBINT x=a..b. f (g x) * g' x)
  have ( $\lambda i. \text{LBINT } x=l i..u i. f (g x) * g' x$ )  $\longrightarrow$  ?l
  by (intro assms interval_integral_Icc_approx_integrable [OF  $\langle a < b \rangle$  approx])
hence ( $\lambda i. (\text{LBINT } y=g (l i)..g (u i). f y)$ )  $\longrightarrow$  ?l
  by (simp add: eq1)
then show ( $\lambda i. \text{set\_lebesgue\_integral lborel } \{g (l i) < .. < g (u i)\} f$ )  $\longrightarrow$  ?l
  unfolding interval_lebesgue_integral_def by (auto simp: less_imp_le)
have  $\bigwedge x i. g (l i) \leq x \implies x \leq g (u i) \implies 0 \leq f x$ 
  using ales f_nonneg img lessb by blast
then show  $\bigwedge x i. x \in \{g (l i) < .. < g (u i)\} \implies 0 \leq f x$ 
  using less_eq_real_def by auto
qed (auto simp: greaterThanLessThan_borel)
thus set_integrable lborel (einterval A B) f
  by (simp add: un)

have (LBINT x=A..B. f x) = (LBINT x=a..b. g' x *R f (g x))
proof (rule interval_integral_substitution_integrable)
  show set_integrable lborel (einterval a b) ( $\lambda x. g' x *R f (g x)$ )
  using integrable_fg by (simp add: ac_simps)
qed fact+
then show (LBINT x=A..B. f x) = (LBINT x=a..b. (f (g x) * g' x))
  by (simp add: ac_simps)
qed

```

```

syntax _complex_lebesgue_borel_integral :: pttrn  $\Rightarrow$  real  $\Rightarrow$  complex
  ((2CLBINT _. _) [0,60] 60)

```

```

translations CLBINT x. f == CONST complex_lebesgue_integral CONST lborel
  ( $\lambda x. f$ )

```

```

syntax _complex_set_lebesgue_borel_integral :: pttrn  $\Rightarrow$  real set  $\Rightarrow$  real  $\Rightarrow$  complex
  ((3CLBINT _:_. _) [0,60,61] 60)

```

```

translations

```

CLBINT $x:A. f == \text{CONST } \text{complex_set_lebesgue_integral } \text{CONST } \text{lborel } A$
 $(\lambda x. f)$

abbreviation *complex_interval_lebesgue_integral* ::
 $\text{real_measure} \Rightarrow \text{ereal} \Rightarrow \text{ereal} \Rightarrow (\text{real} \Rightarrow \text{complex}) \Rightarrow \text{complex}$ **where**
 $\text{complex_interval_lebesgue_integral } M \ a \ b \ f \equiv \text{interval_lebesgue_integral } M \ a \ b \ f$

abbreviation *complex_interval_lebesgue_integrable* ::
 $\text{real_measure} \Rightarrow \text{ereal} \Rightarrow \text{ereal} \Rightarrow (\text{real} \Rightarrow \text{complex}) \Rightarrow \text{bool}$ **where**
 $\text{complex_interval_lebesgue_integrable } M \ a \ b \ f \equiv \text{interval_lebesgue_integrable } M \ a \ b \ f$

syntax
 $_ascii_complex_interval_lebesgue_borel_integral :: \text{pttrn} \Rightarrow \text{ereal} \Rightarrow \text{ereal} \Rightarrow$
 $\text{real} \Rightarrow \text{complex}$
 $((\lambda \text{CLBINT } _ = _ \dots _) [0,60,60,61] \ 60)$

translations
 $\text{CLBINT } x=a..b. f == \text{CONST } \text{complex_interval_lebesgue_integral } \text{CONST}$
 $\text{lborel } a \ b \ (\lambda x. f)$

proposition *interval_integral_norm*:
fixes $f :: \text{real} \Rightarrow 'a :: \{\text{banach}, \text{second_countable_topology}\}$
shows $\text{interval_lebesgue_integrable } \text{lborel } a \ b \ f \implies a \leq b \implies$
 $\text{norm } (\text{LBINT } t=a..b. f \ t) \leq \text{LBINT } t=a..b. \text{norm } (f \ t)$
using $\text{integral_norm_bound}[\text{of } \text{lborel } \lambda x. \text{indicator } (e\text{interval } a \ b) \ x \ *_R \ f \ x]$
by $(\text{auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def}$
 $\text{set_lebesgue_integral_def})$

proposition *interval_integral_norm2*:
 $\text{interval_lebesgue_integrable } \text{lborel } a \ b \ f \implies$
 $\text{norm } (\text{LBINT } t=a..b. f \ t) \leq |\text{LBINT } t=a..b. \text{norm } (f \ t)|$
proof $(\text{induct } a \ b \ \text{rule: linorder_wlog})$
case $(\text{sym } a \ b)$ **then show** $?case$
by $(\text{simp add: interval_integral_endpoints_reverse}[\text{of } a \ b] \ \text{interval_integrable_endpoints_reverse}[\text{of}$
 $a \ b])$
next
case $(\text{le } a \ b)$
then have $|\text{LBINT } t=a..b. \text{norm } (f \ t)| = \text{LBINT } t=a..b. \text{norm } (f \ t)$
using $\text{integrable_norm}[\text{of } \text{lborel } \lambda x. \text{indicator } (e\text{interval } a \ b) \ x \ *_R \ f \ x]$
by $(\text{auto simp: interval_lebesgue_integral_def interval_lebesgue_integrable_def}$
 $\text{set_lebesgue_integral_def}$
 $\text{intro!: integral_nonneg_AE } \text{abs_of_nonneg})$
then show $?case$
using le **by** $(\text{simp add: interval_integral_norm})$
qed

```

lemma integral_cos:  $t \neq 0 \implies \text{LBINT } x=a..b. \cos (t * x) = \sin (t * b) / t - \sin (t * a) / t$ 
  apply (intro interval_integral_FTC_finite_continuous_intros)
  by (auto intro!: derivative_eq_intros simp: has_real_derivative_iff_has_vector_derivative[symmetric])

end

```

9.11 Integration by Substitution for the Lebesgue Integral

```

theory Lebesgue_Integral_Substitution
imports Interval_Integral
begin

```

```

lemma nn_integral_substitution_aux:
  fixes  $f :: \text{real} \Rightarrow \text{ennreal}$ 
  assumes  $Mf: f \in \text{borel\_measurable borel}$ 
  assumes  $\text{nonnegf}: \bigwedge x. f x \geq 0$ 
  assumes  $\text{derivg}: \bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x)$ 
  assumes  $\text{contg}' : \text{continuous\_on } \{a..b\} g'$ 
  assumes  $\text{derivg\_nonneg}: \bigwedge x. x \in \{a..b\} \implies g' x \geq 0$ 
  assumes  $a < b$ 
  shows  $(\int^{+x}. f x * \text{indicator } \{g a..g b\} x \partial \text{lborel}) =$ 
     $(\int^{+x}. f (g x) * g' x * \text{indicator } \{a..b\} x \partial \text{lborel})$ 
proof -
  from  $\langle a < b \rangle$  have [simp]:  $a \leq b$  by simp
  from  $\text{derivg}$  have  $\text{contg}: \text{continuous\_on } \{a..b\} g$  by (rule has_real_derivative_imp_continuous_on)
  from this and  $\text{contg}'$  have  $Mg: \text{set\_borel\_measurable borel } \{a..b\} g$  and
     $Mg': \text{set\_borel\_measurable borel } \{a..b\} g'$ 
  by (simp_all only: set_measurable_continuous_on_ivl)
  from  $\text{derivg}$  have  $\text{derivg}' : \bigwedge x. x \in \{a..b\} \implies (g \text{ has\_vector\_derivative } g' x) \text{ (at } x)$ 
  by (simp only: has_real_derivative_iff_has_vector_derivative)

  have  $\text{real\_ind}[simp]: \bigwedge A x. \text{enn2real } (\text{indicator } A x) = \text{indicator } A x$ 
  by (auto split: split_indicator)
  have  $\text{ennreal\_ind}[simp]: \bigwedge A x. \text{ennreal } (\text{indicator } A x) = \text{indicator } A x$ 
  by (auto split: split_indicator)
  have [simp]:  $\bigwedge x A. \text{indicator } A (g x) = \text{indicator } (g - ' A) x$ 
  by (auto split: split_indicator)

  from  $\text{derivg}$   $\text{derivg\_nonneg}$  have  $\text{monog}: \bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies$ 
 $g x \leq g y$ 
  by (rule deriv_nonneg_imp_mono) simp_all
  with  $\text{monog}$  have [simp]:  $g a \leq g b$  by (auto intro: mono_onD)

  show ?thesis

```

```

proof (induction rule: borel_measurable_induct[OF Mf, case_names cong set
mult add sup])
  case (cong f1 f2)
  from cong.hyps(3) have f1 = f2 by auto
  with cong show ?case by simp
next
  case (set A)
  from set.hyps show ?case
  proof (induction rule: borel_set_induct)
    case empty
    thus ?case by simp
  next
  case (interval c d)
  {
    fix u v :: real assume asm: {u..v}  $\subseteq$  {g a..g b} u  $\leq$  v

    obtain u' v' where u'v': {a..b}  $\cap$  g- '{u..v} = {u'..v'} u'  $\leq$  v' g u' = u g
    v' = v

    using asm by (rule_tac continuous_interval_vimage_Int[OF contg
monog, of u v]) simp_all
    hence {u'..v'}  $\subseteq$  {a..b} {u'..v'}  $\subseteq$  g - '{u..v} by blast+
    with u'v'(2) have u'  $\in$  g - '{u..v} v'  $\in$  g - '{u..v} by auto
    from u'v'(1) have [simp]: {a..b}  $\cap$  g - '{u..v}  $\in$  sets borel by simp

    have A: continuous_on {min u' v'..max u' v'} g'
      by (simp only: u'v' max_absorb2 min_absorb1)
      (intro continuous_on_subset[OF contg], insert u'v', auto)
    have  $\bigwedge x. x \in \{u'..v'\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x \text{ within } \{u'..v'\})$ 
      using asm by (intro has_field_derivative_subset[OF derivg] subsetD[OF
{u'..v'}  $\subseteq$  {a..b}]) auto
    hence B:  $\bigwedge x. \min u' v' \leq x \implies x \leq \max u' v' \implies$ 
      (g has_vector_derivative g' x) (at x within {min u' v'..max u'
v'})
      by (simp only: u'v' max_absorb2 min_absorb1)
      (auto simp: has_real_derivative_iff_has_vector_derivative)
    have integrable lborel ( $\lambda x. \text{indicator } (\{a..b\} \cap g - '{u..v}) x *_{\mathbb{R}} g' x$ )
      using set_integrable_subset borel_integrable_atLeastAtMost'[OF contg]
    by (metis {u'..v'}  $\subseteq$  {a..b} eucl_ivals(5) set_integrable_def sets_lborel
u'v'(1))
    hence ( $\int^+ x. \text{ennreal } (g' x) * \text{indicator } (\{a..b\} \cap g - '{u..v}) x \partial \text{lborel} =$ 
      (LBINT x:{a..b}  $\cap$  g- '{u..v}. g' x)
      unfolding set_lebesgue_integral_def
      by (subst nn_integral_eq_integral[symmetric])
      (auto intro!: derivg_nonneg nn_integral_cong split: split_indicator)
    also from interval_integral_FTC_finite[OF A B]
      have (LBINT x:{a..b}  $\cap$  g- '{u..v}. g' x) = v - u
      by (simp add: u'v' interval_integral_Icc {u  $\leq$  v})
    finally have ( $\int^+ x. \text{ennreal } (g' x) * \text{indicator } (\{a..b\} \cap g - '{u..v}) x$ 
 $\partial \text{lborel} =$ 

```



```

      ennreal (v - u) .
} note A = this

have (∫+x. indicator {c..d} (g x) * ennreal (g' x) * indicator {a..b} x ∂lborel)
=
  (∫+x. ennreal (g' x) * indicator ({a..b} ∩ g - ' {c..d}) x ∂lborel)
  by (intro nn_integral_cong) (simp split: split_indicator)
  also have {a..b} ∩ g - ' {c..d} = {a..b} ∩ g - ' {max (g a) c..min (g b) d}
  using ‹a ≤ b› ‹c ≤ d›
  by (auto intro!: monog intro: order.trans)
  also have (∫+x. ennreal (g' x) * indicator ... x ∂lborel) =
    (if max (g a) c ≤ min (g b) d then min (g b) d - max (g a) c else 0)
    using ‹c ≤ d› by (simp add: A)
  also have ... = (∫+x. indicator ({g a..g b} ∩ {c..d}) x ∂lborel)
  by (subst nn_integral_indicator) (auto intro!: measurable_sets Mg simp:)
  also have ... = (∫+x. indicator {c..d} x * indicator {g a..g b} x ∂lborel)
  by (intro nn_integral_cong) (auto split: split_indicator)
  finally show ?case ..

next

case (compl A)
note ‹A ∈ sets borel›[measurable]
from emeasure_mono[of A ∩ {g a..g b} {g a..g b} lborel]
have [simp]: emeasure lborel (A ∩ {g a..g b}) ≠ top by (auto simp: top_unique)
have [simp]: g - ' A ∩ {a..b} ∈ sets borel
  by (rule set_borel_measurable_sets[OF Mg]) auto
have [simp]: g - ' (-A) ∩ {a..b} ∈ sets borel
  by (rule set_borel_measurable_sets[OF Mg]) auto

have (∫+x. indicator (-A) x * indicator {g a..g b} x ∂lborel) =
  (∫+x. indicator (-A ∩ {g a..g b}) x ∂lborel)
  by (rule nn_integral_cong) (simp split: split_indicator)
  also from compl have ... = emeasure lborel ({g a..g b} - A) using de-
rivg_nonneg
  by (simp add: vimage_Cmpl diff_eq Int_commute[of -A])
  also have {g a..g b} - A = {g a..g b} - A ∩ {g a..g b} by blast
  also have emeasure lborel ... = g b - g a - emeasure lborel (A ∩ {g a..g b})
  using ‹A ∈ sets borel› by (subst emeasure_Diff) auto
  also have emeasure lborel (A ∩ {g a..g b}) =
    ∫+x. indicator A x * indicator {g a..g b} x ∂lborel
  using ‹A ∈ sets borel›
  by (subst nn_integral_indicator[symmetric], simp, intro nn_integral_cong)
  (simp split: split_indicator)
  also have ... = ∫+x. indicator (g - ' A ∩ {a..b}) x * ennreal (g' x * indicator
{a..b} x) ∂lborel (is _ = ?I)
  by (subst compl.IH, intro nn_integral_cong) (simp split: split_indicator)
  also have g b - g a = (LBINT x:{a..b}. g' x) using derivg'
  unfolding set_lebesgue_integral_def

```

```

    by (intro integral_FTC_atLeastAtMost[symmetric])
      (auto intro: continuous_on_subset[OF contg] has_field_derivative_subset[OF
derivg]
        has_vector_derivative_at_within)
    also have ennreal ... = (∫+ x. g' x * indicator {a..b} x ∂lborel)
    using borel_integrable_atLeastAtMost'[OF contg] unfolding set_lebesgue_integral_def
      by (subst nn_integral_eq_integral)
      (simp_all add: mult.commute derivg_nonneg set_integrable_def split:
split_indicator)
    also have Mg'': (λx. indicator (g - ' A ∩ {a..b}) x * ennreal (g' x * indicator
{a..b} x))
      ∈ borel_measurable borel using Mg'
    by (intro borel_measurable_times_ennreal borel_measurable_indicator)
      (simp_all add: mult.commute set_borel_measurable_def)
    have le: (∫+ x. indicator (g - ' A ∩ {a..b}) x * ennreal (g' x * indicator {a..b}
x) ∂lborel) ≤
      (∫+ x. ennreal (g' x) * indicator {a..b} x ∂lborel)
    by (intro nn_integral_mono) (simp split: split_indicator add: derivg_nonneg)
    note integrable = borel_integrable_atLeastAtMost'[OF contg]
    with le have notinf: (∫+ x. indicator (g - ' A ∩ {a..b}) x * ennreal (g' x *
indicator {a..b} x) ∂lborel) ≠ top
    by (auto simp: real_integrable_def nn_integral_set_ennreal mult.commute
top_unique set_integrable_def)
    have (∫+ x. g' x * indicator {a..b} x ∂lborel) - ?I =
      ∫+ x. ennreal (g' x * indicator {a..b} x) -
      indicator (g - ' A ∩ {a..b}) x * ennreal (g' x * indicator {a..b}
x) ∂lborel
    apply (intro nn_integral_diff[symmetric])
    apply (insert Mg', simp add: mult.commute set_borel_measurable_def) []
    apply (insert Mg'', simp) []
    apply (simp split: split_indicator add: derivg_nonneg)
    apply (rule notinf)
    apply (simp split: split_indicator add: derivg_nonneg)
    done
    also have ... = ∫+ x. indicator (-A) (g x) * ennreal (g' x) * indicator {a..b}
x ∂lborel
      by (intro nn_integral_cong) (simp split: split_indicator)
    finally show ?case .

next
case (union f)
then have [simp]: ∧i. {a..b} ∩ g - ' f i ∈ sets borel
  by (subst Int_commute, intro set_borel_measurable_sets[OF Mg]) auto
have g - ' (∪ i. f i) ∩ {a..b} = (∪ i. {a..b} ∩ g - ' f i) by auto
hence g - ' (∪ i. f i) ∩ {a..b} ∈ sets borel by (auto simp del: UN_simps)

have (∫+ x. indicator (∪ i. f i) x * indicator {g a..g b} x ∂lborel) =
  ∫+ x. indicator (∪ i. {g a..g b} ∩ f i) x ∂lborel
  by (intro nn_integral_cong) (simp split: split_indicator)

```

```

also from union have ... = emeasure lborel ( $\bigcup i. \{g \ a..g \ b\} \cap f \ i$ ) by simp
also from union have ... = ( $\sum i. \text{emeasure lborel } (\{g \ a..g \ b\} \cap f \ i)$ )
by (intro suminf_emeasure[symmetric]) (auto simp: disjoint_family_on_def)
also from union have ... = ( $\sum i. \int^+ x. \text{indicator } (\{g \ a..g \ b\} \cap f \ i) \ x \ \partial \text{lborel}$ )
by simp
also have ( $\lambda i. \int^+ x. \text{indicator } (\{g \ a..g \ b\} \cap f \ i) \ x \ \partial \text{lborel}$ ) =
  ( $\lambda i. \int^+ x. \text{indicator } (f \ i) \ x * \text{indicator } \{g \ a..g \ b\} \ x \ \partial \text{lborel}$ )
  by (intro ext nn_integral_cong) (simp split: split_indicator)
also from union.IH have ( $\sum i. \int^+ x. \text{indicator } (f \ i) \ x * \text{indicator } \{g \ a..g \ b\}$ 
 $x \ \partial \text{lborel}$ ) =
  ( $\sum i. \int^+ x. \text{indicator } (f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\} \ x$ 
 $\partial \text{lborel}$ ) by simp
also have ( $\lambda i. \int^+ x. \text{indicator } (f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\}$ 
 $x \ \partial \text{lborel}$ ) =
  ( $\lambda i. \int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator}$ 
 $(\{a..b\} \cap g \ -' \ f \ i) \ x \ \partial \text{lborel}$ )
  by (intro ext nn_integral_cong) (simp split: split_indicator)
also have ( $\sum i. \dots \ i$ ) =  $\int^+ x. (\sum i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (\{a..b\} \cap g \ -' \ f \ i) \ x) \ \partial \text{lborel}$ 
  using Mg'
  apply (intro nn_integral_suminf[symmetric])
  apply (rule borel_measurable_times_ennreal, simp add: mult commute set_borel_measurable_def)
  apply (rule borel_measurable_indicator, subst sets_lborel)
  apply (simp_all split: split_indicator add: derivg_nonneg)
  done
also have ( $\lambda x \ i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (\{a..b\} \cap g \ -' \ f \ i) \ x$ ) =
  ( $\lambda x \ i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (g \ -' \ f \ i) \ x$ )
  by (intro ext) (simp split: split_indicator)
also have ( $\int^+ x. (\sum i. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \text{indicator } (g \ -' \ f \ i) \ x) \ \partial \text{lborel}$ ) =
   $\int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * (\sum i. \text{indicator } (g \ -' \ f \ i) \ x) \ \partial \text{lborel}$ 
  by (intro nn_integral_cong) (auto split: split_indicator simp: derivg_nonneg)
also from union have ( $\lambda x. \sum i. \text{indicator } (g \ -' \ f \ i) \ x :: \text{ennreal}$ ) = ( $\lambda x. \text{indicator } (\bigcup i. g \ -' \ f \ i) \ x$ )
  by (intro ext suminf_indicator) (auto simp: disjoint_family_on_def)
also have ( $\int^+ x. \text{ennreal } (g' \ x * \text{indicator } \{a..b\} \ x) * \dots \ x \ \partial \text{lborel}$ ) =
  ( $\int^+ x. \text{indicator } (\bigcup i. f \ i) \ (g \ x) * \text{ennreal } (g' \ x) * \text{indicator } \{a..b\}$ 
 $x \ \partial \text{lborel}$ )
  by (intro nn_integral_cong) (simp split: split_indicator)
finally show ?case .
qed

next
case (mult f c)
note Mf[measurable] =  $\langle f \in \text{borel\_measurable borel}$ 
let ?I = indicator {a..b}

```

```

have ( $\lambda x. f (g x * ?I x) * \text{ennreal} (g' x * ?I x) \in \text{borel\_measurable borel}$ ) using
Mg Mg'
  by (intro borel_measurable_times_ennreal measurable_compose[OF _ Mf])
      (simp_all add: mult.commute set_borel_measurable_def)
  also have ( $\lambda x. f (g x * ?I x) * \text{ennreal} (g' x * ?I x) = (\lambda x. f (g x) * \text{ennreal} (g' x) * ?I x)$ )
  by (intro ext) (simp split: split_indicator)
  finally have  $Mf' : (\lambda x. f (g x) * \text{ennreal} (g' x) * ?I x) \in \text{borel\_measurable borel}$ 
.

with mult show ?case
  by (subst (1 2 3) mult_ac, subst (1 2) nn_integral_cmult) (simp_all add:
mult_ac)

next
case (add f2 f1)
  let  $?I = \text{indicator } \{a..b\}$ 
  {
    fix  $f :: \text{real} \Rightarrow \text{ennreal}$  assume  $Mf : f \in \text{borel\_measurable borel}$ 
    have ( $\lambda x. f (g x * ?I x) * \text{ennreal} (g' x * ?I x) \in \text{borel\_measurable borel}$ )
using Mg Mg'
      by (intro borel_measurable_times_ennreal measurable_compose[OF _ Mf])
          (simp_all add: mult.commute set_borel_measurable_def)
      also have ( $\lambda x. f (g x * ?I x) * \text{ennreal} (g' x * ?I x) = (\lambda x. f (g x) * \text{ennreal} (g' x) * ?I x)$ )
      by (intro ext) (simp split: split_indicator)
      finally have ( $\lambda x. f (g x) * \text{ennreal} (g' x) * ?I x \in \text{borel\_measurable borel}$ ) .
  } note  $Mf' = \text{this}[OF \langle f1 \in \text{borel\_measurable borel} \rangle] \text{this}[OF \langle f2 \in \text{borel\_measurable borel} \rangle]$ 

  have ( $\int^+ x. (f1 x + f2 x) * \text{indicator } \{g a..g b\} x \partial \text{lborel} =$ 
 $\int^+ x. f1 x * \text{indicator } \{g a..g b\} x + f2 x * \text{indicator } \{g a..g b\} x$ 
 $\partial \text{lborel}$ )
    by (intro nn_integral_cong) (simp split: split_indicator)
  also from add have  $\dots = \int^+ x. f1 (g x) * \text{ennreal} (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel} +$ 
 $\int^+ x. f2 (g x) * \text{ennreal} (g' x) * \text{indicator } \{a..b\} x$ 
 $\partial \text{lborel}$ )
    by (simp_all add: nn_integral_add)
  also from add have  $\dots = \int^+ x. f1 (g x) * \text{ennreal} (g' x) * \text{indicator } \{a..b\} x +$ 
 $\int^+ x. f2 (g x) * \text{ennreal} (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ )
    by (intro nn_integral_add[symmetric])
      (auto simp add: Mf' derivg_nonneg split: split_indicator)
  also have  $\dots = \int^+ x. (f1 (g x) + f2 (g x)) * \text{ennreal} (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ 
  by (intro nn_integral_cong) (simp split: split_indicator add: distrib_right)
  finally show ?case .

```

next

```

case (sup F)
{
  fix i
  let ?I = indicator {a..b}
  have (λx. F i (g x * ?I x) * ennreal (g' x * ?I x)) ∈ borel_measurable borel
using Mg Mg'
  by (rule_tac borel_measurable_times_ennreal, rule_tac measurable_compose[OF
_ sup.hyps(1)])
  (simp_all add: mult.commute set_borel_measurable_def)
  also have (λx. F i (g x * ?I x) * ennreal (g' x * ?I x)) = (λx. F i (g x) *
ennreal (g' x) * ?I x)
  by (intro ext) (simp split: split_indicator)
  finally have ... ∈ borel_measurable borel .
} note Mf' = this

have (∫+x. (SUP i. F i x) * indicator {g a..g b} x ∂lborel) =
  ∫+x. (SUP i. F i x * indicator {g a..g b} x) ∂lborel
  by (intro nn_integral_cong) (simp split: split_indicator)
also from sup have ... = (SUP i. ∫+x. F i x * indicator {g a..g b} x ∂lborel)
  by (intro nn_integral_monotone_convergence_SUP)
  (auto simp: incseq_def le_fun_def split: split_indicator)
also from sup have ... = (SUP i. ∫+x. F i (g x) * ennreal (g' x) * indicator
{a..b} x ∂lborel)
  by simp
also from sup have ... = ∫+x. (SUP i. F i (g x) * ennreal (g' x) * indicator
{a..b} x) ∂lborel
  by (intro nn_integral_monotone_convergence_SUP[symmetric])
  (auto simp: incseq_def le_fun_def derivg_nonneg Mf' split: split_indicator
intro!: mult_right_mono)
also from sup have ... = ∫+x. (SUP i. F i (g x)) * ennreal (g' x) * indicator
{a..b} x ∂lborel
  by (subst mult_assoc, subst mult_commute, subst SUP_mult_left_ennreal)
  (auto split: split_indicator simp: derivg_nonneg mult_ac)
  finally show ?case by (simp add: image_comp)
qed
qed

```

theorem *nn_integral_substitution:*

```

fixes f :: real ⇒ real
assumes Mf[measurable]: set_borel_measurable borel {g a..g b} f
assumes derivg: ∧x. x ∈ {a..b} ⇒ (g has_real_derivative g' x) (at x)
assumes contg': continuous_on {a..b} g'
assumes derivg_nonneg: ∧x. x ∈ {a..b} ⇒ g' x ≥ 0
assumes a ≤ b
shows (∫+x. f x * indicator {g a..g b} x ∂lborel) =
  (∫+x. f (g x) * g' x * indicator {a..b} x ∂lborel)
proof (cases a = b)
assume a ≠ b
with ⟨a ≤ b⟩ have a < b by auto

```

```

let ?f' =  $\lambda x. f x * \text{indicator } \{g a..g b\} x$ 

from derivg derivg_nonneg have monog:  $\bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies$ 
 $g x \leq g y$ 
  by (rule deriv_nonneg_imp_mono) simp_all
have bounds:  $\bigwedge x. x \geq a \implies x \leq b \implies g x \geq g a \bigwedge x. x \geq a \implies x \leq b \implies g x$ 
 $\leq g b$ 
  by (auto intro: monog)

from derivg_nonneg have nonneg:
   $\bigwedge f x. x \geq a \implies x \leq b \implies g' x \neq 0 \implies f x * \text{ennreal } (g' x) \geq 0 \implies f x \geq 0$ 
  by (force simp: field_simps)
have nonneg':  $\bigwedge x. a \leq x \implies x \leq b \implies \neg 0 \leq f (g x) \implies 0 \leq f (g x) * g' x$ 
 $\implies g' x = 0$ 
  by (metis atLeastAtMost_iff derivg_nonneg eq_iff mult_eq_0_iff mult_le_0_iff)

have ( $\int^+ x. f x * \text{indicator } \{g a..g b\} x \partial \text{lborel}$ ) =
  ( $\int^+ x. \text{ennreal } (?f' x) * \text{indicator } \{g a..g b\} x \partial \text{lborel}$ )
  by (intro nn_integral_cong)
  (auto split: split_indicator split_max simp: zero_ennreal.rep_eq ennreal_neg)
also have ... =  $\int^+ x. ?f' (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ 
using Mf
  by (subst nn_integral_substitution_aux[OF _ _ derivg contg' derivg_nonneg
  <a < b>])
  (auto simp add: mult.commute set_borel_measurable_def)
also have ... =  $\int^+ x. f (g x) * \text{ennreal } (g' x) * \text{indicator } \{a..b\} x \partial \text{lborel}$ 
  by (intro nn_integral_cong) (auto split: split_indicator simp: max_def dest:
  bounds)
also have ... =  $\int^+ x. \text{ennreal } (f (g x) * g' x * \text{indicator } \{a..b\} x) \partial \text{lborel}$ 
  by (intro nn_integral_cong) (auto simp: mult.commute derivg_nonneg en-
  nreal_mult' split: split_indicator)
finally show ?thesis .
qed auto

```

theorem integral_substitution:

```

assumes integrable: set_integrable lborel {g a..g b} f
assumes derivg:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x)$ 
assumes contg': continuous_on {a..b} g'
assumes derivg_nonneg:  $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$ 
assumes a ≤ b
shows set_integrable lborel {a..b} ( $\lambda x. f (g x) * g' x$ )
  and (LBINT x. f x * indicator {g a..g b} x) = (LBINT x. f (g x) * g' x *
  indicator {a..b} x)

```

proof –

```

from derivg have contg: continuous_on {a..b} g by (rule has_real_derivative_imp_continuous_on)
with contg' have Mg: set_borel_measurable borel {a..b} g
  and Mg': set_borel_measurable borel {a..b} g'
  by (simp_all only: set_measurable_continuous_on_ivl)
from derivg derivg_nonneg have monog:  $\bigwedge x y. a \leq x \implies x \leq y \implies y \leq b \implies$ 

```

```

g x ≤ g y
  by (rule deriv_nonneg_imp_mono) simp_all

have (λx. ennreal (f x) * indicator {g a..g b} x) =
  (λx. ennreal (f x * indicator {g a..g b} x))
  by (intro ext) (simp split: split_indicator)
with integrable have M1: (λx. f x * indicator {g a..g b} x) ∈ borel_measurable
borel
  by (force simp: mult.commute set_integrable_def)
from integrable have M2: (λx. -f x * indicator {g a..g b} x) ∈ borel_measurable
borel
  by (force simp: mult.commute set_integrable_def)

have (LBINT x. (f x :: real) * indicator {g a..g b} x) =
  enn2real (∫+ x. ennreal (f x) * indicator {g a..g b} x ∂lborel) -
  enn2real (∫+ x. ennreal (- (f x)) * indicator {g a..g b} x ∂lborel) using
integrable
  unfolding set_integrable_def
  by (subst real_lebesgue_integral_def) (simp_all add: nn_integral_set_ennreal
mult.commute)
also have *: (∫+ x. ennreal (f x) * indicator {g a..g b} x ∂lborel) =
  (∫+ x. ennreal (f x * indicator {g a..g b} x) ∂lborel)
  by (intro nn_integral_cong) (simp split: split_indicator)
also from M1 * have A: (∫+ x. ennreal (f x * indicator {g a..g b} x) ∂lborel)
=
  (∫+ x. ennreal (f (g x) * g' x * indicator {a..b} x) ∂lborel)
  by (subst nn_integral_substitution[OF__deriv contg' derivg_nonneg ⟨a ≤ b⟩])
  (auto simp: nn_integral_set_ennreal mult.commute set_borel_measurable_def)
also have **: (∫+ x. ennreal (- (f x)) * indicator {g a..g b} x ∂lborel) =
  (∫+ x. ennreal (- (f x) * indicator {g a..g b} x) ∂lborel)
  by (intro nn_integral_cong) (simp split: split_indicator)
also from M2 ** have B: (∫+ x. ennreal (- (f x) * indicator {g a..g b} x)
∂lborel) =
  (∫+ x. ennreal (- (f (g x)) * g' x * indicator {a..b} x) ∂lborel)
  by (subst nn_integral_substitution[OF__deriv contg' derivg_nonneg ⟨a ≤ b⟩])
  (auto simp: nn_integral_set_ennreal mult.commute set_borel_measurable_def)

also {
  from integrable have Mf: set_borel_measurable borel {g a..g b} f
  unfolding set_borel_measurable_def set_integrable_def by simp
  from measurable_compose Mg Mf Mg' borel_measurable_times
  have (λx. f (g x * indicator {a..b} x) * indicator {g a..g b} (g x * indicator
{a..b} x) *
    (g' x * indicator {a..b} x)) ∈ borel_measurable borel (is ?f ∈ _)
  by (simp add: mult.commute set_borel_measurable_def)
  also have ?f = (λx. f (g x) * g' x * indicator {a..b} x)
  using monog by (intro ext) (auto split: split_indicator)
  finally show set_integrable lborel {a..b} (λx. f (g x) * g' x)
  using A B integrable unfolding real_integrable_def set_integrable_def

```

```

    by (simp_all add: nn_integral_set_ennreal_mult_commute)
  } note integrable' = this

  have enn2real (∫+ x. ennreal (f (g x) * g' x * indicator {a..b} x) ∂lborel) -
    enn2real (∫+ x. ennreal (-f (g x) * g' x * indicator {a..b} x)
∂lborel) =
    (LBINT x. f (g x) * g' x * indicator {a..b} x)
  using integrable' unfolding set_integrable_def
  by (subst real_lebesgue_integral_def) (simp_all add: field_simps)
  finally show (LBINT x. f x * indicator {g a..g b} x) =
    (LBINT x. f (g x) * g' x * indicator {a..b} x) .
qed

```

```

theorem interval_integral_substitution:
  assumes integrable: set_integrable lborel {g a..g b} f
  assumes derivg:  $\bigwedge x. x \in \{a..b\} \implies (g \text{ has\_real\_derivative } g' x) \text{ (at } x)$ 
  assumes contg': continuous_on {a..b} g'
  assumes derivg_nonneg:  $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$ 
  assumes a ≤ b
  shows set_integrable lborel {a..b} (λx. f (g x) * g' x)
    and (LBINT x=g a..g b. f x) = (LBINT x=a..b. f (g x) * g' x)
  apply (rule interval_substitution[OF assms], simp, simp)
  apply (subst (1 2) interval_integral_Icc, fact)
  apply (rule deriv_nonneg_imp_mono[OF derivg derivg_nonneg], simp, simp,
fact)
  using integral_substitution(2)[OF assms]
  apply (simp add: mult_commute set_lebesgue_integral_def)
  done

```

```

lemma set_borel_integrable_singleton[simp]: set_integrable lborel {x} (f :: real ⇒
real)
  unfolding set_integrable_def
  by (subst integrable_discrete_difference[where X={x} and g=λ_. 0]) auto
end

```

9.12 The Volume of an n -Dimensional Ball

```

theory Ball_Volume
  imports Gamma_Function Lebesgue_Integral_Substitution
begin

```

We define the volume of the unit ball in terms of the Gamma function. Note that the dimension need not be an integer; we also allow fractional dimensions, although we do not use this case or prove anything about it for now.

```

definition unit_ball_vol :: real ⇒ real where
  unit_ball_vol n = pi powr (n / 2) / Gamma (n / 2 + 1)

```


lemma *unit_ball_vol_pos* [simp]: $n \geq 0 \implies \text{unit_ball_vol } n > 0$
by (force simp: unit_ball_vol_def intro: divide_nonneg_pos)

lemma *unit_ball_vol_nonneg* [simp]: $n \geq 0 \implies \text{unit_ball_vol } n \geq 0$
by (simp add: dual_order.strict_implies_order)

We first need the value of the following integral, which is at the core of computing the measure of an $n + 1$ -dimensional ball in terms of the measure of an n -dimensional one.

lemma *emeasure_cball_aux_integral*:

$$\left(\int^+ x. \text{indicator } \{0..1\} x * \text{sqrt } (1 - x^2) \wedge n \text{ } \partial \text{lborel} \right) = \text{ennreal } (\text{Beta } (1 / 2) (\text{real } n / 2 + 1))$$

proof –

have $((\lambda t. t \text{ powr } (-1 / 2) * (1 - t) \text{ powr } (\text{real } n / 2)) \text{ has_integral } \text{Beta } (1 / 2) (\text{real } n / 2 + 1)) \{0..1\}$
using *has_integral_Beta_real*[of $1/2$ $n/2 + 1$] **by** *simp*
from *nn_integral_has_integral_lebesgue*[OF *this*] **have**
 $\text{ennreal } (\text{Beta } (1 / 2) (\text{real } n / 2 + 1)) =$
 $\text{nn_integral lborel } (\lambda t. \text{ennreal } (t \text{ powr } (-1 / 2) * (1 - t) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0^2..1^2\} t))$
by (*simp add: mult_ac ennreal_mult' ennreal_indicator*)
also have $\dots = \left(\int^+ x. \text{ennreal } (x^2 \text{ powr } - (1 / 2) * (1 - x^2) \text{ powr } (\text{real } n / 2) * (2 * x) * \text{indicator } \{0..1\} x) \partial \text{lborel} \right)$
by (*subst nn_integral_substitution*[**where** $g = \lambda x. x^2$ **and** $g' = \lambda x. 2 * x$])
(auto intro!: derivative_eq_intros continuous_intros simp: set_borel_measurable_def)
also have $\dots = \left(\int^+ x. 2 * \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel} \right)$
by (*intro nn_integral_cong_AE AE_I*[of $_ _ \{0\}$])
(auto simp: indicator_def powr_minus powr_half_sqrt field_split_simps ennreal_mult')
also have $\dots = \left(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel} \right) +$
 $\left(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{0..1\} x) \partial \text{lborel} \right)$
(is $_ = ?I + _$) **by** (*simp add: mult_2 nn_integral_add*)
also have $?I = \left(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{-1..0\} x) \partial \text{lborel} \right)$
by (*subst nn_integral_real_affine*[of $_ -1 0$])
(auto simp: indicator_def intro!: nn_integral_cong)
hence $?I + ?I = \dots + ?I$ **by** *simp*
also have $\dots = \left(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * (\text{indicator } \{-1..0\} x + \text{indicator } \{0..1\} x)) \partial \text{lborel} \right)$
by (*subst nn_integral_add* [*symmetric*]) *(auto simp: algebra_simps)*
also have $\dots = \left(\int^+ x. \text{ennreal } ((1 - x^2) \text{ powr } (\text{real } n / 2) * \text{indicator } \{-1..1\} x) \partial \text{lborel} \right)$
by (*intro nn_integral_cong_AE AE_I*[of $_ _ \{0\}$]) *(auto simp: indicator_def)*

also have $\dots = (\int^+ x. \text{ennreal } (\text{indicator } \{-1..1\} x * \text{sqrt } (1 - x^2) ^ n)$
 $\partial \text{lborel})$
by (*intro nn_integral_cong_AE AE_I[of _ _ {1, -1}]*)
(auto simp: powr_half_sqrt [symmetric] indicator_def abs_square_le_1
abs_square_eq_1 powr_def exp_of_nat_mult [symmetric] emeasure_lborel_countable)
finally show *?thesis ..*
qed

lemma *real_sqrt_le_iff'*: $x \geq 0 \implies y \geq 0 \implies \text{sqrt } x \leq y \longleftrightarrow x \leq y ^ 2$
using *real_le_sqrt sqrt_le_D* **by** *blast*

Isabelle's type system makes it very difficult to do an induction over the dimension of a Euclidean space type, because the type would change in the inductive step. To avoid this problem, we instead formulate the problem in a more concrete way by unfolding the definition of the Euclidean norm.

lemma *emeasure_cball_aux*:
assumes *finite A r > 0*
shows $\text{emeasure } (Pi_M A (\lambda_. \text{lborel}))$
 $(\{f. \text{sqrt } (\sum_{i \in A}. (f i)^2) \leq r\} \cap \text{space } (Pi_M A (\lambda_. \text{lborel}))) =$
 $\text{ennreal } (\text{unit_ball_vol } (\text{real } (\text{card } A)) * r ^ \text{card } A)$
using *assms*
proof (*induction arbitrary: r*)
case (*empty r*)
thus *?case*
by (*simp add: unit_ball_vol_def space_PiM*)
next
case (*insert i A r*)
interpret *product_sigma_finite* $\lambda_. \text{lborel}$
by *standard*
have $\text{emeasure } (Pi_M (\text{insert } i A) (\lambda_. \text{lborel}))$
 $(\{f. \text{sqrt } (\sum_{i \in \text{insert } i A}. (f i)^2) \leq r\} \cap \text{space } (Pi_M (\text{insert } i A) (\lambda_.$
 $\text{lborel}))) =$
 $\text{nn_integral } (Pi_M (\text{insert } i A) (\lambda_. \text{lborel}))$
 $(\text{indicator } (\{f. \text{sqrt } (\sum_{i \in \text{insert } i A}. (f i)^2) \leq r\} \cap$
 $\text{space } (Pi_M (\text{insert } i A) (\lambda_. \text{lborel}))))$
by (*subst nn_integral_indicator*) *auto*
also have $\dots = (\int^+ y. \int^+ x. \text{indicator } (\{f. \text{sqrt } ((f i)^2 + (\sum_{i \in A}. (f i)^2)) \leq$
 $r\} \cap$
 $\text{space } (Pi_M (\text{insert } i A) (\lambda_. \text{lborel}))) (x(i := y))$
 $\partial Pi_M A (\lambda_. \text{lborel}) \partial \text{lborel})$
using *insert.premis insert.hyps* **by** (*subst product_nn_integral_insert_rev*) *auto*
also have $\dots = (\int^+ (y::\text{real}). \int^+ x. \text{indicator } \{-r..r\} y * \text{indicator } (\{f. \text{sqrt}$
 $((\sum_{i \in A}. (f i)^2) \leq$
 $\text{sqrt } (r ^ 2 - y ^ 2)\} \cap \text{space } (Pi_M A (\lambda_. \text{lborel}))) x \partial Pi_M A (\lambda_.$
 $\text{lborel}) \partial \text{lborel})$
proof (*intro nn_integral_cong, goal_cases*)
case (*1 y f*)
have $*$: $y \in \{-r..r\}$ **if** $y ^ 2 + c \leq r ^ 2$ $c \geq 0$ **for** c
proof -

```

    have  $y^2 \leq y^2 + c$  using that by simp
    also have  $\dots \leq r^2$  by fact
    finally show ?thesis
      using  $\langle r > 0 \rangle$  by (simp add: power2_le_iff_abs_le abs_if_split: if_splits)
  qed
  have  $(\sum_{x \in A}. (if\ x = i\ then\ y\ else\ f\ x)^2) = (\sum_{x \in A}. (f\ x)^2)$ 
    using insert.hyps by (intro sum.cong) auto
  thus ?case using 1  $\langle r > 0 \rangle$ 
  by (auto simp: sum_nonneg real_sqrt_le_iff' indicator_def PiE_def space_PiM
dest!: *)
  qed
  also have  $\dots = (\int^+ (y::real). indicator\ \{-r..r\}\ y * (\int^+ x. indicator\ (\{f.\ sqrt\ ((\sum_{i \in A}. (f\ i)^2)) \leq sqrt\ (r^2 - y^2)\} \cap space\ (Pi_M\ A\ (\lambda_.\ lborel)))\ x \partial Pi_M\ A\ (\lambda_.\ lborel))\ \partial lborel)$ 
    by (subst nn_integral_cmult) auto
  also have  $\dots = (\int^+ (y::real). indicator\ \{-r..r\}\ y * emeasure\ (Pi_M\ A\ (\lambda_.\ lborel))\ (\{f.\ sqrt\ ((\sum_{i \in A}. (f\ i)^2)) \leq sqrt\ (r^2 - y^2)\} \cap space\ (Pi_M\ A\ (\lambda_.\ lborel)))\ \partial lborel)$ 
    using  $\langle finite\ A \rangle$  by (intro nn_integral_cong, subst nn_integral_indicator) auto
  also have  $\dots = (\int^+ (y::real). indicator\ \{-r..r\}\ y * ennreal\ (unit\_ball\_vol\ (real\ (card\ A))) * (sqrt\ (r^2 - y^2))^{\text{card } A}\ \partial lborel)$ 
  proof (intro nn_integral_cong_AE, goal_cases)
  case 1
  have  $AE\ y\ in\ lborel.\ y \notin \{-r, r\}$ 
  by (intro AE_not_in countable_imp_null_set lborel) auto
  thus ?case
  proof eventually_elim
  case (elim y)
  show ?case
  proof (cases  $y \in \{-r <..< r\}$ )
  case True
  hence  $y^2 < r^2$  by (subst real_sqrt_less_iff [symmetric]) auto
  thus ?thesis by (subst insert.IH) (auto)
  qed
  qed
  qed
  also have  $\dots = ennreal\ (unit\_ball\_vol\ (real\ (card\ A))) * (\int^+ (y::real). indicator\ \{-r..r\}\ y * (sqrt\ (r^2 - y^2))^{\text{card } A}\ \partial lborel)$ 
  by (subst nn_integral_cmult [symmetric])
  (auto simp: mult_ac ennreal_mult' [symmetric] indicator_def intro!: nn_integral_cong)
  also have  $(\int^+ (y::real). indicator\ \{-r..r\}\ y * (sqrt\ (r^2 - y^2))^{\text{card } A}\ \partial lborel) = (\int^+ (y::real). r^{\text{card } A} * indicator\ \{-1..1\}\ y * (sqrt\ (1 - y^2))^{\text{card } A}\ \partial (distr\ lborel\ borel\ ((*)\ (1/r))))$ 
    using  $\langle r > 0 \rangle$ 
  by (subst nn_integral_distr)

```

```

      (auto simp: indicator_def field_simps real_sqrt_divide intro!: nn_integral_cong)
    also have ... = (∫+ x. ennreal (r ^ Suc (card A)) *
      (indicator {- 1..1} x * sqrt (1 - x2) ^ card A) ∂lborel) using ‹r >
0›
    by (subst lborel_distr_mult) (auto simp: nn_integral_density ennreal_mult'
[symmetric] mult_ac)
    also have ... = ennreal (r ^ Suc (card A)) * (∫+ x. indicator {- 1..1} x *
      sqrt (1 - x2) ^ card A ∂lborel)
    by (subst nn_integral_cmult) auto
    also note emeasure_cball_aux_integral
    also have ennreal (unit_ball_vol (real (card A))) * (ennreal (r ^ Suc (card A)))
*
      ennreal (Beta (1/2) (card A / 2 + 1)) =
      ennreal (unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) * r ^
Suc (card A))
    using ‹r > 0› by (simp add: ennreal_mult' [symmetric] mult_ac)
    also have unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) = unit_ball_vol
(Suc (card A))
    by (auto simp: unit_ball_vol_def Beta_def Gamma_eq_zero_iff field_simps
Gamma_one_half_real_powr_half_sqrt [symmetric] powr_add [symmetric])
    also have Suc (card A) = card (insert i A) using insert.hyps by simp
    finally show ?case .
qed

```

We now get the main theorem very easily by just applying the above lemma.

context

```

fixes c :: 'a :: euclidean_space and r :: real
assumes r: r ≥ 0

```

begin

theorem emeasure_cball:

```

emeasure lborel (cball c r) = ennreal (unit_ball_vol (DIM('a)) * r ^ DIM('a))

```

proof (cases r = 0)

```

case False

```

```

with r have r: r > 0 by simp

```

```

have (lborel :: 'a measure) =

```

```

  distr (Pi_M Basis (λ_. lborel)) borel (λf. ∑ b∈Basis. f b *R b)

```

```

  by (rule lborel_eq)

```

```

also have emeasure ... (cball 0 r) =

```

```

  emeasure (Pi_M Basis (λ_. lborel))

```

```

  ({y. dist 0 (∑ b∈Basis. y b *R b :: 'a) ≤ r} ∩ space (Pi_M Basis (λ_.
lborel)))

```

```

  by (subst emeasure_distr) (auto simp: cball_def)

```

```

also have {f. dist 0 (∑ b∈Basis. f b *R b :: 'a) ≤ r} = {f. sqrt (∑ i∈Basis. (f
i)2) ≤ r}

```

```

  by (subst euclidean_dist_l2) (auto simp: L2_set_def)

```

```

also have emeasure (Pi_M Basis (λ_. lborel)) (... ∩ space (Pi_M Basis (λ_.
lborel))) =

```

```

  ennreal (unit_ball_vol (real DIM('a)) * r ^ DIM('a))

```

```

  using r by (subst emeasure_cball_aux) simp_all
  also have emeasure lborel (cball 0 r :: 'a set) =
    emeasure (distr lborel borel ( $\lambda x. c + x$ )) (cball c r)
  by (subst emeasure_distr) (auto simp: cball_def dist_norm norm_minus_commute)
  also have distr lborel borel ( $\lambda x. c + x$ ) = lborel
  using lborel_affine[of 1 c] by (simp add: density_1)
  finally show ?thesis .
qed auto

```

corollary content_cball:

```

content (cball c r) = unit_ball_vol (DIM('a)) * r ^ DIM('a)
by (simp add: measure_def emeasure_cball r)

```

corollary emeasure_ball:

```

emeasure lborel (ball c r) = ennreal (unit_ball_vol (DIM('a)) * r ^ DIM('a))

```

proof –

```

from negligible_sphere[of c r] have sphere c r ∈ null_sets lborel
by (auto simp: null_sets_completion_iff negligible_iff_null_sets negligible_convex_frontier)
hence emeasure lborel (ball c r ∪ sphere c r :: 'a set) = emeasure lborel (ball c
r :: 'a set)
by (intro emeasure_Un_null_set) auto
also have ball c r ∪ sphere c r = (cball c r :: 'a set) by auto
also have emeasure lborel ... = ennreal (unit_ball_vol (real DIM('a)) * r ^
DIM('a))
by (rule emeasure_cball)
finally show ?thesis ..
qed

```

corollary content_ball:

```

content (ball c r) = unit_ball_vol (DIM('a)) * r ^ DIM('a)
by (simp add: measure_def r_emeasure_ball)

```

end

Lastly, we now prove some nicer explicit formulas for the volume of the unit balls in the cases of even and odd integer dimensions.

lemma unit_ball_vol_even:

```

unit_ball_vol (real (2 * n)) = pi ^ n / fact n
by (simp add: unit_ball_vol_def add_ac powr_realpow Gamma_fact)

```

lemma unit_ball_vol_odd':

```

unit_ball_vol (real (2 * n + 1)) = pi ^ n / pochhammer (1 / 2) (Suc n)

```

and unit_ball_vol_odd:

```

unit_ball_vol (real (2 * n + 1)) =
(2 ^ (2 * Suc n) * fact (Suc n)) / fact (2 * Suc n) * pi ^ n

```

proof –

```

have unit_ball_vol (real (2 * n + 1)) =
pi powr (real n + 1 / 2) / Gamma (1 / 2 + real (Suc n))
by (simp add: unit_ball_vol_def field_simps)

```

also have $\text{pochhammer } (1 / 2) (Suc\ n) = \text{Gamma } (1 / 2 + \text{real } (Suc\ n)) / \text{Gamma } (1 / 2)$
by $(\text{intro pochhammer_Gamma})\ \text{auto}$
hence $\text{Gamma } (1 / 2 + \text{real } (Suc\ n)) = \text{sqrt } \pi * \text{pochhammer } (1 / 2) (Suc\ n)$
by $(\text{simp add: Gamma_one_half_real})$
also have $\pi^{\text{powr } (\text{real } n + 1 / 2) / \dots} = \pi^n / \text{pochhammer } (1 / 2) (Suc\ n)$
by $(\text{simp add: powr_add powr_half_sqrt powr_realpow})$
finally show $\text{unit_ball_vol } (\text{real } (2 * n + 1)) = \dots$
also have $\text{pochhammer } (1 / 2 :: \text{real}) (Suc\ n) = \text{fact } (2 * Suc\ n) / (2^{(2 * Suc\ n)} * \text{fact } (Suc\ n))$
using $\text{fact_double[of } Suc\ n, \text{ where } ?'a = \text{real}]$ **by** $(\text{simp add: divide_simps mult_ac})$
also have $\pi^n / \dots = (2^{(2 * Suc\ n)} * \text{fact } (Suc\ n)) / \text{fact } (2 * Suc\ n) * \pi^n$
by simp
finally show $\text{unit_ball_vol } (\text{real } (2 * n + 1)) = \dots$
qed

lemma $\text{unit_ball_vol_numeral}$:

$\text{unit_ball_vol } (\text{numeral } (\text{Num.Bit0 } n)) = \pi^n \text{ numeral } n / \text{fact } (\text{numeral } n)$ (**is** $?th1$)
 $\text{unit_ball_vol } (\text{numeral } (\text{Num.Bit1 } n)) = 2^{(2 * Suc\ (\text{numeral } n))} * \text{fact } (Suc\ (\text{numeral } n)) / \text{fact } (2 * Suc\ (\text{numeral } n)) * \pi^n$ (**is** $?th2$)

proof –

have $\text{numeral } (\text{Num.Bit0 } n) = (2 * \text{numeral } n :: \text{nat})$
by $(\text{simp only: numeral_Bit0 mult_2 ring_distrib})$
also have $\text{unit_ball_vol } \dots = \pi^n \text{ numeral } n / \text{fact } (\text{numeral } n)$
by $(\text{rule unit_ball_vol_even})$
finally show $?th1$ **by** simp

next

have $\text{numeral } (\text{Num.Bit1 } n) = (2 * \text{numeral } n + 1 :: \text{nat})$
by $(\text{simp only: numeral_Bit1 mult_2})$
also have $\text{unit_ball_vol } \dots = 2^{(2 * Suc\ (\text{numeral } n))} * \text{fact } (Suc\ (\text{numeral } n)) / \text{fact } (2 * Suc\ (\text{numeral } n)) * \pi^n$
by $(\text{rule unit_ball_vol_odd})$
finally show $?th2$ **by** simp

qed

lemmas $\text{eval_unit_ball_vol} = \text{unit_ball_vol_numeral fact_numeral}$

Just for fun, we compute the volume of unit balls for a few dimensions.

lemma $\text{unit_ball_vol_0 [simp]}$: $\text{unit_ball_vol } 0 = 1$
using $\text{unit_ball_vol_even[of } 0]$ **by** simp

lemma $\text{unit_ball_vol_1 [simp]}$: $\text{unit_ball_vol } 1 = 2$
using $\text{unit_ball_vol_odd[of } 0]$ **by** simp

corollary

```

  unit_ball_vol_2: unit_ball_vol 2 = pi
  and unit_ball_vol_3: unit_ball_vol 3 = 4 / 3 * pi
  and unit_ball_vol_4: unit_ball_vol 4 = pi2 / 2
  and unit_ball_vol_5: unit_ball_vol 5 = 8 / 15 * pi2
  by (simp_all add: eval_unit_ball_vol)

```

corollary *circle_area*:

```

  r ≥ 0 ⇒ content (ball c r :: (real ^ 2) set) = r ^ 2 * pi
  by (simp add: content_ball_unit_ball_vol_2)

```

corollary *sphere_volume*:

```

  r ≥ 0 ⇒ content (ball c r :: (real ^ 3) set) = 4 / 3 * r ^ 3 * pi
  by (simp add: content_ball_unit_ball_vol_3)

```

Useful equivalent forms

corollary *content_ball_eq_0_iff* [simp]: $\text{content} (\text{ball } c \ r) = 0 \iff r \leq 0$

proof –

```

  have r > 0 ⇒ content (ball c r) > 0
  by (simp add: content_ball_unit_ball_vol_def)
  then show ?thesis
  by (fastforce simp: ball_empty)

```

qed

corollary *content_ball_gt_0_iff* [simp]: $0 < \text{content} (\text{ball } z \ r) \iff 0 < r$

by (auto simp: zero_less_measure_iff)

corollary *content_cball_eq_0_iff* [simp]: $\text{content} (\text{cball } c \ r) = 0 \iff r \leq 0$

proof (cases r = 0)

```

  case False
  moreover have r > 0 ⇒ content (cball c r) > 0
  by (simp add: content_cball_unit_ball_vol_def)
  ultimately show ?thesis
  by fastforce

```

qed *auto*

corollary *content_cball_gt_0_iff* [simp]: $0 < \text{content} (\text{cball } z \ r) \iff 0 < r$

by (auto simp: zero_less_measure_iff)

end

9.13 Integral Test for Summability

theory *Integral_Test*

imports *Henstock_Kurzweil_Integration*

begin

The integral test for summability. We show here that for a decreasing non-

negative function, the infinite sum over that function evaluated at the natural numbers converges iff the corresponding integral converges.

As a useful side result, we also provide some results on the difference between the integral and the partial sum. (This is useful e.g. for the definition of the Euler-Mascheroni constant)

```

locale antimono_fun_sum_integral_diff =
  fixes f :: real  $\Rightarrow$  real
  assumes dec:  $\bigwedge x y. x \geq 0 \implies x \leq y \implies f x \geq f y$ 
  assumes nonneg:  $\bigwedge x. x \geq 0 \implies f x \geq 0$ 
  assumes cont: continuous_on {0..} f
begin

```

```

definition sum_integral_diff_series n =  $(\sum k \leq n. f (of\_nat\ k)) - (integral\ \{0..of\_nat\ n\}\ f)$ 

```

```

lemma sum_integral_diff_series_nonneg:

```

```

  sum_integral_diff_series n  $\geq 0$ 

```

```

proof -

```

```

  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]

```

```

  let ?int =  $\lambda a b. integral\ \{of\_nat\ a..of\_nat\ b\}\ f$ 

```

```

  have  $-sum\_integral\_diff\_series\ n = ?int\ 0\ n - (\sum k \leq n. f (of\_nat\ k))$ 

```

```

    by (simp add: sum_integral_diff_series_def)

```

```

  also have  $?int\ 0\ n = (\sum k < n. ?int\ k\ (Suc\ k))$ 

```

```

  proof (induction n)

```

```

    case (Suc n)

```

```

    have  $?int\ 0\ (Suc\ n) = ?int\ 0\ n + ?int\ n\ (Suc\ n)$ 

```

```

    by (intro integral_combine[symmetric] int) simp_all

```

```

    with Suc show ?case by simp

```

```

  qed simp_all

```

```

  also have  $\dots \leq (\sum k < n. integral\ \{of\_nat\ k..of\_nat\ (Suc\ k)\}\ (\lambda _::real. f (of\_nat\ k)))$ 

```

```

    by (intro sum_mono integral_le int) (auto intro: dec)

```

```

  also have  $\dots = (\sum k < n. f (of\_nat\ k))$  by simp

```

```

  also have  $\dots - (\sum k \leq n. f (of\_nat\ k)) = -(\sum k \in \{..n\} - \{..<n\}. f (of\_nat\ k))$ 

```

```

    by (subst sum_diff) auto

```

```

  also have  $\dots \leq 0$  by (auto intro!: sum_nonneg nonneg)

```

```

  finally show sum_integral_diff_series n  $\geq 0$  by simp

```

```

qed

```

```

lemma sum_integral_diff_series_antimono:

```

```

  assumes  $m \leq n$ 

```

```

  shows sum_integral_diff_series m  $\geq sum\_integral\_diff\_series\ n$ 

```

```

proof -

```

```

  let ?int =  $\lambda a b. integral\ \{of\_nat\ a..of\_nat\ b\}\ f$ 

```

```

  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]

```

```

  have d_mono: sum_integral_diff_series (Suc n)  $\leq sum\_integral\_diff\_series\ n$ 

```

```

for n

```

```

  proof -

```



```

fix n :: nat
have sum_integral_diff_series (Suc n) - sum_integral_diff_series n =
  f (of_nat (Suc n)) + (?int 0 n - ?int 0 (Suc n))
  unfolding sum_integral_diff_series_def by (simp add: algebra_simps)
also have ?int 0 n - ?int 0 (Suc n) = -?int n (Suc n)
  by (subst integral_combine [symmetric, of of_nat 0 of_nat n of_nat (Suc
n)])
  (auto intro!: int simp: algebra_simps)
also have ?int n (Suc n) ≥ integral {of_nat n..of_nat (Suc n)} (λ_::real. f
(of_nat (Suc n)))
  by (intro integral_le int) (auto intro: dec)
  hence f (of_nat (Suc n)) + -?int n (Suc n) ≤ 0 by (simp add: algebra_simps)
  finally show sum_integral_diff_series (Suc n) ≤ sum_integral_diff_series n
by simp
qed
with assms show ?thesis
  by (induction rule: inc_induct) (auto intro: order.trans[OF _ d_mono])
qed

lemma sum_integral_diff_series_Bseq: Bseq sum_integral_diff_series
proof -
  from sum_integral_diff_series_nonneg and sum_integral_diff_series_antimono
  have norm (sum_integral_diff_series n) ≤ sum_integral_diff_series 0 for n
by simp
  thus Bseq sum_integral_diff_series by (rule BseqI')
qed

lemma sum_integral_diff_series_monoseq: monoseq sum_integral_diff_series
  using sum_integral_diff_series_antimono unfolding monoseq_def by blast

lemma sum_integral_diff_series_convergent: convergent sum_integral_diff_series
  using sum_integral_diff_series_Bseq sum_integral_diff_series_monoseq
  by (blast intro!: Bseq_monoseq_convergent)

theorem integral_test:
  summable (λn. f (of_nat n)) ↔ convergent (λn. integral {0..of_nat n} f)
proof -
  have summable (λn. f (of_nat n)) ↔ convergent (λn. ∑ k≤n. f (of_nat k))
  by (simp add: summable_iff_convergent')
  also have ... ↔ convergent (λn. integral {0..of_nat n} f)
  proof
    assume convergent (λn. ∑ k≤n. f (of_nat k))
    from convergent_diff[OF this sum_integral_diff_series_convergent]
    show convergent (λn. integral {0..of_nat n} f)
    unfolding sum_integral_diff_series_def by simp
  next
    assume convergent (λn. integral {0..of_nat n} f)
    from convergent_add[OF this sum_integral_diff_series_convergent]
    show convergent (λn. ∑ k≤n. f (of_nat k)) unfolding sum_integral_diff_series_def

```

3158

```
by simp
qed
finally show ?thesis by simp
qed

end

end
```

9.14 Continuity of the indefinite integral; improper integral theorem

```
theory Improper_Integral
  imports Equivalence_Lebesgue_Henstock_Integration
begin
```

9.14.1 Equiintegrability

The definition here only really makes sense for an elementary set. We just use compact intervals in applications below.

definition *equiintegrable_on* (**infixr** *equiintegrable'_on* 46)

```
where F equiintegrable_on I ≡
  (∀ f ∈ F. f integrable_on I) ∧
  (∀ e > 0. ∃ γ. gauge γ ∧
    (∀ f ∈ F. f integrable_on I ∧ γ fine D
      → norm ((∑ (x,K) ∈ D. content K *R f x) - integral I f)
        < e))
```

lemma *equiintegrable_on_integrable*:

```
[[F equiintegrable_on I; f ∈ F]] ⇒ f integrable_on I
using equiintegrable_on_def by metis
```

lemma *equiintegrable_on_sing* [*simp*]:

```
{f} equiintegrable_on cbox a b ↔ f integrable_on cbox a b
by (simp add: equiintegrable_on_def has_integral_integral has_integral_integrable_on_def)
```

lemma *equiintegrable_on_subset*: $[[F \text{ equiintegrable_on } I; G \subseteq F]] \Rightarrow G \text{ equiintegrable_on } I$

unfolding *equiintegrable_on_def* *Ball_def*

by (*erule conj_forward imp_forward all_forward ex_forward | blast*)⁺

lemma *equiintegrable_on_Un*:

assumes *F equiintegrable_on I G equiintegrable_on I*

shows $(F \cup G) \text{ equiintegrable_on } I$

unfolding *equiintegrable_on_def*

proof (*intro conjI impI allI*)

show $\forall f \in F \cup G. f \text{ integrable_on } I$

```

using assms unfolding equiintegrable_on_def by blast
show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
   $(\forall f \mathcal{D}. f \in F \cup G \wedge$ 
     $\mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma \text{ fine } \mathcal{D} \longrightarrow$ 
     $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon)$ 
  if  $\varepsilon > 0$  for  $\varepsilon$ 
proof –
  obtain  $\gamma 1$  where gauge  $\gamma 1$ 
  and  $\gamma 1: \bigwedge f \mathcal{D}. f \in F \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma 1 \text{ fine } \mathcal{D}$ 
     $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
  using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by auto
  obtain  $\gamma 2$  where gauge  $\gamma 2$ 
  and  $\gamma 2: \bigwedge f \mathcal{D}. f \in G \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge \gamma 2 \text{ fine } \mathcal{D}$ 
     $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
  using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by auto
  have gauge  $(\lambda x. \gamma 1 x \cap \gamma 2 x)$ 
  using  $\langle \text{gauge } \gamma 1 \rangle \langle \text{gauge } \gamma 2 \rangle$  by blast
  moreover have  $\forall f \mathcal{D}. f \in F \cup G \wedge \mathcal{D} \text{ tagged\_division\_of } I \wedge (\lambda x. \gamma 1 x \cap$ 
 $\gamma 2 x) \text{ fine } \mathcal{D} \longrightarrow$ 
     $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon$ 
  using  $\gamma 1 \ \gamma 2$  by (auto simp: fine_Int)
  ultimately show ?thesis
  by (intro exI conjI assumption+)
qed
qed

```

lemma *equiintegrable_on_insert*:

```

assumes f integrable_on cbox a b F equiintegrable_on cbox a b
shows (insert f F) equiintegrable_on cbox a b
by (metis assms equiintegrable_on_Un equiintegrable_on_sing insert_is_Un)

```

lemma *equiintegrable_cmul*:

```

assumes F: F equiintegrable_on I
shows  $(\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\}) \text{ equiintegrable\_on } I$ 
unfolding equiintegrable_on_def
proof (intro conjI impI allI ballI)
show f integrable_on I
  if  $f \in (\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\})$ 
  for  $f :: 'a \Rightarrow 'b$ 
  using that assms equiintegrable_on_integrable integrable_cmul by blast
show  $\exists \gamma. \text{gauge } \gamma \wedge (\forall f \mathcal{D}. f \in (\bigcup c \in \{-k..k\}. \bigcup f \in F. \{(\lambda x. c *_R f x)\}) \wedge \mathcal{D}$ 
 $\text{tagged\_division\_of } I$ 
   $\wedge \gamma \text{ fine } \mathcal{D} \longrightarrow \text{norm } ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) <$ 
 $\varepsilon)$ 
  if  $\varepsilon > 0$  for  $\varepsilon$ 
proof –
  obtain  $\gamma$  where gauge  $\gamma$ 

```

and $\gamma: \bigwedge f \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged_division_of } I; \gamma \text{ fine } \mathcal{D} \rrbracket$
 $\implies \text{norm} ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon /$
 $(|k| + 1)$
using *assms* $\langle \varepsilon > 0 \rangle$ **unfolding** *equiintegrable_on_def*
by (*metis add commute add.right_neutral add_strict_mono divide_pos_pos*
norm_eq_zero real_norm_def zero_less_norm_iff zero_less_one)
moreover have $\text{norm} ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R c *_R (f x)) - \text{integral } I$
 $(\lambda x. c *_R f x)) < \varepsilon$
if $c: c \in \{-k..k\}$
and $f \in F \mathcal{D} \text{ tagged_division_of } I \gamma \text{ fine } \mathcal{D}$
for $\mathcal{D} c f$
proof –
have $\text{norm} ((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K *_R c *_R f x) - \text{integral}$
 $I (\lambda x. c *_R f x))$
 $= |c| * \text{norm} ((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K *_R f x) - \text{integral}$
 $I f)$
by (*simp add: algebra_simps scale_sum_right case_prod_unfold flip:*
norm_scaleR)
also have $\dots \leq (|k| + 1) * \text{norm} ((\sum x \in \mathcal{D}. \text{case } x \text{ of } (x, K) \Rightarrow \text{content } K$
 $*_R f x) - \text{integral } I f)$
using c **by** (*auto simp: mult_right_mono*)
also have $\dots < (|k| + 1) * (\varepsilon / (|k| + 1))$
by (*rule mult_strict_left_mono*) (*use* γ *less_eq_real_def* **that** **in** *auto*)
also have $\dots = \varepsilon$
by *auto*
finally show *?thesis* .
qed
ultimately show *?thesis*
by (*rule_tac x= γ in exI*) *auto*
qed
qed

lemma *equiintegrable_add:*

assumes $F: F \text{ equiintegrable_on } I$ **and** $G: G \text{ equiintegrable_on } I$
shows $(\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\}) \text{ equiintegrable_on } I$
unfolding *equiintegrable_on_def*
proof (*intro conjI impI allI ballI*)
show $f \text{ integrable_on } I$
if $f \in (\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\})$ **for** f
using *that equiintegrable_on_integrable assms* **by** (*auto intro: integrable_add*)
show $\exists \gamma. \text{gauge } \gamma \wedge (\forall f \mathcal{D}. f \in (\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x + g x)\}) \wedge \mathcal{D}$
 $\text{tagged_division_of } I$
 $\wedge \gamma \text{ fine } \mathcal{D} \longrightarrow \text{norm} ((\sum (x, K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) <$
 $\varepsilon)$
if $\varepsilon > 0$ **for** ε
proof –
obtain $\gamma 1$ **where** *gauge* $\gamma 1$
and $\gamma 1: \bigwedge f \mathcal{D}. \llbracket f \in F; \mathcal{D} \text{ tagged_division_of } I; \gamma 1 \text{ fine } \mathcal{D} \rrbracket$

```

       $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) < \varepsilon/2$ 
using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
obtain  $\gamma 2$  where gauge  $\gamma 2$ 
  and  $\gamma 2: \bigwedge g \mathcal{D}. \llbracket g \in G; \mathcal{D} \text{ tagged\_division\_of } I; \gamma 2 \text{ fine } \mathcal{D} \rrbracket$ 
       $\implies \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g x) - \text{integral } I g) < \varepsilon/2$ 
using assms  $\langle \varepsilon > 0 \rangle$  unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
have gauge  $(\lambda x. \gamma 1 x \cap \gamma 2 x)$ 
  using  $\langle \text{gauge } \gamma 1 \rangle \langle \text{gauge } \gamma 2 \rangle$  by blast
moreover have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R h x) - \text{integral } I h) < \varepsilon$ 
if  $h: h \in (\bigcup f \in F. \bigcup g \in G. \{\lambda x. f x + g x\})$ 
  and  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of } I$  and fine:  $(\lambda x. \gamma 1 x \cap \gamma 2 x) \text{ fine } \mathcal{D}$ 
for  $h \mathcal{D}$ 
proof -
obtain  $f g$  where  $f \in F g \in G$  and heq:  $h = (\lambda x. f x + g x)$ 
  using h by blast
then have int:  $f \text{ integrable\_on } I g \text{ integrable\_on } I$ 
  using  $F G \text{ equiintegrable\_on\_def}$  by blast+
have  $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R h x) - \text{integral } I h)$ 
   $= \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x + \text{content } K *_R g x) - (\text{integral } I f + \text{integral } I g))$ 
  by (simp add: heq algebra_simps integral_add int)
also have  $\dots = \text{norm } (((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) +$ 
 $(\sum (x,K) \in \mathcal{D}. \text{content } K *_R g x) - \text{integral } I g))$ 
  by (simp add: sum.distrib algebra_simps case_prod_unfold)
also have  $\dots \leq \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f x) - \text{integral } I f) +$ 
 $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g x) - \text{integral } I g)$ 
  by (metis (mono_tags) add_diff_eq norm_triangle_ineq)
also have  $\dots < \varepsilon/2 + \varepsilon/2$ 
  using  $\gamma 1 [OF \langle f \in F \rangle \mathcal{D}] \gamma 2 [OF \langle g \in G \rangle \mathcal{D}]$  fine by (simp add: fine_Int)
finally show ?thesis by simp
qed
ultimately show ?thesis
  by meson
qed

```

lemma *equiintegrable_minus*:

```

assumes  $F \text{ equiintegrable\_on } I$ 
shows  $(\bigcup f \in F. \{(\lambda x. - f x)\}) \text{ equiintegrable\_on } I$ 
by (force intro: equiintegrable_on_subset [OF equiintegrable_cmul [OF assms, of 1]])

```

lemma *equiintegrable_diff*:

```

assumes  $F: F \text{ equiintegrable\_on } I$  and  $G: G \text{ equiintegrable\_on } I$ 
shows  $(\bigcup f \in F. \bigcup g \in G. \{(\lambda x. f x - g x)\}) \text{ equiintegrable\_on } I$ 
by (rule equiintegrable_on_subset [OF equiintegrable_add [OF F equiintegrable_minus [OF G]]] auto)

```

lemma *equiintegrable_sum*:
fixes $F :: ('a::euclidean_space \Rightarrow 'b::euclidean_space)$ set
assumes F *equiintegrable_on_cbox* a b
shows $(\bigcup I \in \text{Collect finite. } \bigcup c \in \{c. (\forall i \in I. c\ i \geq 0) \wedge \text{sum } c\ I = 1\}.$
 $\bigcup f \in I \rightarrow F. \{(\lambda x. \text{sum } (\lambda i::'j. c\ i *_R f\ i\ x)\ I)\})$ *equiintegrable_on_cbox*
 a b
(is $?G$ *equiintegrable_on_* $_)$
unfolding *equiintegrable_on_def*
proof (*intro conjI impI allI ballI*)
show f *integrable_on_cbox* a b **if** $f \in ?G$ **for** f
using *that assms* **by** (*auto simp: equiintegrable_on_def intro!: integrable_sum*
integrable_cmul)
show $\exists \gamma. \text{gauge } \gamma$
 $\wedge (\forall g\ \mathcal{D}. g \in ?G \wedge \mathcal{D}$ *tagged_division_of_cbox* a $b \wedge \gamma$ *fine* \mathcal{D}
 $\rightarrow \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g\ x) - \text{integral } (\text{cbox } a\ b)\ g)$
 $< \varepsilon)$
if $\varepsilon > 0$ **for** ε
proof –
obtain γ **where** *gauge* γ
and $\gamma: \bigwedge f\ \mathcal{D}. \llbracket f \in F; \mathcal{D}$ *tagged_division_of_cbox* a $b; \gamma$ *fine* $\mathcal{D} \rrbracket$
 $\Rightarrow \text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ x) - \text{integral } (\text{cbox } a\ b)$
 $f) < \varepsilon / 2$
using *assms* $\langle \varepsilon > 0 \rangle$ **unfolding** *equiintegrable_on_def* **by** (*meson half_gt_zero_iff*)
moreover **have** $\text{norm } ((\sum (x,K) \in \mathcal{D}. \text{content } K *_R g\ x) - \text{integral } (\text{cbox } a$
 $b)\ g) < \varepsilon$
if $g: g \in ?G$
and $\mathcal{D}: \mathcal{D}$ *tagged_division_of_cbox* a b
and *fine:* γ *fine* \mathcal{D}
for $\mathcal{D}\ g$
proof –
obtain $I\ c\ f$ **where** *finite* I **and** $0: \bigwedge i::'j. i \in I \Rightarrow 0 \leq c\ i$
and $1: \text{sum } c\ I = 1$ **and** $f: f \in I \rightarrow F$ **and** *geq:* $g = (\lambda x. \sum i \in I. c\ i *_R f$
 $i\ x)$
using g **by** *auto*
have *fi_int:* $f\ i$ *integrable_on_cbox* a b **if** $i \in I$ **for** i
by (*metis* Pi_iff *assms equiintegrable_on_def* f *that*)
have $*$: $\text{integral } (\text{cbox } a\ b)\ (\lambda x. c\ i *_R f\ i\ x) = (\sum (x, K) \in \mathcal{D}. \text{integral } K\ (\lambda x.$
 $c\ i *_R f\ i\ x))$
if $i \in I$ **for** i
proof –
have $f\ i$ *integrable_on_cbox* a b
by (*metis* Pi_iff *assms equiintegrable_on_def* f *that*)
then show *?thesis*
by (*intro* \mathcal{D} *integrable_cmul integral_combine_tagged_division_topdown*)
qed
have *finite* \mathcal{D}
using \mathcal{D} **by** *blast*
have *swap:* $(\sum (x,K) \in \mathcal{D}. \text{content } K *_R (\sum i \in I. c\ i *_R f\ i\ x))$
 $= (\sum i \in I. c\ i *_R (\sum (x,K) \in \mathcal{D}. \text{content } K *_R f\ i\ x))$

```

    by (simp add: scale_sum_right case_prod_unfold algebra_simps) (rule
sum.swap)
  have norm (( $\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g x$ ) - integral (cbox a b) g)
    = norm (( $\sum i \in I. c i *_{\mathbb{R}} ((\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f i x) - \text{integral}
(cbox a b) (f i))$ ))
    unfolding geq_swap
    by (simp add: scaleR_right.sum algebra_simps integral_sum fi_int inte-
grable_cmul ⟨finite I⟩ sum_subtractf flip: sum_diff)
    also have ... ≤ ( $\sum i \in I. c i * \varepsilon / 2$ )
    proof (rule sum_norm_le)
      show norm (c i *ℝ (( $\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f i x$ ) - integral (cbox
a b) (f i))) ≤ c i * ε / 2
        if i ∈ I for i
      proof -
        have norm (( $\sum (x, K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f i x$ ) - integral (cbox a b) (f i))
          ≤ ε/2
          using γ [OF _ D fine, of f i] funcset_mem [OF f] that by auto
          then show ?thesis
            using that by (auto simp: 0 mult.assoc intro: mult_left_mono)
        qed
      qed
    also have ... < ε
      using 1 <ε > 0 by (simp add: flip: sum_divide_distrib sum_distrib_right)
    finally show ?thesis .
  qed
  ultimately show ?thesis
    by (rule_tac x=γ in exI) auto
  qed
qed

```

corollary *equiintegrable_sum_real:*

```

  fixes F :: (real ⇒ 'b::euclidean_space) set
  assumes F equiintegrable_on {a..b}
  shows (⋃ I ∈ Collect finite. ⋃ c ∈ {c. (∀ i ∈ I. c i ≥ 0) ∧ sum c I = 1}.
    ⋃ f ∈ I → F. {(λx. sum (λi. c i *ℝ f i x) I)})
    equiintegrable_on {a..b}
  using equiintegrable_sum [of F a b] assms by auto

```

Basic combining theorems for the interval of integration.

lemma *equiintegrable_on_null [simp]:*

```

  content(cbox a b) = 0 ⇒ F equiintegrable_on cbox a b
  unfolding equiintegrable_on_def
  by (metis diff_zero gauge_trivial integrable_on_null integral_null norm_zero
sum_content_null)

```

Main limit theorem for an equiintegrable sequence.

theorem *equiintegrable_limit:*

```

  fixes g :: 'a :: euclidean_space ⇒ 'b :: banach
  assumes feq: range f equiintegrable_on cbox a b

```

```

and to_g:  $\bigwedge x. x \in \text{cbox } a \ b \implies (\lambda n. f \ n \ x) \longrightarrow g \ x$ 
shows g integrable_on cbox a b  $\wedge (\lambda n. \text{integral } (\text{cbox } a \ b) \ (f \ n)) \longrightarrow \text{integral}$ 
(cbox a b) g
proof –
have Cauchy  $(\lambda n. \text{integral}(\text{cbox } a \ b) \ (f \ n))$ 
proof (clarsimp simp add: Cauchy_def)
fix e::real
assume  $0 < e$ 
then have e3:  $0 < e/3$ 
by simp
then obtain  $\gamma$  where gauge  $\gamma$ 
and  $\gamma: \bigwedge n \ \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; \ \gamma \text{ fine } \mathcal{D} \rrbracket$ 
 $\implies \text{norm}((\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ n \ x) - \text{integral } (\text{cbox}$ 
a b) (f n)) < e/3
using feq unfolding equiintegrable_on_def
by (meson image_eqI iso_tuple_UNIV_I)
obtain  $\mathcal{D}$  where  $\mathcal{D}: \mathcal{D} \text{ tagged\_division\_of } (\text{cbox } a \ b)$  and  $\gamma \text{ fine } \mathcal{D}$  finite  $\mathcal{D}$ 
by (meson ‹gauge ‹ $\gamma$ › fine_division_exists tagged_division_of_finite)
with  $\gamma$  have  $\delta T: \bigwedge n. \text{dist } ((\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ n \ x)) \ (\text{integral } (\text{cbox}$ 
a b) (f n)) < e/3
by (force simp: dist_norm)
have  $(\lambda n. \sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ n \ x) \longrightarrow (\sum (x,K) \in \mathcal{D}. \text{content } K$ 
 $\ *_R \ g \ x)$ 
using  $\mathcal{D}$  to_g by (auto intro!: tendsto_sum tendsto_scaleR)
then have Cauchy  $(\lambda n. \sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ n \ x)$ 
by (meson convergent_eq_Cauchy)
with e3 obtain M where
 $M: \bigwedge m \ n. \llbracket m \geq M; \ n \geq M \rrbracket \implies \text{dist } (\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ m \ x)$ 
 $(\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ n \ x)$ 
 $< e/3$ 
unfolding Cauchy_def by blast
have  $\bigwedge m \ n. \llbracket m \geq M; \ n \geq M; \ \text{dist } (\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f \ m \ x) \ (\sum (x,K) \in \mathcal{D}. \text{content } K \ *_R \ f$ 
 $\ n \ x) < e/3 \rrbracket$ 
 $\implies \text{dist } (\text{integral } (\text{cbox } a \ b) \ (f \ m)) \ (\text{integral } (\text{cbox } a \ b) \ (f \ n)) < e$ 
by (metis ‹ $\delta T$ › dist_commute dist_triangle_third [OF __ ‹ $\delta T$ ›])
then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (\text{integral } (\text{cbox } a \ b) \ (f \ m)) \ (\text{integral } (\text{cbox}$ 
a b) (f n)) < e
using M by auto
qed
then obtain L where  $L: (\lambda n. \text{integral } (\text{cbox } a \ b) \ (f \ n)) \longrightarrow L$ 
by (meson convergent_eq_Cauchy)
have (g has_integral L) (cbox a b)
proof (clarsimp simp: has_integral)
fix e::real assume  $0 < e$ 
then have e2:  $0 < e/2$ 
by simp
then obtain  $\gamma$  where gauge  $\gamma$ 
and  $\gamma: \bigwedge n \ \mathcal{D}. \llbracket \mathcal{D} \text{ tagged\_division\_of } \text{cbox } a \ b; \ \gamma \text{ fine } \mathcal{D} \rrbracket$ 

```



```

     $\implies \text{norm}((\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ n \ x) - \text{integral } (\text{cbox } a \ b) (f \ n)) < e/2$ 
    using feq unfolding equiintegrable_on_def
    by (meson image_eqI iso_tuple_UNIV_I)
  moreover
  have norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x$ ) - L) < e
    if  $\mathcal{D}$  tagged_division_of cbox a b  $\gamma$  fine  $\mathcal{D}$  for  $\mathcal{D}$ 
  proof -
    have norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x$ ) - L)  $\leq e/2$ 
    proof (rule Lim_norm_ubound)
      show  $(\lambda n. (\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ n \ x) - \text{integral } (\text{cbox } a \ b) (f \ n))$ 
 $\longrightarrow (\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x) - L$ 
      using to_g that L
      by (intro tendsto_diff tendsto_sum) (auto simp: tag_in_interval tendsto_scaleR)
    show  $\forall_F n$  in sequentially.
      norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ n \ x$ ) -  $\text{integral } (\text{cbox } a \ b) (f \ n)$ )  $\leq e/2$ 
      by (intro eventuallyI less_imp_le  $\gamma$  that)
    qed auto
    with  $\langle 0 < e \rangle$  show ?thesis
      by linarith
    qed
  ultimately
  show  $\exists \gamma. \text{gauge } \gamma \wedge$ 
    ( $\forall \mathcal{D}. \mathcal{D}$  tagged_division_of cbox a b  $\wedge \gamma$  fine  $\mathcal{D} \longrightarrow$ 
      norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} g \ x$ ) - L) < e)
    by meson
  qed
  with L show ?thesis
    by (simp add:  $\langle \lambda n. \text{integral } (\text{cbox } a \ b) (f \ n) \longrightarrow L \rangle$  has_integral_integrable_integral)
  qed

```

lemma equiintegrable_reflect:

```

  assumes F equiintegrable_on cbox a b
  shows  $(\lambda f. f \circ \text{uminus}) ' F$  equiintegrable_on cbox (-b) (-a)
  proof -
    have  $\S: \exists \gamma. \text{gauge } \gamma \wedge$ 
      ( $\forall f \mathcal{D}. f \in (\lambda f. f \circ \text{uminus}) ' F \wedge \mathcal{D}$  tagged_division_of cbox (-b) (-a)  $\wedge \gamma$  fine  $\mathcal{D} \longrightarrow$ 
        norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x$ ) -  $\text{integral } (\text{cbox } (-b) (-a)) f$ ) < e)
      if gauge  $\gamma$  and
         $\gamma: \wedge f \mathcal{D}. \llbracket f \in F; \mathcal{D}$  tagged_division_of cbox a b;  $\gamma$  fine  $\mathcal{D} \rrbracket \implies$ 
          norm (( $\sum (x,K) \in \mathcal{D}. \text{content } K *_{\mathbb{R}} f \ x$ ) -  $\text{integral } (\text{cbox } a \ b) f$ )
        < e for e  $\gamma$ 
      proof (intro exI, safe)
        show gauge  $(\lambda x. \text{uminus } ' \gamma (-x))$ 

```

```

    by (metis ‹gauge  $\gamma$ › gauge_reflect)
  show norm (( $\sum (x,K) \in \mathcal{D}$ . content  $K *_R (f \circ \text{uminus}) x$ ) - integral (cbox (-
b) (- a)) (f  $\circ$  uminus)) < e
    if f  $\in F$  and tag:  $\mathcal{D}$  tagged_division_of cbox (- b) (- a)
      and fine: ( $\lambda x$ . uminus ‹ $\gamma$  (- x)›) fine  $\mathcal{D}$  for f  $\mathcal{D}$ 
  proof -
    have 1: ( $\lambda(x,K)$ . (- x, uminus ‹K›)) ‹ $\mathcal{D}$  tagged_partial_division_of cbox a b
      if  $\mathcal{D}$  tagged_partial_division_of cbox (- b) (- a)
    proof -
      have - y  $\in$  cbox a b
        if  $\bigwedge x K$ . (x,K)  $\in \mathcal{D} \implies x \in K \wedge K \subseteq$  cbox (- b) (- a)  $\wedge (\exists a b$ . K =
cbox a b)
          (x, Y)  $\in \mathcal{D} y \in Y$  for x Y y
    proof -
      have y  $\in$  uminus ‹cbox a b›
        using that by auto
      then show - y  $\in$  cbox a b
        by force
    qed
    with that show ?thesis
      by (fastforce simp: tagged_partial_division_of_def interior_negations
image_iff)
    qed
    have 2:  $\exists K$ . ( $\exists x$ . (x,K)  $\in (\lambda(x,K)$ . (- x, uminus ‹K›)) ‹ $\mathcal{D}$ ›)  $\wedge x \in K$ 
      if  $\bigcup \{K$ .  $\exists x$ . (x,K)  $\in \mathcal{D}\} =$  cbox (- b) (- a) x  $\in$  cbox a b for x
    proof -
      have xm: x  $\in$  uminus ‹ $\bigcup \{A$ .  $\exists a$ . (a, A)  $\in \mathcal{D}\}$ ›
        by (simp add: that)
      then obtain a X where -x  $\in X$  (a, X)  $\in \mathcal{D}$ 
        by auto
      then show ?thesis
        by (metis (no_types, lifting) add.inverse_inverse image_iff pair_imageI)
    qed
    have 3:  $\bigwedge x X y$ . [ $\mathcal{D}$  tagged_partial_division_of cbox (- b) (- a); (x, X)  $\in$ 
 $\mathcal{D}$ ; y  $\in X$ ]  $\implies - y \in$  cbox a b
      by (metis (no_types, lifting) equation_minus_iff imageE subsetD tagged_partial_division_ofD(3)
uminus_interval_vector)
    have tag': ( $\lambda(x,K)$ . (- x, uminus ‹K›)) ‹ $\mathcal{D}$  tagged_division_of cbox a b›
      using tag by (auto simp: tagged_division_of_def dest: 1 2 3)
    have fine':  $\gamma$  fine ( $\lambda(x,K)$ . (- x, uminus ‹K›)) ‹ $\mathcal{D}$ ›
      using fine by (fastforce simp: fine_def)
    have inj: inj_on ( $\lambda(x,K)$ . (- x, uminus ‹K›))  $\mathcal{D}$ 
      unfolding inj_on_def by force
    have eq: content (uminus ‹I›) = content I
      if I: (x, I)  $\in \mathcal{D}$  and fnz: f (- x)  $\neq 0$  for x I
    proof -
      obtain a b where I = cbox a b
        using tag I that by (force simp: tagged_division_of_def tagged_partial_division_of_def)
      then show ?thesis

```

```

      using content_image_affinity_cbox [of -1 0] by auto
    qed
  have ( $\sum (x,K) \in (\lambda(x,K). (- x, uminus ' K)) ' \mathcal{D}. content K *_R f x =$ 
    ( $\sum (x,K) \in \mathcal{D}. content K *_R f (- x)$ )
    by (auto simp add: eq_sum.reindex [OF inj] intro!: sum.cong)
  then show ?thesis
    using  $\gamma$  [OF ⟨f ∈ F⟩ tag' fine'] integral_reflect
    by (metis (mono_tags, lifting) Henstock_Kurzweil_Integration.integral_cong
    comp_apply split_def sum.cong)
  qed
  qed
  show ?thesis
    using assms
    apply (auto simp: equiintegrable_on_def)
    subgoal for f
      by (metis (mono_tags, lifting) comp_apply integrable_eq integrable_reflect)
    using § by fastforce
  qed

```

9.14.2 Subinterval restrictions for equiintegrable families

First, some technical lemmas about minimizing a "flat" part of a sum over a division.

lemma *lemma0*:

```

  assumes  $i \in Basis$ 
  shows  $content (cbox u v) / (interval\_upperbound (cbox u v) \cdot i - interval\_lowerbound (cbox u v) \cdot i) =$ 
    ( $if content (cbox u v) = 0 then 0$ 
    else  $\prod j \in Basis - \{i\}. interval\_upperbound (cbox u v) \cdot j - interval\_lowerbound (cbox u v) \cdot j$ )
  proof (cases  $content (cbox u v) = 0$ )
  case True
  then show ?thesis by simp
  next
  case False
  then show ?thesis
    using prod_subset_diff [of {i} Basis] assms
    by (force simp: content_cbox_if divide_simps split: if_split_asm)
  qed

```

lemma *content_division_lemma1*:

```

  assumes  $div: \mathcal{D} \text{ division\_of } S$  and  $S: S \subseteq cbox a b$  and  $i: i \in Basis$ 
  and  $mt: \bigwedge K. K \in \mathcal{D} \implies content K \neq 0$ 
  and  $disj: (\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = a \cdot i\} \neq \{\}) \vee (\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\})$ 
  shows  $(b \cdot i - a \cdot i) * (\sum K \in \mathcal{D}. content K / (interval\_upperbound K \cdot i - interval\_lowerbound K \cdot i))$ 
     $\leq content(cbox a b)$  (is ?lhs  $\leq$  ?rhs)

```

proof –

have *finite* \mathcal{D}

using *div* **by** *blast*

define *extend* **where**

extend $\equiv \lambda K. \text{cbox } (\sum j \in \text{Basis}. \text{if } j = i \text{ then } (a \cdot i) *_R i \text{ else } (\text{interval_lowerbound } K \cdot j) *_R j)$

$(\sum j \in \text{Basis}. \text{if } j = i \text{ then } (b \cdot i) *_R i \text{ else } (\text{interval_upperbound } K \cdot j) *_R j)$

have *div_subset_cbox*: $\bigwedge K. K \in \mathcal{D} \implies K \subseteq \text{cbox } a \ b$

using *S div* **by** *auto*

have $\bigwedge K. K \in \mathcal{D} \implies K \neq \{\}$

using *div* **by** *blast*

have *extend_cbox*: $\bigwedge K. K \in \mathcal{D} \implies \exists a \ b. \text{extend } K = \text{cbox } a \ b$

using *extend_def* **by** *blast*

have *extend*: *extend* $K \neq \{\}$ *extend* $K \subseteq \text{cbox } a \ b$ **if** $K: K \in \mathcal{D}$ **for** K

proof –

obtain $u \ v$ **where** $K: K = \text{cbox } u \ v \ K \neq \{\}$ $K \subseteq \text{cbox } a \ b$

using *K cbox_division_memE* [*OF _ div*] **by** (*meson div_subset_cbox*)

with i **show** *extend* $K \subseteq \text{cbox } a \ b$

by (*auto simp: extend_def subset_box box_ne_empty*)

have $a \cdot i \leq b \cdot i$

using K **by** (*metis bot.extremum_uniqueI box_ne_empty(1) i*)

with K **show** *extend* $K \neq \{\}$

by (*simp add: extend_def i box_ne_empty*)

qed

have *int_extend_disjoint*:

$\text{interior}(\text{extend } K1) \cap \text{interior}(\text{extend } K2) = \{\}$ **if** $K: K1 \in \mathcal{D} \ K2 \in \mathcal{D} \ K1 \neq K2$ **for** $K1 \ K2$

proof –

obtain $u \ v$ **where** $K1: K1 = \text{cbox } u \ v \ K1 \neq \{\}$ $K1 \subseteq \text{cbox } a \ b$

using *K cbox_division_memE* [*OF _ div*] **by** (*meson div_subset_cbox*)

obtain $w \ z$ **where** $K2: K2 = \text{cbox } w \ z \ K2 \neq \{\}$ $K2 \subseteq \text{cbox } a \ b$

using *K cbox_division_memE* [*OF _ div*] **by** (*meson div_subset_cbox*)

have *cboxes*: $\text{cbox } u \ v \in \mathcal{D} \ \text{cbox } w \ z \in \mathcal{D} \ \text{cbox } u \ v \neq \text{cbox } w \ z$

using $K1 \ K2$ **that** **by** *auto*

with *div* **have** $\text{interior}(\text{cbox } u \ v) \cap \text{interior}(\text{cbox } w \ z) = \{\}$

by *blast*

moreover

have $\exists x. x \in \text{box } u \ v \wedge x \in \text{box } w \ z$

if $x \in \text{interior}(\text{extend } K1) \ x \in \text{interior}(\text{extend } K2)$ **for** x

proof –

have $a \cdot i < x \cdot i \ x \cdot i < b \cdot i$

and $ux: \bigwedge k. k \in \text{Basis} - \{i\} \implies u \cdot k < x \cdot k$

and $xv: \bigwedge k. k \in \text{Basis} - \{i\} \implies x \cdot k < v \cdot k$

and $wx: \bigwedge k. k \in \text{Basis} - \{i\} \implies w \cdot k < x \cdot k$

and $xz: \bigwedge k. k \in \text{Basis} - \{i\} \implies x \cdot k < z \cdot k$

using *that* $K1 \ K2 \ i$ **by** (*auto simp: extend_def box_ne_empty mem_box*)

have $\text{box } u \ v \neq \{\}$ $\text{box } w \ z \neq \{\}$

using *cboxes interior_cbox* **by** (*auto simp: content_eq_0_interior dest: mt*)

```

then obtain q s
  where q:  $\bigwedge k. k \in \text{Basis} \implies w \cdot k < q \cdot k \wedge q \cdot k < z \cdot k$ 
    and s:  $\bigwedge k. k \in \text{Basis} \implies u \cdot k < s \cdot k \wedge s \cdot k < v \cdot k$ 
  by (meson all_not_in_conv mem_box(1))
show ?thesis using disj
proof
  assume  $\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 
  then have wva:  $(\text{cbox } u \ v) \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 
    and wza:  $(\text{cbox } w \ z) \cap \{x. x \cdot i = a \cdot i\} \neq \{\}$ 
    using cboxes by (auto simp: content_eq_0_interior)
  then obtain r t where  $r \cdot i = a \cdot i$  and  $r: \bigwedge k. k \in \text{Basis} \implies w \cdot k \leq r \cdot k \wedge r \cdot k \leq z \cdot k$ 
    and  $t \cdot i = a \cdot i$  and  $t: \bigwedge k. k \in \text{Basis} \implies u \cdot k \leq t \cdot k \wedge t \cdot k \leq v \cdot k$ 
    by (fastforce simp: mem_box)
  have u:  $u \cdot i < q \cdot i$ 
    using i K2(1) K2(3)  $\langle t \cdot i = a \cdot i \rangle$  q s t [OF i] by (force simp: subset_box)
  have w:  $w \cdot i < s \cdot i$ 
    using i K1(1) K1(3)  $\langle r \cdot i = a \cdot i \rangle$  s r [OF i] by (force simp: subset_box)
  define  $\xi$  where  $\xi \equiv (\sum j \in \text{Basis}. \text{if } j = i \text{ then } \min (q \cdot i) (s \cdot i) *_{\mathbb{R}} i \text{ else } (x \cdot j) *_{\mathbb{R}} j)$ 
  have [simp]:  $\xi \cdot j = (\text{if } j = i \text{ then } \min (q \cdot j) (s \cdot j) \text{ else } x \cdot j)$  if  $j \in \text{Basis}$ 
for j
  unfolding  $\xi\_def$ 
  by (intro sum_if_inner that  $\langle i \in \text{Basis} \rangle$ )
show ?thesis
proof (intro exI conjI)
  have  $\min (q \cdot i) (s \cdot i) < v \cdot i$ 
    using i s by fastforce
  with  $\langle i \in \text{Basis} \rangle$  s u ux xv
  show  $\xi \in \text{box } u \ v$ 
    by (force simp: mem_box)
  have  $\min (q \cdot i) (s \cdot i) < z \cdot i$ 
    using i q by force
  with  $\langle i \in \text{Basis} \rangle$  q w wx xz
  show  $\xi \in \text{box } w \ z$ 
    by (force simp: mem_box)
qed
next
  assume  $\forall K \in \mathcal{D}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
  then have wva:  $(\text{cbox } u \ v) \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
    and wza:  $(\text{cbox } w \ z) \cap \{x. x \cdot i = b \cdot i\} \neq \{\}$ 
    using cboxes by (auto simp: content_eq_0_interior)
  then obtain r t where  $r \cdot i = b \cdot i$  and  $r: \bigwedge k. k \in \text{Basis} \implies w \cdot k \leq r \cdot k \wedge r \cdot k \leq z \cdot k$ 
    and  $t \cdot i = b \cdot i$  and  $t: \bigwedge k. k \in \text{Basis} \implies u \cdot k \leq t \cdot k \wedge t \cdot k \leq v \cdot k$ 
    by (fastforce simp: mem_box)
  have z:  $s \cdot i < z \cdot i$ 

```

```

    using K1(1) K1(3) ⟨r · i = b · i⟩ r [OF i] i s by (force simp: subset_box)
    have v: q · i < v · i
    using K2(1) K2(3) ⟨t · i = b · i⟩ t [OF i] i q by (force simp: subset_box)
    define ξ where ξ ≡ (∑ j ∈ Basis. if j = i then max (q · i) (s · i) *R i else
(x · j) *R j)
    have [simp]: ξ · j = (if j = i then max (q · j) (s · j) else x · j) if j ∈ Basis
for j
    unfolding ξ_def
    by (intro sum_if_inner that ⟨i ∈ Basis⟩)
    show ?thesis
    proof (intro exI conjI)
      show ξ ∈ box u v
      using ⟨i ∈ Basis⟩ s by (force simp: mem_box ux v xv)
      show ξ ∈ box w z
      using ⟨i ∈ Basis⟩ q by (force simp: mem_box wx xz z)
    qed
  qed
  qed
  ultimately show ?thesis by auto
  qed
  define interv_diff where interv_diff ≡ λK. λi::'a. interval_upperbound K · i
- interval_lowerbound K · i
  have ?lhs = (∑ K ∈ D. (b · i - a · i) * content K / (interv_diff K i))
  by (simp add: sum_distrib_left interv_diff_def)
  also have ... = sum (content ∘ extend) D
  proof (rule sum.cong [OF refl])
    fix K assume K ∈ D
    then obtain u v where K: K = cbox u v cbox u v ≠ {} K ⊆ cbox a b
    using cbox_division_memE [OF _ div] div_subset_cbox by metis
    then have uv: u · i < v · i
    using mt [OF ⟨K ∈ D⟩] ⟨i ∈ Basis⟩ content_eq_0 by fastforce
    have insert_i (Basis ∩ -{i}) = Basis
    using ⟨i ∈ Basis⟩ by auto
    then have (b · i - a · i) * content K / (interv_diff K i)
      = (b · i - a · i) * (∏ i ∈ insert_i (Basis ∩ -{i}). v · i - u · i) /
(interv_diff (cbox u v) i)
    using K box_ne_empty(1) content_cbox by fastforce
    also have ... = (∏ x ∈ Basis. if x = i then b · x - a · x
      else (interval_upperbound (cbox u v) - interval_lowerbound (cbox
u v)) · x)
    using ⟨i ∈ Basis⟩ K uv by (simp add: prod.If_cases interv_diff_def) (simp
add: algebra_simps)
    also have ... = (∏ k ∈ Basis.
      (∑ j ∈ Basis. if j = i then (b · i - a · i) *R i
        else ((interval_upperbound (cbox u v) -
interval_lowerbound (cbox u v)) · j) *R j) · k)
    using ⟨i ∈ Basis⟩ by (subst prod.cong [OF refl sum_if_inner]; simp)
    also have ... = (∏ k ∈ Basis.
      (∑ j ∈ Basis. if j = i then (b · i) *R i else (interval_upperbound

```

```

(cbox u v) · j) *R j) · k -
      (∑ j∈Basis. if j = i then (a · i) *R i else (interval_lowerbound
(cbox u v) · j) *R j) · k)
  using ‹i ∈ Basis›
  by (intro prod.cong [OF refl]) (subst sum_if_inner; simp add: algebra_simps)+
  also have ... = (content ∘ extend) K
    using ‹i ∈ Basis› K box_ne_empty ‹K ∈ ℳ› extend(1)
    by (auto simp add: extend_def content_cbox_if)
  finally show (b · i - a · i) * content K / (interval_diff K i) = (content ∘
extend) K .
qed
also have ... = sum content (extend ‘ ℳ)
proof -
  have [[K1 ∈ ℳ; K2 ∈ ℳ; K1 ≠ K2; extend K1 = extend K2]] ⇒ content
(extend K1) = 0 for K1 K2
    using int_extend_disjoint [of K1 K2] extend_def by (simp add: con-
tent_eq_0_interior)
  then show ?thesis
    by (simp add: comm_monoid_add_class.sum_reindex_nontrivial [OF ‹finite
ℳ›])
qed
also have ... ≤ ?rhs
proof (rule subadditive_content_division)
  show extend ‘ ℳ division_of ∪ (extend ‘ ℳ)
    using int_extend_disjoint by (auto simp: division_of_def ‹finite ℳ› extend
extend_cbox)
  show ∪ (extend ‘ ℳ) ⊆ cbox a b
    using extend by fastforce
qed
finally show ?thesis .
qed

```

proposition *sum_content_area_over_thin_division:*

```

assumes div: ℳ division_of S and S: S ⊆ cbox a b and i: i ∈ Basis
  and a · i ≤ c c ≤ b · i
  and nonmt: ∧K. K ∈ ℳ ⇒ K ∩ {x. x · i = c} ≠ {}
shows (b · i - a · i) * (∑ K∈ℳ. content K / (interval_upperbound K · i -
interval_lowerbound K · i))
  ≤ 2 * content(cbox a b)
proof (cases content(cbox a b) = 0)
  case True
  have (∑ K∈ℳ. content K / (interval_upperbound K · i - interval_lowerbound
K · i)) = 0
    using S div by (force intro!: sum.neutral content_0_subset [OF True])
  then show ?thesis
    by (auto simp: True)
next
  case False

```

```

then have content(cbox a b) > 0
  using zero_less_measure_iff by blast
then have a · i < b · i if i ∈ Basis for i
  using content_pos_lt_eq that by blast
have finite  $\mathcal{D}$ 
  using div by blast
define Dlec where Dlec ≡ {L ∈ (λL. L ∩ {x. x · i ≤ c}) ‘ $\mathcal{D}$ . content L ≠ 0}
define Dgec where Dgec ≡ {L ∈ (λL. L ∩ {x. x · i ≥ c}) ‘ $\mathcal{D}$ . content L ≠ 0}
define a' where a' ≡ (∑ j ∈ Basis. (if j = i then c else a · j) *R j)
define b' where b' ≡ (∑ j ∈ Basis. (if j = i then c else b · j) *R j)
define interv_diff where interv_diff ≡ λK. λi::'a. interval_upperbound K · i
– interval_lowerbound K · i
have Dlec_cbox: ∧K. K ∈ Dlec ⇒ ∃ a b. K = cbox a b
  using interval_split [OF i] div by (fastforce simp: Dlec_def division_of_def)
then have lec_is_cbox: [content (L ∩ {x. x · i ≤ c}) ≠ 0; L ∈  $\mathcal{D}$ ] ⇒ ∃ a b.
L ∩ {x. x · i ≤ c} = cbox a b for L
  using Dlec_def by blast
have Dgec_cbox: ∧K. K ∈ Dgec ⇒ ∃ a b. K = cbox a b
  using interval_split [OF i] div by (fastforce simp: Dgec_def division_of_def)
then have gec_is_cbox: [content (L ∩ {x. x · i ≥ c}) ≠ 0; L ∈  $\mathcal{D}$ ] ⇒ ∃ a b.
L ∩ {x. x · i ≥ c} = cbox a b for L
  using Dgec_def by blast

have zero_left: ∧x y. [x ∈  $\mathcal{D}$ ; y ∈  $\mathcal{D}$ ; x ≠ y; x ∩ {x. x · i ≤ c} = y ∩ {x. x · i
≤ c}]
  ⇒ content (y ∩ {x. x · i ≤ c}) = 0
  by (metis division_split_left_inj [OF div] lec_is_cbox content_eq_0_interior)
have zero_right: ∧x y. [x ∈  $\mathcal{D}$ ; y ∈  $\mathcal{D}$ ; x ≠ y; x ∩ {x. c ≤ x · i} = y ∩ {x. c
≤ x · i}]
  ⇒ content (y ∩ {x. c ≤ x · i}) = 0
  by (metis division_split_right_inj [OF div] gec_is_cbox content_eq_0_interior)

have (b' · i – a · i) * (∑ K ∈ Dlec. content K / interv_diff K i) ≤ content(cbox
a b')
  unfolding interv_diff_def
proof (rule content_division_lemma1)
  show Dlec division_of ∪ Dlec
  unfolding division_of_def
proof (intro conjI ballI Dlec_cbox)
  show ∧K1 K2. [K1 ∈ Dlec; K2 ∈ Dlec] ⇒ K1 ≠ K2 → interior K1 ∩
interior K2 = {}
  by (clarsimp simp: Dlec_def) (use div in auto)
qed (use ⟨finite  $\mathcal{D}$ ⟩ Dlec_def in auto)
show ∪ Dlec ⊆ cbox a b'
  using Dlec_def div S by (auto simp: b'_def division_of_def mem_box)
show (∀ K ∈ Dlec. K ∩ {x. x · i = a · i} ≠ {}) ∨ (∀ K ∈ Dlec. K ∩ {x. x · i =
b' · i} ≠ {})
  using nonmt by (fastforce simp: Dlec_def b'_def i)
qed (use i Dlec_def in auto)

```


moreover
have $(\sum K \in D_{lec}. \text{content } K / (\text{interv_diff } K \ i)) = (\sum K \in (\lambda K. K \cap \{x. x \cdot i \leq c\}) \text{ ' } \mathcal{D}. \text{content } K / \text{interv_diff } K \ i)$
unfolding *Dlec_def* **using** $\langle \text{finite } \mathcal{D} \rangle$ **by** (*auto simp: sum.mono_neutral_left*)
moreover have ... =
 $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K)$
by (*simp add: zero_left sum.reindex_nontrivial [OF \langle finite \mathcal{D} \rangle]*)
moreover have $(b' \cdot i - a \cdot i) = (c - a \cdot i)$
by (*simp add: b'_def i*)
ultimately
have *lec*: $(c - a \cdot i) * (\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K)$
 $\leq \text{content}(cbox \ a \ b')$
by *simp*

have $(b \cdot i - a' \cdot i) * (\sum K \in D_{gec}. \text{content } K / (\text{interv_diff } K \ i)) \leq \text{content}(cbox \ a' \ b)$
unfolding *interv_diff_def*
proof (*rule content_division_lemma1*)
show *Dgec* *division_of* $\bigcup D_{gec}$
unfolding *division_of_def*
proof (*intro conjI ballI Dgec_cbox*)
show $\bigwedge K1 \ K2. \llbracket K1 \in D_{gec}; K2 \in D_{gec} \rrbracket \implies K1 \neq K2 \longrightarrow \text{interior } K1 \cap \text{interior } K2 = \{\}$
by (*clarsimp simp: Dgec_def*) (*use div in auto*)
qed (*use \langle finite \mathcal{D} \rangle Dgec_def in auto*)
show $\bigcup D_{gec} \subseteq cbox \ a' \ b$
using *Dgec_def div S* **by** (*auto simp: a'_def division_of_def mem_box*)
show $(\forall K \in D_{gec}. K \cap \{x. x \cdot i = a' \cdot i\} \neq \{\}) \vee (\forall K \in D_{gec}. K \cap \{x. x \cdot i = b \cdot i\} \neq \{\})$
using *nonmt* **by** (*fastforce simp: Dgec_def a'_def i*)
qed (*use i Dgec_def in auto*)

moreover
have $(\sum K \in D_{gec}. \text{content } K / (\text{interv_diff } K \ i)) = (\sum K \in (\lambda K. K \cap \{x. c \leq x \cdot i\}) \text{ ' } \mathcal{D}. \text{content } K / \text{interv_diff } K \ i)$
unfolding *Dgec_def* **using** $\langle \text{finite } \mathcal{D} \rangle$ **by** (*auto simp: sum.mono_neutral_left*)
moreover have ... =
 $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$
by (*simp add: zero_right sum.reindex_nontrivial [OF \langle finite \mathcal{D} \rangle]*)
moreover have $(b \cdot i - a' \cdot i) = (b \cdot i - c)$
by (*simp add: a'_def i*)
ultimately
have *gec*: $(b \cdot i - c) * (\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv_diff } K \ i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$
 $\leq \text{content}(cbox \ a' \ b)$
by *simp*

```

show ?thesis
proof (cases c = a · i ∨ c = b · i)
  case True
  then show ?thesis
  proof
    assume c: c = a · i
    moreover
      have (∑ j∈Basis. (if j = i then a · i else a · j) *R j) = a
        using euclidean_representation [of a] sum.cong [OF refl, of Basis λi. (a ·
i) *R i] by presburger
      ultimately have a' = a
        by (simp add: i a'_def cong: if_cong)
      then have content (cbox a' b) ≤ 2 * content (cbox a b) by simp
    moreover
      have eq: (∑ K∈D. content (K ∩ {x. a · i ≤ x · i}) / interv_diff (K ∩ {x.
a · i ≤ x · i}) i)
        = (∑ K∈D. content K / interv_diff K i)
        (is sum ?f _ = sum ?g _)
      proof (rule sum.cong [OF refl])
      fix K assume K ∈ D
      then have a · i ≤ x · i if x ∈ K for x
        by (metis S UnionI div division_ofD(6) i mem_box(2) subsetCE that)
      then have K ∩ {x. a · i ≤ x · i} = K
        by blast
      then show ?f K = ?g K
        by simp
    qed
  ultimately show ?thesis
    using gec c eq interv_diff_def by auto
next
  assume c: c = b · i
  moreover have (∑ j∈Basis. (if j = i then b · i else b · j) *R j) = b
    using euclidean_representation [of b] sum.cong [OF refl, of Basis λi. (b ·
i) *R i] by presburger
  ultimately have b' = b
    by (simp add: i b'_def cong: if_cong)
  then have content (cbox a b') ≤ 2 * content (cbox a b) by simp
  moreover
    have eq: (∑ K∈D. content (K ∩ {x. x · i ≤ b · i}) / interv_diff (K ∩ {x. x
· i ≤ b · i}) i)
      = (∑ K∈D. content K / interv_diff K i)
      (is sum ?f _ = sum ?g _)
    proof (rule sum.cong [OF refl])
    fix K assume K ∈ D
    then have x · i ≤ b · i if x ∈ K for x
      by (metis S UnionI div division_ofD(6) i mem_box(2) subsetCE that)
    then have K ∩ {x. x · i ≤ b · i} = K
      by blast

```

```

    then show ?f K = ?g K
      by simp
  qed
  ultimately show ?thesis
    using lec c eq interv_diff_def by auto
  qed
next
case False
  have prod_if:  $(\prod k \in \text{Basis} \cap - \{i\}. f k) = (\prod k \in \text{Basis}. f k) / f i$  if  $f i \neq 0$ 
  for f
  proof -
    have  $f i * \prod f (\text{Basis} \cap - \{i\}) = \prod f \text{Basis}$ 
      using that mk_disjoint_insert [OF i]
    by (metis Int_insert_left_if0 finite_Basis finite_insert le_iff_inf order_refl
prod.insert subset_Compl_singleton)
    then show ?thesis
      by (metis nonzero_mult_div_cancel_left that)
  qed
  have abc:  $a \cdot i < c < b \cdot i$ 
    using False assms by auto
  then have  $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K)$ 
     $\leq \text{content}(c \text{ box } a b) / (c - a \cdot i)$ 
     $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$ 
     $\leq \text{content}(c \text{ box } a' b) / (b \cdot i - c)$ 
    using lec gec by (simp_all add: field_split_simps)
  moreover
  have  $(\sum K \in \mathcal{D}. \text{content } K / (\text{interv\_diff } K i))$ 
     $\leq (\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K) +$ 
     $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K)$ 
    (is ?lhs  $\leq$  ?rhs)
  proof -
    have ?lhs  $\leq$ 
       $(\sum K \in \mathcal{D}. ((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \leq c\}))) K) +$ 
       $((\lambda K. \text{content } K / (\text{interv\_diff } K i)) \circ ((\lambda K. K \cap \{x. x \cdot i \geq c\}))) K$ 
      (is sum ?f  $\leq$  sum ?g  $\_$ )
    proof (rule sum_mono)
      fix K assume  $K \in \mathcal{D}$ 
      then obtain u v where  $uv: K = c \text{ box } u v$ 
        using div by blast
      obtain u' v' where  $uv': c \text{ box } u v \cap \{x. x \cdot i \leq c\} = c \text{ box } u v'$ 
         $c \text{ box } u v \cap \{x. c \leq x \cdot i\} = c \text{ box } u' v$ 
         $\bigwedge k. k \in \text{Basis} \implies u' \cdot k = (\text{if } k = i \text{ then } \max(u \cdot i) c$ 
        else  $u \cdot k)$ 
    end
  end

```

```

       $\bigwedge k. k \in \text{Basis} \implies v' \cdot k = (\text{if } k = i \text{ then } \min(v \cdot i) \text{ c}$ 
else  $v \cdot k)$ 
    using  $i$  by (auto simp: interval_split)
    have *:  $\llbracket \text{content } (\text{cbox } u \ v') = 0; \text{content } (\text{cbox } u' \ v) = 0 \rrbracket \implies \text{content } (\text{cbox}$ 
 $u \ v) = 0$ 
       $\text{content } (\text{cbox } u' \ v) \neq 0 \implies \text{content } (\text{cbox } u \ v) \neq 0$ 
       $\text{content } (\text{cbox } u \ v') \neq 0 \implies \text{content } (\text{cbox } u \ v) \neq 0$ 
    using  $i \ uv \ uv'$  by (auto simp: content_eq_0 le_max_iff_disj min_le_iff_disj
split: if_split_asm intro: order_trans)
    have uniq:  $\bigwedge j. \llbracket j \in \text{Basis}; \neg u \cdot j \leq v \cdot j \rrbracket \implies j = i$ 
      by (metis  $\langle K \in \mathcal{D} \rangle$  box_ne_empty(1) div_division_of_def uv)
    show  $?f \ K \leq ?g \ K$ 
      using  $i \ uv \ uv'$  by (auto simp add: interv_diff_def lemma0 dest: uniq *
intro!: prod_nonneg)
    qed
    also have  $\dots = ?rhs$ 
      by (simp add: sum.distrib)
    finally show  $?thesis$  .
  qed
  moreover have  $\text{content } (\text{cbox } a \ b') / (c - a \cdot i) = \text{content } (\text{cbox } a \ b) / (b \cdot i$ 
 $- a \cdot i)$ 
    using  $i \ abc$ 
    apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq in-
ner_diff)
    apply (auto simp: if_distrib if_distrib [of  $\lambda f. f \ x$  for  $x$ ] prod.If_cases [of
Basis  $\lambda x. x = i$ , simplified] prod_if field_simps)
    done
  moreover have  $\text{content } (\text{cbox } a' \ b) / (b \cdot i - c) = \text{content } (\text{cbox } a \ b) / (b \cdot i$ 
 $- a \cdot i)$ 
    using  $i \ abc$ 
    apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq in-
ner_diff)
    apply (auto simp: if_distrib prod.If_cases [of Basis  $\lambda x. x = i$ , simplified]
prod_if field_simps)
    done
  ultimately
  have  $(\sum K \in \mathcal{D}. \text{content } K / (\text{interv\_diff } K \ i)) \leq 2 * \text{content } (\text{cbox } a \ b) / (b \cdot$ 
 $i - a \cdot i)$ 
    by linarith
  then show  $?thesis$ 
    using  $abc \ \text{interv\_diff\_def}$  by (simp add: field_split_simps)
  qed
  qed

```

proposition *bounded_equiintegral_over_thin_tagged_partial_division:*

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$

assumes $F: F \text{ equiintegrable_on } \text{cbox } a \ b$ **and** $f: f \in F$ **and** $0 < \varepsilon$

and $\text{norm}_f: \bigwedge h \ x. \llbracket h \in F; x \in \text{cbox } a \ b \rrbracket \implies \text{norm}(h \ x) \leq \text{norm}(f \ x)$

```

obtains  $\gamma$  where gauge  $\gamma$ 
   $\wedge c \ i \ S \ h. \llbracket c \in \text{cbox } a \ b; \ i \in \text{Basis}; \ S \ \text{tagged\_partial\_division\_of } \text{cbox } a \ b;$ 
 $\neq \{\}$ 
   $\gamma \ \text{fine } S; \ h \in F; \ \wedge x \ K. \ (x, K) \in S \implies (K \cap \{x. \ x \cdot i = c \cdot i\})$ 
 $\implies (\sum (x, K) \in S. \ \text{norm } (\text{integral } K \ h)) < \varepsilon$ 
proof (cases content(cbox a b) = 0)
  case True
  show ?thesis
  proof
    show gauge  $(\lambda x. \ \text{ball } x \ 1)$ 
      by (simp add: gauge_trivial)
    show  $(\sum (x, K) \in S. \ \text{norm } (\text{integral } K \ h)) < \varepsilon$ 
      if S tagged_partial_division_of cbox a b  $(\lambda x. \ \text{ball } x \ 1)$  fine S for S and
h:: 'a  $\Rightarrow$  'b
    proof -
      have  $(\sum (x, K) \in S. \ \text{norm } (\text{integral } K \ h)) = 0$ 
        using that True content_0_subset
        by (fastforce simp: tagged_partial_division_of_def intro: sum.neutral)
      with  $\langle 0 < \varepsilon \rangle$  show ?thesis
        by simp
    qed
  qed
next
  case False
  then have contab_gt0: content(cbox a b) > 0
    by (simp add: zero_less_measure_iff)
  then have a_less_b:  $\wedge i. \ i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    by (auto simp: content_pos_lt_eq)
  obtain  $\gamma_0$  where gauge  $\gamma_0$ 
    and  $\gamma_0: \wedge S \ h. \llbracket S \ \text{tagged\_partial\_division\_of } \text{cbox } a \ b; \ \gamma_0 \ \text{fine } S; \ h \in F \rrbracket$ 
 $\implies (\sum (x, K) \in S. \ \text{norm } (\text{content } K \ *_{\mathbb{R}} \ h \ x - \text{integral } K$ 
h)) <  $\varepsilon/2$ 
  proof -
    obtain  $\gamma$  where gauge  $\gamma$ 
      and  $\gamma: \wedge f \ \mathcal{D}. \llbracket f \in F; \ \mathcal{D} \ \text{tagged\_division\_of } \text{cbox } a \ b; \ \gamma \ \text{fine } \mathcal{D} \rrbracket$ 
 $\implies \text{norm } ((\sum (x, K) \in \mathcal{D}. \ \text{content } K \ *_{\mathbb{R}} \ f \ x) - \text{integral}$ 
(cbox a b) f)
 $< \varepsilon / (5 * (\text{Suc } \text{DIM}('b)))$ 
    proof -
      have e5:  $\varepsilon / (5 * (\text{Suc } \text{DIM}('b))) > 0$ 
        using  $\langle \varepsilon > 0 \rangle$  by auto
      then show ?thesis
        using F that by (auto simp: equiintegrable_on_def)
    qed
  show ?thesis
  proof
    show gauge  $\gamma$ 
      by (rule  $\langle$ gauge  $\gamma$  $\rangle$ )

```

```

show  $(\sum (x,K) \in S. \text{norm} (\text{content } K *_R h x - \text{integral } K h)) < \varepsilon/2$ 
  if  $S$  tagged_partial_division_of_cbox  $a b \gamma$  fine  $S h \in F$  for  $S h$ 
proof –
  have  $(\sum (x,K) \in S. \text{norm} (\text{content } K *_R h x - \text{integral } K h)) \leq 2 * \text{real}$ 
   $\text{DIM}(b) * (\varepsilon/(5 * \text{Suc } \text{DIM}(b)))$ 
  proof (rule Henstock_lemma_part2 [of h a b])
    show  $h$  integrable_on_cbox  $a b$ 
    using that F equiintegrable_on_def by metis
    show gauge  $\gamma$ 
    by (rule gauge  $\gamma$ )
  qed (use that  $\langle \varepsilon > 0 \rangle \gamma$  in auto)
  also have  $\dots < \varepsilon/2$ 
    using  $\langle \varepsilon > 0 \rangle$  by (simp add: divide_simps)
  finally show ?thesis .
qed
qed
define  $\gamma$  where  $\gamma \equiv \lambda x. \gamma 0 x \cap$ 
   $\text{ball } x ((\varepsilon/8 / (\text{norm}(f x) + 1)) * (\text{INF } m \in \text{Basis}. b \cdot m - a$ 
   $\cdot m) / \text{content}(cbox a b))$ 
  define interv_diff where interv_diff  $\equiv \lambda K. \lambda i::'a. \text{interval\_upperbound } K \cdot i$ 
   $- \text{interval\_lowerbound } K \cdot i$ 
  have  $8 * \text{content} (cbox a b) + \text{norm} (f x) * (8 * \text{content} (cbox a b)) > 0$  for  $x$ 
  by (metis add.right_neutral add_pos_pos contab_gt0 mult_pos_pos mult_zero_left
  norm_eq_zero zero_less_norm_iff zero_less_numeral)
  then have gauge  $(\lambda x. \text{ball } x$ 
   $(\varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / ((8 * \text{norm} (f x) + 8) * \text{content} (cbox a b))))$ 
  using  $\langle 0 < \text{content} (cbox a b) \rangle \langle 0 < \varepsilon \rangle$  a_less_b
  by (auto simp add: gauge_def field_split_simps add_nonneg_eq_0_iff finite_less_Inf_iff)
  then have gauge  $\gamma$ 
  unfolding  $\gamma\_def$  using  $\langle \text{gauge } \gamma 0 \rangle$  gauge_Int by auto
moreover
  have  $(\sum (x,K) \in S. \text{norm} (\text{integral } K h)) < \varepsilon$ 
  if  $c \in cbox a b$   $i \in \text{Basis}$  and  $S: S$  tagged_partial_division_of_cbox  $a b$ 
  and  $\gamma$  fine  $S h \in F$  and ne:  $\bigwedge x K. (x,K) \in S \implies K \cap \{x. x \cdot i = c \cdot i\}$ 
 $\neq \{\}$  for  $c i S h$ 
proof –
  have  $cbox c b \subseteq cbox a b$ 
  by (meson mem_box(2) order_refl subset_box(1) that(1))
  have finite  $S$ 
  using  $S$  unfolding tagged_partial_division_of_def by blast
  have  $\gamma 0$  fine  $S$  and fineS:
   $(\lambda x. \text{ball } x (\varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / ((8 * \text{norm} (f x) + 8) * \text{content} (cbox a b))))$  fine  $S$ 
  using  $\langle \gamma$  fine  $S \rangle$  by (auto simp: gamma_def fine_Int)
  then have  $(\sum (x,K) \in S. \text{norm} (\text{content } K *_R h x - \text{integral } K h)) < \varepsilon/2$ 
  by (intro  $\gamma 0$  that fineS)

```

```

moreover have  $(\sum (x,K) \in S. \text{norm} (\text{integral } K \ h) - \text{norm} (\text{content } K \ *_R \ h \ x - \text{integral } K \ h)) \leq \varepsilon/2$ 
proof -
  have  $(\sum (x,K) \in S. \text{norm} (\text{integral } K \ h) - \text{norm} (\text{content } K \ *_R \ h \ x - \text{integral } K \ h))$ 
     $\leq (\sum (x,K) \in S. \text{norm} (\text{content } K \ *_R \ h \ x))$ 
  proof (clarify intro!: sum_mono)
    fix  $x \ K$ 
    assume  $xK: (x,K) \in S$ 
    have  $\text{norm} (\text{integral } K \ h) - \text{norm} (\text{content } K \ *_R \ h \ x - \text{integral } K \ h) \leq$ 
 $\text{norm} (\text{integral } K \ h - (\text{integral } K \ h - \text{content } K \ *_R \ h \ x))$ 
    by (metis norm_minus_commute norm_triangle_ineq2)
    also have  $\dots \leq \text{norm} (\text{content } K \ *_R \ h \ x)$ 
    by simp
    finally show  $\text{norm} (\text{integral } K \ h) - \text{norm} (\text{content } K \ *_R \ h \ x - \text{integral } K \ h) \leq \text{norm} (\text{content } K \ *_R \ h \ x)$  .
  qed
  also have  $\dots \leq (\sum (x,K) \in S. \varepsilon/4 * (b \cdot i - a \cdot i) / \text{content} (\text{cbox } a \ b) * \text{content } K / \text{interv\_diff } K \ i)$ 
  proof (clarify intro!: sum_mono)
    fix  $x \ K$ 
    assume  $xK: (x,K) \in S$ 
    then have  $x: x \in \text{cbox } a \ b$ 
    using  $S$  unfolding tagged_partial_division_of_def by (meson subset_iff)
    show  $\text{norm} (\text{content } K \ *_R \ h \ x) \leq \varepsilon/4 * (b \cdot i - a \cdot i) / \text{content} (\text{cbox } a \ b) * \text{content } K / \text{interv\_diff } K \ i$ 
    proof (cases content  $K = 0$ )
      case True
        then show ?thesis by simp
      next
        case False
          then have  $Kgt0: \text{content } K > 0$ 
          using zero_less_measure_iff by blast
          moreover
            obtain  $u \ v$  where  $uv: K = \text{cbox } u \ v$ 
            using  $S \ \langle(x,K) \in S\rangle$  unfolding tagged_partial_division_of_def by blast
          then have  $u\_less\_v: \bigwedge i. i \in \text{Basis} \implies u \cdot i < v \cdot i$ 
          using content_pos_lt_eq uv Kgt0 by blast
          then have  $\text{dist\_uv}: \text{dist } u \ v > 0$ 
          using that by auto
          ultimately have  $\text{norm} (h \ x) \leq (\varepsilon * (b \cdot i - a \cdot i)) / (4 * \text{content} (\text{cbox } a \ b) * \text{interv\_diff } K \ i)$ 
          proof -
            have  $\text{dist } x \ u < \varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / (4 * (\text{norm} (f \ x) + 1) * \text{content} (\text{cbox } a \ b)) / 2$ 
             $\text{dist } x \ v < \varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / (4 * (\text{norm} (f \ x) + 1) * \text{content} (\text{cbox } a \ b)) / 2$ 
            using fineS u_less_v uv xK
  
```

```

    by (force simp: fine_def mem_box field_simps dest!: bspec)+
    moreover have  $\varepsilon * (\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) / (4 * (\text{norm } (f x) + 1) * \text{content } (\text{cbox } a \ b)) / 2$ 
       $\leq \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f x) + 1) * \text{content } (\text{cbox } a \ b))$ 
    / 2
  proof (intro mult_left_mono divide_right_mono)
    show  $(\text{INF } m \in \text{Basis}. b \cdot m - a \cdot m) \leq b \cdot i - a \cdot i$ 
      using  $\langle i \in \text{Basis} \rangle$  by (auto intro!: cInf_le_finite)
    qed (use  $\langle 0 < \varepsilon \rangle$  in auto)
    ultimately
    have  $\text{dist } x \ u < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f x) + 1) * \text{content } (\text{cbox } a \ b)) / 2$ 
       $\text{dist } x \ v < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f x) + 1) * \text{content } (\text{cbox } a \ b)) / 2$ 
    by linarith+
    then have  $\text{dvw: dist } u \ v < \varepsilon * (b \cdot i - a \cdot i) / (4 * (\text{norm } (f x) + 1) * \text{content } (\text{cbox } a \ b))$ 
      using dist_triangle_half_r by blast
    have  $\text{uvi: } |v \cdot i - u \cdot i| \leq \text{norm } (v - u)$ 
    by (metis inner_commute inner_diff_right  $\langle i \in \text{Basis} \rangle$  Basis_le_norm)
    have  $\text{norm } (h x) \leq \text{norm } (f x)$ 
      using  $x$  that by (auto simp: norm_f)
    also have  $\dots < (\text{norm } (f x) + 1)$ 
      by simp
    also have  $\dots < \varepsilon * (b \cdot i - a \cdot i) / \text{dist } u \ v / (4 * \text{content } (\text{cbox } a \ b))$ 
    proof -
      have  $0 < \text{norm } (f x) + 1$ 
      by (simp add: add_commute add_pos_nonneg)
      then show ?thesis
        using  $\text{dvw dist}_uv \text{contab}_gt0$ 
        by (simp only: mult_ac divide_simps) auto
    qed
    also have  $\dots = \varepsilon * (b \cdot i - a \cdot i) / \text{norm } (v - u) / (4 * \text{content } (\text{cbox } a \ b))$ 
      by (simp add: dist_norm norm_minus_commute)
    also have  $\dots \leq \varepsilon * (b \cdot i - a \cdot i) / |v \cdot i - u \cdot i| / (4 * \text{content } (\text{cbox } a \ b))$ 
    proof (intro mult_right_mono divide_left_mono divide_right_mono uvi)
      show  $\text{norm } (v - u) * |v \cdot i - u \cdot i| > 0$ 
        using  $u\_less\_v$  [OF  $\langle i \in \text{Basis} \rangle$ ]
        by (auto simp: less_eq_real_def zero_less_mult_iff that)
      show  $\varepsilon * (b \cdot i - a \cdot i) \geq 0$ 
        using  $a\_less\_b \langle 0 < \varepsilon \rangle \langle i \in \text{Basis} \rangle$  by force
    qed auto
    also have  $\dots = \varepsilon * (b \cdot i - a \cdot i) / (4 * \text{content } (\text{cbox } a \ b) * \text{interv\_diff } K \ i)$ 
      using  $uv$  False that(2)  $u\_less\_v$   $\text{interv\_diff\_def}$  by fastforce
    finally show ?thesis by simp

```



```

    qed
    with Kgt0 have norm (content K *R h x) ≤ content K * ((ε/4 * (b · i
  - a · i) / content (cbox a b)) / interv_diff K i)
      using mult_left_mono by fastforce
    also have ... = ε/4 * (b · i - a · i) / content (cbox a b) * content K /
  interv_diff K i
      by (simp add: field_split_simps)
    finally show ?thesis .
  qed
  qed
  also have ... = (∑ K∈snd ' S. ε/4 * (b · i - a · i) / content (cbox a b) *
  content K / interv_diff K i)
      unfolding interv_diff_def
    apply (rule sum.over_tagged_division_lemma [OF tagged_partial_division_of_Union_self
  [OF S]])
      apply (simp add: box_eq_empty(1) content_eq_0)
    done
  also have ... = ε/2 * ((b · i - a · i) / (2 * content (cbox a b)) * (∑ K∈snd
  ' S. content K / interv_diff K i))
      by (simp add: interv_diff_def sum_distrib_left mult.assoc)
  also have ... ≤ (ε/2) * 1
      proof (rule mult_left_mono)
        have (b · i - a · i) * (∑ K∈snd ' S. content K / interv_diff K i) ≤ 2 *
  content (cbox a b)
            unfolding interv_diff_def
          proof (rule sum_content_area_over_thin_division)
            show snd ' S division_of ⋃(snd ' S)
          by (auto intro: S tagged_partial_division_of_Union_self division_of_tagged_division)
            show ⋃(snd ' S) ⊆ cbox a b
            using S unfolding tagged_partial_division_of_def by force
            show a · i ≤ c · i c · i ≤ b · i
            using mem_box(2) that by blast+
          qed (use that in auto)
        then show (b · i - a · i) / (2 * content (cbox a b)) * (∑ K∈snd ' S.
  content K / interv_diff K i) ≤ 1
            by (simp add: contab_gt0)
          qed (use ‹0 < ε› in auto)
        finally show ?thesis by simp
      qed
  qed
  then have (∑ (x,K) ∈ S. norm (integral K h)) - (∑ (x,K) ∈ S. norm (content
  K *R h x - integral K h)) ≤ ε/2
      by (simp add: Groups_Big.sum_subtractf [symmetric])
    ultimately show (∑ (x,K) ∈ S. norm (integral K h)) < ε
      by linarith
  qed
  ultimately show ?thesis using that by auto
  qed

```

proposition *equiintegrable_halfspace_restrictions_le*:
fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes $F: F$ *equiintegrable_on cbox a b* **and** $f: f \in F$
and $norm_f: \bigwedge h x. \llbracket h \in F; x \in cbox\ a\ b \rrbracket \implies norm(h\ x) \leq norm(f\ x)$
shows $(\bigcup i \in Basis. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i \leq c \text{ then } h\ x \text{ else } 0)\})$
equiintegrable_on cbox a b
proof (*cases content(cbox a b) = 0*)
case *True*
then show *?thesis by simp*
next
case *False*
then have $content(cbox\ a\ b) > 0$
using *zero_less_measure_iff* **by** *blast*
then have $a \cdot i < b \cdot i$ **if** $i \in Basis$ **for** i
using *content_pos_lt_eq* **that** **by** *blast*
have $int_F: f$ *integrable_on cbox a b* **if** $f \in F$ **for** f
using F **that** **by** (*simp add: equiintegrable_on_def*)
let $?CI = \lambda K\ h\ x. content\ K\ *R\ h\ x - integral\ K\ h$
show *?thesis*
unfolding *equiintegrable_on_def*
proof (*intro conjI; clarify*)
show $int_lec: \llbracket i \in Basis; h \in F \rrbracket \implies (\lambda x. \text{if } x \cdot i \leq c \text{ then } h\ x \text{ else } 0)$
integrable_on cbox a b for i c h
using *integrable_restrict_Int* [of $\{x. x \cdot i \leq c\}$ h]
by (*simp add: inf_commute int_F integrable_split(1)*)
show $\exists \gamma. gauge\ \gamma \wedge$
 $(\forall f\ T. f \in (\bigcup i \in Basis. \bigcup c. \bigcup h \in F. \{ \lambda x. \text{if } x \cdot i \leq c \text{ then } h\ x \text{ else } 0 \}))$
 \wedge
 $T\ tagged_division_of\ cbox\ a\ b \wedge \gamma\ fine\ T \longrightarrow$
 $norm\ ((\sum (x,K) \in T. content\ K\ *R\ f\ x) - integral\ (cbox\ a\ b)\ f)$
 $< \varepsilon)$
if $\varepsilon > 0$ **for** ε
proof –
obtain $\gamma 0$ **where** *gauge $\gamma 0$ and $\gamma 0$:*
 $\bigwedge c\ i\ S\ h. \llbracket c \in cbox\ a\ b; i \in Basis; S\ tagged_partial_division_of\ cbox\ a\ b;$
 $\gamma 0\ fine\ S; h \in F; \bigwedge x\ K. (x,K) \in S \implies (K \cap \{x. x \cdot i = c \cdot$
 $i\} \neq \{\}) \rrbracket$
 $\implies (\sum (x,K) \in S. norm\ (integral\ K\ h)) < \varepsilon/12$
proof (*rule bounded_equiintegral_over_thin_tagged_partial_division [OF F*
 $f, of \langle \varepsilon/12 \rangle$ *]*)
show $\bigwedge h\ x. \llbracket h \in F; x \in cbox\ a\ b \rrbracket \implies norm\ (h\ x) \leq norm\ (f\ x)$
by (*auto simp: norm_f*)
qed (*use $\langle \varepsilon > 0 \rangle$ in auto*)
obtain $\gamma 1$ **where** *gauge $\gamma 1$*
and $\gamma 1: \bigwedge h\ T. \llbracket h \in F; T\ tagged_division_of\ cbox\ a\ b; \gamma 1\ fine\ T \rrbracket$
 $\implies norm\ ((\sum (x,K) \in T. content\ K\ *R\ h\ x) - integral$
 $(cbox\ a\ b)\ h)$
 $< \varepsilon / (\gamma * (Suc\ DIM('b)))$

```

proof –
  have e5:  $\varepsilon / (\gamma * (\text{Suc } \text{DIM}('b))) > 0$ 
    using  $\langle \varepsilon > 0 \rangle$  by auto
  then show ?thesis
    using F that by (auto simp: equiintegrable_on_def)
qed
have h_less3:  $(\sum (x,K) \in T. \text{norm } (?CI\ K\ h\ x)) < \varepsilon / 3$ 
  if T tagged_partial_division_of_cbox_a_b  $\gamma 1$  fine T h  $\in F$  for T h
proof –
  have  $(\sum (x,K) \in T. \text{norm } (?CI\ K\ h\ x)) \leq 2 * \text{real } \text{DIM}('b) * (\varepsilon / (\gamma * \text{Suc } \text{DIM}('b)))$ 
    proof (rule Henstock_lemma_part2 [of h a b])
      show h integrable_on_cbox_a_b
        using that F equiintegrable_on_def by metis
      qed (use that  $\langle \varepsilon > 0 \rangle$   $\langle \text{gauge } \gamma 1 \rangle$   $\gamma 1$  in auto)
    also have ...  $< \varepsilon / 3$ 
      using  $\langle \varepsilon > 0 \rangle$  by (simp add: divide_simps)
    finally show ?thesis .
qed
have *:  $\text{norm } ((\sum (x,K) \in T. \text{content } K *_R f\ x) - \text{integral } (\text{cbox } a\ b)\ f) < \varepsilon$ 
  if f: f =  $(\lambda x. \text{if } x \cdot i \leq c \text{ then } h\ x \text{ else } 0)$ 
  and T: T tagged_division_of_cbox_a_b
  and fine:  $(\lambda x. \gamma 0\ x \cap \gamma 1\ x)$  fine T and i  $\in \text{Basis}$  h  $\in F$  for f T i c h
proof (cases  $a \cdot i \leq c \wedge c \leq b \cdot i$ )
  case True
  have finite T
    using T by blast
  define T' where T'  $\equiv \{(x,K) \in T. K \cap \{x. x \cdot i \leq c\} \neq \{\}\}$ 
  then have  $T' \subseteq T$ 
    by auto
  then have finite T'
    using  $\langle \text{finite } T \rangle$  infinite_super by blast
  have T'_tagged: T' tagged_partial_division_of_cbox_a_b
by (meson T  $\langle T' \subseteq T \rangle$  tagged_division_of_def tagged_partial_division_subset)
  have fine':  $\gamma 0$  fine T'  $\gamma 1$  fine T'
    using  $\langle T' \subseteq T \rangle$  fine_Int fine_subset fine by blast
  have int_KK':  $(\sum (x,K) \in T. \text{integral } K\ f) = (\sum (x,K) \in T'. \text{integral } K\ f)$ 
  proof (rule sum.mono_neutral_right [OF  $\langle \text{finite } T \rangle$   $\langle T' \subseteq T \rangle$ ])
    show  $\forall i \in T - T'. (\text{case } i \text{ of } (x, K) \Rightarrow \text{integral } K\ f) = 0$ 
      using f  $\langle \text{finite } T \rangle$   $\langle T' \subseteq T \rangle$  integral_restrict_Int [of  $\_ \{x. x \cdot i \leq c\}$  h]
      by (auto simp: T'_def Int_commute)
    qed
  have  $(\sum (x,K) \in T. \text{content } K *_R f\ x) = (\sum (x,K) \in T'. \text{content } K *_R f\ x)$ 
x)
proof (rule sum.mono_neutral_right [OF  $\langle \text{finite } T \rangle$   $\langle T' \subseteq T \rangle$ ])
  show  $\forall i \in T - T'. (\text{case } i \text{ of } (x, K) \Rightarrow \text{content } K *_R f\ x) = 0$ 
    using T f  $\langle \text{finite } T \rangle$   $\langle T' \subseteq T \rangle$  by (force simp: T'_def)
qed
moreover have  $\text{norm } ((\sum (x,K) \in T'. \text{content } K *_R f\ x) - \text{integral } (\text{cbox } a\ b)\ f) < \varepsilon$ 

```

```

a b) f) < ε
proof -
  have *: norm y < ε if norm x < ε/3 norm(x - y) ≤ 2 * ε/3 for x y::'b
  proof -
    have norm y ≤ norm x + norm(x - y)
    by (metis norm_minus_commute norm_triangle_sub)
    also have ... < ε/3 + 2*ε/3
    using that by linarith
    also have ... = ε
    by simp
    finally show ?thesis .
qed
have norm (∑ (x,K) ∈ T'. ?CI K h x)
  ≤ (∑ (x,K) ∈ T'. norm (?CI K h x))
  by (simp add: norm_sum_split_def)
also have ... < ε/3
  by (intro h_less3 T'_tagged fine' that)
finally have norm (∑ (x,K) ∈ T'. ?CI K h x) < ε/3 .
moreover have integral (cbox a b) f = (∑ (x,K) ∈ T. integral K f)
using int_lec that by (auto simp: integral_combine_tagged_division_topdown)
moreover have norm (∑ (x,K) ∈ T'. ?CI K h x - ?CI K f x)
  ≤ 2*ε/3
proof -
  define T'' where T'' ≡ {(x,K) ∈ T'. ¬ (K ⊆ {x. x · i ≤ c})}
  then have T'' ⊆ T'
  by auto
  then have finite T''
  using ⟨finite T'⟩ infinite_super by blast
  have T''_tagged: T'' tagged_partial_division_of cbox a b
  using T'_tagged ⟨T'' ⊆ T'⟩ tagged_partial_division_subset by blast
  have fine'': γ0 fine T'' γ1 fine T''
  using ⟨T'' ⊆ T'⟩ fine' by (blast intro: fine_subset)+
  have (∑ (x,K) ∈ T'. ?CI K h x - ?CI K f x)
    = (∑ (x,K) ∈ T''. ?CI K h x - ?CI K f x)
  proof (clarify intro!: sum_mono_neutral_right [OF ⟨finite T'⟩ ⟨T'' ⊆
T'⟩])
    fix x K
    assume (x,K) ∈ T' (x,K) ∉ T''
    then have x ∈ K x · i ≤ c {x. x · i ≤ c} ∩ K = K
    using T''_def T'_tagged tagged_partial_division_of_def by blast+
    then show ?CI K h x - ?CI K f x = 0
    using integral_restrict_Int [of _ {x. x · i ≤ c} h] by (auto simp: f)
  qed
moreover have norm (∑ (x,K) ∈ T''. ?CI K h x - ?CI K f x) ≤ 2*ε/3
proof -
  define A where A ≡ {(x,K) ∈ T''. x · i ≤ c}
  define B where B ≡ {(x,K) ∈ T''. x · i > c}
  then have A ⊆ T'' B ⊆ T'' and disj: A ∩ B = {} and T''_eq: T''
= A ∪ B

```

```

    by (auto simp: A_def B_def)
  then have finite A finite B
    using ‹finite T''› by (auto intro: finite_subset)
  have A_tagged: A tagged_partial_division_of_cbox a b
    using T''_tagged ‹A ⊆ T''› tagged_partial_division_subset by blast
  have fineA:  $\gamma 0$  fine A  $\gamma 1$  fine A
    using ‹A ⊆ T''› fine'' by (blast intro: fine_subset)+
  have B_tagged: B tagged_partial_division_of_cbox a b
    using T''_tagged ‹B ⊆ T''› tagged_partial_division_subset by blast
  have fineB:  $\gamma 0$  fine B  $\gamma 1$  fine B
    using ‹B ⊆ T''› fine'' by (blast intro: fine_subset)+
  have norm  $(\sum (x,K) \in T''. ?CI K h x - ?CI K f x)$ 
     $\leq (\sum (x,K) \in T''. norm (?CI K h x - ?CI K f x))$ 
    by (simp add: norm_sum_split_def)
  also have ... =  $(\sum (x,K) \in A. norm (?CI K h x - ?CI K f x)) +$ 
     $(\sum (x,K) \in B. norm (?CI K h x - ?CI K f x))$ 
    by (simp add: sum.union_disjoint T''_eq_disj ‹finite A› ‹finite B›)
  also have ... =  $(\sum (x,K) \in A. norm (integral K h - integral K f)) +$ 
     $(\sum (x,K) \in B. norm (?CI K h x + integral K f))$ 
  by (auto simp: A_def B_def f_norm_minus_commute intro!: sum.cong
  arg_cong2 [where f= (+)])
  also have ...  $\leq (\sum (x,K) \in A. norm (integral K h)) +$ 
     $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. x \cdot i \leq c\})) ' A. norm$ 
    (integral K h))
    +  $(\sum (x,K) \in B. norm (?CI K h x)) +$ 
     $(\sum (x,K) \in B. norm (integral K h)) +$ 
     $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\})) ' B. norm$ 
    (integral K h))
  proof (rule add_mono)
    show  $(\sum (x,K) \in A. norm (integral K h - integral K f))$ 
       $\leq (\sum (x,K) \in A. norm (integral K h)) +$ 
       $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. x \cdot i \leq c\})) ' A. norm$ 
      (integral K h))
    proof (subst sum.reindex_nontrivial [OF ‹finite A›], clarsimp)
      fix x K L
      assume  $(x,K) \in A$   $(x,L) \in A$ 
      and int_ne0:  $integral (L \cap \{x. x \cdot i \leq c\}) h \neq 0$ 
      and eq:  $K \cap \{x. x \cdot i \leq c\} = L \cap \{x. x \cdot i \leq c\}$ 
      have False if  $K \neq L$ 
      proof -
        obtain u v where  $uv: L = cbox u v$ 
          using T'_tagged ‹(x, L) ∈ A› ‹A ⊆ T''› ‹T'' ⊆ T'› by (blast
          dest: tagged_partial_division_ofD)
        have interior  $(K \cap \{x. x \cdot i \leq c\}) = \{\}$ 
          proof (rule tagged_division_split_left_inj [OF _ ‹(x,K) ∈ A›
          ‹(x,L) ∈ A›])
            show A tagged_division_of  $\bigcup (snd ' A)$ 
              using A_tagged tagged_partial_division_of_Union_self by
              auto

```

```

      show  $K \cap \{x. x \cdot i \leq c\} = L \cap \{x. x \cdot i \leq c\}$ 
      using  $eq \langle i \in Basis \rangle$  by auto
    qed (use that in auto)
  then show False
  using interval_split [OF  $\langle i \in Basis \rangle$ ] int_ne0 content_eq_0_interior
  eq uv by fastforce
  qed
  then show  $K = L$  by blast
next
show  $(\sum (x,K) \in A. norm (integral K h - integral K f))$ 
   $\leq (\sum (x,K) \in A. norm (integral K h)) +$ 
   $sum ((\lambda(x,K). norm (integral K h)) \circ (\lambda(x,K). (x,K \cap \{x.$ 
 $x \cdot i \leq c\}))) A$ 
  using integral_restrict_Int [of  $\_ \{x. x \cdot i \leq c\} h] f$ 
  by (auto simp: Int_commute A_def [symmetric] sum.distrib
  [symmetric] intro!: sum_mono norm_triangle_ineq4)
  qed
next
show  $(\sum (x,K) \in B. norm (?CI K h x + integral K f))$ 
   $\leq (\sum (x,K) \in B. norm (?CI K h x)) + (\sum (x,K) \in B. norm (integral$ 
 $K h)) +$ 
   $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\})) \text{ ' } B. norm (integral$ 
 $K h))$ 
  proof (subst sum.reindex_nontrivial [OF  $\langle finite B \rangle$ ], clarsimp)
  fix  $x K L$ 
  assume  $(x,K) \in B (x,L) \in B$ 
  and int_ne0:  $integral (L \cap \{x. c \leq x \cdot i\}) h \neq 0$ 
  and eq:  $K \cap \{x. c \leq x \cdot i\} = L \cap \{x. c \leq x \cdot i\}$ 
  have False if  $K \neq L$ 
  proof -
  obtain  $u v$  where  $uv: L = cbox u v$ 
  using  $T'_{tagged} \langle (x, L) \in B \rangle \langle B \subseteq T'' \rangle \langle T'' \subseteq T' \rangle$  by (blast
  dest: tagged_partial_division_ofD)
  have interior  $(K \cap \{x. c \leq x \cdot i\}) = \{\}$ 
  proof (rule tagged_division_split_right_inj [OF  $\_ \langle (x,K) \in B \rangle$ 
   $\langle (x,L) \in B \rangle$ ])
  show  $B$  tagged_division_of  $\bigcup (snd \text{ ' } B)$ 
  using  $B_{tagged}$  tagged_partial_division_of_Union_self by
  auto
  show  $K \cap \{x. c \leq x \cdot i\} = L \cap \{x. c \leq x \cdot i\}$ 
  using  $eq \langle i \in Basis \rangle$  by auto
  qed (use that in auto)
  then show False
  using interval_split [OF  $\langle i \in Basis \rangle$ ] int_ne0
  content_eq_0_interior eq uv by fastforce
  qed
  then show  $K = L$  by blast
next
show  $(\sum (x,K) \in B. norm (?CI K h x + integral K f))$ 

```

```

      ≤ (∑ (x,K) ∈ B. norm (?CI K h x)) +
        (∑ (x,K) ∈ B. norm (integral K h)) + sum ((λ(x,K). norm
(integral K h)) ◦ (λ(x,K). (x,K ∩ {x. c ≤ x · i}))) B
    proof (clarsimp simp: B_def [symmetric] sum.distrib [symmetric]
intro!: sum_mono)
      fix x K
      assume (x,K) ∈ B
      have *: i = i1 + i2 ⇒ norm(c + i1) ≤ norm c + norm i +
norm(i2)
      for i::'b and c i1 i2
      by (metis add.commute add.left_commute add_diff_cancel_right'
dual_order.refl norm_add_rule_thm norm_triangle_ineq4)
      obtain u v where uv: K = cbox u v
      using T'_tagged ⟨(x,K) ∈ B⟩ ⟨B ⊆ T'⟩ ⟨T' ⊆ T'⟩ by (blast
dest: tagged_partial_division_ofD)
      have huv: h integrable_on cbox u v
      proof (rule integrable_on_subcbox)
      show cbox u v ⊆ cbox a b
      using B_tagged ⟨(x,K) ∈ B⟩ uv by (blast dest: tagged_partial_division_ofD)
      show h integrable_on cbox a b
      by (simp add: int_F ⟨h ∈ F⟩)
      qed
      have integral K h = integral K f + integral (K ∩ {x. c ≤ x · i}) h
      using integral_restrict_Int [of _ {x. x · i ≤ c} h] f uv ⟨i ∈
Basis⟩
      by (simp add: Int_commute integral_split [OF huv ⟨i ∈ Basis⟩])
      then show norm (?CI K h x + integral K f)
      ≤ norm (?CI K h x) + norm (integral K h) + norm (integral
(K ∩ {x. c ≤ x · i}) h)
      by (rule *)
      qed
      qed
      also have ... ≤ 2*ε/3
      proof -
      have overlap: K ∩ {x. x · i = c} ≠ {} if (x,K) ∈ T'' for x K
      proof -
      obtain y y' where y: y' ∈ K c < y' · i y ∈ K y · i ≤ c
      using that T''_def T'_def ⟨(x,K) ∈ T''⟩ by fastforce
      obtain u v where uv: K = cbox u v
      using T''_tagged ⟨(x,K) ∈ T''⟩ by (blast dest: tagged_partial_division_ofD)
      then have connected K
      by (simp add: is_interval_connected)
      then have (∃ z ∈ K. z · i = c)
      using y connected_ivt_component by fastforce
      then show ?thesis
      by fastforce
      qed
      have **: [x < ε/12; y < ε/12; z ≤ ε/2] ⇒ x + y + z ≤ 2 * ε/3 for

```

$x\ y\ z$

```

    by auto
  show ?thesis
proof (rule **)
  have cb_ab:  $(\sum j \in \text{Basis}. \text{if } j = i \text{ then } c *_{\mathbb{R}} i \text{ else } (a \cdot j) *_{\mathbb{R}} j) \in$ 
cbox a b
    using  $\langle i \in \text{Basis} \rangle \text{True} \langle \bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i \rangle$ 
    by (force simp add: mem_box sum_if_inner [where f =  $\lambda j. c$ ])
  show  $(\sum (x,K) \in A. \text{norm} (\text{integral } K\ h)) < \varepsilon/12$ 
    using  $\langle i \in \text{Basis} \rangle \langle A \subseteq T'' \rangle \text{overlap}$ 
    by (force simp add: sum_if_inner [where f =  $\lambda j. c$ ]
      intro!:  $\gamma 0$  [OF cb_ab  $\langle i \in \text{Basis} \rangle A\_tagged\ fineA(1) \langle h \in F \rangle$ ])
  let ?F =  $\lambda(x,K). (x, K \cap \{x. x \cdot i \leq c\})$ 
  have 1: ?F ' A tagged_partial_division_of cbox a b
    unfolding tagged_partial_division_of_def
  proof (intro conjI strip)
    show  $\bigwedge x\ K. (x, K) \in ?F ' A \implies \exists a\ b. K = \text{cbox } a\ b$ 
      using A_tagged_interval_split(1) [OF  $\langle i \in \text{Basis} \rangle$ , of _ _ c]
      by (force dest: tagged_partial_division_ofD(4))
    show  $\bigwedge x\ K. (x, K) \in ?F ' A \implies x \in K$ 
  using A_def A_tagged by (fastforce dest: tagged_partial_division_ofD)
  qed (use A_tagged in (fastforce dest: tagged_partial_division_ofD))+
  have 2:  $\gamma 0\ fine (\lambda(x,K). (x, K \cap \{x. x \cdot i \leq c\})) ' A$ 
    using fineA(1) fine_def by fastforce
  show  $(\sum (x,K) \in (\lambda(x,K). (x, K \cap \{x. x \cdot i \leq c\})) ' A. \text{norm} (\text{integral}$ 
K h)) <  $\varepsilon/12$ 
    using  $\langle i \in \text{Basis} \rangle \langle A \subseteq T'' \rangle \text{overlap}$ 
    by (force simp add: sum_if_inner [where f =  $\lambda j. c$ ]
      intro!:  $\gamma 0$  [OF cb_ab  $\langle i \in \text{Basis} \rangle 1\ 2 \langle h \in F \rangle$ ])
  have *:  $\llbracket x < \varepsilon/3; y < \varepsilon/12; z < \varepsilon/12 \rrbracket \implies x + y + z \leq \varepsilon/2$  for x
y z
    by auto
  show  $(\sum (x,K) \in B. \text{norm} (?CI\ K\ h\ x)) +$ 
 $(\sum (x,K) \in B. \text{norm} (\text{integral } K\ h)) +$ 
 $(\sum (x,K) \in (\lambda(x,K). (x, K \cap \{x. c \leq x \cdot i\})) ' B. \text{norm} (\text{integral}$ 
K h))
     $\leq \varepsilon/2$ 
  proof (rule *)
    show  $(\sum (x,K) \in B. \text{norm} (?CI\ K\ h\ x)) < \varepsilon/3$ 
      by (intro h_less3 B_tagged_fineB that)
    show  $(\sum (x,K) \in B. \text{norm} (\text{integral } K\ h)) < \varepsilon/12$ 
    using  $\langle i \in \text{Basis} \rangle \langle B \subseteq T'' \rangle \text{overlap}$ 
    by (force simp add: sum_if_inner [where f =  $\lambda j. c$ ]
      intro!:  $\gamma 0$  [OF cb_ab  $\langle i \in \text{Basis} \rangle B\_tagged\ fineB(1) \langle h \in F \rangle$ ])
    let ?F =  $\lambda(x,K). (x, K \cap \{x. c \leq x \cdot i\})$ 
    have 1: ?F ' B tagged_partial_division_of cbox a b
      unfolding tagged_partial_division_of_def
    proof (intro conjI strip)
      show  $\bigwedge x\ K. (x, K) \in ?F ' B \implies \exists a\ b. K = \text{cbox } a\ b$ 

```



```

      using B_tagged interval_split(2) [OF ‹i ∈ Basis›, of _ _ c]
      by (force dest: tagged_partial_division_ofD(4))
      show  $\bigwedge x K. (x, K) \in ?F \text{ ' } B \implies x \in K$ 
      using B_def B_tagged by (fastforce dest: tagged_partial_division_ofD)
    qed (use B_tagged in ‹fastforce dest: tagged_partial_division_ofD›)+
      have 2:  $\gamma 0 \text{ fine } (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\})) \text{ ' } B$ 
      using fineB(1) fine_def by fastforce
      show  $(\sum (x,K) \in (\lambda(x,K). (x,K \cap \{x. c \leq x \cdot i\})) \text{ ' } B. \text{ norm } (integral K h)) < \varepsilon/12$ 
      using ‹i ∈ Basis› ‹A ⊆ T''› overlap
      by (force simp add: B_def sum_if_inner [where f = λj. c]
        intro!:  $\gamma 0$  [OF cb_ab ‹i ∈ Basis› 1 2 ‹h ∈ F›])
    qed
  qed
  qed
  finally show ?thesis .
  qed
  ultimately show ?thesis by metis
  qed
  ultimately show ?thesis
  by (simp add: sum_subtractf [symmetric] int_KK' *)
  qed
  ultimately show ?thesis by metis
next
case False
then consider  $c < a \cdot i \mid b \cdot i < c$ 
  by auto
then show ?thesis
proof cases
case 1
then have f0:  $f x = 0$  if  $x \in cbox a b$  for  $x$ 
  using that f ‹i ∈ Basis› mem_box(2) by force
then have int_f0:  $integral (cbox a b) f = 0$ 
  by (simp add: integral_cong)
have f0_tag:  $f x = 0$  if  $(x,K) \in T$  for  $x K$ 
  using T f0 that by (meson tag_in_interval)
then have  $(\sum (x,K) \in T. content K *_R f x) = 0$ 
  by (metis (mono_tags, lifting) real_vector.scale_eq_0_iff split_conv
    sum.neutral surj_pair)
then show ?thesis
  using ‹0 < ε› by (simp add: int_f0)
next
case 2
then have fh:  $f x = h x$  if  $x \in cbox a b$  for  $x$ 
  using that f ‹i ∈ Basis› mem_box(2) by force
then have int_f:  $integral (cbox a b) f = integral (cbox a b) h$ 
  using integral_cong by blast
have fh_tag:  $f x = h x$  if  $(x,K) \in T$  for  $x K$ 
  using T fh that by (meson tag_in_interval)

```

```

then have fh: ( $\sum (x,K) \in T. \text{content } K *_R f x$ ) = ( $\sum (x,K) \in T. \text{content } K *_R h x$ )
  by (metis (mono_tags, lifting) split_cong sum.cong)
show ?thesis
  unfolding fh int_f
proof (rule less_trans [OF  $\gamma 1$ ])
  show  $\gamma 1$  fine T
    by (meson fine fine_Int)
  show  $\varepsilon / (\gamma * \text{Suc } \text{DIM}('b)) < \varepsilon$ 
    using  $\langle 0 < \varepsilon \rangle$  by (force simp: divide_simps)+
  qed (use that in auto)
qed
qed
have gauge ( $\lambda x. \gamma 0 x \cap \gamma 1 x$ )
  by (simp add:  $\langle \text{gauge } \gamma 0 \rangle \langle \text{gauge } \gamma 1 \rangle$  gauge_Int)
then show ?thesis
  by (auto intro: *)
qed
qed
qed

```

corollary equiintegrable_halfspace_restrictions_ge:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes F: F equiintegrable_on cbox a b and f: f  $\in$  F
  and norm_f:  $\bigwedge h x. \llbracket h \in F; x \in \text{cbox } a \ b \rrbracket \implies \text{norm}(h x) \leq \text{norm}(f x)$ 
shows ( $\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{\lambda x. \text{if } x \cdot i \geq c \text{ then } h x \text{ else } 0\}$ )
  equiintegrable_on cbox a b
proof -
  have *: ( $\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in (\lambda f. f \circ \text{uminus}) ' F. \{\lambda x. \text{if } x \cdot i \leq c \text{ then } h x \text{ else } 0\}$ )
    equiintegrable_on cbox (- b) (- a)
  proof (rule equiintegrable_halfspace_restrictions_le)
    show ( $\lambda f. f \circ \text{uminus}$ ) ' F equiintegrable_on cbox (- b) (- a)
      using F equiintegrable_reflect by blast
    show  $f \circ \text{uminus} \in (\lambda f. f \circ \text{uminus}) ' F$ 
      using f by auto
    show  $\bigwedge h x. \llbracket h \in (\lambda f. f \circ \text{uminus}) ' F; x \in \text{cbox } (- b) \ (- a) \rrbracket \implies \text{norm}(h x) \leq \text{norm}((f \circ \text{uminus}) x)$ 
      using f unfolding comp_def image_iff
      by (metis (no_types, lifting) equation_minus_iff imageE norm_f uminus_interval_vector)
    qed
  have eq: ( $\lambda f. f \circ \text{uminus}$ ) '
    ( $\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{\lambda x. \text{if } x \cdot i \leq c \text{ then } (h \circ \text{uminus}) x \text{ else } 0\}$ ) =
    ( $\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{\lambda x. \text{if } c \leq x \cdot i \text{ then } h x \text{ else } 0\}$ ) (is ?lhs =
  ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs

```

```

    using minus_le_iff by fastforce
  show ?rhs  $\subseteq$  ?lhs
  apply clarsimp
  apply (rule_tac x= $\lambda x. \text{if } c \leq (-x) \cdot i \text{ then } h(-x) \text{ else } 0$  in image_eqI)
    using le_minus_iff by fastforce+
  qed
  show ?thesis
  using equiintegrable_reflect [OF *] by (auto simp: eq)
qed

```

corollary *equiintegrable_halfspace_restrictions_lt:*

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes F: F equiintegrable_on cbox a b and f: f  $\in$  F
    and norm_f:  $\bigwedge h x. \llbracket h \in F; x \in \text{cbox } a \text{ b} \rrbracket \Longrightarrow \text{norm}(h x) \leq \text{norm}(f x)$ 
  shows  $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i < c \text{ then } h x \text{ else } 0)\})$  equiintegrable_on cbox a b
    (is ?G equiintegrable_on cbox a b)
  proof -
    have *:  $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } c \leq x \cdot i \text{ then } h x \text{ else } 0)\})$  equiintegrable_on cbox a b
      using equiintegrable_halfspace_restrictions_ge [OF F f] norm_f by auto
    have  $(\lambda x. \text{if } x \cdot i < c \text{ then } h x \text{ else } 0) = (\lambda x. h x - (\text{if } c \leq x \cdot i \text{ then } h x \text{ else } 0))$ 
      if  $i \in \text{Basis } h \in F$  for  $i c h$ 
      using that by force
    then show ?thesis
      by (blast intro: equiintegrable_on_subset [OF equiintegrable_diff [OF F *]])
  qed

```

corollary *equiintegrable_halfspace_restrictions_gt:*

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes F: F equiintegrable_on cbox a b and f: f  $\in$  F
    and norm_f:  $\bigwedge h x. \llbracket h \in F; x \in \text{cbox } a \text{ b} \rrbracket \Longrightarrow \text{norm}(h x) \leq \text{norm}(f x)$ 
  shows  $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } x \cdot i > c \text{ then } h x \text{ else } 0)\})$  equiintegrable_on cbox a b
    (is ?G equiintegrable_on cbox a b)
  proof -
    have *:  $(\bigcup i \in \text{Basis}. \bigcup c. \bigcup h \in F. \{(\lambda x. \text{if } c \geq x \cdot i \text{ then } h x \text{ else } 0)\})$  equiintegrable_on cbox a b
      using equiintegrable_halfspace_restrictions_le [OF F f] norm_f by auto
    have  $(\lambda x. \text{if } x \cdot i > c \text{ then } h x \text{ else } 0) = (\lambda x. h x - (\text{if } c \geq x \cdot i \text{ then } h x \text{ else } 0))$ 
      if  $i \in \text{Basis } h \in F$  for  $i c h$ 
      using that by force
    then show ?thesis
      by (blast intro: equiintegrable_on_subset [OF equiintegrable_diff [OF F *]])
  qed

```

proposition *equiintegrable_closed_interval_restrictions:*

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes f: f integrable_on cbox a b

```

```

shows ( $\bigcup c d. \{(\lambda x. \text{if } x \in \text{cbox } c d \text{ then } f x \text{ else } 0)\}$ ) equiintegrable_on cbox a b
proof –
  let ?g =  $\lambda B c d x. \text{if } \forall i \in B. c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i \text{ then } f x \text{ else } 0$ 
  have *: insert f ( $\bigcup c d. \{?g B c d\}$ ) equiintegrable_on cbox a b if  $B \subseteq \text{Basis}$  for
  B
proof –
  have finite B
  using finite_Basis finite_subset  $\langle B \subseteq \text{Basis} \rangle$  by blast
then show ?thesis using  $\langle B \subseteq \text{Basis} \rangle$ 
proof (induction B)
  case empty
  with f show ?case by auto
next
  case (insert i B)
  then have  $i \in \text{Basis } B \subseteq \text{Basis}$ 
  by auto
  have *: norm (h x) ≤ norm (f x)
  if  $h \in \text{insert } f (\bigcup c d. \{?g B c d\})$   $x \in \text{cbox } a b$  for  $h x$ 
  using that by auto
  define F where  $F \equiv (\bigcup i \in \text{Basis}.$ 
     $\bigcup \xi. \bigcup h \in \text{insert } f (\bigcup i \in \text{Basis}.$ 
     $\bigcup \psi. \bigcup h \in \text{insert } f (\bigcup c d. \{?g B c d\}).$ 
     $\{\lambda x. \text{if } x \cdot i \leq \psi \text{ then } h x \text{ else } 0\}).$ 
     $\{\lambda x. \text{if } \xi \leq x \cdot i \text{ then } h x \text{ else } 0\})$ 
  show ?case
  proof (rule equiintegrable_on_subset)
  have F equiintegrable_on cbox a b
  unfolding F_def
  proof (rule equiintegrable_halfspace_restrictions_ge)
  show insert f ( $\bigcup i \in \text{Basis}.$ 
     $\bigcup \xi. \bigcup h \in \text{insert } f (\bigcup c d. \{?g B c d\}).$ 
     $\{\lambda x. \text{if } x \cdot i \leq \xi \text{ then } h x \text{ else } 0\})$  equiintegrable_on cbox a b
  by (intro * f equiintegrable_on_insert equiintegrable_halfspace_restrictions_le
    [OF insert.IH insertI1]  $\langle B \subseteq \text{Basis} \rangle$ )
  show norm(h x) ≤ norm(f x)
  if  $h \in \text{insert } f (\bigcup i \in \text{Basis}.$ 
     $\bigcup \xi. \bigcup h \in \text{insert } f (\bigcup c d. \{?g B c d\}).$ 
     $\{\lambda x. \text{if } x \cdot i \leq \xi \text{ then } h x \text{ else } 0\})$ 
  if  $x \in \text{cbox } a b$  for  $h x$ 
  using that by auto
  qed auto
then show insert f F
  equiintegrable_on cbox a b
  by (blast intro: f equiintegrable_on_insert)
  show insert f ( $\bigcup c d. \{\lambda x. \text{if } \forall j \in \text{insert } i B. c \cdot j \leq x \cdot j \wedge x \cdot j \leq d \cdot j$ 
  then } f x \text{ else } 0\})
   $\subseteq \text{insert } f F$ 
  using  $\langle i \in \text{Basis} \rangle$ 
  apply clarify
  apply (simp add: F_def)
  apply (drule_tac x=i in bspec, assumption)
  apply (drule_tac x=c · i in spec, clarify)

```

```

    apply (drule_tac x=i in bspec, assumption)
    apply (drule_tac x=d · i in spec)
    apply (clarisimp simp: fun_eq_iff)
    apply (drule_tac x=c in spec)
    apply (drule_tac x=d in spec)
    apply (simp split: if_split_asm)
  done
qed
qed
qed
show ?thesis
  by (rule equiintegrable_on_subset [OF * [OF subset_refl]]) (auto simp: mem_box)
qed

```

9.14.3 Continuity of the indefinite integral

proposition *indefinite_integral_continuous*:

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: euclidean_space$

assumes $int_f: f \text{ integrable_on } cbox\ a\ b$

and $c \in cbox\ a\ b$ **and** $d: d \in cbox\ a\ b$ $0 < \varepsilon$

obtains δ **where** $0 < \delta$

$\bigwedge c' d'. \llbracket c' \in cbox\ a\ b; d' \in cbox\ a\ b; norm(c' - c) \leq \delta; norm(d' - d) \leq \delta \rrbracket$
 $\implies norm(integral(cbox\ c'\ d')\ f - integral(cbox\ c\ d)\ f) < \varepsilon$

proof –

{ **assume** $\exists c' d'. c' \in cbox\ a\ b \wedge d' \in cbox\ a\ b \wedge norm(c' - c) \leq \delta \wedge norm(d' - d) \leq \delta \wedge$

$norm(integral(cbox\ c'\ d')\ f - integral(cbox\ c\ d)\ f) \geq \varepsilon$

(**is** $\exists c' d'. ?\Phi\ c'\ d'\ \delta$) **if** $0 < \delta$ **for** δ

then have $\exists c' d'. ?\Phi\ c'\ d'\ (1 / Suc\ n)$ **for** n

by *simp*

then obtain $u\ v$ **where** $\bigwedge n. ?\Phi\ (u\ n)\ (v\ n)\ (1 / Suc\ n)$

by *metis*

then have $u: u\ n \in cbox\ a\ b$ **and** $norm_u: norm(u\ n - c) \leq 1 / Suc\ n$

and $v: v\ n \in cbox\ a\ b$ **and** $norm_v: norm(v\ n - d) \leq 1 / Suc\ n$

and $\varepsilon: \varepsilon \leq norm\ (integral\ (cbox\ (u\ n)\ (v\ n))\ f - integral\ (cbox\ c\ d)\ f)$ **for**

n

by *blast+*

then have *False*

proof –

have $u\ v\ n: cbox\ (u\ n)\ (v\ n) \subseteq cbox\ a\ b$ **for** n

by (*meson* $u\ v\ mem_box(2)\ subset_box(1)$)

define S **where** $S \equiv \bigcup i \in Basis. \{x. x \cdot i = c \cdot i\} \cup \{x. x \cdot i = d \cdot i\}$

have *negligible* S

unfolding S_def **by** *force*

then have $int_f': (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f\ x) \text{ integrable_on } cbox\ a\ b$

by (*force* *intro: integrable_spike* *assms*)

have $get_n: \exists n. \forall m \geq n. x \in cbox\ (u\ m)\ (v\ m) \longleftrightarrow x \in cbox\ c\ d$ **if** $x: x \notin S$

for x

```

proof -
  define  $\varepsilon$  where  $\varepsilon \equiv \text{Min } ((\lambda i. \text{min } |x \cdot i - c \cdot i| |x \cdot i - d \cdot i|) \text{ 'Basis})$ 
  have  $\varepsilon > 0$ 
    using  $\langle x \notin S \rangle$  by (auto simp: S_def  $\varepsilon\_def$ )
  then obtain  $n$  where  $n \neq 0$  and  $n: 1 / (\text{real } n) < \varepsilon$ 
    by (metis inverse_eq_divide real_arch_inverse)
  have  $\text{emin}: \varepsilon \leq \text{min } |x \cdot i - c \cdot i| |x \cdot i - d \cdot i|$  if  $i \in \text{Basis}$  for  $i$ 
    unfolding  $\varepsilon\_def$ 
  by (meson Min.coboundedI euclidean_space_class.finite_Basis finite_imageI
  image_iff that)
  have  $1 / \text{real } (\text{Suc } n) < \varepsilon$ 
    using  $n \langle n \neq 0 \rangle \langle \varepsilon > 0 \rangle$  by (simp add: field_simps)
  have  $x \in \text{cbox } (u \ m) \ (v \ m) \longleftrightarrow x \in \text{cbox } c \ d$  if  $m \geq n$  for  $m$ 
  proof -
    have *:  $[|u - c| \leq n; |v - d| \leq n; N < |x - c|; N < |x - d|; n \leq N]$ 
       $\implies u \leq x \wedge x \leq v \longleftrightarrow c \leq x \wedge x \leq d$  for  $N \ n \ u \ v \ c \ d$  and  $x::\text{real}$ 
    by linarith
    have  $(u \ m \cdot i \leq x \cdot i \wedge x \cdot i \leq v \ m \cdot i) = (c \cdot i \leq x \cdot i \wedge x \cdot i \leq d \cdot i)$ 
      if  $i \in \text{Basis}$  for  $i$ 
    proof (rule *)
      show  $|u \ m \cdot i - c \cdot i| \leq 1 / \text{Suc } m$ 
        using norm_u [of m]
        by (metis (full_types) order_trans Basis_le_norm inner_commute
  inner_diff_right that)
      show  $|v \ m \cdot i - d \cdot i| \leq 1 / \text{real } (\text{Suc } m)$ 
        using norm_v [of m]
        by (metis (full_types) order_trans Basis_le_norm inner_commute
  inner_diff_right that)
      show  $1/n < |x \cdot i - c \cdot i| \ 1/n < |x \cdot i - d \cdot i|$ 
        using  $n \langle n \neq 0 \rangle \text{emin}$  [OF  $\langle i \in \text{Basis} \rangle$ ]
        by (simp_all add: inverse_eq_divide)
      show  $1 / \text{real } (\text{Suc } m) \leq 1 / \text{real } n$ 
        using  $\langle n \neq 0 \rangle \langle m \geq n \rangle$  by (simp add: field_split_simps)
    qed
  then show ?thesis by (simp add: mem_box)
  qed
  then show ?thesis by blast
  qed
  have 1:  $\text{range } (\lambda n \ x. \text{if } x \in \text{cbox } (u \ n) \ (v \ n) \text{ then if } x \in S \text{ then } 0 \text{ else } f \ x \text{ else } 0)$ 
    equiintegrable_on cbox a b
  by (blast intro: equiintegrable_on_subset [OF equiintegrable_closed_interval_restrictions
  [OF int_f]])
  have 2:  $(\lambda n. \text{if } x \in \text{cbox } (u \ n) \ (v \ n) \text{ then if } x \in S \text{ then } 0 \text{ else } f \ x \text{ else } 0)$ 
     $\longrightarrow (\text{if } x \in \text{cbox } c \ d \text{ then if } x \in S \text{ then } 0 \text{ else } f \ x \text{ else } 0)$  for  $x$ 
  by (fastforce simp: dest: get_n intro: tendsto_eventually_eventually_sequentiallyI)
  have [simp]:  $\text{cbox } c \ d \cap \text{cbox } a \ b = \text{cbox } c \ d$ 
    using  $c \ d$  by (force simp: mem_box)
  have [simp]:  $\text{cbox } (u \ n) \ (v \ n) \cap \text{cbox } a \ b = \text{cbox } (u \ n) \ (v \ n)$  for  $n$ 
    using  $u \ v$  by (fastforce simp: mem_box intro: order.trans)

```

```

have  $\bigwedge y A. y \in A - S \implies f y = (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f x) y$ 
  by simp
then have  $\bigwedge A. \text{integral } A (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f (x)) = \text{integral } A (\lambda x.$ 
f (x))
  by (blast intro: integral_spike [OF ‹negligible S›])
moreover
obtain N where  $\text{dist } (\text{integral } (\text{cbox } (u \ N) \ (v \ N)) (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f$ 
x))
   $(\text{integral } (\text{cbox } c \ d) (\lambda x. \text{if } x \in S \text{ then } 0 \text{ else } f x)) < \varepsilon$ 
  using equiintegrable_limit [OF 1 2] ‹0 < ε› by (force simp: integral_restrict_Int
lim_sequentially)
  ultimately have  $\text{dist } (\text{integral } (\text{cbox } (u \ N) \ (v \ N)) f) (\text{integral } (\text{cbox } c \ d) f)$ 
< ε
  by simp
then show False
  by (metis dist_norm_not_le ε)
qed
}
then show ?thesis
  by (meson not_le that)
qed

```

corollary *indefinite_integral_uniformly_continuous:*

```

fixes f :: 'a :: euclidean_space  $\Rightarrow$  'b :: euclidean_space
assumes f integrable_on cbox a b
shows uniformly_continuous_on (cbox (Pair a a) (Pair b b)) ( $\lambda y. \text{integral } (\text{cbox}$ 
(fst y) (snd y)) f)
proof –
  show ?thesis
proof (rule compact_uniformly_continuous, clarsimp simp add: continuous_on_iff)
  fix c d and  $\varepsilon :: \text{real}$ 
  assume c: c ∈ cbox a b and d: d ∈ cbox a b and  $0 < \varepsilon$ 
  obtain  $\delta$  where  $0 < \delta$  and  $\delta$ :
     $\bigwedge c' d'. \llbracket c' \in \text{cbox } a \ b; d' \in \text{cbox } a \ b; \text{norm}(c' - c) \leq \delta; \text{norm}(d' - d)$ 
 $\leq \delta \rrbracket$ 
     $\implies \text{norm}(\text{integral}(\text{cbox } c' \ d') f -$ 
 $\text{integral}(\text{cbox } c \ d) f) < \varepsilon$ 
  using indefinite_integral_continuous ‹0 < ε› assms c d by blast
show  $\exists \delta > 0. \forall x' \in \text{cbox } (a, a) \ (b, b).$ 
 $\text{dist } x' \ (c, d) < \delta \implies$ 
 $\text{dist } (\text{integral } (\text{cbox } (\text{fst } x') \ (\text{snd } x')) f)$ 
 $(\text{integral } (\text{cbox } c \ d) f)$ 
 $< \varepsilon$ 
  using  $\langle 0 < \delta \rangle$ 
  by (force simp: dist_norm intro: δ order_trans [OF norm_fst_le] order_trans
[OF norm_snd_le] less_imp_le)
qed auto
qed

```

corollary *bounded_integrals_over_subintervals:*

fixes $f :: 'a :: euclidean_space \Rightarrow 'b :: euclidean_space$

assumes f *integrable_on* $cbox\ a\ b$

shows $bounded\ \{integral\ (cbox\ c\ d)\ f\ |\ c\ d.\ cbox\ c\ d \subseteq cbox\ a\ b\}$

proof –

have $bounded\ ((\lambda y.\ integral\ (cbox\ (fst\ y)\ (snd\ y))\ f)\ 'cbox\ (a,\ a)\ (b,\ b))$
(is $bounded\ ?I$)

by (*blast intro: bounded_cbox bounded_uniformly_continuous_image indefinite_integral_uniformly_continuous [OF assms]*)

then obtain B **where** $B > 0$ **and** $B: \bigwedge x.\ x \in ?I \implies norm\ x \leq B$

by (*auto simp: bounded_pos*)

have $norm\ x \leq B$ **if** $x = integral\ (cbox\ c\ d)\ f$ $cbox\ c\ d \subseteq cbox\ a\ b$ **for** $x\ c\ d$

proof (*cases cbox c d = {}*)

case *True*

with $\langle 0 < B \rangle$ **that show** $?thesis$ **by** *auto*

next

case *False*

then have $\exists x \in cbox\ (a,a)\ (b,b).\ integral\ (cbox\ c\ d)\ f = integral\ (cbox\ (fst\ x)\ (snd\ x))\ f$

using that by (*metis cbox_Pair_iff interval_subset_is_interval is_interval_cbox prod.sel*)

then show $?thesis$

using B *that(1)* **by** *blast*

qed

then show $?thesis$

by (*blast intro: boundedI*)

qed

An existence theorem for "improper" integrals. Hake's theorem implies that if the integrals over subintervals have a limit, the integral exists. We only need to assume that the integrals are bounded, and we get absolute integrability, but we also need a (rather weak) bound assumption on the function.

theorem *absolutely_integrable_improper:*

fixes $f :: 'M :: euclidean_space \Rightarrow 'N :: euclidean_space$

assumes $int_f: \bigwedge c\ d.\ cbox\ c\ d \subseteq box\ a\ b \implies f$ *integrable_on* $cbox\ c\ d$

and $bo: bounded\ \{integral\ (cbox\ c\ d)\ f\ |\ c\ d.\ cbox\ c\ d \subseteq box\ a\ b\}$

and $absi: \bigwedge i.\ i \in Basis$

$\implies \exists g.\ g$ *absolutely_integrable_on* $cbox\ a\ b \wedge$

$((\forall x \in cbox\ a\ b.\ f\ x \cdot i \leq g\ x) \vee (\forall x \in cbox\ a\ b.\ f\ x \cdot i \geq g\ x))$

shows f *absolutely_integrable_on* $cbox\ a\ b$

proof (*cases content(cbox a b) = 0*)

case *True*

then show $?thesis$

by *auto*

next

case *False*

then have $pos: content(cbox\ a\ b) > 0$

using *zero_less_measure_iff* **by** *blast*


```

show ?thesis
  unfolding absolutely_integrable_componentwise_iff [where f = f]
proof
  fix j::'N
  assume j ∈ Basis
  then obtain g where absint_g: g absolutely_integrable_on cbox a b
    and g: (∀ x ∈ cbox a b. f x · j ≤ g x) ∨ (∀ x ∈ cbox a b. f x · j ≥
g x)
    using absi by blast
  have int_gab: g integrable_on cbox a b
    using absint_g set_lebesgue_integral_eq_integral(1) by blast
  define α where α ≡ λk. a + (b - a) /R real k
  define β where β ≡ λk. b - (b - a) /R real k
  define I where I ≡ λk. cbox (α k) (β k)
  have ISuc_box: I (Suc n) ⊆ box a b for n
    using pos unfolding I_def
    by (intro subset_box_imp) (auto simp: α_def β_def content_pos_lt_eq
algebra_simps)
  have ISucSuc: I (Suc n) ⊆ I (Suc (Suc n)) for n
  proof -
    have ∧i. i ∈ Basis
      ⇒ a · i / Suc n + b · i / (real n + 2) ≤ b · i / Suc n + a · i /
(real n + 2)
    using pos
    by (simp add: content_pos_lt_eq divide_simps) (auto simp: algebra_simps)
  then show ?thesis
    unfolding I_def
    by (intro subset_box_imp) (auto simp: algebra_simps inverse_eq_divide
α_def β_def)
  qed
  have getN: ∃ N::nat. ∀ k. k ≥ N → x ∈ I k
  if x: x ∈ box a b for x
  proof -
    define Δ where Δ ≡ (∪ i ∈ Basis. {((x - a) · i) / ((b - a) · i), (b - x) ·
i / ((b - a) · i)})
    obtain N where N: real N > 1 / Inf Δ
    using reals_Archimedean2 by blast
    moreover have Δ: Inf Δ > 0
    using that by (auto simp: Δ_def finite_less_Inf_iff mem_box algebra_simps
divide_simps)
    ultimately have N > 0
    using of_nat_0_less_iff by fastforce
  show ?thesis
  proof (intro exI impI allI)
    fix k assume N ≤ k
    with ⟨0 < N⟩ have k > 0
    by linarith
    have xa_gt: (x - a) · i > ((b - a) · i) / (real k) if i ∈ Basis for i
  proof -

```

```

have *: Inf Δ ≤ ((x - a) · i) / ((b - a) · i)
  unfolding Δ_def using that by (force intro: cInf_le_finite)
have 1 / Inf Δ ≥ ((b - a) · i) / ((x - a) · i)
  using le_imp_inverse_le [OF * Δ]
  by (simp add: field_simps)
with N have k > ((b - a) · i) / ((x - a) · i)
  using ⟨N ≤ k⟩ by linarith
with x that show ?thesis
  by (auto simp: mem_box algebra_simps field_split_simps)
qed
have bx_gt: (b - x) · i > ((b - a) · i) / k if i ∈ Basis for i
proof -
  have *: Inf Δ ≤ ((b - x) · i) / ((b - a) · i)
    using that unfolding Δ_def by (force intro: cInf_le_finite)
  have 1 / Inf Δ ≥ ((b - a) · i) / ((b - x) · i)
    using le_imp_inverse_le [OF * Δ]
    by (simp add: field_simps)
  with N have k > ((b - a) · i) / ((b - x) · i)
    using ⟨N ≤ k⟩ by linarith
  with x that show ?thesis
    by (auto simp: mem_box algebra_simps field_split_simps)
qed
show x ∈ I k
  using that Δ ⟨k > 0⟩ unfolding I_def
  by (auto simp: α_def β_def mem_box algebra_simps divide_inverse dest:
xa_gt bx_gt)
qed
qed
obtain Bf where Bf: ∧c d. cbox c d ⊆ box a b ⇒ norm (integral (cbox c d)
f) ≤ Bf
  using bo unfolding bounded_iff by blast
obtain Bg where Bg: ∧c d. cbox c d ⊆ cbox a b ⇒ |integral (cbox c d) g| ≤
Bg
  using bounded_integrals_over_subintervals [OF int_gab] unfolding bounded_iff
real_norm_def by blast
show (λx. f x · j) absolutely_integrable_on cbox a b
  using g
proof — A lot of duplication in the two proofs
assume fg [rule_format]: ∀ x ∈ cbox a b. f x · j ≤ g x
have (λx. (f x · j)) = (λx. g x - (g x - (f x · j)))
  by simp
moreover have (λx. g x - (g x - (f x · j))) integrable_on cbox a b
proof (rule Henstock_Kurzweil_Integration.integrable_diff [OF int_gab])
define φ where φ ≡ λk x. if x ∈ I (Suc k) then g x - f x · j else 0
have (λx. g x - f x · j) integrable_on box a b
proof (rule monotone_convergence_increasing [of φ, THEN conjunct1])
have *: I (Suc k) ∩ box a b = I (Suc k) for k
  using box_subset_cbox ISuc_box by fastforce
show φ k integrable_on box a b for k

```

```

proof -
  have  $I (Suc\ k) \subseteq cbox\ a\ b$ 
    using * box_subset_cbox by blast
  moreover have  $(\lambda m. f\ m \cdot j)$  integrable_on  $I (Suc\ k)$ 
    by (metis ISuc_box I_def int_f integrable_component)
  ultimately have  $(\lambda m. g\ m - f\ m \cdot j)$  integrable_on  $I (Suc\ k)$ 
    by (metis Henstock_Kurzweil_Integration.integrable_diff I_def int_gab
    integrable_on_subcbox)
  then show ?thesis
    by (simp add: *  $\varphi$ _def integrable_restrict_Int)
qed
show  $\varphi\ k\ x \leq \varphi (Suc\ k)\ x$  if  $x \in box\ a\ b$  for  $k\ x$ 
  using ISucSuc_box_subset_cbox that by (force simp:  $\varphi$ _def intro!: fg)
show  $(\lambda k. \varphi\ k\ x) \longrightarrow g\ x - f\ x \cdot j$  if  $x: x \in box\ a\ b$  for  $x$ 
proof (rule tendsto_eventually)
  obtain  $N::nat$  where  $N: \bigwedge k. k \geq N \implies x \in I\ k$ 
    using getN [OF  $x$ ] by blast
  show  $\forall_F\ k$  in sequentially.  $\varphi\ k\ x = g\ x - f\ x \cdot j$ 
  proof
    fix  $k::nat$  assume  $N \leq k$ 
    have  $x \in I (Suc\ k)$ 
      by (metis  $\langle N \leq k \rangle$  le_Suc_eq  $N$ )
    then show  $\varphi\ k\ x = g\ x - f\ x \cdot j$ 
      by (simp add:  $\varphi$ _def)
  qed
qed
have  $|\text{integral}\ (box\ a\ b)\ (\lambda x. \text{if}\ x \in I (Suc\ k)\ \text{then}\ g\ x - f\ x \cdot j\ \text{else}\ 0)| \leq$ 
 $Bg + Bf$  for  $k$ 
proof -
  have  $ABK\_def [simp]: I (Suc\ k) \cap box\ a\ b = I (Suc\ k)$ 
    using ISuc_box by (simp add: Int_absorb2)
  have  $int\_fI: f$  integrable_on  $I (Suc\ k)$ 
    using ISuc_box I_def int_f by auto
  moreover
  have  $|\text{integral}\ (I (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq norm\ (\text{integral}\ (I (Suc\ k))\ f)$ 
    by (simp add: Basis_le_norm int_fI  $\langle j \in Basis \rangle$ )
  with ISuc_box  $ABK\_def$  have  $|\text{integral}\ (I (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq Bf$ 
    by (metis  $Bf\ I\_def\ \langle j \in Basis \rangle$  int_fI integrable_component_eq
    norm_bound_Basis_le)
  ultimately
  have  $|\text{integral}\ (I (Suc\ k))\ g - \text{integral}\ (I (Suc\ k))\ (\lambda x. f\ x \cdot j)| \leq Bg$ 
    +  $Bf$ 
    using * box_subset_cbox unfolding I_def
    by (blast intro:  $Bg$  add_mono order_trans [OF abs_triangle_ineq4])
  moreover have  $g$  integrable_on  $I (Suc\ k)$ 
    by (metis ISuc_box I_def int_gab integrable_on_open_interval
    integrable_on_subcbox)
  moreover have  $(\lambda x. f\ x \cdot j)$  integrable_on  $I (Suc\ k)$ 
    using int_fI by (simp add: integrable_component)

```

```

ultimately show ?thesis
  by (simp add: integral_restrict_Int integral_diff)
qed
then show bounded (range ( $\lambda k. \text{integral } (\text{box } a \ b) (\varphi \ k)$ ))
  by (auto simp add: bounded_iff  $\varphi\_def$ )
qed
then show ( $\lambda x. g \ x - f \ x \cdot j$ ) integrable_on cbox a b
  by (simp add: integrable_on_open_interval)
qed
ultimately have ( $\lambda x. f \ x \cdot j$ ) integrable_on cbox a b
  by auto
then show ?thesis
using absolutely_integrable_component_ubound [OF _ absint_g] fg by force
next
assume gf [rule_format]:  $\forall x \in \text{cbox } a \ b. g \ x \leq f \ x \cdot j$ 
have ( $\lambda x. (f \ x \cdot j) = (\lambda x. ((f \ x \cdot j) - g \ x) + g \ x)$ )
  by simp
moreover have ( $\lambda x. (f \ x \cdot j - g \ x) + g \ x$ ) integrable_on cbox a b
proof (rule Henstock_Kurzweil_Integration.integrable_add [OF _ int_gab])
let  $\varphi = \lambda k \ x. \text{if } x \in I(\text{Suc } k) \text{ then } f \ x \cdot j - g \ x \text{ else } 0$ 
have ( $\lambda x. f \ x \cdot j - g \ x$ ) integrable_on box a b
proof (rule monotone_convergence_increasing [of  $\varphi$ , THEN conjunct1])
have *:  $I(\text{Suc } k) \cap \text{box } a \ b = I(\text{Suc } k)$  for k
  using box_subset_cbox ISuc_box by fastforce
show  $\varphi \ k$  integrable_on box a b for k
proof (simp add: integrable_restrict_Int integral_restrict_Int *)
show ( $\lambda x. f \ x \cdot j - g \ x$ ) integrable_on  $I(\text{Suc } k)$ 
by (metis ISuc_box Henstock_Kurzweil_Integration.integrable_diff I_def
int_f int_gab integrable_component integrable_on_open_interval integrable_on_subcbox)
qed
show  $\varphi \ k \ x \leq \varphi (\text{Suc } k) \ x$  if  $x \in \text{box } a \ b$  for k x
  using ISucSuc box_subset_cbox that by (force simp: I_def intro: gf)
show ( $\lambda k. \varphi \ k \ x$ )  $\longrightarrow f \ x \cdot j - g \ x$  if  $x \in \text{box } a \ b$  for x
proof (rule tendsto_eventually)
obtain  $N::\text{nat}$  where  $N: \bigwedge k. k \geq N \implies x \in I \ k$ 
  using getN [OF x] by blast
then show  $\forall_F k$  in sequentially.  $\varphi \ k \ x = f \ x \cdot j - g \ x$ 
  by (metis (no_types, lifting) eventually_at_top_linorder I le_Suc_eq)
qed
have |integral (box a b)
  ( $\lambda x. \text{if } x \in I(\text{Suc } k) \text{ then } f \ x \cdot j - g \ x \text{ else } 0$ )|  $\leq B_f + B_g$  for k
proof -
define ABK where  $ABK \equiv \text{cbox } (a + (b - a) /_R (1 + \text{real } k)) (b -$ 
 $(b - a) /_R (1 + \text{real } k))$ 
have  $ABK\_eq$  [simp]:  $ABK \cap \text{box } a \ b = ABK$ 
  using * I_def  $\alpha\_def$   $\beta\_def$   $ABK\_def$  by auto
have int_fI:  $f$  integrable_on ABK
  unfolding ABK_def
  using ISuc_box I_def  $\alpha\_def$   $\beta\_def$  int_f by force

```

```

    then have  $(\lambda x. f x \cdot j)$  integrable_on ABK
      by (simp add: integrable_component)
    moreover have  $g$  integrable_on ABK
      by (metis ABK_def ABK_eq IntE box_subset_cbox int_gab integrable_on_subcbox subset_eq)
    moreover
    have  $|\text{integral ABK } (\lambda x. f x \cdot j)| \leq \text{norm } (\text{integral ABK } f)$ 
      by (simp add: Basis_le_norm int_fI ⟨j ∈ Basis⟩)
    then have  $|\text{integral ABK } (\lambda x. f x \cdot j)| \leq Bf$ 
      by (metis ABK_eq ABK_def Bf IntE dual_order.trans subset_eq)
    ultimately show ?thesis
      using * box_subset_cbox
      apply (simp add: integral_restrict_Int integral_diff ABK_def I_def
        α_def β_def)
      by (blast intro: Bg add_mono order_trans [OF abs_triangle_ineq4])
    qed
    then show bounded (range  $(\lambda k. \text{integral } (\text{box } a \ b) \ (\varphi \ k))$ )
      by (auto simp add: bounded_iff)
    qed
    then show  $(\lambda x. f x \cdot j - g \ x)$  integrable_on cbox a b
      by (simp add: integrable_on_open_interval)
    qed
    ultimately have  $(\lambda x. f x \cdot j)$  integrable_on cbox a b
      by auto
    then show ?thesis
      using absint_g absolutely_integrable_absolutely_integrable_lbound gf by
blast
    qed
  qed
  qed

```

9.14.4 Second mean value theorem and corollaries

lemma level_approx:

fixes $f :: \text{real} \Rightarrow \text{real}$ and $n :: \text{nat}$

assumes $f: \bigwedge x. x \in S \implies 0 \leq f x \wedge f x \leq 1$ and $x \in S \ n \neq 0$

shows $|f x - (\sum k = \text{Suc } 0..n. \text{if } k / n \leq f x \text{ then inverse } n \text{ else } 0)| < \text{inverse } n$
 (is ?lhs < _)

proof -

have $n * f x \geq 0$

using assms by auto

then obtain $m :: \text{nat}$ where $m: \text{floor}(n * f x) = \text{int } m$

using nonneg_int_cases zero_le_floor by blast

then have $kn: \text{real } k / \text{real } n \leq f x \iff k \leq m$ for k

using $\langle n \neq 0 \rangle$ by (simp add: field_split_simps) linarith

then have $\text{Suc } n / \text{real } n \leq f x \iff \text{Suc } n \leq m$

by blast

have $\text{real } n * f x \leq \text{real } n$

by (simp add: $\langle x \in S \rangle f \text{ mult_left_le}$)

```

then have  $m \leq n$ 
  using  $m$  by linarith
have  $?lhs = |f x - (\sum k \in \{Suc\ 0..n\} \cap \{..m\}. inverse\ n)|$ 
  by (subst sum.inter_restrict) (auto simp: kn)
also have  $\dots < inverse\ n$ 
  using  $\langle m \leq n \rangle \langle n \neq 0 \rangle m$ 
  by (simp add: min_absorb2 field_split_simps) linarith
finally show ?thesis .
qed

```

lemma *SMVT_lemma2*:

```

fixes  $f :: real \Rightarrow real$ 
assumes  $f: f\ integrable\_on\ \{a..b\}$ 
  and  $g: \bigwedge x\ y. x \leq y \implies g\ x \leq g\ y$ 
shows  $(\bigcup y::real. \{\lambda x. if\ g\ x \geq y\ then\ f\ x\ else\ 0\})\ equiintegrable\_on\ \{a..b\}$ 
proof -
  have  $ffab: \{f\}\ equiintegrable\_on\ \{a..b\}$ 
    by (metis equiintegrable_on_sing f interval_cbox)
  then have  $ff: \{f\}\ equiintegrable\_on\ (cbox\ a\ b)$ 
    by simp
  have  $ge: (\bigcup c. \{\lambda x. if\ x \geq c\ then\ f\ x\ else\ 0\})\ equiintegrable\_on\ \{a..b\}$ 
    using equiintegrable_halfspace_restrictions_ge [OF ff] by auto
  have  $gt: (\bigcup c. \{\lambda x. if\ x > c\ then\ f\ x\ else\ 0\})\ equiintegrable\_on\ \{a..b\}$ 
    using equiintegrable_halfspace_restrictions_gt [OF ff] by auto
  have  $0: \{(\lambda x. 0)\}\ equiintegrable\_on\ \{a..b\}$ 
    by (metis box_real(2) equiintegrable_on_sing integrable_0)
  have  $\dagger: (\lambda x. if\ g\ x \geq y\ then\ f\ x\ else\ 0) \in \{(\lambda x. 0), f\} \cup (\bigcup z. \{\lambda x. if\ z < x\ then\ f\ x\ else\ 0\}) \cup (\bigcup z. \{\lambda x. if\ z \leq x\ then\ f\ x\ else\ 0\})$ 
    for  $y$ 
  proof (cases  $(\forall x. g\ x \geq y) \vee (\forall x. \neg (g\ x \geq y))$ )
    let  $?mu = Inf\ \{x. g\ x \geq y\}$ 
    case False
    have  $lower: ?mu \leq x\ \text{if}\ g\ x \geq y\ \text{for}\ x$ 
    proof (rule cInf_lower)
      show  $x \in \{x. y \leq g\ x\}$ 
        using False by (auto simp: that)
      show bdd_below  $\{x. y \leq g\ x\}$ 
        by (metis False bdd_belowI dual_order.trans g linear mem_Collect_eq)
    qed
    have  $greatest: ?mu \geq z\ \text{if}\ (\bigwedge x. g\ x \geq y \implies z \leq x)\ \text{for}\ z$ 
    by (metis False cInf_greatest empty_iff mem_Collect_eq that)
  show ?thesis
  proof (cases  $g\ ?mu \geq y$ )
    case True
    then obtain  $\zeta$  where  $\zeta: \bigwedge x. g\ x \geq y \iff x \geq \zeta$ 
      by (metis g lower order.trans) — in fact  $y$  is  $Inf\ \{x. y \leq g\ x\}$ 
    then show ?thesis
      by (force simp: \zeta)
  qed

```

```

next
  case False
  have  $(y \leq g\ x) \longleftrightarrow (? \mu < x)$  for  $x$ 
  proof
    show  $? \mu < x$  if  $y \leq g\ x$ 
      using that False less_eq_real_def lower by blast
    show  $y \leq g\ x$  if  $? \mu < x$ 
      by (metis g greatest le_less_trans that less_le_trans linear not_less)
  qed
  then obtain  $\zeta$  where  $\zeta: \bigwedge x. g\ x \geq y \longleftrightarrow x > \zeta ..$ 
  then show ?thesis
    by (force simp: \zeta)
  qed
qed auto
show ?thesis
  using † by (simp add: UN_subset_iff equiintegrable_on_subset [OF equiintegrable_on_Un [OF gt equiintegrable_on_Un [OF ge equiintegrable_on_Un [OF ffab 0]]]])
qed

```

lemma *SMVT_lemma4*:

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $f: f \text{ integrable\_on } \{a..b\}$ 
  and  $a \leq b$ 
  and  $g: \bigwedge x\ y. x \leq y \implies g\ x \leq g\ y$ 
  and  $01: \bigwedge x. \llbracket a \leq x; x \leq b \rrbracket \implies 0 \leq g\ x \wedge g\ x \leq 1$ 
obtains  $c$  where  $a \leq c \leq b$   $((\lambda x. g\ x *_{\mathbb{R}} f\ x) \text{ has\_integral } \text{integral } \{c..b\} f)$ 
 $\{a..b\}$ 
proof –
  have connected  $((\lambda x. \text{integral } \{x..b\} f) ' \{a..b\})$ 
    by (simp add: findefinite_integral_continuous_1' connected_continuous_image)
  moreover have compact  $((\lambda x. \text{integral } \{x..b\} f) ' \{a..b\})$ 
    by (simp add: compact_continuous_image findefinite_integral_continuous_1')
  ultimately obtain  $m\ M$  where int_fab:  $(\lambda x. \text{integral } \{x..b\} f) ' \{a..b\} = \{m..M\}$ 
    using connected_compact_interval_1 by meson
  have  $\exists c. c \in \{a..b\} \wedge$ 
     $\text{integral } \{c..b\} f =$ 
     $\text{integral } \{a..b\} (\lambda x. (\sum k = 1..n. \text{if } g\ x \geq \text{real } k / \text{real } n \text{ then inverse } n$ 
 $*_{\mathbb{R}} f\ x \text{ else } 0))$  for  $n$ 
  proof (cases n=0)
    case True
    then show ?thesis
      using  $\langle a \leq b \rangle$  by auto
  next
  case False
  have  $(\bigcup c::\text{real}. \{\lambda x. \text{if } g\ x \geq c \text{ then } f\ x \text{ else } 0\}) \text{ equiintegrable\_on } \{a..b\}$ 
    using SMVT_lemma2  $[OF\ f\ g]$  .

```

```

then have int: ( $\lambda x. \text{if } g \ x \geq c \ \text{then } f \ x \ \text{else } 0$ ) integrable_on {a..b} for c
  by (simp add: equiintegrable_on_def)
have int': ( $\lambda x. \text{if } g \ x \geq c \ \text{then } u * f \ x \ \text{else } 0$ ) integrable_on {a..b} for c u
proof -
  have ( $\lambda x. \text{if } g \ x \geq c \ \text{then } u * f \ x \ \text{else } 0$ ) = ( $\lambda x. u * (\text{if } g \ x \geq c \ \text{then } f \ x \ \text{else } 0)$ )
0))
  by (force simp: if_distrib)
then show ?thesis
  using integrable_on_cmult_left [OF int] by simp
qed
have  $\exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if } g \ x \geq y \ \text{then } f \ x \ \text{else } 0) = \text{integral } \{d..b\} f$  for y
proof -
  let ?X = {x. g x  $\geq$  y}
  have *:  $\exists a. ?X = \{a..\} \vee ?X = \{a<..\}$ 
  if 1: ?X  $\neq$  {} and 2: ?X  $\neq$  UNIV
proof -
  let ? $\mu$  = Inf {x. g x  $\geq$  y}
  have lower: ? $\mu$   $\leq$  x if g x  $\geq$  y for x
  proof (rule cInf_lower)
    show x  $\in$  {x. y  $\leq$  g x}
    using 1 2 by (auto simp: that)
    show bdd_below {x. y  $\leq$  g x}
    unfolding bdd_below_def
  by (metis 2 UNIV_eq_I dual_order.trans g less_eq_real_def mem_Collect_eq not_le)
qed
have greatest: ? $\mu$   $\geq$  z if  $\bigwedge x. g \ x \geq y \implies z \leq x$  for z
  by (metis cInf_greatest mem_Collect_eq that 1)
show ?thesis
proof (cases g ? $\mu$   $\geq$  y)
  case True
  then obtain  $\zeta$  where  $\zeta: \bigwedge x. g \ x \geq y \longleftrightarrow x \geq \zeta$ 
  by (metis g lower order.trans) — in fact y is Inf {x. y  $\leq$  g x}
  then show ?thesis
  by (force simp:  $\zeta$ )
next
  case False
  have (y  $\leq$  g x) = (? $\mu$  < x) for x
  proof
    show ? $\mu$  < x if y  $\leq$  g x
    using that False less_eq_real_def lower by blast
    show y  $\leq$  g x if ? $\mu$  < x
    by (metis g greatest le_less_trans that less_le_trans linear not_less)
  qed
  then obtain  $\zeta$  where  $\zeta: \bigwedge x. g \ x \geq y \longleftrightarrow x > \zeta$  ..
  then show ?thesis
  by (force simp:  $\zeta$ )
qed

```



```

qed
then consider ?X = {} | ?X = UNIV | (intv) d where ?X = {d..} ∨ ?X
= {d<..}
  by metis
  then have  $\exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if } x \in ?X \text{ then } f x \text{ else } 0) =$ 
integral {d..b} f
  proof cases
    case (intv d)
      show ?thesis
      proof (cases d < a)
        case True
          with intv have integral {a..b} ( $\lambda x. \text{if } y \leq g x \text{ then } f x \text{ else } 0$ ) = integral
{a..b} f
          by (intro Henstock_Kurzweil_Integration.integral_cong) force
          then show ?thesis
          by (rule_tac x=a in exI) (simp add: <a ≤ b)
        next
        case False
          show ?thesis
          proof (cases b < d)
            case True
              have integral {a..b} ( $\lambda x. \text{if } x \in \{x. y \leq g x\} \text{ then } f x \text{ else } 0$ ) = integral
{a..b} ( $\lambda x. 0$ )
              by (rule Henstock_Kurzweil_Integration.integral_cong) (use intv True
in fastforce)
              then show ?thesis
              using <a ≤ b> by auto
            next
            case False
              with <¬ d < a> have eq: {d..} ∩ {a..b} = {d..b} {d<..} ∩ {a..b} =
{d<..b}
              by force+
              moreover have integral {d<..b} f = integral {d..b} f
              by (rule integral_spike_set [OF empty_imp_negligible_negligible_subset
[OF negligible_sing [of d]]]) auto
              ultimately
              have integral {a..b} ( $\lambda x. \text{if } x \in \{x. y \leq g x\} \text{ then } f x \text{ else } 0$ ) = integral
{d..b} f
              unfolding integral_restrict_Int using intv by presburger
              moreover have d ∈ {a..b}
              using <¬ d < a> <a ≤ b> False by auto
              ultimately show ?thesis
              by auto
            qed
          qed
        qed
      qed (use <a ≤ b> in auto)
    then show ?thesis
    by auto
  qed

```

```

then have  $\forall k. \exists d. d \in \{a..b\} \wedge \text{integral } \{a..b\} (\lambda x. \text{if real } k / \text{real } n \leq g \ x$ 
then f x else 0) = integral {d..b} f
by meson
then obtain d where dab:  $\bigwedge k. d \ k \in \{a..b\}$ 
and deq:  $\bigwedge k::\text{nat}. \text{integral } \{a..b\} (\lambda x. \text{if } k/n \leq g \ x \text{ then } f \ x \text{ else } 0) = \text{integral}$ 
{d k..b} f
by metis
have ( $\sum k = 1..n. \text{integral } \{a..b\} (\lambda x. \text{if real } k / \text{real } n \leq g \ x \text{ then } f \ x \text{ else } 0)$ )
/R n  $\in \{m..M\}$ 
unfolding scaleR_right.sum
proof (intro conjI allI impI convex [THEN iffD1, rule_format])
show integral {a..b} ( $\lambda x a. \text{if real } k / \text{real } n \leq g \ x a \text{ then } f \ x a \text{ else } 0) \in \{m..M\}$ 
for k
by (metis (no_types, lifting) deq image_eqI int_fab dab)
qed (use <n  $\neq$  0> in auto)
then have  $\exists c. c \in \{a..b\} \wedge$ 
 $\text{integral } \{c..b\} f = \text{inverse } n \ *_R (\sum k = 1..n. \text{integral } \{a..b\} (\lambda x. \text{if } g$ 
 $x \geq \text{real } k / \text{real } n \text{ then } f \ x \text{ else } 0))$ 
by (metis (no_types, lifting) int_fab imageE)
then show ?thesis
by (simp add: sum_distrib_left if_distrib integral_sum int' flip: integral_mult_right
cong: if_cong)
qed
then obtain c where cab:  $\bigwedge n. c \ n \in \{a..b\}$ 
and c:  $\bigwedge n. \text{integral } \{c \ n..b\} f = \text{integral } \{a..b\} (\lambda x. (\sum k = 1..n. \text{if } g \ x \geq \text{real}$ 
k / real n then f x /R n else 0))
by metis
obtain d and  $\sigma :: \text{nat} \Rightarrow \text{nat}$ 
where d  $\in \{a..b\}$  and  $\sigma$ : strict_mono  $\sigma$  and d:  $(c \circ \sigma) \longrightarrow d$  and non0:
 $\bigwedge n. \sigma \ n \geq \text{Suc } 0$ 
proof -
have compact{a..b}
by auto
with cab obtain d and s0
where d  $\in \{a..b\}$  and s0: strict_mono s0 and tends:  $(c \circ s0) \longrightarrow d$ 
unfolding compact_def
using that by blast
show thesis
proof
show d  $\in \{a..b\}$ 
by fact
show strict_mono (s0  $\circ$  Suc)
using s0 by (auto simp: strict_mono_def)
show  $(c \circ (s0 \circ \text{Suc})) \longrightarrow d$ 
by (metis tends LIMSEQ_subseq_LIMSEQ Suc_less_eq comp_assoc
strict_mono_def)
show  $\bigwedge n. (s0 \circ \text{Suc}) \ n \geq \text{Suc } 0$ 
by (metis comp_apply le0 not_less_eq_eq old.nat.exhaust s0 seq_suble)
qed

```

```

qed
define  $\varphi$  where  $\varphi \equiv \lambda n x. \sum k = \text{Suc } 0.. \sigma n. \text{ if } k / (\sigma n) \leq g x \text{ then } f x /_R (\sigma n) \text{ else } 0$ 
define  $\psi$  where  $\psi \equiv \lambda n x. \sum k = \text{Suc } 0.. \sigma n. \text{ if } k / (\sigma n) \leq g x \text{ then } \text{inverse } (\sigma n) \text{ else } 0$ 
have **:  $(\lambda x. g x *_R f x) \text{ integrable\_on } \text{cbox } a b \wedge (\lambda n. \text{ integral } (\text{cbox } a b) (\varphi n)) \longrightarrow \text{ integral } (\text{cbox } a b) (\lambda x. g x *_R f x)$ 
proof (rule equiintegrable_limit)
  have †:  $((\lambda n. \lambda x. (\sum k = \text{Suc } 0.. n. \text{ if } k / n \leq g x \text{ then } \text{inverse } n *_R f x \text{ else } 0)) \text{ ' } \{\text{Suc } 0..\}) \text{ equiintegrable\_on } \{a..b\}$ 
  proof -
    have *:  $(\bigcup c::\text{real}. \{\lambda x. \text{ if } g x \geq c \text{ then } f x \text{ else } 0\}) \text{ equiintegrable\_on } \{a..b\}$ 
    using SMVT_lemma2 [OF f g] .
    show ?thesis
    apply (rule equiintegrable_on_subset [OF equiintegrable_sum_real [OF *]], clarify)
    apply (rule_tac a={Suc 0..n} in UN_I, force)
    apply (rule_tac a= $\lambda k. \text{inverse } n$  in UN_I, auto)
    apply (rule_tac x= $\lambda k x. \text{ if } \text{real } k / \text{real } n \leq g x \text{ then } f x \text{ else } 0$  in bexI)
    apply (force intro: sum.cong)+
    done
  qed
show range  $\varphi$  equiintegrable_on cbox a b
unfolding  $\varphi\_def$ 
by (auto simp: non0 intro: equiintegrable_on_subset [OF †])
show  $(\lambda n. \varphi n x) \longrightarrow g x *_R f x$ 
if  $x \in \text{cbox } a b$  for x
proof -
  have eq:  $\varphi n x = \psi n x *_R f x$  for n
  by (auto simp:  $\varphi\_def \psi\_def \text{sum\_distrib\_right if\_distrib intro: sum.cong}$ )
show ?thesis
unfolding eq
proof (rule tendsto_scaleR [OF _ tendsto_const])
  show  $(\lambda n. \psi n x) \longrightarrow g x$ 
  unfolding lim_sequentially_dist_real_def
  proof (intro allI impI)
    fix  $e :: \text{real}$ 
    assume  $e > 0$ 
    then obtain N where  $N \neq 0 \ 0 < \text{inverse } (\text{real } N) \text{ and } N: \text{inverse } (\text{real } N) < e$ 
    using real_arch_inverse by metis
    moreover have  $|\psi n x - g x| < \text{inverse } (\text{real } N) \text{ if } n \geq N$  for n
  proof -
    have  $|g x - \psi n x| < \text{inverse } (\text{real } (\sigma n))$ 
    unfolding  $\psi\_def$ 
    proof (rule level_approx [of {a..b} g])
      show  $\sigma n \neq 0$ 
      by (metis Suc_n_not_le_n non0)
    qed
  qed (use x 01 non0 in auto)
  qed

```

```

    also have ... ≤ inverse N
    using seq_suble [OF σ] ⟨N ≠ 0⟩ non0 that by (auto intro: order_trans
simp: field_split_simps)
    finally show ?thesis
    by linarith
  qed
  ultimately show ∃ N. ∀ n ≥ N. |ψ n x - g x| < e
  using less_trans by blast
  qed
  qed
  qed
  show thesis
  proof
    show a ≤ d d ≤ b
    using ⟨d ∈ {a..b}⟩ atLeastAtMost_iff by blast+
    show ((λx. g x *R f x) has_integral integral {d..b} f) {a..b}
    unfolding has_integral_iff
  proof
    show (λx. g x *R f x) integrable_on {a..b}
    using ** by simp
    show integral {a..b} (λx. g x *R f x) = integral {d..b} f
  proof (rule tendsto_unique)
    show (λn. integral {c(σ n)..b} f) ⟶ integral {a..b} (λx. g x *R f x)
    using ** by (simp add: c_φ_def)
    have continuous (at d within {a..b}) (λx. integral {x..b} f)
    using indefinite_integral_continuous_1' [OF f] ⟨d ∈ {a..b}⟩
    by (simp add: continuous_on_eq_continuous_within)
    then show (λn. integral {c(σ n)..b} f) ⟶ integral {d..b} f
    using d cab unfolding o_def
    by (simp add: continuous_within_sequentially_o_def)
  qed auto
  qed
  qed
  qed

```

theorem second_mean_value_theorem_full:

```

  fixes f :: real ⇒ real
  assumes f: f integrable_on {a..b} and a ≤ b
  and g: ∧ x y. [a ≤ x; x ≤ y; y ≤ b] ⟹ g x ≤ g y
  obtains c where c ∈ {a..b}
  and ((λx. g x * f x) has_integral (g a * integral {a..c} f + g b * integral {c..b}
f)) {a..b}
  proof -
    have gab: g a ≤ g b
    using ⟨a ≤ b⟩ g by blast
    then consider g a < g b | g a = g b
    by linarith

```

```

then show thesis
proof cases
  case 1
    define h where  $h \equiv \lambda x. \text{if } x < a \text{ then } 0 \text{ else if } b < x \text{ then } 1$ 
       $\text{else } (g\ x - g\ a) / (g\ b - g\ a)$ 
    obtain c where  $a \leq c \leq b$  and c:  $((\lambda x. h\ x *_{\mathbb{R}} f\ x) \text{ has\_integral } \text{integral } \{c..b\} f) \{a..b\}$ 
    proof (rule SMVT_lemma4 [OF  $f\ \langle a \leq b \rangle$ , of h])
      show  $h\ x \leq h\ y \wedge 0 \leq h\ x \wedge h\ x \leq 1$  if  $x \leq y$  for  $x\ y$ 
      using that gab by (auto simp: divide_simps g_h_def)
    qed
    show ?thesis
    proof
      show  $c \in \{a..b\}$ 
      using  $\langle a \leq c \rangle \langle c \leq b \rangle$  by auto
      have I:  $((\lambda x. g\ x * f\ x - g\ a * f\ x) \text{ has\_integral } (g\ b - g\ a) * \text{integral } \{c..b\} f) \{a..b\}$ 
      proof (subst has_integral_cong)
        show  $g\ x * f\ x - g\ a * f\ x = (g\ b - g\ a) * h\ x *_{\mathbb{R}} f\ x$ 
        if  $x \in \{a..b\}$  for  $x$ 
        using 1 that by (simp add: h_def field_split_simps)
        show  $((\lambda x. (g\ b - g\ a) * h\ x *_{\mathbb{R}} f\ x) \text{ has\_integral } (g\ b - g\ a) * \text{integral } \{c..b\} f) \{a..b\}$ 
        using has_integral_mult_right [OF c, of  $g\ b - g\ a$ ] .
      qed
      have II:  $((\lambda x. g\ a * f\ x) \text{ has\_integral } g\ a * \text{integral } \{a..b\} f) \{a..b\}$ 
      using has_integral_mult_right [where  $c = g\ a$ , OF integrable_integral [OF f]] .
      have  $((\lambda x. g\ x * f\ x) \text{ has\_integral } (g\ b - g\ a) * \text{integral } \{c..b\} f + g\ a * \text{integral } \{a..b\} f) \{a..b\}$ 
      using has_integral_add [OF I II] by simp
      then show  $((\lambda x. g\ x * f\ x) \text{ has\_integral } g\ a * \text{integral } \{a..c\} f + g\ b * \text{integral } \{c..b\} f) \{a..b\}$ 
      by (simp add: algebra_simps flip: integral_combine [OF  $\langle a \leq c \rangle \langle c \leq b \rangle$ ] f)
    qed
  next
    case 2
      show ?thesis
      proof
        show  $a \in \{a..b\}$ 
        by (simp add:  $\langle a \leq b \rangle$ )
        have  $((\lambda x. g\ x * f\ x) \text{ has\_integral } g\ a * \text{integral } \{a..b\} f) \{a..b\}$ 
        proof (rule has_integral_eq)
          show  $((\lambda x. g\ a * f\ x) \text{ has\_integral } g\ a * \text{integral } \{a..b\} f) \{a..b\}$ 
          using f has_integral_mult_right by blast
          show  $g\ a * f\ x = g\ x * f\ x$ 
          if  $x \in \{a..b\}$  for  $x$ 
          by (metis atLeastAtMost_iff g_less_eq_real_def not_le that 2)
        qed
      qed

```

3210

```

then show (( $\lambda x. g x * f x$ ) has_integral  $g a * \text{integral } \{a..a\} f + g b * \text{integral } \{a..b\} f$ )  $\{a..b\}$ 
by (simp add: 2)
qed
qed
qed

```

corollary *second_mean_value_theorem*:

```

fixes  $f :: \text{real} \Rightarrow \text{real}$ 
assumes  $f: f \text{ integrable\_on } \{a..b\}$  and  $a \leq b$ 
and  $g: \bigwedge x y. \llbracket a \leq x; x \leq y; y \leq b \rrbracket \Longrightarrow g x \leq g y$ 
obtains  $c$  where  $c \in \{a..b\}$ 
 $\text{integral } \{a..b\} (\lambda x. g x * f x) = g a * \text{integral } \{a..c\} f + g b * \text{integral } \{c..b\} f$ 
using second_mean_value_theorem_full [where  $g=g$ , OF assms]
by (metis (full_types) integral_unique)

```

end

9.15 Continuous Extensions of Functions

```

theory Continuous_Extension
imports Starlike
begin

```

9.15.1 Partitions of unity subordinate to locally finite open coverings

A difference from HOL Light: all summations over infinite sets equal zero, so the "support" must be made explicit in the summation below!

proposition *subordinate_partition_of_unity*:

```

fixes  $S :: 'a::\text{metric\_space}$  set
assumes  $S \subseteq \bigcup C$  and  $opC: \bigwedge T. T \in C \Longrightarrow \text{open } T$ 
and  $fin: \bigwedge x. x \in S \Longrightarrow \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in C. U \cap V \neq \{\}\}$ 
obtains  $F :: ['a \text{ set}, 'a] \Rightarrow \text{real}$ 
where  $\bigwedge U. U \in C \Longrightarrow \text{continuous\_on } S (F U) \wedge (\forall x \in S. 0 \leq F U x)$ 
and  $\bigwedge x U. \llbracket U \in C; x \in S; x \notin U \rrbracket \Longrightarrow F U x = 0$ 
and  $\bigwedge x. x \in S \Longrightarrow \text{supp\_sum } (\lambda W. F W x) C = 1$ 
and  $\bigwedge x. x \in S \Longrightarrow \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in C. \exists x \in V. F U x \neq 0\}$ 

```

proof (*cases* $\exists W. W \in C \wedge S \subseteq W$)

case *True*

then obtain W **where** $W \in C \wedge S \subseteq W$ **by** *metis*

then show *?thesis*

by (*rule_tac* $F = \lambda V x. \text{if } V = W \text{ then } 1 \text{ else } 0$ **in that**) (*auto simp: supp_sum_def support_on_def*)

next

```

case False
  have nonneg:  $0 \leq \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$  for  $x$ 
    by (simp add: supp_sum_def sum_nonneg)
  have sd_pos:  $0 < \text{setdist } \{x\} (S - V)$  if  $V \in \mathcal{C}$   $x \in S$   $x \in V$  for  $V$   $x$ 
  proof -
    have closedin (top_of_set S) (S - V)
      by (simp add: Diff_Diff_Int closedin_def opC openin_open_Int ⟨V ∈ C⟩)
    with that False setdist_pos_le [of {x} S - V]
    show ?thesis
      using setdist_gt_0_closedin by fastforce
  qed
  have ss_pos:  $0 < \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$  if  $x \in S$  for  $x$ 
  proof -
    obtain U where  $U \in \mathcal{C}$   $x \in U$  using ⟨x ∈ S⟩ ⟨S ⊆ ⋃ C⟩
      by blast
    obtain V where open V  $x \in V$  finite {U ∈ C. U ∩ V ≠ {}}
      using ⟨x ∈ S⟩ fin by blast
    then have *: finite {A ∈ C. ¬ S ⊆ A ∧ x ∉ closure (S - A)}
      using closure_def that by (blast intro: rev_finite_subset)
    have  $x \notin \text{closure } (S - U)$ 
      using ⟨U ∈ C⟩ ⟨x ∈ U⟩ opC open_Int_closure_eq_empty by fastforce
    then show ?thesis
      apply (simp add: setdist_eq_0_sing_1 supp_sum_def support_on_def)
      apply (rule ordered_comm_monoid_add_class.sum_pos2 [OF *, of U])
      using ⟨U ∈ C⟩ ⟨x ∈ U⟩ False
      apply (auto simp: sd_pos that)
      done
  qed
  define F where
     $F \equiv \lambda W x. \text{if } x \in S \text{ then } \text{setdist } \{x\} (S - W) / \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C} \text{ else } 0$ 
  show ?thesis
  proof (rule_tac F = F in that)
    have continuous_on S (F U) if  $U \in \mathcal{C}$  for  $U$ 
    proof -
      have *: continuous_on S ( $\lambda x. \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$ )
      proof (clarsimp simp add: continuous_on_eq_continuous_within)
        fix x assume  $x \in S$ 
        then obtain X where open X and  $x: x \in S \cap X$  and finX: finite {U ∈
          C. U ∩ X ≠ {}}
          using assms by blast
        then have OSX: openin (top_of_set S) (S ∩ X) by blast
        have sumeq:  $\bigwedge x. x \in S \cap X \implies$ 
           $(\sum V \mid V \in \mathcal{C} \wedge V \cap X \neq \{. \text{setdist } \{x\} (S - V))$ 
           $= \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S - V)) \mathcal{C}$ 
        apply (simp add: supp_sum_def)
        apply (rule sum_mono_neutral_right [OF finX])
        apply (auto simp: setdist_eq_0_sing_1 support_on_def subset_iff)
        apply (meson DiffI closure_subset disjoint_iff_not_equal subsetCE)
      qed
    qed
  end

```

```

done
show continuous (at x within S) ( $\lambda x. \text{supp\_sum } (\lambda V. \text{setdist } \{x\} (S -$ 
V)) C)
  apply (rule continuous_transform_within_openin
    [where  $f = \lambda x. (\text{sum } (\lambda V. \text{setdist } \{x\} (S - V)) \{V \in \mathcal{C}. V \cap X$ 
 $\neq \{\}\})$ 
      and  $S = S \cap X$ ])
  apply (rule continuous_intros continuous_at_setdist continuous_at_imp_continuous_at_within
OSX x)+
  apply (simp add: sumeq)
done
qed
show ?thesis
  apply (simp add: F_def)
  apply (rule continuous_intros *)+
  using ss_pos apply force
done
qed
moreover have  $\llbracket U \in \mathcal{C}; x \in S \rrbracket \implies 0 \leq F U x$  for  $U x$ 
  using nonneg [of x] by (simp add: F_def field_split_simps)
ultimately show  $\bigwedge U. U \in \mathcal{C} \implies \text{continuous\_on } S (F U) \wedge (\forall x \in S. 0 \leq F$ 
U x)
  by metis
next
show  $\bigwedge x U. \llbracket U \in \mathcal{C}; x \in S; x \notin U \rrbracket \implies F U x = 0$ 
  by (simp add: setdist_eq_0_sing_1 closure_def F_def)
next
show  $\text{supp\_sum } (\lambda W. F W x) \mathcal{C} = 1$  if  $x \in S$  for  $x$ 
  using that ss_pos [OF that]
  by (simp add: F_def field_split_simps supp_sum_divide_distrib [symmetric])
next
show  $\exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. \exists x \in V. F U x \neq 0\}$  if  $x \in S$ 
for  $x$ 
  using fin [OF that] that
  by (fastforce simp: setdist_eq_0_sing_1 closure_def F_def elim!: rev_finite_subset)
qed
qed

```

9.15.2 Urysohn's Lemma for Euclidean Spaces

For Euclidean spaces the proof is easy using distances.

```

lemma Urysohn_both_ne:
  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and ST:  $S \cap T = \{\}$   $S \neq \{\}$   $T \neq \{\}$   $a \neq b$ 
  obtains  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{real\_normed\_vector}$ 
  where continuous_on U f
     $\bigwedge x. x \in U \implies f x \in \text{closed\_segment } a b$ 
     $\bigwedge x. x \in U \implies (f x = a \longleftrightarrow x \in S)$ 

```



```

       $\bigwedge x. x \in U \implies (f x = b \longleftrightarrow x \in T)$ 
proof -
  have S0:  $\bigwedge x. x \in U \implies \text{setdist } \{x\} S = 0 \longleftrightarrow x \in S$ 
    using  $\langle S \neq \{\} \rangle$  US setdist_eq_0_closedin by auto
  have T0:  $\bigwedge x. x \in U \implies \text{setdist } \{x\} T = 0 \longleftrightarrow x \in T$ 
    using  $\langle T \neq \{\} \rangle$  UT setdist_eq_0_closedin by auto
  have sdpos:  $0 < \text{setdist } \{x\} S + \text{setdist } \{x\} T$  if  $x \in U$  for  $x$ 
  proof -
    have  $\neg (\text{setdist } \{x\} S = 0 \wedge \text{setdist } \{x\} T = 0)$ 
      using assms by (metis IntI empty_iff setdist_eq_0_closedin that)
    then show ?thesis
      by (metis add.left_neutral add.right_neutral add_pos_pos linorder_neqE_linordered_idom
not_le setdist_pos_le)
    qed
  define f where  $f \equiv \lambda x. a + (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T))$ 
*_R (b - a)
  show ?thesis
  proof (rule_tac f = f in that)
    show continuous_on U f
      using sdpos unfolding f_def
      by (intro continuous_intros | force)+
    show  $f x \in \text{closed\_segment } a b$  if  $x \in U$  for  $x$ 
      unfolding f_def
      apply (simp add: closed_segment_def)
      apply (rule_tac  $x = (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T))$  in exI)
      using sdpos that apply (simp add: algebra_simps)
      done
    show  $\bigwedge x. x \in U \implies (f x = a \longleftrightarrow x \in S)$ 
      using S0  $\langle a \neq b \rangle$  f_def sdpos by force
    show  $(f x = b \longleftrightarrow x \in T)$  if  $x \in U$  for  $x$ 
    proof -
      have  $f x = b \longleftrightarrow (\text{setdist } \{x\} S / (\text{setdist } \{x\} S + \text{setdist } \{x\} T)) = 1$ 
        unfolding f_def
        by (metis add_diff_cancel_left'  $\langle a \neq b \rangle$  diff_add_cancel eq_iff_diff_eq_0
scaleR_cancel_right scaleR_one)
      also have ...  $\longleftrightarrow \text{setdist } \{x\} T = 0 \wedge \text{setdist } \{x\} S \neq 0$ 
        using sdpos that
        by (simp add: field_split_simps) linarith
      also have ...  $\longleftrightarrow x \in T$ 
        using  $\langle S \neq \{\} \rangle$   $\langle T \neq \{\} \rangle$   $\langle S \cap T = \{\} \rangle$  that
        by (force simp: S0 T0)
      finally show ?thesis .
    qed
  qed
qed
lemma Urysohn_local_strong_aux:
  assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T

```

```

    and  $S \cap T = \{ \}$   $a \neq b$   $S \neq \{ \}$ 
obtains  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
where  $continuous\_on\ U\ f$ 
 $\bigwedge x. x \in U \Longrightarrow f\ x \in closed\_segment\ a\ b$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = a \longleftrightarrow x \in S)$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = b \longleftrightarrow x \in T)$ 
proof (cases  $T = \{ \}$ )
case True show ?thesis
proof (cases  $S = U$ )
case True with  $\langle T = \{ \} \rangle \langle a \neq b \rangle$  show ?thesis
by (rule_tac  $f = \lambda x. a$  in that) (auto)
next
case False
with  $US\ closedin\_subset$  obtain  $c$  where  $c: c \in U\ c \notin S$ 
by fastforce
obtain  $f$  where  $f: continuous\_on\ U\ f$ 
 $\bigwedge x. x \in U \Longrightarrow f\ x \in closed\_segment\ a\ (midpoint\ a\ b)$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = midpoint\ a\ b \longleftrightarrow x = c)$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = a \longleftrightarrow x \in S)$ 
apply (rule Urysohn_both_ne [of  $U\ S\ \{c\}$   $a\ midpoint\ a\ b$ ])
using  $c \in S \neq \{ \}$  assms apply simp_all
apply (metis midpoint_eq_endpoint)
done
show ?thesis
apply (rule_tac  $f=f$  in that)
using  $\langle S \neq \{ \} \rangle \langle T = \{ \} \rangle f \langle a \neq b \rangle$ 
apply simp_all
apply (metis (no_types) closed_segment_commute csegment_midpoint_subset
midpoint_sym subset_iff)
apply (metis closed_segment_commute midpoint_sym notin_segment_midpoint)
done
qed
next
case False
show ?thesis
using Urysohn_both_ne [OF  $US\ UT\ \langle S \cap T = \{ \} \rangle \langle S \neq \{ \} \rangle \langle T \neq \{ \} \rangle \langle a \neq b \rangle$ ] that
by blast
qed

proposition Urysohn_local_strong:
assumes  $US: closedin\ (top\_of\_set\ U)\ S$ 
and  $UT: closedin\ (top\_of\_set\ U)\ T$ 
and  $S \cap T = \{ \}$   $a \neq b$ 
obtains  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
where  $continuous\_on\ U\ f$ 
 $\bigwedge x. x \in U \Longrightarrow f\ x \in closed\_segment\ a\ b$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = a \longleftrightarrow x \in S)$ 
 $\bigwedge x. x \in U \Longrightarrow (f\ x = b \longleftrightarrow x \in T)$ 

```

```

proof (cases  $S = \{\}$ )
  case True show ?thesis
  proof (cases  $T = \{\}$ )
    case True show ?thesis
    proof (rule_tac  $f = \lambda x. \text{midpoint } a \ b$  in that)
      show continuous_on  $U$  ( $\lambda x. \text{midpoint } a \ b$ )
        by (intro continuous_intros)
      show  $\text{midpoint } a \ b \in \text{closed\_segment } a \ b$ 
        using csegment_midpoint_subset by blast
      show ( $\text{midpoint } a \ b = a$ ) = ( $x \in S$ ) for  $x$ 
        using  $\langle S = \{\} \rangle \langle a \neq b \rangle$  by simp
      show ( $\text{midpoint } a \ b = b$ ) = ( $x \in T$ ) for  $x$ 
        using  $\langle T = \{\} \rangle \langle a \neq b \rangle$  by simp
    qed
  next
  case False
  with Urysohn_local_strong_aux [OF UT US] assms show ?thesis
    by (smt (verit) True closed_segment_commute inf_bot_right that)
  qed
next
  case False
  with Urysohn_local_strong_aux [OF assms] show ?thesis
    using that by blast
qed

```

```

lemma Urysohn_local:
  assumes US: closedin (top_of_set  $U$ )  $S$ 
    and UT: closedin (top_of_set  $U$ )  $T$ 
    and  $S \cap T = \{\}$ 
  obtains  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
    where continuous_on  $U$   $f$ 
       $\bigwedge x. x \in U \Longrightarrow f \ x \in \text{closed\_segment } a \ b$ 
       $\bigwedge x. x \in S \Longrightarrow f \ x = a$ 
       $\bigwedge x. x \in T \Longrightarrow f \ x = b$ 

```

```

proof (cases  $a = b$ )
  case True then show ?thesis
    by (rule_tac  $f = \lambda x. b$  in that) (auto)
  next
  case False
  with Urysohn_local_strong [OF assms] show ?thesis
    by (smt (verit) US UT closedin_imp_subset subset_eq that)
qed

```

```

lemma Urysohn_strong:
  assumes US: closed  $S$ 
    and UT: closed  $T$ 
    and  $S \cap T = \{\}$   $a \neq b$ 
  obtains  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
    where continuous_on  $UNIV$   $f$ 

```

$$\begin{aligned} &\bigwedge x. f x \in \text{closed_segment } a \ b \\ &\bigwedge x. f x = a \longleftrightarrow x \in S \\ &\bigwedge x. f x = b \longleftrightarrow x \in T \end{aligned}$$

using *assms* by (auto intro: *Urysohn_local_strong* [of *UNIV S T*])

proposition *Urysohn*:

assumes *US*: closed *S*

and *UT*: closed *T*

and $S \cap T = \{\}$

obtains $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$

where *continuous_on UNIV f*

$$\bigwedge x. f x \in \text{closed_segment } a \ b$$

$$\bigwedge x. x \in S \Longrightarrow f x = a$$

$$\bigwedge x. x \in T \Longrightarrow f x = b$$

using *assms* by (auto intro: *Urysohn_local* [of *UNIV S T a b*])

9.15.3 Dugundji's Extension Theorem and Tietze Variants

See [3].

theorem *Dugundji*:

fixes $f :: 'a::\{\text{metric_space,second_countable_topology}\} \Rightarrow 'b::\text{real_inner}$

assumes *convex C* $C \neq \{\}$

and *cloin*: closedin (*top_of_set U*) *S*

and *contf*: *continuous_on S f* and $f ' S \subseteq C$

obtains *g* where *continuous_on U g* $g ' U \subseteq C$

$$\bigwedge x. x \in S \Longrightarrow g x = f x$$

proof (cases $S = \{\}$)

case *True* show *thesis*

proof

show *continuous_on U* ($\lambda x. \text{SOME } y. y \in C$)

by (*rule continuous_intros*)

show ($\lambda x. \text{SOME } y. y \in C$) ' $U \subseteq C$

by (*simp add*: $\langle C \neq \{\} \rangle$ *image_subsetI some_in_eq*)

qed (use *True in auto*)

next

case *False*

then have *sd_pos*: $\bigwedge x. \llbracket x \in U; x \notin S \rrbracket \Longrightarrow 0 < \text{setdist } \{x\} S$

using *setdist_eq_0_closedin* [OF *cloin*] *le_less setdist_pos_le* by *fastforce*

define \mathcal{B} where $\mathcal{B} = \{\text{ball } x (\text{setdist } \{x\} S / 2) \mid x. x \in U - S\}$

have [*simp*]: $\bigwedge T. T \in \mathcal{B} \Longrightarrow \text{open } T$

by (*auto simp*: $\mathcal{B_def}$)

have *USS*: $U - S \subseteq \bigcup \mathcal{B}$

by (*auto simp*: *sd_pos* $\mathcal{B_def}$)

obtain \mathcal{C} where *USSub*: $U - S \subseteq \bigcup \mathcal{C}$

and *nbrhd*: $\bigwedge U. U \in \mathcal{C} \Longrightarrow \text{open } U \wedge (\exists T. T \in \mathcal{B} \wedge U \subseteq T)$

and *fin*: $\bigwedge x. x \in U - S \Longrightarrow \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U. U \in \mathcal{C} \wedge U$

$\cap V \neq \{\}$

by (*rule paracompact* [OF *USS*]) *auto*

have $\exists v a. v \in U \wedge v \notin S \wedge a \in S \wedge$

```

       $T \subseteq \text{ball } v (\text{setdist } \{v\} S / 2) \wedge$ 
       $\text{dist } v a \leq 2 * \text{setdist } \{v\} S \text{ if } T \in \mathcal{C} \text{ for } T$ 
proof –
  obtain  $v$  where  $v: T \subseteq \text{ball } v (\text{setdist } \{v\} S / 2) v \in U v \notin S$ 
    using  $\langle T \in \mathcal{C} \rangle \text{ nbrhd}$  by (force simp:  $\mathcal{B}$ _def)
  then obtain  $a$  where  $a \in S \text{ dist } v a < 2 * \text{setdist } \{v\} S$ 
    using  $\text{setdist\_ltE}$  [of  $\{v\} S 2 * \text{setdist } \{v\} S$ ]
    using  $\text{False sd\_pos}$  by force
  with  $v$  show ?thesis
    by force
qed
then obtain  $\mathcal{V} \mathcal{A}$  where
   $\text{VA: } \bigwedge T. T \in \mathcal{C} \implies \mathcal{V} T \in U \wedge \mathcal{V} T \notin S \wedge \mathcal{A} T \in S \wedge$ 
     $T \subseteq \text{ball } (\mathcal{V} T) (\text{setdist } \{\mathcal{V} T\} S / 2) \wedge$ 
     $\text{dist } (\mathcal{V} T) (\mathcal{A} T) \leq 2 * \text{setdist } \{\mathcal{V} T\} S$ 
  by metis
have  $\text{sdle: setdist } \{\mathcal{V} T\} S \leq 2 * \text{setdist } \{v\} S \text{ if } T \in \mathcal{C} v \in T \text{ for } T v$ 
  using  $\text{setdist\_Lipschitz}$  [of  $\mathcal{V} T S v$ ]  $\text{VA}$  [OF  $\langle T \in \mathcal{C} \rangle \langle v \in T \rangle$ ] by auto
have  $\text{d6: dist } a (\mathcal{A} T) \leq 6 * \text{dist } a v \text{ if } T \in \mathcal{C} v \in T a \in S \text{ for } T v a$ 
proof –
  have  $\text{dist } (\mathcal{V} T) v < \text{setdist } \{\mathcal{V} T\} S / 2$ 
    using that  $\text{VA mem\_ball}$  by blast
  also have  $\dots \leq \text{setdist } \{v\} S$ 
    using  $\text{sdle}$  [OF  $\langle T \in \mathcal{C} \rangle \langle v \in T \rangle$ ] by simp
  also have  $vS: \text{setdist } \{v\} S \leq \text{dist } a v$ 
    by (simp add: setdist_le_dist setdist_sym  $\langle a \in S \rangle$ )
  finally have  $\text{VTV: dist } (\mathcal{V} T) v < \text{dist } a v .$ 
  have  $\text{VTS: setdist } \{\mathcal{V} T\} S \leq 2 * \text{dist } a v$ 
    using sdle that  $vS$  by force
  have  $\text{dist } a (\mathcal{A} T) \leq \text{dist } a v + \text{dist } v (\mathcal{V} T) + \text{dist } (\mathcal{V} T) (\mathcal{A} T)$ 
    by (metis add.commute add_le_cancel_left dist_commute dist_triangle2
dist_triangle_le)
  also have  $\dots \leq \text{dist } a v + \text{dist } a v + \text{dist } (\mathcal{V} T) (\mathcal{A} T)$ 
    using  $\text{VTV}$  by (simp add: dist_commute)
  also have  $\dots \leq 2 * \text{dist } a v + 2 * \text{setdist } \{\mathcal{V} T\} S$ 
    using  $\text{VA}$  [OF  $\langle T \in \mathcal{C} \rangle$ ] by auto
  finally show ?thesis
    using  $\text{VTS}$  by linarith
qed
obtain  $H :: ['a \text{ set}, 'a] \Rightarrow \text{real}$ 
  where  $\text{Hcont: } \bigwedge Z. Z \in \mathcal{C} \implies \text{continuous\_on } (U-S) (H Z)$ 
  and  $\text{Hge0: } \bigwedge Z x. \llbracket Z \in \mathcal{C}; x \in U-S \rrbracket \implies 0 \leq H Z x$ 
  and  $\text{Heq0: } \bigwedge x Z. \llbracket Z \in \mathcal{C}; x \in U-S; x \notin Z \rrbracket \implies H Z x = 0$ 
  and  $\text{H1: } \bigwedge x. x \in U-S \implies \text{supp\_sum } (\lambda W. H W x) \mathcal{C} = 1$ 
  and  $\text{Hfin: } \bigwedge x. x \in U-S \implies \exists V. \text{open } V \wedge x \in V \wedge \text{finite } \{U \in \mathcal{C}. \exists x \in V. H U x \neq 0\}$ 
  apply (rule subordinate_partition_of_unity [OF  $\text{USsub\_fin}$ ])
  using nbrhd by auto
define  $g$  where  $g \equiv \lambda x. \text{if } x \in S \text{ then } f x \text{ else } \text{supp\_sum } (\lambda T. H T x *_{\mathbb{R}} f(\mathcal{A}$ 

```

```

T)) C
show ?thesis
proof (rule that)
  show continuous_on U g
  proof (clarsimp simp: continuous_on_eq_continuous_within)
    fix a assume a ∈ U
    show continuous (at a within U) g
    proof (cases a ∈ S)
      case True show ?thesis
        proof (clarsimp simp add: continuous_within_topological)
          fix W
          assume open W g a ∈ W
          then obtain e where 0 < e and e: ball (f a) e ⊆ W
            using openE True g_def by auto
          have continuous (at a within S) f
            using True contf continuous_on_eq_continuous_within by blast
          then obtain d where 0 < d
            and d: ∧x. [x ∈ S; dist x a < d] ⇒ dist (f x) (f a) < e
            using continuous_within_eps_delta ⟨0 < e⟩ by force
          have g y ∈ ball (f a) e if y ∈ U and y: y ∈ ball a (d / 6) for y
            proof (cases y ∈ S)
              case True
                then have dist (f a) (f y) < e
                  by (metis ball_divide_subset_numeral dist_commute in_mono mem_ball
y d)
            then show ?thesis
              by (simp add: True g_def)
        next
          case False
            have *: dist (f (A T)) (f a) < e if T ∈ C H T y ≠ 0 for T
              proof -
                have y ∈ T
                  using Heq0 that False ⟨y ∈ U⟩ by blast
                have dist (A T) a < d
                  using d6 [OF ⟨T ∈ C⟩ ⟨y ∈ T⟩ ⟨a ∈ S⟩] y
                  by (simp add: dist_commute mult.commute)
                then show ?thesis
                  using VA [OF ⟨T ∈ C⟩] by (auto simp: d)
            qed
          have supp_sum (λT. H T y *_R f (A T)) C ∈ ball (f a) e
            apply (rule convex_supp_sum [OF convex_ball])
            apply (simp_all add: False H1 Hge0 ⟨y ∈ U⟩)
            by (metis dist_commute *)
          then show ?thesis
            by (simp add: False g_def)
        qed
      then show ∃ A. open A ∧ a ∈ A ∧ (∀ y ∈ U. y ∈ A → g y ∈ W)
        apply (rule_tac x = ball a (d / 6) in exI)
        using e ⟨0 < d⟩ by fastforce

```

```

qed
next
case False
obtain N where N: open N a ∈ N
  and finN: finite {U ∈ C. ∃ a ∈ N. H U a ≠ 0}
  using Hfin False ⟨a ∈ U⟩ by auto
have oUS: openin (top_of_set U) (U - S)
  using cloin by (simp add: openin_diff)
have HcontU: continuous (at a within U) (H T) if T ∈ C for T
  using Hcont [OF ⟨T ∈ C⟩] False ⟨a ∈ U⟩ ⟨T ∈ C⟩
apply (simp add: continuous_on_eq_continuous_within continuous_within)
  apply (rule Lim_transform_within_set)
  using oUS
  apply (force simp: eventually_at openin_contains_ball dist_commute
dest!: bspec)+
  done
show ?thesis
proof (rule continuous_transform_within_openin [OF _ oUS])
show continuous (at a within U) (λx. supp_sum (λT. H T x *_R f) (A T))
C)
proof (rule continuous_transform_within_openin)
show continuous (at a within U)
  (λx. ∑ T ∈ {U ∈ C. ∃ x ∈ N. H U x ≠ 0}. H T x *_R f) (A T))
  by (force intro: continuous_intros HcontU)+
next
show openin (top_of_set U) ((U - S) ∩ N)
  using N oUS openin_trans by blast
next
show a ∈ (U - S) ∩ N using False ⟨a ∈ U⟩ N by blast
next
show ∧x. x ∈ (U - S) ∩ N ⇒
  (∑ T ∈ {U ∈ C. ∃ x ∈ N. H U x ≠ 0}. H T x *_R f) (A T))
  = supp_sum (λT. H T x *_R f) (A T)) C
  by (auto simp: supp_sum_def support_on_def
intro: sum.mono_neutral_right [OF finN])
qed
next
show a ∈ U - S using False ⟨a ∈ U⟩ by blast
next
show ∧x. x ∈ U - S ⇒ supp_sum (λT. H T x *_R f) (A T)) C = g x
  by (simp add: g_def)
qed
qed
show g ' U ⊆ C
  using ⟨f ' S ⊆ C⟩ VA
  by (fastforce simp: g_def Hge0 intro!: convex_supp_sum [OF ⟨convex C⟩]
H1)
show ∧x. x ∈ S ⇒ g x = f x

```

```

    by (simp add: g_def)
  qed
qed

```

corollary *Tietze*:

```

fixes f :: 'a::{metric_space,second_countable_topology}  $\Rightarrow$  'b::real_inner
assumes continuous_on S f
  and closedin (top_of_set U) S
  and 0  $\leq$  B
  and  $\bigwedge x. x \in S \implies \text{norm}(f x) \leq B$ 
obtains g where continuous_on U g  $\bigwedge x. x \in S \implies g x = f x$ 
   $\bigwedge x. x \in U \implies \text{norm}(g x) \leq B$ 
using assms by (auto simp: image_subset_iff intro: Dugundji [of cball 0 B U S f])

```

corollary *Tietze_closed_interval*:

```

fixes f :: 'a::{metric_space,second_countable_topology}  $\Rightarrow$  'b::euclidean_space
assumes continuous_on S f
  and closedin (top_of_set U) S
  and cbox a b  $\neq$  {}
  and  $\bigwedge x. x \in S \implies f x \in \text{cbox } a \text{ } b$ 
obtains g where continuous_on U g  $\bigwedge x. x \in S \implies g x = f x$ 
   $\bigwedge x. x \in U \implies g x \in \text{cbox } a \text{ } b$ 
apply (rule Dugundji [of cbox a b U S f])
using assms by auto

```

corollary *Tietze_closed_interval_1*:

```

fixes f :: 'a::{metric_space,second_countable_topology}  $\Rightarrow$  real
assumes continuous_on S f
  and closedin (top_of_set U) S
  and a  $\leq$  b
  and  $\bigwedge x. x \in S \implies f x \in \text{cbox } a \text{ } b$ 
obtains g where continuous_on U g  $\bigwedge x. x \in S \implies g x = f x$ 
   $\bigwedge x. x \in U \implies g x \in \text{cbox } a \text{ } b$ 
apply (rule Dugundji [of cbox a b U S f])
using assms by (auto simp: image_subset_iff)

```

corollary *Tietze_open_interval*:

```

fixes f :: 'a::{metric_space,second_countable_topology}  $\Rightarrow$  'b::euclidean_space
assumes continuous_on S f
  and closedin (top_of_set U) S
  and box a b  $\neq$  {}
  and  $\bigwedge x. x \in S \implies f x \in \text{box } a \text{ } b$ 
obtains g where continuous_on U g  $\bigwedge x. x \in S \implies g x = f x$ 
   $\bigwedge x. x \in U \implies g x \in \text{box } a \text{ } b$ 
apply (rule Dugundji [of box a b U S f])
using assms by auto

```


corollary *Tietze_open_interval_1*:
fixes $f :: 'a::\{\text{metric_space,second_countable_topology}\} \Rightarrow \text{real}$
assumes *continuous_on S f*
and *closedin (top_of_set U) S*
and $a < b$
and $\text{no: } \bigwedge x. x \in S \implies f x \in \text{box } a b$
obtains g **where** *continuous_on U g* $\bigwedge x. x \in S \implies g x = f x$
 $\bigwedge x. x \in U \implies g x \in \text{box } a b$
apply (*rule Dugundji [of box a b U S f]*)
using *assms* **by** (*auto simp: image_subset_iff*)

corollary *Tietze_unbounded*:
fixes $f :: 'a::\{\text{metric_space,second_countable_topology}\} \Rightarrow 'b::\text{real_inner}$
assumes *continuous_on S f*
and *closedin (top_of_set U) S*
obtains g **where** *continuous_on U g* $\bigwedge x. x \in S \implies g x = f x$
apply (*rule Dugundji [of UNIV U S f]*)
using *assms* **by** *auto*

end

9.16 Equivalence Between Classical Borel Measurability and HOL Light's

theory *Equivalence_Measurable_On_Borel*
imports *Equivalence_Lebesgue_Henstock_Integration Improper_Integral Continuous_Extension*
begin

9.16.1 Austin's Lemma

lemma *Austin_Lemma*:
fixes $\mathcal{D} :: 'a::\text{euclidean_space}$ *set set*
assumes *finite D* **and** $\mathcal{D}: \bigwedge D. D \in \mathcal{D} \implies \exists k a b. D = \text{cbox } a b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$
obtains \mathcal{C} **where** $\mathcal{C} \subseteq \mathcal{D}$ *pairwise disjoint C*
 $\text{measure lebesgue } (\bigcup \mathcal{C}) \geq \text{measure lebesgue } (\bigcup \mathcal{D}) / 3^{\wedge} (\text{DIM}('a))$
using *assms*
proof (*induction card D arbitrary: D thesis rule: less_induct*)
case *less*
show *?case*
proof (*cases D = {}*)
case *True*
then show *thesis*
using *less* **by** *auto*
next
case *False*
then have $\text{Max } (\text{Sigma_Algebra.measure lebesgue } ' \mathcal{D}) \in \text{Sigma_Algebra.measure}$

```

lebesgue 'D
  using Max_in_finite_imageI ⟨finite D⟩ by blast
  then obtain D where D ∈ D and measure lebesgue D = Max (measure
lebesgue 'D)
  by auto
  then have D:  $\bigwedge C. C \in \mathcal{D} \implies \text{measure lebesgue } C \leq \text{measure lebesgue } D$ 
  by (simp add: ⟨finite D⟩)
  let ?E = {C. C ∈ D - {D} ∧ disjoint C D}
  obtain D' where D'sub: D' ⊆ ?E and D'dis: pairwise disjoint D'
  and D'm: measure lebesgue (⋃ D') ≥ measure lebesgue (⋃ ?E) / 3 ^ (DIM('a))
  proof (rule less.hyphs)
    have *: ?E ⊆ D
      using ⟨D ∈ D⟩ by auto
    then show card ?E < card D finite ?E
      by (auto simp: ⟨finite D⟩ psubset_card_mono)
    show  $\exists k a b. D = \text{cbox } a b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$  if D ∈ ?E for D
      using less.premis(3) that by auto
  qed
  then have [simp]:  $\bigcup D' - D = \bigcup D'$ 
  by (auto simp: disjoint_iff)
  show ?thesis
  proof (rule less.premis)
    show insert D D' ⊆ D
      using D'sub ⟨D ∈ D⟩ by blast
    show disjoint (insert D D')
      using D'dis D'sub by (fastforce simp add: pairwise_def disjoint_sym)
    obtain a3 b3 where m3: content (cbox a3 b3) = 3 ^ DIM('a) * measure
lebesgue D
    and sub3:  $\bigwedge C. [C \in \mathcal{D}; \neg \text{disjoint } C D] \implies C \subseteq \text{cbox } a3 b3$ 
    proof -
      obtain k a b where ab: D = cbox a b and k:  $\bigwedge i. i \in \text{Basis} \implies b \cdot i - a \cdot i$ 
      = k
      using less.premis ⟨D ∈ D⟩ by meson
      then have eqk:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i \leq b \cdot i \iff k \geq 0$ 
      by force
      show thesis
      proof
        let ?a = (a + b) /R 2 - (3/2) *R (b - a)
        let ?b = (a + b) /R 2 + (3/2) *R (b - a)
        have eq:  $(\prod i \in \text{Basis}. b \cdot i * 3 - a \cdot i * 3) = (\prod i \in \text{Basis}. b \cdot i - a \cdot i) *$ 
3 ^ DIM('a)
        by (simp add: comm_monoid_mult_class.prod.distrib flip: left_diff_distrib
inner_diff_left)
        show content (cbox ?a ?b) = 3 ^ DIM('a) * measure lebesgue D
          by (simp add: content_cbox_if box_eq_empty algebra_simps eq ab k)
        show C ⊆ cbox ?a ?b if C ∈ D and CD:  $\neg \text{disjoint } C D$  for C
        proof -
          obtain k' a' b' where ab': C = cbox a' b' and k':  $\bigwedge i. i \in \text{Basis} \implies$ 
b'·i - a'·i = k'

```

```

    using less.premis ⟨C ∈ D⟩ by meson
  then have eqk':  $\bigwedge i. i \in \text{Basis} \implies a' \cdot i \leq b' \cdot i \iff k' \geq 0$ 
    by force
  show ?thesis
  proof (clarsimp simp add: disjoint_interval disjnt_def ab ab' not_less
subset_box algebra_simps)
    show  $a \cdot i * 2 \leq a' \cdot i + b \cdot i \wedge a \cdot i + b' \cdot i \leq b \cdot i * 2$ 
      if * [rule_format]:  $\forall j \in \text{Basis}. a' \cdot j \leq b' \cdot j$  and  $i \in \text{Basis}$  for i
    proof -
      have  $a' \cdot i \leq b' \cdot i \wedge a \cdot i \leq b \cdot i \wedge a \cdot i \leq b' \cdot i \wedge a' \cdot i \leq b \cdot i$ 
        using ⟨i ∈ Basis⟩ CD by (simp_all add: disjoint_interval disjnt_def
ab ab' not_less)
      then show ?thesis
        using D [OF ⟨C ∈ D⟩] ⟨i ∈ Basis⟩
        apply (simp add: ab ab' k k' eqk eqk' content_cbox_cases)
        using k k' by fastforce
    qed
  qed
qed
qed
qed
qed
have Dlm:  $\bigwedge D. D \in \mathcal{D} \implies D \in \text{lmeasurable}$ 
  using less.premis(3) by blast
have measure_lebesgue ( $\bigcup \mathcal{D}$ )  $\leq$  measure_lebesgue (cbox a3 b3  $\cup$  ( $\bigcup \mathcal{D} - \text{cbox}$ 
a3 b3))
  proof (rule measure_mono_fmeasurable)
    show  $\bigcup \mathcal{D} \in \text{sets lebesgue}$ 
      using Dlm ⟨finite D⟩ by blast
    show  $\text{cbox } a3 \text{ } b3 \cup (\bigcup \mathcal{D} - \text{cbox } a3 \text{ } b3) \in \text{lmeasurable}$ 
      by (simp add: Dlm fmeasurable.Un fmeasurable.finite_Union less.premis(2)
subset_eq)
    qed auto
  also have ... = content (cbox a3 b3) + measure_lebesgue ( $\bigcup \mathcal{D} - \text{cbox } a3$ 
b3)
    by (simp add: Dlm fmeasurable.finite_Union less.premis(2) measure_Un2
subsetI)
  also have ...  $\leq$  (measure_lebesgue D + measure_lebesgue ( $\bigcup \mathcal{D}'$ )) * 3 ^
DIM('a)
  proof -
    have ( $\bigcup \mathcal{D} - \text{cbox } a3 \text{ } b3$ )  $\subseteq$   $\bigcup ?\mathcal{E}$ 
      using sub3 by fastforce
    then have measure_lebesgue ( $\bigcup \mathcal{D} - \text{cbox } a3 \text{ } b3$ )  $\leq$  measure_lebesgue ( $\bigcup ?\mathcal{E}$ )
    proof (rule measure_mono_fmeasurable)
      show  $\bigcup \mathcal{D} - \text{cbox } a3 \text{ } b3 \in \text{sets lebesgue}$ 
        by (simp add: Dlm fmeasurableD less.premis(2) sets.Diff sets.finite_Union
subsetI)
      show  $\bigcup \{C \in \mathcal{D} - \{D\}. \text{disjnt } C \text{ } D\} \in \text{lmeasurable}$ 
        using Dlm less.premis(2) by auto
    qed
  qed

```

```

then have measure lebesgue ( $\bigcup \mathcal{D} - \text{cbox } a3 \text{ } b3$ ) /  $3 \wedge \text{DIM}('a) \leq \text{measure lebesgue } (\bigcup \mathcal{D}')$ 
using  $\mathcal{D}'m$  by (simp add: field_split_simps)
then show ?thesis
by (simp add: m3 field_simps)
qed
also have  $\dots \leq \text{measure lebesgue } (\bigcup (\text{insert } D \mathcal{D}')) * 3 \wedge \text{DIM}('a)$ 
proof (simp add: Dlm ⟨D ∈ D⟩)
show measure lebesgue  $D + \text{measure lebesgue } (\bigcup \mathcal{D}') \leq \text{measure lebesgue } (D \cup \bigcup \mathcal{D}')$ 
proof (subst measure_Un2)
show  $\bigcup \mathcal{D}' \in \text{lmeasurable}$ 
by (meson Dlm ⟨insert D D' ⊆ D⟩ fmeasurable.finite_Union less.prem2 finite_subset subset_eq subset_insertI)
show measure lebesgue  $D + \text{measure lebesgue } (\bigcup \mathcal{D}') \leq \text{measure lebesgue } D + \text{measure lebesgue } (\bigcup \mathcal{D}' - D)$ 
using  $\langle \text{insert } D \mathcal{D}' \subseteq \mathcal{D} \rangle$  infinite_super less.prem2 by force
qed (simp add: Dlm ⟨D ∈ D⟩)
qed
finally show measure lebesgue  $(\bigcup \mathcal{D}) / 3 \wedge \text{DIM}('a) \leq \text{measure lebesgue } (\bigcup (\text{insert } D \mathcal{D}'))$ 
by (simp add: field_split_simps)
qed
qed
qed

```

9.16.2 A differentiability-like property of the indefinite integral.

proposition *integrable_ccontinuous_explicit:*

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$

assumes $\bigwedge a \ b::'a. f \text{ integrable_on } \text{cbox } a \ b$

obtains N **where**

negligible N

$\bigwedge x \ e. \llbracket x \notin N; 0 < e \rrbracket \implies$

$\exists d > 0. \forall h. 0 < h \wedge h < d \longrightarrow$

$\text{norm}(\text{integral } (\text{cbox } x \ (x + h *_{\mathbb{R}} \text{One})) f /_{\mathbb{R}} h \wedge \text{DIM}('a) - f$

$x) < e$

proof –

define BOX **where** $BOX \equiv \lambda h. \lambda x::'a. \text{cbox } x \ (x + h *_{\mathbb{R}} \text{One})$

define $BOX2$ **where** $BOX2 \equiv \lambda h. \lambda x::'a. \text{cbox } (x - h *_{\mathbb{R}} \text{One}) \ (x + h *_{\mathbb{R}} \text{One})$

define i **where** $i \equiv \lambda h \ x. \text{integral } (BOX \ h \ x) f /_{\mathbb{R}} h \wedge \text{DIM}('a)$

define Ψ **where** $\Psi \equiv \lambda x \ r. \forall d > 0. \exists h. 0 < h \wedge h < d \wedge r \leq \text{norm}(i \ h \ x - f \ x)$

let $?N = \{x. \exists e > 0. \Psi \ x \ e\}$

have $\exists N. \text{negligible } N \wedge (\forall x \ e. x \notin N \wedge 0 < e \longrightarrow \neg \Psi \ x \ e)$

proof (*rule exI ; intro conjI allI impI*)

let $?M = \bigcup n. \{x. \Psi \ x \ (\text{inverse}(\text{real } n + 1))\}$

have *negligible* $(\{x. \Psi \ x \ \mu\} \cap \text{cbox } a \ b)$

if $\mu > 0$ **for** $a \ b \ \mu$

```

proof (cases negligible(cbox a b))
  case True
  then show ?thesis
    by (simp add: negligible_Int)
next
  case False
  then have box a b  $\neq$  {}
    by (simp add: negligible_interval)
  then have ab:  $\bigwedge i. i \in \text{Basis} \implies a \cdot i < b \cdot i$ 
    by (simp add: box_ne_empty)
  show ?thesis
    unfolding negligible_outer_le
  proof (intro allI impI)
    fix e::real
    let ?ee = (e *  $\mu$ ) / 2 / 6 ^ (DIM('a))
    assume e > 0
    then have gt0: ?ee > 0
      using  $\langle \mu > 0 \rangle$  by auto
    have f': f integrable_on cbox (a - One) (b + One)
      using assms by blast
    obtain  $\gamma$  where gauge  $\gamma$ 
      and  $\gamma$ :  $\bigwedge p. \llbracket p \text{ tagged\_partial\_division\_of } (cbox (a - One) (b + One));$ 
 $\gamma \text{ fine } p \rrbracket$ 
       $\implies (\sum (x, k) \in p. \text{norm } (\text{content } k *_{\mathbb{R}} f x - \text{integral } k f)) < ?ee$ 
    using Henstock_lemma [OF f' gt0] that by auto
    let ?E = {x. x  $\in$  cbox a b  $\wedge$   $\Psi$  x  $\mu$ }
    have  $\exists h > 0. \text{BOX } h x \subseteq \gamma x \wedge$ 
       $\text{BOX } h x \subseteq \text{cbox } (a - \text{One}) (b + \text{One}) \wedge \mu \leq \text{norm } (i h x - f x)$ 
    if x  $\in$  cbox a b  $\Psi$  x  $\mu$  for x
    proof -
      obtain d where d > 0 and d: ball x d  $\subseteq$   $\gamma$  x
      using gaugeD [OF  $\langle \text{gauge } \gamma \rangle$ , of x] openE by blast
      then obtain h where 0 < h h < 1 and hless: h < d / real DIM('a)
        and mule:  $\mu \leq \text{norm } (i h x - f x)$ 
      using  $\langle \Psi x \mu \rangle$  [unfolded  $\Psi$ _def, rule_format, of min 1 (d / DIM('a))]
      by auto
    show ?thesis
    proof (intro exI conjI)
      show 0 < h  $\mu \leq \text{norm } (i h x - f x)$  by fact+
      have BOX h x  $\subseteq$  ball x d
      proof (clarsimp simp: BOX_def mem_box dist_norm algebra_simps)
        fix y
        assume  $\forall i \in \text{Basis}. x \cdot i \leq y \cdot i \wedge y \cdot i \leq h + x \cdot i$ 
        then have lt:  $|(x - y) \cdot i| < d / \text{real } \text{DIM}('a)$  if i  $\in$  Basis for i
          using hless that by (force simp: inner_diff_left)
        have norm (x - y)  $\leq (\sum i \in \text{Basis}. |(x - y) \cdot i|)$ 
          using norm_le_l1 by blast
        also have ... < d
          using sum_bounded_above_strict [of Basis  $\lambda i. |(x - y) \cdot i|$  d /

```

```

DIM('a), OF lt]
  by auto
  finally show norm (x - y) < d .
qed
with d show BOX h x ⊆ γ x
  by blast
show BOX h x ⊆ cbox (a - One) (b + One)
  using that ⟨h < 1⟩
by (force simp: BOX_def mem_box algebra_simps intro: subset_box_imp)
qed
qed
then obtain η where h0: ⋀x. x ∈ ?E ⇒ η x > 0
  and BOX_γ: ⋀x. x ∈ ?E ⇒ BOX (η x) x ⊆ γ x
  and ⋀x. x ∈ ?E ⇒ BOX (η x) x ⊆ cbox (a - One) (b + One) ∧ μ ≤
norm (i (η x) x - f x)
  by simp metis
then have BOX_cbox: ⋀x. x ∈ ?E ⇒ BOX (η x) x ⊆ cbox (a - One) (b
+ One)
  and μ_le: ⋀x. x ∈ ?E ⇒ μ ≤ norm (i (η x) x - f x)
  by blast+
define γ' where γ' ≡ λx. if x ∈ cbox a b ∧ Ψ x μ then ball x (η x) else γ x
have gauge γ'
  using ⟨gauge γ⟩ by (auto simp: h0 gauge_def γ'_def)
obtain D where countable D
  and D: ⋃D ⊆ cbox a b
  ∧K. K ∈ D ⇒ interior K ≠ {} ∧ (∃c d. K = cbox c d)
  and Dcovered: ⋀K. K ∈ D ⇒ ∃x. x ∈ cbox a b ∧ Ψ x μ ∧ x ∈ K ∧ K
⊆ γ' x
  and subUD: ?E ⊆ ⋃D
  by (rule covering_lemma [of ?E a b γ']) (simp_all add: Bex_def ⟨box a b
≠ {}⟩ ⟨gauge γ'⟩)
then have D ⊆ sets lebesgue
  by fastforce
show ∃T. {x. Ψ x μ} ∩ cbox a b ⊆ T ∧ T ∈ lmeasurable ∧ measure lebesgue
T ≤ e
proof (intro exI conjI)
show {x. Ψ x μ} ∩ cbox a b ⊆ ⋃D
  apply auto
  using subUD by auto
have mUE: measure lebesgue (⋃E) ≤ measure lebesgue (cbox a b)
  if E ⊆ D finite E for E
proof (rule measure_mono_fmeasurable)
show ⋃E ⊆ cbox a b
  using D(1) that(1) by blast
show ⋃E ∈ sets lebesgue
  by (metis D(2) fmeasurable.finite_Union fmeasurableD lmeasurable_cbox
subset_eq that)
qed auto
then show ⋃D ∈ lmeasurable

```

```

    by (metis  $\mathcal{D}(2)$  ‹countable  $\mathcal{D}$ › fmeasurable_Union_bound lmeasurable_cbox)
  then have leab:  $\text{measure lebesgue } (\bigcup \mathcal{D}) \leq \text{measure lebesgue } (\text{cbox } a \ b)$ 
  by (meson  $\mathcal{D}(1)$  fmeasurableD lmeasurable_cbox measure_mono_fmeasurable)
  obtain  $\mathcal{F}$  where  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$ 
    and  $\mathcal{F}$ :  $\text{measure lebesgue } (\bigcup \mathcal{D}) \leq 2 * \text{measure lebesgue } (\bigcup \mathcal{F})$ 
  proof (cases  $\text{measure lebesgue } (\bigcup \mathcal{D}) = 0$ )
    case True
      then show ?thesis
        by (force intro: that [where  $\mathcal{F} = \{\}$ ])
    next
      case False
        obtain  $\mathcal{F}$  where  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$ 
          and  $\mathcal{F}$ :  $\text{measure lebesgue } (\bigcup \mathcal{D}) / 2 < \text{measure lebesgue } (\bigcup \mathcal{F})$ 
          proof (rule measure_countable_Union_approachable [of  $\mathcal{D}$  measure lebesgue  $(\bigcup \mathcal{D}) / 2$  content (cbox a b)])
            show countable  $\mathcal{D}$ 
              by fact
            show  $0 < \text{measure lebesgue } (\bigcup \mathcal{D}) / 2$ 
              using False by (simp add: zero_less_measure_iff)
            show Dlm:  $D \in \text{lmeasurable}$  if  $D \in \mathcal{D}$  for  $D$ 
              using  $\mathcal{D}(2)$  that by blast
            show  $\text{measure lebesgue } (\bigcup \mathcal{F}) \leq \text{content } (\text{cbox } a \ b)$ 
              if  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$  for  $\mathcal{F}$ 
            proof -
              have  $\text{measure lebesgue } (\bigcup \mathcal{F}) \leq \text{measure lebesgue } (\bigcup \mathcal{D})$ 
              proof (rule measure_mono_fmeasurable)
                show  $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$ 
                  by (simp add: Sup_subset_mono ‹ $\mathcal{F} \subseteq \mathcal{D}$ ›)
                show  $\bigcup \mathcal{F} \in \text{sets lebesgue}$ 
                  by (meson Dlm fmeasurableD sets.finite_Union subset_eq that)
                show  $\bigcup \mathcal{D} \in \text{lmeasurable}$ 
                  by fact
              qed
            qed
            also have  $\dots \leq \text{measure lebesgue } (\text{cbox } a \ b)$ 
            proof (rule measure_mono_fmeasurable)
              show  $\bigcup \mathcal{D} \in \text{sets lebesgue}$ 
                by (simp add: ‹ $\bigcup \mathcal{D} \in \text{lmeasurable}$ › fmeasurableD)
            qed (auto simp: $\mathcal{D}(1)$ )
            finally show ?thesis
              by simp
          qed
        qed auto
      then show ?thesis
        using that by auto
    qed
  obtain tag where tag_in_E:  $\bigwedge D. D \in \mathcal{D} \implies \text{tag } D \in ?E$ 
    and tag_in_self:  $\bigwedge D. D \in \mathcal{D} \implies \text{tag } D \in D$ 
    and tag_sub:  $\bigwedge D. D \in \mathcal{D} \implies D \subseteq \gamma' (\text{tag } D)$ 

```

```

    using Dcovered by simp metis
  then have sub_ball_tag:  $\bigwedge D. D \in \mathcal{D} \implies D \subseteq \text{ball } (\text{tag } D) (\eta (\text{tag } D))$ 
    by (simp add:  $\gamma'$ _def)
  define  $\Phi$  where  $\Phi \equiv \lambda D. \text{BOX } (\eta(\text{tag } D)) (\text{tag } D)$ 
  define  $\Phi 2$  where  $\Phi 2 \equiv \lambda D. \text{BOX} 2 (\eta(\text{tag } D)) (\text{tag } D)$ 
  obtain  $\mathcal{C}$  where  $\mathcal{C} \subseteq \Phi 2 \text{ ' } \mathcal{F}$  pairwise disjnt  $\mathcal{C}$ 
    measure lebesgue  $(\bigcup \mathcal{C}) \geq \text{measure lebesgue } (\bigcup (\Phi 2 \text{ ' } \mathcal{F})) / 3 \wedge (\text{DIM}('a))$ 
  proof (rule Austin_Lemma)
    show finite  $(\Phi 2 \text{ ' } \mathcal{F})$ 
      using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
    have  $\exists k a b. \Phi 2 D = \text{cbox } a b \wedge (\forall i \in \text{Basis}. b \cdot i - a \cdot i = k)$  if  $D \in \mathcal{F}$ 
  for D
    apply (rule_tac  $x=2 * \eta(\text{tag } D)$  in exI)
    apply (rule_tac  $x=\text{tag } D - \eta(\text{tag } D) * R$  One in exI)
    apply (rule_tac  $x=\text{tag } D + \eta(\text{tag } D) * R$  One in exI)
    using that
    apply (auto simp:  $\Phi 2$ _def  $\text{BOX} 2$ _def algebra_simps)
    done
  then show  $\bigwedge D. D \in \Phi 2 \text{ ' } \mathcal{F} \implies \exists k a b. D = \text{cbox } a b \wedge (\forall i \in \text{Basis}. b$ 
 $\cdot i - a \cdot i = k)$ 
    by blast
  qed auto
  then obtain  $\mathcal{G}$  where  $\mathcal{G} \subseteq \mathcal{F}$  and disj: pairwise disjnt  $(\Phi 2 \text{ ' } \mathcal{G})$ 
    and measure lebesgue  $(\bigcup (\Phi 2 \text{ ' } \mathcal{G})) \geq \text{measure lebesgue } (\bigcup (\Phi 2 \text{ ' } \mathcal{F})) / 3$ 
 $\wedge (\text{DIM}('a))$ 
    unfolding  $\Phi 2$ _def subset_image_iff
    by (meson empty_subsetI equals0D pairwise_imageI)
  moreover
  have measure lebesgue  $(\bigcup (\Phi 2 \text{ ' } \mathcal{G})) * 3 \wedge \text{DIM}('a) \leq e/2$ 
  proof -
    have finite  $\mathcal{G}$ 
      using  $\langle \text{finite } \mathcal{F} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle$  infinite_super by blast
    have  $\text{BOX} 2$ _m:  $\bigwedge x. x \in \text{tag ' } \mathcal{G} \implies \text{BOX} 2 (\eta x) x \in \text{lmeasurable}$ 
      by (auto simp:  $\text{BOX} 2$ _def)
    have  $\text{BOX}$ _m:  $\bigwedge x. x \in \text{tag ' } \mathcal{G} \implies \text{BOX } (\eta x) x \in \text{lmeasurable}$ 
      by (auto simp:  $\text{BOX}$ _def)
    have  $\text{BOX}$ _sub:  $\text{BOX } (\eta x) x \subseteq \text{BOX} 2 (\eta x) x$  for x
      by (auto simp:  $\text{BOX}$ _def  $\text{BOX} 2$ _def subset_box algebra_simps)
    have DISJ2:  $\text{BOX} 2 (\eta (\text{tag } X)) (\text{tag } X) \cap \text{BOX} 2 (\eta (\text{tag } Y)) (\text{tag } Y)$ 
  = {}
    if  $X \in \mathcal{G} Y \in \mathcal{G} \text{tag } X \neq \text{tag } Y$  for X Y
  proof -
    obtain i where i:  $i \in \text{Basis tag } X \cdot i \neq \text{tag } Y \cdot i$ 
      using  $\langle \text{tag } X \neq \text{tag } Y \rangle$  by (auto simp: euclidean_eq_iff [of tag X])
    have XY:  $X \in \mathcal{D} Y \in \mathcal{D}$ 
      using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle$  that by auto
    then have  $0 \leq \eta (\text{tag } X) \ 0 \leq \eta (\text{tag } Y)$ 
      by (meson h0 le_cases not_le tag_in_E)+
    with XY i have  $\text{BOX} 2 (\eta (\text{tag } X)) (\text{tag } X) \neq \text{BOX} 2 (\eta (\text{tag } Y)) (\text{tag } Y)$ 

```



```

Y)
  unfolding eq_iff
  by (fastforce simp add: BOX2_def subset_box algebra_simps)
  then show ?thesis
    using disj that by (auto simp: pairwise_def disjnt_def  $\Phi$ 2_def)
  qed
  then have BOX2_disj: pairwise ( $\lambda x y.$  negligible (BOX2 ( $\eta$  x) x  $\cap$  BOX2
( $\eta$  y) y)) (tag 'G)
    by (simp add: pairwise_imageI)
  then have BOX_disj: pairwise ( $\lambda x y.$  negligible (BOX ( $\eta$  x) x  $\cap$  BOX
( $\eta$  y) y)) (tag 'G)
  proof (rule pairwise_mono)
    show negligible (BOX ( $\eta$  x) x  $\cap$  BOX ( $\eta$  y) y)
      if negligible (BOX2 ( $\eta$  x) x  $\cap$  BOX2 ( $\eta$  y) y) for x y
    by (metis (no_types, opaque_lifting) that Int_mono negligible_subset
BOX_sub)
  qed auto

  have eq:  $\bigwedge \text{box. } (\lambda D. \text{box } (\eta (\text{tag } D)) (\text{tag } D)) \text{' } \mathcal{G} = (\lambda t. \text{box } (\eta t) t) \text{' } \mathcal{G}$ 
  by (simp add: image_comp)
  have measure_lebesgue (BOX2 ( $\eta$  t) t) *  $3^{\wedge} \text{DIM('}a\text{'})$ 
    = measure_lebesgue (BOX ( $\eta$  t) t) *  $(2*3)^{\wedge} \text{DIM('}a\text{'})$ 
  if t  $\in$  tag 'G for t
  proof -
    have content (cbox (t -  $\eta$  t *R One) (t +  $\eta$  t *R One))
      = content (cbox t (t +  $\eta$  t *R One)) *  $2^{\wedge} \text{DIM('}a\text{'})$ 
    using that by (simp add: algebra_simps content_cbox_if_box_eq_empty)
    then show ?thesis
      by (simp add: BOX2_def BOX_def flip: power_mult_distrib)
  qed
  then have measure_lebesgue ( $\bigcup (\Phi$ 2 'G)) *  $3^{\wedge} \text{DIM('}a\text{'}) = \text{measure}$ 
lebesgue ( $\bigcup (\Phi$  'G)) *  $6^{\wedge} \text{DIM('}a\text{'})$ 
  unfolding  $\Phi$ _def  $\Phi$ 2_def eq
  by (simp add: measure_negligible_finite_Union_image
<finite G> BOX2_m BOX_m BOX2_disj BOX_disj sum_distrib_right
del: UN_simps)
  also have ...  $\leq e/2$ 
  proof -
    have  $\mu * \text{measure lebesgue } (\bigcup D \in \mathcal{G}. \Phi D) \leq \mu * (\sum D \in \Phi \text{' } \mathcal{G}. \text{measure}$ 
lebesgue D)
    using < $\mu > 0$ > <finite G> by (force simp: BOX_m  $\Phi$ _def fmeasurableD
intro: measure_Union_le)
    also have ... =  $(\sum D \in \Phi \text{' } \mathcal{G}. \text{measure lebesgue } D * \mu)$ 
      by (metis mult.commute sum_distrib_right)
    also have ...  $\leq (\sum (x, K) \in (\lambda D. (\text{tag } D, \Phi D)) \text{' } \mathcal{G}. \text{norm } (\text{content}$ 
K *R f x - integral K f))
  proof (rule sum_le_included; clarify?)
    fix D

```

```

assume D ∈ G
then have η (tag D) > 0
  using ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ h0 tag_in_E by auto
then have m_Φ: measure lebesgue (Φ D) > 0
  by (simp add: Φ_def BOX_def algebra_simps)
have μ ≤ norm (i (η(tag D)) (tag D) - f(tag D))
  using μ_le ⟨D ∈ G⟩ ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ tag_in_E by auto
also have ... = norm ((content (Φ D) *R f(tag D) - integral (Φ D)
f) /R measure lebesgue (Φ D))
  using m_Φ
  unfolding i_def Φ_def BOX_def
by (simp add: algebra_simps content_cbox_plus_norm_minus_commute)
  finally have measure lebesgue (Φ D) * μ ≤ norm (content (Φ D) *R
f(tag D) - integral (Φ D) f)
  using m_Φ by simp (simp add: field_simps)
then show ∃ y ∈ (λD. (tag D, Φ D)) 'G.
  snd y = Φ D ∧ measure lebesgue (Φ D) * μ ≤ (case y of (x, k)
⇒ norm (content k *R f x - integral k f))
  using ⟨D ∈ G⟩ by auto
qed (use ⟨finite G⟩ in auto)
also have ... < ?ee
proof (rule γ)
  show (λD. (tag D, Φ D)) 'G tagged_partial_division_of cbox (a -
One) (b + One)
  unfolding tagged_partial_division_of_def
proof (intro conjI allI impI ; clarify ?)
  show tag D ∈ Φ D
  if D ∈ G for D
  using that ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ h0 tag_in_E
  by (auto simp: Φ_def BOX_def mem_box algebra_simps
eucl_less_le_not_le_in_mono)
  show y ∈ cbox (a - One) (b + One) if D ∈ G y ∈ Φ D for D y
  using that BOX_cbox Φ_def ⟨F ⊆ D⟩ ⟨G ⊆ F⟩ tag_in_E by
blast

  show tag D = tag E ∧ Φ D = Φ E
  if D ∈ G E ∈ G and ne: interior (Φ D) ∩ interior (Φ E) ≠ {}
for D E
proof -
  have BOX2 (η (tag D)) (tag D) ∩ BOX2 (η (tag E)) (tag E) =
{} ∨ tag E = tag D
  using DISJ2 ⟨D ∈ G⟩ ⟨E ∈ G⟩ by force
  then have BOX (η (tag D)) (tag D) ∩ BOX (η (tag E)) (tag E)
= {} ∨ tag E = tag D
  using BOX_sub by blast
  then show tag D = tag E ∧ Φ D = Φ E
  by (metis Φ_def interior_Int interior_empty ne)
qed
qed (use ⟨finite G⟩ Φ_def BOX_def in auto)
show γ fine (λD. (tag D, Φ D)) 'G

```

```

      unfolding fine_def  $\Phi$ _def using BOX_ $\gamma$   $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$   $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$ 
tag_in_E by blast
      qed
      finally show ?thesis
        using  $\langle \mu > 0 \rangle$  by (auto simp: field_split_simps)
    qed
    finally show ?thesis .
  qed
  moreover
  have measure_lebesgue  $(\bigcup \mathcal{F}) \leq$  measure_lebesgue  $(\bigcup (\Phi 2' \mathcal{F}))$ 
  proof (rule measure_mono_fmeasurable)
    have  $D \subseteq$  ball (tag D)  $(\eta(\text{tag } D))$  if  $D \in \mathcal{F}$  for D
      using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  sub_ball_tag that by blast
    moreover have ball (tag D)  $(\eta(\text{tag } D)) \subseteq$  BOX2  $(\eta(\text{tag } D))$  (tag D) if
  D  $\in \mathcal{F}$  for D
    proof (clarsimp simp:  $\Phi 2$ _def BOX2_def mem_box algebra_simps
  dist_norm)
      fix x and i::'a
      assume norm (tag D - x)  $< \eta(\text{tag } D)$  and  $i \in$  Basis
      then have |tag D  $\cdot i - x \cdot i| \leq \eta(\text{tag } D)$ 
        by (metis eucl_less_le_not_le inner_commute inner_diff_right
  norm_bound_Basis_le)
      then show tag D  $\cdot i \leq x \cdot i + \eta(\text{tag } D) \wedge x \cdot i \leq \eta(\text{tag } D) +$  tag D
         $\cdot i$ 
        by (simp add: abs_diff_le_iff)
    qed
    ultimately show  $\bigcup \mathcal{F} \subseteq \bigcup (\Phi 2' \mathcal{F})$ 
      by (force simp:  $\Phi 2$ _def)
    show  $\bigcup \mathcal{F} \in$  sets_lebesgue
      using  $\langle \text{finite } \mathcal{F} \rangle$   $\langle \mathcal{D} \subseteq$  sets_lebesgue  $\rangle$   $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  by blast
    show  $\bigcup (\Phi 2' \mathcal{F}) \in$  lmeasurable
      unfolding  $\Phi 2$ _def BOX2_def using  $\langle \text{finite } \mathcal{F} \rangle$  by blast
  qed
  ultimately
  have measure_lebesgue  $(\bigcup \mathcal{F}) \leq e/2$ 
    by (auto simp: field_split_simps)
  then show measure_lebesgue  $(\bigcup \mathcal{D}) \leq e$ 
    using  $\mathcal{F}$  by linarith
  qed
  qed
  then have  $\bigwedge j.$  negligible  $\{x. \Psi x (\text{inverse}(\text{real } j + 1))\}$ 
    using negligible_on_intervals
    by (metis (full_types) inverse_positive_iff_positive le_add_same_cancel1
  linorder_not_le nat_le_real_less not_add_less1 of_nat_0)
  then have negligible ?M
    by auto
  moreover have ?N  $\subseteq$  ?M
  proof (clarsimp simp: dist_norm)

```

```

fix y e
assume 0 < e
  and ye [rule_format]:  $\Psi$  y e
  then obtain k where k: 0 < k inverse (real k + 1) < e
  by (metis One_nat_def add.commute less_add_same_cancel2 less_imp_inverse_less
less_trans neq0_conv of_nat_1 of_nat_Suc reals_Archimedean zero_less_one)
  with ye show  $\exists n. \Psi$  y (inverse (real n + 1))
    apply (rule_tac x=k in exI)
    unfolding  $\Psi$ _def
    by (force intro: less_le_trans)
qed
ultimately show negligible ?N
  by (blast intro: negligible_subset)
show  $\neg \Psi$  x e if x  $\notin$  ?N  $\wedge$  0 < e for x e
  using that by blast
qed
with that show ?thesis
  unfolding i_def BOX_def  $\Psi$ _def by (fastforce simp add: not_le)
qed

```

9.16.3 HOL Light measurability

definition measurable_on :: ('a::euclidean_space \Rightarrow 'b::real_normed_vector) \Rightarrow 'a set \Rightarrow bool

(infixr measurable'_on 46)

where f measurable_on S \equiv

$$\exists N g. \text{negligible } N \wedge$$

$$(\forall n. \text{continuous_on } UNIV (g\ n)) \wedge$$

$$(\forall x. x \notin N \longrightarrow (\lambda n. g\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } f\ x \text{ else } 0))$$

lemma measurable_on_UNIV:

($\lambda x. \text{if } x \in S \text{ then } f\ x \text{ else } 0$) measurable_on UNIV \longleftrightarrow f measurable_on S
by (auto simp: measurable_on_def)

lemma measurable_on_spike_set:

assumes f: f measurable_on S **and** neg: negligible ((S - T) \cup (T - S))
shows f measurable_on T

proof -

obtain N **and** F

where N: negligible N

and conF: $\bigwedge n. \text{continuous_on } UNIV (F\ n)$

and tendsF: $\bigwedge x. x \notin N \implies (\lambda n. F\ n\ x) \longrightarrow (\text{if } x \in S \text{ then } f\ x \text{ else } 0)$

using f **by** (auto simp: measurable_on_def)

show ?thesis

unfolding measurable_on_def

proof (intro exI conjI allI impI)

show continuous_on UNIV ($\lambda x. F\ n\ x$) **for** n

by (intro conF continuous_intros)

show negligible (N \cup (S - T) \cup (T - S))

```

    by (metis (full_types) N neg negligible_Un_eq)
  show  $(\lambda n. F n x) \longrightarrow (if\ x \in T\ then\ f\ x\ else\ 0)$ 
    if  $x \notin (N \cup (S - T) \cup (T - S))$  for  $x$ 
    using that tendsF [of  $x$ ] by auto
qed
qed

```

Various common equivalent forms of function measurability.

```

lemma measurable_on_0 [simp]:  $(\lambda x. 0)$  measurable_on  $S$ 
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show  $(\lambda n. 0) \longrightarrow (if\ x \in S\ then\ 0::'b\ else\ 0)$  for  $x$ 
    by force
qed auto

```

```

lemma measurable_on_scaleR_const:
  assumes  $f: f$  measurable_on  $S$ 
  shows  $(\lambda x. c *_{\mathbb{R}} f x)$  measurable_on  $S$ 
proof -
  obtain  $NF$  and  $F$ 
  where  $NF: negligible\ NF$ 
    and  $conF: \bigwedge n. continuous\_on\ UNIV\ (F\ n)$ 
    and  $tendsF: \bigwedge x. x \notin NF \implies (\lambda n. F\ n\ x) \longrightarrow (if\ x \in S\ then\ f\ x\ else\ 0)$ 
  using  $f$  by (auto simp: measurable_on_def)
  show ?thesis
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show  $continuous\_on\ UNIV\ (\lambda x. c *_{\mathbb{R}} F\ n\ x)$  for  $n$ 
    by (intro conF continuous_intros)
  show  $(\lambda n. c *_{\mathbb{R}} F\ n\ x) \longrightarrow (if\ x \in S\ then\ c *_{\mathbb{R}} f\ x\ else\ 0)$ 
    if  $x \notin NF$  for  $x$ 
    using tendssto_scaleR [OF tendssto_const tendsF, of  $x$ ] that by auto
  qed (auto simp: NF)
qed

```

```

lemma measurable_on_cmul:
  fixes  $c :: real$ 
  assumes  $f$  measurable_on  $S$ 
  shows  $(\lambda x. c * f x)$  measurable_on  $S$ 
  using measurable_on_scaleR_const [OF assms] by simp

```

```

lemma measurable_on_cdivide:
  fixes  $c :: real$ 
  assumes  $f$  measurable_on  $S$ 
  shows  $(\lambda x. f x / c)$  measurable_on  $S$ 
proof (cases  $c=0$ )
  case False
  then show ?thesis

```

3234

```

    using measurable_on_cmul [of f S 1/c]
    by (simp add: assms)
qed auto

```

```

lemma measurable_on_minus:
  f measurable_on S  $\implies$  ( $\lambda x. -(f x)$ ) measurable_on S
  using measurable_on_scaleR_const [of f S -1] by auto

```

```

lemma continuous_imp_measurable_on:
  continuous_on UNIV f  $\implies$  f measurable_on UNIV
  unfolding measurable_on_def
  apply (rule_tac x={ } in exI)
  apply (rule_tac x= $\lambda n. f$  in exI, auto)
  done

```

```

proposition integrable_subintervals_imp_measurable:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes  $\bigwedge a b. f$  integrable_on cbox a b
  shows f measurable_on UNIV

```

```

proof -
  define BOX where BOX  $\equiv$   $\lambda h. \lambda x::'a. cbox x (x + h *R One)$ 
  define i where i  $\equiv$   $\lambda h x. integral (BOX h x) f /R h ^ DIM('a)$ 
  obtain N where negligible N
  and k:  $\bigwedge x e. \llbracket x \notin N; 0 < e \rrbracket$ 
            $\implies \exists d > 0. \forall h. 0 < h \wedge h < d \implies$ 
             norm (integral (cbox x (x + h *R One)) f /R h ^ DIM('a) - f x)
< e

```

```

  using integrable_ccontinuous_explicit assms by blast
  show ?thesis
  unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show continuous_on UNIV (( $\lambda n x. i$  (inverse (Suc n)) x) n) for n
    proof (clarsimp simp: continuous_on_iff)
      show  $\exists d > 0. \forall x'. dist x' x < d \implies dist (i (inverse (1 + real n)) x') (i$ 
(inverse (1 + real n)) x) < e
      if 0 < e
      for x e
    proof -
      let ?e = e / (1 + real n) ^ DIM('a)
      have ?e > 0
      using <e > 0 by auto
      moreover have x  $\in$  cbox (x - 2 *R One) (x + 2 *R One)
      by (simp add: mem_box inner_diff_left inner_left_distrib)
      moreover have x + One /R real (Suc n)  $\in$  cbox (x - 2 *R One) (x + 2
*R One)
      by (auto simp: mem_box inner_diff_left inner_left_distrib field_simps)
      ultimately obtain  $\delta > 0$ 

```

```

and  $\delta$ :  $\bigwedge c' d'. \llbracket c' \in \text{cbox } (x - 2 *_{\mathbb{R}} \text{One}) (x + 2 *_{\mathbb{R}} \text{One});$ 
       $d' \in \text{cbox } (x - 2 *_{\mathbb{R}} \text{One}) (x + 2 *_{\mathbb{R}} \text{One});$ 
       $\text{norm}(c' - x) \leq \delta; \text{norm}(d' - (x + \text{One} /_{\mathbb{R}} \text{Suc } n)) \leq \delta \rrbracket$ 
       $\implies \text{norm}(\text{integral}(\text{cbox } c' d') f - \text{integral}(\text{cbox } x (x + \text{One}$ 
/_{\mathbb{R}} \text{Suc } n)) f) < ?e
  by (blast intro: indefinite_integral_continuous [of f _ _ x] assms)
show ?thesis
proof (intro exI impI conjI allI)
  show  $\min \delta 1 > 0$ 
    using  $\langle \delta > 0 \rangle$  by auto
  show  $\text{dist } (i (\text{inverse } (1 + \text{real } n)) y) (i (\text{inverse } (1 + \text{real } n)) x) < e$ 
    if  $\text{dist } y x < \min \delta 1$  for  $y$ 
  proof -
    have  $\text{no}: \text{norm } (y - x) < 1$ 
      using that by (auto simp: dist_norm)
    have  $\text{le1}: \text{inverse } (1 + \text{real } n) \leq 1$ 
      by (auto simp: field_split_simps)
    have  $\text{norm } (\text{integral } (\text{cbox } y (y + \text{One} /_{\mathbb{R}} \text{real } (\text{Suc } n))) f$ 
       $- \text{integral } (\text{cbox } x (x + \text{One} /_{\mathbb{R}} \text{real } (\text{Suc } n))) f)$ 
       $< e / (1 + \text{real } n) ^ \text{DIM}(a)$ 
  proof (rule  $\delta$ )
    show  $y \in \text{cbox } (x - 2 *_{\mathbb{R}} \text{One}) (x + 2 *_{\mathbb{R}} \text{One})$ 
    using no by (auto simp: mem_box algebra_simps dest: Basis_le_norm
[of _  $y-x$ ])
    show  $y + \text{One} /_{\mathbb{R}} \text{real } (\text{Suc } n) \in \text{cbox } (x - 2 *_{\mathbb{R}} \text{One}) (x + 2 *_{\mathbb{R}} \text{One})$ 
  proof (simp add: dist_norm mem_box algebra_simps, intro ballI conjI)
    fix  $i::'a$ 
    assume  $i \in \text{Basis}$ 
    then have  $1: |y \cdot i - x \cdot i| < 1$ 
    by (metis inner_commute inner_diff_right no_norm_bound_Basis_lt)
    moreover have  $\dots < (2 + \text{inverse } (1 + \text{real } n)) 1 \leq 2 - \text{inverse}$ 
 $(1 + \text{real } n)$ 
      by (auto simp: field_simps)
    ultimately show  $x \cdot i \leq y \cdot i + (2 + \text{inverse } (1 + \text{real } n))$ 
       $y \cdot i + \text{inverse } (1 + \text{real } n) \leq x \cdot i + 2$ 
      by linarith+
    qed
    show  $\text{norm } (y - x) \leq \delta \text{norm } (y + \text{One} /_{\mathbb{R}} \text{real } (\text{Suc } n) - (x + \text{One}$ 
/_{\mathbb{R}} \text{real } (\text{Suc } n))) \leq \delta
      using that by (auto simp: dist_norm)
    qed
  then show ?thesis
  using that by (simp add: dist_norm i_def BOX_def flip: scaleR_diff_right)
(simp add: field_simps)
  qed
  qed
  qed
  show negligible  $N$ 

```

```

    by (simp add: ‹negligible N›)
  show (λn. i (inverse (Suc n)) x) ⟶ (if x ∈ UNIV then f x else 0)
  if x ∉ N for x
  unfolding lim_sequentially
proof clarsimp
  show ∃ no. ∀ n ≥ no. dist (i (inverse (1 + real n)) x) (f x) < e
  if 0 < e for e
  proof -
    obtain d where d > 0
    and d: ∧h. [0 < h; h < d] ⟹
      norm (integral (cbox x (x + h *R One)) f /R h ^ DIM('a) - f x) < e
    using k [of x e] ‹x ∉ N› ‹0 < e› by blast
    then obtain M where M: M ≠ 0 0 < inverse (real M) inverse (real M)
  < d
    using real_arch_invD by auto
  show ?thesis
  proof (intro exI allI impI)
    show dist (i (inverse (1 + real n)) x) (f x) < e
    if M ≤ n for n
    proof -
      have *: 0 < inverse (1 + real n) inverse (1 + real n) ≤ inverse M
      using that ‹M ≠ 0› by auto
      show ?thesis
      using that M
      apply (simp add: i_def BOX_def dist_norm)
      apply (blast intro: le_less_trans * d)
      done
    qed
  qed
  qed
  qed
  qed
  qed
  qed

```

9.16.4 Composing continuous and measurable functions; a few variants

```

lemma measurable_on_compose_continuous:
  assumes f: f measurable_on UNIV and g: continuous_on UNIV g
  shows (g ∘ f) measurable_on UNIV
proof -
  obtain N and F
  where negligible N
  and conF: ∧n. continuous_on UNIV (F n)
  and tendsF: ∧x. x ∉ N ⟹ (λn. F n x) ⟶ f x
  using f by (auto simp: measurable_on_def)
  show ?thesis
  unfolding measurable_on_def
  proof (intro exI conjI allI impI)

```



```

show negligible N
  by fact
show continuous_on UNIV (g ∘ (F n)) for n
  using conF continuous_on_compose continuous_on_subset g by blast
show (λn. (g ∘ F n) x) ⟶ (if x ∈ UNIV then (g ∘ f) x else 0)
  if x ∉ N for x :: 'a
  using that g tendsF by (auto simp: continuous_on_def intro: tendsto_compose)
qed
qed

```

```

lemma measurable_on_compose_continuous_0:
  assumes f: f measurable_on S and g: continuous_on UNIV g and g 0 = 0
  shows (g ∘ f) measurable_on S
proof -
  have f': (λx. if x ∈ S then f x else 0) measurable_on UNIV
  using f measurable_on_UNIV by blast
  show ?thesis
  using measurable_on_compose_continuous [OF f' g]
  by (simp add: measurable_on_UNIV o_def if_distrib ⟨g 0 = 0⟩ cong: if_cong)
qed

```

```

lemma measurable_on_compose_continuous_box:
  assumes fm: f measurable_on UNIV and fab: ∧x. f x ∈ box a b
  and contg: continuous_on (box a b) g
  shows (g ∘ f) measurable_on UNIV
proof -
  have ∃γ. (∀n. continuous_on UNIV (γ n)) ∧ (∀x. x ∉ N ⟶ (λn. γ n x)
  ⟶ g (f x))
  if negligible N
  and conth [rule_format]: ∀n. continuous_on UNIV (λx. h n x)
  and tends [rule_format]: ∀x. x ∉ N ⟶ (λn. h n x) ⟶ f x
  for N and h :: nat ⇒ 'a ⇒ 'b
proof -
  define ϑ where ϑ ≡ λn x. (∑ i∈Basis. (max (a·i + (b·i - a·i) / real (n+2))
  (min ((h n x)·i
  (b·i - (b·i - a·i) / real (n+2)))))) *R i
  have aibi: ∧i. i ∈ Basis ⟹ a · i < b · i
  using box_ne_empty(2) fab by auto
  then have *: ∧i n. i ∈ Basis ⟹ a · i + real n * (a · i) < b · i + real n * (b
  · i)
  by (meson add_mono_thms_linordered_field(3) less_eq_real_def mult_left_mono
  of_nat_0_le_iff)
  show ?thesis
proof (intro exI conjI allI impI)
  show continuous_on UNIV (g ∘ (ϑ n)) for n :: nat
  unfolding ϑ_def
  apply (intro continuous_on_compose2 [OF contg] continuous_intros conth)
  apply (auto simp: aibi * mem_box less_max_iff_disj min_less_iff_disj)

```

```

field_split_simps)
  done
  show (λn. (g ∘ ϑ n) x) ⟶ g (f x)
  if x ∉ N for x
  unfolding o_def
  proof (rule isCont_tendsto_compose [where g=g])
    show isCont g (f x)
      using contg fab continuous_on_eq_continuous_at by blast
    have (λn. ϑ n x) ⟶ (∑ i∈Basis. max (a · i) (min (f x · i) (b · i)) *R
i)

      unfolding ϑ_def
    proof (intro tendsto_intros ⟨x ∉ N⟩ tends)
      fix i::'b
      assume i ∈ Basis
      have a: (λn. a · i + (b · i - a · i) / real n) ⟶ a · i + 0
        by (intro tendsto_add lim_const_over_n tendsto_const)
      show (λn. a · i + (b · i - a · i) / real (n + 2)) ⟶ a · i
        using LIMSEQ_ignore_initial_segment [where k=2, OF a] by simp
      have b: (λn. b · i - (b · i - a · i) / (real n)) ⟶ b · i - 0
        by (intro tendsto_diff lim_const_over_n tendsto_const)
      show (λn. b · i - (b · i - a · i) / real (n + 2)) ⟶ b · i
        using LIMSEQ_ignore_initial_segment [where k=2, OF b] by simp
    qed
  also have (∑ i∈Basis. max (a · i) (min (f x · i) (b · i)) *R i) = (∑ i∈Basis.
(f x · i) *R i)
    using fab by (auto simp add: mem_box intro: sum.cong)
  also have ... = f x
    using euclidean_representation by blast
  finally show (λn. ϑ n x) ⟶ f x .
  qed
  qed
  qed
  then show ?thesis
    using fm by (auto simp: measurable_on_def)
  qed

```

lemma measurable_on_Pair:

assumes f: f measurable_on S and g: g measurable_on S

shows (λx. (f x, g x)) measurable_on S

proof -

obtain NF and F

where NF: negligible NF

and conF: $\bigwedge n. \text{continuous_on UNIV } (F n)$

and tendsF: $\bigwedge x. x \notin NF \implies (\lambda n. F n x) \longrightarrow (\text{if } x \in S \text{ then } f x \text{ else } 0)$

using f by (auto simp: measurable_on_def)

obtain NG and G

where NG: negligible NG

and conG: $\bigwedge n. \text{continuous_on UNIV } (G n)$

and tendsG: $\bigwedge x. x \notin NG \implies (\lambda n. G n x) \longrightarrow (\text{if } x \in S \text{ then } g x \text{ else } 0)$

```

  using g by (auto simp: measurable_on_def)
show ?thesis
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show negligible (NF  $\cup$  NG)
  by (simp add: NF NG)
  show continuous_on UNIV ( $\lambda x. (F n x, G n x)$ ) for n
  using conF conG continuous_on_Pair by blast
  show ( $\lambda n. (F n x, G n x)$ )  $\longrightarrow$  (if  $x \in S$  then (f x, g x) else 0)
  if  $x \notin NF \cup NG$  for x
  using tendsto_Pair [OF tendsF tendsG, of x x] that unfolding zero_prod_def
  by (simp add: split: if_split_asm)
qed
qed

```

lemma *measurable_on_combine*:

```

  assumes f: f measurable_on S and g: g measurable_on S
  and h: continuous_on UNIV ( $\lambda x. h (fst x) (snd x)$ ) and h 0 0 = 0
  shows ( $\lambda x. h (f x) (g x)$ ) measurable_on S
proof -
  have *: ( $\lambda x. h (f x) (g x)$ ) = ( $\lambda x. h (fst x) (snd x)$ )  $\circ$  ( $\lambda x. (f x, g x)$ )
  by auto
  show ?thesis
  unfolding * by (auto simp: measurable_on_compose_continuous_0 measurable_on_Pair assms)
qed

```

lemma *measurable_on_add*:

```

  assumes f: f measurable_on S and g: g measurable_on S
  shows ( $\lambda x. f x + g x$ ) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

```

lemma *measurable_on_diff*:

```

  assumes f: f measurable_on S and g: g measurable_on S
  shows ( $\lambda x. f x - g x$ ) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

```

lemma *measurable_on_scaleR*:

```

  assumes f: f measurable_on S and g: g measurable_on S
  shows ( $\lambda x. f x *_{\mathbb{R}} g x$ ) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

```

lemma *measurable_on_sum*:

```

  assumes finite I  $\wedge$   $i. i \in I \implies f i$  measurable_on S
  shows ( $\lambda x. \text{sum } (\lambda i. f i x) I$ ) measurable_on S
  using assms by (induction I) (auto simp: measurable_on_add)

```

lemma *measurable_on_spike*:

```

  assumes f: f measurable_on T and negligible S and gf:  $\bigwedge x. x \in T - S \implies g$ 

```

3240

```

x = f x
shows g measurable_on T
proof -
  obtain NF and F
  where NF: negligible NF
    and conF:  $\bigwedge n. \text{continuous\_on UNIV } (F n)$ 
    and tendsF:  $\bigwedge x. x \notin NF \implies (\lambda n. F n x) \longrightarrow (\text{if } x \in T \text{ then } f x \text{ else } 0)$ 
  using f by (auto simp: measurable_on_def)
  show ?thesis
  unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (NF  $\cup$  S)
    by (simp add: NF <negligible S>)
    show  $\bigwedge x. x \notin NF \cup S \implies (\lambda n. F n x) \longrightarrow (\text{if } x \in T \text{ then } g x \text{ else } 0)$ 
    by (metis (full_types) Diff_iff Un_iff gf_tendsF)
  qed (auto simp: conF)
qed

```

```

proposition indicator_measurable_on:
  assumes S  $\in$  sets lebesgue
  shows indicat_real S measurable_on UNIV
proof -
  { fix n::nat
    let ? $\varepsilon$  = (1::real) / (2 * 2 $^n$ )
    have  $\varepsilon$ : ? $\varepsilon$  > 0
    by auto
    obtain T where closed T T  $\subseteq$  S S-T  $\in$  lmeasurable and ST: emeasure
    lebesgue (S - T) < ? $\varepsilon$ 
    by (meson  $\varepsilon$  assms sets_lebesgue_inner_closed)
    obtain U where open U S  $\subseteq$  U (U - S)  $\in$  lmeasurable and US: emeasure
    lebesgue (U - S) < ? $\varepsilon$ 
    by (meson  $\varepsilon$  assms sets_lebesgue_outer_open)
    have eq:  $-T \cap U = (S-T) \cup (U - S)$ 
    using <T  $\subseteq$  S> <S  $\subseteq$  U> by auto
    have emeasure lebesgue ((S-T)  $\cup$  (U - S))  $\leq$  emeasure lebesgue (S - T) +
    emeasure lebesgue (U - S)
    using <S - T  $\in$  lmeasurable> <U - S  $\in$  lmeasurable> emeasure_subadditive
  by blast
  also have ... < ? $\varepsilon$  + ? $\varepsilon$ 
  using ST US add_mono_enreal by metis
  finally have le: emeasure lebesgue ( $-T \cap U$ ) < enreal (1 / 2 $^n$ )
  by (simp add: eq)
  have 1: continuous_on (T  $\cup$  -U) (indicat_real S)
  unfolding indicator_def of_bool_def
  proof (rule continuous_on_cases [OF <closed T>])
  show closed (- U)
  using <open U> by blast
  show continuous_on T ( $\lambda x. 1::real$ ) continuous_on (- U) ( $\lambda x. 0::real$ )
  by (auto simp: continuous_on)

```

```

    show  $\forall x. x \in T \wedge x \notin S \vee x \in -U \wedge x \in S \longrightarrow (1::real) = 0$ 
      using  $\langle T \subseteq S \rangle \langle S \subseteq U \rangle$  by auto
  qed
  have 2: closedin (top_of_set UNIV) (T  $\cup$  -U)
    using  $\langle closed\ T \rangle \langle open\ U \rangle$  by auto
  obtain g where continuous_on UNIV g  $\wedge x. x \in T \cup -U \implies g\ x = indicat\_real\ S\ x$ 
     $\wedge x. norm(g\ x) \leq 1$ 
    by (rule Tietze [OF 1 2, of 1]) auto
  with le have  $\exists g\ E. continuous\_on\ UNIV\ g \wedge (\forall x \in -E. g\ x = indicat\_real\ S\ x) \wedge$ 
     $(\forall x. norm(g\ x) \leq 1) \wedge E \in sets\ lebesgue \wedge emeasure\ lebesgue$ 
     $E < ennreal\ (1 / 2^{\wedge}n)$ 
    apply (rule_tac  $x=g$  in exI)
    apply (rule_tac  $x=-T \cap U$  in exI)
    using  $\langle S - T \in lmeasurable \rangle \langle U - S \in lmeasurable \rangle$  eq by auto
  }
  then obtain g E where cont:  $\bigwedge n. continuous\_on\ UNIV\ (g\ n)$ 
    and geq:  $\bigwedge n\ x. x \in -E\ n \implies g\ n\ x = indicat\_real\ S\ x$ 
    and ng1:  $\bigwedge n\ x. norm(g\ n\ x) \leq 1$ 
    and Eset:  $\bigwedge n. E\ n \in sets\ lebesgue$ 
    and Em:  $\bigwedge n. emeasure\ lebesgue\ (E\ n) < ennreal\ (1 / 2^{\wedge}n)$ 
    by metis
  have null: limsup E  $\in null\_sets\ lebesgue$ 
  proof (rule borel_cantelli_limsup1 [OF Eset])
    show emeasure lebesgue (E n)  $< \infty$  for n
      by (metis Em infinity_ennreal_def order.asym top.not_eq_extremum)
    show summable  $(\lambda n. measure\ lebesgue\ (E\ n))$ 
  proof (rule summable_comparison_test' [OF summable_geometric, of 1/2 0])
    show norm (measure lebesgue (E n))  $\leq (1/2)^{\wedge}n$  for n
      using Em [of n] by (simp add: measure_def enn2real_leI power_one_over)
  qed auto
  qed
  have tends:  $(\lambda n. g\ n\ x) \longrightarrow indicat\_real\ S\ x$  if  $x \notin limsup\ E$  for x
  proof -
    have  $\forall_F\ n\ in\ sequentially. x \in -E\ n$ 
      using that by (simp add: mem_limsup_iff not_frequently)
    then show ?thesis
      unfolding tendsto_iff_dist_real_def
      by (simp add: eventually_mono geq)
  qed
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (limsup E)
      using negligible_iff_null_sets null by blast
    show continuous_on UNIV (g n) for n
      using cont by blast
  qed (use tends in auto)
  qed

```

lemma *measurable_on_restrict*:

assumes f : f measurable_on UNIV **and** S : $S \in \text{sets lebesgue}$

shows $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0)$ measurable_on UNIV

proof –

have *indicat_real S measurable_on UNIV*

by (*simp add: S indicator_measurable_on*)

then show *?thesis*

using *measurable_on_scaleR [OF _ f, of indicat_real S]*

by (*simp add: indicator_scaleR_eq_if*)

qed

lemma *measurable_on_const_UNIV*: $(\lambda x. k)$ measurable_on UNIV

by (*simp add: continuous_imp_measurable_on*)

lemma *measurable_on_const [simp]*: $S \in \text{sets lebesgue} \implies (\lambda x. k)$ measurable_on S

using *measurable_on_UNIV measurable_on_const_UNIV measurable_on_restrict*
by *blast*

lemma *simple_function_indicator_representation_real*:

fixes $f :: 'a \Rightarrow \text{real}$

assumes f : *simple_function M f* **and** x : $x \in \text{space } M$ **and** nn : $\bigwedge x. f x \geq 0$

shows $f x = (\sum y \in f \text{ ' space } M. y * \text{indicator } (f \text{ - ' } \{y\} \cap \text{space } M) x)$

proof –

have f' : *simple_function M (ennreal o f)*

by (*simp add: f*)

have $*$: $f x =$

enn2real

$(\sum y \in \text{ennreal ' } f \text{ ' space } M.$

$y * \text{indicator } ((\text{ennreal o } f) \text{ - ' } \{y\} \cap \text{space } M) x)$

using *arg_cong [OF simple_function_indicator_representation [OF f' x], of enn2real, simplified nn o_def] nn*

unfolding *o_def image_comp*

by (*metis enn2real_ennreal*)

have *enn2real* $(\sum y \in \text{ennreal ' } f \text{ ' space } M. \text{if } \text{ennreal } (f x) = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0)$

$= \text{sum } (\text{enn2real o } (\lambda y. \text{if } \text{ennreal } (f x) = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0))$

$(\text{ennreal ' } f \text{ ' space } M)$

by (*rule enn2real_sum*) *auto*

also have $\dots = \text{sum } (\text{enn2real o } (\lambda y. \text{if } \text{ennreal } (f x) = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0) \text{ o } \text{ennreal})$

$(f \text{ ' space } M)$

by (*rule sum_reindex*) (*use nn in <auto simp: inj_on_def intro: sum.cong>*)

also have $\dots = (\sum y \in f \text{ ' space } M. \text{if } f x = y \wedge x \in \text{space } M \text{ then } y \text{ else } 0)$

using *nn*

by (*auto simp: inj_on_def intro: sum.cong*)

finally show *?thesis*

by (*subst **) (*simp add: enn2real_sum indicator_def of_bool_def if_distrib*)

cong: if_cong)
qed

lemma simple_function_induct_real

[consumes 1, case_names cong set mult add, induct set: simple_function]:
fixes $u :: 'a \Rightarrow \text{real}$
assumes u : simple_function M u
assumes cong: $\bigwedge f g. \text{simple_function } M f \Longrightarrow \text{simple_function } M g \Longrightarrow (AE\ x$
in $M. f\ x = g\ x) \Longrightarrow P\ f \Longrightarrow P\ g$
assumes set: $\bigwedge A. A \in \text{sets } M \Longrightarrow P(\text{indicator } A)$
assumes mult: $\bigwedge u\ c. P\ u \Longrightarrow P(\lambda x. c * u\ x)$
assumes add: $\bigwedge u\ v. P\ u \Longrightarrow P\ v \Longrightarrow P(\lambda x. u\ x + v\ x)$
and nn: $\bigwedge x. u\ x \geq 0$
shows $P\ u$
proof (rule cong)
from AE_space show $AE\ x$ in $M. (\sum_{y \in u\ ' \text{space } M. y * \text{indicator } (u\ -\ ' \{y\} \cap \text{space } M)\ x) = u\ x$
proof eventually_elim
fix x assume $x: x \in \text{space } M$
from simple_function_indicator_representation_real[OF $u\ x$] nn
show $(\sum_{y \in u\ ' \text{space } M. y * \text{indicator } (u\ -\ ' \{y\} \cap \text{space } M)\ x) = u\ x$
by metis
qed
next
from u have finite $(u\ ' \text{space } M)$
unfolding simple_function_def by auto
then show $P(\lambda x. \sum_{y \in u\ ' \text{space } M. y * \text{indicator } (u\ -\ ' \{y\} \cap \text{space } M)\ x)$
proof induct
case empty
then show ?case
using set[of {}] by (simp add: indicator_def[abs_def])
next
case (insert $a\ F$)
have eq: $\sum \{y. u\ x = y \wedge (y = a \vee y \in F) \wedge x \in \text{space } M\}$
= $(\text{if } u\ x = a \wedge x \in \text{space } M \text{ then } a \text{ else } 0) + \sum \{y. u\ x = y \wedge y \in F$
 $\wedge x \in \text{space } M\}$ for x
proof (cases $x \in \text{space } M$)
case True
have *: $\{y. u\ x = y \wedge (y = a \vee y \in F)\} = \{y. u\ x = a \wedge y = a\} \cup \{y. u\ x$
 $= y \wedge y \in F\}$
by auto
show ?thesis
using insert by (simp add: * True)
qed auto
have $a: P(\lambda x. a * \text{indicator } (u\ -\ ' \{a\} \cap \text{space } M)\ x)$
proof (intro mult set)
show $u\ -\ ' \{a\} \cap \text{space } M \in \text{sets } M$
using u by auto
qed

```

    show ?case
      using nn insert a
      by (simp add: eq indicator_times_eq_if [where f =  $\lambda x. a$ ] add)
    qed
  next
    show simple_function M ( $\lambda x. (\sum_{y \in u \text{ ' space } M. y * \text{indicator } (u - \{y\} \cap \text{space } M) x)$ )
      apply (subst simple_function_cong)
      apply (rule simple_function_indicator_representation_real[symmetric])
      apply (auto intro: u nn)
      done
    qed fact

```

```

proposition simple_function_measurable_on_UNIV:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes f: simple_function_lebesgue f and nn:  $\bigwedge x. f x \geq 0$ 
  shows f measurable_on UNIV
  using f
proof (induction f)
  case (cong f g)
  then obtain N where negligible N  $\{x. g x \neq f x\} \subseteq N$ 
    by (auto simp: eventually_ae_filter_negligible eq_commute)
  then show ?case
    by (blast intro: measurable_on_spike cong)
  next
  case (set S)
  then show ?case
    by (simp add: indicator_measurable_on)
  next
  case (mult u c)
  then show ?case
    by (simp add: measurable_on_cmul)
  case (add u v)
  then show ?case
    by (simp add: measurable_on_add)
  qed (auto simp: nn)

```

```

lemma simple_function_lebesgue_if:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  real
  assumes f: simple_function_lebesgue f and S:  $S \in \text{sets lebesgue}$ 
  shows simple_function_lebesgue ( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ )
proof -
  have ffin: finite (range f) and fsets:  $\forall x. f - \{f x\} \in \text{sets lebesgue}$ 
    using f by (auto simp: simple_function_def)
  have finite (f ' S)
    by (meson finite_subset_subset_image_iff ffin top_greatest)
  moreover have finite (( $\lambda x. 0::real$ ) ' T) for T :: 'a set
    by (auto simp: image_def)
  moreover have if_sets: ( $\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0$ ) - \{f a\} \in \text{sets lebesgue}

```



```

for a
proof -
  have *:  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) -' \{f a\}$ 
    =  $(\text{if } f a = 0 \text{ then } -S \cup f -' \{f a\} \text{ else } (f -' \{f a\}) \cap S)$ 
  by (auto simp: split: if_split_asm)
  show ?thesis
    unfolding * by (metis Compl_in_sets_lebesgue S sets.Int sets.Un fsets)
qed
moreover have  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) -' \{0\} \in \text{sets lebesgue}$ 
proof (cases  $0 \in \text{range } f$ )
  case True
  then show ?thesis
    by (metis (no_types, lifting) if_sets rangeE)
  next
  case False
  then have  $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0) -' \{0\} = -S$ 
    by auto
  then show ?thesis
    by (simp add: Compl_in_sets_lebesgue S)
qed
ultimately show ?thesis
  by (auto simp: simple_function_def)
qed

corollary simple_function_measurable_on:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $f$ :  $\text{simple\_function lebesgue } f$  and  $nn$ :  $\bigwedge x. f x \geq 0$  and  $S$ :  $S \in \text{sets lebesgue}$ 
  shows  $f$  measurable_on  $S$ 
  by (simp add: measurable_on_UNIV [symmetric, of f] S f simple_function_lebesgue_if nn simple_function_measurable_on_UNIV)

lemma
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{ordered\_euclidean\_space}$ 
  assumes  $f$ :  $f$  measurable_on  $S$  and  $g$ :  $g$  measurable_on  $S$ 
  shows measurable_on_sup:  $(\lambda x. \text{sup } (f x) (g x))$  measurable_on  $S$ 
  and measurable_on_inf:  $(\lambda x. \text{inf } (f x) (g x))$  measurable_on  $S$ 
proof -
  obtain  $NF$  and  $F$ 
  where  $NF$ : negligible  $NF$ 
  and  $conF$ :  $\bigwedge n. \text{continuous\_on UNIV } (F n)$ 
  and  $tendsF$ :  $\bigwedge x. x \notin NF \implies (\lambda n. F n x) \longrightarrow (\text{if } x \in S \text{ then } f x \text{ else } 0)$ 
  using  $f$  by (auto simp: measurable_on_def)
  obtain  $NG$  and  $G$ 
  where  $NG$ : negligible  $NG$ 
  and  $conG$ :  $\bigwedge n. \text{continuous\_on UNIV } (G n)$ 
  and  $tendsG$ :  $\bigwedge x. x \notin NG \implies (\lambda n. G n x) \longrightarrow (\text{if } x \in S \text{ then } g x \text{ else } 0)$ 
  using  $g$  by (auto simp: measurable_on_def)
  show  $(\lambda x. \text{sup } (f x) (g x))$  measurable_on  $S$ 

```

```

unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show continuous_on UNIV ( $\lambda x. \sup (F n x) (G n x)$ ) for  $n$ 
    unfolding sup_max eucl_sup by (intro conF conG continuous_intros)
  show ( $\lambda n. \sup (F n x) (G n x)$ )  $\longrightarrow$  (if  $x \in S$  then  $\sup (f x) (g x)$  else 0)
    if  $x \notin NF \cup NG$  for  $x$ 
    using tendsto_sup [OF tendsF tendsG, of  $x x$ ] that by auto
qed (simp add: NF NG)
show ( $\lambda x. \inf (f x) (g x)$ ) measurable_on S
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show continuous_on UNIV ( $\lambda x. \inf (F n x) (G n x)$ ) for  $n$ 
    unfolding inf_min eucl_inf by (intro conF conG continuous_intros)
  show ( $\lambda n. \inf (F n x) (G n x)$ )  $\longrightarrow$  (if  $x \in S$  then  $\inf (f x) (g x)$  else 0)
    if  $x \notin NF \cup NG$  for  $x$ 
    using tendsto_inf [OF tendsF tendsG, of  $x x$ ] that by auto
qed (simp add: NF NG)
qed

proposition measurable_on_componentwise_UNIV:
   $f$  measurable_on UNIV  $\longleftrightarrow$  ( $\forall i \in \text{Basis}. (\lambda x. (f x \cdot i) *_R i)$  measurable_on UNIV)
  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof
    fix  $i::'b$ 
    assume  $i \in \text{Basis}$ 
    have cont: continuous_on UNIV ( $\lambda x. (x \cdot i) *_R i$ )
      by (intro continuous_intros)
    show ( $\lambda x. (f x \cdot i) *_R i$ ) measurable_on UNIV
      using measurable_on_compose_continuous [OF L cont]
      by (simp add: o_def)
  qed
next
  assume ?rhs
  then have  $\exists N g. \text{negligible } N \wedge$ 
    ( $\forall n. \text{continuous\_on UNIV } (g n)$ )  $\wedge$ 
    ( $\forall x. x \notin N \longrightarrow (\lambda n. g n x) \longrightarrow (f x \cdot i) *_R i$ )
  if  $i \in \text{Basis}$  for  $i$ 
  by (simp add: measurable_on_def that)
then obtain  $N g$  where  $N: \bigwedge i. i \in \text{Basis} \implies \text{negligible } (N i)$ 
  and cont:  $\bigwedge i n. i \in \text{Basis} \implies \text{continuous\_on UNIV } (g i n)$ 
  and tends:  $\bigwedge i x. \llbracket i \in \text{Basis}; x \notin N i \rrbracket \implies (\lambda n. g i n x) \longrightarrow (f x \cdot i) *_R i$ 
  by metis
show ?lhs
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show negligible ( $\bigcup i \in \text{Basis}. N i$ )

```

```

    using N eucl.finite_Basis by blast
  show continuous_on UNIV ( $\lambda x. (\sum_{i \in \text{Basis}} g\ i\ n\ x)$ ) for n
    by (intro continuous_intros cont)
next
  fix x
  assume  $x \notin (\bigcup_{i \in \text{Basis}} N\ i)$ 
  then have  $\bigwedge i. i \in \text{Basis} \implies x \notin N\ i$ 
    by auto
  then have  $(\lambda n. (\sum_{i \in \text{Basis}} g\ i\ n\ x)) \longrightarrow (\sum_{i \in \text{Basis}} (f\ x \cdot i) *_{\mathbb{R}} i)$ 
    by (intro tends_tendsto_intros)
  then show  $(\lambda n. (\sum_{i \in \text{Basis}} g\ i\ n\ x)) \longrightarrow (\text{if } x \in \text{UNIV} \text{ then } f\ x \text{ else } 0)$ 
    by (simp add: euclidean_representation)
qed
qed

```

corollary *measurable_on_componentwise*:

```

  f measurable_on S  $\longleftrightarrow (\forall i \in \text{Basis}. (\lambda x. (f\ x \cdot i) *_{\mathbb{R}} i)$  measurable_on S)
  apply (subst measurable_on_UNIV [symmetric])
  apply (subst measurable_on_componentwise_UNIV)
  apply (simp add: measurable_on_UNIV if_distrib [of  $\lambda x. \text{inner } x \_$ ] if_distrib
    [of  $\lambda x. \text{scaleR } x \_$ ] cong: if_cong)
  done

```

lemma *borel_measurable_implies_simple_function_sequence_real*:

```

  fixes u :: 'a  $\Rightarrow$  real
  assumes u[measurable]:  $u \in \text{borel\_measurable } M$  and nn:  $\bigwedge x. u\ x \geq 0$ 
  shows  $\exists f. \text{incseq } f \wedge (\forall i. \text{simple\_function } M\ (f\ i)) \wedge (\forall x. \text{bdd\_above } (\text{range }
    (\lambda i. f\ i\ x))) \wedge$ 
    ( $\forall i\ x. 0 \leq f\ i\ x \wedge u = (\text{SUP } i. f\ i)$ )

```

proof –

```

  define f where [abs_def]:
     $f\ i\ x = \text{real\_of\_int } (\text{floor } ((\text{min } i\ (u\ x)) * 2^i)) / 2^i$  for i x

```

```

  have [simp]:  $0 \leq f\ i\ x$  for i x
    by (auto simp: f_def intro!: divide_nonneg_nonneg mult_nonneg_nonneg nn)

```

```

  have *:  $2^n * \text{real\_of\_int } x = \text{real\_of\_int } (2^n * x)$  for n x
    by simp

```

```

  have  $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = \text{real\_of\_int } \lfloor i * 2^i \rfloor$  for i
    by (intro arg_cong[where f=real_of_int]) simp
  then have [simp]:  $\text{real\_of\_int } \lfloor \text{real } i * 2^i \rfloor = i * 2^i$  for i
    unfolding floor_of_nat by simp

```

```

  have bdd:  $\text{bdd\_above } (\text{range } (\lambda i. f\ i\ x))$  for x
    by (rule bdd_aboveI [where M = u x]) (auto simp: f_def field_simps min_def)

```

```

have incseq f
proof (intro monoI le_funI)
  fix m n :: nat and x assume m ≤ n
  moreover
  { fix d :: nat
    have [2^d::real] * [2^m * (min (of_nat m) (u x))] ≤ [2^d * (2^m * (min
(of_nat m) (u x)))]
      by (rule le_mult_floor) (auto simp: nn)
    also have ... ≤ [2^d * (2^m * (min (of_nat d + of_nat m) (u x)))]
      by (intro floor_mono mult_mono min_mono)
        (auto simp: nn min_less_iff_disj of_nat_less_top)
    finally have f m x ≤ f(m + d) x
      unfolding f_def
      by (auto simp: field_simps power_add * simp del: of_int_mult) }
  ultimately show f m x ≤ f n x
  by (auto simp: le_iff_add)
qed
then have inc_f: incseq (λi. f i x) for x
  by (auto simp: incseq_def le_fun_def)
moreover
have simple_function M (f i) for i
proof (rule simple_function_borel_measurable)
  have [(min (of_nat i) (u x)) * 2^i] ≤ [int i * 2^i] for x
    by (auto split: split_min intro!: floor_mono)
  then have f i ' space M ⊆ (λn. real_of_int n / 2^i) ' {0 .. of_nat i * 2^i}
    unfolding floor_of_int by (auto simp: f_def nn intro!: imageI)
  then show finite (f i ' space M)
    by (rule finite_subset) auto
  show f i ∈ borel_measurable M
    unfolding f_def enn2real_def by measurable
qed
moreover
{ fix x
  have (SUP i. (f i x)) = u x
  proof -
    obtain n where u x ≤ of_nat n using real_arch_simple by auto
    then have min_eq_r: ∀_F i in sequentially. min (real i) (u x) = u x
      by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)
    have (λi. real_of_int [min (real i) (u x) * 2^i] / 2^i) → u x
      proof (rule tendsto_sandwich)
        show (λn. u x - (1/2)^n) → u x
          by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
        show ∀_F n in sequentially. real_of_int [min (real n) (u x) * 2^n] / 2^n
n ≤ u x
          using min_eq_r by eventually_elim (auto simp: field_simps)
        have *: u x * (2^n * 2^n) ≤ 2^n + 2^n * real_of_int [u x * 2^n] for
n
          using real_of_int_floor_ge_diff_one[of u x * 2^n, THEN mult_left_mono,
of 2^n]

```

```

    by (auto simp: field_simps)
    show  $\forall_F n$  in sequentially.  $u x - (1/2)^n \leq \text{real\_of\_int } \lfloor \min(\text{real } n) (u x) * 2^n \rfloor / 2^n$ 
    using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
  qed auto
  then have  $(\lambda i. (f i x)) \longrightarrow u x$ 
  by (simp add: f_def)
  from LIMSEQ_unique LIMSEQ_incseq_SUP [OF bdd_inc_f] this
  show ?thesis
  by blast
  qed }
  ultimately show ?thesis
  by (intro exI [of _  $\lambda i x. f i x$ ] (auto simp: <incseq f> bdd_image_comp))
  qed

```

lemma *homeomorphic_open_interval_UNIV*:

```

  fixes a b:: real
  assumes a < b
  shows {a<..b} homeomorphic (UNIV::real set)
  proof -
    have {a<..b} = ball ((b+a) / 2) ((b-a) / 2)
    using assms
    by (auto simp: dist_real_def abs_if field_split_simps split: if_split_asm)
  then show ?thesis
  by (simp add: homeomorphic_ball_UNIV assms)
  qed

```

proposition *homeomorphic_box_UNIV*:

```

  fixes a b:: 'a::euclidean_space
  assumes box a b  $\neq \{\}$ 
  shows box a b homeomorphic (UNIV::'a set)
  proof -
    have {a  $\cdot$  i <..b  $\cdot$  i} homeomorphic (UNIV::real set) if i  $\in$  Basis for i
    using assms box_ne_empty that by (blast intro: homeomorphic_open_interval_UNIV)
    then have  $\exists f g. (\forall x. a \cdot i < x \wedge x < b \cdot i \longrightarrow g (f x) = x) \wedge$ 
       $(\forall y. a \cdot i < g y \wedge g y < b \cdot i \wedge f(g y) = y) \wedge$ 
      continuous_on {a  $\cdot$  i <..b  $\cdot$  i} f  $\wedge$ 
      continuous_on (UNIV::real set) g
    if i  $\in$  Basis for i
    using that by (auto simp: homeomorphic_minimal_mem_box Ball_def)
    then obtain f g where gf:  $\bigwedge i x. \llbracket i \in \text{Basis}; a \cdot i < x; x < b \cdot i \rrbracket \Longrightarrow g i (f i x) = x$ 
      and fg:  $\bigwedge i y. i \in \text{Basis} \Longrightarrow a \cdot i < g i y \wedge g i y < b \cdot i \wedge f i (g i y) = y$ 
    and contf:  $\bigwedge i. i \in \text{Basis} \Longrightarrow \text{continuous\_on } \{a \cdot i <..b \cdot i\} (f i)$ 
    and contg:  $\bigwedge i. i \in \text{Basis} \Longrightarrow \text{continuous\_on } (\text{UNIV::real set}) (g i)$ 
    by metis
    define F where F  $\equiv \lambda x. \sum_{i \in \text{Basis}} (f i (x \cdot i)) *_R i$ 

```

```

define G where  $G \equiv \lambda x. \sum_{i \in \text{Basis}} (g\ i\ (x \cdot i)) *_{\mathbb{R}} i$ 
show ?thesis
  unfolding homeomorphic_minimal
proof (intro exI conjI ballI)
  show  $G\ y \in \text{box}\ a\ b$  for y
    using fg by (simp add: G_def mem_box)
  show  $G\ (F\ x) = x$  if  $x \in \text{box}\ a\ b$  for x
    using that by (simp add: F_def G_def gf mem_box euclidean_representation)
  show  $F\ (G\ y) = y$  for y
    by (simp add: F_def G_def fg mem_box euclidean_representation)
  show continuous_on ( $\text{box}\ a\ b$ ) F
    unfolding F_def
  proof (intro continuous_intros continuous_on_compose2 [OF contf continuous_on_inner])
    show  $(\lambda x. x \cdot i) \text{ ' box}\ a\ b \subseteq \{a \cdot i <..< b \cdot i\}$  if i  $\in \text{Basis}$  for i
      using that by (auto simp: mem_box)
    qed
  show continuous_on UNIV G
    unfolding G_def
    by (intro continuous_intros continuous_on_compose2 [OF contg continuous_on_inner]) auto
  qed auto
qed

```

lemma *diff_null_sets_lebesgue*: $\llbracket N \in \text{null_sets}\ (\text{lebesgue_on}\ S); X - N \in \text{sets}\ (\text{lebesgue_on}\ S); N \subseteq X \rrbracket$
 $\implies X \in \text{sets}\ (\text{lebesgue_on}\ S)$
by (*metis Int_Diff_Un inf commute inf.orderE null_setsD2 sets.Un*)

lemma *borel_measurable_diff_null*:
fixes *f* $:: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *N*: $N \in \text{null_sets}\ (\text{lebesgue_on}\ S)$ **and** *S*: $S \in \text{sets}\ \text{lebesgue}$
shows $f \in \text{borel_measurable}\ (\text{lebesgue_on}\ (S - N)) \longleftrightarrow f \in \text{borel_measurable}\ (\text{lebesgue_on}\ S)$
unfolding *in_borel_measurable_space_lebesgue_on_sets_restrict_UNIV*
proof (*intro ball_cong iffI*)
show $f \text{ - ' } T \cap S \in \text{sets}\ (\text{lebesgue_on}\ S)$
if $f \text{ - ' } T \cap (S - N) \in \text{sets}\ (\text{lebesgue_on}\ (S - N))$ **for** *T*
proof -
have $N \cap S = N$
by (*metis N S inf.orderE null_sets_restrict_space*)
moreover **have** $N \cap S \in \text{sets}\ \text{lebesgue}$
by (*metis N S inf.orderE null_setsD2 null_sets_restrict_space*)
moreover **have** $f \text{ - ' } T \cap S \cap (f \text{ - ' } T \cap N) \in \text{sets}\ \text{lebesgue}$
by (*metis N S completion.complete inf.absorb2 inf_le2 inf_mono null_sets_restrict_space*)
ultimately **show** *?thesis*
by (*metis Diff_Int_distrib Int_Diff_Un S inf_le2 sets.Diff sets.Un sets_restrict_space_iff*)

```

space_lebesgue_on space_restrict_space that)
qed
show  $f^{-1} T \cap (S-N) \in \text{sets } (\text{lebesgue\_on } (S-N))$ 
  if  $f^{-1} T \cap S \in \text{sets } (\text{lebesgue\_on } S)$  for  $T$ 
proof -
  have  $(S - N) \cap f^{-1} T = (S - N) \cap (f^{-1} T \cap S)$ 
  by blast
  then have  $(S - N) \cap f^{-1} T \in \text{sets.restricted\_space } \text{lebesgue } (S - N)$ 
  by (metis S image_iff sets.Int_space_eq2 sets_restrict_space_iff that)
  then show ?thesis
  by (simp add: inf commute sets_restrict_space)
qed
qed auto

lemma lebesgue_measurable_diff_null:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $N \in \text{null\_sets } \text{lebesgue}$ 
  shows  $f \in \text{borel\_measurable } (\text{lebesgue\_on } (-N)) \iff f \in \text{borel\_measurable } \text{lebesgue}$ 
  by (simp add: Compl_eq_Diff_UNIV assms borel_measurable_diff_null lebesgue_on_UNIV_eq)

proposition measurable_on_imp_borel_measurable_lebesgue_UNIV:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes  $f \text{ measurable\_on } UNIV$ 
  shows  $f \in \text{borel\_measurable } \text{lebesgue}$ 
proof -
  obtain  $N$  and  $F$ 
  where  $NF$ : negligible  $N$ 
  and  $conF$ :  $\bigwedge n. \text{continuous\_on } UNIV (F n)$ 
  and  $tendsF$ :  $\bigwedge x. x \notin N \implies (\lambda n. F n x) \longrightarrow f x$ 
  using assms by (auto simp: measurable_on_def)
  obtain  $N$  where  $N \in \text{null\_sets } \text{lebesgue}$   $f \in \text{borel\_measurable } (\text{lebesgue\_on } (-N))$ 
  proof
    show  $f \in \text{borel\_measurable } (\text{lebesgue\_on } (-N))$ 
    proof (rule borel_measurable_LIMSEQ_metric)
      show  $F i \in \text{borel\_measurable } (\text{lebesgue\_on } (-N))$  for  $i$ 
      by (meson Compl_in_sets_lebesgue NF conF continuous_imp_measurable_on_sets_lebesgue
        continuous_on_subset negligible_imp_sets_subset_UNIV)
      show  $(\lambda i. F i x) \longrightarrow f x$  if  $x \in \text{space } (\text{lebesgue\_on } (-N))$  for  $x$ 
      using that
      by (simp add: tendsF)
    qed
  qed
  show  $N \in \text{null\_sets } \text{lebesgue}$ 
  using  $NF$  negligible_iff_null_sets by blast
qed
then show ?thesis

```

using *lebesgue_measurable_diff_null* by *blast*
 qed

corollary *measurable_on_imp_borel_measurable_lebesgue*:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
 assumes f measurable_on S and $S: S \in \text{sets lebesgue}$
 shows $f \in \text{borel_measurable (lebesgue_on } S)$

proof –

have $(\lambda x. \text{if } x \in S \text{ then } f x \text{ else } 0)$ measurable_on *UNIV*

using *assms(1) measurable_on_UNIV* by *blast*

then show *?thesis*

by (*simp add: borel_measurable_if_D measurable_on_imp_borel_measurable_lebesgue_UNIV*)

qed

proposition *measurable_on_limit*:

fixes $f :: \text{nat} \Rightarrow 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
 assumes $f: \bigwedge n. f n$ measurable_on S and $N: \text{negligible } N$
 and $\text{lim}: \bigwedge x. x \in S - N \implies (\lambda n. f n x) \longrightarrow g x$
 shows g measurable_on S

proof –

have *box* $(0::'b)$ *One* homeomorphic (*UNIV::'b set*)

by (*simp add: homeomorphic_box_UNIV*)

then obtain $h h': 'b \Rightarrow 'b$ where $hh': \bigwedge x. x \in \text{box } 0 \text{ One} \implies h (h' x) = x$

and $h'im: h' \text{ ' box } 0 \text{ One} = \text{UNIV}$

and *conth*: continuous_on *UNIV* h

and *conth'*: continuous_on $(\text{box } 0 \text{ One}) h'$

and $h'h: \bigwedge y. h' (h y) = y$

and *rangeh*: range $h = \text{box } 0 \text{ One}$

by (*auto simp: homeomorphic_def homeomorphism_def*)

have *norm* $y \leq \text{DIM}('b)$ if $y: y \in \text{box } 0 \text{ One}$ for $y::'b$

proof –

have *y01*: $0 < y \cdot i \cdot i < 1$ if $i \in \text{Basis}$ for i

using *that y* by (*auto simp: mem_box*)

have *norm* $y \leq (\sum i \in \text{Basis}. |y \cdot i|)$

using *norm_le_l1* by *blast*

also have $\dots \leq (\sum i::'b \in \text{Basis}. 1)$

proof (*rule sum_mono*)

show $|y \cdot i| \leq 1$ if $i \in \text{Basis}$ for i

using *y01 that* by *fastforce*

qed

also have $\dots \leq \text{DIM}('b)$

by *auto*

finally show *?thesis* .

qed

then have *norm_le*: $\text{norm}(h y) \leq \text{DIM}('b)$ for y

by (*metis UNIV_I image_eqI rangeh*)

have $(h' \circ (h \circ (\lambda x. \text{if } x \in S \text{ then } g x \text{ else } 0)))$ measurable_on *UNIV*

proof (*rule measurable_on_compose_continuous_box*)


```

let ? $\chi$  = h  $\circ$  ( $\lambda$ x. if x  $\in$  S then g x else 0)
let ?f =  $\lambda$ n. h  $\circ$  ( $\lambda$ x. if x  $\in$  S then f n x else 0)
show ? $\chi$  measurable_on UNIV
proof (rule integrable_subintervals_imp_measurable)
  show ? $\chi$  integrable_on cbox a b for a b
  proof (rule integrable_spike_set)
    show ? $\chi$  integrable_on (cbox a b - N)
    proof (rule dominated_convergence_integrable)
      show const: ( $\lambda$ x. DIM('b)) integrable_on cbox a b - N
      by (simp add: N has_integral_iff_integrable_const integrable_negligible
integrable_setdiff negligible_diff)
      show norm ((h  $\circ$  ( $\lambda$ x. if x  $\in$  S then g x else 0)) x)  $\leq$  DIM('b) if x  $\in$  cbox
a b - N for x
      using that norm_le by (simp add: o_def)
      show ( $\lambda$ k. ?f k x)  $\longrightarrow$  ? $\chi$  x if x  $\in$  cbox a b - N for x
      using that lim [of x] conth
      by (auto simp: continuous_on_def intro: tendsto_compose)
      show (?f n) absolutely_integrable_on cbox a b - N for n
    proof (rule measurable_bounded_by_integrable_imp_absolutely_integrable)
      show ?f n  $\in$  borel_measurable (lebesgue_on (cbox a b - N))
      proof (rule measurable_on_imp_borel_measurable_lebesgue [OF mea-
surable_on_spike_set])
        show ?f n measurable_on cbox a b
        unfolding measurable_on_UNIV [symmetric, of _ cbox a b]
        proof (rule measurable_on_restrict)
          have f': ( $\lambda$ x. if x  $\in$  S then f n x else 0) measurable_on UNIV
          by (simp add: f measurable_on_UNIV)
          show ?f n measurable_on UNIV
          using measurable_on_compose_continuous [OF f' conth] by auto
        qed auto
        show negligible (sym_diff (cbox a b) (cbox a b - N))
        by (auto intro: negligible_subset [OF N])
        show cbox a b - N  $\in$  sets lebesgue
        by (simp add: N negligible_imp_sets sets.Diff)
      qed
      show cbox a b - N  $\in$  sets lebesgue
      by (simp add: N negligible_imp_sets sets.Diff)
      show norm (?f n x)  $\leq$  DIM('b)
      if x  $\in$  cbox a b - N for x
      using that local.norm_le by simp
    qed (auto simp: const)
  qed
  show negligible {x  $\in$  cbox a b - N - cbox a b. ? $\chi$  x  $\neq$  0}
  by (auto simp: empty_imp_negligible)
  have {x  $\in$  cbox a b - (cbox a b - N). ? $\chi$  x  $\neq$  0}  $\subseteq$  N
  by auto
  then show negligible {x  $\in$  cbox a b - (cbox a b - N). ? $\chi$  x  $\neq$  0}
  using N negligible_subset by blast
qed

```

3254

```
qed
show ? $\chi$   $x \in \text{box } 0 \text{ One}$  for  $x$ 
  using rangeh by auto
show continuous_on ( $\text{box } 0 \text{ One}$ )  $h'$ 
  by (rule conth')
qed
then show ?thesis
  by (simp add: o_def h'h measurable_on_UNIV)
qed
```

```
lemma measurable_on_if_simple_function_limit:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  shows  $\llbracket \bigwedge n. g \ n \ \text{measurable\_on} \ UNIV; \bigwedge n. \text{finite} \ (\text{range} \ (g \ n)); \bigwedge x. (\lambda n. g \ n \ x) \longrightarrow f \ x \rrbracket$ 
 $\implies f \ \text{measurable\_on} \ UNIV$ 
  by (force intro: measurable_on_limit [where N={}])
```

```
lemma lebesgue_measurable_imp_measurable_on_nnreal_UNIV:
  fixes  $u :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $u \in \text{borel\_measurable\_lebesgue}$  and  $nn: \bigwedge x. u \ x \geq 0$ 
  shows  $u \ \text{measurable\_on} \ UNIV$ 
proof -
  obtain  $f$  where incseq  $f$  and  $f: \forall i. \text{simple\_function\_lebesgue} \ (f \ i)$ 
  and  $bdd: \bigwedge x. \text{bdd\_above} \ (\text{range} \ (\lambda i. f \ i \ x))$ 
  and  $nnf: \bigwedge i \ x. 0 \leq f \ i \ x$  and  $*$ :  $u = (\text{SUP } i. f \ i)$ 
  using borel_measurable_implies_simple_function_sequence_real  $nn \ u$  by metis
  show ?thesis
  unfolding  $*$ 
  proof (rule measurable_on_if_simple_function_limit [of concl: Sup (range f)])
    show  $(f \ i) \ \text{measurable\_on} \ UNIV$  for  $i$ 
      by (simp add: f nnf simple_function_measurable_on_UNIV)
    show finite ( $\text{range} \ (f \ i)$ ) for  $i$ 
      by (metis f simple_function_def space_borel space_completion space_lborel)
    show  $(\lambda i. f \ i \ x) \longrightarrow \text{Sup} \ (\text{range} \ f) \ x$  for  $x$ 
      proof -
        have incseq  $(\lambda i. f \ i \ x)$ 
          using  $\langle \text{incseq } f \rangle$  apply (auto simp: incseq_def)
          by (simp add: le_funD)
        then show ?thesis
          by (metis SUP_apply bdd LIMSEQ_incseq_SUP)
      qed
    qed
  qed
qed
```

```
lemma lebesgue_measurable_imp_measurable_on_nnreal:
  fixes  $u :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes  $u \in \text{borel\_measurable\_lebesgue}$   $\bigwedge x. u \ x \geq 0$   $S \in \text{sets\_lebesgue}$ 
```

```

shows  $u$  measurable_on  $S$ 
unfolding measurable_on_UNIV [symmetric, of  $u$ ]
using assms
by (auto intro: lebesgue_measurable_imp_measurable_on_nnreal_UNIV)

```

lemma *lebesgue_measurable_imp_measurable_on_real*:

```

fixes  $u :: 'a::euclidean\_space \Rightarrow real$ 
assumes  $u: u \in \text{borel\_measurable\_lebesgue}$  and  $S: S \in \text{sets\_lebesgue}$ 
shows  $u$  measurable_on  $S$ 

```

proof –

```

let  $?f = \lambda x. |u\ x| + u\ x$ 
let  $?g = \lambda x. |u\ x| - u\ x$ 
have  $?f$  measurable_on  $S$   $?g$  measurable_on  $S$ 
  using  $S\ u$  by (auto intro: lebesgue_measurable_imp_measurable_on_nnreal)
then have  $(\lambda x. (?f\ x - ?g\ x) / 2)$  measurable_on  $S$ 
  using measurable_on_cdivide measurable_on_diff by blast
then show  $?thesis$ 
  by auto

```

qed

proposition *lebesgue_measurable_imp_measurable_on*:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $f: f \in \text{borel\_measurable\_lebesgue}$  and  $S: S \in \text{sets\_lebesgue}$ 
shows  $f$  measurable_on  $S$ 
unfolding measurable_on_componentwise [of  $f$ ]

```

proof

```

fix  $i :: 'b$ 
assume  $i \in \text{Basis}$ 
have  $(\lambda x. (f\ x \cdot i)) \in \text{borel\_measurable\_lebesgue}$ 
  using  $\langle i \in \text{Basis} \rangle$  borel_measurable_euclidean_space  $f$  by blast
then have  $(\lambda x. (f\ x \cdot i))$  measurable_on  $S$ 
  using  $S$  lebesgue_measurable_imp_measurable_on_real by blast
then show  $(\lambda x. (f\ x \cdot i) *_{\mathbb{R}} i)$  measurable_on  $S$ 
  by (intro measurable_on_scaleR measurable_on_const  $S$ )

```

qed

proposition *measurable_on_iff_borel_measurable*:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
assumes  $S \in \text{sets\_lebesgue}$ 
shows  $f$  measurable_on  $S \iff f \in \text{borel\_measurable (lebesgue\_on } S)$  (is  $?lhs = ?rhs$ )

```

proof

```

show  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
  if  $f$  measurable_on  $S$ 
  using that by (simp add: assms measurable_on_imp_borel_measurable_lebesgue)

```

next

```

assume  $f \in \text{borel\_measurable (lebesgue\_on } S)$ 
then have  $(\lambda a. \text{if } a \in S \text{ then } f\ a \text{ else } 0)$  measurable_on UNIV

```

```

  by (simp add: assms borel_measurable_if_lebesgue_measurable_imp_measurable_on)
  then show f measurable_on S
    using measurable_on_UNIV by blast
qed

```

9.16.5 Monotonic functions are Lebesgue integrable

lemma *integrable_mono_on_nonneg*:

```

  fixes f :: real  $\Rightarrow$  real
  assumes mon: mono_on {a..b} f and 0:  $\bigwedge x. 0 \leq f x$ 
  shows integrable (lebesgue_on {a..b}) f
  proof -
    have space lborel = space lebesgue sets borel  $\subseteq$  sets lebesgue
      by force+
    then have fborel: f  $\in$  borel_measurable (lebesgue_on {a..b})
      by (metis mon borel_measurable_mono_on_fnc borel_measurable_subalgebra
        mono_restrict_space space_lborel space_restrict_space)
    then obtain g where g: incseq g and simple:  $\bigwedge i. \text{simple\_function } (\text{lebesgue\_on } \{a..b\}) (g i)$ 
      and bdd:  $(\forall x. \text{bdd\_above } (\text{range } (\lambda i. g i x)))$  and nonneg:  $\forall i x. 0 \leq g i x$ 
      and fsup: f = (SUP i. g i)
    by (metis borel_measurable_implies_simple_function_sequence_real 0)
    have f' : {a..b}  $\subseteq$  {f a..f b}
      using assms by (auto simp: mono_on_def)
    have g_le_f: g i x  $\leq$  f x for i x
    proof -
      have bdd_above (( $\lambda h. h x$ ) ' range g)
        using bdd cSUP_lessD linorder_not_less by fastforce
      then show ?thesis
        by (metis SUP_apply UNIV_I bdd cSUP_upper fsup)
    qed
    then have gfb: g i x  $\leq$  f b if x  $\in$  {a..b} for i x
      by (smt (verit, best) mon atLeastAtMost_iff mono_on_def that)
    have g_le: g i x  $\leq$  g j x if i  $\leq$  j for i j x
      using g by (simp add: incseq_def le_funD that)
    show integrable (lebesgue_on {a..b}) (f)
    proof (rule integrable_dominated_convergence)
      show f  $\in$  borel_measurable (lebesgue_on {a..b})
        using fborel by blast
      have  $\bigwedge x. (\lambda i. g i x) \longrightarrow (\text{SUP } h \in \text{range } g. h x)$ 
      proof (rule order_tendstoI)
        show  $\forall_F i$  in sequentially. y < g i x
          if y < (SUP h  $\in$  range g. h x) for x y
        proof -
          from that obtain h where h: h  $\in$  range g y < h x
            using g_le_f by (subst (asm) less_cSUP_iff) fastforce+
          then show ?thesis
            by (smt (verit, ccfv_SIG) eventually_sequentially g_le imageE)
        qed
      qed
    end
  end

```

```

qed
show  $\forall_F i$  in sequentially.  $g\ i\ x < y$ 
  if  $(SUP\ h \in range\ g.\ h\ x) < y$  for  $x\ y$ 
  by (smt (verit, best) that Sup_apply g_le_f always_eventually fsup im-
age_cong)
qed
then show AE  $x$  in lebesgue_on  $\{a..b\}$ .  $(\lambda i.\ g\ i\ x) \longrightarrow f\ x$ 
  by (simp add: fsup)
fix  $i$ 
show  $g\ i \in borel\_measurable\ (lebesgue\_on\ \{a..b\})$ 
  using borel_measurable_simple_function simple by blast
show AE  $x$  in lebesgue_on  $\{a..b\}$ .  $norm\ (g\ i\ x) \leq f\ b$ 
  by (simp add: gfb nonneg Measure_Space.AE_I' [of {}])
qed auto
qed

```

lemma integrable_mono_on:

```

fixes  $f :: real \Rightarrow real$ 
assumes mono_on  $\{a..b\}$   $f$ 
shows integrable (lebesgue_on  $\{a..b\}$ )  $f$ 
proof -
define  $f'$  where  $f' \equiv \lambda x.$  if  $x \in \{a..b\}$  then  $f\ x - f\ a$  else  $0$ 
have mono_on  $\{a..b\}$   $f'$ 
  by (smt (verit, best) assms  $f'_def$  mono_on_def)
moreover have  $0: \bigwedge x.$   $0 \leq f'\ x$ 
  by (smt (verit, best) assms atLeastAtMost_iff  $f'_def$  mono_on_def)
ultimately have integrable (lebesgue_on  $\{a..b\}$ )  $f'$ 
  using integrable_mono_on_nonneg by presburger
then have integrable (lebesgue_on  $\{a..b\}$ )  $(\lambda x.$   $f'\ x + f\ a)$ 
  by force
moreover have space lborel = space lebesgue sets borel  $\subseteq$  sets lebesgue
  by force+
then have fborel:  $f \in borel\_measurable\ (lebesgue\_on\ \{a..b\})$ 
  by (metis assms borel_measurable_mono_on_fnc borel_measurable_subalgebra
mono_restrict_space space_lborel space_restrict_space)
ultimately show ?thesis
  by (rule integrable_cong_AE_imp) (auto simp add:  $f'_def$ )
qed

```

lemma integrable_on_mono_on:

```

fixes  $f :: real \Rightarrow real$ 
assumes mono_on  $\{a..b\}$   $f$ 
shows  $f$  integrable_on  $\{a..b\}$ 
  by (simp add: assms integrable_mono_on integrable_on_lebesgue_on)

```

9.16.6 Measurability on generalisations of the binary product

lemma measurable_on_bilinear:

```

fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::euclidean_space
assumes h: bilinear h and f: f measurable_on S and g: g measurable_on S
shows ( $\lambda x. h (f x) (g x)$ ) measurable_on S
proof (rule measurable_on_combine [where h = h])
  show continuous_on UNIV ( $\lambda x. h (fst x) (snd x)$ )
    by (simp add: bilinear_continuous_on_compose [OF continuous_on_fst continuous_on_snd h])
  show h 0 0 = 0
  by (simp add: bilinear_lzero h)
qed (auto intro: assms)

```

lemma borel_measurable_bilinear:

```

fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::euclidean_space
assumes bilinear h f  $\in$  borel_measurable (lebesgue_on S) g  $\in$  borel_measurable (lebesgue_on S)
and S: S  $\in$  sets lebesgue
shows ( $\lambda x. h (f x) (g x)$ )  $\in$  borel_measurable (lebesgue_on S)
using assms measurable_on_bilinear [of h f S g]
by (simp flip: measurable_on_iff_borel_measurable)

```

lemma absolutely_integrable_bounded_measurable_product:

```

fixes h :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space  $\Rightarrow$  'c::euclidean_space
assumes bilinear h and f: f  $\in$  borel_measurable (lebesgue_on S) S  $\in$  sets lebesgue
and bou: bounded (f ' S) and g: g absolutely_integrable_on S
shows ( $\lambda x. h (f x) (g x)$ ) absolutely_integrable_on S
proof -
  obtain B where B > 0 and B:  $\bigwedge x y. \text{norm } (h x y) \leq B * \text{norm } x * \text{norm } y$ 
    using bilinear_bounded_pos <bilinear h> by blast
  obtain C where C > 0 and C:  $\bigwedge x. x \in S \implies \text{norm } (f x) \leq C$ 
    using bounded_pos by (metis bou imageI)
  show ?thesis
  proof (rule measurable_bounded_by_integrable_imp_absolutely_integrable [OF _ <S  $\in$  sets lebesgue>])
    show norm (h (f x) (g x))  $\leq B * C * \text{norm}(g x)$  if x  $\in$  S for x
      by (meson less_le mult_left_mono mult_right_mono norm_ge_zero order_trans that <B > 0> B C)
    show ( $\lambda x. h (f x) (g x)$ )  $\in$  borel_measurable (lebesgue_on S)
      using <bilinear h> f g
    by (blast intro: borel_measurable_bilinear dest: absolutely_integrable_measurable)
  show ( $\lambda x. B * C * \text{norm}(g x)$ ) integrable_on S
    using <0 < B> <0 < C> absolutely_integrable_on_def g by auto
qed
qed

```

lemma absolutely_integrable_bounded_measurable_product_real:

```

fixes f :: real  $\Rightarrow$  real
assumes f  $\in$  borel_measurable (lebesgue_on S) S  $\in$  sets lebesgue
and bounded (f ' S) and g absolutely_integrable_on S

```

shows $(\lambda x. f x * g x)$ *absolutely_integrable_on* S
using *absolutely_integrable_bounded_measurable_product bilinear_times assms*
by *blast*

lemma *borel_measurable_AE*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
assumes $f \in \text{borel_measurable lebesgue}$ **and** $ae: AE\ x\ \text{in lebesgue. } f\ x = g\ x$
shows $g \in \text{borel_measurable lebesgue}$
proof –
obtain N **where** $N: N \in \text{null_sets lebesgue} \wedge x. x \notin N \implies f\ x = g\ x$
using *ae unfolding completion.AE_iff_null_sets* **by** *auto*
have f *measurable_on UNIV*
by (*simp add: assms lebesgue_measurable_imp_measurable_on*)
then have g *measurable_on UNIV*
by (*metis Diff_iff N measurable_on_spike negligible_iff_null_sets*)
then show *?thesis*
using *measurable_on_imp_borel_measurable_lebesgue_UNIV* **by** *blast*
qed

lemma *has_bochner_integral_combine*:

fixes $f :: \text{real} \Rightarrow 'a::euclidean_space$
assumes $a \leq c\ c \leq b$
and $ac: \text{has_bochner_integral lebesgue_on } \{a..c\} f\ i$
and $cb: \text{has_bochner_integral lebesgue_on } \{c..b\} f\ j$
shows $\text{has_bochner_integral lebesgue_on } \{a..b\} f\ (i + j)$
proof –
have $i: \text{has_bochner_integral lebesgue } (\lambda x. \text{indicator } \{a..c\}\ x *_{\mathbb{R}} f\ x)\ i$
and $j: \text{has_bochner_integral lebesgue } (\lambda x. \text{indicator } \{c..b\}\ x *_{\mathbb{R}} f\ x)\ j$
using *assms* **by** (*auto simp: has_bochner_integral_restrict_space*)
have $AE: AE\ x\ \text{in lebesgue. } \text{indicat_real } \{a..c\}\ x *_{\mathbb{R}} f\ x + \text{indicat_real } \{c..b\}\ x *_{\mathbb{R}} f\ x = \text{indicat_real } \{a..b\}\ x *_{\mathbb{R}} f\ x$
proof (*rule AE_I'*)
have $eq: \text{indicat_real } \{a..c\}\ x *_{\mathbb{R}} f\ x + \text{indicat_real } \{c..b\}\ x *_{\mathbb{R}} f\ x = \text{indicat_real } \{a..b\}\ x *_{\mathbb{R}} f\ x$ **if** $x \neq c$ **for** x
using *assms that* **by** (*auto simp: indicator_def*)
then show $\{x \in \text{space lebesgue. } \text{indicat_real } \{a..c\}\ x *_{\mathbb{R}} f\ x + \text{indicat_real } \{c..b\}\ x *_{\mathbb{R}} f\ x \neq \text{indicat_real } \{a..b\}\ x *_{\mathbb{R}} f\ x\} \subseteq \{c\}$
by *auto*
qed *auto*
have $\text{has_bochner_integral lebesgue } (\lambda x. \text{indicator } \{a..b\}\ x *_{\mathbb{R}} f\ x)\ (i + j)$
proof (*rule has_bochner_integralI_AE [OF has_bochner_integral_add [OF i j] _ AE]*)
have $eq: \text{indicat_real } \{a..c\}\ x *_{\mathbb{R}} f\ x + \text{indicat_real } \{c..b\}\ x *_{\mathbb{R}} f\ x = \text{indicat_real } \{a..b\}\ x *_{\mathbb{R}} f\ x$ **if** $x \neq c$ **for** x
using *assms that* **by** (*auto simp: indicator_def*)
show $(\lambda x. \text{indicat_real } \{a..b\}\ x *_{\mathbb{R}} f\ x) \in \text{borel_measurable lebesgue}$
proof (*rule borel_measurable_AE [OF borel_measurable_add AE]*)
show $(\lambda x. \text{indicator } \{a..c\}\ x *_{\mathbb{R}} f\ x) \in \text{borel_measurable lebesgue}$

```

      ( $\lambda x. \text{indicator } \{c..b\} x *_{\mathbb{R}} f x$ )  $\in$  borel_measurable lebesgue
    using i j by auto
  qed
  qed
  then show ?thesis
    by (simp add: has_bochner_integral_restrict_space)
  qed

lemma integrable_combine:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  assumes integrable (lebesgue_on {a..c}) f integrable (lebesgue_on {c..b}) f
    and a  $\leq$  c c  $\leq$  b
  shows integrable (lebesgue_on {a..b}) f
  using assms has_bochner_integral_combine has_bochner_integral_iff by blast

lemma integral_combine:
  fixes f :: real  $\Rightarrow$  'a::euclidean_space
  assumes f: integrable (lebesgue_on {a..b}) f and a  $\leq$  c c  $\leq$  b
  shows  $\text{integral}^L$  (lebesgue_on {a..b}) f =  $\text{integral}^L$  (lebesgue_on {a..c}) f +
 $\text{integral}^L$  (lebesgue_on {c..b}) f
  proof -
    have i: has_bochner_integral (lebesgue_on {a..c}) f ( $\text{integral}^L$  (lebesgue_on {a..c})
    f)
      using integrable_subinterval  $\langle c \leq b \rangle$  f has_bochner_integral_iff by fastforce
    have j: has_bochner_integral (lebesgue_on {c..b}) f ( $\text{integral}^L$  (lebesgue_on {c..b})
    f)
      using integrable_subinterval  $\langle a \leq c \rangle$  f has_bochner_integral_iff by fastforce
    show ?thesis
      by (meson  $\langle a \leq c \rangle \langle c \leq b \rangle$  has_bochner_integral_combine has_bochner_integral_iff
    i j)
  qed

lemma has_bochner_integral_null [intro]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes N  $\in$  null_sets lebesgue
  shows has_bochner_integral (lebesgue_on N) f 0
  unfolding has_bochner_integral_iff — strange that the proof's so long
  proof
    show integrable (lebesgue_on N) f
    proof (subst integrable_restrict_space)
      show N  $\cap$  space lebesgue  $\in$  sets lebesgue
        using assms by force
      show integrable lebesgue ( $\lambda x. \text{indicat\_real } N x *_{\mathbb{R}} f x$ )
    proof (rule integrable_cong_AE_imp)
      show integrable lebesgue ( $\lambda x. 0$ )
        by simp
      show *: AE x in lebesgue. 0 =  $\text{indicat\_real } N x *_{\mathbb{R}} f x$ 
        using assms
        by (simp add: indicator_def completion.null_sets_iff_AE eventually_mono)
    end
  end

```



```

    show ( $\lambda x. \text{indicat\_real } N \ x \ *_{\mathbb{R}} \ f \ x$ )  $\in$  borel_measurable lebesgue
    by (auto intro: borel_measurable_AE [OF _ *])
  qed
qed
show integralL (lebesgue_on N) f = 0
proof (rule integral_eq_zero_AE)
  show AE x in lebesgue_on N. f x = 0
  by (rule AE_I' [where N=N]) (auto simp: assms null_setsD2 null_sets_restrict_space)
qed
qed
end

lemma has_bochner_integral_null_eq[simp]:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes N  $\in$  null_sets lebesgue
  shows has_bochner_integral (lebesgue_on N) f i  $\longleftrightarrow$  i = 0
  using assms has_bochner_integral_eq by blast
end

```

9.17 Embedding Measure Spaces with a Function

```

theory Embed_Measure
imports Binary_Product_Measure
begin

```

Given a measure space on some carrier set Ω and a function f , we can define a push-forward measure on the carrier set $f(\Omega)$ whose σ -algebra is the one generated by mapping f over the original sigma algebra.

This is useful e.g. when f is injective, i.e. it is some kind of “tagging” function. For instance, suppose we have some algebraic datatype of values with various constructors, including a constructor *RealVal* for real numbers. Then *embed_measure* allows us to lift a measure on real numbers to the appropriate subset of that algebraic datatype.

```

definition embed_measure :: 'a measure  $\Rightarrow$  ('a  $\Rightarrow$  'b)  $\Rightarrow$  'b measure where
  embed_measure M f = measure_of (f ' space M) {f ' A | A. A  $\in$  sets M}
  ( $\lambda A. \text{emeasure } M \ (f \text{ -' } A \cap \text{space } M)$ )

```

```

lemma space_embed_measure: space (embed_measure M f) = f ' space M
unfolding embed_measure_def
by (subst space_measure_of) (auto dest: sets_sets_into_space)

```

```

lemma sets_embed_measure':
  assumes inj: inj_on f (space M)
  shows sets (embed_measure M f) = {f ' A | A. A  $\in$  sets M}
  unfolding embed_measure_def
proof (intro sigma_algebra_sets_measure_of_eq_sigma_algebra_iff2 [THEN iffD2]
  conjI allI ballI impI)
  fix s assume s  $\in$  {f ' A | A. A  $\in$  sets M}

```

then obtain s' where $s'_props: s = f' s' s' \in sets M$ by *auto*
 hence $f' space M - s = f' (space M - s')$ using *inj*
 by (*auto dest: inj_onD sets.sets_into_space*)
 also have $\dots \in \{f' A \mid A. A \in sets M\}$ using s'_props by *auto*
 finally show $f' space M - s \in \{f' A \mid A. A \in sets M\}$.

next

fix $A :: nat \Rightarrow _$ assume $range A \subseteq \{f' A \mid A. A \in sets M\}$
 then obtain A' where $A': \bigwedge i. A i = f' A' i \wedge i. A' i \in sets M$
 by (*auto simp: subset_eq choice_iff*)
 then have $(\bigcup x. f' A' x) = f' (\bigcup x. A' x)$ by *blast*
 with A' show $(\bigcup i. A i) \in \{f' A \mid A. A \in sets M\}$
 by *simp blast*

qed (*auto dest: sets.sets_into_space*)

lemma *the_inv_into_vimage*:

$inj_on f X \implies A \subseteq X \implies the_inv_into X f -' A \cap (f'X) = f' A$
 by (*auto simp: the_inv_into_f_f*)

lemma *sets_embed_eq_vimage_algebra*:

assumes *inj_on f (space M)*

shows $sets (embed_measure M f) = sets (vimage_algebra (f' space M) (the_inv_into (space M) f) M)$

by (*auto simp: sets_embed_measure'[OF assms] Pi_iff the_inv_into_f_f assms sets_vimage_algebra2 Setcompr_eq_image dest: sets.sets_into_space intro!: image_cong the_inv_into_vimage[symmetric]*)

lemma *sets_embed_measure*:

assumes *inj: inj f*

shows $sets (embed_measure M f) = \{f' A \mid A. A \in sets M\}$

using *assms* by (*subst sets_embed_measure'*) (*auto intro!: inj_onI dest: injD*)

lemma *in_sets_embed_measure*: $A \in sets M \implies f' A \in sets (embed_measure M f)$

unfolding *embed_measure_def*

by (*intro in_measure_of*) (*auto dest: sets.sets_into_space*)

lemma *measurable_embed_measure1*:

assumes $g: (\lambda x. g (f x)) \in measurable M N$

shows $g \in measurable (embed_measure M f) N$

unfolding *measurable_def*

proof *safe*

fix A assume $A \in sets N$

with g have $(\lambda x. g (f x)) -' A \cap space M \in sets M$

by (*rule measurable_sets*)

then have $f' ((\lambda x. g (f x)) -' A \cap space M) \in sets (embed_measure M f)$

by (*rule in_sets_embed_measure*)

also have $f' ((\lambda x. g (f x)) -' A \cap space M) = g -' A \cap space (embed_measure M f)$

```

  by (auto simp: space_embed_measure)
  finally show  $g - ' A \cap \text{space } (\text{embed\_measure } M f) \in \text{sets } (\text{embed\_measure } M f)$  .
qed (insert measurable_space[OF assms], auto simp: space_embed_measure)

```

```

lemma measurable_embed_measure2':
  assumes inj_on f (space M)
  shows  $f \in \text{measurable } M (\text{embed\_measure } M f)$ 
proof -
  {
    fix A assume A:  $A \in \text{sets } M$ 
    also from A have  $A = A \cap \text{space } M$  by auto
    also have  $\dots = f - ' f - ' A \cap \text{space } M$  using A assms
    by (auto dest: inj_onD sets.sets_into_space)
    finally have  $f - ' f - ' A \cap \text{space } M \in \text{sets } M$  .
  }
  thus ?thesis using assms unfolding embed_measure_def
  by (intro measurable_measure_of) (auto dest: sets.sets_into_space)
qed

```

```

lemma measurable_embed_measure2:
  assumes [simp]: inj f shows  $f \in \text{measurable } M (\text{embed\_measure } M f)$ 
  by (auto simp: inj_vimage_image_eq embed_measure_def
    intro!: measurable_measure_of dest: sets.sets_into_space)

```

```

lemma embed_measure_eq_distr':
  assumes inj_on f (space M)
  shows  $\text{embed\_measure } M f = \text{distr } M (\text{embed\_measure } M f) f$ 
proof -
  have  $\text{distr } M (\text{embed\_measure } M f) f =$ 
     $\text{measure\_of } (f - ' \text{space } M) \{f - ' A \mid A. A \in \text{sets } M\}$ 
     $(\lambda A. \text{emeasure } M (f - ' A \cap \text{space } M))$  unfolding distr_def
  by (simp add: space_embed_measure sets_embed_measure'[OF assms])
  also have  $\dots = \text{embed\_measure } M f$  unfolding embed_measure_def ..
  finally show ?thesis ..
qed

```

```

lemma embed_measure_eq_distr:
   $\text{inj } f \implies \text{embed\_measure } M f = \text{distr } M (\text{embed\_measure } M f) f$ 
  by (rule embed_measure_eq_distr') (auto intro!: inj_onI dest: injD)

```

```

lemma nn_integral_embed_measure':
   $\text{inj\_on } f (\text{space } M) \implies g \in \text{borel\_measurable } (\text{embed\_measure } M f) \implies$ 
   $\text{nn\_integral } (\text{embed\_measure } M f) g = \text{nn\_integral } M (\lambda x. g (f x))$ 
  apply (subst embed_measure_eq_distr', simp)
  apply (subst nn_integral_distr)
  apply (simp_all add: measurable_embed_measure2')
  done

```

lemma *nn_integral_embed_measure*:

inj f $\implies g \in \text{borel_measurable } (\text{embed_measure } M f) \implies$
 $\text{nn_integral } (\text{embed_measure } M f) g = \text{nn_integral } M (\lambda x. g (f x))$
by (*erule nn_integral_embed_measure'*[*OF subset_inj_on*]) *simp*

lemma *emeasure_embed_measure'*:

assumes *inj_on f* (*space M*) $A \in \text{sets } (\text{embed_measure } M f)$
shows $\text{emeasure } (\text{embed_measure } M f) A = \text{emeasure } M (f \text{ -' } A \cap \text{space } M)$
by (*subst embed_measure_eq_distr'*[*OF assms(1)*])
(*simp add: emeasure_distr'*[*OF measurable_embed_measure2'*[*OF assms(1)*]]
assms(2))]

lemma *emeasure_embed_measure*:

assumes *inj f* $A \in \text{sets } (\text{embed_measure } M f)$
shows $\text{emeasure } (\text{embed_measure } M f) A = \text{emeasure } M (f \text{ -' } A \cap \text{space } M)$
using *assms* **by** (*intro emeasure_embed_measure'*) (*auto intro!: inj_onI dest: injD*)

lemma *embed_measure_comp*:

assumes [*simp*]: *inj f inj g*
shows $\text{embed_measure } (\text{embed_measure } M f) g = \text{embed_measure } M (g \circ f)$
proof –
have [*simp*]: *inj* ($\lambda x. g (f x)$) **by** (*subst o_def[symmetric]*) (*auto intro: inj_compose*)
note *measurable_embed_measure2[measurable]*
have $\text{embed_measure } (\text{embed_measure } M f) g =$
 $\text{distr } M (\text{embed_measure } (\text{embed_measure } M f) g) (g \circ f)$
by (*subst (1 2) embed_measure_eq_distr*)
(*simp_all add: distr_distr sets_embed_measure cong: distr_cong*)
also have $\dots = \text{embed_measure } M (g \circ f)$
by (*subst (3) embed_measure_eq_distr, simp add: o_def, rule distr_cong*)
(*auto simp: sets_embed_measure o_def image_image[symmetric]*)
(*intro: inj_compose cong: distr_cong*)
finally show *?thesis* .
qed

lemma *sigma_finite_embed_measure*:

assumes *sigma_finite_measure M* **and** *inj: inj f*
shows $\text{sigma_finite_measure } (\text{embed_measure } M f)$
proof –
from *assms(1)* **interpret** *sigma_finite_measure M* .
from *sigma_finite_countable* **obtain** *A* **where**
 $A_props: \text{countable } A \ A \subseteq \text{sets } M \ \cup A = \text{space } M \ \wedge X. X \in A \implies \text{emeasure}$
 $M X \neq \infty$ **by** *blast*
from *A_props* **have** *countable* ((\cdot) *f*'*A*) **by** *auto*
moreover
from *inj* **and** *A_props* **have** ((\cdot) *f*'*A*) $\subseteq \text{sets } (\text{embed_measure } M f)$
by (*auto simp: sets_embed_measure*)
moreover
from *A_props* **and** *inj* **have** $\cup ((\cdot) f'A) = \text{space } (\text{embed_measure } M f)$

```

    by (auto simp: space_embed_measure intro!: imageI)
  moreover
  from A_props and inj have  $\forall a \in (\cdot) f \text{ ' } A. \text{emeasure (embed\_measure } M f) a \neq$ 
 $\infty$ 
  by (intro ballI, subst emeasure_embed_measure)
  (auto simp: inj_vimage_image_eq intro: in_sets_embed_measure)
  ultimately show ?thesis by - (standard, blast)
qed

```

lemma *embed_measure_count_space'*:

```

  inj_on f A  $\implies$  embed_measure (count_space A) f = count_space (f'A)
  apply (subst distr_bij_count_space[of f A f'A, symmetric])
  apply (simp add: inj_on_def bij_betw_def)
  apply (subst embed_measure_eq_distr')
  apply simp
  apply (auto 4 3 intro!: measure_eqI imageI simp add: sets_embed_measure' subset_image_iff)
  apply (subst (1 2) emeasure_distr)
  apply (auto simp: space_embed_measure sets_embed_measure')
  done

```

lemma *embed_measure_count_space*:

```

  inj f  $\implies$  embed_measure (count_space A) f = count_space (f'A)
  by (rule embed_measure_count_space')(erule subset_inj_on, simp)

```

lemma *sets_embed_measure_alt*:

```

  inj f  $\implies$  sets (embed_measure M f) = (( $\cdot$ ) f) ' sets M
  by (auto simp: sets_embed_measure)

```

lemma *emeasure_embed_measure_image'*:

```

  assumes inj_on f (space M) X  $\in$  sets M
  shows emeasure (embed_measure M f) (f'X) = emeasure M X
proof -
  from assms have emeasure (embed_measure M f) (f'X) = emeasure M (f -' f
  ' X  $\cap$  space M)
  by (subst emeasure_embed_measure') (auto simp: sets_embed_measure')
  also from assms have f -' f ' X  $\cap$  space M = X by (auto dest: inj_onD
  sets.sets_into_space)
  finally show ?thesis .
qed

```

lemma *emeasure_embed_measure_image*:

```

  inj f  $\implies$  X  $\in$  sets M  $\implies$  emeasure (embed_measure M f) (f'X) = emeasure
  M X
  by (simp_all add: emeasure_embed_measure in_sets_embed_measure inj_vimage_image_eq)

```

lemma *embed_measure_eq_iff*:

```

  assumes inj f
  shows embed_measure A f = embed_measure B f  $\iff$  A = B (is ?M = ?N)

```

\longleftrightarrow $_$)

proof

from *assms* **have** $I: \text{inj } ((\cdot) f)$ **by** (*auto intro: injI dest: injD*)

assume *asm*: $?M = ?N$

hence $\text{sets } (\text{embed_measure } A f) = \text{sets } (\text{embed_measure } B f)$ **by** *simp*

with *assms* **have** $\text{sets } A = \text{sets } B$ **by** (*simp only: I inj_image_eq_iff sets_embed_measure_alt*)

moreover {

fix X **assume** $X \in \text{sets } A$

from *asm* **have** $\text{emeasure } ?M (f'X) = \text{emeasure } ?N (f'X)$ **by** *simp*

with $\langle X \in \text{sets } A \rangle$ **and** $\langle \text{sets } A = \text{sets } B \rangle$ **and** *assms*

have $\text{emeasure } A X = \text{emeasure } B X$ **by** (*simp add: emeasure_embed_measure_image*)

}

ultimately show $A = B$ **by** (*rule measure_eqI*)

qed *simp*

lemma *the_inv_into_in_Pi*: $\text{inj_on } f A \implies \text{the_inv_into } A f \in f' A \rightarrow A$

by (*auto simp: the_inv_into_f_f*)

lemma *map_prod_image*: $\text{map_prod } f g' (A \times B) = (f'A) \times (g'B)$

using *map_prod_surj_on[OF refl refl]* .

lemma *map_prod_vimage*: $\text{map_prod } f g -' (A \times B) = (f-'A) \times (g-'B)$

by *auto*

lemma *embed_measure_prod*:

assumes $f: \text{inj } f$ **and** $g: \text{inj } g$ **and** [*simp*]: *sigma_finite_measure* M *sigma_finite_measure* N

shows $\text{embed_measure } M f \otimes_M \text{embed_measure } N g = \text{embed_measure } (M \otimes_M N) (\lambda(x, y). (f x, g y))$

(**is** $?L = _$)

unfolding *map_prod_def[symmetric]*

proof (*rule pair_measure_eqI*)

have $fg[\text{simp}]$: $\bigwedge A. \text{inj_on } (\text{map_prod } f g) A \bigwedge A. \text{inj_on } f A \bigwedge A. \text{inj_on } g A$

using $f g$ **by** (*auto simp: inj_on_def*)

note *complete_lattice_class.SUP_insert[simp del]* *ccSUP_insert[simp del]*

ccSUP_insert[simp del]

show $\text{sets } ?L = \text{sets } (\text{embed_measure } (M \otimes_M N) (\text{map_prod } f g))$

unfolding *map_prod_def[symmetric]*

apply (*simp add: sets_pair_eq_setsfst_snd sets_embed_eq_vimage_algebra cong: vimage_algebra_cong*)

apply (*subst sets_vimage_SUP_eq[where Y=space (M \otimes_M N)]*)

apply (*simp_all add: space_pair_measure[symmetric]*)

apply (*auto simp add: the_inv_into_f_f*

simp del: map_prod_simp

del: prod_fun_imageE) []

apply *auto* []

apply (*subst (1 2 3 4) vimage_algebra_vimage_algebra_eq*)

apply (*simp_all add: the_inv_into_in_Pi Pi_iff[of snd] Pi_iff[of fst] space_pair_measure*)

```

apply (simp_all add: Pi_iff[of snd] Pi_iff[of fst] the_inv_into_in_Pi vimage_algebra_vimage_algebra_eq
  space_pair_measure[symmetric] map_prod_image[symmetric])
apply (intro arg_cong[where f=sets] arg_cong[where f=Sup] arg_cong2[where f=insert] vimage_algebra_cong)
apply (auto simp: map_prod_image the_inv_into_f_f
  simp del: map_prod simp del: prod_fun_imageE)
apply (simp_all add: the_inv_into_f_f space_pair_measure)
done

note measurable_embed_measure2[measurable]
fix A B assume AB: A ∈ sets (embed_measure M f) B ∈ sets (embed_measure N g)
moreover have f -' A × g -' B ∩ space (M ⊗M N) = (f -' A ∩ space M) × (g -' B ∩ space N)
by (auto simp: space_pair_measure)
ultimately show emeasure (embed_measure M f) A * emeasure (embed_measure N g) B =
  emeasure (embed_measure (M ⊗M N) (map_prod f g)) (A × B)
by (simp add: map_prod_vimage_sets[symmetric] emeasure_embed_measure
  sigma_finite_measure.emeasure_pair_measure_Times)
qed (insert assms, simp_all add: sigma_finite_embed_measure)

```

lemma mono_embed_measure:

```

  space M = space M' ⇒ sets M ⊆ sets M' ⇒ sets (embed_measure M f) ⊆
sets (embed_measure M' f)
unfolding embed_measure_def
apply (subst (1 2) sets_measure_of)
apply (blast dest: sets.sets_into_space)
apply (blast dest: sets.sets_into_space)
apply simp
apply (intro sigma_sets_mono')
apply safe
apply (simp add: subset_eq)
apply metis
done

```

lemma density_embed_measure:

```

assumes inj: inj f and Mg[measurable]: g ∈ borel_measurable (embed_measure M f)
shows density (embed_measure M f) g = embed_measure (density M (g ∘ f)) f
(is ?M1 = ?M2)
proof (rule measure_eqI)
fix X assume X: X ∈ sets ?M1
from inj have Mf[measurable]: f ∈ measurable M (embed_measure M f)
by (rule measurable_embed_measure2)
from Mg and X have emeasure ?M1 X = ∫+ x. g x * indicator X x ∂embed_measure M f
by (subst emeasure_density) simp_all

```

also from X have $\dots = \int^+ x. g (f x) * \text{indicator } X (f x) \partial M$
by (*subst embed_measure_eq_distr*[*OF inj*], *subst nn_integral_distr*) *auto*
also have $\dots = \int^+ x. g (f x) * \text{indicator } (f -' X \cap \text{space } M) x \partial M$
by (*intro nn_integral_cong*) (*auto split: split_indicator*)
also from X have $\dots = \text{emeasure } (\text{density } M (g \circ f)) (f -' X \cap \text{space } M)$
by (*subst emeasure_density*) (*simp_all add: measurable_comp*[*OF Mf Mg*]
measurable_sets[*OF Mf*])
also from X and inj have $\dots = \text{emeasure } ?M2 X$
by (*subst emeasure_embed_measure*) (*simp_all add: sets_embed_measure*)
finally show $\text{emeasure } ?M1 X = \text{emeasure } ?M2 X$.
qed (*simp_all add: sets_embed_measure inj*)

lemma *density_embed_measure'*:

assumes *inj: inj f* **and** *inv: $\bigwedge x. f' (f x) = x$* **and** *Mg*[*measurable*]: $g \in \text{borel_measurable } M$

shows $\text{density } (\text{embed_measure } M f) (g \circ f') = \text{embed_measure } (\text{density } M g) f$
proof –

have $\text{density } (\text{embed_measure } M f) (g \circ f') = \text{embed_measure } (\text{density } M (g \circ f' \circ f)) f$

by (*rule density_embed_measure*[*OF inj*])
(*rule measurable_comp*, *rule measurable_embed_measure1*, *subst measurable_cong*,
rule inv, *rule measurable_ident_sets*, *simp*, *rule Mg*)

also have $\text{density } M (g \circ f' \circ f) = \text{density } M g$

by (*intro density_cong*) (*subst measurable_cong*, *simp add: o_def inv*, *simp_all add: Mg inv*)

finally show *?thesis* .

qed

lemma *inj_on_image_subset_iff*:

assumes *inj_on f* $C A \subseteq C$ $B \subseteq C$

shows $f -' A \subseteq f -' B \iff A \subseteq B$

proof (*intro iffI subsetI*)

fix x **assume** $A: f -' A \subseteq f -' B$ **and** $B: x \in A$

from B **have** $f x \in f -' A$ **by** *blast*

with A **have** $f x \in f -' B$ **by** *blast*

then obtain y **where** $f x = f y$ **and** $y \in B$ **by** *blast*

with *assms* **and** B **have** $x = y$ **by** (*auto dest: inj_onD*)

with $\langle y \in B \rangle$ **show** $x \in B$ **by** *simp*

qed *auto*

lemma *AE_embed_measure'*:

assumes *inj: inj_on f* (*space M*)

shows $(\text{AE } x \text{ in } \text{embed_measure } M f. P x) \iff (\text{AE } x \text{ in } M. P (f x))$

proof

let $?M = \text{embed_measure } M f$

assume $\text{AE } x \text{ in } ?M. P x$

then obtain A **where** $A _ \text{props}: A \in \text{sets } ?M$ $\text{emeasure } ?M A = 0$ $\{x \in \text{space}$


```

?M.  $\neg P x \} \subseteq A$ 
  by (force elim: AE_E)
  then obtain A' where A'_props:  $A = f \text{ ` } A' A' \in \text{sets } M$  by (auto simp:
sets_embed_measure' inj)
  moreover have B:  $\{x \in \text{space } ?M. \neg P x\} = f \text{ ` } \{x \in \text{space } M. \neg P (f x)\}$ 
    by (auto simp: inj space_embed_measure)
  from A'_props(3) have  $\{x \in \text{space } M. \neg P (f x)\} \subseteq A'$ 
    by (subst (asm) B, subst (asm) A'_props, subst (asm) inj_on_image_subset_iff[OF
inj])
      (insert A'_props, auto dest: sets.sets_into_space)
  moreover from A'_props A'_props have  $\text{emeasure } M A' = 0$ 
    by (simp add: emeasure_embed_measure_image' inj)
  ultimately show  $AE x \text{ in } M. P (f x)$  by (intro AE_I)
next
let ?M = embed_measure M f
assume AE x in M. P (f x)
then obtain A where A_props:  $A \in \text{sets } M \text{ emeasure } M A = 0 \{x \in \text{space } M.
\neg P (f x)\} \subseteq A$ 
  by (force elim: AE_E)
  hence  $f'A \in \text{sets } ?M \text{ emeasure } ?M (f'A) = 0 \{x \in \text{space } ?M. \neg P x\} \subseteq f'A$ 
    by (auto simp: space_embed_measure emeasure_embed_measure_image' sets_embed_measure'
inj)
  thus  $AE x \text{ in } ?M. P x$  by (intro AE_I)
qed

```

lemma AE_embed_measure:

```

  assumes inj: inj f
  shows  $(AE x \text{ in } \text{embed\_measure } M f. P x) \longleftrightarrow (AE x \text{ in } M. P (f x))$ 
  using assms by (intro AE_embed_measure') (auto intro!: inj_onI dest: injD)

```

lemma nn_integral_monotone_convergence_SUP_countable:

```

  fixes f :: 'a  $\Rightarrow$  'b  $\Rightarrow$  ennreal
  assumes nonempty:  $Y \neq \{\}$ 
  and chain: Complete_Partial_Order.chain ( $\leq$ ) (f ` Y)
  and countable: countable B
  shows  $(\int^+ x. (\text{SUP } i \in Y. f i x) \partial \text{count\_space } B) = (\text{SUP } i \in Y. (\int^+ x. f i x
\partial \text{count\_space } B))$ 
    (is ?lhs = ?rhs)

```

proof –

```

  let ?f =  $(\lambda i x. f i (\text{from\_nat\_into } B x) * \text{indicator } (\text{to\_nat\_on } B \text{ ` } B) x)$ 
  have ?lhs =  $\int^+ x. (\text{SUP } i \in Y. f i (\text{from\_nat\_into } B (\text{to\_nat\_on } B x)))
\partial \text{count\_space } B$ 
    by (rule nn_integral_cong) (simp add: countable)
  also have  $\dots = \int^+ x. (\text{SUP } i \in Y. f i (\text{from\_nat\_into } B x)) \partial \text{count\_space}
(\text{to\_nat\_on } B \text{ ` } B)$ 
    by (simp add: embed_measure_count_space'[symmetric] inj_on_to_nat_on
countable nn_integral_embed_measure' measurable_embed_measure1)
  also have  $\dots = \int^+ x. (\text{SUP } i \in Y. ?f i x) \partial \text{count\_space } UNIV$ 
    by (simp add: nn_integral_count_space_indicator ennreal_indicator[symmetric])

```

```

SUP_mult_right_ennreal nonempty)
  also have ... = (SUP i ∈ Y. ∫+ x. ?f i x ∂count_space UNIV)
  proof(rule nn_integral_monotone_convergence_SUP_nat)
    show Complete_Partial_Order.chain (≤) (?f ' Y)
    by(rule chain_imageI[OF chain, unfolded image_image])(auto intro!: le_funI
split: split_indicator dest: le_funD)
  qed fact
  also have ... = (SUP i ∈ Y. ∫+ x. f i (from_nat_into B x) ∂count_space
(to_nat_on B ' B))
  by(simp add: nn_integral_count_space_indicator)
  also have ... = (SUP i ∈ Y. ∫+ x. f i (from_nat_into B (to_nat_on B x))
∂count_space B)
  by(simp add: embed_measure_count_space'[symmetric] inj_on_to_nat_on
countable nn_integral_embed_measure' measurable_embed_measure1)
  also have ... = ?rhs
  by(intro arg_cong2[where f = λA f. Sup (f ' A)] ext nn_integral_cong_AE)(simp_all
add: AE_count_space countable)
  finally show ?thesis .
qed
end

```

9.18 Brouwer's Fixed Point Theorem

```

theory Brouwer_Fixpoint
  imports Homeomorphism Derivative
begin

```

9.18.1 Retractions

```

lemma retract_of_contractible:
  assumes contractible T S retract_of T
  shows contractible S
  using assms
  apply (clarsimp simp add: retract_of_def contractible_def retraction_def homo-
topic_with_image_subset_iff_funcset)
  apply (rule_tac x=r a in exI)
  apply (rule_tac x=r ∘ h in exI)
  apply (intro conjI continuous_intros continuous_on_compose)
  apply (erule continuous_on_subset | force)+
done

```

```

lemma retract_of_path_connected:
  [[path_connected T; S retract_of T]] ⇒ path_connected S
  by (metis path_connected_continuous_image retract_of_def retraction)

```

```

lemma retract_of_simply_connected:
  assumes T: simply_connected T and S retract_of T
  shows simply_connected S

```

```

proof -
  obtain r where r: retraction T S r
    using assms by (metis retract_of_def)
  have S  $\subseteq$  T
    by (meson  $\langle$ retraction T S r $\rangle$  retraction)
  then have  $(\lambda a. a) \in S \rightarrow T$ 
    by blast
  then show ?thesis
    using simply_connected_retraction_gen [OF T]
    by (metis (no_types) r retraction retraction_refl)
qed

```

lemma retract_of_homotopically_trivial:

```

assumes ts: T retract_of S
  and hom:  $\bigwedge f g. \llbracket$ continuous_on U f;  $f \in U \rightarrow S$ ;
           continuous_on U g;  $g \in U \rightarrow S$  $\rrbracket$ 
            $\implies$  homotopic_with_canon  $(\lambda x. True)$  U S f g
  and continuous_on U f  $f \in U \rightarrow T$ 
  and continuous_on U g  $g \in U \rightarrow T$ 
shows homotopic_with_canon  $(\lambda x. True)$  U T f g

```

proof -

```

obtain r where r  $\in S \rightarrow S$  continuous_on S r  $\forall x \in S. r (r x) = r x$  T = r ' S
  using ts by (auto simp: retract_of_def retraction)
then obtain k where Retracts S r T k
  unfolding Retracts_def using continuous_on_id by blast
then show ?thesis
  by (rule Retracts.homotopically_trivial_retraction_gen) (use assms hom in

```

force)+

qed

lemma retract_of_homotopically_trivial_null:

```

assumes ts: T retract_of S
  and hom:  $\bigwedge f. \llbracket$ continuous_on U f;  $f \in U \rightarrow S$  $\rrbracket$ 
            $\implies \exists c. homotopic_with_canon (\lambda x. True) U S f (\lambda x. c)$ 
  and continuous_on U f  $f \in U \rightarrow T$ 
obtains c where homotopic_with_canon  $(\lambda x. True)$  U T f  $(\lambda x. c)$ 

```

proof -

```

obtain r where r  $\in S \rightarrow S$  continuous_on S r  $\forall x \in S. r (r x) = r x$  T = r ' S
  using ts by (auto simp: retract_of_def retraction)

```

```

then obtain k where Retracts S r T k
  unfolding Retracts_def by fastforce

```

then show ?thesis

proof (rule Retracts.homotopically_trivial_retraction_null_gen)

```

  show  $\bigwedge f. \llbracket$ continuous_on U f;  $f \in U \rightarrow S$  $\rrbracket$ 
     $\implies \exists c. homotopic_with_canon (\lambda a. True) U S f (\lambda x. c)$ 

```

using hom by blast

qed (use assms that in auto)

qed

lemma *retraction_openin_vimage_iff*:
openin (top_of_set S) (S ∩ r -‘ U) ↔ openin (top_of_set T) U
if *retraction S T r* **and** $U \subseteq T$
by (*simp add: retraction_openin_vimage_iff that*)

lemma *retract_of_locally_compact*:
fixes $S :: 'a :: \{\text{heine_borel, real_normed_vector}\}$ *set*
shows $\llbracket \text{locally_compact } S; T \text{ retract_of } S \rrbracket \implies \text{locally_compact } T$
by (*metis locally_compact_closedin closedin_retract*)

lemma *homotopic_into_retract*:
 $\llbracket f \in S \rightarrow T; g \in S \rightarrow T; T \text{ retract_of } U; \text{homotopic_with_canon } (\lambda x. \text{True}) S U f g \rrbracket$
 $\implies \text{homotopic_with_canon } (\lambda x. \text{True}) S T f g$
apply (*subst (asm) homotopic_with_def*)
apply (*simp add: homotopic_with_retract_of_def retraction_def Pi_iff, clarify*)
apply (*rule_tac x=r ∘ h in exI*)
by (*smt (verit, ccfv_SIG) comp_def continuous_on_compose continuous_on_subset image_subset_iff*)

lemma *retract_of_locally_connected*:
assumes *locally_connected T S retract_of T*
shows *locally_connected S*
using *assms*
by (*metis retraction_openin_vimage_iff idempotent_imp_retraction locally_connected_quotient_image retract_ofE*)

lemma *retract_of_locally_path_connected*:
assumes *locally_path_connected T S retract_of T*
shows *locally_path_connected S*
using *assms*
by (*metis retraction_openin_vimage_iff idempotent_imp_retraction locally_path_connected_quotient_image retract_ofE*)

A few simple lemmas about deformation retracts

lemma *deformation_retract_imp_homotopy_eqv*:
fixes $S :: 'a :: \text{euclidean_space}$ *set*
assumes *homotopic_with_canon* $(\lambda x. \text{True}) S S \text{ id } r$ **and** $r: \text{retraction } S T r$
shows $S \text{ homotopy_eqv } T$
proof –
have *homotopic_with_canon* $(\lambda x. \text{True}) S S (\text{id} \circ r) \text{id}$
by (*simp add: assms(1) homotopic_with_symD*)
moreover have *homotopic_with_canon* $(\lambda x. \text{True}) T T (r \circ \text{id}) \text{id}$
using r **unfolding** *retraction_def*
by (*metis eq_id_iff homotopic_with_id2 topspace_euclidean_subtopology*)
ultimately
show *?thesis*
unfolding *homotopy_equivalent_space_def*
by (*smt (verit, del_insts) continuous_map_id continuous_map_subtopology_eu*)

id_def r retraction retraction_comp subset_refl)

qed

lemma *deformation_retract*:

fixes $S :: 'a::euclidean_space\ set$

shows $(\exists r. homotopic_with_canon (\lambda x. True) S S id r \wedge retraction S T r)$

\longleftrightarrow

$T retract_of S \wedge (\exists f. homotopic_with_canon (\lambda x. True) S S id f \wedge f \in S \rightarrow T)$

(is *?lhs = ?rhs*)

proof

assume *?lhs*

then show *?rhs*

by (*auto simp: retract_of_def retraction_def*)

next

assume *R: ?rhs*

have $\bigwedge r f. \llbracket T \subseteq S; continuous_on S r; homotopic_with_canon (\lambda x. True) S S id f; \rrbracket$

$f \in S \rightarrow T; r \in S \rightarrow T; \forall x \in T. r x = x \rrbracket$
 $\implies homotopic_with_canon (\lambda x. True) S S f r$

apply (*rule_tac f = r o f and g = r o id in homotopic_with_eq*)

apply (*rule_tac Y = S in homotopic_with_compose_continuous_left*)

apply (*auto simp: homotopic_with_sym Pi_iff*)

done

with *R homotopic_with_trans show ?lhs*

unfolding *retract_of_def retraction_def* **by** *blast*

qed

lemma *deformation_retract_of_contractible_sing*:

fixes $S :: 'a::euclidean_space\ set$

assumes *contractible S a ∈ S*

obtains *r where homotopic_with_canon (\lambda x. True) S S id r retraction S {a} r*

proof –

have $\{a\} retract_of S$

by (*simp add: ‹a ∈ S›*)

moreover have *homotopic_with_canon (\lambda x. True) S S id (\lambda x. a)*

using *assms*

by (*auto simp: contractible_def homotopic_into_contractible image_subset_iff*)

moreover have $(\lambda x. a) \in S \rightarrow \{a\}$

by (*simp add: image_subsetI*)

ultimately show *?thesis*

by (*metis that deformation_retract*)

qed

lemma *continuous_on_compact_surface_projection_aux*:

fixes $S :: 'a::t2_space\ set$

assumes *compact S S ⊆ T image q T ⊆ S*

and *contp: continuous_on T p*

```

    and  $\bigwedge x. x \in S \implies q x = x$ 
    and [simp]:  $\bigwedge x. x \in T \implies q(p x) = q x$ 
    and  $\bigwedge x. x \in T \implies p(q x) = p x$ 
    shows continuous_on T q
  proof -
    have *: image p T = image p S
      using assms by auto (metis imageI subset_iff)
    have contp': continuous_on S p
      by (rule continuous_on_subset [OF contp ‹S ⊆ T›])
    have continuous_on (p ' T) q
      by (simp add: * assms(1) assms(2) assms(5) continuous_on_inv contp' rev_subsetD)
    then have continuous_on T (q ∘ p)
      by (rule continuous_on_compose [OF contp])
    then show ?thesis
      by (rule continuous_on_eq [of _ q ∘ p]) (simp add: o_def)
  qed

```

lemma *continuous_on_compact_surface_projection*:

```

  fixes S :: 'a::real_normed_vector set
  assumes compact S
    and S: S ⊆ V - {0} and cone V
    and iff:  $\bigwedge k. x \in V - \{0\} \implies 0 < k \wedge (k *_R x) \in S \longleftrightarrow d x = k$ 
  shows continuous_on (V - {0}) ( $\lambda x. d x *_R x$ )
  proof (rule continuous_on_compact_surface_projection_aux [OF ‹compact S› S])
    show ( $\lambda x. d x *_R x$ ) ' (V - {0}) ⊆ S
      using iff by auto
    show continuous_on (V - {0}) ( $\lambda x. inverse(norm x) *_R x$ )
      by (intro continuous_intros) force
    show  $\bigwedge x. x \in S \implies d x *_R x = x$ 
      by (metis S zero_less_one local.iff scaleR_one subset_eq)
    show d (x /R norm x) *_R (x /R norm x) = d x *_R x if x ∈ V - {0} for x
      using iff [of inverse(norm x) *_R x norm x * d x, symmetric] iff that ‹cone V›
      by (simp add: field_simps cone_def zero_less_mult_iff)
    show d x *_R x /R norm (d x *_R x) = x /R norm x if x ∈ V - {0} for x
  proof -
    have 0 < d x
      using local.iff that by blast
    then show ?thesis
      by simp
  qed
  qed

```

9.18.2 Kuhn Simplices

lemma *bij_betw_singleton_eq*:

```

  assumes f: bij_betw f A B and g: bij_betw g A B and a: a ∈ A
  assumes eq: ( $\bigwedge x. x \in A \implies x \neq a \implies f x = g x$ )
  shows f a = g a

```

proof –

have $f \text{ ` } (A - \{a\}) = g \text{ ` } (A - \{a\})$
 by (intro image_cong) (simp_all add: eq)
 then have $B - \{f a\} = B - \{g a\}$
 using $f g a$ by (auto simp: bij_betw_def inj_on_image_set_diff set_eq_iff)
 moreover have $f a \in B$ $g a \in B$
 using $f g a$ by (auto simp: bij_betw_def)
 ultimately show ?thesis
 by auto

qed

lemmas swap_apply1 = swap_apply(1)

lemmas swap_apply2 = swap_apply(2)

lemma pointwise_minimal_pointwise_maximal:

fixes $s :: (\text{nat} \Rightarrow \text{nat}) \text{ set}$

assumes finite s

and $s \neq \{\}$

and $\forall x \in s. \forall y \in s. x \leq y \vee y \leq x$

shows $\exists a \in s. \forall x \in s. a \leq x$

and $\exists a \in s. \forall x \in s. x \leq a$

using assms

proof (induct s rule: finite_ne_induct)

case (insert $b s$)

assume *: $\forall x \in \text{insert } b s. \forall y \in \text{insert } b s. x \leq y \vee y \leq x$

then obtain $u l$ where $l \in s \forall b \in s. l \leq b$ $u \in s \forall b \in s. b \leq u$

using insert by auto

with * show $\exists a \in \text{insert } b s. \forall x \in \text{insert } b s. a \leq x$ $\exists a \in \text{insert } b s. \forall x \in \text{insert } b s.$
 $x \leq a$

by (metis insert_iff order.trans)+

qed auto

lemma kuhn_labelling_lemma:

fixes $P Q :: 'a :: \text{euclidean_space} \Rightarrow \text{bool}$

assumes $\forall x. P x \longrightarrow P (f x)$

and $\forall x. P x \longrightarrow (\forall i \in \text{Basis}. Q i \longrightarrow 0 \leq x \cdot i \wedge x \cdot i \leq 1)$

shows $\exists l. (\forall x. \forall i \in \text{Basis}. l x i \leq (1 :: \text{nat})) \wedge$

$(\forall x. \forall i \in \text{Basis}. P x \wedge Q i \wedge (x \cdot i = 0) \longrightarrow (l x i = 0)) \wedge$

$(\forall x. \forall i \in \text{Basis}. P x \wedge Q i \wedge (x \cdot i = 1) \longrightarrow (l x i = 1)) \wedge$

$(\forall x. \forall i \in \text{Basis}. P x \wedge Q i \wedge (l x i = 0) \longrightarrow x \cdot i \leq f x \cdot i) \wedge$

$(\forall x. \forall i \in \text{Basis}. P x \wedge Q i \wedge (l x i = 1) \longrightarrow f x \cdot i \leq x \cdot i)$

proof –

{ fix $x i$

let $?R = \lambda y. (P x \wedge Q i \wedge x \cdot i = 0 \longrightarrow y = (0 :: \text{nat})) \wedge$

$(P x \wedge Q i \wedge x \cdot i = 1 \longrightarrow y = 1) \wedge$

$(P x \wedge Q i \wedge y = 0 \longrightarrow x \cdot i \leq f x \cdot i) \wedge$

$(P x \wedge Q i \wedge y = 1 \longrightarrow f x \cdot i \leq x \cdot i)$

{ assume $P x Q i i \in \text{Basis}$ with assms have $0 \leq f x \cdot i \wedge f x \cdot i \leq 1$ by
 auto }

```

    then have  $i \in \text{Basis} \implies ?R\ 0 \vee ?R\ 1$  by auto }
  then show ?thesis
    unfolding all_conj_distrib[symmetric] Ball_def
    by (subst choice_iff[symmetric])+ blast
qed

```

The key "counting" observation, somewhat abstracted

lemma kuhn_counting_lemma:

```

  fixes bnd compo compo' face S F
  defines nF s == card {f ∈ F. face f s ∧ compo' f}
  assumes [simp, intro]: finite F — faces and [simp, intro]: finite S — simplices
    and  $\bigwedge f. f \in F \implies \text{bnd } f \implies \text{card } \{s \in S. \text{face } f s\} = 1$ 
    and  $\bigwedge f. f \in F \implies \neg \text{bnd } f \implies \text{card } \{s \in S. \text{face } f s\} = 2$ 
    and  $\bigwedge s. s \in S \implies \text{compo } s \implies nF\ s = 1$ 
    and  $\bigwedge s. s \in S \implies \neg \text{compo } s \implies nF\ s = 0 \vee nF\ s = 2$ 
    and odd (card {f ∈ F. compo' f ∧ bnd f})
  shows odd (card {s ∈ S. compo s})
proof -
  have  $(\sum s \mid s \in S \wedge \neg \text{compo } s. nF\ s) + (\sum s \mid s \in S \wedge \text{compo } s. nF\ s) =$ 
 $(\sum s \in S. nF\ s)$ 
  by (subst sum.union_disjoint[symmetric]) (auto intro!: sum.cong)
  also have ... =  $(\sum s \in S. \text{card } \{f \in \{f \in F. \text{compo}' f \wedge \text{bnd } f\}. \text{face } f s\}) +$ 
 $(\sum s \in S. \text{card } \{f \in \{f \in F. \text{compo}' f \wedge \neg \text{bnd } f\}. \text{face } f s\})$ 
  unfolding sum.distrib[symmetric]
  by (subst card_Un_disjoint[symmetric])
  (auto simp: nF_def intro!: sum.cong arg_cong[where f=card])
  also have ... =  $1 * \text{card } \{f \in F. \text{compo}' f \wedge \text{bnd } f\} + 2 * \text{card } \{f \in F. \text{compo}' f$ 
 $\wedge \neg \text{bnd } f\}$ 
  using assms(4,5) by (fastforce intro!: arg_cong2[where f=(+)] sum_multicount)
  finally have odd  $((\sum s \mid s \in S \wedge \neg \text{compo } s. nF\ s) + \text{card } \{s \in S. \text{compo } s\})$ 
  using assms(6,8) by simp
  moreover have  $(\sum s \mid s \in S \wedge \neg \text{compo } s. nF\ s) =$ 
 $(\sum s \mid s \in S \wedge \neg \text{compo } s \wedge nF\ s = 0. nF\ s) + (\sum s \mid s \in S \wedge \neg \text{compo } s \wedge$ 
 $nF\ s = 2. nF\ s)$ 
  using assms(7) by (subst sum.union_disjoint[symmetric]) (fastforce intro!:
  sum.cong)+
  ultimately show ?thesis
  by auto
qed

```

The odd/even result for faces of complete vertices, generalized

lemma kuhn_complete_lemma:

```

  assumes [simp]: finite simplices
    and face:  $\bigwedge f s. \text{face } f s \longleftrightarrow (\exists a \in s. f = s - \{a\})$ 
    and card_s[simp]:  $\bigwedge s. s \in \text{simplices} \implies \text{card } s = n + 2$ 
    and rl_bd:  $\bigwedge s. s \in \text{simplices} \implies \text{rl } 's \subseteq \{.. \text{Suc } n\}$ 
    and bnd:  $\bigwedge f s. s \in \text{simplices} \implies \text{face } f s \implies \text{bnd } f \implies \text{card } \{s \in \text{simplices}. \text{face}$ 
 $f s\} = 1$ 

```



```

  and nbnd:  $\bigwedge f s. s \in \text{simplices} \implies \text{face } f s \implies \neg \text{bnd } f \implies \text{card } \{s \in \text{simplices}. \text{face } f s\} = 2$ 
  and odd_card:  $\text{odd } (\text{card } \{f. (\exists s \in \text{simplices}. \text{face } f s) \wedge \text{rl } 'f = \{..n\} \wedge \text{bnd } f\})$ 
  shows  $\text{odd } (\text{card } \{s \in \text{simplices}. (\text{rl } 's = \{..Suc\ } n)\})$ 
proof (rule kuhn_counting_lemma)
  have finite_s[simp]:  $\bigwedge s. s \in \text{simplices} \implies \text{finite } s$ 
  by (metis add_is_0 zero_neq_numeral card.infinite assms(3))

  let ?F =  $\{f. \exists s \in \text{simplices}. \text{face } f s\}$ 
  have F_eq:  $?F = (\bigcup s \in \text{simplices}. \bigcup a \in s. \{s - \{a\}\})$ 
  by (auto simp: face)
  show finite ?F
  using  $\langle \text{finite simplices} \rangle$  unfolding F_eq by auto

  show  $\text{card } \{s \in \text{simplices}. \text{face } f s\} = 1$  if  $f \in ?F \text{ bnd } f$  for  $f$ 
  using bnd that by auto

  show  $\text{card } \{s \in \text{simplices}. \text{face } f s\} = 2$  if  $f \in ?F \neg \text{bnd } f$  for  $f$ 
  using nbnd that by auto

  show  $\text{odd } (\text{card } \{f \in \{f. \exists s \in \text{simplices}. \text{face } f s\}. \text{rl } 'f = \{..n\} \wedge \text{bnd } f\})$ 
  using odd_card by simp

  fix s assume s[simp]:  $s \in \text{simplices}$ 
  let ?S =  $\{f \in \{f. \exists s \in \text{simplices}. \text{face } f s\}. \text{face } f s \wedge \text{rl } 'f = \{..n\}\}$ 
  have ?S =  $(\lambda a. s - \{a\}) ' \{a \in s. \text{rl } '(s - \{a\}) = \{..n\}\}$ 
  using s by (fastforce simp: face)
  then have card_S:  $\text{card } ?S = \text{card } \{a \in s. \text{rl } '(s - \{a\}) = \{..n\}\}$ 
  by (auto intro!: card_image inj_onI)

  { assume rl:  $\text{rl } 's = \{..Suc\ } n$ 
  then have inj_rl:  $\text{inj\_on } \text{rl } s$ 
  by (intro eq_card_imp_inj_on) auto
  moreover obtain a where  $\text{rl } a = \text{Suc } n \ a \in s$ 
  by (metis atMost_iff image_iff le_Suc_eq rl)
  ultimately have n:  $\{..n\} = \text{rl } '(s - \{a\})$ 
  by (auto simp: inj_on_image_set_diff rl)
  have  $\{a \in s. \text{rl } '(s - \{a\}) = \{..n\}\} = \{a\}$ 
  using inj_rl  $\langle a \in s \rangle$  by (auto simp: n inj_on_image_eq_iff[OF inj_rl])
  then show  $\text{card } ?S = 1$ 
  unfolding card_S by simp }

  { assume rl:  $\text{rl } 's \neq \{..Suc\ } n$ 
  show  $\text{card } ?S = 0 \vee \text{card } ?S = 2$ 
  proof cases
  assume *:  $\{..n\} \subseteq \text{rl } 's$ 
  with rl_rl_bd[OF s] have rl_s:  $\text{rl } 's = \{..n\}$ 
  by (auto simp: atMost_Suc subset_insert_iff split: if_split_asm)
  then have  $\neg \text{inj\_on } \text{rl } s$ 

```

```

    by (intro pigeonhole) simp
  then obtain a b where ab: a ∈ s b ∈ s rl a = rl b a ≠ b
    by (auto simp: inj_on_def)
  then have eq: rl ' (s - {a}) = rl ' s
    by auto
  with ab have inj: inj_on rl (s - {a})
    by (intro eq_card_imp_inj_on) (auto simp: rl_s card_Diff_singleton_if)

{ fix x assume x ∈ s x ∉ {a, b}
  then have rl ' s - {rl x} = rl ' ((s - {a}) - {x})
    by (auto simp: eq inj_on_image_set_diff[OF inj])
  also have ... = rl ' (s - {x})
    using ab ⟨x ∉ {a, b}⟩ by auto
  also assume ... = rl ' s
  finally have False
    using ⟨x ∈ s⟩ by auto }
moreover
{ fix x assume x ∈ {a, b} with ab have x ∈ s ∧ rl ' (s - {x}) = rl ' s
  by (simp add: set_eq_iff_image_iff Bex_def) metis }
ultimately have {a ∈ s. rl ' (s - {a}) = {..n}} = {a, b}
  unfolding rl_s[symmetric] by fastforce
with ⟨a ≠ b⟩ show card ?S = 0 ∨ card ?S = 2
  unfolding card_S by simp
next
assume ¬ {..n} ⊆ rl ' s
then have ∧x. rl ' (s - {x}) ≠ {..n}
  by auto
then show card ?S = 0 ∨ card ?S = 2
  unfolding card_S by simp
qed }
qed fact

```

locale *kuhn_simplex* =

```

  fixes p n and base upd and S :: (nat ⇒ nat) set
  assumes base: base ∈ {..< n} → {..< p}
  assumes base_out: ∧i. n ≤ i ⇒ base i = p
  assumes upd: bij_betw upd {..< n} {..< n}
  assumes s_pre: S = (λi j. if j ∈ upd {..< i} then Suc (base j) else base j) ' {..
n}
begin

```

definition *enum* i j = (if j ∈ upd {..< i} then Suc (base j) else base j)

```

lemma s_eq: S = enum ' {.. n}
  unfolding s_pre enum_def[abs_def] ..

```

```

lemma upd_space: i < n ⇒ upd i < n
  using upd by (auto dest!: bij_betwE)

```

```

lemma s_space:  $S \subseteq \{.. < n\} \rightarrow \{.. p\}$ 
proof -
  { fix i assume  $i \leq n$  then have  $enum\ i \in \{.. < n\} \rightarrow \{.. p\}$ 
    proof (induct i)
      case 0 then show ?case
        using base by (auto simp: Pi_iff less_imp_le enum_def)
      next
        case (Suc i) with base show ?case
          by (auto simp: Pi_iff Suc_le_eq less_imp_le enum_def intro: upd_space)
    qed }
  then show ?thesis
    by (auto simp: s_eq)
qed

lemma inj_upd:  $inj\_on\ upd\ \{.. < n\}$ 
  using upd by (simp add: bij_betw_def)

lemma inj_enum:  $inj\_on\ enum\ \{.. n\}$ 
proof -
  { fix x y :: nat assume  $x \neq y\ x \leq n\ y \leq n$ 
    with upd have  $upd\ \{.. < x\} \neq upd\ \{.. < y\}$ 
    by (subst inj_on_image_eq_iff[where C= $\{.. < n\}$ ]) (auto simp: bij_betw_def)
    then have  $enum\ x \neq enum\ y$ 
    by (auto simp: enum_def fun_eq_iff) }
  then show ?thesis
    by (auto simp: inj_on_def)
qed

lemma enum_0:  $enum\ 0 = base$ 
  by (simp add: enum_def[abs_def])

lemma base_in_s:  $base \in S$ 
  unfolding s_eq by (subst enum_0[symmetric]) auto

lemma enum_in:  $i \leq n \implies enum\ i \in S$ 
  unfolding s_eq by auto

lemma one_step:
  assumes a:  $a \in S\ j < n$ 
  assumes *:  $\bigwedge a'. a' \in S \implies a' \neq a \implies a' j = p'$ 
  shows  $a j \neq p'$ 
proof
  assume  $a j = p'$ 
  with * a have  $\bigwedge a'. a' \in S \implies a' j = p'$ 
    by auto
  then have  $\bigwedge i. i \leq n \implies enum\ i j = p'$ 
    unfolding s_eq by auto
  from this[of 0] this[of n] have  $j \notin upd\ \{.. < n\}$ 
    by (auto simp: enum_def fun_eq_iff split: if_split_asm)

```

3280

with $\text{upd } \langle j < n \rangle$ **show** False
by ($\text{auto simp: bij_betw_def}$)
qed

lemma upd_inj : $i < n \implies j < n \implies \text{upd } i = \text{upd } j \longleftrightarrow i = j$
using upd **by** ($\text{auto simp: bij_betw_def inj_on_eq_iff}$)

lemma upd_surj : $\text{upd } \{.. < n\} = \{.. < n\}$
using upd **by** ($\text{auto simp: bij_betw_def}$)

lemma in_upd_image : $A \subseteq \{.. < n\} \implies i < n \implies \text{upd } i \in \text{upd } A \longleftrightarrow i \in A$
using $\text{inj_on_image_mem_iff}$ [$\text{of upd } \{.. < n\}$] upd
by ($\text{auto simp: bij_betw_def}$)

lemma enum_inj : $i \leq n \implies j \leq n \implies \text{enum } i = \text{enum } j \longleftrightarrow i = j$
using inj_enum **by** ($\text{auto simp: inj_on_eq_iff}$)

lemma in_enum_image : $A \subseteq \{.. n\} \implies i \leq n \implies \text{enum } i \in \text{enum } A \longleftrightarrow i \in A$
using $\text{inj_on_image_mem_iff}$ [OF inj_enum] **by** auto

lemma enum_mono : $i \leq n \implies j \leq n \implies \text{enum } i \leq \text{enum } j \longleftrightarrow i \leq j$
by ($\text{auto simp: enum_def le_fun_def in_upd_image Ball_def[symmetric]}$)

lemma enum_strict_mono : $i \leq n \implies j \leq n \implies \text{enum } i < \text{enum } j \longleftrightarrow i < j$
using enum_mono [$\text{of } i \ j$] enum_inj [$\text{of } i \ j$] **by** ($\text{auto simp: le_less}$)

lemma chain : $a \in S \implies b \in S \implies a \leq b \vee b \leq a$
by ($\text{auto simp: s_eq enum_mono}$)

lemma less : $a \in S \implies b \in S \implies a < b \implies a < b$
using chain [$\text{of } a \ b$] **by** ($\text{auto simp: less_fun_def le_fun_def not_le[symmetric]}$)

lemma enum_0_bot : $a \in S \implies a = \text{enum } 0 \longleftrightarrow (\forall a' \in S. a \leq a')$
unfolding s_eq **by** ($\text{auto simp: enum_mono Ball_def}$)

lemma enum_n_top : $a \in S \implies a = \text{enum } n \longleftrightarrow (\forall a' \in S. a' \leq a)$
unfolding s_eq **by** ($\text{auto simp: enum_mono Ball_def}$)

lemma enum_Suc : $i < n \implies \text{enum } (\text{Suc } i) = (\text{enum } i)(\text{upd } i := \text{Suc } (\text{enum } i (\text{upd } i)))$
by ($\text{auto simp: fun_eq_iff enum_def upd_inj}$)

lemma enum_eq_p : $i \leq n \implies n \leq j \implies \text{enum } i \ j = p$
by ($\text{induct } i$) ($\text{auto simp: enum_Suc enum_0 base_out upd_space not_less[symmetric]}$)

lemma out_eq_p : $a \in S \implies n \leq j \implies a \ j = p$
unfolding s_eq **by** ($\text{auto simp: enum_eq_p}$)

```

lemma s_le_p:  $a \in S \implies a j \leq p$ 
  using out_eq_p[of a j] s_space by (cases j < n) auto

lemma le_Suc_base:  $a \in S \implies a j \leq \text{Suc } (\text{base } j)$ 
  unfolding s_eq by (auto simp: enum_def)

lemma base_le:  $a \in S \implies \text{base } j \leq a j$ 
  unfolding s_eq by (auto simp: enum_def)

lemma enum_le_p:  $i \leq n \implies j < n \implies \text{enum } i j \leq p$ 
  using enum_in[of i] s_space by auto

lemma enum_less:  $a \in S \implies i < n \implies \text{enum } i < a \longleftrightarrow \text{enum } (\text{Suc } i) \leq a$ 
  unfolding s_eq by (auto simp: enum_strict_mono enum_mono)

lemma ksimplex_0:
   $n = 0 \implies S = \{(\lambda x. p)\}$ 
  using s_eq enum_def base_out by auto

lemma replace_0:
  assumes  $j < n$   $a \in S$  and  $p: \forall x \in S - \{a\}. x j = 0$  and  $x \in S$ 
  shows  $x \leq a$ 
proof cases
  assume  $x \neq a$ 
  have  $a j \neq 0$ 
    using assms by (intro one_step[where  $a=a$ ]) auto
  with less[OF  $\langle x \in S \rangle \langle a \in S \rangle$ , of j] p[rule_format, of x]  $\langle x \in S \rangle \langle x \neq a \rangle$ 
  show ?thesis
    by auto
qed simp

lemma replace_1:
  assumes  $j < n$   $a \in S$  and  $p: \forall x \in S - \{a\}. x j = p$  and  $x \in S$ 
  shows  $a \leq x$ 
proof cases
  assume  $x \neq a$ 
  have  $a j \neq p$ 
    using assms by (intro one_step[where  $a=a$ ]) auto
  with enum_le_p[of  $\_ j$ ]  $\langle j < n \rangle \langle a \in S \rangle$ 
  have  $a j < p$ 
    by (auto simp: less_le s_eq)
  with less[OF  $\langle a \in S \rangle \langle x \in S \rangle$ , of j] p[rule_format, of x]  $\langle x \in S \rangle \langle x \neq a \rangle$ 
  show ?thesis
    by auto
qed simp

end

locale kuhn_simplex_pair = s: kuhn_simplex p n b_s u_s s + t: kuhn_simplex

```

3282

```
p n b_t u_t t
  for p n b_s u_s s b_t u_t t
begin

lemma enum_eq:
  assumes l:  $i \leq l \leq j$  and  $j + d \leq n$ 
  assumes eq:  $s.enum \{i .. j\} = t.enum \{i + d .. j + d\}$ 
  shows  $s.enum l = t.enum (l + d)$ 
using l proof (induct l rule: dec_induct)
  case base
  then have  $s.enum i \in t.enum \{i + d .. j + d\}$  and  $t.enum (i + d) \in s.enum \{i .. j\}$ 
  using eq by auto
  from t  $\langle i \leq j \rangle \langle j + d \leq n \rangle$  have  $s.enum i \leq t.enum (i + d)$ 
  by (auto simp: s.enum_mono)
  moreover from s  $\langle i \leq j \rangle \langle j + d \leq n \rangle$  have  $t.enum (i + d) \leq s.enum i$ 
  by (auto simp: t.enum_mono)
  ultimately show ?case
  by auto
next
case (step l)
moreover from step.premis  $\langle j + d \leq n \rangle$  have
   $s.enum l < s.enum (Suc l)$ 
   $t.enum (l + d) < t.enum (Suc l + d)$ 
  by (simp_all add: s.enum_strict_mono t.enum_strict_mono)
moreover have
   $s.enum (Suc l) \in t.enum \{i + d .. j + d\}$ 
   $t.enum (Suc l + d) \in s.enum \{i .. j\}$ 
  using step  $\langle j + d \leq n \rangle$  eq by (auto simp: s.enum_inj t.enum_inj)
ultimately have  $s.enum (Suc l) = t.enum (Suc (l + d))$ 
  using  $\langle j + d \leq n \rangle$ 
  by (intro antisym s.enum_less[THEN iffD1] t.enum_less[THEN iffD1])
  (auto intro!: s.enum_in t.enum_in)
then show ?case by simp
qed
```

```
lemma ksimpler_eq_bot:
  assumes a:  $a \in s \wedge a'. a' \in s \implies a \leq a'$ 
  assumes b:  $b \in t \wedge b'. b' \in t \implies b \leq b'$ 
  assumes eq:  $s - \{a\} = t - \{b\}$ 
  shows  $s = t$ 
proof cases
  assume  $n = 0$  with s.ksimpler_0 t.ksimpler_0 show ?thesis by simp
next
  assume  $n \neq 0$ 
  have  $s.enum 0 = (s.enum (Suc 0)) (u_s 0 := s.enum (Suc 0) (u_s 0) - 1)$ 
     $t.enum 0 = (t.enum (Suc 0)) (u_t 0 := t.enum (Suc 0) (u_t 0) - 1)$ 
  using  $\langle n \neq 0 \rangle$  by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have  $e0: a = s.enum 0 \ b = t.enum 0$ 
```

```

    using a b by (simp_all add: s.enum_0_bot t.enum_0_bot)
  moreover
  { fix j assume 0 < j j ≤ n
    moreover have s - {a} = s.enum ' {Suc 0 .. n} t - {b} = t.enum ' {Suc 0
  .. n}
    unfolding s.s_eq t.s_eq e0 by (auto simp: s.enum_inj t.enum_inj)
    ultimately have s.enum j = t.enum j
      using enum_eq[of 1 j n 0] eq by auto }
  note enum_eq = this
  then have s.enum (Suc 0) = t.enum (Suc 0)
    using ⟨n ≠ 0⟩ by auto
  moreover
  { fix j assume Suc j < n
    with enum_eq[of Suc j] enum_eq[of Suc (Suc j)]
    have u_s (Suc j) = u_t (Suc j)
      using s.enum_Suc[of Suc j] t.enum_Suc[of Suc j]
      by (auto simp: fun_eq_iff split: if_split_asm) }
  then have  $\bigwedge j. 0 < j \implies j < n \implies u_s j = u_t j$ 
    by (auto simp: gr0_conv_Suc)
  with ⟨n ≠ 0⟩ have u_t 0 = u_s 0
    by (intro bij_betw_singleton_eq[OF t.upd s.upd, of 0]) auto
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

```

lemma *ksimplex_eq_top*:

```

  assumes a: a ∈ s  $\bigwedge$  a'. a' ∈ s  $\implies$  a' ≤ a
  assumes b: b ∈ t  $\bigwedge$  b'. b' ∈ t  $\implies$  b' ≤ b
  assumes eq: s - {a} = t - {b}
  shows s = t
proof (cases n)
  assume n = 0 with s.ksimplex_0 t.ksimplex_0 show ?thesis by simp
next
  case (Suc n')
  have s.enum n = (s.enum n') (u_s n' := Suc (s.enum n' (u_s n')))
    t.enum n = (t.enum n') (u_t n' := Suc (t.enum n' (u_t n')))
    using Suc by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have en: a = s.enum n b = t.enum n
    using a b by (simp_all add: s.enum_n_top t.enum_n_top)
  moreover
  { fix j assume j < n
    moreover have s - {a} = s.enum ' {0 .. n'} t - {b} = t.enum ' {0 .. n'}
      unfolding s.s_eq t.s_eq en by (auto simp: s.enum_inj t.enum_inj Suc)
    ultimately have s.enum j = t.enum j
      using enum_eq[of 0 j n' 0] eq Suc by auto }
  note enum_eq = this
  then have s.enum n' = t.enum n'

```

```

    using Suc by auto
  moreover
  { fix j assume j < n'
    with enum_eq[of j] enum_eq[of Suc j]
    have u_s j = u_t j
      using s.enum_Suc[of j] t.enum_Suc[of j]
      by (auto simp: Suc fun_eq_iff split: if_split_asm) }
  then have  $\bigwedge j. j < n' \implies u_s j = u_t j$ 
    by (auto simp: gr0_conv_Suc)
  then have u_t n' = u_s n'
    by (intro bij_betw_singleton_eq[OF t.upd s.upd, of n']) (auto simp: Suc)
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

end

```

inductive *ksimplex* for $p\ n :: \text{nat}$ where

ksimplex: $\text{kuhn_simplex } p\ n\ \text{base } \text{upd } s \implies \text{ksimplex } p\ n\ s$

lemma *finite_ksimplexes*: $\text{finite } \{s. \text{ksimplex } p\ n\ s\}$

proof (rule *finite_subset*)

```

  { fix a s assume ksimplex p n s a ∈ s
    then obtain b u where kuhn_simplex p n b u s by (auto elim: ksimplex.cases)
    then interpret kuhn_simplex p n b u s .
    from s_space ⟨a ∈ s⟩ out_eq_p[OF ⟨a ∈ s⟩]
    have a ∈ (λf x. if n ≤ x then p else f x) ‘({.. $n$ } →E {.. p})
      by (auto simp: image_iff subset_eq Pi_iff split: if_split_asm
        intro!: bexI[of _ restrict a {.. $n$ ]]) }
    then show {s. ksimplex p n s} ⊆ Pow ((λf x. if n ≤ x then p else f x) ‘({.. $n$ }
      →E {.. p}))
      by auto
  qed (simp add: finite_PiE)

```

lemma *ksimplex_card*:

assumes *ksimplex* $p\ n\ s$ shows $\text{card } s = \text{Suc } n$

using *assms* **proof** *cases*

case (*ksimplex* $u\ b$)

then interpret *kuhn_simplex* $p\ n\ u\ b\ s$.

show *?thesis*

by (*simp* add: *card_image_s_eq_inj_enum*)

qed

lemma *simplex_top_face*:

assumes $0 < p \forall x \in s'. x\ n = p$

shows $\text{ksimplex } p\ n\ s' \longleftrightarrow (\exists s\ a. \text{ksimplex } p\ (\text{Suc } n)\ s \wedge a \in s \wedge s' = s - \{a\})$

using *assms*


```

proof safe
  fix s a assume ksimplex p (Suc n) s and a: a ∈ s and na: ∀x∈s - {a}. x n =
  p
  then show ksimplex p n (s - {a})
  proof cases
    case (ksimplex base upd)
      then interpret kuhn_simplex p Suc n base upd s .

      have a n < p
        using one_step[of a n p] na ‹a∈s› s_space by (auto simp: less_le)
      then have a = enum 0
        using ‹a ∈ s› na by (subst enum_0_bot) (auto simp: le_less intro!: less[of a
  - n])
      then have s_eq: s - {a} = enum ‹Suc ‹{.. n}›
        using s_eq by (simp add: atMost_Suc_eq_insert_0 insert_ident in_enum_image
  subset_eq)
      then have enum 1 ∈ s - {a}
        by auto
      then have upd 0 = n
        using ‹a n < p› ‹a = enum 0› na[rule_format, of enum 1]
        by (auto simp: fun_eq_iff enum_Suc split: if_split_asm)
      then have bij_betw upd (Suc ‹{..<n}›) {..<n}
        using upd
        by (subst notIn_Un_bij_betw3[where b=0])
          (auto simp: lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)
      then have bij_betw (upd∘Suc) {..<n} {..<n}
        by (rule bij_betw_trans[rotated]) (auto simp: bij_betw_def)

      have a n = p - 1
        using enum_Suc[of 0] na[rule_format, OF ‹enum 1 ∈ s - {a}›] ‹a = enum
  0› by (auto simp: ‹upd 0 = n›)

      show ?thesis
      proof (rule ksimplex.intros, standard)
        show bij_betw (upd∘Suc) {..<n} {..<n} by fact
        show base(n := p) ∈ {..<n} → {..<p} ∧ i. n ≤ i ⇒ (base(n := p)) i = p
          using base base_out by (auto simp: Pi_iff)

        have ∧i. Suc ‹{..<i}› = {..<Suc i} - {0}
          by (auto simp: image_iff Ball_def) arith
        then have upd_Suc: ∧i. i ≤ n ⇒ (upd∘Suc) ‹{..<i}› = upd ‹{..<Suc i}
  - {n}›
          using ‹upd 0 = n› upd_inj by (auto simp add: image_iff less_Suc_eq_0_disj)
        have n_in_upd: ∧i. n ∈ upd ‹{..<Suc i}›
          using ‹upd 0 = n› by auto

        define f' where f' i j =
          (if j ∈ (upd∘Suc) {..<i} then Suc ((base(n := p)) j) else (base(n := p)) j)
        for i j

```

```

{ fix x i
  assume i [arith]: i ≤ n
  with upd_Suc have (upd ∘ Suc) ‘{..} = upd ‘{..} - {n} .
  with ⟨a n < p⟩ ⟨a = enum 0⟩ ⟨upd 0 = n⟩ ⟨a n = p - 1⟩
  have enum (Suc i) x = f' i x
    by (auto simp add: f'_def enum_def) }
then show s - {a} = f' ‘{.. n}
  unfolding s_eq image_comp by (intro image_cong) auto
qed
qed
next
assume ksimplex p n s' and *: ∀ x ∈ s'. x n = p
then show ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s - {a}
proof cases
  case (ksimplex base upd)
  then interpret kuhn_simplex p n base upd s' .
  define b where b = base (n := p - 1)
  define u where u i = (case i of 0 ⇒ n | Suc i ⇒ upd i) for i

  have ksimplex p (Suc n) (s' ∪ {b})
  proof (rule ksimplex.intros, standard)
    show b ∈ {..} → {..}
    using base ⟨0 < p⟩ unfolding lessThan_Suc b_def by (auto simp: PiE_iff)
    show ∧ i. Suc n ≤ i ⇒ b i = p
    using base_out by (auto simp: b_def)

  have bij_betw u (Suc ‘{..} ∪ {0}) ({..} ∪ {u 0})
    using upd
  by (intro notIn_Un_bij_betw) (auto simp: u_def bij_betw_def image_comp
comp_def inj_on_def)
  then show bij_betw u {..} {..}
  by (simp add: u_def lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)

  define f' where f' i j = (if j ∈ u ‘{..} then Suc (b j) else b j) for i j

  have u_eq: ∧ i. i ≤ n ⇒ u ‘{..} = upd ‘{..} ∪ {n}
    by (auto simp: u_def image_iff upd_inj Ball_def split: nat.split) arith

  { fix x have x ≤ n ⇒ n ∉ upd ‘{..}
    using upd_space by (simp add: image_iff neq_iff) }
  note n_not_upd = this

  have *: f' ‘{.. Suc n} = f' ‘(Suc ‘{.. n} ∪ {0})
    unfolding atMost_Suc_eq_insert_0 by simp
  also have ... = (f' ∘ Suc) ‘{.. n} ∪ {b}
    by (auto simp: f'_def)
  also have (f' ∘ Suc) ‘{.. n} = s'
    using ⟨0 < p⟩ base_out[of n]
  unfolding s_eq enum_def[abs_def] f'_def[abs_def] upd_space

```

```

    by (intro image_cong) (simp_all add: u_eq b_def fun_eq_iff n_not_upd)
  finally show  $s' \cup \{b\} = f' \text{ ` } \{.. \text{ Suc } n\} ..$ 
qed
moreover have  $b \notin s'$ 
  using *  $\langle 0 < p \rangle$  by (auto simp: b_def)
ultimately show ?thesis by auto
qed
qed

```

lemma *ksimplex_replace_0*:

```

  assumes  $s$ : ksimplex  $p$   $n$   $s$  and  $a$ :  $a \in s$ 
  assumes  $j$ :  $j < n$  and  $p$ :  $\forall x \in s - \{a\}. x j = 0$ 
  shows  $\text{card } \{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = 1$ 
  using  $s$ 
proof cases
  case (ksimplex  $b\_s$   $u\_s$ )

  { fix  $t$   $b$  assume ksimplex  $p$   $n$   $t$ 
    then obtain  $b\_t$   $u\_t$  where kuhn_simplex  $p$   $n$   $b\_t$   $u\_t$   $t$ 
      by (auto elim: ksimplex.cases)
    interpret kuhn_simplex_pair  $p$   $n$   $b\_s$   $u\_s$   $s$   $b\_t$   $u\_t$   $t$ 
      by intro_locales fact+

    assume  $b$ :  $b \in t t - \{b\} = s - \{a\}$ 
    with  $a$   $j$   $p$   $s$ .replace_0[of  $a$ ]  $t$ .replace_0[of  $b$ ] have  $s = t$ 
      by (intro ksimplex_eq_top[of  $a$   $b$ ]) auto }
    then have  $\{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = \{s\}$ 
      using  $s$   $\langle a \in s \rangle$  by auto
    then show ?thesis
      by simp
  }
qed

```

lemma *ksimplex_replace_1*:

```

  assumes  $s$ : ksimplex  $p$   $n$   $s$  and  $a$ :  $a \in s$ 
  assumes  $j$ :  $j < n$  and  $p$ :  $\forall x \in s - \{a\}. x j = p$ 
  shows  $\text{card } \{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = 1$ 
  using  $s$ 
proof cases
  case (ksimplex  $b\_s$   $u\_s$ )

  { fix  $t$   $b$  assume ksimplex  $p$   $n$   $t$ 
    then obtain  $b\_t$   $u\_t$  where kuhn_simplex  $p$   $n$   $b\_t$   $u\_t$   $t$ 
      by (auto elim: ksimplex.cases)
    interpret kuhn_simplex_pair  $p$   $n$   $b\_s$   $u\_s$   $s$   $b\_t$   $u\_t$   $t$ 
      by intro_locales fact+

    assume  $b$ :  $b \in t t - \{b\} = s - \{a\}$ 
    with  $a$   $j$   $p$   $s$ .replace_1[of  $a$ ]  $t$ .replace_1[of  $b$ ] have  $s = t$ 
      by (intro ksimplex_eq_bot[of  $a$   $b$ ]) auto }
  }

```

3288

then have $\{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = \{s\}$
using $s \langle a \in s \rangle$ **by** *auto*
then show *?thesis*
by *simp*
qed

lemma *ksimplex_replace_2*:

assumes $s: \text{ksimplex } p \ n \ s$ **and** $a \in s$ **and** $n \neq 0$
and $lb: \forall j < n. \exists x \in s - \{a\}. x \ j \neq 0$
and $ub: \forall j < n. \exists x \in s - \{a\}. x \ j \neq p$
shows $\text{card } \{s'. \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s' - \{b\} = s - \{a\})\} = 2$
using s

proof *cases*

case (*ksimplex_base_upd*)

then interpret *kuhn_simplex* $p \ n$ *base_upd* s .

from $\langle a \in s \rangle$ **obtain** i **where** $i \leq n$ $a = \text{enum } i$
unfolding *s_eq* **by** *auto*

from $\langle i \leq n \rangle$ **have** $i = 0 \vee i = n \vee (0 < i \wedge i < n)$
by *linarith*

then have $\exists! s'. s' \neq s \wedge \text{ksimplex } p \ n \ s' \wedge (\exists b \in s'. s - \{a\} = s' - \{b\})$

proof (*elim_disjE conjE*)

assume $i = 0$

define *rot* **where** [*abs_def*]: $\text{rot } i = (\text{if } i + 1 = n \text{ then } 0 \text{ else } i + 1)$ **for** i
let $?upd = \text{upd} \circ \text{rot}$

have $\text{rot}: \text{bij_betw } \text{rot } \{.. < n\} \ \{.. < n\}$

by (*auto simp: bij_betw_def inj_on_def image_iff Ball_def rot_def*)
arith+

from $\text{rot } \text{upd}$ **have** $\text{bij_betw } ?upd \ \{.. < n\} \ \{.. < n\}$

by (*rule bij_betw_trans*)

define f' **where** [*abs_def*]: $f' \ i \ j =$

$(\text{if } j \in ?upd \ \{.. < i\} \text{ then } \text{Suc } (\text{enum } (\text{Suc } 0) \ j) \text{ else } \text{enum } (\text{Suc } 0) \ j)$ **for** $i \ j$

interpret $b: \text{kuhn_simplex } p \ n \ \text{enum } (\text{Suc } 0) \ \text{upd} \circ \text{rot } f' \ \langle .. \ n \rangle$

proof

from $\langle a = \text{enum } i \rangle$ $ub \ \langle n \neq 0 \rangle$ $\langle i = 0 \rangle$

obtain i' **where** $i' \leq n$ $\text{enum } i' \neq \text{enum } 0$ $\text{enum } i' \ (\text{upd } 0) \neq p$

unfolding *s_eq* **by** (*auto intro: upd_space simp: enum_inj*)

then have $\text{enum } 1 \leq \text{enum } i' \ \text{enum } i' \ (\text{upd } 0) < p$

using $\text{enum_le_p}[\text{of } i' \ \text{upd } 0]$ **by** (*auto simp: enum_inj enum_mono upd_space*)

then have $\text{enum } 1 \ (\text{upd } 0) < p$

by (*auto simp: le_fun_def intro: le_less_trans*)

then show $\text{enum } (\text{Suc } 0) \in \{.. < n\} \rightarrow \{.. < p\}$

using $\text{base } \langle n \neq 0 \rangle$ **by** (*auto simp: enum_0 enum_Suc PiE_iff extensional_def upd_space*)

```

{ fix i assume n ≤ i then show enum (Suc 0) i = p
  using ⟨n ≠ 0⟩ by (auto simp: enum_eq_p) }
show bij_betw ?upd {.. $n$ } {.. $n$ } by fact
qed (simp add: f'_def)
have ks_f': ksimplex p n (f' ' {.. $n$ })
  by rule unfold_locales

have b_enum: b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
with b.inj_enum have inj_f': inj_on f' {.. $n$ } by simp

have f'_eq_enum: f' j = enum (Suc j) if j < n for j
proof -
  from that have rot ' {.. $j$ } = {0 <.. $Suc j$ }
  by (auto simp: rot_def image_Suc_lessThan cong: image_cong_simp)
  with that ⟨n ≠ 0⟩ show ?thesis
  by (simp only: f'_def enum_def fun_eq_iff image_comp [symmetric])
    (auto simp add: upd_inj)
qed
then have enum ' Suc ' {.. $n$ } = f' ' {.. $n$ }
  by (force simp: enum_inj)
also have Suc ' {.. $n$ } = {.. $n$ } - {0}
  by (auto simp: image_iff Ball_def) arith
also have {.. $n$ } = {.. $n$ } - {n}
  by auto
finally have eq: s - {a} = f' ' {.. $n$ } - {f' n}
  unfolding s_eq ⟨a = enum i⟩ ⟨i = 0⟩
  by (simp add: inj_on_image_set_diff[OF inj_enum] inj_on_image_set_diff[OF
inj_f'])

have enum 0 < f' 0
  using ⟨n ≠ 0⟩ by (simp add: enum_strict_mono f'_eq_enum)
also have ... < f' n
  using ⟨n ≠ 0⟩ b.enum_strict_mono[of 0 n] unfolding b_enum by simp
finally have a ≠ f' n
  using ⟨a = enum i⟩ ⟨i = 0⟩ by auto

{ fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
  obtain b u where kuhn_simplex p n b u t
  using ⟨ksimplex p n t⟩ by (auto elim: ksimplex.cases)
  then interpret t: kuhn_simplex p n b u t .

  { fix x assume x ∈ s x ≠ a
    then have x (upd 0) = enum (Suc 0) (upd 0)
      by (auto simp: ⟨a = enum i⟩ ⟨i = 0⟩ s_eq_enum_def enum_inj) }
  then have eq_upd0: ∀ x ∈ t - {c}. x (upd 0) = enum (Suc 0) (upd 0)
    unfolding eq_sma[symmetric] by auto
  then have c (upd 0) ≠ enum (Suc 0) (upd 0)
    using ⟨n ≠ 0⟩ by (intro t.one_step[OF ⟨c ∈ t⟩ ]) (auto simp: upd_space)

```

```

then have c (upd 0) < enum (Suc 0) (upd 0) ∨ c (upd 0) > enum (Suc 0)
(upd 0)
  by auto
then have t = s ∨ t = f' ' {..n}
proof (elim disjE conjE)
  assume *: c (upd 0) < enum (Suc 0) (upd 0)
  interpret st: kuhn_simplex_pair p n base upd s b u t ..
  { fix x assume x ∈ t with * <c∈t> eq_upd0[rule_format, of x] have c ≤ x
    by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
  note top = this
  have s = t
    using <a = enum i> <i = 0> <c ∈ t>
    by (intro st.ksimplex_eq_bot[OF _ _ _ _ eq_sma])
      (auto simp: s_eq_enum_mono t.s_eq t.enum_mono top)
  then show ?thesis by simp
next
assume *: c (upd 0) > enum (Suc 0) (upd 0)
interpret st: kuhn_simplex_pair p n enum (Suc 0) upd ∘ rot f' ' {.. n} b
u t ..
  have eq: f' ' {..n} - {f' n} = t - {c}
    using eq_sma eq by simp
  { fix x assume x ∈ t with * <c∈t> eq_upd0[rule_format, of x] have x ≤ c
    by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
  note top = this
  have f' ' {..n} = t
    using <a = enum i> <i = 0> <c ∈ t>
    by (intro st.ksimplex_eq_top[OF _ _ _ _ eq])
      (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono b_enum[symmetric]
top)
  then show ?thesis by simp
qed }
with ks_f' eq <a ≠ f' n> <n ≠ 0> show ?thesis
  apply (intro ex1I[of _ f' ' {.. n}])
  apply auto []
  apply metis
done
next
assume i = n
from <n ≠ 0> obtain n' where n': n = Suc n'
  by (cases n) auto

define rot where rot i = (case i of 0 ⇒ n' | Suc i ⇒ i) for i
let ?upd = upd ∘ rot

have rot: bij_betw rot {..<n} {..<n}
  by (auto simp: bij_betw_def inj_on_def image_iff Bex_def rot_def n' split:
nat.splits)
  arith
from rot upd have bij_betw ?upd {..<n} {..<n}

```

```

    by (rule bij_betw_trans)

  define b where b = base (upd n' := base (upd n') - 1)
  define f' where [abs_def]: f' i j = (if j ∈ ?upd' {..< i} then Suc (b j) else b
j) for i j

interpret b: kuhn_simplex p n b upd ∘ rot f' ' {.. n}
proof
  { fix i assume n ≤ i then show b i = p
    using base_out[of i] upd_space[of n'] by (auto simp: b_def n') }
  show b ∈ {..<n} → {..<p}
    using base ⟨n ≠ 0⟩ upd_space[of n']
    by (auto simp: b_def PiE_def Pi_iff Ball_def upd_space extensional_def
n')

  show bij_betw ?upd {..<n} {..<n} by fact
qed (simp add: f'_def)
have f': b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
have ks_f': ksimplex p n (b.enum ' {.. n})
  unfolding f' by rule unfold_locales

have 0 < n
  using ⟨n ≠ 0⟩ by auto

{ from ⟨a = enum i⟩ ⟨n ≠ 0⟩ ⟨i = n⟩ lb upd_space[of n']
  obtain i' where i' ≤ n enum i' ≠ enum n 0 < enum i' (upd n')
    unfolding s_eq by (auto simp: enum_inj n')
  moreover have enum i' (upd n') = base (upd n')
    unfolding enum_def using ⟨i' ≤ n⟩ ⟨enum i' ≠ enum n⟩ by (auto simp:
n' upd_inj enum_inj)
  ultimately have 0 < base (upd n')
    by auto }
then have benum1: b.enum (Suc 0) = base
  unfolding b.enum_Suc[OF ⟨0 < n⟩] b.enum_0 by (auto simp: b_def rot_def)

have [simp]: ∧j. Suc j < n ⇒ rot ' {..< Suc j} = {n'} ∪ {..< j}
  by (auto simp: rot_def image_iff Ball_def split: nat.splits)
have rot_simps: ∧j. rot (Suc j) = j rot 0 = n'
  by (simp_all add: rot_def)

{ fix j assume j: Suc j ≤ n then have b.enum (Suc j) = enum j
  by (induct j) (auto simp: benum1 enum_0 b.enum_Suc enum_Suc rot_simps)
}

note b_enum_eq_enum = this
then have enum ' {..< n} = b.enum ' Suc ' {..< n}
  by (auto simp: image_comp intro!: image_cong)
also have Suc ' {..< n} = {.. n} - {0}
  by (auto simp: image_iff Ball_def) arith
also have {..< n} = {.. n} - {n}

```

```

by auto
finally have eq: s - {a} = b.enum ' {.. n} - {b.enum 0}
unfolding s_eq ⟨a = enum i⟩ ⟨i = n⟩
using inj_on_image_set_diff[OF inj_enum Diff_subset, of {n}]
inj_on_image_set_diff[OF b.inj_enum Diff_subset, of {0}]
by (simp add: comp_def)

have b.enum 0 ≤ b.enum n
by (simp add: b.enum_mono)
also have b.enum n < enum n
using ⟨n ≠ 0⟩ by (simp add: enum_strict_mono b.enum_eq_enum n')
finally have a ≠ b.enum 0
using ⟨a = enum i⟩ ⟨i = n⟩ by auto

{ fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
obtain b' u where kuhn_simplex p n b' u t
using ⟨ksimplex p n t⟩ by (auto elim: ksimplex.cases)
then interpret t: kuhn_simplex p n b' u t .

{ fix x assume x ∈ s x ≠ a
then have x (upd n') = enum n' (upd n')
by (auto simp: ⟨a = enum i⟩ n' ⟨i = n⟩ s_eq_enum_def enum_inj
in_upd_image) }
then have eq_upd0: ∀ x ∈ t - {c}. x (upd n') = enum n' (upd n')
unfolding eq_sma[symmetric] by auto
then have c (upd n') ≠ enum n' (upd n')
using ⟨n ≠ 0⟩ by (intro t.one_step[OF ⟨c ∈ t⟩]) (auto simp: n' upd_space[unfolded
n'])
then have c (upd n') < enum n' (upd n') ∨ c (upd n') > enum n' (upd n')
by auto
then have t = s ∨ t = b.enum ' {..n}
proof (elim disjE conjE)
assume *: c (upd n') > enum n' (upd n')
interpret st: kuhn_simplex_pair p n base upd s b' u t ..
{ fix x assume x ∈ t with * ⟨c ∈ t⟩ eq_upd0[rule_format, of x] have x ≤ c
by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
note top = this
have s = t
using ⟨a = enum i⟩ ⟨i = n⟩ ⟨c ∈ t⟩
by (intro st.ksimplex_eq_top[OF _ _ _ _ eq_sma])
(auto simp: s_eq_enum_mono t.s_eq t.enum_mono top)
then show ?thesis by simp
}
next
assume *: c (upd n') < enum n' (upd n')
interpret st: kuhn_simplex_pair p n b upd ∘ rot f' ' {.. n} b' u t ..
have eq: f' ' {..n} - {b.enum 0} = t - {c}
using eq_sma eq f' by simp
{ fix x assume x ∈ t with * ⟨c ∈ t⟩ eq_upd0[rule_format, of x] have c ≤ x
by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
}

```



```

note bot = this
have f' ' {..n} = t
  using ⟨a = enum i⟩ ⟨i = n⟩ ⟨c ∈ t⟩
  by (intro st.ksimplex_eq_bot[OF _ _ _ _ eq])
      (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono bot)
with f' show ?thesis by simp
qed }
with ks_f' eq ⟨a ≠ b.enum 0⟩ ⟨n ≠ 0⟩ show ?thesis
apply (intro ex1I[of _ b.enum ' {.. n}])
apply fastforce
applymetis
done

next
assume i: 0 < i i < n
define i' where i' = i - 1
with i have Suc i' < n
  by simp
with i have Suc_i': Suc i' = i
  by (simp add: i'_def)

let ?upd = Fun.swap i' i upd
from i upd have bij_betw ?upd {..<n} {..<n}
  by (subst bij_betw_swap_iff) (auto simp: i'_def)

define f' where [abs_def]: f' i j = (if j ∈ ?upd' {..<i} then Suc (base j) else
base j)
for i j
interpret b: kuhn_simplex p n base ?upd f' ' {.. n}
proof
  show base ∈ {..<n} → {..<p} by (rule base)
  { fix i assume n ≤ i then show base i = p by (rule base_out) }
  show bij_betw ?upd {..<n} {..<n} by fact
qed (simp add: f'_def)
have f': b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
have ks_f': ksimplex p n (b.enum ' {.. n})
  unfolding f' by rule unfold_locales

have {i} ⊆ {..n}
  using i by auto
{ fix j assume j ≤ n
with i Suc_i' have enum j = b.enum j ↔ j ≠ i
  unfolding fun_eq_iff enum_def b.enum_def image_comp [symmetric]
  apply (cases ⟨i = j⟩)
  apply (metis imageI in_upd_image lessI lessThan_iff lessThan_subset_iff
order_less_le transpose_apply_first)
  by (metis lessThan_iff linorder_not_less not_less_eq_eq order_less_le
transpose_image_eq)
}
note enum_eq_benum = this

```

```

then have enum ' ( $\{.. n\} - \{i\}$ ) = b.enum ' ( $\{.. n\} - \{i\}$ )
  by (intro image_cong) auto
then have eq:  $s - \{a\} = b.enum ' \{.. n\} - \{b.enum i\}$ 
  unfolding s_eq ⟨ $a = enum i$ ⟩
  using inj_on_image_set_diff[OF inj_enum Diff_subset ⟨ $\{i\} \subseteq \{..n\}$ ⟩]
    inj_on_image_set_diff[OF b.inj_enum Diff_subset ⟨ $\{i\} \subseteq \{..n\}$ ⟩]
  by (simp add: comp_def)

have  $a \neq b.enum i$ 
  using ⟨ $a = enum i$ ⟩ enum_eq_benum i by auto

{ fix t c assume ksimplex p n t c  $\in t$  and eq_sma:  $s - \{a\} = t - \{c\}$ 
  obtain b' u where kuhn_simplex p n b' u t
    using ⟨ksimplex p n t⟩ by (auto elim: ksimplex.cases)
  then interpret t: kuhn_simplex p n b' u t .
  have enum  $i' \in s - \{a\}$  enum  $(i + 1) \in s - \{a\}$ 
    using ⟨ $a = enum i$ ⟩ i_enum_in by (auto simp: enum_inj i'_def)
  then obtain l k where
    l:  $t.enum l = enum i' l \leq n$  t.enum  $l \neq c$  and
    k:  $t.enum k = enum (i + 1) k \leq n$  t.enum  $k \neq c$ 
    unfolding eq_sma by (auto simp: t.s_eq)
  with i have  $t.enum l < t.enum k$ 
    by (simp add: enum_strict_mono i'_def)
  with ⟨ $l \leq n$ ⟩ ⟨ $k \leq n$ ⟩ have  $l < k$ 
    by (simp add: t.enum_strict_mono)
  { assume Suc l = k
    have enum (Suc (Suc i')) = t.enum (Suc l)
      using i by (simp add: k ⟨Suc l = k⟩ i'_def)
    then have False
      using ⟨ $l < k$ ⟩ ⟨ $k \leq n$ ⟩ ⟨Suc i' < n⟩
      by (auto simp: t.enum_Suc enum_Suc l upd_inj fun_eq_iff split:
if_split_asm)
      (metis Suc_lessD n_not_Suc_n upd_inj) }
  with ⟨ $l < k$ ⟩ have Suc l < k
    by arith
  have c_eq:  $c = t.enum (Suc l)$ 
  proof (rule ccontr)
    assume  $c \neq t.enum (Suc l)$ 
    then have  $t.enum (Suc l) \in s - \{a\}$ 
      using ⟨ $l < k$ ⟩ ⟨ $k \leq n$ ⟩ by (simp add: t.s_eq eq_sma)
    then obtain j where  $t.enum (Suc l) = enum j$   $j \leq n$   $enum j \neq enum i$ 
      unfolding s_eq ⟨ $a = enum i$ ⟩ by auto
    with i have  $t.enum (Suc l) \leq t.enum l \vee t.enum k \leq t.enum (Suc l)$ 
      by (auto simp: i'_def enum_mono enum_inj l k)
    with ⟨Suc l < k⟩ ⟨ $k \leq n$ ⟩ show False
      by (simp add: t.enum_mono)
  }
qed

{ have  $t.enum (Suc (Suc l)) \in s - \{a\}$ 

```

```

    unfolding eq_sma c_eq t.s_eq using ‹Suc l < k› ‹k ≤ n› by (auto simp:
t.enum_inj)
  then obtain j where eq: t.enum (Suc (Suc l)) = enum j and j ≤ n j ≠ i
    by (auto simp: s_eq ‹a = enum i›)
  moreover have enum i' < t.enum (Suc (Suc l))
    unfolding l(1)[symmetric] using ‹Suc l < k› ‹k ≤ n› by (auto simp:
t.enum_strict_mono)
  ultimately have i' < j
    using i by (simp add: enum_strict_mono i'_def)
  with ‹j ≠ i› ‹j ≤ n› have t.enum k ≤ t.enum (Suc (Suc l))
    unfolding i'_def by (simp add: enum_mono k eq)
  then have k ≤ Suc (Suc l)
    using ‹k ≤ n› ‹Suc l < k› by (simp add: t.enum_mono) }
  with ‹Suc l < k› have Suc (Suc l) = k by simp
  then have enum (Suc (Suc i')) = t.enum (Suc (Suc l))
    using i by (simp add: k i'_def)
  also have ... = (enum i') (u l := Suc (enum i' (u l)), u (Suc l) := Suc
(enum i' (u (Suc l))))
    using ‹Suc l < k› ‹k ≤ n› by (simp add: t.enum_Suc l t.upd_inj)
  finally have (u l = upd i' ∧ u (Suc l) = upd (Suc i')) ∨
    (u l = upd (Suc i') ∧ u (Suc l) = upd i')
    using ‹Suc i' < n› by (auto simp: enum_Suc fun_eq_iff split: if_split_asm)

  then have t = s ∨ t = b.enum ' {..n}
  proof (elim disjE conjE)
    assume u: u l = upd i'
    have c = t.enum (Suc l) unfolding c_eq ..
    also have t.enum (Suc l) = enum (Suc i')
    using u ‹l < k› ‹k ≤ n› ‹Suc i' < n› by (simp add: enum_Suc t.enum_Suc
l)

    also have ... = a
      using ‹a = enum i› i by (simp add: i'_def)
    finally show ?thesis
      using eq_sma ‹a ∈ s› ‹c ∈ t› by auto
  next
    assume u: u l = upd (Suc i')
    define B where B = b.enum ' {..n}
    have b.enum i' = enum i'
      using enum_eq_benum[of i'] i by (auto simp: i'_def gr0_conv_Suc)
    have c = t.enum (Suc l) unfolding c_eq ..
    also have t.enum (Suc l) = b.enum (Suc i')
      using u ‹l < k› ‹k ≤ n› ‹Suc i' < n›
      by (simp_all add: enum_Suc t.enum_Suc l b.enum_Suc ‹b.enum i' =
enum i'›)
      (simp add: Suc_i')
    also have ... = b.enum i
      using i by (simp add: i'_def)
    finally have c = b.enum i .
  then have t - {c} = B - {c} c ∈ B

```

```

      unfolding eq_sma[symmetric] eq B_def using i by auto
    with ⟨c ∈ t⟩ have t = B
      by auto
    then show ?thesis
      by (simp add: B_def)
  qed }
with ks_f' eq ⟨a ≠ b.enum i⟩ ⟨n ≠ 0⟩ ⟨i ≤ n⟩ show ?thesis
  apply (intro ex1I[of _ b.enum ' {.. n}])
  apply auto []
  apply metis
  done
qed
then show ?thesis
  using s ⟨a ∈ s⟩ by (simp add: card_2_iff' Ex1_def) metis
qed

```

Hence another step towards concreteness.

lemma *kuhn_simplex_lemma*:

```

  assumes ∀ s. ksimplex p (Suc n) s ⟶ rl ' s ⊆ {.. Suc n}
    and odd (card {f. ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ (f = s - {a}) ∧
      rl ' f = {..n} ∧ ((∃ j ≤ n. ∀ x ∈ f. x j = 0) ∨ (∃ j ≤ n. ∀ x ∈ f. x j = p))})
  shows odd (card {s. ksimplex p (Suc n) s ∧ rl ' s = {..Suc n}})
proof (rule kuhn_complete_lemma[OF finite_ksimplexes refl, unfolded mem_Collect_eq,
  where bnd=λf. (∃ j ∈ {..n}. ∀ x ∈ f. x j = 0) ∨ (∃ j ∈ {..n}. ∀ x ∈ f. x j = p)],
  safe del: notI)

```

```

  have *: ∧ x y. x = y ⟹ odd (card x) ⟹ odd (card y)
    by auto
  show odd (card {f. (∃ s ∈ {s. ksimplex p (Suc n) s}. ∃ a ∈ s. f = s - {a}) ∧
    rl ' f = {..n} ∧ ((∃ j ∈ {..n}. ∀ x ∈ f. x j = 0) ∨ (∃ j ∈ {..n}. ∀ x ∈ f. x j = p))})
    apply (rule *[OF _ assms(2)])
    apply (auto simp: atLeast0AtMost)
  done

```

next

```

fix s assume s: ksimplex p (Suc n) s
then show card s = n + 2
  by (simp add: ksimplex_card)

fix a assume a: a ∈ s then show rl a ≤ Suc n
  using assms(1) s by (auto simp: subset_eq)

```

```

let ?S = {t. ksimplex p (Suc n) t ∧ (∃ b ∈ t. s - {a} = t - {b})}
{ fix j assume j: j ≤ n ∀ x ∈ s - {a}. x j = 0
  with s a show card ?S = 1
    using ksimplex_replace_0[of p n + 1 s a j]
    by (subst eq_commute) simp }

```

```

{ fix j assume j: j ≤ n ∀ x ∈ s - {a}. x j = p
  with s a show card ?S = 1
    using ksimplex_replace_1[of p n + 1 s a j]
    by (subst eq_commute) simp }

{ assume card ?S ≠ 2 ¬ (∃ j ∈ {..n}. ∀ x ∈ s - {a}. x j = p)
  with s a show ∃ j ∈ {..n}. ∀ x ∈ s - {a}. x j = 0
    using ksimplex_replace_2[of p n + 1 s a]
    by (subst (asm) eq_commute) auto }
qed

```

Reduced labelling

definition *reduced* :: $\text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat}$ **where** *reduced* $n\ x = (\text{LEAST } k. k = n \vee x\ k \neq 0)$

lemma *reduced_labelling*:

shows $\text{reduced } n\ x \leq n$
and $\forall i < \text{reduced } n\ x. x\ i = 0$
and $\text{reduced } n\ x = n \vee x (\text{reduced } n\ x) \neq 0$

proof –

show $\text{reduced } n\ x \leq n$
unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *auto*
show $\forall i < \text{reduced } n\ x. x\ i = 0$
unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *fastforce*+
show $\text{reduced } n\ x = n \vee x (\text{reduced } n\ x) \neq 0$
unfolding *reduced_def* **by** (rule *LeastI2_wellorder*[**where** $a=n$]) *fastforce*+
qed

lemma *reduced_labelling_unique*:

$r \leq n \implies \forall i < r. x\ i = 0 \implies r = n \vee x\ r \neq 0 \implies \text{reduced } n\ x = r$
by (*metis linorder_less_linear linorder_not_le reduced_labelling*)

lemma *reduced_labelling_zero*: $j < n \implies x\ j = 0 \implies \text{reduced } n\ x \neq j$

using *reduced_labelling*[of $n\ x$] **by** *auto*

lemma *reduce_labelling_zero*[*simp*]: $\text{reduced } 0\ x = 0$

by (rule *reduced_labelling_unique*) *auto*

lemma *reduced_labelling_nonzero*: $j < n \implies x\ j \neq 0 \implies \text{reduced } n\ x \leq j$

using *reduced_labelling*[of $n\ x$] **by** (*elim allE*[**where** $x=j$]) *auto*

lemma *reduced_labelling_Suc*: $\text{reduced } (\text{Suc } n)\ x \neq \text{Suc } n \implies \text{reduced } (\text{Suc } n)\ x = \text{reduced } n\ x$

using *reduced_labelling*[of $\text{Suc } n\ x$]
by (*intro reduced_labelling_unique*[*symmetric*]) *auto*

lemma *complete_face_top*:

assumes $\forall x \in f. \forall j \leq n. x\ j = 0 \longrightarrow \text{lab } x\ j = 0$

```

    and  $\forall x \in f. \forall j \leq n. x j = p \longrightarrow \text{lab } x j = 1$ 
    and  $\text{eq}: (\text{reduced } (\text{Suc } n) \circ \text{lab}) ' f = \{..n\}$ 
  shows  $((\exists j \leq n. \forall x \in f. x j = 0) \vee (\exists j \leq n. \forall x \in f. x j = p)) \longleftrightarrow (\forall x \in f. x n = p)$ 
proof (safe del: disjCI)
  fix  $x j$  assume  $j: j \leq n \forall x \in f. x j = 0$ 
  { fix  $x$  assume  $x \in f$  with assms  $j$  have  $\text{reduced } (\text{Suc } n) (\text{lab } x) \neq j$ 
    by (intro reduced_labelling_zero) auto }
  moreover have  $j \in (\text{reduced } (\text{Suc } n) \circ \text{lab}) ' f$ 
    using  $j \text{ eq}$  by auto
  ultimately show  $x n = p$ 
    by force
next
fix  $x j$  assume  $j: j \leq n \forall x \in f. x j = p$  and  $x: x \in f$ 
have  $j = n$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  { fix  $x$  assume  $x \in f$ 
    with assms  $j$  have  $\text{reduced } (\text{Suc } n) (\text{lab } x) \leq j$ 
      by (intro reduced_labelling_nonzero) auto
    then have  $\text{reduced } (\text{Suc } n) (\text{lab } x) \neq n$ 
      using  $\langle j \neq n \rangle \langle j \leq n \rangle$  by simp }
  moreover
have  $n \in (\text{reduced } (\text{Suc } n) \circ \text{lab}) ' f$ 
    using  $\text{eq}$  by auto
  ultimately show False
    by force
qed
moreover have  $j \in (\text{reduced } (\text{Suc } n) \circ \text{lab}) ' f$ 
  using  $j \text{ eq}$  by auto
ultimately show  $x n = p$ 
  using  $j x$  by auto
qed auto

```

Hence we get just about the nice induction.

lemma *kuhn_induction*:

```

assumes  $0 < p$ 
  and  $\text{lab}_0: \forall x. \forall j \leq n. (\forall j. x j \leq p) \wedge x j = 0 \longrightarrow \text{lab } x j = 0$ 
  and  $\text{lab}_1: \forall x. \forall j \leq n. (\forall j. x j \leq p) \wedge x j = p \longrightarrow \text{lab } x j = 1$ 
  and  $\text{odd}: \text{odd } (\text{card } \{s. \text{ksimplex } p \ n \ s \wedge (\text{reduced } n \circ \text{lab}) ' s = \{..n\}\})$ 
shows  $\text{odd } (\text{card } \{s. \text{ksimplex } p \ (\text{Suc } n) \ s \wedge (\text{reduced } (\text{Suc } n) \circ \text{lab}) ' s = \{.. \text{Suc } n\}\})$ 
proof –
  let  $?rl = \text{reduced } (\text{Suc } n) \circ \text{lab}$  and  $?ext = \lambda f v. \exists j \leq n. \forall x \in f. x j = v$ 
  let  $?ext = \lambda s. (\exists j \leq n. \forall x \in s. x j = 0) \vee (\exists j \leq n. \forall x \in s. x j = p)$ 
  have  $\forall s. \text{ksimplex } p \ (\text{Suc } n) \ s \longrightarrow ?rl ' s \subseteq \{.. \text{Suc } n\}$ 
    by (simp add: reduced_labelling_subset_eq)
  moreover
  have  $\{s. \text{ksimplex } p \ n \ s \wedge (\text{reduced } n \circ \text{lab}) ' s = \{..n\}\} =$ 
     $\{f. \exists s a. \text{ksimplex } p \ (\text{Suc } n) \ s \wedge a \in s \wedge f = s - \{a\} \wedge ?rl ' f = \{..n\} \wedge$ 

```

```

?ext f}
proof (intro set_eqI, safe del: disjCI equalityI disjE)
  fix s assume s: ksimplex p n s and rl: (reduced n o lab) ' s = {..n}
from s obtain u b where kuhn_simplex p n u b s by (auto elim: ksimplex.cases)
then interpret kuhn_simplex p n u b s .
have all_eq_p:  $\forall x \in s. x n = p$ 
  by (auto simp: out_eq_p)
moreover
  { fix x assume x  $\in$  s
    with lab_1[rule_format, of n x] all_eq_p s_le_p[of x]
    have ?rl x  $\leq$  n
      by (auto intro!: reduced_labelling_nonzero)
    then have ?rl x = reduced n (lab x)
      by (auto intro!: reduced_labelling_Suc) }
then have ?rl ' s = {..n}
  using rl by (simp cong: image_cong)
moreover
obtain t a where ksimplex p (Suc n) t a  $\in$  t s = t - {a}
  using s unfolding simplex_top_face[OF <0 < p> all_eq_p] by auto
ultimately
show  $\exists t a. ksimplex p (Suc n) t \wedge a \in t \wedge s = t - \{a\} \wedge ?rl ' s = \{..n\} \wedge$ 
?ext s
  by auto
next
fix x s a assume s: ksimplex p (Suc n) s and rl: ?rl ' (s - {a}) = {.. n}
  and a: a  $\in$  s and ?ext (s - {a})
  from s obtain u b where kuhn_simplex p (Suc n) u b s by (auto elim:
ksimplex.cases)
then interpret kuhn_simplex p Suc n u b s .
have all_eq_p:  $\forall x \in s. x (Suc n) = p$ 
  by (auto simp: out_eq_p)

  { fix x assume x  $\in$  s - {a}
    then have ?rl x  $\in$  ?rl ' (s - {a})
      by auto
    then have ?rl x  $\leq$  n
      unfolding rl by auto
    then have ?rl x = reduced n (lab x)
      by (auto intro!: reduced_labelling_Suc) }
then show rl': (reduced n o lab) ' (s - {a}) = {..n}
  unfolding rl[symmetric] by (intro image_cong) auto

from <?ext (s - {a})>
have all_eq_p:  $\forall x \in s - \{a\}. x n = p$ 
proof (elim disjE exE conjE)
  fix j assume j  $\leq$  n  $\forall x \in s - \{a\}. x j = 0$ 
  with lab_0[rule_format, of j] all_eq_p s_le_p
  have  $\bigwedge x. x \in s - \{a\} \implies reduced (Suc n) (lab x) \neq j$ 
    by (intro reduced_labelling_zero) auto

```

```

moreover have  $j \in ?rl \text{ ' } (s - \{a\})$ 
  using  $\langle j \leq n \rangle$  unfolding  $rl$  by auto
ultimately show ?thesis
  by force
next
fix  $j$  assume  $j \leq n$  and  $eq\_p: \forall x \in s - \{a\}. x j = p$ 
show ?thesis
proof cases
  assume  $j = n$  with  $eq\_p$  show ?thesis by simp
next
assume  $j \neq n$ 
  { fix  $x$  assume  $x \in s - \{a\}$ 
    have  $reduced\ n\ (lab\ x) \leq j$ 
    proof (rule reduced_labelling_nonzero)
      show  $lab\ x\ j \neq 0$ 
      using  $lab\_1[rule\_format, of\ j\ x]\ x\ s\_le\_p[of\ x]\ eq\_p\ \langle j \leq n \rangle$  by auto
      show  $j < n$ 
      using  $\langle j \leq n \rangle\ \langle j \neq n \rangle$  by simp
    }
  qed
  then have  $reduced\ n\ (lab\ x) \neq n$ 
    using  $\langle j \leq n \rangle\ \langle j \neq n \rangle$  by simp }
  moreover have  $n \in (reduced\ n\ o\ lab) \text{ ' } (s - \{a\})$ 
    unfolding  $rl'$  by auto
  ultimately show ?thesis
    by force
  qed
qed
show  $ksimplex\ p\ n\ (s - \{a\})$ 
  unfolding  $simplex\_top\_face[OF\ \langle 0 < p \rangle\ all\_eq\_p]$  using  $s\ a$  by auto
qed
ultimately show ?thesis
  using assms by (intro kuhn_simplex_lemma) auto
qed

```

And so we get the final combinatorial result.

```

lemma  $ksimplex\_0: ksimplex\ p\ 0\ s \longleftrightarrow s = \{(\lambda x. p)\}$ 
proof
  assume  $ksimplex\ p\ 0\ s$  then show  $s = \{(\lambda x. p)\}$ 
    by (blast dest: kuhn_simplex.ksimplex_0 elim: ksimplex.cases)
next
  assume  $s = \{(\lambda x. p)\}$ 
  show  $ksimplex\ p\ 0\ s$ 
  proof (intro ksimplex, unfold_locales)
    show  $(\lambda \_. p) \in \{..\langle 0::nat \rangle\} \rightarrow \{..\langle p \rangle\}$  by auto
    show  $bij\_betw\ id\ \{..\langle 0 \rangle\}\ \{..\langle 0 \rangle\}$ 
      by simp
    qed (auto simp: s)
  qed

```


lemma *kuhn_combinatorial*:

```

assumes  $0 < p$ 
  and  $\forall x j. (\forall j. x j \leq p) \wedge j < n \wedge x j = 0 \longrightarrow \text{lab } x j = 0$ 
  and  $\forall x j. (\forall j. x j \leq p) \wedge j < n \wedge x j = p \longrightarrow \text{lab } x j = 1$ 
shows  $\text{odd} (\text{card } \{s. \text{ksimplex } p \ n \ s \wedge (\text{reduced } n \circ \text{lab}) \ 's = \{..n\}\})$ 
  (is odd ( $\text{card } (?M \ n)$ ))
using assms
proof (induct n)
  case 0 then show ?case
    by (simp add: ksimplex_0 cong: conj_cong)
next
  case (Suc n)
  then have  $\text{odd} (\text{card } (?M \ n))$ 
    by force
  with Suc show ?case
    using kuhn_induction[of p n] by (auto simp: comp_def)
qed

```

lemma *kuhn_lemma*:

```

fixes  $n \ p :: \text{nat}$ 
assumes  $0 < p$ 
  and  $\forall x. (\forall i < n. x \ i \leq p) \longrightarrow (\forall i < n. \text{label } x \ i = (0 :: \text{nat}) \vee \text{label } x \ i = 1)$ 
  and  $\forall x. (\forall i < n. x \ i \leq p) \longrightarrow (\forall i < n. x \ i = 0 \longrightarrow \text{label } x \ i = 0)$ 
  and  $\forall x. (\forall i < n. x \ i \leq p) \longrightarrow (\forall i < n. x \ i = p \longrightarrow \text{label } x \ i = 1)$ 
obtains  $q$  where  $\forall i < n. q \ i < p$ 
  and  $\forall i < n. \exists r \ s. (\forall j < n. q \ j \leq r \ j \wedge r \ j \leq q \ j + 1) \wedge (\forall j < n. q \ j \leq s \ j \wedge s \ j$ 
 $\leq q \ j + 1) \wedge \text{label } r \ i \neq \text{label } s \ i$ 
proof –
  let ?rl = reduced n o label
  let ?A =  $\{s. \text{ksimplex } p \ n \ s \wedge ?rl \ 's = \{..n\}\}$ 
  have  $\text{odd} (\text{card } ?A)$ 
    using assms by (intro kuhn_combinatorial[of p n label]) auto
  then have  $?A \neq \{\}$ 
    by (rule odd_card_imp_not_empty)
  then obtain  $s \ b \ u$  where kuhn_simplex p n b u s and  $rl: ?rl \ 's = \{..n\}$ 
    by (auto elim: ksimplex.cases)
  interpret kuhn_simplex p n b u s by fact

  show ?thesis
proof (intro that[of b] allI impI)
  fix  $i$ 
  assume  $i < n$ 
  then show  $b \ i < p$ 
    using base by auto
next
  fix  $i$ 
  assume  $i < n$ 
  then have  $i \in \{.. n\} \ \text{Suc } i \in \{.. n\}$ 
    by auto

```

then obtain $u\ v$ **where** $u: u \in s\ Suc\ i = ?rl\ u$ **and** $v: v \in s\ i = ?rl\ v$
unfolding $rl[symmetric]$ **by** $blast$

have $label\ u\ i \neq label\ v\ i$
using $reduced_labelling\ [of\ n\ label\ u]\ reduced_labelling\ [of\ n\ label\ v]$
 $u(2)[symmetric]\ v(2)[symmetric]\ \langle i < n \rangle$
by $auto$

moreover
have $b\ j \leq u\ j\ u\ j \leq b\ j + 1\ b\ j \leq v\ j\ v\ j \leq b\ j + 1$ **if** $j < n$ **for** j
using $that\ base_le[OF\ \langle u \in s \rangle]\ le_Suc_base[OF\ \langle u \in s \rangle]\ base_le[OF\ \langle v \in s \rangle]$
 $le_Suc_base[OF\ \langle v \in s \rangle]$
by $auto$

ultimately show $\exists r\ s. (\forall j < n. b\ j \leq r\ j \wedge r\ j \leq b\ j + 1) \wedge$
 $(\forall j < n. b\ j \leq s\ j \wedge s\ j \leq b\ j + 1) \wedge label\ r\ i \neq label\ s\ i$
by $blast$

qed
qed

Main result for the unit cube

lemma $kuhn_labelling_lemma'$:

assumes $(\forall x::nat \Rightarrow real. P\ x \longrightarrow P\ (f\ x))$
and $\forall x. P\ x \longrightarrow (\forall i::nat. Q\ i \longrightarrow 0 \leq x\ i \wedge x\ i \leq 1)$
shows $\exists l. (\forall x\ i. l\ x\ i \leq (1::nat)) \wedge$
 $(\forall x\ i. P\ x \wedge Q\ i \wedge x\ i = 0 \longrightarrow l\ x\ i = 0) \wedge$
 $(\forall x\ i. P\ x \wedge Q\ i \wedge x\ i = 1 \longrightarrow l\ x\ i = 1) \wedge$
 $(\forall x\ i. P\ x \wedge Q\ i \wedge l\ x\ i = 0 \longrightarrow x\ i \leq f\ x\ i) \wedge$
 $(\forall x\ i. P\ x \wedge Q\ i \wedge l\ x\ i = 1 \longrightarrow f\ x\ i \leq x\ i)$
unfolding $all_conj_distrib\ [symmetric]$
apply $(subst\ choice_iff[symmetric])+$
by $(metis\ assms\ choice_iff\ bot_nat_0.extremum\ nle_le\ zero_neq_one)$

9.18.3 Brouwer's fixed point theorem

We start proving Brouwer's fixed point theorem for the unit cube = $cbox\ 0\ One$.
One.

lemma $brouwer_cube$:

fixes $f :: 'a::euclidean_space \Rightarrow 'a$
assumes $continuous_on\ (cbox\ 0\ One)\ f$
and $f\ 'cbox\ 0\ One \subseteq cbox\ 0\ One$
shows $\exists x \in cbox\ 0\ One. f\ x = x$
proof $(rule\ ccontr)$
define n **where** $n = DIM('a)$
have $n: 1 \leq n\ 0 < n\ n \neq 0$
unfolding n_def **by** $(auto\ simp: Suc_le_eq)$
assume $\neg\ ?thesis$
then have $*: \neg (\exists x \in cbox\ 0\ One. f\ x - x = 0)$
by $auto$
obtain d **where**

```

    d: d > 0  $\wedge$  x. x  $\in$  cbox 0 One  $\implies$  d  $\leq$  norm (f x - x)
  using brouwer_compactness_lemma[OF compact_cbox_*] assms
  by (metis (no_types, lifting) continuous_on_cong continuous_on_diff continuous_on_id)
  have *:  $\forall$  x. x  $\in$  cbox 0 One  $\longrightarrow$  f x  $\in$  cbox 0 One
     $\forall$  x. x  $\in$  (cbox 0 One::'a set)  $\longrightarrow$  ( $\forall$  i $\in$ Basis. True  $\longrightarrow$  0  $\leq$  x  $\cdot$  i  $\wedge$  x  $\cdot$  i  $\leq$  1)
  using assms(2)[unfolded image_subset_iff Ball_def]
  unfolding cbox_def
  by auto
  obtain label :: 'a  $\Rightarrow$  'a  $\Rightarrow$  nat where label [rule_format]:
     $\forall$  x.  $\forall$  i $\in$ Basis. label x i  $\leq$  1
     $\forall$  x.  $\forall$  i $\in$ Basis. x  $\in$  cbox 0 One  $\wedge$  x  $\cdot$  i = 0  $\longrightarrow$  label x i = 0
     $\forall$  x.  $\forall$  i $\in$ Basis. x  $\in$  cbox 0 One  $\wedge$  x  $\cdot$  i = 1  $\longrightarrow$  label x i = 1
     $\forall$  x.  $\forall$  i $\in$ Basis. x  $\in$  cbox 0 One  $\wedge$  label x i = 0  $\longrightarrow$  x  $\cdot$  i  $\leq$  f x  $\cdot$  i
     $\forall$  x.  $\forall$  i $\in$ Basis. x  $\in$  cbox 0 One  $\wedge$  label x i = 1  $\longrightarrow$  f x  $\cdot$  i  $\leq$  x  $\cdot$  i
  using kuhn_labelling_lemma[OF*] by auto
  note label = this [rule_format]
  have lem1:  $\forall$  x $\in$ cbox 0 One.  $\forall$  y $\in$ cbox 0 One.  $\forall$  i $\in$ Basis. label x i  $\neq$  label y i  $\longrightarrow$ 
    |f x  $\cdot$  i - x  $\cdot$  i|  $\leq$  norm (f y - f x) + norm (y - x)
  proof safe
    fix x y :: 'a
    assume x: x  $\in$  cbox 0 One and y: y  $\in$  cbox 0 One
    fix i
    assume i: label x i  $\neq$  label y i i  $\in$  Basis
    have *:  $\wedge$  x y fx fy :: real. x  $\leq$  fx  $\wedge$  fy  $\leq$  y  $\vee$  fx  $\leq$  x  $\wedge$  y  $\leq$  fy  $\implies$ 
      |fx - x|  $\leq$  |fy - fx| + |y - x| by auto
    have |(f x - x)  $\cdot$  i|  $\leq$  |(f y - f x)  $\cdot$  i| + |(y - x)  $\cdot$  i|
    proof (cases label x i = 0)
      case True
      then have fxy:  $\neg$  f y  $\cdot$  i  $\leq$  y  $\cdot$  i  $\implies$  f x  $\cdot$  i  $\leq$  x  $\cdot$  i
        by (metis True i label(1) label(5) le_antisym less_one not_le_imp_less y)
      show ?thesis
      unfolding inner_simps
      by (rule *) (auto simp: True i label x y fxy)
    next
      case False
      then show ?thesis
      using label [OF <i  $\in$  Basis>] i(1) x y
      by (smt (verit, ccfv_threshold) inner_diff_left less_one order_le_less)
    qed
    also have ...  $\leq$  norm (f y - f x) + norm (y - x)
      by (simp add: add_mono i(2) norm_bound_Basis_le)
    finally show |f x  $\cdot$  i - x  $\cdot$  i|  $\leq$  norm (f y - f x) + norm (y - x)
      unfolding inner_simps .
  qed
  qed
  have  $\exists e > 0$ .  $\forall$  x $\in$ cbox 0 One.  $\forall$  y $\in$ cbox 0 One.  $\forall$  z $\in$ cbox 0 One.  $\forall$  i $\in$ Basis.
    norm (x - z) < e  $\longrightarrow$  norm (y - z) < e  $\longrightarrow$  label x i  $\neq$  label y i  $\longrightarrow$ 
    |(f(z) - z)  $\cdot$  i| < d / (real n)
  proof -

```

```

have d': d / real n / 8 > 0
  using d(1) by (simp add: n_def)
have *: uniformly_continuous_on (cbox 0 One) f
  by (rule compact_uniformly_continuous[OF assms(1) compact_cbox])
obtain e where e:
  e > 0
   $\bigwedge x x'. x \in \text{cbox } 0 \text{ One} \implies$ 
   $x' \in \text{cbox } 0 \text{ One} \implies$ 
   $\text{norm } (x' - x) < e \implies$ 
   $\text{norm } (f x' - f x) < d / \text{real } n / 8$ 
  using *[unfolded uniformly_continuous_on_def, rule_format, OF d']
  unfolding dist_norm
  by blast
show ?thesis
proof (intro exI conjI ballI impI)
  show 0 < min (e / 2) (d / real n / 8)
    using d' e by auto
  fix x y z i
  assume as:
    x  $\in$  cbox 0 One y  $\in$  cbox 0 One z  $\in$  cbox 0 One
    norm (x - z) < min (e / 2) (d / real n / 8)
    norm (y - z) < min (e / 2) (d / real n / 8)
    label x i  $\neq$  label y i
  assume i: i  $\in$  Basis
  have *:  $\bigwedge z fz x fx n1 n2 n3 n4 d4 d :: \text{real}. |fx - x| \leq n1 + n2 \implies$ 
     $|fz - fz| \leq n3 \implies |x - z| \leq n4 \implies$ 
     $n1 < d4 \implies n2 < 2 * d4 \implies n3 < d4 \implies n4 < d4 \implies$ 
     $(8 * d4 = d) \implies |fz - z| < d$ 
    by auto
  show |(f z - z)  $\cdot$  i| < d / real n
    unfolding inner_simps
  proof (rule *)
    show |f x  $\cdot$  i - x  $\cdot$  i|  $\leq$  norm (f y - f x) + norm (y - x)
      using as(1) as(2) as(6) i lem1 by blast
    show norm (f x - f z) < d / real n / 8
      using d' e as by auto
    show |f x  $\cdot$  i - f z  $\cdot$  i|  $\leq$  norm (f x - f z) |x  $\cdot$  i - z  $\cdot$  i|  $\leq$  norm (x - z)
      unfolding inner_diff_left[symmetric]
      by (rule Basis_le_norm[OF i])+
    have tria: norm (y - x)  $\leq$  norm (y - z) + norm (x - z)
      using dist_triangle[of y x z, unfolded dist_norm]
      unfolding norm_minus_commute
      by auto
    also have ... < e / 2 + e / 2
      using as(4) as(5) by auto
    finally show norm (f y - f x) < d / real n / 8
      using as(1) as(2) e(2) by auto
  have norm (y - z) + norm (x - z) < d / real n / 8 + d / real n / 8
    using as(4) as(5) by auto

```

```

    with tria show norm (y - x) < 2 * (d / real n / 8)
    by auto
  qed (use as in auto)
qed
qed
then
obtain e where e:
  e > 0
   $\bigwedge x y z i. x \in \text{cbox } 0 \text{ One} \implies$ 
   $y \in \text{cbox } 0 \text{ One} \implies$ 
   $z \in \text{cbox } 0 \text{ One} \implies$ 
   $i \in \text{Basis} \implies$ 
   $\text{norm } (x - z) < e \wedge \text{norm } (y - z) < e \wedge \text{label } x \ i \neq \text{label } y \ i \implies$ 
   $|(f z - z) \cdot i| < d / \text{real } n$ 
  by blast
obtain p :: nat where p: 1 + real n / e  $\leq$  real p
  using real_arch_simple ..
have 1 + real n / e > 0
  using e(1) n by (simp add: add_pos_pos)
then have p > 0
  using p by auto

obtain b :: nat  $\implies$  'a where b: bij_betw b {.. $n$ } Basis
  by atomize_elim (auto simp: n_def intro!: finite_same_card_bij)
define b' where b' = inv_into {.. $n$ } b
then have b': bij_betw b' Basis {.. $n$ }
  using bij_betw_inv_into[OF b] by auto
then have b'_Basis:  $\bigwedge i. i \in \text{Basis} \implies b' \ i \in \{.. $n$ \}$ 
  unfolding bij_betw_def by (auto simp: set_eq_iff)
have bb'[simp]:  $\bigwedge i. i \in \text{Basis} \implies b \ (b' \ i) = i$ 
  unfolding b'_def
  using b
  by (auto simp: f_inv_into_f bij_betw_def)
have b'b[simp]:  $\bigwedge i. i < n \implies b' \ (b \ i) = i$ 
  unfolding b'_def
  using b
  by (auto simp: inv_into_f_eq bij_betw_def)
have *:  $\bigwedge x :: \text{nat}. x = 0 \vee x = 1 \iff x \leq 1$ 
  by auto
have b'':  $\bigwedge j. j < n \implies b \ j \in \text{Basis}$ 
  using b unfolding bij_betw_def by auto
have q1:  $0 < p \ \forall x. (\forall i < n. x \ i \leq p) \longrightarrow$ 
   $(\forall i < n. (\text{label } (\sum_{i \in \text{Basis}} (\text{real } (x \ (b' \ i)) / \text{real } p) *_R \ i) \circ b) \ i = 0 \vee$ 
   $(\text{label } (\sum_{i \in \text{Basis}} (\text{real } (x \ (b' \ i)) / \text{real } p) *_R \ i) \circ b) \ i = 1)$ 
  unfolding *
  using  $\langle p > 0 \rangle \langle n > 0 \rangle$ 
  using label(1)[OF b'']
  by auto
{ fix x :: nat  $\implies$  nat and i assume  $\forall i < n. x \ i \leq p \ i < n \ x \ i = p \vee x \ i = 0$ 

```

```

then have (∑ i∈Basis. (real (x (b' i)) / real p) *R i) ∈ (cbox 0 One::'a set)
  using b'_Basis
  by (auto simp: cbox_def bij_betw_def zero_le_divide_iff divide_le_eq_1) }
note cube = this
have q2: ∀ x. (∀ i<n. x i ≤ p) → (∀ i<n. x i = 0 →
  (label (∑ i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 0)
  unfolding o_def using cube <p > 0> by (intro allI impI label(2)) (auto simp:
b'')
have q3: ∀ x. (∀ i<n. x i ≤ p) → (∀ i<n. x i = p →
  (label (∑ i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 1)
  using cube <p > 0> unfolding o_def by (intro allI impI label(3)) (auto simp:
b'')
obtain q where q:
  ∀ i<n. q i < p
  ∀ i<n.
    ∃ r s. (∀ j<n. q j ≤ r j ∧ r j ≤ q j + 1) ∧
      (∀ j<n. q j ≤ s j ∧ s j ≤ q j + 1) ∧
      (label (∑ i∈Basis. (real (r (b' i)) / real p) *R i) ∘ b) i ≠
      (label (∑ i∈Basis. (real (s (b' i)) / real p) *R i) ∘ b) i
  by (rule kuhn_lemma[OF q1 q2 q3])
define z :: 'a where z = (∑ i∈Basis. (real (q (b' i)) / real p) *R i)
have ∃ i∈Basis. d / real n ≤ |(f z - z)·i|
proof (rule ccontr)
  have ∀ i∈Basis. q (b' i) ∈ {0..p}
    using q(1) b'
    by (auto intro: less_imp_le simp: bij_betw_def)
  then have z ∈ cbox 0 One
    unfolding z_def cbox_def
    using b'_Basis
    by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
  then have d_fz_z: d ≤ norm (f z - z)
    by (rule d)
  assume ¬ ?thesis
  then have as: ∀ i∈Basis. |f z · i - z · i| < d / real n
    using <n > 0>
    by (auto simp: not_le inner_diff)
  have norm (f z - z) ≤ (∑ i∈Basis. |f z · i - z · i|)
    unfolding inner_diff_left[symmetric]
    by (rule norm_le_l1)
  also have ... < (∑ (i::'a) ∈ Basis. d / real n)
    by (meson as finite_Basis nonempty_Basis sum_strict_mono)
  also have ... = d
    using DIM_positive[where 'a='a] by (auto simp: n_def)
  finally show False
    using d_fz_z by auto
qed
then obtain i where i: i ∈ Basis d / real n ≤ |(f z - z) · i| ..
have *: b' i < n
  using i and b'[unfolded bij_betw_def]

```

```

by auto
obtain r s where rs:
   $\bigwedge j. j < n \implies q j \leq r j \wedge r j \leq q j + 1$ 
   $\bigwedge j. j < n \implies q j \leq s j \wedge s j \leq q j + 1$ 
  (label  $(\sum_{i \in \text{Basis.}} (\text{real } (r (b' i)) / \text{real } p) *_{\mathbb{R}} i) \circ b) (b' i) \neq$ 
    (label  $(\sum_{i \in \text{Basis.}} (\text{real } (s (b' i)) / \text{real } p) *_{\mathbb{R}} i) \circ b) (b' i)$ 
  using q(2)[rule_format, OF *] by blast
have b'_im:  $\bigwedge i. i \in \text{Basis} \implies b' i < n$ 
  using b' unfolding bij_betw_def by auto
define r' :: 'a where r' =  $(\sum_{i \in \text{Basis.}} (\text{real } (r (b' i)) / \text{real } p) *_{\mathbb{R}} i)$ 
have  $\bigwedge i. i \in \text{Basis} \implies r (b' i) \leq p$ 
  using b'_im q(1) rs(1) by fastforce
then have r'  $\in \text{cbox } 0 \text{ One}$ 
  unfolding r'_def cbox_def
  using b'_Basis
  by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
define s' :: 'a where s' =  $(\sum_{i \in \text{Basis.}} (\text{real } (s (b' i)) / \text{real } p) *_{\mathbb{R}} i)$ 
have  $\bigwedge i. i \in \text{Basis} \implies s (b' i) \leq p$ 
  using b'_im q(1) rs(2) by fastforce
then have s'  $\in \text{cbox } 0 \text{ One}$ 
  unfolding s'_def cbox_def
  using b'_Basis by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
have z  $\in \text{cbox } 0 \text{ One}$ 
  unfolding z_def cbox_def
  using b'_Basis q(1)[rule_format, OF b'_im]  $\langle p > 0 \rangle$ 
  by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1 less_imp_le)
{
  have  $(\sum_{i \in \text{Basis.}} |\text{real } (r (b' i)) - \text{real } (q (b' i))|) \leq (\sum_{(i::'a) \in \text{Basis.}} 1)$ 
    by (rule sum_mono) (use rs(1)[OF b'_im] in force)
  also have  $\dots < e * \text{real } p$ 
    using p  $\langle e > 0 \rangle \langle p > 0 \rangle$ 
    by (auto simp: field_simps n_def)
  finally have  $(\sum_{i \in \text{Basis.}} |\text{real } (r (b' i)) - \text{real } (q (b' i))|) < e * \text{real } p .$ 
}
}
moreover
{
  have  $(\sum_{i \in \text{Basis.}} |\text{real } (s (b' i)) - \text{real } (q (b' i))|) \leq (\sum_{(i::'a) \in \text{Basis.}} 1)$ 
    by (rule sum_mono) (use rs(2)[OF b'_im] in force)
  also have  $\dots < e * \text{real } p$ 
    using p  $\langle e > 0 \rangle \langle p > 0 \rangle$ 
    by (auto simp: field_simps n_def)
  finally have  $(\sum_{i \in \text{Basis.}} |\text{real } (s (b' i)) - \text{real } (q (b' i))|) < e * \text{real } p .$ 
}
}
ultimately
have norm (r' - z) < e and norm (s' - z) < e
  unfolding r'_def s'_def z_def
  using  $\langle p > 0 \rangle$ 
  apply (rule_tac[!] le_less_trans[OF norm_le_l1])
  apply (auto simp: field_simps sum_divide_distrib[symmetric] inner_diff_left)

```

```

done
then have |(f z - z) · i| < d / real n
  using rs(3) i
  unfolding r'_def[symmetric] s'_def[symmetric] o_def bb'
  by (intro e(2)[OF ‹r'∈cbox 0 One› ‹s'∈cbox 0 One› ‹z∈cbox 0 One›]) auto
then show False
  using i by auto
qed

```

Next step is to prove it for nonempty interiors.

```

lemma brouwer_weak:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes compact S
    and convex S
    and interior S ≠ {}
    and continuous_on S f
    and f ∈ S → S
  obtains x where x ∈ S and f x = x
proof -
  let ?U = cbox 0 One :: 'a set
  have ∑ Basis /R 2 ∈ interior ?U
  proof (rule interiorI)
    let ?I = (∩ i∈Basis. {x::'a. 0 < x · i} ∩ {x. x · i < 1})
    show open ?I
      by (intro open_INT finite_Basis ballI open_Int, auto intro: open_Collect_less
simp: continuous_on_inner)
    show ∑ Basis /R 2 ∈ ?I
      by simp
    show ?I ⊆ cbox 0 One
      unfolding cbox_def by force
  qed
  then have *: interior ?U ≠ {} by fast
  have *: ?U homeomorphic S
    using homeomorphic_convex_compact[OF convex_box(1) compact_cbox *
assms(2,1,3)] .
  have ∀ f. continuous_on ?U f ∧ f ∈ ?U → ?U → (∃ x∈?U. f x = x)
    using brouwer_cube by auto
  then show ?thesis
    unfolding homeomorphic_fixpoint_property[OF *]
    using assms
    by (auto intro: that)
qed

```

Then the particular case for closed balls.

```

lemma brouwer_ball:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes e > 0
    and continuous_on (cball a e) f
    and f ∈ cball a e → cball a e

```



```

obtains  $x$  where  $x \in cball\ a\ e$  and  $f\ x = x$ 
using brouwer_weak[OF compact_cball convex_cball, of a e f]
unfolding interior_cball ball_eq_empty
using assms by auto

```

And finally we prove Brouwer's fixed point theorem in its general version.

```

theorem brouwer:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'a$ 
  assumes  $S: compact\ S\ convex\ S\ S \neq \{\}$ 
    and  $contf: continuous\_on\ S\ f$ 
    and  $fim: f \in S \rightarrow S$ 
  obtains  $x$  where  $x \in S$  and  $f\ x = x$ 
proof -
  have  $\exists e > 0. S \subseteq cball\ 0\ e$ 
    using compact_imp_bounded[OF ‹compact S›] unfolding bounded_pos
    by auto
  then obtain  $e$  where  $e: e > 0\ S \subseteq cball\ 0\ e$ 
    by blast
  have  $\exists x \in cball\ 0\ e. (f \circ closest\_point\ S)\ x = x$ 
  proof (rule_tac brouwer_ball[OF e(1)])
    show  $continuous\_on\ (cball\ 0\ e)\ (f \circ closest\_point\ S)$ 
      by (meson assms closest_point_in_set compact_eq_bounded_closed contf
continuous_on_closest_point
continuous_on_compose continuous_on_subset image_subsetI)
    show  $f \circ closest\_point\ S \in cball\ 0\ e \rightarrow cball\ 0\ e$ 
      by (smt (verit) Pi_iff assms(1) assms(3) closest_point_in_set comp_apply
compact_eq_bounded_closed e(2) fim subset_eq)
    qed (use assms in auto)
  then obtain  $x$  where  $x: x \in cball\ 0\ e\ (f \circ closest\_point\ S)\ x = x ..$ 
  with  $S$  have  $x \in S$ 
    by (metis PiE closest_point_in_set comp_apply compact_imp_closed fim)
  then have  $*$ :  $closest\_point\ S\ x = x$ 
    by (rule closest_point_self)
  show thesis
proof
  show  $closest\_point\ S\ x \in S$ 
    by (simp add: ‹x \in S›)
  show  $f\ (closest\_point\ S\ x) = closest\_point\ S\ x$ 
    using  $*$   $x$  by auto
  qed
qed

```

9.18.4 Applications

So we get the no-retraction theorem.

```

corollary no_retraction_cball:
  fixes  $a :: 'a::euclidean\_space$ 
  assumes  $e > 0$ 
  shows  $\neg (frontier\ (cball\ a\ e)\ retract\_of\ (cball\ a\ e))$ 

```

proof

```

assume *: frontier (cball a e) retract_of (cball a e)
have **:  $\bigwedge xa. a - (2 *_R a - xa) = - (a - xa)$ 
  using scaleR_left_distrib[of 1 1 a] by auto
obtain x where x:  $x \in \{x. \text{norm } (a - x) = e\}$   $2 *_R a - x = x$ 
proof (rule retract_fixpoint_property[OF *, of  $\lambda x. \text{scaleR } 2 a - x$ ])
  show continuous_on (frontier (cball a e)) ((-) (2 *_R a))
    by (intro continuous_intros)
  show (-) (2 *_R a)  $\in$  frontier (cball a e)  $\rightarrow$  frontier (cball a e)
    by clarsimp (metis ** dist_norm norm_minus_cancel)
qed (auto simp: dist_norm intro: brouwer_ball[OF assms])
then have scaleR 2 a = scaleR 1 x + scaleR 1 x
  by (auto simp: algebra_simps)
then have a = x
  unfolding scaleR_left_distrib[symmetric] by auto
then show False
  using x assms by auto

```

qed

corollary contractible_sphere:

```

fixes a :: 'a::euclidean_space
shows contractible(sphere a r)  $\longleftrightarrow$   $r \leq 0$ 

```

proof (cases $0 < r$)

case True

then show ?thesis

```

  unfolding contractible_def nullhomotopic_from_sphere_extension

```

```

  using no_retraction_cball [OF True, of a]

```

```

  by (auto simp: retract_of_def retraction_def)

```

next

case False

then show ?thesis

```

  unfolding contractible_def nullhomotopic_from_sphere_extension

```

```

  using less_eq_real_def by auto

```

qed

corollary connected_sphere_eq:

```

fixes a :: 'a :: euclidean_space
shows connected(sphere a r)  $\longleftrightarrow$   $2 \leq \text{DIM}('a) \vee r \leq 0$ 
  (is ?lhs = ?rhs)

```

proof (cases r 0::real rule: linorder_cases)

case less

then show ?thesis **by** auto

next

case equal

then show ?thesis **by** auto

next

case greater

show ?thesis

proof

```

assume  $L$ : ?lhs
have  $False$  if  $1$ :  $DIM('a) = 1$ 
proof -
  obtain  $x y$  where  $xy$ :  $sphere\ a\ r = \{x,y\}$   $x \neq y$ 
    using  $sphere\_1D\_doubleton$  [ $OF\ 1\ greater$ ]
    by ( $metis\ dist\_self\ greater\ insertI1\ less\_add\_same\_cancel1\ mem\_sphere$ 
 $mult\_2\ not\_le\ zero\_le\_dist$ )
    then have  $finite$  ( $sphere\ a\ r$ )
      by  $auto$ 
    with  $L$   $\langle r > 0 \rangle xy$  show  $False$ 
      using  $connected\_finite\_iff\_sing$  by  $auto$ 
qed
with  $greater$  show ?rhs
  by ( $metis\ DIM\_ge\_Suc0\ One\_nat\_def\ Suc\_1\ le\_antisym\ not\_less\_eq\_eq$ )
next
  assume ?rhs
  then show ?lhs
    using  $connected\_sphere\ greater$  by  $auto$ 
qed
qed

corollary  $path\_connected\_sphere\_eq$ :
  fixes  $a :: 'a :: euclidean\_space$ 
  shows  $path\_connected(sphere\ a\ r) \longleftrightarrow 2 \leq DIM('a) \vee r \leq 0$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using  $connected\_sphere\_eq\ path\_connected\_imp\_connected$  by  $blast$ 
next
  assume  $R$ : ?rhs
  then show ?lhs
    by ( $auto\ simp: contractible\_imp\_path\_connected\ contractible\_sphere\ path\_connected\_sphere$ )
qed

proposition  $frontier\_subset\_retraction$ :
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes  $bounded\ S$  and  $fros$ :  $frontier\ S \subseteq T$ 
    and  $contf$ :  $continuous\_on\ (closure\ S)\ f$ 
    and  $fm$ :  $f \in S \rightarrow T$ 
    and  $fid$ :  $\bigwedge x. x \in T \implies f\ x = x$ 
  shows  $S \subseteq T$ 
proof ( $rule\ ccontr$ )
  assume  $\neg S \subseteq T$ 
  then obtain  $a$  where  $a \in S$   $a \notin T$  by  $blast$ 
  define  $g$  where  $g \equiv \lambda z. if\ z \in closure\ S\ then\ f\ z\ else\ z$ 
  have  $continuous\_on\ (closure\ S \cup closure(-S))\ g$ 
    unfolding  $g\_def$  using  $fros\ fid\ frontier\_closures$ 
    by ( $intro\ continuous\_on\_cases$ ) ( $auto\ simp: contf$ )

```

```

moreover have  $\text{closure } S \cup \text{closure}(- S) = \text{UNIV}$ 
  using  $\text{closure\_Un}$  by  $\text{fastforce}$ 
ultimately have  $\text{contg: continuous\_on UNIV } g$  by  $\text{metis}$ 
obtain  $B$  where  $0 < B$  and  $B: \text{closure } S \subseteq \text{ball } a B$ 
  using  $\langle \text{bounded } S \rangle \text{bounded\_subset\_ballD}$  by  $\text{blast}$ 
have  $\text{notga: } g \ x \neq a$  for  $x$ 
  unfolding  $g\_def$  using  $\text{fros } \text{fim } \langle a \notin T \rangle$ 
  by  $(\text{metis } \text{PiE } \text{Un\_iff } \langle a \in S \rangle \text{closure\_Un\_frontier } \text{fid } \text{subsetD})$ 
define  $h$  where  $h \equiv (\lambda y. a + (B / \text{norm}(y - a)) *_R (y - a)) \circ g$ 
have  $\neg (\text{frontier } (\text{cball } a B) \text{ retract\_of } (\text{cball } a B))$ 
  by  $(\text{metis } \text{no\_retraction\_cball } \langle 0 < B \rangle)$ 
then have  $\bigwedge k. \neg \text{retraction } (\text{cball } a B) (\text{frontier } (\text{cball } a B)) k$ 
  by  $(\text{simp add: retract\_of\_def})$ 
moreover have  $\text{retraction } (\text{cball } a B) (\text{frontier } (\text{cball } a B)) h$ 
  unfolding  $\text{retraction\_def}$ 
proof  $(\text{intro } \text{conjI } \text{ballI})$ 
  show  $\text{frontier } (\text{cball } a B) \subseteq \text{cball } a B$ 
    by  $\text{force}$ 
  show  $\text{continuous\_on } (\text{cball } a B) h$ 
    unfolding  $h\_def$ 
    by  $(\text{intro } \text{continuous\_intros})$   $(\text{use } \text{contg } \text{continuous\_on\_subset } \text{notga } \text{in } \text{auto})$ 
  show  $h \in \text{cball } a B \rightarrow \text{frontier } (\text{cball } a B)$ 
    using  $\langle 0 < B \rangle$  by  $(\text{auto } \text{simp: } h\_def \text{notga } \text{dist\_norm})$ 
  show  $\bigwedge x. x \in \text{frontier } (\text{cball } a B) \implies h \ x = x$ 
    using  $\text{notga } \langle 0 < B \rangle$ 
    apply  $(\text{simp add: } g\_def \ h\_def \ \text{field\_simps})$ 
    by  $(\text{metis } B \ \text{dist\_commute } \ \text{dist\_norm } \ \text{mem\_ball } \ \text{order\_less\_irrefl } \ \text{subset\_eq})$ 
qed
ultimately show  $\text{False}$  by  $\text{simp}$ 
qed

```

Punctured affine hulls, etc

lemma $\text{rel_frontier_deformation_retract_of_punctured_convex}$:

fixes $S :: 'a::\text{euclidean_space } \text{set}$

assumes $\text{convex } S \ \text{convex } T \ \text{bounded } S$

and $\text{arelS: } a \in \text{rel_interior } S$

and $\text{relS: } \text{rel_frontier } S \subseteq T$

and $\text{affS: } T \subseteq \text{affine hull } S$

obtains r **where** $\text{homotopic_with_canon } (\lambda x. \text{True}) (T - \{a\}) (T - \{a\}) \text{id } r$
 $\text{retraction } (T - \{a\}) (\text{rel_frontier } S) r$

proof –

have $\exists d. 0 < d \wedge (a + d *_R l) \in \text{rel_frontier } S \wedge$

$(\forall e. 0 \leq e \wedge e < d \rightarrow (a + e *_R l) \in \text{rel_interior } S)$

if $(a + l) \in \text{affine hull } S \ l \neq 0$ **for** l

using $\text{ray_to_rel_frontier } [\text{OF } \langle \text{bounded } S \rangle \ \text{arelS}]$ **that** **by** metis

then obtain dd

where $dd1: \bigwedge l. [(a + l) \in \text{affine hull } S; l \neq 0] \implies 0 < dd \ l \wedge (a + dd \ l *_R l) \in \text{rel_frontier } S$

```

    and dd2:  $\bigwedge l e. \llbracket (a + l) \in \text{affine hull } S; e < dd \ l; 0 \leq e; l \neq 0 \rrbracket$ 
       $\implies (a + e *_R l) \in \text{rel\_interior } S$ 
  by metis+
  have aaффS:  $a \in \text{affine hull } S$ 
  by (meson arelS subsetD hull_inc rel_interior_subset)
  have (( $\lambda z. z - a$ ) ' ( $\text{affine hull } S - \{a\}$ )) = (( $\lambda z. z - a$ ) ' ( $\text{affine hull } S$ )) -
  {0}
  by auto
  moreover have continuous_on ((( $\lambda z. z - a$ ) ' ( $\text{affine hull } S$ )) - {0}) ( $\lambda x. dd$ 
 $x *_R x$ )
  proof (rule continuous_on_compact_surface_projection)
    show compact (rel_frontier (( $\lambda z. z - a$ ) '  $S$ ))
    by (simp add: boundedS bounded_translation_minus compact_rel_frontier_bounded)
    have reeq: rel_frontier (( $\lambda z. z - a$ ) '  $S$ ) = ( $\lambda z. z - a$ ) ' rel_frontier  $S$ 
    using rel_frontier_translation [of  $-a$ ] add commute by simp
    also have ...  $\subseteq$  ( $\lambda z. z - a$ ) ' ( $\text{affine hull } S$ ) - {0}
    using rel_frontier_affine_hull arelS rel_frontier_def by fastforce
    finally show rel_frontier (( $\lambda z. z - a$ ) '  $S$ )  $\subseteq$  ( $\lambda z. z - a$ ) ' ( $\text{affine hull } S$ ) -
  {0} .
    show cone (( $\lambda z. z - a$ ) ' ( $\text{affine hull } S$ ))
    by (rule subspace_imp_cone)
    (use aaффS in <simp add: subspace_affine_image_comp o_def affine_translation_aux
  [of  $a$ ]>))
    show ( $0 < k \wedge k *_R x \in \text{rel\_frontier } ((\lambda z. z - a) ' S)$ )  $\longleftrightarrow$  ( $dd \ x = k$ )
    if  $x: x \in (\lambda z. z - a) ' (\text{affine hull } S) - \{0\}$  for  $k \ x$ 
  proof
    show  $dd \ x = k \implies 0 < k \wedge k *_R x \in \text{rel\_frontier } ((\lambda z. z - a) ' S)$ 
    using dd1 [of  $x$ ] that image_iff by (fastforce simp add: reeq)
  next
    assume  $k: 0 < k \wedge k *_R x \in \text{rel\_frontier } ((\lambda z. z - a) ' S)$ 
    have False if  $dd \ x < k$ 
    proof -
      have  $k \neq 0 \ a + k *_R x \in \text{closure } S$ 
      using  $k$  closure_translation [of  $-a$ ]
      by (auto simp: rel_frontier_def cong: image_cong_simp)
      then have segsub: open_segment  $a$  ( $a + k *_R x$ )  $\subseteq$  rel_interior  $S$ 
      by (metis rel_interior_closure_convex_segment [OF <convex S> arelS])
      have  $x \neq 0$  and xаффS:  $a + x \in \text{affine hull } S$ 
      using  $x$  by auto
      then have  $0 < dd \ x$  and inS:  $a + dd \ x *_R x \in \text{rel\_frontier } S$ 
      using dd1 by auto
      moreover have  $a + dd \ x *_R x \in \text{open\_segment } a$  ( $a + k *_R x$ )
      unfolding in_segment
    proof (intro conjI exI)
      show  $a + dd \ x *_R x = (1 - dd \ x / k) *_R a + (dd \ x / k) *_R (a + k *_R x)$ 
      using  $k$  by (simp add: that algebra_simps)
    qed (use < $x \neq 0$ > < $0 < dd \ x$ > that in auto)
    ultimately show ?thesis
    using segsub by (auto simp: rel_frontier_def)
  
```

```

qed
moreover have False if  $k < dd\ x$ 
  using  $x\ k$  that rel_frontier_def
  by (fastforce simp: algebra_simps reeq dest!: dd2)
ultimately show  $dd\ x = k$ 
  by fastforce
qed
qed
ultimately have *: continuous_on (( $\lambda z. z - a$ ) ' (affine hull  $S - \{a\}$ )) ( $\lambda x. dd$ 
 $x *_R x$ )
  by auto
have continuous_on (affine hull  $S - \{a\}$ ) (( $\lambda x. a + dd\ x *_R x$ )  $\circ$  ( $\lambda z. z - a$ ))
  by (intro * continuous_intros continuous_on_compose)
with affS have contdd: continuous_on ( $T - \{a\}$ ) (( $\lambda x. a + dd\ x *_R x$ )  $\circ$  ( $\lambda z. z - a$ ))
  by (blast intro: continuous_on_subset)
show ?thesis
proof
  show homotopic_with_canon ( $\lambda x. True$ ) ( $T - \{a\}$ ) ( $T - \{a\}$ ) id ( $\lambda x. a + dd$ 
 $(x-a) *_R (x-a)$ )
  proof (rule homotopic_with_linear)
    show continuous_on ( $T - \{a\}$ ) id
      by (intro continuous_intros continuous_on_compose)
    show continuous_on ( $T - \{a\}$ ) ( $\lambda x. a + dd\ (x-a) *_R (x-a)$ )
      using contdd by (simp add: o_def)
    show closed_segment (id  $x$ ) ( $a + dd\ (x-a) *_R (x-a)$ )  $\subseteq T - \{a\}$ 
      if  $x \in T - \{a\}$  for  $x$ 
    proof (clarsimp simp: in_segment, intro conjI)
      fix  $u::real$  assume  $u: 0 \leq u \leq 1$ 
      have  $a + dd\ (x-a) *_R (x-a) \in T$ 
      by (metis DiffD1 DiffD2 add commute add.right_neutral affS dd1 diff_add_cancel
relS singletonI subsetCE that)
      then show  $(1 - u) *_R x + u *_R (a + dd\ (x-a) *_R (x-a)) \in T$ 
        using convexD [OF  $\langle convex\ T \rangle$ ] that u by simp
      have iff:  $(1 - u) *_R x + u *_R (a + dd\ (x-a) *_R (x-a)) = a \iff$ 
 $(1 - u + u * d) *_R (x-a) = 0$  for  $d$ 
        by (auto simp: algebra_simps)
      have  $x \in T\ x \neq a$  using that by auto
      then have axa:  $a + (x-a) \in \text{affine hull } S$ 
        by (metis (no_types) add commute affS diff_add_cancel rev_subsetD)
      then have  $\neg dd\ (x-a) \leq 0 \wedge a + dd\ (x-a) *_R (x-a) \in \text{rel\_frontier } S$ 
        using  $\langle x \neq a \rangle$  dd1 by fastforce
      with  $\langle x \neq a \rangle$  show  $(1 - u) *_R x + u *_R (a + dd\ (x-a) *_R (x-a)) \neq a$ 
        using less_eq_real_def mult_le_0_iff not_less u by (fastforce simp: iff)
    qed
  qed
show retraction ( $T - \{a\}$ ) (rel_frontier  $S$ ) ( $\lambda x. a + dd\ (x-a) *_R (x-a)$ )
proof (simp add: retraction_def, intro conjI ballI)
  show rel_frontier  $S \subseteq T - \{a\}$ 

```

```

    using arelS relS rel_frontier_def by fastforce
  show continuous_on (T - {a}) ( $\lambda x. a + dd (x-a) *_R (x-a)$ )
    using contdd by (simp add: o_def)
  show ( $\lambda x. a + dd (x-a) *_R (x-a)$ )  $\in$  (T - {a})  $\rightarrow$  rel_frontier S
    unfolding Pi_iff using affS dd1 subset_eq by force
  show  $a + dd (x-a) *_R (x-a) = x$  if  $x: x \in$  rel_frontier S for x
  proof -
    have  $x \neq a$ 
      using that arelS by (auto simp: rel_frontier_def)
    have False if  $dd (x-a) < 1$ 
      proof -
        have  $x \in$  closure S
          using x by (auto simp: rel_frontier_def)
        then have segsub: open_segment a x  $\subseteq$  rel_interior S
          by (metis rel_interior_closure_convex_segment [OF ‹convex S› arelS])
        have xaffS:  $x \in$  affine hull S
          using affS relS x by auto
        then have  $0 < dd (x-a)$  and inS:  $a + dd (x-a) *_R (x-a) \in$  rel_frontier
          S
          using dd1 by (auto simp: ‹ $x \neq a$ ›)
        moreover have  $a + dd (x-a) *_R (x-a) \in$  open_segment a x
          unfolding in_segment
        proof (intro exI conjI)
          show  $a + dd (x-a) *_R (x-a) = (1 - dd (x-a)) *_R a + (dd (x-a)) *_R$ 
            x
            by (simp add: algebra_simps)
          qed (use ‹ $x \neq a$ › ‹ $0 < dd (x-a)$ › that in auto)
          ultimately show ?thesis
            using segsub by (auto simp: rel_frontier_def)
          qed
        moreover have False if  $1 < dd (x-a)$ 
          using x that dd2 [of x - a 1] ‹ $x \neq a$ › closure_affine_hull
          by (auto simp: rel_frontier_def)
        ultimately have  $dd (x-a) = 1$  — similar to another proof above
          by fastforce
        with that show ?thesis
          by (simp add: rel_frontier_def)
        qed
      qed
    qed
  qed
  qed
  qed

```

corollary rel_frontier_retract_of_punctured_affine_hull:

```

  fixes S :: 'a::euclidean_space set
  assumes bounded S convex S a  $\in$  rel_interior S
  shows rel_frontier S retract_of (affine hull S - {a})
  by (meson assms convex_affine_hull dual_order.refl rel_frontier_affine_hull
    rel_frontier_deformation_retract_of_punctured_convex retract_of_def)

```

corollary *rel_boundary_retract_of_punctured_affine_hull*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes $\text{compact } S \text{ convex } S \ a \in \text{rel_interior } S$

shows $(S - \text{rel_interior } S) \text{ retract_of } (\text{affine hull } S - \{a\})$

by (*metis* *assms* *closure_closed* *compact_eq_bounded_closed* *rel_frontier_def* *rel_frontier_retract_of_punctured_affine_hull*)

lemma *homotopy_eqv_rel_frontier_punctured_convex*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes $\text{convex } S \text{ bounded } S \ a \in \text{rel_interior } S \ \text{convex } T \ \text{rel_frontier } S \subseteq T \ T$

$\subseteq \text{affine hull } S$

shows $(\text{rel_frontier } S) \text{ homotopy_eqv } (T - \{a\})$

by (*meson* *assms* *deformation_retract_imp_homotopy_eqv* *homotopy_equivalent_space_sym*

rel_frontier_deformation_retract_of_punctured_convex[*of S T*])

lemma *homotopy_eqv_rel_frontier_punctured_affine_hull*:

fixes $S :: 'a::\text{euclidean_space}$ set

assumes $\text{convex } S \ \text{bounded } S \ a \in \text{rel_interior } S$

shows $(\text{rel_frontier } S) \text{ homotopy_eqv } (\text{affine hull } S - \{a\})$

by (*simp* *add: assms* *homotopy_eqv_rel_frontier_punctured_convex* *rel_frontier_affine_hull*)

lemma *path_connected_sphere_gen*:

assumes $\text{convex } S \ \text{bounded } S \ \text{aff_dim } S \neq 1$

shows $\text{path_connected}(\text{rel_frontier } S)$

proof –

have $\text{convex } (\text{closure } S)$

using *assms* **by** *auto*

then show *?thesis*

by (*metis* *Diff_empty* *aff_dim_affine_hull* *assms* *convex_affine_hull* *convex_imp_path_connected* *equalsOI*

path_connected_punctured_convex *rel_frontier_def* *rel_frontier_retract_of_punctured_affine_hull* *retract_of_path_connected*)

qed

lemma *connected_sphere_gen*:

assumes $\text{convex } S \ \text{bounded } S \ \text{aff_dim } S \neq 1$

shows $\text{connected}(\text{rel_frontier } S)$

by (*simp* *add: assms* *path_connected_imp_connected* *path_connected_sphere_gen*)

Borsuk-style characterization of separation

lemma *continuous_on_Borsuk_map*:

$a \notin S \implies \text{continuous_on } S \ (\lambda x. \text{inverse}(\text{norm } (x-a)) *_{\mathbb{R}} (x-a))$

by (*rule* *continuous_intros* | *force*)**+**

lemma *Borsuk_map_into_sphere*:

$(\lambda x. \text{inverse}(\text{norm } (x-a)) *_{\mathbb{R}} (x-a)) \in S \rightarrow \text{sphere } 0 \ 1 \longleftrightarrow (a \notin S)$

proof –


```

have  $\bigwedge x. \llbracket a \notin S; x \in S \rrbracket \implies \text{inverse} (\text{norm} (x-a)) * \text{norm} (x-a) = 1$ 
  by (metis left_inverse norm_eq_zero right_minus_eq)
then show ?thesis
  by force
qed

```

```

lemma Borsuk_maps_homotopic_in_path_component:
  assumes path_component  $(- S) a b$ 
  shows homotopic_with_canon  $(\lambda x. \text{True}) S (\text{sphere } 0 \ 1)$ 
     $(\lambda x. \text{inverse}(\text{norm}(x-a)) *_{\mathbb{R}} (x-a))$ 
     $(\lambda x. \text{inverse}(\text{norm}(x-b)) *_{\mathbb{R}} (x-b))$ 

```

```

proof -
  obtain g where  $g: \text{path } g \text{ path\_image } g \subseteq -S \text{ pathstart } g = a \text{ pathfinish } g = b$ 
  using assms by (auto simp: path_component_def)
  define h where  $h \equiv \lambda z. (\text{snd } z - (g \circ \text{fst}) z) /_{\mathbb{R}} \text{norm} (\text{snd } z - (g \circ \text{fst}) z)$ 
  have continuous_on  $(\{0..1\} \times S) h$ 
  unfolding h_def using g by (intro continuous_intros) (auto simp: path_defs)
  moreover
  have  $h' (\{0..1\} \times S) \subseteq \text{sphere } 0 \ 1$ 
  unfolding h_def using g by (auto simp: divide_simps path_defs)
  ultimately show ?thesis
  using g by (auto simp: h_def path_defs homotopic_with_def)
qed

```

```

lemma non_extensible_Borsuk_map:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes compact S and cin:  $C \in \text{components}(- S)$  and boc: bounded C and  $a \in C$ 
  shows  $\neg (\exists g. \text{continuous\_on} (S \cup C) g \wedge$ 
     $g \in (S \cup C) \rightarrow \text{sphere } 0 \ 1 \wedge$ 
     $(\forall x \in S. g \ x = \text{inverse}(\text{norm}(x-a)) *_{\mathbb{R}} (x-a)))$ 

```

```

proof -
  have closed S using assms by (simp add: compact_imp_closed)
  have  $C \subseteq -S$ 
  using assms by (simp add: in_components_subset)
  with  $\langle a \in C \rangle$  have  $a \notin S$  by blast
  then have ceq:  $C = \text{connected\_component\_set} (- S) a$ 
  by (metis  $\langle a \in C \rangle$  cin components_iff_connected_component_eq)
  then have bounded  $(S \cup \text{connected\_component\_set} (- S) a)$ 
  using  $\langle \text{compact } S \rangle$  boc compact_imp_bounded by auto
  with bounded_subset_ballD obtain r where  $0 < r$  and  $r: (S \cup \text{connected\_component\_set} (- S) a) \subseteq \text{ball } a \ r$ 
  by blast
  { fix g
    assume continuous_on  $(S \cup C) g$ 
       $g \in (S \cup C) \rightarrow \text{sphere } 0 \ 1$ 
      and [simp]:  $\bigwedge x. x \in S \implies g \ x = (x-a) /_{\mathbb{R}} \text{norm} (x-a)$ 
    then have norm_g1 [simp]:  $\bigwedge x. x \in S \cup C \implies \text{norm} (g \ x) = 1$ 
      by force
  }

```

```

have cb_eq: cball a r = (S ∪ connected_component_set (- S) a) ∪
  (cball a r - connected_component_set (- S) a)
  using ball_subset_cball [of a r] r by auto
have cont1: continuous_on (S ∪ connected_component_set (- S) a)
  (λx. a + r *R g x)
  using ⟨continuous_on (S ∪ C) g⟩ ceq
  by (intro continuous_intros) blast
have cont2: continuous_on (cball a r - connected_component_set (- S) a)
  (λx. a + r *R ((x-a) /R norm (x-a)))
  by (rule continuous_intros | force simp: ⟨a ∉ S⟩)+
have 1: continuous_on (cball a r)
  (λx. if connected_component (- S) a x
    then a + r *R g x
    else a + r *R ((x-a) /R norm (x-a)))
  apply (subst cb_eq)
  apply (rule continuous_on_cases [OF _ _ cont1 cont2])
  using ⟨closed S⟩ ceq cin
  by (force simp: closed_Diff open_Cmpl closed_Un_complement_component
open_connected_component)+
  have 2: (λx. a + r *R g x) ‘ (cball a r ∩ connected_component_set (- S) a)
    ⊆ sphere a r
    using ⟨0 < r⟩ by (force simp: dist_norm ceq)
  have retraction (cball a r) (sphere a r)
    (λx. if x ∈ connected_component_set (- S) a
      then a + r *R g x
      else a + r *R ((x-a) /R norm (x-a)))
    using ⟨0 < r⟩ ⟨a ∉ S⟩ ⟨a ∈ C⟩ r
    by (auto simp: norm_minus_commute retraction_def Pi_iff ceq dist_norm
abs_if
  mult_less_0_iff divide_simps 1 2)
  then have False
  using no_retraction_cball
    [OF ⟨0 < r⟩, of a, unfolded retract_of_def, simplified, rule_format,
  of λx. if x ∈ connected_component_set (- S) a
    then a + r *R g x
    else a + r *R inverse(norm(x-a)) *R (x-a)]
  by blast
}
then show ?thesis
  by blast
qed

```

Proving surjectivity via Brouwer fixpoint theorem

lemma brouwer_surjective:

fixes $f :: 'n::euclidean_space \Rightarrow 'n$

assumes $T: compact\ T\ convex\ T\ T \neq \{\}$

and $f: continuous_on\ T\ f$

and $\bigwedge x\ y. [x \in S; y \in T] \implies x + (y - f\ y) \in T$

```

  and  $x \in S$ 
  shows  $\exists y \in T. f y = x$ 
  proof -
  have *:  $\bigwedge x y. f y = x \longleftrightarrow x + (y - f y) = y$ 
    by (auto simp add: algebra_simps)
  show ?thesis
    unfolding *
  proof (rule brouwer[OF T])
    show continuous_on T ( $\lambda y. x + (y - f y)$ )
      by (intro continuous_intros f)
    qed (use assms in auto)
  qed

```

```

lemma brouwer_surjective_cball:
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'n$ 
  assumes continuous_on (cball a e) f
    and  $e > 0$ 
    and  $x \in S$ 
    and  $\bigwedge x y. [x \in S; y \in \text{cball } a \text{ e}] \implies x + (y - f y) \in \text{cball } a \text{ e}$ 
  shows  $\exists y \in \text{cball } a \text{ e}. f y = x$ 
  by (smt (verit, best) assms brouwer_surjective cball_eq_empty compact_cball
convex_cball)

```

Inverse function theorem

See Sussmann: "Multidifferential calculus", Theorem 2.1.1

```

lemma sussmann_open_mapping:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow 'b::euclidean\_space$ 
  assumes open S
    and contf: continuous_on S f
    and  $x \in S$ 
    and derf: (f has_derivative f') (at x)
    and bounded_linear g'  $f' \circ g' = \text{id}$ 
    and  $T \subseteq S$ 
    and  $x: x \in \text{interior } T$ 
  shows  $f x \in \text{interior } (f ' T)$ 
  proof -
  interpret f': bounded_linear f'
    using assms unfolding has_derivative_def by auto
  interpret g': bounded_linear g'
    using assms by auto
  obtain B where  $B: 0 < B \ \forall x. \text{norm } (g' x) \leq \text{norm } x * B$ 
    using bounded_linear.pos_bounded[OF assms(5)] by blast
  hence *:  $1 / (2 * B) > 0$  by auto
  obtain e0 where e0:
     $0 < e0$ 
     $\forall y. \text{norm } (y - x) < e0 \implies \text{norm } (f y - f x - f' (y - x)) \leq 1 / (2 * B) * \text{norm } (y - x)$ 
    using derf unfolding has_derivative_at_alt

```

```

using * by blast
obtain e1 where e1: 0 < e1 cball x e1 ⊆ T
using mem_interior_cball x by blast
have *: 0 < e0 / B 0 < e1 / B using e0 e1 B by auto
obtain e where e: 0 < e e < e0 / B e < e1 / B
using field_lbound_gt_zero[OF *] by blast
have lem: ∃ y ∈ cball (f x) e. f (x + g' (y - f x)) = z if z ∈ cball (f x) (e / 2) for
z
proof (rule brouwer_surjective_cball)
have z: z ∈ S if as: y ∈ cball (f x) e z = x + (g' y - g' (f x)) for y z
proof-
have dist x z = norm (g' (f x) - g' y)
unfolding as(2) and dist_norm by auto
also have ... ≤ norm (f x - y) * B
by (metis B(2) g'.diff)
also have ... ≤ e * B
by (metis B(1) dist_norm mem_cball mult_le_cancel_right_pos that(1))
also have ... ≤ e1
using B(1) e(3) pos_less_divide_eq by fastforce
finally have z ∈ cball x e1
by force
then show z ∈ S
using e1 assms(7) by auto
qed
show continuous_on (cball (f x) e) (λy. f (x + g' (y - f x)))
unfolding g'.diff
proof (rule continuous_on_compose2 [OF __ order_refl, of __ f])
show continuous_on ((λy. x + (g' y - g' (f x))) ' cball (f x) e) f
by (rule continuous_on_subset[OF contf]) (use z in blast)
show continuous_on (cball (f x) e) (λy. x + (g' y - g' (f x)))
by (intro continuous_intros linear_continuous_on[OF ‹bounded_linear g'›])
qed
next
fix y z
assume y: y ∈ cball (f x) (e / 2) and z: z ∈ cball (f x) e
have norm (g' (z - f x)) ≤ norm (z - f x) * B
using B by auto
also have ... ≤ e * B
by (metis B(1) z dist_norm mem_cball norm_minus_commute mult_le_cancel_right_pos)
also have ... < e0
using B(1) e(2) pos_less_divide_eq by blast
finally have *: norm (x + g' (z - f x) - x) < e0
by auto
have **: f x + f' (x + g' (z - f x) - x) = z
using assms(6)[unfolded o_def id_def, THEN cong]
by auto
have norm (f x - (y + (z - f (x + g' (z - f x)))))) ≤
norm (f (x + g' (z - f x)) - z) + norm (f x - y)
using norm_triangle_ineq[of f (x + g'(z - f x)) - z f x - y]

```

```

    by (auto simp add: algebra_simps)
  also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + norm (f x - y)
    using e0(2)[rule_format, OF *]
    by (simp only: algebra_simps **) auto
  also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + e/2
    using y by (auto simp: dist_norm)
  also have ... ≤ 1 / (B * 2) * B * norm (z - f x) + e/2
    using * B by (auto simp add: field_simps)
  also have ... ≤ 1 / 2 * norm (z - f x) + e/2
    by auto
  also have ... ≤ e/2 + e/2
    using B(1) ⟨norm (z - f x) * B ≤ e * B⟩ by auto
  finally show y + (z - f (x + g' (z - f x))) ∈ cball (f x) e
    by (auto simp: dist_norm)
qed (use e that in auto)
show ?thesis
  unfolding mem_interior
proof (intro exI conjI subsetI)
  fix y
  assume y ∈ ball (f x) (e / 2)
  then have *: y ∈ cball (f x) (e / 2)
    by auto
  obtain z where z: z ∈ cball (f x) e ∧ f (x + g' (z - f x)) = y
    using lem * by blast
  then have norm (g' (z - f x)) ≤ norm (z - f x) * B
    using B
    by (auto simp add: field_simps)
  also have ... ≤ e * B
  by (metis B(1) dist_norm mem_cball norm_minus_commute mult_le_cancel_right_pos
z(1))
  also have ... ≤ e1
    using e B unfolding less_divide_eq by auto
  finally have x + g'(z - f x) ∈ T
  by (metis add_diff_cancel_diff_diff_add dist_norm e1(2) mem_cball norm_minus_commute
subset_eq)
  then show y ∈ f ' T
    using z by auto
qed (use e in auto)
qed

```

Hence the following eccentric variant of the inverse function theorem. This has no continuity assumptions, but we do need the inverse function. We could put $f' \circ g = I$ but this happens to fit with the minimal linear algebra theory I've set up so far.

lemma *has_derivative_inverse_strong*:
fixes $f :: 'n::euclidean_space \Rightarrow 'n$
assumes S : open S $x \in S$
and *contf*: continuous_on S f
and *gf*: $\bigwedge x. x \in S \implies g (f x) = x$

3322

```

    and derf: (f has_derivative f') (at x)
    and id: f' o g' = id
  shows (g has_derivative g') (at (f x))
proof -
  have linf: bounded_linear f'
    using derf unfolding has_derivative_def by auto
  then have ling: bounded_linear g'
    unfolding linear_conv_bounded_linear[symmetric]
    using id right_inverse_linear by blast
  moreover have g' o f' = id
    using id linear_inverse_left linear_linear linf ling by blast
  moreover have *:  $\bigwedge T. \llbracket T \subseteq S; x \in \text{interior } T \rrbracket \implies f x \in \text{interior } (f' T)$ 
    using S derf contf id ling sussmann_open_mapping by blast
  have continuous (at (f x)) g
    unfolding continuous_at Lim_at
  proof (intro strip)
    fix e :: real
    assume e > 0
    then have fx ∈ interior (f' (ball x e ∩ S))
      by (simp add: * S interior_open)
    then obtain d where d: 0 < d ball (f x) d ⊆ f' (ball x e ∩ S)
      unfolding mem_interior by blast
    show  $\exists d > 0. \forall y. 0 < \text{dist } y (f x) \wedge \text{dist } y (f x) < d \longrightarrow \text{dist } (g y) (g (f x)) < e$ 
  proof (intro exI allI impI conjI)
    fix y
    assume 0 < dist y (f x) ∧ dist y (f x) < d
    then have gy ∈ g' f' (ball x e ∩ S)
      by (metis d(2) dist_commute mem_ball rev_image_eqI subset_iff)
    then show dist (g y) (g (f x)) < e
      using ⟨x ∈ S⟩ by (simp add: gf_dist_commute image_iff)
    qed (use d in auto)
  qed
  moreover have fx ∈ interior (f' S)
    using * S interior_eq by blast
  moreover have f (g y) = y if y ∈ interior (f' S) for y
    by (metis gf_imageE interiorE subsetD that)
  ultimately show ?thesis using assms
    by (metis has_derivative_inverse_basic_x open_interior)
qed

```

A rewrite based on the other domain.

```

lemma has_derivative_inverse_strong_x:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes open S
  and g y ∈ S
  and continuous_on S f
  and  $\bigwedge x. x \in S \implies g (f x) = x$ 
  and (f has_derivative f') (at (g y))

```

```

  and  $f' \circ g' = id$ 
  and  $f: f (g y) = y$ 
  shows  $(g \text{ has\_derivative } g') (at y)$ 
  using  $has\_derivative\_inverse\_strong[OF \text{assms}(1-6)]$  by  $(simp \text{add: } f)$ 

```

On a region.

```

theorem  $has\_derivative\_inverse\_on$ :
  fixes  $f :: 'n::euclidean\_space \Rightarrow 'n$ 
  assumes  $open S$ 
  and  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f'(x)) (at x)$ 
  and  $\bigwedge x. x \in S \implies g (f x) = x$ 
  and  $f' x \circ g' x = id$ 
  and  $x \in S$ 
  shows  $(g \text{ has\_derivative } g'(x)) (at (f x))$ 
  by  $(meson \text{assms } continuous\_on\_eq\_continuous\_at \text{ has\_derivative\_continuous}$ 
 $has\_derivative\_inverse\_strong)$ 

```

end

9.19 Fashoda Meet Theorem

```

theory  $Fashoda\_Theorem$ 
imports  $Brouwer\_Fixpoint Path\_Connected Cartesian\_Euclidean\_Space$ 
begin

```

9.19.1 Bijections between intervals

```

definition  $interval\_bij :: 'a \times 'a \Rightarrow 'a \times 'a \Rightarrow 'a \Rightarrow 'a::euclidean\_space$ 
  where  $interval\_bij =$ 
   $(\lambda(a, b) (u, v) x. (\sum i \in Basis. (u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i))$ 
 $*_R i))$ 

```

```

lemma  $interval\_bij\_affine$ :
   $interval\_bij (a,b) (u,v) = (\lambda x. (\sum i \in Basis. ((v \cdot i - u \cdot i) / (b \cdot i - a \cdot i) * (x \cdot i)) *_R$ 
 $i) +$ 
   $(\sum i \in Basis. (u \cdot i - (v \cdot i - u \cdot i) / (b \cdot i - a \cdot i) * (a \cdot i)) *_R i))$ 
  by  $(simp \text{add: } interval\_bij\_def \text{ algebra\_simps } add\_divide\_distrib \text{ diff\_divide\_distrib}$ 
 $flip: \text{sum.distrib } scaleR\_add\_left)$ 

```

```

lemma  $continuous\_interval\_bij$ :
  fixes  $a b :: 'a::euclidean\_space$ 
  shows  $continuous (at x) (interval\_bij (a, b) (u, v))$ 
  by  $(auto \text{simp add: } divide\_inverse \text{ interval\_bij\_def } intro!: \text{continuous\_sum } con-$ 
 $tinuous\_intros)$ 

```

```

lemma  $continuous\_on\_interval\_bij$ :  $continuous\_on s (interval\_bij (a, b) (u, v))$ 
  by  $(metis \text{continuous\_at\_imp\_continuous\_on } continuous\_interval\_bij)$ 

```

lemma *in_interval_interval_bij*:
fixes $a\ b\ u\ v\ x :: 'a::\text{euclidean_space}$
assumes $x \in \text{cbox } a\ b$
and $\text{cbox } u\ v \neq \{\}$
shows $\text{interval_bij } (a, b) (u, v) x \in \text{cbox } u\ v$
proof –
have $\bigwedge i. i \in \text{Basis} \implies u \cdot i \leq u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i)$
by (*smt (verit) assms box_ne_empty(1) divide_nonneg_nonneg mem_box(2) mult_nonneg_nonneg*)
moreover
have $\bigwedge i. i \in \text{Basis} \implies u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i) \leq v \cdot i$
apply (*simp add: divide_simps algebra_simps*)
by (*smt (verit, best) assms box_ne_empty(1) left_diff_distrib mem_box(2) mult commute mult_left_mono*)
ultimately show *?thesis*
by (*force simp only: interval_bij_def split_conv mem_box inner_sum_left_Basis*)
qed

lemma *interval_bij_bij*:
 $\forall (i::'a::\text{euclidean_space}) \in \text{Basis}. a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i \implies$
 $\text{interval_bij } (a, b) (u, v) (\text{interval_bij } (u, v) (a, b) x) = x$
by (*auto simp: interval_bij_def euclidean_eq_iff [where 'a='a]*)

lemma *interval_bij_bij_cart*: **fixes** $x::\text{real}^n$ **assumes** $\forall i. a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i$
shows $\text{interval_bij } (a, b) (u, v) (\text{interval_bij } (u, v) (a, b) x) = x$
using *assms* **by** (*intro interval_bij_bij*) (*auto simp: Basis_vec_def inner_axis*)

9.19.2 Fashoda meet theorem

lemma *infnorm_2*:
fixes $x :: \text{real}^2$
shows $\text{infnorm } x = \max |x\$1| |x\$2|$
unfolding *infnorm_cart UNIV_2* **by** (*rule cSup_eq*) *auto*

lemma *infnorm_eq_1_2*:
fixes $x :: \text{real}^2$
shows $\text{infnorm } x = 1 \iff$
 $|x\$1| \leq 1 \wedge |x\$2| \leq 1 \wedge (x\$1 = -1 \vee x\$1 = 1 \vee x\$2 = -1 \vee x\$2 = 1)$
unfolding *infnorm_2* **by** *auto*

lemma *infnorm_eq_1_imp*:
fixes $x :: \text{real}^2$
assumes $\text{infnorm } x = 1$
shows $|x\$1| \leq 1$ **and** $|x\$2| \leq 1$
using *assms* **unfolding** *infnorm_eq_1_2* **by** *auto*


```

proposition fashoda_unit:
  fixes f g :: real  $\Rightarrow$  real2
  assumes f ' {-1 .. 1}  $\subseteq$  cbox (-1) 1
    and g ' {-1 .. 1}  $\subseteq$  cbox (-1) 1
    and continuous_on {-1 .. 1} f
    and continuous_on {-1 .. 1} g
    and f (- 1)$1 = - 1
    and f 1$1 = 1 g (- 1) $2 = -1
    and g 1 $2 = 1
  shows  $\exists s \in \{-1 .. 1\}. \exists t \in \{-1 .. 1\}. f s = g t$ 
proof (rule ccontr)
  assume  $\neg$  ?thesis
  note as = this[unfolded bex_simps,rule_format]
  define sqprojection
    where [abs_def]: sqprojection z = (inverse (infnorm z)) *R z for z :: real2
  define negatex :: real2  $\Rightarrow$  real2
    where negatex x = (vector [-(x$1), x$2]) for x
  have inf_negatex:  $\bigwedge z :: \text{real}^2. \text{infnorm} (\text{negatex } z) = \text{infnorm } z$ 
    unfolding negatex_def infnorm_2 vector_2 by auto
  have inf_eq1:  $\bigwedge z. z \neq 0 \implies \text{infnorm} (\text{sqprojection } z) = 1$ 
    unfolding sqprojection_def infnorm_mul[unfolded scalar_mult_eq_scaleR]
    by (simp add: real_abs_infnorm infnorm_eq_0)
  let ?F =  $\lambda w :: \text{real}^2. (f \circ (\lambda x. x \$ 1)) w - (g \circ (\lambda x. x \$ 2)) w$ 
  have *:  $\bigwedge i. (\lambda x :: \text{real}^2. x \$ i) ' \text{cbox} (-1) 1 = \{-1..1\}$ 
  proof
    show  $(\lambda x :: \text{real}^2. x \$ i) ' \text{cbox} (-1) 1 \subseteq \{-1..1\}$  for i
      by (auto simp: mem_box_cart)
    show  $\{-1..1\} \subseteq (\lambda x :: \text{real}^2. x \$ i) ' \text{cbox} (-1) 1$  for i
      by (clarsimp simp: image_iff mem_box_cart Bex_def) (metis (no_types,
opaque_lifting) vec_component)
  qed
  {
    fix x
    assume x  $\in (\lambda w. (f \circ (\lambda x. x \$ 1)) w - (g \circ (\lambda x. x \$ 2)) w) ' (\text{cbox} (-1) 1)$ 
    (1 :: real2)
    then obtain w :: real2 where w:
      w  $\in \text{cbox} (-1) 1$ 
      x =  $(f \circ (\lambda x. x \$ 1)) w - (g \circ (\lambda x. x \$ 2)) w$ 
    unfolding image_iff ..
    then have x  $\neq 0$ 
      using as[of w$1 w$2] by (auto simp: mem_box_cart atLeastAtMost_iff)
  } note x0 = this
  let ?CB11 = cbox (-1) (1 :: real2)
  obtain x :: real2 where x:
    x  $\in \text{cbox} (-1) 1$ 
    (negatex  $\circ$  sqprojection  $\circ (\lambda w. (f \circ (\lambda x. x \$ 1)) w - (g \circ (\lambda x. x \$ 2)) w)$ ) x
  = x
  proof (rule brouwer_weak[of ?CB11 negatex  $\circ$  sqprojection  $\circ$  ?F])
    show compact ?CB11 convex ?CB11

```

```

    by (rule compact_cbox_convex_box)+
  have box (- 1) (1::real^2) ≠ {}
    unfolding interval_eq_empty_cart by auto
  then show interior ?CB11 ≠ {}
    by simp
  have negatex (x + y) $ i = (negatex x + negatex y) $ i ∧ negatex (c *R x) $
i = (c *R negatex x) $ i
    for i x y c
    using exhaust_2 [of i] by (auto simp: negatex_def)
  then have bounded_linear negatex
    by (simp add: bounded_linearI' vec_eq_iff)
  then show continuous_on ?CB11 (negatex ∘ sqprojection ∘ ?F)
    unfolding sqprojection_def
    apply (intro continuous_intros continuous_on_component | use * assms in
presburger)+
    apply (simp_all add: infnorm_eq_0 x0 linear_continuous_on)
    done
  have (negatex ∘ sqprojection ∘ ?F) ' ?CB11 ⊆ ?CB11
  proof clarsimp
    fix y :: real^2
    assume y: y ∈ ?CB11
    have ?F y ≠ 0
      by (rule x0) (use y in auto)
    then have *: infnorm (sqprojection (?F y)) = 1
      using inf_eq1 by blast
    show negatex (sqprojection (f (y $ 1) - g (y $ 2))) ∈ cbox (-1) 1
      unfolding mem_box_cart interval_cbox_cart infnorm_2
    by (smt (verit, del_insts) * component_le_infnorm_cart inf_neg neg_one_index
o_apply one_index)
    qed
  then show negatex ∘ sqprojection ∘ ?F ∈ ?CB11 → ?CB11
    by blast
  qed
  have ?F x ≠ 0
    by (rule x0) (use x in auto)
  then have *: infnorm (sqprojection (?F x)) = 1
    using inf_eq1 by blast
  have nx: infnorm x = 1
    by (metis (no_types, lifting) * inf_neg o_apply x(2))
  have iff: 0 < sqprojection x $ i ↔ 0 < x $ i sqprojection x $ i < 0 ↔ x $ i < 0
if x ≠ 0 for x i
  proof -
    have *: inverse (infnorm x) > 0
      by (simp add: infnorm_pos_lt that)
    then show (0 < sqprojection x $ i) = (0 < x $ i)
      by (simp add: sqprojection_def zero_less_mult_iff)
    show (sqprojection x $ i < 0) = (x $ i < 0)
      unfolding sqprojection_def
      by (metis * pos_less_divideR_eq scaleR_zero_right vector_scaleR_component)

```

```

qed
have x1: x $ 1 ∈ {- 1..1::real} x $ 2 ∈ {- 1..1::real}
  using x(1) unfolding mem_box_cart by auto
then have nz: f (x $ 1) - g (x $ 2) ≠ 0
  using as by auto
consider x $ 1 = -1 | x $ 1 = 1 | x $ 2 = -1 | x $ 2 = 1
  using nx unfolding infnorm_eq_1_2 by auto
then show False
proof cases
case 1
then have *: f (x $ 1) $ 1 = - 1
  using assms(5) by auto
have sqprojection (f (x$1) - g (x$2)) $ 1 > 0
  by (smt (verit) 1 negatex_def o_apply vector_2(1) x(2))
moreover
from x1 have g (x $ 2) ∈ cbox (-1) 1
  using assms(2) by blast
ultimately show False
  unfolding iff[OF nz] vector_component_simps * mem_box_cart
  using not_le by auto
next
case 2
then have *: f (x $ 1) $ 1 = 1
  using assms(6) by auto
have sqprojection (f (x$1) - g (x$2)) $ 1 < 0
  by (smt (verit) 2 negatex_def o_apply vector_2(1) x(2) zero_less_one)
moreover have g (x $ 2) ∈ cbox (-1) 1
  using assms(2) x1 by blast
ultimately show False
  unfolding iff[OF nz] vector_component_simps * mem_box_cart
  using not_le by auto
next
case 3
then have *: g (x $ 2) $ 2 = - 1
  using assms(7) by auto
moreover have sqprojection (f (x$1) - g (x$2)) $ 2 < 0
  by (smt (verit, ccfv_SIG) 3 negatex_def o_apply vector_2(2) x(2))
moreover from x1 have f (x $ 1) ∈ cbox (-1) 1
  using assms(1) by blast
ultimately show False
  by (smt (verit, del_insts) iff(2) mem_box_cart(2) neg_one_index nz vec-
tor_minus_component)
next
case 4
then have *: g (x $ 2) $ 2 = 1
  using assms(8) by auto
have sqprojection (f (x$1) - g (x$2)) $ 2 > 0
  by (smt (verit, best) 4 negatex_def o_apply vector_2(2) x(2))
moreover

```

```

    from  $x1$  have  $f(x \ \$ \ 1) \in \text{cbox } (-1) \ 1$ 
      using  $\text{assms}(1)$  by  $\text{blast}$ 
    ultimately show  $\text{False}$ 
    by ( $\text{smt } (\text{verit}) * \text{iff}(1) \text{ mem\_box\_cart}(2) \text{ nz one\_index vector\_minus\_component}$ )
  qed
qed

```

proposition *fashoda_unit_path*:

```

fixes  $f \ g :: \text{real} \Rightarrow \text{real}^2$ 
assumes  $\text{path } f$ 
  and  $\text{path } g$ 
  and  $\text{path\_image } f \subseteq \text{cbox } (-1) \ 1$ 
  and  $\text{path\_image } g \subseteq \text{cbox } (-1) \ 1$ 
  and  $(\text{pathstart } f)\$1 = -1$ 
  and  $(\text{pathfinish } f)\$1 = 1$ 
  and  $(\text{pathstart } g)\$2 = -1$ 
  and  $(\text{pathfinish } g)\$2 = 1$ 
obtains  $z$  where  $z \in \text{path\_image } f$  and  $z \in \text{path\_image } g$ 
proof -
note  $\text{assms} = \text{assms}[\text{unfolded path\_def pathstart\_def pathfinish\_def path\_image\_def}]$ 
define  $\text{iscale}$  where  $[\text{abs\_def}]: \text{iscale } z = \text{inverse } 2 *_{\mathbb{R}} (z + 1)$  for  $z :: \text{real}$ 
have  $\text{isc}: \text{iscale } \{-1..1\} \subseteq \{0..1\}$ 
  unfolding  $\text{iscale\_def}$  by  $\text{auto}$ 
have  $\exists s \in \{-1..1\}. \exists t \in \{-1..1\}. (f \circ \text{iscale}) s = (g \circ \text{iscale}) t$ 
proof (rule fashoda_unit)
  show  $(f \circ \text{iscale}) \{-1..1\} \subseteq \text{cbox } (-1) \ 1$   $(g \circ \text{iscale}) \{-1..1\} \subseteq \text{cbox } (-1) \ 1$ 
    using  $\text{isc}$  and  $\text{assms}(3-4)$  by ( $\text{auto simp add: image\_comp [symmetric]}$ )
  have  $*$ :  $\text{continuous\_on } \{-1..1\} \text{ iscale}$ 
    unfolding  $\text{iscale\_def}$  by ( $\text{rule continuous\_intros}$ )+
  show  $\text{continuous\_on } \{-1..1\} (f \circ \text{iscale})$ 
    using  $*$   $\text{assms}(1)$   $\text{continuous\_on\_compose continuous\_on\_subset isc}$  by  $\text{blast}$ 
  show  $\text{continuous\_on } \{-1..1\} (g \circ \text{iscale})$ 
    by ( $\text{meson } * \text{assms}(2) \text{continuous\_on\_compose continuous\_on\_subset isc}$ )
  have  $*$ :  $(1 / 2) *_{\mathbb{R}} (1 + (1::\text{real}^1)) = 1$ 
    unfolding  $\text{vec\_eq\_iff}$  by  $\text{auto}$ 
  show  $(f \circ \text{iscale}) (-1) \$ 1 = -1$ 
    and  $(f \circ \text{iscale}) 1 \$ 1 = 1$ 
    and  $(g \circ \text{iscale}) (-1) \$ 2 = -1$ 
    and  $(g \circ \text{iscale}) 1 \$ 2 = 1$ 
    unfolding  $\text{o\_def iscale\_def}$  using  $\text{assms}$  by ( $\text{auto simp add: } *$ )
  qed
  then obtain  $s \ t$  where  $st: s \in \{-1..1\} \ t \in \{-1..1\} \ (f \circ \text{iscale}) s = (g \circ \text{iscale}) t$ 
    by  $\text{auto}$ 
  show  $\text{thesis}$ 
proof
  show  $f(\text{iscale } s) \in \text{path\_image } f$ 
    by ( $\text{metis image\_eqI image\_subset\_iff isc path\_image\_def st}(1)$ )

```

```

    show  $f \text{ (iscale } s) \in \text{path\_image } g$ 
      by (metis comp_def image_eqI image_subset_iff isc_path_image_def st(2)
st(3))
    qed
  qed

theorem fashoda:
  fixes  $b :: \text{real}^2$ 
  assumes path  $f$ 
    and path  $g$ 
    and  $\text{path\_image } f \subseteq \text{cbox } a \ b$ 
    and  $\text{path\_image } g \subseteq \text{cbox } a \ b$ 
    and  $(\text{pathstart } f)\$1 = a\$1$ 
    and  $(\text{pathfinish } f)\$1 = b\$1$ 
    and  $(\text{pathstart } g)\$2 = a\$2$ 
    and  $(\text{pathfinish } g)\$2 = b\$2$ 
  obtains  $z$  where  $z \in \text{path\_image } f$  and  $z \in \text{path\_image } g$ 
proof -
  fix  $P \ Q \ S$ 
  presume  $P \vee Q \vee S \implies \text{thesis}$  and  $Q \implies \text{thesis}$  and  $S \implies \text{thesis}$ 
  then show thesis
    by auto
next
  have  $\text{cbox } a \ b \neq \{\}$ 
    using  $\text{assms}(3)$  using  $\text{path\_image\_nonempty}[of \ f]$  by auto
  then have  $a \leq b$ 
    unfolding  $\text{interval\_eq\_empty\_cart\_less\_eq\_vec\_def}$  by (auto simp add: not_less)
  then show  $a\$1 = b\$1 \vee a\$2 = b\$2 \vee (a\$1 < b\$1 \wedge a\$2 < b\$2)$ 
    unfolding  $\text{less\_eq\_vec\_def\_forall\_2}$  by auto
next
  assume as:  $a\$1 = b\$1$ 
  have  $\exists z \in \text{path\_image } g. z\$2 = (\text{pathstart } f)\$2$ 
  proof (rule  $\text{connected\_ivt\_component\_cart}$ )
    show  $\text{pathstart } g \ \$2 \leq \text{pathstart } f \ \$2$ 
      by (metis  $\text{assms}(3)$   $\text{assms}(7)$   $\text{mem\_box\_cart}(2)$   $\text{pathstart\_in\_path\_image}$ 
 $\text{subset\_iff}$ )
    show  $\text{pathstart } f \ \$2 \leq \text{pathfinish } g \ \$2$ 
      by (metis  $\text{assms}(3)$   $\text{assms}(8)$   $\text{in\_mono\_mem\_box\_cart}(2)$   $\text{pathstart\_in\_path\_image}$ )
    show  $\text{connected } (\text{path\_image } g)$ 
      using  $\text{assms}(2)$  by blast
  qed (auto simp:  $\text{path\_defs}$ )
  then obtain  $z :: \text{real}^2$  where  $z: z \in \text{path\_image } g \ z \ \$2 = \text{pathstart } f \ \$2 ..$ 
  have  $z \in \text{cbox } a \ b$ 
    using  $\text{assms}(4)$   $z(1)$  by blast
  then have  $z = f \ 0$ 
    by (smt (verit) as  $\text{assms}(5)$   $\text{exhaust\_2}$   $\text{mem\_box\_cart}(2)$   $\text{nle\_le\_pathstart\_def}$ 
 $\text{vec\_eq\_iff } z(2)$ )
  then show thesis
    by (metis  $\text{path\_defs}(2)$   $\text{pathstart\_in\_path\_image}$  that  $z(1)$ )

```

```

next
  assume as: a$2 = b$2
  have  $\exists z \in \text{path\_image } f. z\$1 = (\text{pathstart } g)\$1$ 
  proof (rule connected_ivt_component_cart)
    show  $\text{pathstart } f \$ 1 \leq \text{pathstart } g \$ 1$ 
      using assms(4) assms(5) mem_box_cart(2) by fastforce
    show  $\text{pathstart } g \$ 1 \leq \text{pathfinish } f \$ 1$ 
      using assms(4) assms(6) mem_box_cart(2) pathstart_in_path_image by
fastforce
    show connected (path_image f)
      by (simp add: assms(1) connected_path_image)
  qed (auto simp: path_defs)
  then obtain z where z:  $z \in \text{path\_image } f \ z \$ 1 = \text{pathstart } g \$ 1 ..$ 
  have  $z \in \text{cbox } a \ b$ 
    using assms(3) z(1) by auto
  then have  $z = g \ 0$ 
    by (smt (verit) as assms(7) exhaust_2 mem_box_cart(2) pathstart_def vec_eq_iff
z(2))
  then show thesis
    by (metis path_defs(2) pathstart_in_path_image that z(1))
next
  assume as:  $a \$ 1 < b \$ 1 \wedge a \$ 2 < b \$ 2$ 
  have int_nem:  $\text{cbox } (-1) \ (1::\text{real}^2) \neq \{\}$ 
  unfolding interval_eq_empty_cart by auto
  obtain z ::  $\text{real}^2$  where z:
     $z \in (\text{interval\_bij } (a, b) \ (-1, 1) \circ f) \ \{0..1\}$ 
     $z \in (\text{interval\_bij } (a, b) \ (-1, 1) \circ g) \ \{0..1\}$ 
  proof (rule fashoda_unit_path)
    show path (interval_bij (a, b) (-1, 1) o f)
      by (meson assms(1) continuous_on_interval_bij path_continuous_image)
    show path (interval_bij (a, b) (-1, 1) o g)
      by (meson assms(2) continuous_on_interval_bij path_continuous_image)
    show path_image (interval_bij (a, b) (-1, 1) o f)  $\subseteq \text{cbox } (-1) \ 1$ 
      using assms(3)
    by (simp add: path_image_def in_interval_interval_bij int_nem subset_eq)
    show path_image (interval_bij (a, b) (-1, 1) o g)  $\subseteq \text{cbox } (-1) \ 1$ 
      using assms(4)
    by (simp add: path_image_def in_interval_interval_bij int_nem subset_eq)
    show  $\text{pathstart } (\text{interval\_bij } (a, b) \ (-1, 1) \circ f) \$ 1 = -1$ 
       $\text{pathfinish } (\text{interval\_bij } (a, b) \ (-1, 1) \circ f) \$ 1 = 1$ 
       $\text{pathstart } (\text{interval\_bij } (a, b) \ (-1, 1) \circ g) \$ 2 = -1$ 
       $\text{pathfinish } (\text{interval\_bij } (a, b) \ (-1, 1) \circ g) \$ 2 = 1$ 
    using assms as
    by (simp_all add: cart_eq_inner_axis pathstart_def pathfinish_def inter-
val_bij_def)
      (simp_all add: inner_axis)
  qed (auto simp: path_defs)
  then obtain zf zg where zf:  $zf \in \{0..1\} \ z = (\text{interval\_bij } (a, b) \ (-1, 1) \circ f)$ 
zf

```

```

      and zg: zg ∈ {0..1} z = (interval_bij (a, b) (- 1, 1) ∘ g) zg
    by blast
  have *: ∀ i. (- 1) $ i < (1::real^2) $ i ∧ a $ i < b $ i
    unfolding forall_2 using as by auto
  show thesis
  proof (rule_tac z=interval_bij (- 1,1) (a,b) z in that)
    show interval_bij (- 1, 1) (a, b) z ∈ path_image f
      using zf by (simp add: interval_bij_bij_cart[OF *] path_image_def)
    show interval_bij (- 1, 1) (a, b) z ∈ path_image g
      using zg by (simp add: interval_bij_bij_cart[OF *] path_image_def)
  qed
qed

```

9.19.3 Some slightly ad hoc lemmas I use below

lemma *segment_vertical*:

```

  fixes a :: real^2
  assumes a$1 = b$1
  shows x ∈ closed_segment a b ⟷
    x$1 = a$1 ∧ x$1 = b$1 ∧ (a$2 ≤ x$2 ∧ x$2 ≤ b$2 ∨ b$2 ≤ x$2 ∧ x$2 ≤
a$2)
  (is _ = ?R)
  proof -
    let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
    {
      presume ?L ⟹ ?R and ?R ⟹ ?L
      then show ?thesis
        unfolding closed_segment_def mem_Collect_eq
        unfolding vec_eq_iff forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
        by blast
    }
    {
      assume ?L
      then obtain u where u:
        x $ 1 = (1 - u) * a $ 1 + u * b $ 1
        x $ 2 = (1 - u) * a $ 2 + u * b $ 2
        0 ≤ u u ≤ 1
      by blast
      { fix b a
        assume b + u * a > a + u * b
        then have (1 - u) * b > (1 - u) * a
          by (auto simp add:field_simps)
        then have b ≥ a
          using not_less_iff_gr_or_eq u(4) by fastforce
        then have u * a ≤ u * b
          by (simp add: mult_left_mono u(3))
      }
    }
  moreover

```

```

{ fix a b
  assume u * b > u * a
  then have (1 - u) * a ≤ (1 - u) * b
    using less_eq_real_def u(3) u(4) by force
  then have a + u * b ≤ b + u * a
    by (auto simp add: field_simps)
} ultimately show ?R
  by (force simp add: u assms field_simps not_le)
}
{
  assume ?R
  then show ?L
  proof (cases x$2 = b$2)
    case True
    with ⟨?R⟩ show ?L
      by (rule_tac x=(x$2 - a$2) / (b$2 - a$2) in exI) (auto simp add:
field_simps)
    next
    case False
    with ⟨?R⟩ show ?L
      by (rule_tac x=1 - (x$2 - b$2) / (a$2 - b$2) in exI) (auto simp add:
field_simps)
  qed
}
qed

```

Essentially duplicate proof that could be done by swapping co-ordinates

lemma *segment_horizontal*:

```

fixes a :: real^2
assumes a$2 = b$2
shows x ∈ closed_segment a b ↔
  x$2 = a$2 ∧ x$2 = b$2 ∧ (a$1 ≤ x$1 ∧ x$1 ≤ b$1 ∨ b$1 ≤ x$1 ∧ x$1 ≤
a$1)
(is _ = ?R)
proof -
  let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
  {
    presume ?L ⇒ ?R and ?R ⇒ ?L
    then show ?thesis
      unfolding closed_segment_def mem_Collect_eq
      unfolding vec_eq_iff_forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
      by blast
  }
  {
    assume ?L
    then obtain u where u:
      x $ 1 = (1 - u) * a $ 1 + u * b $ 1
      x $ 2 = (1 - u) * a $ 2 + u * b $ 2
  }

```



```

    0 ≤ u u ≤ 1
  by blast
{ fix b a
  assume b + u * a > a + u * b
  then have (1 - u) * b > (1 - u) * a
    by (auto simp add: field_simps)
  then have b ≥ a
    by (smt (verit, best) mult_left_mono u(4))
  then have u * a ≤ u * b
    by (simp add: mult_left_mono u(3))
}
moreover
{ fix a b
  assume u * b > u * a
  then have (1 - u) * a ≤ (1 - u) * b
    using less_eq_real_def u(3) u(4) by force
  then have a + u * b ≤ b + u * a
    by (auto simp add: field_simps)
}
ultimately show ?R
  by (force simp add: u assms field_simps not_le intro: )
}
{ assume ?R
  then show ?L
  proof (cases x$1 = b$1)
    case True
      with ‹?R› show ?L
        by (rule_tac x=(x$1 - a$1) / (b$1 - a$1) in exI) (auto simp add:
field_simps)
    next
      case False
        with ‹?R› show ?L
          by (rule_tac x=1 - (x$1 - b$1) / (a$1 - b$1) in exI) (auto simp add:
field_simps)
  qed
}
qed

```

9.19.4 Useful Fashoda corollary pointed out to me by Tom Hales

corollary *fashoda_interlace*:

```

fixes a :: real^2
assumes path f
  and path g
  and paf: path_image f ⊆ cbox a b
  and pag: path_image g ⊆ cbox a b
  and (pathstart f)$2 = a$2
  and (pathfinish f)$2 = a$2

```

```

    and (pathstart g)$2 = a$2
    and (pathfinish g)$2 = a$2
    and (pathstart f)$1 < (pathstart g)$1
    and (pathstart g)$1 < (pathfinish f)$1
    and (pathfinish f)$1 < (pathfinish g)$1
  obtains z where z ∈ path_image f and z ∈ path_image g
proof -
  have cbox a b ≠ {}
    using path_image_nonempty[of f] using assms(3) by auto
  note ab=this[unfolded interval_eq_empty_cart not_ex forall_2 not_less]
  have pathstart f ∈ cbox a b
    and pathfinish f ∈ cbox a b
    and pathstart g ∈ cbox a b
    and pathfinish g ∈ cbox a b
    using pathstart_in_path_image pathfinish_in_path_image
    using assms(3-4)
    by auto
  note startfin = this[unfolded mem_box_cart forall_2]
  let ?P1 = linepath (vector[a$1 - 2, a$2 - 2]) (vector[(pathstart f)$1, a$2 - 2])
  +++
  linepath(vector[(pathstart f)$1, a$2 - 2])(pathstart f) +++ f +++
  linepath(pathfinish f)(vector[(pathfinish f)$1, a$2 - 2]) +++
  linepath(vector[(pathfinish f)$1, a$2 - 2])(vector[b$1 + 2, a$2 - 2])
  let ?P2 = linepath(vector[(pathstart g)$1, (pathstart g)$2 - 3])(pathstart g)
  +++ g +++
  linepath(pathfinish g)(vector[(pathfinish g)$1, a$2 - 1]) +++
  linepath(vector[(pathfinish g)$1, a$2 - 1])(vector[b$1 + 1, a$2 - 1]) +++
  linepath(vector[b$1 + 1, a$2 - 1])(vector[b$1 + 1, b$2 + 3])
  let ?a = vector[a$1 - 2, a$2 - 3]
  let ?b = vector[b$1 + 2, b$2 + 3]
  have P1P2: path_image ?P1 = path_image (linepath (vector[a$1 - 2, a$2 - 2])
  (vector[(pathstart f)$1, a$2 - 2])) ∪
  path_image (linepath(vector[(pathstart f)$1, a$2 - 2])(pathstart f)) ∪ path_image
  f ∪
  path_image (linepath(pathfinish f)(vector[(pathfinish f)$1, a$2 - 2])) ∪
  path_image (linepath(vector[(pathfinish f)$1, a$2 - 2])(vector[b$1 + 2, a$2 - 2]))
  path_image ?P2 = path_image(linepath(vector[(pathstart g)$1, (pathstart g)$2 - 3])(pathstart g)) ∪ path_image g ∪
  path_image(linepath(pathfinish g)(vector[(pathfinish g)$1, a$2 - 1])) ∪
  path_image(linepath(vector[(pathfinish g)$1, a$2 - 1])(vector[b$1 + 1, a$2 - 1])) ∪
  path_image(linepath(vector[b$1 + 1, a$2 - 1])(vector[b$1 + 1, b$2 + 3]))
using assms(1-2)
  by(auto simp add: path_image_join)
  have abab: cbox a b ⊆ cbox ?a ?b
    unfolding interval_cbox_cart[symmetric]
    by (auto simp add: less_eq_vec_def forall_2)
  obtain z where

```

```

  z ∈ path_image
    (linepath (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f $ 1, a $ 2
- 2])) +++
    linepath (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f) +++
    f +++
    linepath (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) +++
    linepath (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2, a $ 2
- 2]))
  z ∈ path_image
    (linepath (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart g) +++
    g +++
    linepath (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1]) +++
    linepath (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1, a $ 2
- 1])) +++
    linepath (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $ 2 + 3]))
  apply (rule fashoda[of ?P1 ?P2 ?a ?b])
  unfolding pathstart_join pathfinish_join pathstart_linepath pathfinish_linepath
vector_2
  proof -
    show path ?P1 and path ?P2
      using assms by auto
    show path_image ?P1 ⊆ cbox ?a ?b path_image ?P2 ⊆ cbox ?a ?b
      unfolding P1P2 path_image_linepath using startfin paf pag
      by (auto simp: mem_box_cart segment_horizontal segment_vertical forall_2)
    show a $ 1 - 2 = a $ 1 - 2
      and b $ 1 + 2 = b $ 1 + 2
      and pathstart g $ 2 - 3 = a $ 2 - 3
      and b $ 2 + 3 = b $ 2 + 3
      by (auto simp add: assms)
  qed
  note z=this[unfolded P1P2 path_image_linepath]
  show thesis
  proof (rule that[of z])
    have (z ∈ closed_segment (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f
$ 1, a $ 2 - 2])) ∨
      z ∈ closed_segment (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f)) ∨
      z ∈ closed_segment (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) ∨
      z ∈ closed_segment (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2,
a $ 2 - 2])) ⇒
      (((z ∈ closed_segment (vector [pathstart g $ 1, pathstart g $ 2 - 3]) (pathstart
g)) ∨
        z ∈ closed_segment (pathfinish g) (vector [pathfinish g $ 1, a $ 2 - 1])) ∨
        z ∈ closed_segment (vector [pathfinish g $ 1, a $ 2 - 1]) (vector [b $ 1 + 1,
a $ 2 - 1])) ∨
        z ∈ closed_segment (vector [b $ 1 + 1, a $ 2 - 1]) (vector [b $ 1 + 1, b $
2 + 3])) ⇒ False
    proof (simp only: segment_vertical segment_horizontal vector_2, goal_cases)
      case prems: 1
      have pathfinish f ∈ cbox a b

```

```

    using assms(3) pathfinish_in_path_image[of f] by auto
  then have  $1 + b \ $ 1 \leq \text{pathfinish } f \ $ 1 \implies \text{False}$ 
    unfolding mem_box_cart forall_2 by auto
  then have  $z \ $ 1 \neq \text{pathfinish } f \ $ 1$ 
    using assms(10) assms(11) prems(2) by auto
  moreover have  $\text{pathstart } f \in \text{cbox } a \ b$ 
    using assms(3) pathstart_in_path_image[of f]
    by auto
  then have  $1 + b \ $ 1 \leq \text{pathstart } f \ $ 1 \implies \text{False}$ 
    unfolding mem_box_cart forall_2
    by auto
  then have  $z \ $ 1 \neq \text{pathstart } f \ $ 1$ 
    using prems(2) using assms ab
    by (auto simp add: field_simps)
  ultimately have *:  $z \ $ 2 = a \ $ 2 - 2$ 
    using prems(1) by auto
  have  $z \ $ 1 \neq \text{pathfinish } g \ $ 1$ 
    using prems(2) assms ab
    by (auto simp add: field_simps *)
  moreover have  $\text{pathstart } g \in \text{cbox } a \ b$ 
    using assms(4) pathstart_in_path_image[of g]
    by auto
  note this[unfolded mem_box_cart forall_2]
  then have  $z \ $ 1 \neq \text{pathstart } g \ $ 1$ 
    using prems(1) assms ab
    by (auto simp add: field_simps *)
  ultimately have  $a \ $ 2 - 1 \leq z \ $ 2 \wedge z \ $ 2 \leq b \ $ 2 + 3 \vee b \ $ 2 + 3 \leq z \ $ 2 \wedge z \ $ 2 \leq a \ $ 2 - 1$ 
    using prems(2) unfolding * assms by (auto simp add: field_simps)
  then show False
    unfolding * using ab by auto
qed
then have  $z \in \text{path\_image } f \vee z \in \text{path\_image } g$ 
  using z unfolding Un_iff by blast
then have  $z': z \in \text{cbox } a \ b$ 
  using assms(3-4) by auto
have  $a \ $ 2 = z \ $ 2 \implies (z \ $ 1 = \text{pathstart } f \ $ 1 \vee z \ $ 1 = \text{pathfinish } f \ $ 1)$ 
 $\implies$ 
   $z = \text{pathstart } f \vee z = \text{pathfinish } f$ 
  unfolding vec_eq_iff forall_2 assms
  by auto
with z' show  $z \in \text{path\_image } f$ 
  using z(1)
  unfolding Un_iff mem_box_cart forall_2
  using assms(5) assms(6) segment_horizontal segment_vertical by auto
have  $a \ $ 2 = z \ $ 2 \implies (z \ $ 1 = \text{pathstart } g \ $ 1 \vee z \ $ 1 = \text{pathfinish } g \ $ 1)$ 
 $\implies$ 
   $z = \text{pathstart } g \vee z = \text{pathfinish } g$ 
  unfolding vec_eq_iff forall_2 assms

```

```

    by auto
  with z' show z ∈ path_image g
  using z(2)
  unfolding Un_iff mem_box_cart forall_2
  using assms(7) assms(8) segment_horizontal segment_vertical by auto
qed
qed
end

```

9.20 Vector Cross Products in 3 Dimensions

```

theory Cross3
  imports Determinants Cartesian_Euclidean_Space
begin

context includes no_Set_Product_syntax
begin — locally disable syntax for set product, to avoid warnings

definition cross3 :: [real^3, real^3] ⇒ real^3 (infixr × 80)
  where a × b ≡
    vector [a$2 * b$3 - a$3 * b$2,
            a$3 * b$1 - a$1 * b$3,
            a$1 * b$2 - a$2 * b$1]

end

bundle cross3_syntax begin
notation cross3 (infixr × 80)
no_notation Product_Type.Times (infixr × 80)
end

bundle no_cross3_syntax begin
no_notation cross3 (infixr × 80)
notation Product_Type.Times (infixr × 80)
end

unbundle cross3_syntax

```

9.20.1 Basic lemmas

```

lemmas cross3_simps = cross3_def inner_vec_def sum_3 det_3 vec_eq_iff vector_def algebra_simps

```

```

lemma dot_cross_self: x · (x × y) = 0 x · (y × x) = 0 (x × y) · y = 0 (y × x) · y = 0
  by (simp_all add: orthogonal_def cross3_simps)

```

```

lemma orthogonal_cross: orthogonal (x × y) x orthogonal (x × y) y

```

3338

orthogonal y (x × y) orthogonal (x × y) x
by (*simp_all add: orthogonal_def dot_cross_self*)

lemma *cross_zero_left [simp]: 0 × x = 0 and cross_zero_right [simp]: x × 0 = 0 for x::real^3*
by (*simp_all add: cross3_simps*)

lemma *cross_skew: (x × y) = -(y × x) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_refl [simp]: x × x = 0 for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_add_left: (x + y) × z = (x × z) + (y × z) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_add_right: x × (y + z) = (x × y) + (x × z) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_mult_left: (c *_R x) × y = c *_R (x × y) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_mult_right: x × (c *_R y) = c *_R (x × y) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_minus_left [simp]: (-x) × y = -(x × y) for x::real^3*
by (*simp add: cross3_simps*)

lemma *cross_minus_right [simp]: x × -y = -(x × y) for x::real^3*
by (*simp add: cross3_simps*)

lemma *left_diff_distrib: (x - y) × z = x × z - y × z for x::real^3*
by (*simp add: cross3_simps*)

lemma *right_diff_distrib: x × (y - z) = x × y - x × z for x::real^3*
by (*simp add: cross3_simps*)

hide_fact (**open**) *left_diff_distrib right_diff_distrib*

proposition *Jacobi: x × (y × z) + y × (z × x) + z × (x × y) = 0 for x::real^3*
by (*simp add: cross3_simps*)

proposition *Lagrange: x × (y × z) = (x · z) *_R y - (x · y) *_R z*
by (*simp add: cross3_simps*) (*metis (full_types) exhaust_3*)

proposition *cross_triple: (x × y) · z = (y × z) · x*
by (*simp add: cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps*)

lemma *cross_components:*

$(x \times y)_1 = x_2 * y_3 - y_2 * x_3$ $(x \times y)_2 = x_3 * y_1 - y_3 * x_1$ $(x \times y)_3 = x_1 * y_2 - y_1 * x_2$

by (simp_all add: cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps)

lemma cross_basis: $(axis\ 1\ 1) \times (axis\ 2\ 1) = axis\ 3\ 1$ $(axis\ 2\ 1) \times (axis\ 1\ 1) = -(axis\ 3\ 1)$

$(axis\ 2\ 1) \times (axis\ 3\ 1) = axis\ 1\ 1$ $(axis\ 3\ 1) \times (axis\ 2\ 1) = -(axis\ 1\ 1)$

$(axis\ 3\ 1) \times (axis\ 1\ 1) = axis\ 2\ 1$ $(axis\ 1\ 1) \times (axis\ 3\ 1) = -(axis\ 2\ 1)$

using exhaust_3

by (force simp add: axis_def cross3_simps)+

lemma cross_basis_nonzero:

$u \neq 0 \implies u \times axis\ 1\ 1 \neq 0 \vee u \times axis\ 2\ 1 \neq 0 \vee u \times axis\ 3\ 1 \neq 0$

by (clarsimp simp add: axis_def cross3_simps) (metis exhaust_3)

lemma cross_dot_cancel:

fixes $x::real^3$

assumes $deq: x \cdot y = x \cdot z$ and $veq: x \times y = x \times z$ and $x: x \neq 0$

shows $y = z$

proof –

have $x \cdot x \neq 0$

by (simp add: x)

then have $y - z = 0$

using veq

by (metis (no_types, lifting) Cross3.right_diff_distrib Lagrange deq eq_iff_diff_eq_0 inner_diff_right scale_eq_0_iff)

then show ?thesis

using eq_iff_diff_eq_0 by blast

qed

lemma norm_cross_dot: $(\text{norm } (x \times y))^2 + (x \cdot y)^2 = (\text{norm } x * \text{norm } y)^2$

unfolding power2_norm_eq_inner power_mult_distrib

by (simp add: cross3_simps power2_eq_square)

lemma dot_cross_det: $x \cdot (y \times z) = \det(\text{vector}[x,y,z])$

by (simp add: cross3_simps)

lemma cross_cross_det: $(w \times x) \times (y \times z) = \det(\text{vector}[w,x,z]) *_R y - \det(\text{vector}[w,x,y]) *_R z$

using exhaust_3 by (force simp add: cross3_simps)

proposition dot_cross: $(w \times x) \cdot (y \times z) = (w \cdot y) * (x \cdot z) - (w \cdot z) * (x \cdot y)$

by (force simp add: cross3_simps)

proposition norm_cross: $(\text{norm } (x \times y))^2 = (\text{norm } x)^2 * (\text{norm } y)^2 - (x \cdot y)^2$

unfolding power2_norm_eq_inner power_mult_distrib

by (simp add: cross3_simps power2_eq_square)

```

lemma cross_eq_0:  $x \times y = 0 \iff \text{collinear}\{0, x, y\}$ 
proof -
  have  $x \times y = 0 \iff \text{norm } (x \times y) = 0$ 
    by simp
  also have  $\dots \iff (\text{norm } x * \text{norm } y)^2 = (x \cdot y)^2$ 
    using norm_cross [of x y] by (auto simp: power_mult_distrib)
  also have  $\dots \iff |x \cdot y| = \text{norm } x * \text{norm } y$ 
    using power2_eq_iff
  by (metis (mono_tags, opaque_lifting) abs_minus abs_norm_cancel abs_power2
norm_mult power_abs real_norm_def)
  also have  $\dots \iff \text{collinear } \{0, x, y\}$ 
    by (rule norm_cauchy_schwarz_equal)
  finally show ?thesis .
qed

```

```

lemma cross_eq_self:  $x \times y = x \iff x = 0 \vee x \times y = y \iff y = 0$ 
apply (metis cross_zero_left dot_cross_self(1) inner_eq_zero_iff)
by (metis cross_zero_right dot_cross_self(2) inner_eq_zero_iff)

```

```

lemma norm_and_cross_eq_0:
   $x \cdot y = 0 \wedge x \times y = 0 \iff x = 0 \vee y = 0$  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (metis cross_dot_cancel cross_zero_right inner_zero_right)
qed auto

```

```

lemma bilinear_cross: bilinear( $\times$ )
apply (auto simp add: bilinear_def linear_def)
apply unfold_locales
apply (simp add: cross_add_right)
apply (simp add: cross_mult_right)
apply (simp add: cross_add_left)
apply (simp add: cross_mult_left)
done

```

9.20.2 Preservation by rotation, or other orthogonal transformation up to sign

```

lemma cross_matrix_mult:  $\text{transpose } A * v ((A * v x) \times (A * v y)) = \det A *_R (x \times y)$ 
apply (simp add: vec_eq_iff )
apply (simp add: vector_matrix_mult_def matrix_vector_mult_def forall_3
cross3_simps)
done

```

```

lemma cross_orthogonal_matrix:
  assumes orthogonal_matrix A

```



```

  shows  $(A *v x) \times (A *v y) = \det A *R (A *v (x \times y))$ 
proof -
  have mat 1 = transpose (A ** transpose A)
    by (metis (no_types) assms orthogonal_matrix_def transpose_mat)
  then show ?thesis
    by (metis (no_types) vector_matrix_mul_rid vector_transpose_matrix cross_matrix_mult
matrix_vector_mul_assoc matrix_vector_mult_scaleR)
qed

```

```

lemma cross_rotation_matrix: rotation_matrix A  $\implies (A *v x) \times (A *v y) =$ 
 $A *v (x \times y)$ 
  by (simp add: rotation_matrix_def cross_orthogonal_matrix)

```

```

lemma cross_rotoinversion_matrix: rotoinversion_matrix A  $\implies (A *v x) \times (A$ 
 $*v y) = - A *v (x \times y)$ 
  by (simp add: rotoinversion_matrix_def cross_orthogonal_matrix scaleR_matrix_vector_assoc)

```

```

lemma cross_orthogonal_transformation:
  assumes orthogonal_transformation f
  shows  $(f x) \times (f y) = \det(\text{matrix } f) *R f(x \times y)$ 
proof -
  have orth: orthogonal_matrix (matrix f)
    using assms orthogonal_transformation_matrix by blast
  have matrix f *v z = f z for z
    using assms orthogonal_transformation_matrix by force
  with cross_orthogonal_matrix [OF orth] show ?thesis
    by simp
qed

```

```

lemma cross_linear_image:
   $\llbracket \text{linear } f; \bigwedge x. \text{norm}(f x) = \text{norm } x; \det(\text{matrix } f) = 1 \rrbracket$ 
 $\implies (f x) \times (f y) = f(x \times y)$ 
  by (simp add: cross_orthogonal_transformation orthogonal_transformation)

```

9.20.3 Continuity

```

lemma continuous_cross:  $\llbracket \text{continuous } F f; \text{continuous } F g \rrbracket \implies \text{continuous } F$ 
 $(\lambda x. (f x) \times (g x))$ 
  apply (subst continuous_componentwise)
  apply (clarsimp simp add: cross3_simps)
  apply (intro continuous_intros; simp)
  done

```

```

lemma continuous_on_cross:
  fixes f :: 'a::t2_space  $\Rightarrow$   $\text{real}^3$ 
  shows  $\llbracket \text{continuous\_on } S f; \text{continuous\_on } S g \rrbracket \implies \text{continuous\_on } S (\lambda x. (f x)$ 
 $\times (g x))$ 
  by (simp add: continuous_on_eq_continuous_within continuous_cross)

```

3342

unbundle *no_cross3_syntax*

end

9.21 Bounded Continuous Functions

theory *Bounded_Continuous_Function*

imports

Topology_Euclidean_Space

Uniform_Limit

begin

9.21.1 Definition

definition *bcontfun* = {*f*. *continuous_on UNIV f* ∧ *bounded (range f)*}

typedef (**overloaded**) (*'a*, *'b*) *bcontfun* ((*_* ⇒_{*C*} *_*) [*22*, *21*] *21*) =

bcontfun::('a::topological_space ⇒ 'b::metric_space) set

morphisms *apply_bcontfun Bcontfun*

by (*auto intro: continuous_intros simp: bounded_def bcontfun_def*)

declare [[*coercion apply_bcontfun :: ('a::topological_space ⇒_{*C*} 'b::metric_space) ⇒ 'a ⇒ 'b*]]

setup_lifting *type_definition_bcontfun*

lemma *continuous_on_apply_bcontfun[intro, simp]: continuous_on T (apply_bcontfun x)*

and *bounded_apply_bcontfun[intro, simp]: bounded (range (apply_bcontfun x))*

using *apply_bcontfun[of x]*

by (*auto simp: bcontfun_def intro: continuous_on_subset*)

lemma *bcontfun_eqI: (∧x. apply_bcontfun f x = apply_bcontfun g x) ⇒ f = g*

by *transfer auto*

lemma *bcontfunE:*

assumes *f ∈ bcontfun*

obtains *g where f = apply_bcontfun g*

by (*blast intro: apply_bcontfun_cases assms*)

lemma *const_bcontfun: (λx. b) ∈ bcontfun*

by (*auto simp: bcontfun_def image_def*)

lift_definition *const_bcontfun::'b::metric_space ⇒ ('a::topological_space ⇒_{*C*} 'b)*

is *λc _. c*

by (*rule const_bcontfun*)

instantiation *bcontfun :: (topological_space, metric_space) metric_space*

begin

lift_definition *dist_bcontfun* :: $'a \Rightarrow_C 'b \Rightarrow 'a \Rightarrow_C 'b \Rightarrow \text{real}$
is $\lambda f g. (\text{SUP } x. \text{dist } (f x) (g x))$.

definition *uniformity_bcontfun* :: $('a \Rightarrow_C 'b \times 'a \Rightarrow_C 'b)$ filter
where *uniformity_bcontfun* = $(\text{INF } e \in \{0 <.. \}. \text{principal } \{(x, y). \text{dist } x y < e\})$

definition *open_bcontfun* :: $('a \Rightarrow_C 'b)$ set \Rightarrow bool
where *open_bcontfun* $S = (\forall x \in S. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in S)$

lemma *bounded_dist_le_SUP_dist*:
 $\text{bounded } (\text{range } f) \Longrightarrow \text{bounded } (\text{range } g) \Longrightarrow \text{dist } (f x) (g x) \leq (\text{SUP } x. \text{dist } (f x) (g x))$
by $(\text{auto intro!}: \text{cSUP_upper bounded_imp_bdd_above bounded_dist_comp})$

lemma *dist_bounded*:
fixes $f g :: 'a \Rightarrow_C 'b$
shows $\text{dist } (f x) (g x) \leq \text{dist } f g$
by $\text{transfer } (\text{auto intro!}: \text{bounded_dist_le_SUP_dist simp: bcontfun_def})$

lemma *dist_bound*:
fixes $f g :: 'a \Rightarrow_C 'b$
assumes $\bigwedge x. \text{dist } (f x) (g x) \leq b$
shows $\text{dist } f g \leq b$
using *assms*
by $\text{transfer } (\text{auto intro!}: \text{cSUP_least})$

lemma *dist_fun_lt_imp_dist_val_lt*:
fixes $f g :: 'a \Rightarrow_C 'b$
assumes $\text{dist } f g < e$
shows $\text{dist } (f x) (g x) < e$
using *dist_bounded assms* **by** $(\text{rule } \text{le_less_trans})$

instance

proof

fix $f g h :: 'a \Rightarrow_C 'b$

show $\text{dist } f g = 0 \longleftrightarrow f = g$

proof

have $\bigwedge x. \text{dist } (f x) (g x) \leq \text{dist } f g$

by $(\text{rule } \text{dist_bounded})$

also assume $\text{dist } f g = 0$

finally show $f = g$

by $(\text{auto simp: apply_bcontfun_inject[symmetric]})$

qed $(\text{auto simp: dist_bcontfun_def intro!}: \text{cSup_eq})$

show $\text{dist } f g \leq \text{dist } f h + \text{dist } g h$

proof $(\text{rule } \text{dist_bound})$

fix x

have $\text{dist } (f x) (g x) \leq \text{dist } (f x) (h x) + \text{dist } (g x) (h x)$

```

    by (rule dist_triangle2)
  also have  $\text{dist } (f x) (h x) \leq \text{dist } f h$ 
    by (rule dist_bounded)
  also have  $\text{dist } (g x) (h x) \leq \text{dist } g h$ 
    by (rule dist_bounded)
  finally show  $\text{dist } (f x) (g x) \leq \text{dist } f h + \text{dist } g h$ 
    by simp
qed
qed (rule open_bcontfun_def uniformity_bcontfun_def)+

end

lift_definition PiC::'a::topological_space set  $\Rightarrow$  ('a  $\Rightarrow$  'b set)  $\Rightarrow$  ('a  $\Rightarrow_C$  'b::metric_space)
set
is  $\lambda I X. \text{Pi } I X \cap \text{bcontfun}$ 
by auto

lemma mem_PiC_iff:  $x \in \text{PiC } I X \iff \text{apply\_bcontfun } x \in \text{Pi } I X$ 
by transfer simp

lemmas mem_PiCD = mem_PiC_iff[THEN iffD1]
and mem_PiCI = mem_PiC_iff[THEN iffD2]

lemma tendsto_bcontfun_uniform_limit:
fixes  $f::i \Rightarrow 'a::topological\_space \Rightarrow_C 'b::metric\_space$ 
assumes  $(f \longrightarrow l) F$ 
shows uniform_limit UNIV  $f l F$ 
proof (rule uniform_limitI)
fix  $e::real$  assume  $e > 0$ 
from tendstoD[OF assms this] have  $\forall_F x \text{ in } F. \text{dist } (f x) l < e$  .
then show  $\forall_F n \text{ in } F. \forall x \in \text{UNIV}. \text{dist } ((f n) x) (l x) < e$ 
by eventually_elim (auto simp: dist_fun_lt_imp_dist_val_lt)
qed

lemma uniform_limit_tendsto_bcontfun:
fixes  $f::i \Rightarrow 'a::topological\_space \Rightarrow_C 'b::metric\_space$ 
and  $l::'a::topological\_space \Rightarrow_C 'b::metric\_space$ 
assumes uniform_limit UNIV  $f l F$ 
shows  $(f \longrightarrow l) F$ 
proof (rule tendstoI)
fix  $e::real$  assume  $e > 0$ 
then have  $e / 2 > 0$  by simp
from uniform_limitD[OF assms this]
have  $\forall_F i \text{ in } F. \forall x. \text{dist } (f i x) (l x) < e / 2$  by simp
then have  $\forall_F x \text{ in } F. \text{dist } (f x) l \leq e / 2$ 
by eventually_elim (blast intro: dist_bound_less_imp_le)
then show  $\forall_F x \text{ in } F. \text{dist } (f x) l < e$ 
by eventually_elim (use  $\langle 0 < e \rangle$  in auto)
qed

```

```

lemma uniform_limit_bcontfunE:
  fixes  $f :: 'i \Rightarrow 'a :: \text{topological\_space} \Rightarrow_C 'b :: \text{metric\_space}$ 
    and  $l :: 'a :: \text{topological\_space} \Rightarrow 'b :: \text{metric\_space}$ 
  assumes uniform_limit_UNIV  $f \ l \ F \ F \neq \text{bot}$ 
  obtains  $l' :: 'a :: \text{topological\_space} \Rightarrow_C 'b :: \text{metric\_space}$ 
  where  $l = l' (f \longrightarrow l') \ F$ 
by (metis (mono_tags, lifting) always_eventually_apply_bcontfun apply_bcontfun_cases
assms
  bcontfun_def mem_Collect_eq uniform_limit_bounded uniform_limit_tendsto_bcontfun
  uniform_limit_theorem)

```

```

lemma closed_PiC:
  fixes  $I :: 'a :: \text{metric\_space} \text{ set}$ 
    and  $X :: 'a \Rightarrow 'b :: \text{complete\_space} \text{ set}$ 
  assumes  $\bigwedge i. i \in I \implies \text{closed} (X \ i)$ 
  shows closed (PiC  $I \ X$ )
  unfolding closed_sequential_limits
proof safe
  fix  $f \ l$ 
  assume seq:  $\forall n. f \ n \in \text{PiC } I \ X$  and lim:  $f \longrightarrow l$ 
  show  $l \in \text{PiC } I \ X$ 
  proof (safe intro!: mem_PiCI)
    fix  $x$  assume  $x \in I$ 
    then have closed ( $X \ x$ )
      using assms by simp
    moreover have eventually ( $\lambda i. f \ i \ x \in X \ x$ ) sequentially
      using seq  $\langle x \in I \rangle$ 
      by (auto intro!: eventuallyI dest!: mem_PiCD simp: Pi_iff)
    moreover note sequentially_bot
    moreover have  $(\lambda n. (f \ n) \ x) \longrightarrow l \ x$ 
      using tendsto_bcontfun_uniform_limit [OF lim]
      by (rule tendsto_uniform_limitI) simp
    ultimately show  $l \ x \in X \ x$ 
      by (rule Lim_in_closed_set)
  qed
qed

```

9.21.2 Complete Space

```

instance bcontfun :: (metric_space, complete_space) complete_space
proof
  fix  $f :: \text{nat} \Rightarrow ('a, 'b) \text{ bcontfun}$ 
  assume Cauchy  $f$  — Cauchy equals uniform convergence
  then obtain  $g$  where uniform_limit_UNIV  $f \ g$  sequentially
    using uniformly_convergent_eq_cauchy [of  $\lambda \_.$  True  $f$ ]
    unfolding Cauchy_def uniform_limit_sequentially_iff
    by (metis dist_fun_lt_imp_dist_val_lt)

```

from *uniform_limit_bcontfunE*[*OF this sequentially_bot*]
obtain l' **where** $g = \text{apply_bcontfun } l' (f \longrightarrow l')$ **by** *metis*
then show *convergent f*
by (*intro convergentI*)
qed

9.21.3 Supremum norm for a normed vector space

instantiation *bcontfun* :: (*topological_space*, *real_normed_vector*) *real_vector*
begin

lemma *uminus_cont*: $f \in \text{bcontfun} \implies (\lambda x. - f x) \in \text{bcontfun}$ **for** $f :: 'a \Rightarrow 'b$
by (*auto simp: bcontfun_def intro!: continuous_intros*)

lemma *plus_cont*: $f \in \text{bcontfun} \implies g \in \text{bcontfun} \implies (\lambda x. f x + g x) \in \text{bcontfun}$
for $f g :: 'a \Rightarrow 'b$
by (*auto simp: bcontfun_def intro!: continuous_intros bounded_plus_comp*)

lemma *minus_cont*: $f \in \text{bcontfun} \implies g \in \text{bcontfun} \implies (\lambda x. f x - g x) \in \text{bcontfun}$
for $f g :: 'a \Rightarrow 'b$
by (*auto simp: bcontfun_def intro!: continuous_intros bounded_minus_comp*)

lemma *scaleR_cont*: $f \in \text{bcontfun} \implies (\lambda x. a *_R f x) \in \text{bcontfun}$ **for** $f :: 'a \Rightarrow 'b$
by (*auto simp: bcontfun_def intro!: continuous_intros bounded_scaleR_comp*)

lemma *bcontfun_normI*: *continuous_on UNIV* $f \implies (\bigwedge x. \text{norm } (f x) \leq b) \implies f \in \text{bcontfun}$
by (*auto simp: bcontfun_def intro: boundedI*)

lift_definition *uminus_bcontfun*::($'a \Rightarrow_C 'b$) $\Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f x. - f x$
by (*rule uminus_cont*)

lift_definition *plus_bcontfun*::($'a \Rightarrow_C 'b$) $\Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f g x. f x + g x$
by (*rule plus_cont*)

lift_definition *minus_bcontfun*::($'a \Rightarrow_C 'b$) $\Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f g x. f x - g x$
by (*rule minus_cont*)

lift_definition *zero_bcontfun*:: $'a \Rightarrow_C 'b$ **is** $\lambda_. 0$
by (*rule const_bcontfun*)

lemma *const_bcontfun_0_eq_0*[*simp*]: *const_bcontfun* $0 = 0$
by *transfer simp*

lift_definition *scaleR_bcontfun*::*real* $\Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda r g x. r *_R g x$
by (*rule scaleR_cont*)

lemmas [simp] =

const_bcontfun.rep_eq
 uminus_bcontfun.rep_eq
 plus_bcontfun.rep_eq
 minus_bcontfun.rep_eq
 zero_bcontfun.rep_eq
 scaleR_bcontfun.rep_eq

instance

by standard (auto intro!: bcontfun_eqI simp: algebra_simps)

end

instantiation bcontfun :: (topological_space, real_normed_vector) real_normed_vector
begin

definition norm_bcontfun :: ('a, 'b) bcontfun \Rightarrow real
 where norm_bcontfun f = dist f 0

definition sgn (f::('a,'b) bcontfun) = f /_R norm f

instance

proof

fix a :: real

fix f g :: ('a, 'b) bcontfun

show dist f g = norm (f - g)

unfolding norm_bcontfun_def

by transfer (simp add: dist_norm)

show norm (f + g) \leq norm f + norm g

unfolding norm_bcontfun_def

by transfer

(auto intro!: cSUP_least norm_triangle_le add_mono bounded_norm_le_SUP_norm
 simp: dist_norm bcontfun_def)

show norm (a *_R f) = |a| * norm f

unfolding norm_bcontfun_def

apply transfer

by (rule trans[OF continuous_at_Sup_mono[symmetric]])

(auto intro!: monoI mult_left_mono continuous_intros bounded_imp_bdd_above
 simp: bounded_norm_comp bcontfun_def image_comp)

qed (auto simp: norm_bcontfun_def sgn_bcontfun_def)

end

lemma norm_bounded:

fixes f :: ('a::topological_space, 'b::real_normed_vector) bcontfun

shows norm (apply_bcontfun f x) \leq norm f

using dist_bounded[of f x 0]

3348

by (simp add: dist_norm)

lemma norm_bound:
fixes f :: ('a::topological_space, 'b::real_normed_vector) bcontfun
assumes $\bigwedge x. \text{norm} (\text{apply_bcontfun } f \ x) \leq b$
shows $\text{norm } f \leq b$
using dist_bound[of f 0 b] assms
by (simp add: dist_norm)

9.21.4 (bounded) continuous extension

lemma continuous_on_cbox_bcontfunE:
fixes f::'a::euclidean_space \Rightarrow 'b::metric_space
assumes continuous_on (cbox a b) f
obtains g::'a \Rightarrow_C 'b where
 $\bigwedge x. x \in \text{cbox } a \ b \implies g \ x = f \ x$
 $\bigwedge x. g \ x = f \ (\text{clamp } a \ b \ x)$
proof -
define g where $g \equiv \text{ext_cont } f \ a \ b$
have $g \in \text{bcontfun}$
using assms
by (auto intro!: continuous_on_ext_cont simp: g_def bcontfun_def)
(auto simp: g_def ext_cont_def
intro!: clamp_bounded compact_imp_bounded[OF compact_continuous_image]
assms)
then obtain h where $h: g = \text{apply_bcontfun } h$ by (rule bcontfunE)
then have $h \ x = f \ x$ if $x \in \text{cbox } a \ b$ for x
by (auto simp: h[symmetric] g_def that)
moreover
have $h \ x = f \ (\text{clamp } a \ b \ x)$ for x
by (auto simp: h[symmetric] g_def ext_cont_def)
ultimately show ?thesis ..
qed

lifting_update bcontfun.lifting

lifting_forget bcontfun.lifting

end

9.22 Infinite Products

theory Infinite_Products
imports Topology_Euclidean_Space Complex_Transcendental
begin

9.22.1 Preliminaries

lemma sum_le_prod:
fixes f :: 'a \Rightarrow 'b :: linordered_semidom


```

assumes  $\bigwedge x. x \in A \implies f x \geq 0$ 
shows  $\text{sum } f A \leq (\prod x \in A. 1 + f x)$ 
using assms
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
    from insert.hyps have  $\text{sum } f A + f x * (\prod x \in A. 1) \leq (\prod x \in A. 1 + f x) + f x$ 
    *  $(\prod x \in A. 1 + f x)$ 
    by (intro add_mono insert_mult_left_mono prod_mono) (auto intro: insert.prems)
    with insert.hyps show ?case by (simp add: algebra_simps)
qed simp_all

lemma prod_le_exp_sum:
  fixes  $f :: 'a \Rightarrow \text{real}$ 
  assumes  $\bigwedge x. x \in A \implies f x \geq 0$ 
  shows  $\text{prod } (\lambda x. 1 + f x) A \leq \text{exp } (\text{sum } f A)$ 
  using assms
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
    have  $(1 + f x) * (\prod x \in A. 1 + f x) \leq \text{exp } (f x) * \text{exp } (\text{sum } f A)$ 
    using insert.prems by (intro mult_mono insert_prod_nonneg exp_ge_add_one_self)
  auto
  with insert.hyps show ?case by (simp add: algebra_simps exp_add)
qed simp_all

```

```

lemma lim_ln_1_plus_x_over_x_at_0:  $(\lambda x::\text{real}. \ln (1 + x) / x) -0 \rightarrow 1$ 
proof (rule lhopital)
  show  $(\lambda x::\text{real}. \ln (1 + x)) -0 \rightarrow 0$ 
  by (rule tendsto_eq_intros refl | simp)+
  have eventually  $(\lambda x::\text{real}. x \in \{-1/2 <.. < 1/2\})$  (nhds 0)
  by (rule eventually_nhds_in_open) auto
  hence *: eventually  $(\lambda x::\text{real}. x \in \{-1/2 <.. < 1/2\})$  (at 0)
  by (rule filter_leD [rotated]) (simp_all add: at_within_def)
  show eventually  $(\lambda x::\text{real}. ((\lambda x. \ln (1 + x)) \text{has\_field\_derivative } \text{inverse } (1 + x)))$  (at x) (at 0)
  using * by eventually_elim (auto intro!: derivative_eq_intros simp: field_simps)
  show eventually  $(\lambda x::\text{real}. ((\lambda x. x) \text{has\_field\_derivative } 1))$  (at x) (at 0)
  using * by eventually_elim (auto intro!: derivative_eq_intros simp: field_simps)
  show  $\forall_F x \text{ in } \text{at } 0. x \neq 0$  by (auto simp: at_within_def eventually_inf_principal)
  show  $(\lambda x::\text{real}. \text{inverse } (1 + x) / 1) -0 \rightarrow 1$ 
  by (rule tendsto_eq_intros refl | simp)+
qed auto

```

9.22.2 Definitions and basic properties

```

definition raw_has_prod ::  $[\text{nat} \Rightarrow 'a::\{t2\_space, \text{comm\_semiring\_1}\}, \text{nat}, 'a]$ 
 $\Rightarrow \text{bool}$ 
  where raw_has_prod  $f M p \equiv (\lambda n. \prod i \leq n. f (i+M)) \longrightarrow p \wedge p \neq 0$ 

```

The nonzero and zero cases, as in *Complex Analysis* by Joseph Bak and Donald J. Newman, page 241

definition

$has_prod :: (nat \Rightarrow 'a::\{t2_space, comm_semiring_1\}) \Rightarrow 'a \Rightarrow bool$ (**infixr** has'_prod 80)

where $f\ has_prod\ p \equiv raw_has_prod\ f\ 0\ p \vee (\exists i\ q. p = 0 \wedge f\ i = 0 \wedge raw_has_prod\ f\ (Suc\ i)\ q)$

definition $convergent_prod :: (nat \Rightarrow 'a :: \{t2_space, comm_semiring_1\}) \Rightarrow bool$
where

$convergent_prod\ f \equiv \exists M\ p. raw_has_prod\ f\ M\ p$

definition $prodinf :: (nat \Rightarrow 'a::\{t2_space, comm_semiring_1\}) \Rightarrow 'a$
(**binder** \prod 10)

where $prodinf\ f = (THE\ p. f\ has_prod\ p)$

lemmas $prod_defs = raw_has_prod_def\ has_prod_def\ convergent_prod_def\ prodinf_def$

lemma $has_prod_subst[trans]: f = g \Longrightarrow g\ has_prod\ z \Longrightarrow f\ has_prod\ z$
by *simp*

lemma $has_prod_cong: (\bigwedge n. f\ n = g\ n) \Longrightarrow f\ has_prod\ c \longleftrightarrow g\ has_prod\ c$
by *presburger*

lemma $raw_has_prod_nonzero$ [*simp*]: $\neg raw_has_prod\ f\ M\ 0$
by (*simp* *add: raw_has_prod_def*)

lemma $raw_has_prod_eq_0$:

fixes $f :: nat \Rightarrow 'a::\{semidom, t2_space\}$

assumes $p: raw_has_prod\ f\ m\ p$ **and** $i: f\ i = 0\ i \geq m$

shows $p = 0$

proof –

have $eq0: (\prod k \leq n. f\ (k+m)) = 0$ **if** $i - m \leq n$ **for** n

proof –

have $\exists k \leq n. f\ (k + m) = 0$

using i **that** **by** *auto*

then show *?thesis*

by *auto*

qed

have $(\lambda n. \prod i \leq n. f\ (i + m)) \longrightarrow 0$

by (*rule* *LIMSEQ_offset* [**where** $k = i - m$]) (*simp* *add: eq0*)

with p **show** *?thesis*

unfolding $raw_has_prod_def$

using *LIMSEQ_unique* **by** *blast*

qed

lemma $raw_has_prod_Suc$:

$raw_has_prod\ f\ (Suc\ M)\ a \longleftrightarrow raw_has_prod\ (\lambda n. f\ (Suc\ n))\ M\ a$

unfolding $raw_has_prod_def$ **by** *auto*

lemma *has_prod_0_iff*: $f \text{ has_prod } 0 \iff (\exists i. f i = 0 \wedge (\exists p. \text{raw_has_prod } f (Suc\ i) p))$

by (*simp add: has_prod_def*)

lemma *has_prod_unique2*:

fixes $f :: nat \Rightarrow 'a :: \{semidom, t2_space\}$

assumes $f \text{ has_prod } a$ $f \text{ has_prod } b$ **shows** $a = b$

using *assms*

by (*auto simp: has_prod_def raw_has_prod_eq_0*) (*meson raw_has_prod_def sequentially_bot tendsto_unique*)

lemma *has_prod_unique*:

fixes $f :: nat \Rightarrow 'a :: \{semidom, t2_space\}$

shows $f \text{ has_prod } s \implies s = \text{prodinf } f$

by (*simp add: has_prod_unique2 prodinf_def the_equality*)

lemma *has_prod_eq_0_iff*:

fixes $f :: nat \Rightarrow 'a :: \{semidom, comm_semiring_1, t2_space\}$

assumes $f \text{ has_prod } P$

shows $P = 0 \iff 0 \in \text{range } f$

proof

assume $0 \in \text{range } f$

then obtain N **where** $N: f\ N = 0$

by *auto*

have *eventually* $(\lambda n. n > N)$ *at_top*

by (*rule eventually_gt_at_top*)

hence *eventually* $(\lambda n. (\prod k < n. f\ k) = 0)$ *at_top*

by *eventually_elim* (*use N in auto*)

hence $(\lambda n. \prod k < n. f\ k) \longrightarrow 0$

by (*simp add: tendsto_eventually*)

moreover have $(\lambda n. \prod k < n. f\ k) \longrightarrow P$

using *assms* **by** (*metis N calculation prod_defs(2) raw_has_prod_eq_0 zero_le*)

ultimately show $P = 0$

using *tendsto_unique* **by** *force*

qed (*use assms in <auto simp: has_prod_def>*)

lemma *has_prod_0D*:

fixes $f :: nat \Rightarrow 'a :: \{semidom, comm_semiring_1, t2_space\}$

shows $f \text{ has_prod } 0 \implies 0 \in \text{range } f$

using *has_prod_eq_0_iff*[*of f 0*] **by** *auto*

lemma *has_prod_zeroI*:

fixes $f :: nat \Rightarrow 'a :: \{semidom, comm_semiring_1, t2_space\}$

assumes $f \text{ has_prod } P$ $f\ n = 0$

shows $P = 0$

using *assms* **by** (*auto simp: has_prod_eq_0_iff*)

lemma *raw_has_prod_in_Reals*:

assumes *raw_has_prod* (*complex_of_real* $\circ z$) $M\ p$

3352

shows $p \in \mathbb{R}$
using *assms* **by** (*auto simp: raw_has_prod_def real_lim_sequentially*)

lemma *raw_has_prod_of_real_iff*: $\text{raw_has_prod } (\text{complex_of_real } \circ z) M (\text{of_real } p) \longleftrightarrow \text{raw_has_prod } z M p$
by (*auto simp: raw_has_prod_def tendsto_of_real_iff simp flip: of_real_prod*)

lemma *convergent_prod_of_real_iff*: $\text{convergent_prod } (\text{complex_of_real } \circ z) \longleftrightarrow \text{convergent_prod } z$
by (*smt (verit, best) Reals_cases convergent_prod_def raw_has_prod_in_Reals raw_has_prod_of_real_iff*)

lemma *convergent_prod_altdef*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{t2_space, comm_semiring_1}\}$
shows $\text{convergent_prod } f \longleftrightarrow (\exists M L. (\forall n \geq M. f n \neq 0) \wedge (\lambda n. \prod_{i \leq n}. f (i+M)) \longrightarrow L \wedge L \neq 0)$

proof

assume *convergent_prod* f
then obtain $M L$ **where** $*$: $(\lambda n. \prod_{i \leq n}. f (i+M)) \longrightarrow L \wedge L \neq 0$
by (*auto simp: prod_defs*)
have $f i \neq 0$ **if** $i \geq M$ **for** i

proof

assume $f i = 0$
have $*$: *eventually* $(\lambda n. (\prod_{i \leq n}. f (i+M)) = 0)$ *sequentially*
using *eventually_ge_at_top*[*of* $i - M$]

proof *eventually_elim*

case (*elim* n)

with $\langle f i = 0 \rangle$ **and** $\langle i \geq M \rangle$ **show** *?case*

by (*auto intro!: bexI*[*of* $i - M$] *prod_zero*)

qed

have $(\lambda n. (\prod_{i \leq n}. f (i+M))) \longrightarrow 0$

unfolding *filterlim_iff*

by (*auto dest!: eventually_nhds_x_imp_x intro!: eventually_mono*[*OF* $*$])

from *tendsto_unique*[*OF* $_ \text{this } *(1)$] **and** $*(2)$

show *False* **by** *simp*

qed

with $*$ **show** $(\exists M L. (\forall n \geq M. f n \neq 0) \wedge (\lambda n. \prod_{i \leq n}. f (i+M)) \longrightarrow L \wedge L \neq 0)$

by *blast*

qed (*auto simp: prod_defs*)

lemma *raw_has_prod_norm*:

fixes $a :: 'a :: \text{real_normed_field}$

assumes *raw_has_prod* $f M a$

shows *raw_has_prod* $(\lambda n. \text{norm } (f n)) M (\text{norm } a)$

using *assms* **by** (*auto simp: raw_has_prod_def prod_norm tendsto_norm*)

lemma *has_prod_norm*:

fixes $a :: 'a :: \text{real_normed_field}$

```

  assumes  $f: f \text{ has\_prod } a$ 
  shows  $(\lambda n. \text{norm } (f \ n)) \text{ has\_prod } (\text{norm } a)$ 
  using  $f$  [unfolded has_prod_def]
proof (elim disjE exE conjE)
  assume  $f0: \text{raw\_has\_prod } f \ 0 \ a$ 
  then show  $(\lambda n. \text{norm } (f \ n)) \text{ has\_prod } \text{norm } a$ 
    using has_prod_def raw_has_prod_norm by blast
next
  fix  $i \ p$ 
  assume  $a = 0$  and  $f \ i = 0$  and  $p: \text{raw\_has\_prod } f \ (\text{Suc } i) \ p$ 
  then have  $Ex (\text{raw\_has\_prod } (\lambda n. \text{norm } (f \ n)) (\text{Suc } i))$ 
    using raw_has_prod_norm by blast
  then show ?thesis
    by (metis  $\langle a = 0 \rangle \langle f \ i = 0 \rangle \text{ has\_prod\_0\_iff norm\_zero}$ )
qed

```

9.22.3 Absolutely convergent products

definition $\text{abs_convergent_prod} :: (\text{nat} \Rightarrow _) \Rightarrow \text{bool}$ **where**
 $\text{abs_convergent_prod } f \longleftrightarrow \text{convergent_prod } (\lambda i. 1 + \text{norm } (f \ i - 1))$

lemma $\text{abs_convergent_prodI}$:
assumes $\text{convergent } (\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f \ i - 1))$
shows $\text{abs_convergent_prod } f$
proof –
from assms **obtain** L **where** $L: (\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f \ i - 1)) \longrightarrow L$
by ($\text{auto simp: convergent_def}$)
have $L \geq 1$
proof (rule tendsto_le)
show $\text{eventually } (\lambda n. (\prod_{i \leq n}. 1 + \text{norm } (f \ i - 1)) \geq 1)$ sequentially
proof ($\text{intro always_eventually allI}$)
fix n
have $(\prod_{i \leq n}. 1 + \text{norm } (f \ i - 1)) \geq (\prod_{i \leq n}. 1)$
by (intro prod_mono) auto
thus $(\prod_{i \leq n}. 1 + \text{norm } (f \ i - 1)) \geq 1$ **by** simp
qed
qed ($\text{use } L \text{ in simp_all}$)
hence $L \neq 0$ **by** auto
with L **show** ?thesis **unfolding** $\text{abs_convergent_prod_def prod_defs}$
by ($\text{intro exI[of_ } 0::\text{nat}] \text{ exI[of_ } L]$) auto
qed

lemma
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{topological_semigroup_mult}, t2_space, \text{idom}\}$
assumes $\text{convergent_prod } f$
shows $\text{convergent_prod_imp_convergent: convergent } (\lambda n. \prod_{i \leq n}. f \ i)$
and $\text{convergent_prod_to_zero_iff [simp]: } (\lambda n. \prod_{i \leq n}. f \ i) \longrightarrow 0 \longleftrightarrow$
 $(\exists i. f \ i = 0)$
proof –

```

from assms obtain  $M L$ 
  where  $M: \bigwedge n. n \geq M \implies f n \neq 0$  and  $(\lambda n. \prod_{i \leq n}. f (i + M)) \longrightarrow L$ 
and  $L \neq 0$ 
  by (auto simp: convergent_prod_altdef)
  note this(2)
  also have  $(\lambda n. \prod_{i \leq n}. f (i + M)) = (\lambda n. \prod_{i=M..M+n}. f i)$ 
  by (intro ext prod.reindex_bij_witness[of _  $\lambda n. n - M$   $\lambda n. n + M$ ]) auto
  finally have  $(\lambda n. (\prod_{i < M}. f i) * (\prod_{i=M..M+n}. f i)) \longrightarrow (\prod_{i < M}. f i) * L$ 
  by (intro tendsto_mult tendsto_const)
  also have  $(\lambda n. (\prod_{i < M}. f i) * (\prod_{i=M..M+n}. f i)) = (\lambda n. (\prod_{i \in \{..<M\} \cup \{M..M+n\}}. f i))$ 
  f i)
  by (subst prod.union_disjoint) auto
  also have  $(\lambda n. \{..<M\} \cup \{M..M+n\}) = (\lambda n. \{..n+M\})$  by auto
  finally have lim:  $(\lambda n. \text{prod } f \ \{..n\}) \longrightarrow \text{prod } f \ \{..<M\} * L$ 
  by (rule LIMSEQ_offset)
  thus convergent  $(\lambda n. \prod_{i \leq n}. f i)$ 
  by (auto simp: convergent_def)

```

```

show  $(\lambda n. \prod_{i \leq n}. f i) \longrightarrow 0 \iff (\exists i. f i = 0)$ 

```

```

proof

```

```

  assume  $\exists i. f i = 0$ 

```

```

  then obtain  $i$  where  $f i = 0$  by auto

```

```

  moreover with  $M$  have  $i < M$  by (cases  $i < M$ ) auto

```

```

  ultimately have  $(\prod_{i < M}. f i) = 0$  by auto

```

```

  with lim show  $(\lambda n. \prod_{i \leq n}. f i) \longrightarrow 0$  by simp

```

```

next

```

```

  assume  $(\lambda n. \prod_{i \leq n}. f i) \longrightarrow 0$ 

```

```

  from tendsto_unique[OF _ this lim] and  $\langle L \neq 0 \rangle$ 

```

```

  show  $\exists i. f i = 0$  by auto

```

```

qed

```

```

qed

```

```

lemma convergent_prod_iff_nz_lim:

```

```

  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{topological\_semigroup\_mult}, t2\_space, \text{idom}\}$ 

```

```

  assumes  $\bigwedge i. f i \neq 0$ 

```

```

  shows convergent_prod  $f \iff (\exists L. (\lambda n. \prod_{i \leq n}. f i) \longrightarrow L \wedge L \neq 0)$ 

```

```

  (is ?lhs  $\iff$  ?rhs)

```

```

proof

```

```

  assume ?lhs then show ?rhs

```

```

  using assms convergentD convergent_prod_imp_convergent convergent_prod_to_zero_iff

```

```

by blast

```

```

next

```

```

  assume ?rhs then show ?lhs

```

```

  unfolding prod_defs

```

```

  by (rule_tac  $x=0$  in exI) auto

```

```

qed

```

```

lemma convergent_prod_iff_convergent:

```

```

  fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{topological\_semigroup\_mult}, t2\_space, \text{idom}\}$ 

```

assumes $\bigwedge i. f\ i \neq 0$
shows $\text{convergent_prod } f \longleftrightarrow \text{convergent } (\lambda n. \prod_{i \leq n}. f\ i) \wedge \text{lim } (\lambda n. \prod_{i \leq n}. f\ i) \neq 0$
by (force simp: convergent_prod_iff_nz_lim assms convergent_def limI)

lemma bounded_imp_convergent_prod:

fixes $a :: \text{nat} \Rightarrow \text{real}$

assumes $1: \bigwedge n. a\ n \geq 1$ **and** bounded: $\bigwedge n. (\prod_{i \leq n}. a\ i) \leq B$

shows convergent_prod a

proof –

have bdd_above (range($\lambda n. \prod_{i \leq n}. a\ i$))

by (meson bdd_aboveI2 bounded)

moreover **have** incseq ($\lambda n. \prod_{i \leq n}. a\ i$)

unfolding mono_def **by** (metis 1 prod_mono2 atMost_subset_iff dual_order.trans finite_atMost zero_le_one)

ultimately obtain p **where** $p: (\lambda n. \prod_{i \leq n}. a\ i) \longrightarrow p$

using LIMSEQ_incseq_SUP **by** blast

then **have** $p \neq 0$

by (metis 1 not_one_le_zero prod_ge_1 LIMSEQ_le_const)

with 1 p **show** ?thesis

by (metis convergent_prod_iff_nz_lim not_one_le_zero)

qed

lemma abs_convergent_prod_altdef:

fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{one}, \text{real_normed_vector}\}$

shows $\text{abs_convergent_prod } f \longleftrightarrow \text{convergent } (\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1))$

proof

assume abs_convergent_prod f

thus convergent ($\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ i - 1)$)

by (auto simp: abs_convergent_prod_def intro!: convergent_prod_imp_convergent)

qed (auto intro: abs_convergent_prodI)

lemma Weierstrass_prod_ineq:

fixes $f :: 'a \Rightarrow \text{real}$

assumes $\bigwedge x. x \in A \implies f\ x \in \{0..1\}$

shows $1 - \text{sum } f\ A \leq (\prod_{x \in A}. 1 - f\ x)$

using assms

proof (induction A rule: infinite_finite_induct)

case (insert x A)

from insert.hyps **and** insert.prem

have $1 - \text{sum } f\ A + f\ x * (\prod_{x \in A}. 1 - f\ x) \leq (\prod_{x \in A}. 1 - f\ x) + f\ x * (\prod_{x \in A}. 1)$

by (intro insert.IH add_mono mult_left_mono prod_mono) auto

with insert.hyps **show** ?case **by** (simp add: algebra_simps)

qed simp_all

lemma norm_prod_minus1_le_prod_minus1:

```

fixes f :: nat => 'a :: {real_normed_div_algebra,comm_ring_1}
shows norm (prod (λn. 1 + f n) A - 1) ≤ prod (λn. 1 + norm (f n)) A - 1
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
  from insert.hyps have
    norm ((∏ n∈insert x A. 1 + f n) - 1) =
      norm ((∏ n∈A. 1 + f n) - 1 + f x * (∏ n∈A. 1 + f n))
  by (simp add: algebra_simps)
  also have ... ≤ norm ((∏ n∈A. 1 + f n) - 1) + norm (f x * (∏ n∈A. 1 + f
n))
  by (rule norm_triangle_ineq)
  also have norm (f x * (∏ n∈A. 1 + f n)) = norm (f x) * (∏ x∈A. norm (1 +
f x))
  by (simp add: prod_norm_norm_mult)
  also have (∏ x∈A. norm (1 + f x)) ≤ (∏ x∈A. norm (1::'a) + norm (f x))
  by (intro prod_mono norm_triangle_ineq ballI conjI) auto
  also have norm (1::'a) = 1 by simp
  also note insert.IH
  also have (∏ n∈A. 1 + norm (f n)) - 1 + norm (f x) * (∏ x∈A. 1 + norm (f
x)) =
    (∏ n∈insert x A. 1 + norm (f n)) - 1
  using insert.hyps by (simp add: algebra_simps)
  finally show ?case by - (simp_all add: mult_left_mono)
qed simp_all

```

```

lemma convergent_prod_imp_ev_nonzero:
fixes f :: nat => 'a :: {t2_space,comm_semiring_1}
assumes convergent_prod f
shows eventually (λn. f n ≠ 0) sequentially
using assms by (auto simp: eventually_at_top_linorder convergent_prod_altdef)

```

```

lemma convergent_prod_imp_LIMSEQ:
fixes f :: nat => 'a :: {real_normed_field}
assumes convergent_prod f
shows f ⟶ 1
proof -
  from assms obtain M L where L: (λn. ∏ i≤n. f (i+M)) ⟶ L ∧ n. n ≥
M ⟹ f n ≠ 0 L ≠ 0
  by (auto simp: convergent_prod_altdef)
  hence L': (λn. ∏ i≤Suc n. f (i+M)) ⟶ L by (subst filterlim_sequentially_Suc)
  have (λn. (∏ i≤Suc n. f (i+M)) / (∏ i≤n. f (i+M))) ⟶ L / L
  using L L' by (intro tendsto_divide) simp_all
  also from L have L / L = 1 by simp
  also have (λn. (∏ i≤Suc n. f (i+M)) / (∏ i≤n. f (i+M))) = (λn. f (n + Suc
M))
  using assms L by (auto simp: fun_eq_iff atMost_Suc)
  finally show ?thesis by (rule LIMSEQ_offset)
qed

```



```

lemma abs_convergent_prod_imp_summable:
  fixes f :: nat ⇒ 'a :: real_normed_div_algebra
  assumes abs_convergent_prod f
  shows summable (λi. norm (f i - 1))
proof -
  from assms have convergent (λn. ∏ i≤n. 1 + norm (f i - 1))
  unfolding abs_convergent_prod_def by (rule convergent_prod_imp_convergent)
  then obtain L where L: (λn. ∏ i≤n. 1 + norm (f i - 1)) ⟶ L
  unfolding convergent_def by blast
  have convergent (λn. ∑ i≤n. norm (f i - 1))
  proof (rule Bseq_monoseq_convergent)
    have eventually (λn. (∏ i≤n. 1 + norm (f i - 1)) < L + 1) sequentially
    using L(1) by (rule order_tendstoD) simp_all
    hence ∀_F x in sequentially. norm (∑ i≤x. norm (f i - 1)) ≤ L + 1
  proof eventually_elim
    case (elim n)
    have norm (∑ i≤n. norm (f i - 1)) = (∑ i≤n. norm (f i - 1))
    unfolding real_norm_def by (intro abs_of_nonneg sum_nonneg) simp_all
    also have ... ≤ (∏ i≤n. 1 + norm (f i - 1)) by (rule sum_le_prod) auto
    also have ... < L + 1 by (rule elim)
    finally show ?case by simp
  qed
  thus Bseq (λn. ∑ i≤n. norm (f i - 1)) by (rule BfunI)
next
  show monoseq (λn. ∑ i≤n. norm (f i - 1))
  by (rule mono_SucI1) auto
qed
thus summable (λi. norm (f i - 1)) by (simp add: summable_iff_convergent')
qed

lemma summable_imp_abs_convergent_prod:
  fixes f :: nat ⇒ 'a :: real_normed_div_algebra
  assumes summable (λi. norm (f i - 1))
  shows abs_convergent_prod f
proof (intro abs_convergent_prodI Bseq_monoseq_convergent)
  show monoseq (λn. ∏ i≤n. 1 + norm (f i - 1))
  by (intro mono_SucI1)
  (auto simp: atMost_Suc algebra_simps intro!: mult_nonneg_nonneg prod_nonneg)
next
  show Bseq (λn. ∏ i≤n. 1 + norm (f i - 1))
  proof (rule Bseq_eventually_mono)
    show eventually (λn. norm (∏ i≤n. 1 + norm (f i - 1)) ≤
      norm (exp (∑ i≤n. norm (f i - 1)))) sequentially
    by (intro always_eventually_allI) (auto simp: abs_prod exp_sum intro!:
      prod_mono)
  next
    from assms have (λn. ∑ i≤n. norm (f i - 1)) ⟶ (∑ i. norm (f i - 1))
    using sums_def_le by blast
    hence (λn. exp (∑ i≤n. norm (f i - 1))) ⟶ exp (∑ i. norm (f i - 1))

```

```

    by (rule tendsto_exp)
  hence convergent ( $\lambda n. \exp (\sum_{i \leq n}. \text{norm } (f i - 1))$ )
    by (rule convergentI)
  thus Bseq ( $\lambda n. \exp (\sum_{i \leq n}. \text{norm } (f i - 1))$ )
    by (rule convergent_imp_Bseq)
qed

```

```

theorem abs_convergent_prod_conv_summable:
  fixes f :: nat  $\Rightarrow$  'a :: real_normed_div_algebra
  shows abs_convergent_prod f  $\longleftrightarrow$  summable ( $\lambda i. \text{norm } (f i - 1)$ )
  by (blast intro: abs_convergent_prod_imp_summable summable_imp_abs_convergent_prod)

```

```

lemma abs_convergent_prod_imp_LIMSEQ:
  fixes f :: nat  $\Rightarrow$  'a :: {comm_ring_1, real_normed_div_algebra}
  assumes abs_convergent_prod f
  shows f  $\longrightarrow$  1
proof -
  from assms have summable ( $\lambda n. \text{norm } (f n - 1)$ )
    by (rule abs_convergent_prod_imp_summable)
  from summable_LIMSEQ_zero[OF this] have ( $\lambda n. f n - 1$ )  $\longrightarrow$  0
    by (simp add: tendsto_norm_zero_iff)
  from tendsto_add[OF this tendsto_const[of 1]] show ?thesis by simp
qed

```

```

lemma abs_convergent_prod_imp_ev_nonzero:
  fixes f :: nat  $\Rightarrow$  'a :: {comm_ring_1, real_normed_div_algebra}
  assumes abs_convergent_prod f
  shows eventually ( $\lambda n. f n \neq 0$ ) sequentially
proof -
  from assms have f  $\longrightarrow$  1
    by (rule abs_convergent_prod_imp_LIMSEQ)
  hence eventually ( $\lambda n. \text{dist } (f n) 1 < 1$ ) at_top
    by (auto simp: tendsto_iff)
  thus ?thesis by eventually_elim auto
qed

```

9.22.4 Ignoring initial segments

```

lemma convergent_prod_offset:
  assumes convergent_prod ( $\lambda n. f (n + m)$ )
  shows convergent_prod f
proof -
  from assms obtain M L where ( $\lambda n. \prod_{k \leq n}. f (k + (M + m))$ )  $\longrightarrow$  L L  $\neq$ 
  0
    by (auto simp: prod_defs add.assoc)
  thus convergent_prod f
    unfolding prod_defs by blast
qed

```

lemma *abs_convergent_prod_offset*:
assumes *abs_convergent_prod* $(\lambda n. f (n + m))$
shows *abs_convergent_prod* f
using *assms unfolding abs_convergent_prod_def* **by** *(rule convergent_prod_offset)*

lemma *raw_has_prod_ignore_initial_segment*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{real_normed_field}$
assumes *raw_has_prod* $f M p N \geq M$
obtains q **where** *raw_has_prod* $f N q$
proof –
have $p: (\lambda n. \prod_{k \leq n}. f (k + M)) \longrightarrow p$ **and** $p \neq 0$
using *assms by (auto simp: raw_has_prod_def)*
then have $nz: \bigwedge n. n \geq M \implies f n \neq 0$
using *assms by (auto simp: raw_has_prod_eq_0)*
define C **where** $C = (\prod_{k < N - M}. f (k + M))$
from nz **have** *[simp]*: $C \neq 0$
by *(auto simp: C_def)*

from p **have** $(\lambda i. \prod_{k \leq i + (N - M)}. f (k + M)) \longrightarrow p$
by *(rule LIMSEQ_ignore_initial_segment)*
also have $(\lambda i. \prod_{k \leq i + (N - M)}. f (k + M)) = (\lambda n. C * (\prod_{k \leq n}. f (k + N)))$
proof *(rule ext, goal_cases)*
case $(1 n)$
have $\{..n+(N-M)\} = \{..<(N-M)\} \cup \{(N-M)..n+(N-M)\}$ **by** *auto*
also have $(\prod_{k \in \dots}. f (k + M)) = C * (\prod_{k=(N-M)..n+(N-M)}. f (k + M))$
unfolding *C_def* **by** *(rule prod.union_disjoint) auto*
also have $(\prod_{k=(N-M)..n+(N-M)}. f (k + M)) = (\prod_{k \leq n}. f (k + (N - M) + M))$
by *(intro ext prod.reindex_bij_witness[of _ $\lambda k. k + (N - M)$ $\lambda k. k - (N - M)$])*
auto
finally show *?case*
using $\langle N \geq M \rangle$ **by** *(simp add: add_ac)*
qed
finally have $(\lambda n. C * (\prod_{k \leq n}. f (k + N)) / C) \longrightarrow p / C$
by *(intro tendsto_divide tendsto_const) auto*
hence $(\lambda n. \prod_{k \leq n}. f (k + N)) \longrightarrow p / C$ **by** *simp*
moreover from $\langle p \neq 0 \rangle$ **have** $p / C \neq 0$ **by** *simp*
ultimately show *?thesis*
using *raw_has_prod_def* **that** **by** *blast*
qed

corollary *convergent_prod_ignore_initial_segment*:
fixes $f :: \text{nat} \Rightarrow 'a :: \text{real_normed_field}$
assumes *convergent_prod* f
shows *convergent_prod* $(\lambda n. f (n + m))$
using *assms*

```

unfolding convergent_prod_def
apply clarify
apply (erule_tac N=M+m in raw_has_prod_ignore_initial_segment)
apply (auto simp add: raw_has_prod_def add_ac)
done

```

```

corollary convergent_prod_ignore_nonzero_segment:
  fixes f :: nat => 'a :: real_normed_field
  assumes f: convergent_prod f and nz:  $\bigwedge i. i \geq M \implies f\ i \neq 0$ 
  shows  $\exists p. \text{raw\_has\_prod } f\ M\ p$ 
  using convergent_prod_ignore_initial_segment [OF f]
  by (metis convergent_LIMSEQ_iff convergent_prod_iff convergent_le_add_same_cancel2
  nz prod_defs(1) zero_order(1))

```

```

corollary abs_convergent_prod_ignore_initial_segment:
  assumes abs_convergent_prod f
  shows abs_convergent_prod ( $\lambda n. f\ (n + m)$ )
  using assms unfolding abs_convergent_prod_def
  by (rule convergent_prod_ignore_initial_segment)

```

9.22.5 More elementary properties

```

theorem abs_convergent_prod_imp_convergent_prod:
  fixes f :: nat => 'a :: {real_normed_div_algebra, complete_space, comm_ring_1}
  assumes abs_convergent_prod f
  shows convergent_prod f
proof -
  from assms have eventually ( $\lambda n. f\ n \neq 0$ ) sequentially
    by (rule abs_convergent_prod_imp_ev_nonzero)
  then obtain N where N:  $f\ n \neq 0$  if  $n \geq N$  for n
    by (auto simp: eventually_at_top_linorder)
  let ?P =  $\lambda n. \prod_{i \leq n}. f\ (i + N)$  and ?Q =  $\lambda n. \prod_{i \leq n}. 1 + \text{norm } (f\ (i + N) -$ 
  1)

```

```

  have Cauchy ?P
  proof (rule CauchyI', goal_cases)
    case (1  $\varepsilon$ )
    from assms have abs_convergent_prod ( $\lambda n. f\ (n + N)$ )
      by (rule abs_convergent_prod_ignore_initial_segment)
    hence Cauchy ?Q
      unfolding abs_convergent_prod_def
      by (intro convergent_Cauchy convergent_prod_imp_convergent)
    from CauchyD[OF this 1] obtain M where M:  $\text{norm } (?Q\ m - ?Q\ n) < \varepsilon$  if
     $m \geq M\ n \geq M$  for m n
      by blast
    show ?case
  proof (rule exI[of _ M], safe, goal_cases)
    case (1 m n)
    have dist (?P m) (?P n) =  $\text{norm } (?P\ n - ?P\ m)$ 

```

```

    by (simp add: dist_norm norm_minus_commute)
  also from 1 have  $\{..n\} = \{..m\} \cup \{m<..n\}$  by auto
  hence  $\text{norm } (?P\ n - ?P\ m) = \text{norm } (?P\ m * (\prod_{k \in \{m<..n\}}. f\ (k + N)) -$ 
 $?P\ m)$ 
    by (subst prod.union_disjoint [symmetric]) (auto simp: algebra_simps)
  also have  $\dots = \text{norm } (?P\ m * ((\prod_{k \in \{m<..n\}}. f\ (k + N)) - 1))$ 
    by (simp add: algebra_simps)
  also have  $\dots = (\prod_{k \leq m}. \text{norm } (f\ (k + N))) * \text{norm } ((\prod_{k \in \{m<..n\}}. f\ (k$ 
 $+ N)) - 1)$ 
    by (simp add: norm_mult_prod_norm)
  also have  $\dots \leq ?Q\ m * ((\prod_{k \in \{m<..n\}}. 1 + \text{norm } (f\ (k + N) - 1)) - 1)$ 
    using norm_prod_minus1_le_prod_minus1 [of  $\lambda k. f\ (k + N) - 1\ \{m<..n\}$ ]
      norm_triangle_ineq [of  $1\ f\ k - 1$  for  $k$ ]
    by (intro mult_mono prod_mono ballI conjI norm_prod_minus1_le_prod_minus1
prod_nonneg) auto
  also have  $\dots = ?Q\ m * (\prod_{k \in \{m<..n\}}. 1 + \text{norm } (f\ (k + N) - 1)) - ?Q$ 
 $m$ 
    by (simp add: algebra_simps)
  also have  $?Q\ m * (\prod_{k \in \{m<..n\}}. 1 + \text{norm } (f\ (k + N) - 1)) =$ 
 $(\prod_{k \in \{..m\} \cup \{m<..n\}}. 1 + \text{norm } (f\ (k + N) - 1))$ 
    by (rule prod.union_disjoint [symmetric]) auto
  also from 1 have  $\{..m\} \cup \{m<..n\} = \{..n\}$  by auto
  also have  $?Q\ n - ?Q\ m \leq \text{norm } (?Q\ n - ?Q\ m)$  by simp
  also from 1 have  $\dots < \varepsilon$  by (intro M) auto
  finally show ?case .
qed
qed
hence conv: convergent ?P by (rule Cauchy_convergent)
then obtain L where  $L: ?P \longrightarrow L$ 
  by (auto simp: convergent_def)

have  $L \neq 0$ 
proof
  assume [simp]:  $L = 0$ 
  from tendsto_norm[OF L] have limit:  $(\lambda n. \prod_{k \leq n}. \text{norm } (f\ (k + N))) \longrightarrow$ 
 $0$ 
    by (simp add: prod_norm)

  from assms have  $(\lambda n. f\ (n + N)) \longrightarrow 1$ 
  by (intro abs_convergent_prod_imp_LIMSEQ abs_convergent_prod_ignore_initial_segment)
  hence eventually  $(\lambda n. \text{norm } (f\ (n + N) - 1) < 1)$  sequentially
    by (auto simp: tendsto_iff dist_norm)
  then obtain M0 where  $M0: \text{norm } (f\ (n + N) - 1) < 1$  if  $n \geq M0$  for  $n$ 
    by (auto simp: eventually_at_top_linorder)

  {
    fix M assume  $M: M \geq M0$ 
    with M0 have  $M: \text{norm } (f\ (n + N) - 1) < 1$  if  $n \geq M$  for  $n$  using that
  }
by simp

```

```

have ( $\lambda n. \prod k \leq n. 1 - \text{norm } (f (k+M+N) - 1)$ )  $\longrightarrow 0$ 
proof (rule tendsto_sandwich)
  show eventually ( $\lambda n. (\prod k \leq n. 1 - \text{norm } (f (k+M+N) - 1)) \geq 0$ )
sequentially
  using M by (intro always_eventually prod_nonneg allI ballI) (auto intro:
less_imp_le)
  have  $\text{norm } (1::'a) - \text{norm } (f (i + M + N) - 1) \leq \text{norm } (f (i + M +$ 
N)) for i
  using norm_triangle_ineq3[of f (i + M + N) 1] by simp
  thus eventually ( $\lambda n. (\prod k \leq n. 1 - \text{norm } (f (k+M+N) - 1)) \leq (\prod k \leq n.$ 
norm (f (k+M+N)))) at_top
  using M by (intro always_eventually allI prod_mono ballI conjI) (auto
intro: less_imp_le)

define C where  $C = (\prod k < M. \text{norm } (f (k + N)))$ 
from N have [simp]:  $C \neq 0$  by (auto simp: C_def)
from L have ( $\lambda n. \text{norm } (\prod k \leq n+M. f (k + N))$ )  $\longrightarrow 0$ 
by (intro LIMSEQ_ignore_initial_segment) (simp add: tendsto_norm_zero_iff)
also have ( $\lambda n. \text{norm } (\prod k \leq n+M. f (k + N)) = (\lambda n. C * (\prod k \leq n. \text{norm}$ 
(f (k + M + N)))))
proof (rule ext, goal_cases)
  case (1 n)
  have  $\{..n+M\} = \{..<M\} \cup \{M..n+M\}$  by auto
  also have  $\text{norm } (\prod k \in \dots f (k + N)) = C * \text{norm } (\prod k = M..n+M. f (k$ 
+ N))
  unfolding C_def by (subst prod.union_disjoint) (auto simp: norm_mult
prod_norm)
  also have  $(\prod k = M..n+M. f (k + N)) = (\prod k \leq n. f (k + N + M))$ 
  by (intro prod.reindex_bij_witness[of _ \lambda i. i + M \lambda i. i - M]) auto
  finally show ?case by (simp add: add_ac prod_norm)
qed
finally have ( $\lambda n. C * (\prod k \leq n. \text{norm } (f (k + M + N))) / C$ )  $\longrightarrow 0 /$ 
C
  by (intro tendsto_divide tendsto_const) auto
  thus ( $\lambda n. \prod k \leq n. \text{norm } (f (k + M + N))$ )  $\longrightarrow 0$  by simp
qed simp_all

have  $1 - (\sum i. \text{norm } (f (i + M + N) - 1)) \leq 0$ 
proof (rule tendsto_le)
  show eventually ( $\lambda n. 1 - (\sum k \leq n. \text{norm } (f (k+M+N) - 1)) \leq$ 
 $(\prod k \leq n. 1 - \text{norm } (f (k+M+N) - 1))$ ) at_top
  using M by (intro always_eventually allI Weierstrass_prod_ineq) (auto
intro: less_imp_le)
  show ( $\lambda n. \prod k \leq n. 1 - \text{norm } (f (k+M+N) - 1)$ )  $\longrightarrow 0$  by fact
  show ( $\lambda n. 1 - (\sum k \leq n. \text{norm } (f (k + M + N) - 1))$ )
 $\longrightarrow 1 - (\sum i. \text{norm } (f (i + M + N) - 1))$ 
  by (intro tendsto_intros summable_LIMSEQ' summable_ignore_initial_segment)

```

```

      abs_convergent_prod_imp_summable assms)
    qed simp_all
    hence  $(\sum i. \text{norm } (f (i + M + N) - 1)) \geq 1$  by simp
    also have  $\dots + (\sum i < M. \text{norm } (f (i + N) - 1)) = (\sum i. \text{norm } (f (i + N) - 1))$ 
  - 1))
    by (intro suminf_split_initial_segment [symmetric] summable_ignore_initial_segment
      abs_convergent_prod_imp_summable assms)
    finally have  $1 + (\sum i < M. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1))$  by simp
  } note * = this

  have  $1 + (\sum i. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1))$ 
  proof (rule tendsto_le)
    show  $(\lambda M. 1 + (\sum i < M. \text{norm } (f (i + N) - 1))) \longrightarrow 1 + (\sum i. \text{norm } (f (i + N) - 1))$ 
    by (intro tendsto_intros summable_LIMSEQ summable_ignore_initial_segment

      abs_convergent_prod_imp_summable assms)
    show eventually  $(\lambda M. 1 + (\sum i < M. \text{norm } (f (i + N) - 1)) \leq (\sum i. \text{norm } (f (i + N) - 1)))$  at_top
    using eventually_ge_at_top[of M0] by eventually_elim (use * in auto)
  qed simp_all
  thus False by simp
qed
with L show ?thesis by (auto simp: prod_defs)
qed

lemma raw_has_prod_cases:
  fixes f :: nat  $\Rightarrow$  'a :: {idom, topological_semigroup_mult, t2_space}
  assumes raw_has_prod f M p
  obtains i where  $i < M$  f i = 0 | p where raw_has_prod f 0 p
proof -
  have  $(\lambda n. \prod i \leq n. f (i + M)) \longrightarrow p$  p  $\neq 0$ 
  using assms unfolding raw_has_prod_def by blast+
  then have  $(\lambda n. \text{prod } f \{.. < M\} * (\prod i \leq n. f (i + M))) \longrightarrow \text{prod } f \{.. < M\} * p$ 
  by (metis tendsto_mult_left)
  moreover have  $\text{prod } f \{.. < M\} * (\prod i \leq n. f (i + M)) = \text{prod } f \{.. n + M\}$  for n
  proof -
    have  $\{.. n + M\} = \{.. < M\} \cup \{M.. n + M\}$ 
    by auto
    then have  $\text{prod } f \{.. n + M\} = \text{prod } f \{.. < M\} * \text{prod } f \{M.. n + M\}$ 
    by simp (subst prod.union_disjoint; force)
    also have  $\dots = \text{prod } f \{.. < M\} * (\prod i \leq n. f (i + M))$ 
    by (metis (mono_tags, lifting) add.left_neutral atMost_atLeast0 prod.shift_bounds_cl_nat_ivl)
    finally show ?thesis by metis
  qed
  ultimately have  $(\lambda n. \text{prod } f \{.. n\}) \longrightarrow \text{prod } f \{.. < M\} * p$ 
  by (auto intro: LIMSEQ_offset [where k=M])

```

3364

then have *raw_has_prod f 0* (*prod f {..*M*} * *p**) **if** $\forall i < M. f i \neq 0$
using $\langle p \neq 0 \rangle$ *assms* **that by** (*auto simp: raw_has_prod_def*)
then show *thesis*
using *that by blast*

qed

corollary *convergent_prod_offset_0*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological_semigroup_mult}, \text{t2_space}\}$
assumes *convergent_prod f* $\wedge i. f i \neq 0$
shows $\exists p. \text{raw_has_prod } f 0 p$
using *assms convergent_prod_def raw_has_prod_cases* **by blast**

lemma *prodinf_eq_lim*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological_semigroup_mult}, \text{t2_space}\}$
assumes *convergent_prod f* $\wedge i. f i \neq 0$
shows $\text{prodinf } f = \text{lim } (\lambda n. \prod_{i \leq n}. f i)$
using *assms convergent_prod_offset_0 [OF assms]*
by (*simp add: prod_defs lim_def*) (*metis (no_types) assms(1) convergent_prod_to_zero_iff*)

lemma *prodinf_eq_lim'*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{idom}, \text{topological_semigroup_mult}, \text{t2_space}\}$
assumes *convergent_prod f* $\wedge i. f i \neq 0$
shows $\text{prodinf } f = \text{lim } (\lambda n. \prod_{i < n}. f i)$
by (*metis assms prodinf_eq_lim LIMSEQ_lessThan_iff_atMost convergent_prod_iff_nz_lim limI*)

lemma *prodinf_eq_prod_lim*:
fixes $a :: 'a :: \{\text{topological_semigroup_mult}, \text{t2_space}, \text{idom}\}$
assumes $(\lambda n. \prod_{k \leq n}. f k) \longrightarrow a$ $a \neq 0$
shows $(\prod k. f k) = a$
by (*metis LIMSEQ_prod_0 LIMSEQ_unique assms convergent_prod_iff_nz_lim limI prodinf_eq_lim*)

lemma *prodinf_eq_prod_lim'*:
fixes $a :: 'a :: \{\text{topological_semigroup_mult}, \text{t2_space}, \text{idom}\}$
assumes $(\lambda n. \prod_{k < n}. f k) \longrightarrow a$ $a \neq 0$
shows $(\prod k. f k) = a$
using *LIMSEQ_lessThan_iff_atMost assms prodinf_eq_prod_lim* **by blast**

lemma *has_prod_one*[*simp, intro*]: $(\lambda n. 1)$ *has_prod 1*
unfolding *prod_defs* **by auto**

lemma *convergent_prod_one*[*simp, intro*]: *convergent_prod* $(\lambda n. 1)$
unfolding *prod_defs* **by auto**

lemma *prodinf_cong*: $(\wedge n. f n = g n) \implies \text{prodinf } f = \text{prodinf } g$
by *presburger*

lemma *convergent_prod_cong*:


```

fixes  $f\ g :: nat \Rightarrow 'a::\{field, topological\_semigroup\_mult, t2\_space\}$ 
assumes  $ev: eventually (\lambda x. f\ x = g\ x) sequentially$  and  $f: \bigwedge i. f\ i \neq 0$  and  $g: \bigwedge i. g\ i \neq 0$ 
shows  $convergent\_prod\ f = convergent\_prod\ g$ 
proof -
from  $assms$  obtain  $N$  where  $N: \forall n \geq N. f\ n = g\ n$ 
  by ( $auto\ simp: eventually\_at\_top\_linorder$ )
define  $C$  where  $C = (\prod_{k < N}. f\ k / g\ k)$ 
with  $g$  have  $C \neq 0$ 
  by ( $simp\ add: f$ )
have  $*$ :  $eventually (\lambda n. prod\ f\ \{..n\} = C * prod\ g\ \{..n\}) sequentially$ 
  using  $eventually\_ge\_at\_top[of\ N]$ 
proof  $eventually\_elim$ 
  case ( $elim\ n$ )
  then have  $\{..n\} = \{..<N\} \cup \{N..n\}$ 
    by  $auto$ 
  also have  $prod\ f\ \dots = prod\ f\ \{..<N\} * prod\ f\ \{N..n\}$ 
    by ( $intro\ prod.union\_disjoint$ )  $auto$ 
  also from  $N$  have  $prod\ f\ \{N..n\} = prod\ g\ \{N..n\}$ 
    by ( $intro\ prod.cong$ )  $simp\_all$ 
  also have  $prod\ f\ \{..<N\} * prod\ g\ \{N..n\} = C * (prod\ g\ \{..<N\} * prod\ g\ \{N..n\})$ 
    unfolding  $C\_def$  by ( $simp\ add: g\ prod\_dividef$ )
  also have  $prod\ g\ \{..<N\} * prod\ g\ \{N..n\} = prod\ g\ (\{..<N\} \cup \{N..n\})$ 
    by ( $intro\ prod.union\_disjoint\ [symmetric]$ )  $auto$ 
  also from  $elim$  have  $\{..<N\} \cup \{N..n\} = \{..n\}$ 
    by  $auto$ 
  finally show  $prod\ f\ \{..n\} = C * prod\ g\ \{..n\}$  .
qed
then have  $cong: convergent (\lambda n. prod\ f\ \{..n\}) = convergent (\lambda n. C * prod\ g\ \{..n\})$ 
  by ( $rule\ convergent\_cong$ )
show  $?thesis$ 
proof
  assume  $cf: convergent\_prod\ f$ 
  with  $f$  have  $\neg (\lambda n. prod\ f\ \{..n\}) \longrightarrow 0$ 
    by  $simp$ 
  then have  $\neg (\lambda n. prod\ g\ \{..n\}) \longrightarrow 0$ 
    using  $\langle C \neq 0 \rangle filterlim\_cong$  by  $fastforce$ 
  then show  $convergent\_prod\ g$ 
    by ( $metis\ convergent\_mult\_const\_iff\ \langle C \neq 0 \rangle\ cong\ cf\ convergent\_LIMSEQ\_iff\ convergent\_prod\_iff\ convergent\ convergent\_prod\_imp\_convergent\ g$ )
  next
  assume  $cg: convergent\_prod\ g$ 
  have  $\exists a. C * a \neq 0 \wedge (\lambda n. prod\ g\ \{..n\}) \longrightarrow a$ 
    by ( $metis\ (no\_types)\ \langle C \neq 0 \rangle\ cg\ convergent\_prod\_iff\_nz\_lim\ divide\_eq\_0\_iff\ g\ nonzero\_mult\_div\_cancel\_right$ )
  then show  $convergent\_prod\ f$ 
    using  $*\ tendsto\_mult\_left\ filterlim\_cong$ 

```

```

    by (fastforce simp add: convergent_prod_iff_nz_lim f)
  qed
qed

lemma has_prod_finite:
  fixes f :: nat  $\Rightarrow$  'a::{semidom,t2_space}
  assumes [simp]: finite N
  and f:  $\bigwedge n. n \notin N \implies f\ n = 1$ 
  shows f has_prod ( $\prod_{n \in N}. f\ n$ )
proof -
  have eq: prod f {.. $n + \text{Suc}(\text{Max } N)$ } = prod f N for n
  proof (rule prod.mono_neutral_right)
    show  $N \subseteq \{.. $n + \text{Suc}(\text{Max } N)$ \}$ 
    by (auto simp: le_Suc_eq trans_le_add2)
    show  $\forall i \in \{.. $n + \text{Suc}(\text{Max } N)$ \} - N. f\ i = 1$ 
    using f by blast
  qed auto
  show ?thesis
  proof (cases  $\forall n \in N. f\ n \neq 0$ )
    case True
    then have prod f N  $\neq 0$ 
    by simp
    moreover have  $(\lambda n. \text{prod } f\ \{..n\}) \longrightarrow \text{prod } f\ N$ 
    by (rule LIMSEQ_offset[of _ Suc (Max N)]) (simp add: eq atLeast0LessThan
del: add_Suc_right)
    ultimately show ?thesis
    by (simp add: raw_has_prod_def has_prod_def)
  next
    case False
    then obtain k where  $k \in N$  and  $f\ k = 0$ 
    by auto
    let ?Z =  $\{n \in N. f\ n = 0\}$ 
    have maxge:  $\text{Max } ?Z \geq n$  if  $f\ n = 0$  for n
    using Max_ge [of ?Z]  $\langle$ finite N $\rangle$   $\langle$ f n = 0 $\rangle$ 
    by (metis (mono_tags) Collect_mem_eq f_finite_Collect_conjI mem_Collect_eq
zero_neq_one)
    let ?q = prod f {Suc (Max ?Z)..Max N}
    have [simp]: ?q  $\neq 0$ 
    using maxge Suc_n_not_le_n le_trans by force
    have eq:  $(\prod_{i \leq n + \text{Max } N}. f\ (\text{Suc } (i + \text{Max } ?Z))) = ?q$  for n
    proof -
      have  $(\prod_{i \leq n + \text{Max } N}. f\ (\text{Suc } (i + \text{Max } ?Z))) = \text{prod } f\ \{\text{Suc } (\text{Max } ?Z)..n + \text{Max } N + \text{Suc } (\text{Max } ?Z)\}$ 
      proof (rule prod.reindex_cong [where l =  $\lambda i. i + \text{Suc } (\text{Max } ?Z)$ , THEN sym])
        show  $\{\text{Suc } (\text{Max } ?Z)..n + \text{Max } N + \text{Suc } (\text{Max } ?Z)\} = (\lambda i. i + \text{Suc } (\text{Max } ?Z))\ \{.. $n + \text{Max } N$ \}$ 
        using le_Suc_ex by fastforce
      qed
    qed
  qed (auto simp: inj_on_def)

```

```

    also have ... = ?q
      by (rule prod_mono_neutral_right)
         (use Max.coboundedI [OF ‹finite N›] f in ‹force+›)
    finally show ?thesis .
  qed
  have q: raw_has_prod f (Suc (Max ?Z)) ?q
  proof (simp add: raw_has_prod_def)
    show (λn. ∏ i≤n. f (Suc (i + Max ?Z))) → ?q
      by (rule LIMSEQ_offset[of _ (Max N)]) (simp add: eq)
  qed
  show ?thesis
    unfolding has_prod_def
  proof (intro disjI2 exI conjI)
    show prod f N = 0
      using ‹f k = 0› ‹k ∈ N› ‹finite N› prod_zero by blast
    show f (Max ?Z) = 0
      using Max_in [of ?Z] ‹finite N› ‹f k = 0› ‹k ∈ N› by auto
  qed (use q in auto)
  qed
  qed

corollary has_prod_0:
  fixes f :: nat ⇒ 'a::{semidom,t2_space}
  assumes ∧n. f n = 1
  shows f has_prod 1
  by (simp add: asms has_prod_cong)

lemma prodinf_zero[simp]: prodinf (λn. 1::'a::real_normed_field) = 1
  using has_prod_unique by force

lemma convergent_prod_finite:
  fixes f :: nat ⇒ 'a::{idom,t2_space}
  assumes finite N ∧n. n ∉ N ⇒ f n = 1
  shows convergent_prod f
  proof -
    have ∃ n p. raw_has_prod f n p
      using asms has_prod_def has_prod_finite by blast
    then show ?thesis
      by (simp add: convergent_prod_def)
  qed

lemma has_prod_If_finite_set:
  fixes f :: nat ⇒ 'a::{idom,t2_space}
  shows finite A ⇒ (λr. if r ∈ A then f r else 1) has_prod (∏ r∈A. f r)
  using has_prod_finite[of A (λr. if r ∈ A then f r else 1)]
  by simp

lemma has_prod_If_finite:
  fixes f :: nat ⇒ 'a::{idom,t2_space}

```

shows $\text{finite } \{r. P r\} \implies (\lambda r. \text{if } P r \text{ then } f r \text{ else } 1) \text{ has_prod } (\prod r \mid P r. f r)$
using $\text{has_prod_If_finite_set}[\text{of } \{r. P r\}]$ **by** simp

lemma $\text{convergent_prod_If_finite_set}[\text{simp, intro}]$:
fixes $f :: \text{nat} \Rightarrow 'a::\{\text{idom}, \text{t2_space}\}$
shows $\text{finite } A \implies \text{convergent_prod } (\lambda r. \text{if } r \in A \text{ then } f r \text{ else } 1)$
by $(\text{simp add: convergent_prod_finite})$

lemma $\text{convergent_prod_If_finite}[\text{simp, intro}]$:
fixes $f :: \text{nat} \Rightarrow 'a::\{\text{idom}, \text{t2_space}\}$
shows $\text{finite } \{r. P r\} \implies \text{convergent_prod } (\lambda r. \text{if } P r \text{ then } f r \text{ else } 1)$
using $\text{convergent_prod_def has_prod_If_finite has_prod_def}$ **by** fastforce

lemma has_prod_single :
fixes $f :: \text{nat} \Rightarrow 'a::\{\text{idom}, \text{t2_space}\}$
shows $(\lambda r. \text{if } r = i \text{ then } f r \text{ else } 1) \text{ has_prod } f i$
using $\text{has_prod_If_finite}[\text{of } \lambda r. r = i]$ **by** simp

The ge1 assumption can probably be weakened, at the expense of extra work

lemma $\text{uniform_limit_prodinf}$:
fixes $f :: \text{nat} \Rightarrow \text{real} \Rightarrow \text{real}$
assumes $\text{uniformly_convergent_on } X (\lambda n x. \prod k < n. f k x)$
and $\text{ge1: } \bigwedge x k . x \in X \implies f k x \geq 1$
shows $\text{uniform_limit } X (\lambda n x. \prod k < n. f k x) (\lambda x. \prod k. f k x) \text{ sequentially}$
proof –
have $ul: \text{uniform_limit } X (\lambda n x. \prod k < n. f k x) (\lambda x. \text{lim } (\lambda n. \prod k < n. f k x))$
 sequentially
using $\text{assms uniformly_convergent_uniform_limit_iff}$ **by** blast
moreover **have** $(\prod k. f k x) = \text{lim } (\lambda n. \prod k < n. f k x)$ **if** $x \in X$ **for** x
proof $(\text{intro prodinf_eq_lim'})$
have $\text{tends: } (\lambda n. \prod k < n. f k x) \longrightarrow \text{lim } (\lambda n. \prod k < n. f k x)$
using $\text{tendsto_uniform_limitI } [OF ul]$ **that** **by** metis
moreover **have** $(\prod k < n. f k x) \geq 1$ **for** n
using ge1 **by** $(\text{simp add: prod_ge_1 that})$
ultimately **have** $\text{lim } (\lambda n. \prod k < n. f k x) \geq 1$
by $(\text{meson LIMSEQ_le_const})$
then **have** $\text{raw_has_prod } (\lambda k. f k x) 0 (\text{lim } (\lambda n. \prod k < n. f k x))$
using $\text{LIMSEQ_lessThan_iff_atMost tends}$ **by** $(\text{auto simp: raw_has_prod_def})$
then **show** $\text{convergent_prod } (\lambda k. f k x)$
unfolding $\text{convergent_prod_def}$ **by** blast
show $\bigwedge k. f k x \neq 0$
by $(\text{smt (verit) ge1 that})$
qed
ultimately **show** $?thesis$
by $(\text{metis (mono_tags, lifting) uniform_limit_cong'})$
qed

context
fixes $f :: \text{nat} \Rightarrow 'a :: \text{real_normed_field}$

begin

lemma *convergent_prod_imp_has_prod*:

assumes *convergent_prod* *f*

shows $\exists p. f \text{ has_prod } p$

proof -

obtain *M p* where *p*: *raw_has_prod* *f* *M* *p*

using *assms convergent_prod_def* by *blast*

then have $p \neq 0$

using *raw_has_prod_nonzero* by *blast*

with *p* have *fnz*: $f\ i \neq 0$ if $i \geq M$ for *i*

using *raw_has_prod_eq_0* that by *blast*

define *C* where $C = (\prod_{n < M}. f\ n)$

show *?thesis*

proof (*cases* $\forall n \leq M. f\ n \neq 0$)

case *True*

then have $C \neq 0$

by (*simp* *add*: *C_def*)

then show *?thesis*

by (*meson* *True* *assms convergent_prod_offset_0* *fnz* *has_prod_def* *nat_le_linear*)

next

case *False*

let *?N* = *GREATEST* *n*. $f\ n = 0$

have *0*: $f\ ?N = 0$

using *fnz* *False*

by (*metis* (*mono_tags*, *lifting*) *GreatestI_ex_nat* *nat_le_linear*)

have $f\ i \neq 0$ if $i > ?N$ for *i*

by (*metis* (*mono_tags*, *lifting*) *Greatest_le_nat* *fnz* *leD* *linear* *that*)

then have $\exists p. \text{raw_has_prod } f\ (Suc\ ?N)\ p$

using *assms* by (*auto* *simp*: *intro!*: *convergent_prod_ignore_nonzero_segment*)

then show *?thesis*

unfolding *has_prod_def* using *0* by *blast*

qed

qed

lemma *convergent_prod_has_prod* [*intro*]:

shows *convergent_prod* *f* $\implies f \text{ has_prod } (\text{prodinf } f)$

unfolding *prodinf_def*

by (*metis* *convergent_prod_imp_has_prod* *has_prod_unique* *theI'*)

lemma *convergent_prod_LIMSEQ*:

shows *convergent_prod* *f* $\implies (\lambda n. \prod_{i \leq n}. f\ i) \longrightarrow \text{prodinf } f$

by (*metis* *convergent_LIMSEQ_iff* *convergent_prod_has_prod* *convergent_prod_imp_convergent*

convergent_prod_to_zero_iff *raw_has_prod_eq_0* *has_prod_def* *prodinf_eq_lim* *zero_le*)

theorem *has_prod_iff*: $f \text{ has_prod } x \iff \text{convergent_prod } f \wedge \text{prodinf } f = x$

proof

3370

```
assume f has_prod x
then show convergent_prod f  $\wedge$  prodinf f = x
  apply safe
  using convergent_prod_def has_prod_def apply blast
  using has_prod_unique by blast
qed auto
```

```
lemma convergent_prod_has_prod_iff: convergent_prod f  $\longleftrightarrow$  f has_prod prod-
inf f
  by (auto simp: has_prod_iff convergent_prod_has_prod)
```

```
lemma prodinf_finite:
  assumes N: finite N
  and f:  $\bigwedge n. n \notin N \implies f n = 1$ 
  shows prodinf f = ( $\prod_{n \in N}. f n$ )
  using has_prod_finite[OF assms, THEN has_prod_unique] by simp
```

end

9.22.6 Infinite products on ordered topological monoids

```
context
  fixes f :: nat  $\Rightarrow$  'a::{linordered_semidom, linorder_topology}
begin
```

```
lemma has_prod_nonzero:
  assumes f has_prod a  $a \neq 0$ 
  shows f k  $\neq 0$ 
  using assms by (auto simp: has_prod_def raw_has_prod_def LIMSEQ_prod_0
LIMSEQ_unique)
```

```
lemma has_prod_le:
  assumes f: f has_prod a and g: g has_prod b and le:  $\bigwedge n. 0 \leq f n \wedge f n \leq g n$ 
  shows a  $\leq b$ 
```

```
proof (cases a=0  $\vee$  b=0)
```

```
case True
```

```
then show ?thesis
```

```
proof
```

```
  assume [simp]: a=0
```

```
  have b  $\geq 0$ 
```

```
  proof (rule LIMSEQ_prod_nonneg)
```

```
    show  $(\lambda n. prod g \{..n\}) \longrightarrow b$ 
```

```
    using g by (auto simp: has_prod_def raw_has_prod_def LIMSEQ_prod_0)
```

```
  qed (use le order_trans in auto)
```

```
  then show ?thesis
```

```
    by auto
```

```
next
```

```
  assume [simp]: b=0
```

```
  then obtain i where g i = 0
```

```

    using g by (auto simp: prod_defs)
  then have f i = 0
    using antisym le by force
  then have a=0
    using f by (auto simp: prod_defs LIMSEQ_prod_0 LIMSEQ_unique)
  then show ?thesis
    by auto
qed
next
case False
then show ?thesis
  using assms
  unfolding has_prod_def raw_has_prod_def
  by (force simp: LIMSEQ_prod_0 intro!: LIMSEQ_le prod_mono)
qed

```

```

lemma prodinf_le:
  assumes f: f has_prod a and g: g has_prod b and le:  $\bigwedge n. 0 \leq f n \wedge f n \leq g n$ 
  shows prodinf f  $\leq$  prodinf g
  using has_prod_le [OF assms] has_prod_unique f g by blast
end

```

```

lemma prod_le_produf:
  fixes f :: nat  $\Rightarrow$  'a::{\linordered_idom,\linorder_topology}
  assumes f has_prod a  $\wedge i. 0 \leq f i \wedge i. i \geq n \implies 1 \leq f i$ 
  shows prod f {.. $n$ }  $\leq$  produf f
  by(rule has_prod_le[OF has_prod_If_finite_set]) (use assms has_prod_unique
in auto)

```

```

lemma produf_nonneg:
  fixes f :: nat  $\Rightarrow$  'a::{\linordered_idom,\linorder_topology}
  assumes f has_prod a  $\wedge i. 1 \leq f i$ 
  shows 1  $\leq$  produf f
  using prod_le_produf[of f a 0] assms
  by (metis order_trans prod_ge_1 zero_le_one)

```

```

lemma produf_le_const:
  fixes f :: nat  $\Rightarrow$  real
  assumes convergent_prod f  $\wedge n. n \geq N \implies$  prod f {.. $n$ }  $\leq x$ 
  shows produf f  $\leq x$ 
  by (metis lessThan_Suc_atMost assms convergent_prod LIMSEQ LIMSEQ_le_const2
atMost_iff lessThan_iff less_le)

```

```

lemma produf_eq_one_iff [simp]:
  fixes f :: nat  $\Rightarrow$  real
  assumes f: convergent_prod f and ge1:  $\bigwedge n. 1 \leq f n$ 
  shows produf f = 1  $\iff (\forall n. f n = 1)$ 

```

proof

```

assume  $\text{prodinf } f = 1$ 
then have  $(\lambda n. \prod_{i < n}. f \ i) \longrightarrow 1$ 
  using  $\text{convergent\_prod\_LIMSEQ}[of\ f]$  assms by  $(\text{simp add: LIMSEQ\_lessThan\_iff\_atMost})$ 
then have  $\bigwedge i. (\prod_{n \in \{i\}}. f \ n) \leq 1$ 
proof  $(\text{rule LIMSEQ\_le\_const})$ 
  have  $1 \leq \text{prod } f \ n$  for  $n$ 
    by  $(\text{simp add: ge1 prod\_ge\_1})$ 
  have  $\text{prod } f \ \{.. < n\} = 1$  for  $n$ 
    by  $(\text{metis } \langle \bigwedge n. 1 \leq \text{prod } f \ n \rangle \langle \text{prodinf } f = 1 \rangle \text{antisym } f \text{convergent\_prod\_has\_prod}$ 
 $\text{ge1 order\_trans prod\_le\_prodinf zero\_le\_one})$ 
  then have  $(\prod_{n \in \{i\}}. f \ n) \leq \text{prod } f \ \{.. < n\}$  if  $n \geq \text{Suc } i$  for  $i \ n$ 
    by  $(\text{metis mult.left\_neutral order\_refl prod.cong prod.neutral\_const prod.lessThan\_Suc})$ 
  then show  $\exists N. \forall n \geq N. (\prod_{n \in \{i\}}. f \ n) \leq \text{prod } f \ \{.. < n\}$  for  $i$ 
    by  $\text{blast}$ 
qed
with  $\text{ge1}$  show  $\forall n. f \ n = 1$ 
  by  $(\text{auto intro!: antisym})$ 
qed  $(\text{metis prodinf\_zero fun\_eq\_iff})$ 

```

lemma prodinf_pos_iff :

```

fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
assumes  $\text{convergent\_prod } f \ \bigwedge n. 1 \leq f \ n$ 
shows  $1 < \text{prodinf } f \iff (\exists i. 1 < f \ i)$ 
using  $\text{prod\_le\_prodinf}[of\ f\ 1]$   $\text{prodinf\_eq\_one\_iff}$ 
by  $(\text{metis convergent\_prod\_has\_prod assms less\_le prodinf\_nonneg})$ 

```

lemma less_1_prodinf2 :

```

fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
assumes  $\text{convergent\_prod } f \ \bigwedge n. 1 \leq f \ n$   $1 < f \ i$ 
shows  $1 < \text{prodinf } f$ 
proof -
  have  $1 < (\prod_{n < \text{Suc } i}. f \ n)$ 
    using  $\text{assms}$  by  $(\text{intro less\_1\_prod2}[where\ i=i])$   $\text{auto}$ 
  also have  $\dots \leq \text{prodinf } f$ 
    by  $(\text{intro prod\_le\_prodinf})$   $(\text{use assms order\_trans zero\_le\_one in } \langle \text{blast+} \rangle)$ 
  finally show  $?thesis$  .
qed

```

lemma less_1_prodinf :

```

fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
shows  $\llbracket \text{convergent\_prod } f; \bigwedge n. 1 < f \ n \rrbracket \implies 1 < \text{prodinf } f$ 
by  $(\text{intro less\_1\_prodinf2}[where\ i=1])$   $(\text{auto intro: less\_imp\_le})$ 

```

lemma prodinf_nonzero :

```

fixes  $f :: \text{nat} \Rightarrow 'a :: \{\text{idom, topological\_semigroup\_mult, t2\_space}\}$ 
assumes  $\text{convergent\_prod } f \ \bigwedge i. f \ i \neq 0$ 
shows  $\text{prodinf } f \neq 0$ 
by  $(\text{metis assms convergent\_prod\_offset\_0 has\_prod\_unique raw\_has\_prod\_def})$ 

```


has_prod_def)

lemma *less_0_produinf*:

fixes $f :: \text{nat} \Rightarrow \text{real}$

assumes f : *convergent_prod* f **and** 0 : $\bigwedge i. f\ i > 0$

shows $0 < \text{produinf } f$

proof –

have $\text{produinf } f \neq 0$

by (*metis* *assms* *less_irrefl* *produinf_nonzero*)

moreover **have** $0 < (\prod_{n < i. f\ n})$ **for** i

by (*simp* *add*: *0_prod_pos*)

then **have** $\text{produinf } f \geq 0$

using *convergent_prod_LIMSEQ* [*OF* f] *LIMSEQ_prod_nonneg* 0 *less_le* **by**

blast

ultimately **show** *?thesis*

by *auto*

qed

lemma *prod_less_produinf2*:

fixes $f :: \text{nat} \Rightarrow \text{real}$

assumes f : *convergent_prod* f **and** 1 : $\bigwedge m. m \geq n \implies 1 \leq f\ m$ **and** 0 : $\bigwedge m. 0 < f\ m$ **and** i : $n \leq i \wedge 1 < f\ i$

shows $\text{prod } f\ \{..<n\} < \text{produinf } f$

proof –

have $\text{prod } f\ \{..<n\} \leq \text{prod } f\ \{..<i\}$

by (*rule* *prod_mono2*) (*use* *assms* *less_le* **in** *auto*)

then **have** $\text{prod } f\ \{..<n\} < f\ i * \text{prod } f\ \{..<i\}$

using *mult_less_le_imp_less*[*of* $1\ f\ i\ \text{prod } f\ \{..<n\}\ \text{prod } f\ \{..<i\}$] *assms*

by (*simp* *add*: *prod_pos*)

moreover **have** $\text{prod } f\ \{..<\text{Suc } i\} \leq \text{produinf } f$

using *prod_le_produinf*[*of* $f\ \text{Suc } i$]

by (*meson* $0\ 1\ \text{Suc } \text{leD}$ *convergent_prod_has_prod* $\langle n \leq i \rangle$ *le_trans* *less_eq_real_def*)

ultimately **show** *?thesis*

by (*metis* *le_less_trans* *mult commute* *not_le* *prod_lessThan_Suc*)

qed

lemma *prod_less_produinf*:

fixes $f :: \text{nat} \Rightarrow \text{real}$

assumes f : *convergent_prod* f **and** 1 : $\bigwedge m. m \geq n \implies 1 < f\ m$ **and** 0 : $\bigwedge m. 0 < f\ m$

shows $\text{prod } f\ \{..<n\} < \text{produinf } f$

by (*meson* $0\ 1\ f\ \text{le_less } \text{prod_less_produinf2}$)

lemma *raw_has_prodI_bounded*:

fixes $f :: \text{nat} \Rightarrow \text{real}$

assumes *pos*: $\bigwedge n. 1 \leq f\ n$

and *le*: $\bigwedge n. (\prod_{i < n. f\ i} \leq x$

shows $\exists p. \text{raw_has_prod } f\ 0\ p$

unfolding *raw_has_prod_def* *add_0_right*

```

proof (rule exI LIMSEQ incseq_SUP conjI)+
  show bdd_above (range ( $\lambda n. \text{prod } f \ \{..n\}$ ))
    by (metis bdd_aboveI2 le_lessThan_Suc_atMost)
  then have ( $\text{SUP } i. \text{prod } f \ \{..i\} > 0$ )
    by (metis UNIV_I cSUP_upper less_le_trans pos_prod_pos zero_less_one)
  then show ( $\text{SUP } i. \text{prod } f \ \{..i\} \neq 0$ )
    by auto
  show incseq ( $\lambda n. \text{prod } f \ \{..n\}$ )
    using pos order_trans [OF zero_le_one] by (auto simp: mono_def intro!:
prod_mono2)
qed

```

```

lemma convergent_prodI_nonneg_bounded:
  fixes  $f :: \text{nat} \Rightarrow \text{real}$ 
  assumes  $\bigwedge n. 1 \leq f \ n \ \wedge \ (\prod_{i < n}. f \ i) \leq x$ 
  shows convergent_prod  $f$ 
  using convergent_prod_def raw_has_prodI_bounded [OF assms] by blast

```

9.22.7 Infinite products on topological spaces

```

context
  fixes  $f \ g :: \text{nat} \Rightarrow 'a::\{t2\_space, \text{topological\_semigroup\_mult}, \text{idom}\}$ 
begin

```

```

lemma raw_has_prod_mult:  $\llbracket \text{raw\_has\_prod } f \ M \ a; \text{raw\_has\_prod } g \ M \ b \rrbracket \implies$ 
 $\text{raw\_has\_prod } (\lambda n. f \ n * g \ n) \ M \ (a * b)$ 
  by (force simp add: prod.distrib tendsto_mult raw_has_prod_def)

```

```

lemma has_prod_mult_nz:  $\llbracket f \text{ has\_prod } a; g \text{ has\_prod } b; a \neq 0; b \neq 0 \rrbracket \implies (\lambda n.$ 
 $f \ n * g \ n) \text{ has\_prod } (a * b)$ 
  by (simp add: raw_has_prod_mult has_prod_def)

```

end

```

context
  fixes  $f \ g :: \text{nat} \Rightarrow 'a::\text{real\_normed\_field}$ 
begin

```

```

lemma has_prod_mult:
  assumes  $f: f \text{ has\_prod } a$  and  $g: g \text{ has\_prod } b$ 
  shows  $(\lambda n. f \ n * g \ n) \text{ has\_prod } (a * b)$ 
  using  $f$  [unfolded has_prod_def]

```

```

proof (elim disjE exE conjE)
  assume  $f0: \text{raw\_has\_prod } f \ 0 \ a$ 
  show ?thesis
    using  $g$  [unfolded has_prod_def]
  proof (elim disjE exE conjE)
    assume  $g0: \text{raw\_has\_prod } g \ 0 \ b$ 

```

```

  with f0 show ?thesis
  by (force simp add: has_prod_def prod.distrib tendsto_mult raw_has_prod_def)
next
fix j q
assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
obtain p where p: raw_has_prod f (Suc j) p
  using f0 raw_has_prod_ignore_initial_segment by blast
then have Ex (raw_has_prod ( $\lambda n. f n * g n$ ) (Suc j))
  using q raw_has_prod_mult by blast
then show ?thesis
  using <b = 0> <g j = 0> has_prod_0_iff by fastforce
qed
next
fix i p
assume a = 0 and f i = 0 and p: raw_has_prod f (Suc i) p
show ?thesis
  using g [unfolded has_prod_def]
proof (elim disjE exE conjE)
  assume g0: raw_has_prod g 0 b
  obtain q where q: raw_has_prod g (Suc i) q
    using g0 raw_has_prod_ignore_initial_segment by blast
  then have Ex (raw_has_prod ( $\lambda n. f n * g n$ ) (Suc i))
    using raw_has_prod_mult p by blast
  then show ?thesis
    using <a = 0> <f i = 0> has_prod_0_iff by fastforce
next
fix j q
assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
obtain p' where p': raw_has_prod f (Suc (max i j)) p'
  by (metis raw_has_prod_ignore_initial_segment max_Suc_Suc max_def p)
moreover
obtain q' where q': raw_has_prod g (Suc (max i j)) q'
  by (metis raw_has_prod_ignore_initial_segment max.cobounded2 max_Suc_Suc
q)
ultimately show ?thesis
  using <b = 0> by (simp add: has_prod_def) (metis <f i = 0> <g j = 0>
raw_has_prod_mult max_def)
qed
qed

lemma convergent_prod_mult:
  assumes f: convergent_prod f and g: convergent_prod g
  shows convergent_prod ( $\lambda n. f n * g n$ )
  unfolding convergent_prod_def
proof -
  obtain M p N q where p: raw_has_prod f M p and q: raw_has_prod g N q
    using convergent_prod_def f g by blast+
  then obtain p' q' where p': raw_has_prod f (max M N) p' and q': raw_has_prod
g (max M N) q'

```

3376

by (meson raw_has_prod ignore_initial_segment max.cobounded1 max.cobounded2)
 then show $\exists M p. \text{raw_has_prod } (\lambda n. f n * g n) M p$
 using raw_has_prod_mult by blast
 qed

lemma prodf_mult: $\text{convergent_prod } f \implies \text{convergent_prod } g \implies \text{prodf } f * \text{prodf } g = (\prod n. f n * g n)$
 by (intro has_prod_unique has_prod_mult convergent_prod_has_prod)

end

context

fixes $f :: 'i \Rightarrow \text{nat} \Rightarrow 'a::\text{real_normed_field}$
 and $I :: 'i \text{ set}$

begin

lemma has_prod_prod: $(\bigwedge i. i \in I \implies (f i) \text{ has_prod } (x i)) \implies (\lambda n. \prod_{i \in I} f i n) \text{ has_prod } (\prod_{i \in I} x i)$
 by (induct I rule: infinite_finite_induct) (auto intro!: has_prod_mult)

lemma prodf_prod: $(\bigwedge i. i \in I \implies \text{convergent_prod } (f i)) \implies (\prod n. \prod_{i \in I} f i n) = (\prod_{i \in I} \prod n. f i n)$
 using has_prod_unique[OF has_prod_prod, OF convergent_prod_has_prod] by simp

lemma convergent_prod_prod: $(\bigwedge i. i \in I \implies \text{convergent_prod } (f i)) \implies \text{convergent_prod } (\lambda n. \prod_{i \in I} f i n)$
 using convergent_prod_has_prod_iff has_prod_prod prodf_prod by force

end

9.22.8 Infinite summability on real normed fields

context

fixes $f :: \text{nat} \Rightarrow 'a::\text{real_normed_field}$

begin

lemma raw_has_prod_Suc_iff: $\text{raw_has_prod } f M (a * f M) \longleftrightarrow \text{raw_has_prod } (\lambda n. f (Suc n)) M a \wedge f M \neq 0$

proof -

have $\text{raw_has_prod } f M (a * f M) \longleftrightarrow (\lambda i. \prod_{j \leq \text{Suc } i} f (j+M)) \longrightarrow a * f M \wedge a * f M \neq 0$

by (subst filterlim_sequentially_Suc) (simp add: raw_has_prod_def)

also have $\dots \longleftrightarrow (\lambda i. (\prod_{j \leq i} f (Suc j + M)) * f M) \longrightarrow a * f M \wedge a * f M \neq 0$

by (simp add: ac_simps atMost_Suc_eq_insert_0 image_Suc_atMost prod.atLeast1_atMost_eq lessThan_Suc_atMost

del: prod.cl_ivl_Suc)

also have $\dots \longleftrightarrow \text{raw_has_prod } (\lambda n. f (Suc n)) M a \wedge f M \neq 0$

```

proof safe
  assume tends:  $(\lambda i. (\prod_{j \leq i}. f (Suc\ j + M)) * f\ M) \longrightarrow a * f\ M$  and 0:  $a * f\ M \neq 0$ 
  with tendsto_divide[OF tends tendsto_const, of f M]
  show raw_has_prod  $(\lambda n. f (Suc\ n))\ M\ a$ 
    by (simp add: raw_has_prod_def)
  qed (auto intro: tendsto_mult_right simp: raw_has_prod_def)
  finally show ?thesis .
qed

lemma has_prod_Suc_iff:
  assumes  $f\ 0 \neq 0$  shows  $(\lambda n. f (Suc\ n))\ has\_prod\ a \longleftrightarrow f\ has\_prod\ (a * f\ 0)$ 
proof (cases  $a = 0$ )
  case True
  then show ?thesis
  proof (simp add: has_prod_def, safe)
    fix  $i\ x$ 
    assume  $f (Suc\ i) = 0$  and raw_has_prod  $(\lambda n. f (Suc\ n))\ (Suc\ i)\ x$ 
    then obtain  $y$  where raw_has_prod  $f (Suc\ (Suc\ i))\ y$ 
    by (metis (no_types) raw_has_prod_eq_0 Suc_n_not_le_n raw_has_prod_Suc_iff
raw_has_prod_ignore_initial_segment raw_has_prod_nonzero linear)
    then show  $\exists i. f\ i = 0 \wedge \exists x (raw\_has\_prod\ f (Suc\ i))$ 
      using  $\langle f (Suc\ i) = 0 \rangle$  by blast
  next
    fix  $i\ x$ 
    assume  $f\ i = 0$  and  $x: raw\_has\_prod\ f (Suc\ i)\ x$ 
    then obtain  $j$  where  $j: i = Suc\ j$ 
    by (metis assms not0_implies_Suc)
    moreover have  $\exists y. raw\_has\_prod\ (\lambda n. f (Suc\ n))\ i\ y$ 
    using  $x$  by (auto simp: raw_has_prod_def)
    then show  $\exists i. f (Suc\ i) = 0 \wedge \exists x (raw\_has\_prod\ (\lambda n. f (Suc\ n))\ (Suc\ i))$ 
      using  $\langle f\ i = 0 \rangle\ j$  by blast
  qed
next
  case False
  then show ?thesis
    by (auto simp: has_prod_def raw_has_prod_Suc_iff assms)
qed

lemma convergent_prod_Suc_iff [simp]:
  shows  $convergent\_prod\ (\lambda n. f (Suc\ n)) = convergent\_prod\ f$ 
proof
  assume convergent_prod  $f$ 
  then obtain  $M\ L$  where  $M\_nz: \forall n \geq M. f\ n \neq 0$  and
     $M\_L: (\lambda n. \prod_{i \leq n}. f (i + M)) \longrightarrow L$  and  $L \neq 0$ 
  unfolding convergent_prod_altdef by auto
  have  $(\lambda n. \prod_{i \leq n}. f (Suc (i + M))) \longrightarrow L / f\ M$ 
  proof -
    have  $(\lambda n. \prod_{i \in \{0..Suc\ n\}}. f (i + M)) \longrightarrow L$ 

```

```

using M_L
apply (subst (asm) filterlim_sequentially_Suc[symmetric])
using atLeast0AtMost by auto
then have (λn. f M * (∏ i∈{0..n}. f (Suc (i + M)))) → L
apply (subst (asm) prod.atLeast0_atMost_Suc_shift)
by simp
then have (λn. (∏ i∈{0..n}. f (Suc (i + M)))) → L/f M
apply (drule_tac tendsto_divide)
using M_nz[rule_format,of M,simplified] by auto
then show ?thesis unfolding atLeast0AtMost .
qed
then show convergent_prod (λn. f (Suc n)) unfolding convergent_prod_altdef
apply (rule_tac exI[where x=M])
apply (rule_tac exI[where x=L/f M])
using M_nz ⟨L≠0⟩ by auto
next
assume convergent_prod (λn. f (Suc n))
then obtain M where ∃ L. (∀ n≥M. f (Suc n) ≠ 0) ∧ (λn. ∏ i≤n. f (Suc (i
+ M))) → L ∧ L ≠ 0
unfolding convergent_prod_altdef by auto
then show convergent_prod f unfolding convergent_prod_altdef
apply (rule_tac exI[where x=Suc M])
using Suc_le_D by auto
qed

lemma raw_has_prod_inverse:
  assumes raw_has_prod f M a shows raw_has_prod (λn. inverse (f n)) M
  (inverse a)
  using assms unfolding raw_has_prod_def by (auto dest: tendsto_inverse simp:
  prod_inversef [symmetric])

lemma has_prod_inverse:
  assumes f has_prod a shows (λn. inverse (f n)) has_prod (inverse a)
  using assms raw_has_prod_inverse unfolding has_prod_def by auto

lemma convergent_prod_inverse:
  assumes convergent_prod f
  shows convergent_prod (λn. inverse (f n))
  using assms unfolding convergent_prod_def by (blast intro: raw_has_prod_inverse
  elim: )

end

context
  fixes f :: nat ⇒ 'a::real_normed_field
begin

lemma raw_has_prod_Suc_iff': raw_has_prod f M a ⟷ raw_has_prod (λn. f
(Suc n)) M (a / f M) ∧ f M ≠ 0

```

by (metis raw_has_prod_eq_0 add commute add.left_neutral raw_has_prod_Suc_iff raw_has_prod_nonzero le_add1 nonzero_mult_div_cancel_right times_divide_eq_left)

lemma has_prod_divide: $f \text{ has_prod } a \implies g \text{ has_prod } b \implies (\lambda n. f\ n / g\ n) \text{ has_prod } (a / b)$
 unfolding divide_inverse by (intro has_prod_inverse has_prod_mult)

lemma convergent_prod_divide:
 assumes $f: \text{convergent_prod } f$ and $g: \text{convergent_prod } g$
 shows $\text{convergent_prod } (\lambda n. f\ n / g\ n)$
 using $f\ g \text{ has_prod_divide has_prod_iff}$ by blast

lemma prodf_divide: $\text{convergent_prod } f \implies \text{convergent_prod } g \implies \text{prodf } f / \text{prodf } g = (\prod n. f\ n / g\ n)$
 by (intro has_prod_unique has_prod_divide convergent_prod_has_prod)

lemma prodf_inverse: $\text{convergent_prod } f \implies (\prod n. \text{inverse } (f\ n)) = \text{inverse } (\prod n. f\ n)$
 by (intro has_prod_unique [symmetric] has_prod_inverse convergent_prod_has_prod)

lemma has_prod_Suc_imp:
 assumes $(\lambda n. f\ (\text{Suc } n)) \text{ has_prod } a$
 shows $f \text{ has_prod } (a * f\ 0)$

proof -

have $f \text{ has_prod } (a * f\ 0)$ when $\text{raw_has_prod } (\lambda n. f\ (\text{Suc } n))\ 0\ a$
 apply (cases $f\ 0=0$)
 using that unfolding has_prod_def raw_has_prod_Suc
 by (auto simp add: raw_has_prod_Suc_iff)

moreover have $f \text{ has_prod } (a * f\ 0)$ when
 $(\exists i\ q. a = 0 \wedge f\ (\text{Suc } i) = 0 \wedge \text{raw_has_prod } (\lambda n. f\ (\text{Suc } n))\ (\text{Suc } i)\ q)$

proof -

from that
 obtain $i\ q$ where $a = 0 \wedge f\ (\text{Suc } i) = 0 \wedge \text{raw_has_prod } (\lambda n. f\ (\text{Suc } n))\ (\text{Suc } i)\ q$
 by auto
 then show ?thesis unfolding has_prod_def
 by (auto intro!: exI [where $x=\text{Suc } i$] simp: raw_has_prod_Suc)

qed

ultimately show $f \text{ has_prod } (a * f\ 0)$ using *assms* unfolding has_prod_def
 by auto

qed

lemma has_prod_iff_shift:
 assumes $\bigwedge i. i < n \implies f\ i \neq 0$
 shows $(\lambda i. f\ (i + n)) \text{ has_prod } a \iff f \text{ has_prod } (a * (\prod i < n. f\ i))$
 using *assms*

proof (induct n arbitrary: a)

case 0

then show ?case by simp

next

3380

```
case (Suc n)
then have  $(\lambda i. f (Suc i + n)) \text{ has\_prod } a \longleftrightarrow (\lambda i. f (i + n)) \text{ has\_prod } (a * f n)$ 
  by (subst has_prod_Suc_iff) auto
with Suc show ?case
  by (simp add: ac_simps)
qed
```

```
corollary has_prod_iff_shift':
  assumes  $\bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } (a / (\prod_{i < n}. f i)) \longleftrightarrow f \text{ has\_prod } a$ 
  by (simp add: assms has_prod_iff_shift)
```

```
lemma has_prod_one_iff_shift:
  assumes  $\bigwedge i. i < n \implies f i = 1$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } a \longleftrightarrow (\lambda i. f i) \text{ has\_prod } a$ 
  by (simp add: assms has_prod_iff_shift)
```

```
lemma convergent_prod_iff_shift [simp]:
  shows convergent_prod  $(\lambda i. f (i + n)) \longleftrightarrow \text{convergent\_prod } f$ 
  apply safe
  using convergent_prod_offset apply blast
  using convergent_prod_ignore_initial_segment convergent_prod_def by blast
```

```
lemma has_prod_split_initial_segment:
  assumes  $f \text{ has\_prod } a \wedge \bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\lambda i. f (i + n)) \text{ has\_prod } (a / (\prod_{i < n}. f i))$ 
  using assms has_prod_iff_shift' by blast
```

```
lemma prodinf_divide_initial_segment:
  assumes convergent_prod  $f \wedge \bigwedge i. i < n \implies f i \neq 0$ 
  shows  $(\prod i. f (i + n)) = (\prod i. f i) / (\prod_{i < n}. f i)$ 
  by (rule has_prod_unique[symmetric]) (auto simp: assms has_prod_iff_shift)
```

```
lemma prodinf_split_initial_segment:
  assumes convergent_prod  $f \wedge \bigwedge i. i < n \implies f i \neq 0$ 
  shows  $\text{prodinf } f = (\prod i. f (i + n)) * (\prod_{i < n}. f i)$ 
  by (auto simp add: assms prodinf_divide_initial_segment)
```

```
lemma prodinf_split_head:
  assumes convergent_prod  $f f 0 \neq 0$ 
  shows  $(\prod n. f (Suc n)) = \text{prodinf } f / f 0$ 
  using prodinf_split_initial_segment[of 1] assms by simp
```

```
lemma has_prod_ignore_initial_segment':
  assumes convergent_prod  $f$ 
  shows  $f \text{ has\_prod } ((\prod_{k < n}. f k) * (\prod k. f (k + n)))$ 
proof (cases  $\exists k < n. f k = 0$ )
  case True
```



```

hence [simp]:  $(\prod_{k < n}. f\ k) = 0$ 
  by (meson finite_lessThan lessThan_iff prod_zero)
thus ?thesis using True assms
  by (metis convergent_prod_has_prod_iff has_prod_zeroI mult_not_zero)
next
  case False
  hence  $(\lambda i. f\ (i + n))$  has_prod (prodimf f / prod f  $\{.. < n\}$ )
    using assms by (intro has_prod_split_initial_segment) (auto simp: convergent_prod_has_prod_iff)
  hence  $\text{prodimf } f = \text{prod } f\ \{.. < n\} * (\prod_{k. f\ (k + n)})$ 
    using False by (simp add: has_prod_iff divide_simps mult_ac)
  thus ?thesis
    using assms by (simp add: convergent_prod_has_prod_iff)
qed

end

context
  fixes  $f :: \text{nat} \Rightarrow 'a :: \text{real\_normed\_field}$ 
begin

lemma convergent_prod_inverse_iff [simp]:  $\text{convergent\_prod } (\lambda n. \text{inverse } (f\ n))$ 
 $\longleftrightarrow \text{convergent\_prod } f$ 
  by (auto dest: convergent_prod_inverse)

lemma convergent_prod_const_iff [simp]:
  fixes  $c :: 'a :: \{\text{real\_normed\_field}\}$ 
  shows  $\text{convergent\_prod } (\lambda_. c) \longleftrightarrow c = 1$ 
proof
  assume  $\text{convergent\_prod } (\lambda_. c)$ 
  then show  $c = 1$ 
    using convergent_prod_imp_LIMSEQ LIMSEQ_unique by blast
next
  assume  $c = 1$ 
  then show  $\text{convergent\_prod } (\lambda_. c)$ 
    by auto
qed

lemma has_prod_power:  $f$  has_prod  $a \implies (\lambda i. f\ i \wedge n)$  has_prod  $(a \wedge n)$ 
  by (induction n) (auto simp: has_prod_mult)

lemma convergent_prod_power:  $\text{convergent\_prod } f \implies \text{convergent\_prod } (\lambda i. f\ i \wedge n)$ 
  by (induction n) (auto simp: convergent_prod_mult)

lemma prodinf_power:  $\text{convergent\_prod } f \implies \text{prodimf } (\lambda i. f\ i \wedge n) = \text{prodimf } f \wedge n$ 
  by (metis has_prod_unique convergent_prod_imp_has_prod has_prod_power)

```

3382

end

9.22.9 Exponentials and logarithms

context

fixes $f :: \text{nat} \Rightarrow 'a::\{\text{real_normed_field}, \text{banach}\}$

begin

lemma *sums_imp_has_prod_exp*:

assumes $f \text{ sums } s$

shows $\text{raw_has_prod } (\lambda i. \text{exp } (f i)) \ 0 \ (\text{exp } s)$

using *assms continuous_on_exp* [of UNIV $\lambda x::'a. x$]

using *continuous_on_tendsto_compose* [of UNIV $\text{exp } (\lambda n. \text{sum } f \ \{\dots n\}) \ s$]

by (*simp add: prod_defs sums_def le_exp_sum*)

lemma *convergent_prod_exp*:

assumes *summable* f

shows *convergent_prod* $(\lambda i. \text{exp } (f i))$

using *sums_imp_has_prod_exp* *assms unfolding summable_def convergent_prod_def*
by *blast*

lemma *prodinf_exp*:

assumes *summable* f

shows *prodinf* $(\lambda i. \text{exp } (f i)) = \text{exp } (\text{suminf } f)$

proof –

have $f \text{ sums } \text{suminf } f$

using *assms* by *blast*

then have $(\lambda i. \text{exp } (f i)) \text{ has_prod } \text{exp } (\text{suminf } f)$

by (*simp add: has_prod_def sums_imp_has_prod_exp*)

then show *thesis*

by (*rule has_prod_unique [symmetric]*)

qed

end

theorem *convergent_prod_iff_summable_real*:

fixes $a :: \text{nat} \Rightarrow \text{real}$

assumes $\bigwedge n. a \ n > 0$

shows *convergent_prod* $(\lambda k. 1 + a \ k) \longleftrightarrow \text{summable } a \ (\text{is } ?lhs = ?rhs)$

proof

assume *?lhs*

then obtain p where $\text{raw_has_prod } (\lambda k. 1 + a \ k) \ 0 \ p$

by (*metis assms add_less_same_cancel2 convergent_prod_offset_0 not_one_less_zero*)

then have *to_p*: $(\lambda n. \prod_{k \leq n}. 1 + a \ k) \longrightarrow p$

by (*auto simp: raw_has_prod_def*)

moreover have *le*: $(\sum_{k \leq n}. a \ k) \leq (\prod_{k \leq n}. 1 + a \ k)$ for n

by (*rule sum_le_prod*) (*use assms less_le in force*)

have $(\prod_{k \leq n}. 1 + a \ k) \leq p$ for n

proof (*rule incseq_le [OF _ to_p]*)

```

    show incseq ( $\lambda n. \prod_{k \leq n}. 1 + a k$ )
      using assms by (auto simp: mono_def order.strict_implies_order intro!:
prod_mono2)
  qed
  with le have ( $\sum_{k \leq n}. a k$ )  $\leq p$  for  $n$ 
    by (metis order_trans)
  with assms bounded_imp_summable show ?rhs
    by (metis not_less order.asym)
next
  assume R: ?rhs
  have ( $\prod_{k \leq n}. 1 + a k$ )  $\leq \exp(\text{suminf } a)$  for  $n$ 
  proof -
    have ( $\prod_{k \leq n}. 1 + a k$ )  $\leq \exp(\sum_{k \leq n}. a k)$  for  $n$ 
      by (rule prod_le_exp_sum) (use assms less_le in force)
    moreover have  $\exp(\sum_{k \leq n}. a k) \leq \exp(\text{suminf } a)$  for  $n$ 
      unfolding exp_le_cancel_iff
      by (meson sum_le_suminf R assms finite_atMost less_eq_real_def)
    ultimately show ?thesis
      by (meson order_trans)
  qed
  then obtain L where L: ( $\lambda n. \prod_{k \leq n}. 1 + a k$ )  $\longrightarrow L$ 
    by (metis assms bounded_imp_convergent_prod convergent_prod_iff_nz_lim
le_add_same_cancel1 le_add_same_cancel2 less_le not_le zero_le_one)
  moreover have L  $\neq 0$ 
  proof
    assume L = 0
    with L have ( $\lambda n. \prod_{k \leq n}. 1 + a k$ )  $\longrightarrow 0$ 
      by simp
    moreover have ( $\prod_{k \leq n}. 1 + a k$ )  $> 1$  for  $n$ 
      by (simp add: assms less_1_prod)
    ultimately show False
      by (meson Lim_bounded2 not_one_le_zero less_imp_le)
  qed
  ultimately show ?lhs
    using assms convergent_prod_iff_nz_lim
    by (metis add_less_same_cancel1 less_le not_le zero_less_one)
qed

lemma exp_suminf_produinf_real:
  fixes f :: nat  $\Rightarrow$  real
  assumes  $ge0: \bigwedge n. f n \geq 0$  and ac: abs_convergent_prod ( $\lambda n. \exp(f n)$ )
  shows produinf ( $\lambda i. \exp(f i)$ ) = exp(suminf f)
proof -
  have summable f
    using ac unfolding abs_convergent_prod_conv_summable
  proof (elim summable_comparison_test')
    fix n
    have  $|f n| = f n$ 
      by (simp add: ge0)

```

3384

```
    also have ... ≤ exp (f n) - 1
      by (metis diff_diff_add exp_ge_add_one_self ge_iff_diff_ge_0)
    finally show norm (f n) ≤ norm (exp (f n) - 1)
      by simp
  qed
  then show ?thesis
    by (simp add: prodinf_exp)
qed
```

```
lemma has_prod_imp_sums_ln_real:
  fixes f :: nat ⇒ real
  assumes raw_has_prod f 0 p and 0:  $\bigwedge x. f x > 0$ 
  shows  $(\lambda i. \ln (f i))$  sums (ln p)
proof -
  have p > 0
    using assms unfolding prod_defs by (metis LIMSEQ_prod_nonneg less_eq_real_def)
  then show ?thesis
    using assms continuous_on_ln [of {0<..}  $\lambda x. x$ ]
    using continuous_on_tendsto_compose [of {0<..}  $\ln (\lambda n. \text{prod } f \{..n\})$  p]
    by (auto simp: prod_defs sums_def le ln_prod order_tendstoD)
qed
```

```
lemma summable_ln_real:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f x > 0$ 
  shows summable  $(\lambda i. \ln (f i))$ 
proof -
  obtain M p where raw_has_prod f M p
    using f convergent_prod_def by blast
  then consider i where  $i < M$   $f i = 0$  | p where raw_has_prod f 0 p
    using raw_has_prod_cases by blast
  then show ?thesis
  proof cases
    case 1
      with 0 show ?thesis
        by (metis less_irrefl)
    next
      case 2
        then show ?thesis
          using 0 has_prod_imp_sums_ln_real summable_def by blast
  qed
qed
```

```
lemma suminf_ln_real:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f x > 0$ 
  shows  $\text{suminf } (\lambda i. \ln (f i)) = \ln (\text{prodinf } f)$ 
proof -
  have f has_prod prodinf f
```

```

    by (simp add: f_has_prod_iff)
  then have raw_has_prod f 0 (prodinf f)
    by (metis 0_has_prod_def less_irrefl)
  then have  $(\lambda i. \ln (f i))$  sums ln (prodinf f)
    using 0_has_prod_imp_sums_ln_real by blast
  then show ?thesis
    by (rule sums_unique [symmetric])
qed

```

```

lemma prodinf_exp_real:
  fixes f :: nat  $\Rightarrow$  real
  assumes f: convergent_prod f and 0:  $\bigwedge x. f x > 0$ 
  shows prodinf f = exp (suminf  $(\lambda i. \ln (f i))$ )
  by (simp add: 0_f_less_0_prodinf_suminf_ln_real)

```

```

theorem Ln_prodinf_complex:
  fixes z :: nat  $\Rightarrow$  complex
  assumes z:  $\bigwedge j. z j \neq 0$  and  $\xi: \xi \neq 0$ 
  shows  $((\lambda n. \prod_{j \leq n}. z j) \longrightarrow \xi) \longleftrightarrow (\exists k. (\lambda n. (\sum_{j \leq n}. \ln (z j))) \longrightarrow Ln \xi + of\_int k * (of\_real(2*pi) * i))$  (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  have pnz:  $(\prod_{j \leq n}. z j) \neq 0$  for n
    using z by auto
  define  $\Theta$  where  $\Theta \equiv Arg \xi + 2*pi$ 
  then have  $\Theta > pi$ 
    using Arg_def mpi_less_Im_Ln by fastforce
  have  $\xi\_eq: \xi = cmod \xi * exp (i * \Theta)$ 
    using Arg_def Arg_eq  $\xi$  unfolding  $\Theta\_def$  by (simp add: algebra_simps exp_add)
  define  $\vartheta$  where  $\vartheta \equiv \lambda n. THE t. is\_Arg (\prod_{j \leq n}. z j) t \wedge t \in \{\Theta - pi <.. \Theta + pi\}$ 
  have uniq:  $\exists ! s. is\_Arg (\prod_{j \leq n}. z j) s \wedge s \in \{\Theta - pi <.. \Theta + pi\}$  for n
    using Argument_exists_unique [OF pnz] by metis
  have  $\vartheta: is\_Arg (\prod_{j \leq n}. z j) (\vartheta n)$  and  $\vartheta\_interval: \vartheta n \in \{\Theta - pi <.. \Theta + pi\}$  for n
  unfolding  $\vartheta\_def$ 
  using theI' [OF uniq] by metis+
  have  $\vartheta\_pos: \bigwedge j. \vartheta j > 0$ 
    using  $\vartheta\_interval$   $\langle \Theta > pi \rangle$  by simp (meson diff_gt_0_iff_gt less_trans)
  have  $(\prod_{j \leq n}. z j) = cmod (\prod_{j \leq n}. z j) * exp (i * \vartheta n)$  for n
    using  $\vartheta$  by (auto simp: is_Arg_def)
  then have eq:  $(\lambda n. \prod_{j \leq n}. z j) = (\lambda n. cmod (\prod_{j \leq n}. z j) * exp (i * \vartheta n))$ 
    by simp
  then have  $(\lambda n. (cmod (\prod_{j \leq n}. z j) * exp (i * (\vartheta n)))) \longrightarrow \xi$ 
    using L by force
  then obtain k where  $k: (\lambda j. \vartheta j - of\_int (k j) * (2 * pi)) \longrightarrow \Theta$ 
    using L by (subst (asm)  $\xi\_eq$ ) (auto simp add: eq_z  $\xi$  polar_convergence)
  moreover have  $\forall_F n$  in sequentially.  $k n = 0$ 

```

```

proof –
  have *:  $kj = 0$  if  $\text{dist } (vj - \text{real\_of\_int } kj * 2) V < 1$   $vj \in \{V - 1 <.. V + 1\}$ 
for  $kj \ vj \ V$ 
    using that by (auto simp: dist_norm)
  have  $\forall_F j$  in sequentially.  $\text{dist } (\vartheta j - \text{of\_int } (k j) * (2 * pi)) \Theta < pi$ 
    using tendstoD [OF k] pi_gt_zero by blast
  then show ?thesis
  proof (rule eventually_mono)
    fix  $j$ 
    assume  $d$ :  $\text{dist } (\vartheta j - \text{real\_of\_int } (k j) * (2 * pi)) \Theta < pi$ 
    show  $k j = 0$ 
      by (rule * [of  $\vartheta j/pi \ \Theta/pi$ ])
        (use  $\vartheta\_interval$  [of  $j$ ]  $d$  in  $\langle \text{simp\_all add: divide_simps dist\_norm} \rangle$ ))
    qed
  qed
ultimately have  $\vartheta \text{to}\Theta$ :  $\vartheta \longrightarrow \Theta$ 
  apply (simp only: tendsto_def)
  apply (erule all_forward imp_forward asm_rl)
  apply (drule (1) eventually_conj)
  apply (auto elim: eventually_mono)
  done
then have to0:  $(\lambda n. |\vartheta (\text{Suc } n) - \vartheta n|) \longrightarrow 0$ 
  by (metis (full_types) diff_self filterlim_sequentially_Suc tendsto_diff tendsto_rabs_zero)
  have  $\exists k$ .  $\text{Im } (\sum_{j \leq n}. \text{Ln } (z j)) - \text{of\_int } k * (2 * pi) = \vartheta n$  for  $n$ 
  proof (rule is_Arg_exp_diff_2pi)
    show is_Arg ( $\text{exp } (\sum_{j \leq n}. \text{Ln } (z j))$ ) ( $\vartheta n$ )
    using pnz  $\vartheta$  by (simp add: is_Arg_def exp_sum prod_norm)
  qed
then have  $\exists k$ .  $(\sum_{j \leq n}. \text{Im } (\text{Ln } (z j))) = \vartheta n + \text{of\_int } k * (2 * pi)$  for  $n$ 
  by (simp add: algebra_simps)
then obtain  $k$  where  $k$ :  $\bigwedge n. (\sum_{j \leq n}. \text{Im } (\text{Ln } (z j))) = \vartheta n + \text{of\_int } (k n) * (2 * pi)$ 
  by metis
obtain  $K$  where  $\forall_F n$  in sequentially.  $k n = K$ 
proof –
  have  $k\_le$ :  $(2 * pi) * |k (\text{Suc } n) - k n| \leq |\vartheta (\text{Suc } n) - \vartheta n| + |\text{Im } (\text{Ln } (z (\text{Suc } n)))|$  for  $n$ 
  proof –
    have  $(\sum_{j \leq \text{Suc } n}. \text{Im } (\text{Ln } (z j))) - (\sum_{j \leq n}. \text{Im } (\text{Ln } (z j))) = \text{Im } (\text{Ln } (z (\text{Suc } n)))$ 
    by simp
    then show ?thesis
    using  $k$  [of  $\text{Suc } n$ ]  $k$  [of  $n$ ] by (auto simp: abs_if algebra_simps)
  qed
have  $z \longrightarrow 1$ 
using L  $\xi$  convergent_prod_iff_nz_lim  $z$  by (blast intro: convergent_prod_imp_LIMSEQ)
with  $z$  have  $(\lambda n. \text{Ln } (z n)) \longrightarrow \text{Ln } 1$ 
using isCont_tendsto_compose [OF continuous_at_Ln] nonpos_Reals_one_I

```

```

by blast
then have  $(\lambda n. Ln (z n)) \longrightarrow 0$ 
  by simp
then have  $(\lambda n. |Im (Ln (z (Suc n)))|) \longrightarrow 0$ 
  by (metis LIMSEQ_unique <z  $\longrightarrow 1$ > continuous_at_Ln filterlim_sequentially_Suc
isCont_tendsto_compose nonpos_Reals_one_I tendsto_Im tendsto_rabs_zero_iff
zero_complex.simps(2))
then have  $\forall_F n$  in sequentially.  $|Im (Ln (z (Suc n)))| < 1$ 
  by (simp add: order_tendsto_iff)
moreover have  $\forall_F n$  in sequentially.  $|\vartheta (Suc n) - \vartheta n| < 1$ 
  using to0 by (simp add: order_tendsto_iff)
ultimately have  $\forall_F n$  in sequentially.  $(2 * \pi) * |k (Suc n) - k n| < 1 + 1$ 
proof (rule eventually_elim2)
  fix n
  assume  $|Im (Ln (z (Suc n)))| < 1$  and  $|\vartheta (Suc n) - \vartheta n| < 1$ 
  with k_le [of n] show  $2 * \pi * real\_of\_int |k (Suc n) - k n| < 1 + 1$ 
    by linarith
qed
then have  $\forall_F n$  in sequentially.  $real\_of\_int |k (Suc n) - k n| < 1$ 
proof (rule eventually_mono)
  fix n :: nat
  assume  $2 * \pi * |k (Suc n) - k n| < 1 + 1$ 
  then have  $|k (Suc n) - k n| < 2 / (2 * \pi)$ 
    by (simp add: field_simps)
  also have  $\dots < 1$ 
    using pi_ge_two by auto
  finally show  $real\_of\_int |k (Suc n) - k n| < 1$  .
qed
then obtain N where  $N: \bigwedge n. n \geq N \implies |k (Suc n) - k n| = 0$ 
  using eventually_sequentially_less_irrefl_of_int_abs by fastforce
have  $k (N+i) = k N$  for i
proof (induction i)
  case (Suc i)
  with N [of N+i] show ?case
    by auto
qed simp
then have  $\bigwedge n. n \geq N \implies k n = k N$ 
  using le_Suc_ex by auto
then show ?thesis
  by (force simp add: eventually_sequentially_intro: that)
qed
with  $\vartheta to \Theta$  have  $(\lambda n. (\sum_{j \leq n}. Im (Ln (z j)))) \longrightarrow \Theta + of\_int K * (2 * \pi)$ 
  by (simp add: k tendsto_add tendsto_mult tendsto_eventually)
moreover have  $(\lambda n. (\sum_{k \leq n}. Re (Ln (z k)))) \longrightarrow Re (Ln \xi)$ 
  using assms continuous_imp_tendsto [OF isCont_ln tendsto_norm [OF L]]
  by (simp add: o_def flip: prod_norm ln_prod)
ultimately show ?rhs
  by (rule_tac x=K+1 in exI) (auto simp: tendsto_complex_iff  $\Theta$ _def Arg_def
assms algebra_simps)

```

```

next
  assume ?rhs
  then obtain r where r: ( $\lambda n. (\sum_{k \leq n}. Ln (z k))$ )  $\longrightarrow$   $Ln \xi + of\_int r * (of\_real(2*pi) * i)$  ..
  have ( $\lambda n. exp (\sum_{k \leq n}. Ln (z k))$ )  $\longrightarrow$   $\xi$ 
  using assms continuous_imp_tendsto [OF isCont_exp r] exp_integer_2pi [of r]
  by (simp add: o_def exp_add algebra_simps)
  moreover have  $exp (\sum_{k \leq n}. Ln (z k)) = (\prod_{k \leq n}. z k)$  for  $n$ 
  by (simp add: exp_sum_add_eq_0_iff assms)
  ultimately show ?lhs
  by auto
qed

```

Prop 17.2 of Bak and Newman, Complex Analysis, p.242

```

proposition convergent_prod_iff_summable_complex:
  fixes  $z :: nat \Rightarrow complex$ 
  assumes  $\bigwedge k. z k \neq 0$ 
  shows  $convergent\_prod (\lambda k. z k) \longleftrightarrow summable (\lambda k. Ln (z k))$  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain p where  $p: (\lambda n. \prod_{k \leq n}. z k) \longrightarrow p$  and  $p \neq 0$ 
  using convergent_prod_LIMSEQ prodinf_nonzero add_eq_0_iff assms by
  fastforce
  then show ?rhs
  using Ln_producing_complex assms
  by (auto simp: prodinf_nonzero summable_def sums_def_le)
next
  assume  $R: ?rhs$ 
  have  $(\prod_{k \leq n}. z k) = exp (\sum_{k \leq n}. Ln (z k))$  for  $n$ 
  by (simp add: exp_sum_add_eq_0_iff assms)
  then have  $(\lambda n. \prod_{k \leq n}. z k) \longrightarrow exp (suminf (\lambda k. Ln (z k)))$ 
  using continuous_imp_tendsto [OF isCont_exp summable_LIMSEQ' [OF R]]
  by (simp add: o_def)
  then show ?lhs
  by (subst convergent_prod_iff_convergent) (auto simp: convergent_def tendsto_Lim assms add_eq_0_iff)
qed

```

Prop 17.3 of Bak and Newman, Complex Analysis

```

proposition summable_imp_convergent_prod_complex:
  fixes  $z :: nat \Rightarrow complex$ 
  assumes  $z: summable (\lambda k. norm (z k))$  and  $non0: \bigwedge k. z k \neq -1$ 
  shows  $convergent\_prod (\lambda k. 1 + z k)$ 
proof -
  obtain  $N$  where  $\bigwedge k. k \geq N \implies norm (z k) < 1/2$ 
  using summable_LIMSEQ_zero [OF z]
  by (metis diff_zero dist_norm half_gt_zero_iff less_numeral_extra(1) lim_sequentially tendsto_norm_zero_iff)

```



```

then have summable ( $\lambda k. Ln (1 + z k)$ )
  by (metis norm_Ln_le summable_comparison_test summable_mult z)
with non0 show ?thesis
  by (simp add: add_eq_0_iff convergent_prod_iff summable_complex)
qed

```

```

corollary summable_imp_convergent_prod_real:
  fixes z :: nat  $\Rightarrow$  real
  assumes z: summable ( $\lambda k. |z k|$ ) and non0:  $\bigwedge k. z k \neq -1$ 
  shows convergent_prod ( $\lambda k. 1 + z k$ )
proof -
  have  $\bigwedge k. (complex\_of\_real \circ z) k \neq -1$ 
    by (metis non0 o_apply of_real_1 of_real_eq_iff of_real_minus)
  with z
  have convergent_prod ( $\lambda k. 1 + (complex\_of\_real \circ z) k$ )
    by (auto intro: summable_imp_convergent_prod_complex)
  then show ?thesis
    using convergent_prod_of_real_iff [of  $\lambda k. 1 + z k$ ] by (simp add: o_def)
qed

```

```

lemma summable_Ln_complex:
  fixes z :: nat  $\Rightarrow$  complex
  assumes convergent_prod z  $\bigwedge k. z k \neq 0$ 
  shows summable ( $\lambda k. Ln (z k)$ )
  using convergent_prod_def assms convergent_prod_iff summable_complex by
  blast

```

9.22.10 Embeddings from the reals into some complete real normed field

```

lemma tendsto_eq_of_real_lim:
  assumes ( $\lambda n. of\_real (f n) :: 'a :: \{complete\_space, real\_normed\_field\}$ )  $\longrightarrow$  q
  shows q = of_real (lim f)
proof -
  have convergent ( $\lambda n. of\_real (f n) :: 'a$ )
    using assms convergent_def by blast
  then have convergent f
    unfolding convergent_def
    by (simp add: convergent_eq_Cauchy Cauchy_def)
  then show ?thesis
    by (metis LIMSEQ_unique assms convergentD sequentially_bot tendsto_Lim
  tendsto_of_real)
qed

```

```

lemma tendsto_eq_of_real:
  assumes ( $\lambda n. of\_real (f n) :: 'a :: \{complete\_space, real\_normed\_field\}$ )  $\longrightarrow$  q
  obtains r where q = of_real r
  using tendsto_eq_of_real_lim assms by blast

```

3390

```
lemma has_prod_of_real_iff [simp]:
  ( $\lambda n. \text{of\_real } (f\ n) :: 'a::\{\text{complete\_space, real\_normed\_field}\}$ ) has_prod_of_real
   $c \longleftrightarrow f \text{ has\_prod } c$ 
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (auto simp: prod_defs LIMSEQ_prod_0 tendsto_of_real_iff simp flip:
of_real_prod)
    using tendsto_eq_of_real
    by (metis of_real_0 tendsto_of_real_iff)
next
  assume ?rhs
  with tendsto_of_real_iff show ?lhs
    by (fastforce simp: prod_defs simp flip: of_real_prod)
qed

end
```

9.23 Sums over Infinite Sets

```
theory Infinite_Set_Sum
  imports Set_Integral Infinite_Sum
begin
```

Conflicting notation from *HOL-Analysis.Infinite_Sum*

```
no_notation Infinite_Sum.abs_summable_on (infixr abs'_summable'_on 46)
```

```
lemma sets_eq_countable:
  assumes countable A space M = A  $\wedge x. x \in A \implies \{x\} \in \text{sets } M$ 
  shows sets M = Pow A
proof (intro equalityI subsetI)
  fix X assume X  $\in$  Pow A
  hence  $(\bigcup x \in X. \{x\}) \in \text{sets } M$ 
    by (intro sets.countable_UN' countable_subset[OF _ assms(1)]) (auto intro!:
assms(3))
  also have  $(\bigcup x \in X. \{x\}) = X$  by auto
  finally show X  $\in$  sets M .
next
  fix X assume X  $\in$  sets M
  from sets.sets_into_space[OF this] and assms
  show X  $\in$  Pow A by simp
qed
```

```
lemma measure_eqI_countable':
  assumes spaces: space M = A space N = A
  assumes sets:  $\wedge x. x \in A \implies \{x\} \in \text{sets } M \wedge x. x \in A \implies \{x\} \in \text{sets } N$ 
  assumes A: countable A
```

```

  assumes eq:  $\bigwedge a. a \in A \implies \text{emeasure } M \{a\} = \text{emeasure } N \{a\}$ 
  shows  $M = N$ 
proof (rule measure_eqI_countable)
  show sets  $M = \text{Pow } A$ 
    by (intro sets_eq_countable assms)
  show sets  $N = \text{Pow } A$ 
    by (intro sets_eq_countable assms)
qed fact+

lemma count_space_PiM_finite:
  fixes  $B :: 'a \Rightarrow 'b \text{ set}$ 
  assumes finite  $A \wedge i. \text{countable } (B i)$ 
  shows  $\text{PiM } A (\lambda i. \text{count\_space } (B i)) = \text{count\_space } (\text{PiE } A B)$ 
proof (rule measure_eqI_countable')
  show space  $(\text{PiM } A (\lambda i. \text{count\_space } (B i))) = \text{PiE } A B$ 
    by (simp add: space_PiM)
  show space  $(\text{count\_space } (\text{PiE } A B)) = \text{PiE } A B$  by simp
next
  fix  $f$  assume  $f: f \in \text{PiE } A B$ 
  hence  $\text{PiE } A (\lambda x. \{f x\}) \in \text{sets } (\text{PiM } A (\lambda i. \text{count\_space } (B i)))$ 
    by (intro sets_PiM_I_finite assms) auto
  also from  $f$  have  $\text{PiE } A (\lambda x. \{f x\}) = \{f\}$ 
    by (intro PiE_singleton) (auto simp: PiE_def)
  finally show  $\{f\} \in \text{sets } (\text{PiM } A (\lambda i. \text{count\_space } (B i)))$  .
next
  interpret product_sigma_finite  $(\lambda i. \text{count\_space } (B i))$ 
  by (intro product_sigma_finite.intro sigma_finite_measure_count_space_countable
  assms)
  thm sigma_finite_measure_count_space
  fix  $f$  assume  $f: f \in \text{PiE } A B$ 
  hence  $\{f\} = \text{PiE } A (\lambda x. \{f x\})$ 
    by (intro PiE_singleton [symmetric]) (auto simp: PiE_def)
  also have  $\text{emeasure } (\text{PiM } A (\lambda i. \text{count\_space } (B i))) \dots =$ 
     $(\prod_{i \in A}. \text{emeasure } (\text{count\_space } (B i)) \{f i\})$ 
    using  $f$  assms by (subst emeasure_PiM) auto
  also have  $\dots = (\prod_{i \in A}. 1)$ 
    by (intro prod.cong refl, subst emeasure_count_space_finite) (use  $f$  in auto)
  also have  $\dots = \text{emeasure } (\text{count\_space } (\text{PiE } A B)) \{f\}$ 
    using  $f$  by (subst emeasure_count_space_finite) auto
  finally show  $\text{emeasure } (\text{PiM } A (\lambda i. \text{count\_space } (B i))) \{f\} =$ 
     $\text{emeasure } (\text{count\_space } (\text{PiE } A B)) \{f\}$  .
qed (simp_all add: countable_PiE assms)

definition abs_summable_on ::
   $('a \Rightarrow 'b :: \{\text{banach, second\_countable\_topology}\}) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ 
  (infix  $\text{abs}'\_summable'\_on$  50)
where
```

$f \text{ abs_summable_on } A \longleftrightarrow \text{integrable } (\text{count_space } A) f$

definition *infsetsum* ::

$('a \Rightarrow 'b :: \{ \text{banach}, \text{second_countable_topology} \}) \Rightarrow 'a \text{ set} \Rightarrow 'b$

where

$\text{infsetsum } f A = \text{lebesgue_integral } (\text{count_space } A) f$

syntax (ASCII)

$_ \text{infsetsum} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?INFSETSUM } _ : _ / _) [0, 51, 10] 10)$

syntax

$_ \text{infsetsum} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?}\sum_{a \in _} / _) [0, 51, 10] 10)$

translations — Beware of argument permutation!

$\sum_{a \in A}. b \Rightarrow \text{CONST } \text{infsetsum } (\lambda i. b) A$

syntax (ASCII)

$_ \text{uinfsetsum} :: \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow 'b \Rightarrow 'b :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?UINFSETSUM } _ : _ / _) [0, 51, 10] 10)$

syntax

$_ \text{uinfsetsum} :: \text{pttrn} \Rightarrow 'b \Rightarrow 'b :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?}\sum_{a \in _} / _) [0, 10] 10)$

translations — Beware of argument permutation!

$\sum_{a \in I}. b \Rightarrow \text{CONST } \text{infsetsum } (\lambda i. b) (\text{CONST UNIV})$

syntax (ASCII)

$_ \text{qinfsetsum} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?QINFSETSUM } _ | / _) [0, 0, 10] 10)$

syntax

$_ \text{qinfsetsum} :: \text{pttrn} \Rightarrow \text{bool} \Rightarrow 'a \Rightarrow 'a :: \{ \text{banach}, \text{second_countable_topology} \}$
 $((\text{?}\sum_{a \in _} | (_) / _) [0, 0, 10] 10)$

translations

$\sum_{a \in P}. t \Rightarrow \text{CONST } \text{infsetsum } (\lambda x. t) \{x. P\}$

print_translation <

let

$\text{fun } \text{sum_tr}' [\text{Abs } (x, Tx, t), \text{Const } (\text{const_syntax} \langle \text{Collect} \rangle, _) \$ \text{Abs } (y, Ty, P)] =$

if $x \langle \rangle y$ then raise Match

else

let

val $x' = \text{Syntax_Trans.mark_bound_body } (x, Tx);$

val $t' = \text{subst_bound } (x', t);$

val $P' = \text{subst_bound } (x', P);$

in

$\text{Syntax.const } \text{syntax_const} \langle _ \text{qinfsetsum} \rangle \$ \text{Syntax_Trans.mark_bound_abs}$
 $(x, Tx) \$ P' \$ t'$

end

```

  | sum_tr' _ = raise Match;
in [(const_syntax <infsetsum>, K sum_tr')] end
>

```

lemma *restrict_count_space_subset*:

$A \subseteq B \implies \text{restrict_space } (\text{count_space } B) A = \text{count_space } A$
by (*subst restrict_count_space*) (*simp_all add: Int_absorb2*)

lemma *abs_summable_on_restrict*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$
assumes $A \subseteq B$

shows $f \text{ abs_summable_on } A \longleftrightarrow (\lambda x. \text{indicator } A \ x *_{\mathbb{R}} f \ x) \text{ abs_summable_on } B$

proof –

have $\text{count_space } A = \text{restrict_space } (\text{count_space } B) A$
by (*rule restrict_count_space_subset [symmetric]*) *fact+*
also have $\text{integrable } \dots f \longleftrightarrow \text{set_integrable } (\text{count_space } B) A \ f$
by (*simp add: integrable_restrict_space set_integrable_def*)
finally show *?thesis*
unfolding *abs_summable_on_def set_integrable_def* .

qed

lemma *abs_summable_on_altdef*: $f \text{ abs_summable_on } A \longleftrightarrow \text{set_integrable } (\text{count_space } \text{UNIV}) A \ f$

unfolding *abs_summable_on_def set_integrable_def*
by (*metis (no_types) inf_top.right_neutral integrable_restrict_space restrict_count_space sets_UNIV*)

lemma *abs_summable_on_altdef'*:

$A \subseteq B \implies f \text{ abs_summable_on } A \longleftrightarrow \text{set_integrable } (\text{count_space } B) A \ f$
unfolding *abs_summable_on_def set_integrable_def*

by (*metis (no_types) Pow_iff abs_summable_on_def inf.orderE integrable_restrict_space restrict_count_space_subset sets_count_space space_count_space*)

lemma *abs_summable_on_norm_iff* [*simp*]:

$(\lambda x. \text{norm } (f \ x)) \text{ abs_summable_on } A \longleftrightarrow f \text{ abs_summable_on } A$
by (*simp add: abs_summable_on_def integrable_norm_iff*)

lemma *abs_summable_on_normI*: $f \text{ abs_summable_on } A \implies (\lambda x. \text{norm } (f \ x)) \text{ abs_summable_on } A$

by *simp*

lemma *abs_summable_complex_of_real* [*simp*]: $(\lambda n. \text{complex_of_real } (f \ n)) \text{ abs_summable_on } A \longleftrightarrow f \text{ abs_summable_on } A$

by (*simp add: abs_summable_on_def complex_of_real_integrable_eq*)

lemma *abs_summable_on_comparison_test*:

assumes $g \text{ abs_summable_on } A$

assumes $\bigwedge x. x \in A \implies \text{norm } (f x) \leq \text{norm } (g x)$
shows $f \text{ abs_summable_on } A$
using *assms Bochner_Integration.integrable_bound*[of count_space A g f]
unfolding *abs_summable_on_def* **by** (auto simp: AE_count_space)

lemma *abs_summable_on_comparison_test'*:

assumes $g \text{ abs_summable_on } A$
assumes $\bigwedge x. x \in A \implies \text{norm } (f x) \leq g x$
shows $f \text{ abs_summable_on } A$

proof (rule *abs_summable_on_comparison_test*[OF *assms(1)*, of f])

fix x **assume** $x \in A$
with *assms(2)* **have** $\text{norm } (f x) \leq g x$.
also have $\dots \leq \text{norm } (g x)$ **by** *simp*
finally show $\text{norm } (f x) \leq \text{norm } (g x)$.

qed

lemma *abs_summable_on_cong* [*cong*]:

$(\bigwedge x. x \in A \implies f x = g x) \implies A = B \implies (f \text{ abs_summable_on } A) \longleftrightarrow (g \text{ abs_summable_on } B)$

unfolding *abs_summable_on_def* **by** (*intro integrable_cong*) auto

lemma *abs_summable_on_cong_neutral*:

assumes $\bigwedge x. x \in A - B \implies f x = 0$
assumes $\bigwedge x. x \in B - A \implies g x = 0$
assumes $\bigwedge x. x \in A \cap B \implies f x = g x$
shows $f \text{ abs_summable_on } A \longleftrightarrow g \text{ abs_summable_on } B$
unfolding *abs_summable_on_altdef set_integrable_def* **using** *assms*
by (*intro Bochner_Integration.integrable_cong refl*)
 (auto simp: *indicator_def split: if_splits*)

lemma *abs_summable_on_restrict'*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second_countable_topology}\}$
assumes $A \subseteq B$

shows $f \text{ abs_summable_on } A \longleftrightarrow (\lambda x. \text{if } x \in A \text{ then } f x \text{ else } 0) \text{ abs_summable_on } B$

by (*subst abs_summable_on_restrict*[OF *assms*]) (*intro abs_summable_on_cong*, auto)

lemma *abs_summable_on_nat_iff*:

$f \text{ abs_summable_on } (A :: \text{nat set}) \longleftrightarrow \text{summable } (\lambda n. \text{if } n \in A \text{ then } \text{norm } (f n) \text{ else } 0)$

proof –

have $f \text{ abs_summable_on } A \longleftrightarrow \text{summable } (\lambda x. \text{norm } (\text{if } x \in A \text{ then } f x \text{ else } 0))$

by (*subst abs_summable_on_restrict'*[of UNIV])

(*simp_all add: abs_summable_on_def integrable_count_space_nat_iff*)

also have $(\lambda x. \text{norm } (\text{if } x \in A \text{ then } f x \text{ else } 0)) = (\lambda x. \text{if } x \in A \text{ then } \text{norm } (f x) \text{ else } 0)$

by auto

finally show ?thesis .
qed

lemma *abs_summable_on_nat_iff'*:
 $f \text{ abs_summable_on } (UNIV :: \text{nat set}) \longleftrightarrow \text{summable } (\lambda n. \text{norm } (f n))$
by (*subst abs_summable_on_nat_iff*) *auto*

lemma *nat_abs_summable_on_comparison_test*:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second_countable_topology}\}$
assumes $g \text{ abs_summable_on } I$
assumes $\bigwedge n. \llbracket n \geq N; n \in I \rrbracket \implies \text{norm } (f n) \leq g n$
shows $f \text{ abs_summable_on } I$
using *assms* **by** (*fastforce simp add: abs_summable_on_nat_iff intro: summable_comparison_test'*)

lemma *abs_summable_comparison_test_ev*:
assumes $g \text{ abs_summable_on } I$
assumes *eventually* $(\lambda x. x \in I \longrightarrow \text{norm } (f x) \leq g x)$ *sequentially*
shows $f \text{ abs_summable_on } I$
by (*metis (no_types, lifting) nat_abs_summable_on_comparison_test eventually_at_top_linorder assms*)

lemma *abs_summable_on_Cauchy*:
 $f \text{ abs_summable_on } (UNIV :: \text{nat set}) \longleftrightarrow (\forall e > 0. \exists N. \forall m \geq N. \forall n. (\sum x = m..<n. \text{norm } (f x)) < e)$
by (*simp add: abs_summable_on_nat_iff' summable_Cauchy sum_nonneg*)

lemma *abs_summable_on_finite* [*simp*]: $\text{finite } A \implies f \text{ abs_summable_on } A$
unfolding *abs_summable_on_def* **by** (*rule integrable_count_space*)

lemma *abs_summable_on_empty* [*simp, intro*]: $f \text{ abs_summable_on } \{\}$
by *simp*

lemma *abs_summable_on_subset*:
assumes $f \text{ abs_summable_on } B$ **and** $A \subseteq B$
shows $f \text{ abs_summable_on } A$
unfolding *abs_summable_on_altdef*
by (*rule set_integrable_subset*) (*insert assms, auto simp: abs_summable_on_altdef*)

lemma *abs_summable_on_union* [*intro*]:
assumes $f \text{ abs_summable_on } A$ **and** $f \text{ abs_summable_on } B$
shows $f \text{ abs_summable_on } (A \cup B)$
using *assms* **unfolding** *abs_summable_on_altdef* **by** (*intro set_integrable_Un*)
auto

lemma *abs_summable_on_insert_iff* [*simp*]:
 $f \text{ abs_summable_on insert } x A \longleftrightarrow f \text{ abs_summable_on } A$
proof *safe*
assume $f \text{ abs_summable_on insert } x A$
thus $f \text{ abs_summable_on } A$

```

    by (rule abs_summable_on_subset) auto
next
  assume f abs_summable_on A
  from abs_summable_on_union[OF this, of {x}]
  show f abs_summable_on insert x A by simp
qed

```

```

lemma abs_summable_sum:
  assumes  $\bigwedge x. x \in A \implies f x \text{ abs\_summable\_on } B$ 
  shows  $(\lambda y. \sum_{x \in A}. f x y) \text{ abs\_summable\_on } B$ 
  using assms unfolding abs_summable_on_def by (intro Bochner_Integration.integrable_sum)

```

```

lemma abs_summable_Re: f abs_summable_on A  $\implies (\lambda x. \text{Re } (f x)) \text{ abs\_summable\_on } A$ 
  by (simp add: abs_summable_on_def)

```

```

lemma abs_summable_Im: f abs_summable_on A  $\implies (\lambda x. \text{Im } (f x)) \text{ abs\_summable\_on } A$ 
  by (simp add: abs_summable_on_def)

```

```

lemma abs_summable_on_finite_diff:
  assumes f abs_summable_on A  $A \subseteq B$  finite (B - A)
  shows f abs_summable_on B
proof -
  have f abs_summable_on (A  $\cup$  (B - A))
    by (intro abs_summable_on_union assms abs_summable_on_finite)
  also from assms have  $A \cup (B - A) = B$  by blast
  finally show ?thesis .
qed

```

```

lemma abs_summable_on_reindex_bij_betw:
  assumes bij_betw g A B
  shows  $(\lambda x. f (g x)) \text{ abs\_summable\_on } A \iff f \text{ abs\_summable\_on } B$ 
proof -
  have *: count_space B = distr (count_space A) (count_space B) g
    by (rule distr_bij_count_space [symmetric]) fact
  show ?thesis unfolding abs_summable_on_def
    by (subst *, subst integrable_distr_eq[of _ _ count_space B])
      (insert assms, auto simp: bij_betw_def)
qed

```

```

lemma abs_summable_on_reindex:
  assumes  $(\lambda x. f (g x)) \text{ abs\_summable\_on } A$ 
  shows f abs_summable_on (g ' A)
proof -
  define g' where g' = inv_into A g
  from assms have  $(\lambda x. f (g x)) \text{ abs\_summable\_on } (g' ' g ' A)$ 
    by (rule abs_summable_on_subset) (auto simp: g'_def inv_into_into)
  also have ?this  $\iff (\lambda x. f (g (g' x))) \text{ abs\_summable\_on } (g ' A)$  unfolding

```



```

g'_def
  by (intro abs_summable_on_reindex_bij_betw [symmetric] inj_on_imp_bij_betw
      inj_on_inv_into) auto
  also have ...  $\longleftrightarrow$  f abs_summable_on (g ' A)
    by (intro abs_summable_on_cong refl) (auto simp: g'_def f_inv_into_f)
  finally show ?thesis .
qed

```

```

lemma abs_summable_on_reindex_iff:
  inj_on g A  $\implies$  ( $\lambda x. f (g x)$ ) abs_summable_on A  $\longleftrightarrow$  f abs_summable_on (g
  ' A)
  by (intro abs_summable_on_reindex_bij_betw inj_on_imp_bij_betw)

```

```

lemma abs_summable_on_Sigma_project2:
  fixes A :: 'a set and B :: 'a  $\Rightarrow$  'b set
  assumes f abs_summable_on (Sigma A B) x  $\in$  A
  shows ( $\lambda y. f (x, y)$ ) abs_summable_on (B x)
proof -
  from assms(2) have f abs_summable_on (Sigma {x} B)
    by (intro abs_summable_on_subset [OF assms(1)]) auto
  also have ?this  $\longleftrightarrow$  ( $\lambda z. f (x, \text{snd } z)$ ) abs_summable_on (Sigma {x} B)
    by (rule abs_summable_on_cong) auto
  finally have ( $\lambda y. f (x, y)$ ) abs_summable_on (snd ' Sigma {x} B)
    by (rule abs_summable_on_reindex)
  also have snd ' Sigma {x} B = B x
    using assms by (auto simp: image_iff)
  finally show ?thesis .
qed

```

```

lemma abs_summable_on_Times_swap:
  f abs_summable_on A  $\times$  B  $\longleftrightarrow$  ( $\lambda(x,y). f (y,x)$ ) abs_summable_on B  $\times$  A
proof -
  have bij: bij_betw ( $\lambda(x,y). (y,x)$ ) (B  $\times$  A) (A  $\times$  B)
    by (auto simp: bij_betw_def inj_on_def)
  show ?thesis
    by (subst abs_summable_on_reindex_bij_betw[OF bij, of f, symmetric])
      (simp_all add: case_prod_unfold)
qed

```

```

lemma abs_summable_on_0 [simp, intro]: ( $\lambda_. 0$ ) abs_summable_on A
  by (simp add: abs_summable_on_def)

```

```

lemma abs_summable_on_uminus [intro]:
  f abs_summable_on A  $\implies$  ( $\lambda x. -f x$ ) abs_summable_on A
  unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_minus)

```

```

lemma abs_summable_on_add [intro]:
  assumes f abs_summable_on A and g abs_summable_on A
  shows ( $\lambda x. f x + g x$ ) abs_summable_on A

```

using *assms* **unfolding** *abs_summable_on_def* **by** (rule *Bochner_Integration.integrable_add*)

lemma *abs_summable_on_diff* [*intro*]:

assumes *f* *abs_summable_on* *A* **and** *g* *abs_summable_on* *A*

shows $(\lambda x. f\ x - g\ x)$ *abs_summable_on* *A*

using *assms* **unfolding** *abs_summable_on_def* **by** (rule *Bochner_Integration.integrable_diff*)

lemma *abs_summable_on_scaleR_left* [*intro*]:

assumes $c \neq 0 \implies f$ *abs_summable_on* *A*

shows $(\lambda x. f\ x *_{\mathbb{R}} c)$ *abs_summable_on* *A*

using *assms* **unfolding** *abs_summable_on_def* **by** (intro *Bochner_Integration.integrable_scaleR_left*)

lemma *abs_summable_on_scaleR_right* [*intro*]:

assumes $c \neq 0 \implies f$ *abs_summable_on* *A*

shows $(\lambda x. c *_{\mathbb{R}} f\ x)$ *abs_summable_on* *A*

using *assms* **unfolding** *abs_summable_on_def* **by** (intro *Bochner_Integration.integrable_scaleR_right*)

lemma *abs_summable_on_cmult_right* [*intro*]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real_normed_algebra, second_countable_topology}\}$

assumes $c \neq 0 \implies f$ *abs_summable_on* *A*

shows $(\lambda x. c * f\ x)$ *abs_summable_on* *A*

using *assms* **unfolding** *abs_summable_on_def* **by** (intro *Bochner_Integration.integrable_mult_right*)

lemma *abs_summable_on_cmult_left* [*intro*]:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real_normed_algebra, second_countable_topology}\}$

assumes $c \neq 0 \implies f$ *abs_summable_on* *A*

shows $(\lambda x. f\ x * c)$ *abs_summable_on* *A*

using *assms* **unfolding** *abs_summable_on_def* **by** (intro *Bochner_Integration.integrable_mult_left*)

lemma *abs_summable_on_prod_PiE*:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{real_normed_field, banach, second_countable_topology}\}$

assumes *finite*: *finite* *A* **and** *countable*: $\bigwedge x. x \in A \implies \text{countable } (B\ x)$

assumes *summable*: $\bigwedge x. x \in A \implies f\ x$ *abs_summable_on* *B* *x*

shows $(\lambda g. \prod_{x \in A}. f\ x\ (g\ x))$ *abs_summable_on* *PiE* *A* *B*

proof –

define *B'* **where** $B' = (\lambda x. \text{if } x \in A \text{ then } B\ x \text{ else } \{\})$

from *assms* **have** [*simp*]: *countable* (*B'* *x*) **for** *x*

by (*auto simp: B'_def*)

then interpret *product_sigma_finite* *count_space* $\circ B'$

unfolding *o_def* **by** (intro *product_sigma_finite.intro* *sigma_finite_measure_count_space_countable*)

from *assms* **have** *integrable* (*PiM* *A* (*count_space* $\circ B'$)) $(\lambda g. \prod_{x \in A}. f\ x\ (g\ x))$

by (*intro* *product_integrable_prod*) (*auto simp: abs_summable_on_def B'_def*)

also have *PiM* *A* (*count_space* $\circ B'$) = *count_space* (*PiE* *A* *B'*)

unfolding *o_def* **using** *finite* **by** (*intro* *count_space_PiM_finite*) *simp_all*

also have *PiE* *A* *B'* = *PiE* *A* *B* **by** (*intro* *PiE_cong*) (*simp_all add: B'_def*)

finally show *?thesis* **by** (*simp add: abs_summable_on_def*)

qed

lemma *not_summable_infsetsum_eq*:

$\neg f \text{ abs_summable_on } A \implies \text{infsetsum } f A = 0$

by (*simp add: abs_summable_on_def infsetsum_def not_integrable_integral_eq*)

lemma *infsetsum_altdef*:

$\text{infsetsum } f A = \text{set_lebesgue_integral } (\text{count_space } UNIV) A f$

unfolding *set_lebesgue_integral_def*

by (*subst integral_restrict_space [symmetric]*)

(*auto simp: restrict_count_space_subset infsetsum_def*)

lemma *infsetsum_altdef'*:

$A \subseteq B \implies \text{infsetsum } f A = \text{set_lebesgue_integral } (\text{count_space } B) A f$

unfolding *set_lebesgue_integral_def*

by (*subst integral_restrict_space [symmetric]*)

(*auto simp: restrict_count_space_subset infsetsum_def*)

lemma *nn_integral_conv_infsetsum*:

assumes $f \text{ abs_summable_on } A \wedge x. x \in A \implies f x \geq 0$

shows $\text{nn_integral } (\text{count_space } A) f = \text{ennreal } (\text{infsetsum } f A)$

using *assms unfolding infsetsum_def abs_summable_on_def*

by (*subst nn_integral_eq_integral*) *auto*

lemma *infsetsum_conv_nn_integral*:

assumes $\text{nn_integral } (\text{count_space } A) f \neq \infty \wedge x. x \in A \implies f x \geq 0$

shows $\text{infsetsum } f A = \text{enn2real } (\text{nn_integral } (\text{count_space } A) f)$

unfolding *infsetsum_def using assms*

by (*subst integral_eq_nn_integral*) *auto*

lemma *infsetsum_cong [cong]*:

$(\wedge x. x \in A \implies f x = g x) \implies A = B \implies \text{infsetsum } f A = \text{infsetsum } g B$

unfolding *infsetsum_def* **by** (*intro Bochner_Integration.integral_cong*) *auto*

lemma *infsetsum_0 [simp]*: $\text{infsetsum } (\lambda_. 0) A = 0$

by (*simp add: infsetsum_def*)

lemma *infsetsum_all_0*: $(\wedge x. x \in A \implies f x = 0) \implies \text{infsetsum } f A = 0$

by *simp*

lemma *infsetsum_nonneg*: $(\wedge x. x \in A \implies f x \geq (0::\text{real})) \implies \text{infsetsum } f A \geq 0$

unfolding *infsetsum_def* **by** (*rule Bochner_Integration.integral_nonneg*) *auto*

lemma *sum_infsetsum*:

assumes $\wedge x. x \in A \implies f x \text{ abs_summable_on } B$

shows $(\sum_{x \in A} \sum_{a \in B} f x a) = (\sum_{a \in B} \sum_{x \in A} f x a)$

using *assms* **by** (*simp add: infsetsum_def abs_summable_on_def Bochner_Integration.integral_sum*)

lemma *Re_infsetsum*: $f \text{ abs_summable_on } A \implies \text{Re } (\text{infsetsum } f A) = (\sum_a x \in A.$

3400

Re (*f x*)
by (*simp add: infsetsum_def abs_summable_on_def*)

lemma *Im_infsetsum*: *f abs_summable_on A* \implies *Im* (*infsetsum f A*) = $(\sum_{a \in A} \text{Im } (f a))$
by (*simp add: infsetsum_def abs_summable_on_def*)

lemma *infsetsum_of_real*:
shows *infsetsum* ($\lambda x. \text{of_real } (f x)$)
:: 'a :: {*real_normed_algebra_1*, *banach*, *second_countable_topology*, *real_inner*}
A =
of_real (*infsetsum f A*)
unfolding *infsetsum_def*
by (*rule integral_bounded_linear'[OF bounded_linear_of_real bounded_linear_inner_left[of 1]]*) *auto*

lemma *infsetsum_finite* [*simp*]: *finite A* \implies *infsetsum f A* = $(\sum_{x \in A} f x)$
by (*simp add: infsetsum_def lebesgue_integral_count_space_finite*)

lemma *infsetsum_nat*:
assumes *f abs_summable_on A*
shows *infsetsum f A* = $(\sum n. \text{if } n \in A \text{ then } f n \text{ else } 0)$
proof –
from *assms* have *infsetsum f A* = $(\sum n. \text{indicator } A n *_{\mathbb{R}} f n)$
unfolding *infsetsum_altdef abs_summable_on_altdef set_lebesgue_integral_def set_integrable_def*
by (*subst integral_count_space_nat*) *auto*
also have $(\lambda n. \text{indicator } A n *_{\mathbb{R}} f n) = (\lambda n. \text{if } n \in A \text{ then } f n \text{ else } 0)$
by *auto*
finally show ?*thesis* .
qed

lemma *infsetsum_nat'*:
assumes *f abs_summable_on UNIV*
shows *infsetsum f UNIV* = $(\sum n. f n)$
using *assms* by (*subst infsetsum_nat*) *auto*

lemma *sums_infsetsum_nat*:
assumes *f abs_summable_on A*
shows $(\lambda n. \text{if } n \in A \text{ then } f n \text{ else } 0)$ *sums infsetsum f A*
proof –
from *assms* have *summable* ($\lambda n. \text{if } n \in A \text{ then } \text{norm } (f n) \text{ else } 0$)
by (*simp add: abs_summable_on_nat_iff*)
also have $(\lambda n. \text{if } n \in A \text{ then } \text{norm } (f n) \text{ else } 0) = (\lambda n. \text{norm } (\text{if } n \in A \text{ then } f n \text{ else } 0))$
by *auto*
finally have *summable* ($\lambda n. \text{if } n \in A \text{ then } f n \text{ else } 0$)
by (*rule summable_norm_cancel*)
with *assms* show ?*thesis*

by (auto simp: sums_iff infsetsum_nat)
qed

lemma *sums_infsetsum_nat'*:
assumes *f abs_summable_on UNIV*
shows *f sums infsetsum f UNIV*
using *sums_infsetsum_nat [OF assms]* by *simp*

lemma *infsetsum_Un_disjoint*:
assumes *f abs_summable_on A f abs_summable_on B A ∩ B = {}*
shows *infsetsum f (A ∪ B) = infsetsum f A + infsetsum f B*
using *assms unfolding infsetsum_altdef abs_summable_on_altdef*
by (*subst set_integral_Un*) *auto*

lemma *infsetsum_Diff*:
assumes *f abs_summable_on B A ⊆ B*
shows *infsetsum f (B - A) = infsetsum f B - infsetsum f A*
proof -
have *infsetsum f ((B - A) ∪ A) = infsetsum f (B - A) + infsetsum f A*
using *assms(2)* by (*intro infsetsum_Un_disjoint abs_summable_on_subset [OF*
assms(1)]) *auto*
also from *assms(2)* have *(B - A) ∪ A = B*
by *auto*
ultimately show *?thesis*
by (*simp add: algebra_simps*)
qed

lemma *infsetsum_Un_Int*:
assumes *f abs_summable_on (A ∪ B)*
shows *infsetsum f (A ∪ B) = infsetsum f A + infsetsum f B - infsetsum f (A*
∩ B)
proof -
have *A ∪ B = A ∪ (B - A ∩ B)*
by *auto*
also have *infsetsum f ... = infsetsum f A + infsetsum f (B - A ∩ B)*
by (*intro infsetsum_Un_disjoint abs_summable_on_subset [OF assms]*) *auto*
also have *infsetsum f (B - A ∩ B) = infsetsum f B - infsetsum f (A ∩ B)*
by (*intro infsetsum_Diff abs_summable_on_subset [OF assms]*) *auto*
finally show *?thesis*
by (*simp add: algebra_simps*)
qed

lemma *infsetsum_reindex_bij_betw*:
assumes *bij_betw g A B*
shows *infsetsum (λx. f (g x)) A = infsetsum f B*
proof -
have *: *count_space B = distr (count_space A) (count_space B) g*
by (*rule distr_bij_count_space [symmetric]*) *fact*
show *?thesis unfolding infsetsum_def*

3402

by (*subst* *, *subst integral distr*[of __ count_space B])
(*insert assms*, *auto simp: bij_betw_def*)

qed

theorem *infsetsum_reindex*:

assumes *inj_on* g A

shows $\text{infsetsum } f (g \text{ ` } A) = \text{infsetsum } (\lambda x. f (g x)) A$

by (*intro infsetsum_reindex_bij_betw [symmetric] inj_on_imp_bij_betw assms*)

lemma *infsetsum_cong_neutral*:

assumes $\bigwedge x. x \in A - B \implies f x = 0$

assumes $\bigwedge x. x \in B - A \implies g x = 0$

assumes $\bigwedge x. x \in A \cap B \implies f x = g x$

shows $\text{infsetsum } f A = \text{infsetsum } g B$

unfolding *infsetsum_altdef set_lebesgue_integral_def* **using** *assms*

by (*intro Bochner_Integration.integral_cong refl*)

(*auto simp: indicator_def split: if_splits*)

lemma *infsetsum_mono_neutral*:

fixes *f g* :: 'a \implies real

assumes *f abs_summable_on* A **and** *g abs_summable_on* B

assumes $\bigwedge x. x \in A \implies f x \leq g x$

assumes $\bigwedge x. x \in A - B \implies f x \leq 0$

assumes $\bigwedge x. x \in B - A \implies g x \geq 0$

shows $\text{infsetsum } f A \leq \text{infsetsum } g B$

using *assms* **unfolding** *infsetsum_altdef set_lebesgue_integral_def abs_summable_on_altdef set_integrable_def*

by (*intro Bochner_Integration.integral_mono*) (*auto simp: indicator_def*)

lemma *infsetsum_mono_neutral_left*:

fixes *f g* :: 'a \implies real

assumes *f abs_summable_on* A **and** *g abs_summable_on* B

assumes $\bigwedge x. x \in A \implies f x \leq g x$

assumes $A \subseteq B$

assumes $\bigwedge x. x \in B - A \implies g x \geq 0$

shows $\text{infsetsum } f A \leq \text{infsetsum } g B$

using $\langle A \subseteq B \rangle$ **by** (*intro infsetsum_mono_neutral assms*) *auto*

lemma *infsetsum_mono_neutral_right*:

fixes *f g* :: 'a \implies real

assumes *f abs_summable_on* A **and** *g abs_summable_on* B

assumes $\bigwedge x. x \in A \implies f x \leq g x$

assumes $B \subseteq A$

assumes $\bigwedge x. x \in A - B \implies f x \leq 0$

shows $\text{infsetsum } f A \leq \text{infsetsum } g B$

using $\langle B \subseteq A \rangle$ **by** (*intro infsetsum_mono_neutral assms*) *auto*

lemma *infsetsum_mono*:

fixes *f g* :: 'a \implies real

assumes f *abs_summable_on* A **and** g *abs_summable_on* A
assumes $\bigwedge x. x \in A \implies f\ x \leq g\ x$
shows $\text{infsetsum } f\ A \leq \text{infsetsum } g\ A$
by (*intro infsetsum_mono_neutral assms*) *auto*

lemma *norm_infsetsum_bound*:
 $\text{norm } (\text{infsetsum } f\ A) \leq \text{infsetsum } (\lambda x. \text{norm } (f\ x))\ A$
unfolding *abs_summable_on_def infsetsum_def*
by (*rule Bochner_Integration.integral_norm_bound*)

theorem *infsetsum_Sigma*:
fixes $A :: 'a\ \text{set}$ **and** $B :: 'a \Rightarrow 'b\ \text{set}$
assumes [*simp*]: *countable* A **and** $\bigwedge i. \text{countable } (B\ i)$
assumes *summable*: f *abs_summable_on* (*Sigma* $A\ B$)
shows $\text{infsetsum } f\ (\text{Sigma } A\ B) = \text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f\ (x, y))\ (B\ x))\ A$
proof –
define B' **where** $B' = (\bigcup_{i \in A}. B\ i)$
have [*simp*]: *countable* B'
unfolding B'_def **by** (*intro countable_UN assms*)
interpret *pair_sigma_finite count_space A count_space B'*
by (*intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable*)
fact+

have *integrable* (*count_space* ($A \times B'$)) ($\lambda z. \text{indicator } (\text{Sigma } A\ B)\ z\ *_R\ f\ z$)
using *summable*
by (*metis* (*mono_tags, lifting*) *abs_summable_on_altdef abs_summable_on_def integrable_cong integrable_mult_indicator set_integrable_def sets_UNIV*)
also have $?this \longleftrightarrow \text{integrable } (\text{count_space } A \otimes_M \text{count_space } B')\ (\lambda(x, y). \text{indicator } (B\ x)\ y\ *_R\ f\ (x, y))$
by (*intro Bochner_Integration.integrable_cong*)
(auto simp: pair_measure_countable indicator_def split: if_splits)
finally have *integrable*: \dots .

have $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f\ (x, y))\ (B\ x))\ A =$
 $(\int x. \text{infsetsum } (\lambda y. f\ (x, y))\ (B\ x)\ \partial \text{count_space } A)$
unfolding *infsetsum_def* **by** *simp*
also have $\dots = (\int x. \int y. \text{indicator } (B\ x)\ y\ *_R\ f\ (x, y)\ \partial \text{count_space } B')$
 $\partial \text{count_space } A)$
proof (*rule Bochner_Integration.integral_cong [OF refl]*)
show $\bigwedge x. x \in \text{space } A \implies$
 $(\sum_{a \in B\ x} f\ (x, a)) = \text{LINT } y | \text{count_space } B'. \text{indicat_real } (B\ x)\ y\ *_R\ f$
 (x, y)
using *infsetsum_altdef'[of _ B]*
unfolding *set_lebesgue_integral_def B'_def*
by *auto*
qed
also have $\dots = (\int (x, y). \text{indicator } (B\ x)\ y\ *_R\ f\ (x, y)\ \partial (\text{count_space } A \otimes_M \text{count_space } B'))$

by (subst integral_fst [OF integrable]) auto
 also have ... = $(\int z. \text{indicator } (\text{Sigma } A \ B) \ z \ *_{\mathbb{R}} \ f \ z \ \partial \text{count_space } (A \times B'))$
 by (intro Bochner_Integration.integral_cong)
 (auto simp: pair_measure_countable indicator_def split: if_splits)
 also have ... = infsetsum f (Sigma A B)
 unfolding set_lebesgue_integral_def [symmetric]
 by (rule infsetsum_altdef' [symmetric]) (auto simp: B'_def)
 finally show ?thesis ..

qed

lemma infsetsum_Sigma':

fixes A :: 'a set and B :: 'a \Rightarrow 'b set
 assumes [simp]: countable A and $\bigwedge i. \text{countable } (B \ i)$
 assumes summable: $(\lambda(x,y). f \ x \ y) \text{ abs_summable_on } (\text{Sigma } A \ B)$
 shows infsetsum $(\lambda x. \text{infsetsum } (\lambda y. f \ x \ y) \ (B \ x)) \ A = \text{infsetsum } (\lambda(x,y). f \ x \ y) \ (\text{Sigma } A \ B)$
 using assms by (subst infsetsum_Sigma) auto

lemma infsetsum_Times:

fixes A :: 'a set and B :: 'b set
 assumes [simp]: countable A and countable B
 assumes summable: $f \text{ abs_summable_on } (A \times B)$
 shows $\text{infsetsum } f \ (A \times B) = \text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f \ (x, y)) \ B) \ A$
 using assms by (subst infsetsum_Sigma) auto

lemma infsetsum_Times':

fixes A :: 'a set and B :: 'b set
 fixes f :: 'a \Rightarrow 'b \Rightarrow 'c :: {banach, second_countable_topology}
 assumes [simp]: countable A and [simp]: countable B
 assumes summable: $(\lambda(x,y). f \ x \ y) \text{ abs_summable_on } (A \times B)$
 shows $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f \ x \ y) \ B) \ A = \text{infsetsum } (\lambda(x,y). f \ x \ y) \ (A \times B)$
 using assms by (subst infsetsum_Times) auto

lemma infsetsum_swap:

fixes A :: 'a set and B :: 'b set
 fixes f :: 'a \Rightarrow 'b \Rightarrow 'c :: {banach, second_countable_topology}
 assumes [simp]: countable A and [simp]: countable B
 assumes summable: $(\lambda(x,y). f \ x \ y) \text{ abs_summable_on } A \times B$
 shows $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f \ x \ y) \ B) \ A = \text{infsetsum } (\lambda y. \text{infsetsum } (\lambda x. f \ x \ y) \ A) \ B$

proof -

from summable have summable': $(\lambda(x,y). f \ y \ x) \text{ abs_summable_on } B \times A$
 by (subst abs_summable_on_Times_swap) auto
 have bij: $\text{bij_betw } (\lambda(x, y). (y, x)) \ (B \times A) \ (A \times B)$
 by (auto simp: bij_betw_def inj_on_def)
 have $\text{infsetsum } (\lambda x. \text{infsetsum } (\lambda y. f \ x \ y) \ B) \ A = \text{infsetsum } (\lambda(x,y). f \ x \ y) \ (A \times B)$
 using summable by (subst infsetsum_Times) auto


```

also have ... = infsetsum ( $\lambda(x,y). f y x$ ) ( $B \times A$ )
  by (subst infsetsum_reindex_bij_betw[OF bij, of  $\lambda(x,y). f x y$ , symmetric])
    (simp_all add: case_prod_unfold)
also have ... = infsetsum ( $\lambda y. infsetsum (\lambda x. f x y) A$ )  $B$ 
  using summable' by (subst infsetsum_Times) auto
finally show ?thesis .
qed

theorem abs_summable_on_Sigma_iff:
  assumes [simp]: countable  $A$  and  $\bigwedge x. x \in A \implies$  countable ( $B x$ )
  shows  $f$  abs_summable_on Sigma  $A B \iff$ 
    ( $\forall x \in A. (\lambda y. f (x, y))$  abs_summable_on  $B x$ )  $\wedge$ 
    ( $(\lambda x. infsetsum (\lambda y. norm (f (x, y))) (B x))$  abs_summable_on  $A$ )

proof safe
  define  $B'$  where  $B' = (\bigcup_{x \in A}. B x)$ 
  have [simp]: countable  $B'$ 
    unfolding  $B'_def$  using assms by auto
  interpret pair_sigma_finite count_space  $A$  count_space  $B'$ 
  by (intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable)
fact+
  {
    assume *:  $f$  abs_summable_on Sigma  $A B$ 
    thus ( $\lambda y. f (x, y)$ ) abs_summable_on  $B x$  if  $x \in A$  for  $x$ 
      using that by (rule abs_summable_on_Sigma_project2)

    have set_integrable (count_space ( $A \times B'$ )) (Sigma  $A B$ ) ( $\lambda z. norm (f z)$ )
      using abs_summable_on_normI[OF *]
      by (subst abs_summable_on_altdef' [symmetric]) (auto simp:  $B'_def$ )
    also have count_space ( $A \times B'$ ) = count_space  $A \otimes_M$  count_space  $B'$ 
      by (simp add: pair_measure_countable)
    finally have integrable (count_space  $A$ )
      ( $\lambda x. lebesgue\_integral$  (count_space  $B'$ )
        ( $\lambda y. indicator$  (Sigma  $A B$ ) ( $x, y$ ) * $_R$  norm ( $f (x, y)$ )))
      unfolding set_integrable_def by (rule integrable_fst')
    also have ?this  $\iff$  integrable (count_space  $A$ )
      ( $\lambda x. lebesgue\_integral$  (count_space  $B'$ )
        ( $\lambda y. indicator$  ( $B x$ )  $y$  * $_R$  norm ( $f (x, y)$ )))
      by (intro integrable_cong_refl) (simp_all add: indicator_def)
    also have ...  $\iff$  integrable (count_space  $A$ ) ( $\lambda x. infsetsum (\lambda y. norm (f (x,$ 
 $y))) (B x)$ )
      unfolding set_lebesgue_integral_def [symmetric]
      by (intro integrable_cong_refl infsetsum_altdef' [symmetric]) (auto simp:
 $B'_def$ )
    also have ...  $\iff$  ( $\lambda x. infsetsum (\lambda y. norm (f (x, y))) (B x)$ ) abs_summable_on
 $A$ 
      by (simp add: abs_summable_on_def)
    finally show ... .
  }
  {

```

```

assume *:  $\forall x \in A. (\lambda y. f(x, y)) \text{ abs\_summable\_on } B$ 
assume  $(\lambda x. \sum_{y \in B} x. \text{ norm } (f(x, y))) \text{ abs\_summable\_on } A$ 
also have  $?this \longleftrightarrow (\lambda x. \int_{y \in B} x. \text{ norm } (f(x, y)) \partial \text{count\_space } B') \text{ abs\_summable\_on } A$ 
by (intro abs\_summable\_on\_cong refl infsetsum\_altdef') (auto simp: B'_def)
also have  $\dots \longleftrightarrow (\lambda x. \int y. \text{ indicator } (\text{Sigma } A \ B) (x, y) *_R \text{ norm } (f(x, y)) \partial \text{count\_space } B')$ 
 $\text{ abs\_summable\_on } A$  (is  $\_ \longleftrightarrow ?h \text{ abs\_summable\_on } \_$ )
unfolding set\_lebesgue\_integral\_def
by (intro abs\_summable\_on\_cong) (auto simp: indicator\_def)
also have  $\dots \longleftrightarrow \text{ integrable } (\text{count\_space } A) ?h$ 
by (simp add: abs\_summable\_on\_def)
finally have **:  $\dots$  .

have  $\text{ integrable } (\text{count\_space } A \otimes_M \text{count\_space } B') (\lambda z. \text{ indicator } (\text{Sigma } A \ B) z *_R f z)$ 
proof (rule Fubini\_integrable, goal\_cases)
case 3
{
fix  $x$  assume  $x: x \in A$ 
with * have  $(\lambda y. f(x, y)) \text{ abs\_summable\_on } B$ 
by blast
also have  $?this \longleftrightarrow \text{ integrable } (\text{count\_space } B')$ 
 $(\lambda y. \text{ indicator } (B \ x) y *_R f(x, y))$ 
unfolding set\_integrable\_def [symmetric]
using  $x$  by (intro abs\_summable\_on\_altdef') (auto simp: B'_def)
also have  $(\lambda y. \text{ indicator } (B \ x) y *_R f(x, y)) =$ 
 $(\lambda y. \text{ indicator } (\text{Sigma } A \ B) (x, y) *_R f(x, y))$ 
using  $x$  by (auto simp: indicator\_def)
finally have  $\text{ integrable } (\text{count\_space } B')$ 
 $(\lambda y. \text{ indicator } (\text{Sigma } A \ B) (x, y) *_R f(x, y))$  .
}
thus  $?case$  by (auto simp: AE\_count\_space)
qed (insert **, auto simp: pair\_measure\_countable)
moreover have  $\text{count\_space } A \otimes_M \text{count\_space } B' = \text{count\_space } (A \times B')$ 
by (simp add: pair\_measure\_countable)
moreover have  $\text{set\_integrable } (\text{count\_space } (A \times B')) (\text{Sigma } A \ B) f \longleftrightarrow$ 
 $f \text{ abs\_summable\_on } \text{Sigma } A \ B$ 
by (rule abs\_summable\_on\_altdef' [symmetric]) (auto simp: B'_def)
ultimately show  $f \text{ abs\_summable\_on } \text{Sigma } A \ B$ 
by (simp add: set\_integrable\_def)
}
qed

```

lemma abs_summable_on_Sigma_project1:

```

assumes  $(\lambda(x,y). f \ x \ y) \text{ abs\_summable\_on } \text{Sigma } A \ B$ 
assumes [simp]:  $\text{countable } A$  and  $\bigwedge x. x \in A \implies \text{countable } (B \ x)$ 
shows  $(\lambda x. \text{ infsetsum } (\lambda y. \text{ norm } (f \ x \ y)) (B \ x)) \text{ abs\_summable\_on } A$ 
using  $\text{assms}$  by (subst (asm) abs\_summable\_on\_Sigma\_iff) auto

```

lemma *abs_summable_on_Sigma_project1'*:
assumes $(\lambda(x,y). f x y)$ *abs_summable_on_Sigma* $A B$
assumes [*simp*]: *countable* A **and** $\bigwedge x. x \in A \implies \text{countable } (B x)$
shows $(\lambda x. \text{infsetsum } (\lambda y. f x y) (B x))$ *abs_summable_on* A
by (*intro abs_summable_on_comparison_test'* [*OF abs_summable_on_Sigma_project1* [*OF assms*]]
norm_infsetsum_bound)

theorem *infsetsum_prod_PiE*:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{real_normed_field, banach, second_countable_topology}\}$
assumes *finite*: *finite* A **and** *countable*: $\bigwedge x. x \in A \implies \text{countable } (B x)$
assumes *summable*: $\bigwedge x. x \in A \implies f x$ *abs_summable_on* $B x$
shows $\text{infsetsum } (\lambda g. \prod_{x \in A}. f x (g x)) (PiE A B) = (\prod_{x \in A}. \text{infsetsum } (f x) (B x))$
proof –
define B' **where** $B' = (\lambda x. \text{if } x \in A \text{ then } B x \text{ else } \{\})$
from *assms* **have** [*simp*]: *countable* $(B' x)$ **for** x
by (*auto simp: B'_def*)
then interpret *product_sigma_finite_count_space* $\circ B'$
unfolding *o_def* **by** (*intro product_sigma_finite.intro sigma_finite_measure_count_space_countable*)
have $\text{infsetsum } (\lambda g. \prod_{x \in A}. f x (g x)) (PiE A B) =$
 $(\int g. (\prod_{x \in A}. f x (g x)) \partial \text{count_space } (PiE A B))$
by (*simp add: infsetsum_def*)
also have $PiE A B = PiE A B'$
by (*intro PiE_cong*) (*simp_all add: B'_def*)
hence $\text{count_space } (PiE A B) = \text{count_space } (PiE A B')$
by *simp*
also have $\dots = PiM A (\text{count_space} \circ B')$
unfolding *o_def* **using** *finite* **by** (*intro count_space_PiM_finite* [*symmetric*])
simp_all
also have $(\int g. (\prod_{x \in A}. f x (g x)) \partial \dots) = (\prod_{x \in A}. \text{infsetsum } (f x) (B' x))$
by (*subst product_integral_prod*)
 $(\text{insert summable_finite, simp_all add: infsetsum_def } B'_\text{def abs_summable_on_def})$
also have $\dots = (\prod_{x \in A}. \text{infsetsum } (f x) (B x))$
by (*intro prod.cong refl*) (*simp_all add: B'_def*)
finally show *?thesis* .
qed

lemma *infsetsum_uminus*: $\text{infsetsum } (\lambda x. -f x) A = -\text{infsetsum } f A$
unfolding *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_minus*)

lemma *infsetsum_add*:
assumes f *abs_summable_on* A **and** g *abs_summable_on* A
shows $\text{infsetsum } (\lambda x. f x + g x) A = \text{infsetsum } f A + \text{infsetsum } g A$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_add*)

lemma *infsetsum_diff*:

assumes *f abs_summable_on A* **and** *g abs_summable_on A*
shows $\text{infsetsum } (\lambda x. f x - g x) A = \text{infsetsum } f A - \text{infsetsum } g A$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_diff*)

lemma *infsetsum_scaleR_left*:

assumes $c \neq 0 \implies f \text{ abs_summable_on } A$
shows $\text{infsetsum } (\lambda x. f x *_{\mathbb{R}} c) A = \text{infsetsum } f A *_{\mathbb{R}} c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_scaleR_left*)

lemma *infsetsum_scaleR_right*:

$\text{infsetsum } (\lambda x. c *_{\mathbb{R}} f x) A = c *_{\mathbb{R}} \text{infsetsum } f A$
unfolding *infsetsum_def abs_summable_on_def*
by (*subst Bochner_Integration.integral_scaleR_right*) *auto*

lemma *infsetsum_cmult_left*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real_normed_algebra, second_countable_topology}\}$
assumes $c \neq 0 \implies f \text{ abs_summable_on } A$
shows $\text{infsetsum } (\lambda x. f x * c) A = \text{infsetsum } f A * c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_mult_left*)

lemma *infsetsum_cmult_right*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real_normed_algebra, second_countable_topology}\}$
assumes $c \neq 0 \implies f \text{ abs_summable_on } A$
shows $\text{infsetsum } (\lambda x. c * f x) A = c * \text{infsetsum } f A$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_mult_right*)

lemma *infsetsum_cdiv*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, real_normed_field, second_countable_topology}\}$
assumes $c \neq 0 \implies f \text{ abs_summable_on } A$
shows $\text{infsetsum } (\lambda x. f x / c) A = \text{infsetsum } f A / c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def* **by** *auto*

lemma

fixes $f :: 'a \Rightarrow 'c :: \{\text{banach, real_normed_field, second_countable_topology}\}$
assumes [*simp*]: *countable A* **and** [*simp*]: *countable B*
assumes *f abs_summable_on A* **and** *g abs_summable_on B*
shows *abs_summable_on_product*: $(\lambda(x,y). f x * g y) \text{ abs_summable_on } A \times B$
and *infsetsum_product*: $\text{infsetsum } (\lambda(x,y). f x * g y) (A \times B) = \text{infsetsum } f A * \text{infsetsum } g B$

proof –

```

from assms show  $(\lambda(x,y). f x * g y)$  abs_summable_on  $A \times B$ 
by (subst abs_summable_on_Sigma_iff)
      (auto intro!: abs_summable_on_cmult_right simp: norm_mult infsetsum_cmult_right)
with assms show infsetsum  $(\lambda(x,y). f x * g y)$   $(A \times B) = \text{infsetsum } f A * \text{infsetsum } g B$ 
by (subst infsetsum_Sigma)
      (auto simp: infsetsum_cmult_left infsetsum_cmult_right)
qed

```

lemma *abs_summable_finite_sumsI*:

```

assumes  $\bigwedge F. \text{finite } F \implies F \subseteq S \implies \text{sum } (\lambda x. \text{norm } (f x)) F \leq B$ 
shows f abs_summable_on  $S$ 

```

proof –

```

have main:  $f \text{ abs_summable_on } S \wedge \text{infsetsum } (\lambda x. \text{norm } (f x)) S \leq B$  if  $\langle B \geq 0 \rangle$  and  $\langle S \neq \{\} \rangle$ 

```

proof –

```

define  $M \text{ normf}$  where  $M = \text{count\_space } S$  and  $\text{normf } x = \text{ennreal } (\text{norm } (f x))$  for  $x$ 

```

```

have  $\text{sum normf } F \leq \text{ennreal } B$ 

```

```

if finite  $F$  and  $F \subseteq S$  and

```

```

 $\bigwedge F. \text{finite } F \implies F \subseteq S \implies (\sum_{i \in F}. \text{ennreal } (\text{norm } (f i))) \leq \text{ennreal } B$ 

```

and

```

 $\text{ennreal } 0 \leq \text{ennreal } B$  for  $F$ 

```

```

using that unfolding normf_def[symmetric] by simp

```

```

hence  $\text{normf}_B$ : finite  $F \implies F \subseteq S \implies \text{sum normf } F \leq \text{ennreal } B$  for  $F$ 

```

```

using assms[THEN ennreal_leI]

```

```

by auto

```

```

have  $\text{integral}^S M g \leq B$  if simple_function  $M g$  and  $g \leq \text{normf}$  for  $g$ 

```

proof –

```

define  $gS$  where  $gS = g \text{ ` } S$ 

```

```

have finite  $gS$ 

```

```

using that unfolding gS_def M_def simple_function_count_space by simp

```

```

have  $gS \neq \{\}$  unfolding  $gS\_def$  using  $\langle S \neq \{\} \rangle$  by auto

```

```

define part where  $\text{part } r = g \text{ ` } \{r\} \cap S$  for  $r$ 

```

```

have  $r\_finite$ :  $r < \infty$  if  $r : gS$  for  $r$ 

```

```

using  $\langle g \leq \text{normf} \rangle$  that unfolding gS_def le_fun_def normf_def apply

```

auto

```

using ennreal_less_top neq_top_trans top.not_eq_extremum by blast

```

```

define  $B' r$  where  $B' r = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq \text{part } r\}. \text{sum normf } F)$ 

```

for r

```

have  $B'_{fin}$ :  $B' r < \infty$  for  $r$ 

```

proof –

```

have  $B' r \leq (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq \text{part } r\}. \text{sum normf } F)$ 

```

```

unfolding  $B'\_def$ 

```

```

by (metis (mono_tags, lifting) SUP_least SUP_upper)

```

```

also have  $\dots \leq B$ 

```

```

using  $\text{normf}_B$  unfolding part_def

```

```

by (metis (no_types, lifting) Int_subset_iff SUP_least mem_Collect_eq)

```

```

also have  $\dots < \infty$ 

```

```

    by simp
    finally show ?thesis by simp
qed
have sumB': sum B' gS ≤ ennreal B + ε if ε > 0 for ε
proof -
  define N εN where N = card gS and εN = ε / N
  have N > 0
    unfolding N_def using ⟨gS ≠ {}⟩ ⟨finite gS⟩
    by (simp add: card_gt_0_iff)
  from εN_def that have εN > 0
    by (simp add: ennreal_of_nat_eq_real_of_nat ennreal_zero_less_divide)
  have c1: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r
    if B' r = 0 for r
    using that by auto
  have c2: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r if B' r ≠
0 for r
proof-
  have B' r - εN < B' r
    using B'fin ⟨0 < εN⟩ ennreal_between that by fastforce
  have B' r - εN < Sup (sum normf ' {F. finite F ∧ F ⊆ part r} ) ⇒
    ∃ F. B' r - εN ≤ sum normf F ∧ finite F ∧ F ⊆ part r
  by (metis (no_types, lifting) leD le_cases less_SUP_iff mem_Collect_eq)
  hence B' r - εN < B' r ⇒ ∃ F. B' r - εN ≤ sum normf F ∧ finite F
∧ F ⊆ part r
    by (subst (asm) (2) B'_def)
  then obtain F where B' r - εN ≤ sum normf F and finite F and F
⊆ part r
    using ⟨B' r - εN < B' r⟩ by auto
  thus ∃ F. B' r ≤ sum normf F + εN ∧ finite F ∧ F ⊆ part r
    by (metis add.commute ennreal_minus_le_iff)
qed
have ∀ x. ∃ y. B' x ≤ sum normf y + εN ∧
  finite y ∧ y ⊆ part x
  using c1 c2
  by blast
hence ∃ F. ∀ x. B' x ≤ sum normf (F x) + εN ∧ finite (F x) ∧ F x ⊆ part
x
  by metis
then obtain F where F: sum normf (F r) + εN ≥ B' r and Ffin: finite
(F r) and Fpartr: F r ⊆ part r for r
  using atomize_elim by auto
have w1: finite gS
  by (simp add: ⟨finite gS⟩)
have w2: ∀ i ∈ gS. finite (F i)
  by (simp add: Ffin)
have False
  if ∧ r. F r ⊆ g - ' {r} ∧ F r ⊆ S
  and i ∈ gS and j ∈ gS and i ≠ j and x ∈ F i and x ∈ F j
  for i j x

```

```

    by (metis subsetD that(1) that(4) that(5) that(6) vimage_singleton_eq)

  hence w3:  $\forall i \in gS. \forall j \in gS. i \neq j \longrightarrow F i \cap F j = \{\}$ 
    using Fpartr[unfolded part_def] by auto
  have w4:  $sum\ normf (\bigcup (F ` gS)) + \varepsilon = sum\ normf (\bigcup (F ` gS)) + \varepsilon$ 
    by simp
  have  $sum\ B' gS \leq (\sum r \in gS. sum\ normf (F r) + \varepsilon N)$ 
    using F by (simp add: sum_mono)
  also have  $\dots = (\sum r \in gS. sum\ normf (F r)) + (\sum r \in gS. \varepsilon N)$ 
    by (simp add: sum.distrib)
  also have  $\dots = (\sum r \in gS. sum\ normf (F r)) + (card\ gS * \varepsilon N)$ 
    by auto
  also have  $\dots = (\sum r \in gS. sum\ normf (F r)) + \varepsilon$ 
    unfolding  $\varepsilon N\_def\ N\_def[symmetric]$  using  $\langle N > 0 \rangle$ 
  by (simp add: ennreal_times_divide mult.commute mult_divide_eq_ennreal)
  also have  $\dots = sum\ normf (\bigcup (F ` gS)) + \varepsilon$ 
    using w1 w2 w3 w4
    by (subst sum.UNION_disjoint[symmetric])
  also have  $\dots \leq B + \varepsilon$ 
    using  $\langle finite\ gS \rangle\ normf\_B\ add\_right\_mono\ Ffin\ Fpartr$  unfolding
part_def
    by (simp add:  $\langle gS \neq \{\} \rangle\ cSUP\_least$ )
  finally show ?thesis
    by auto
qed
  hence  $sum\ B': sum\ B' gS \leq B$ 
    using  $ennreal\_le\_epsilon\ ennreal\_less\_zero\_iff$  by blast
  have  $\forall r. \exists y. r \in gS \longrightarrow B' r = ennreal\ y$ 
    using  $B'fin\ less\_top\_ennreal$  by auto
  hence  $\exists B''. \forall r. r \in gS \longrightarrow B' r = ennreal (B'' r)$ 
    by (rule_tac choice)
  then obtain  $B''$  where  $B'': B' r = ennreal (B'' r)$  if  $r \in gS$  for  $r$ 
    by atomize_elim
  have  $cases[case\_names\ zero\ finite\ infinite]: P$  if  $r=0 \implies P$  and  $finite (part\ r) \implies P$ 
    and  $infinite (part\ r) \implies r \neq 0 \implies P$  for  $P\ r$ 
    using that by metis
  have  $emeasure\_B': r * emeasure\ M (part\ r) \leq B' r$  if  $r : gS$  for  $r$ 
  proof (cases rule:cases[of r])
  case zero
  thus ?thesis by simp
  next
  case finite
  have  $s1: sum\ g\ F \leq sum\ normf\ F$ 
    if  $F \in \{F. finite\ F \wedge F \subseteq part\ r\}$ 
    for  $F$ 
    using  $\langle g \leq normf \rangle$ 
    by (simp add: le_fun_def sum_mono)

```

```

have r * of_nat (card (part r)) = r * (∑ x∈part r. 1) by simp
also have ... = (∑ x∈part r. r)
  using mult.commute by auto
also have ... = (∑ x∈part r. g x)
  unfolding part_def by auto
also have ... ≤ (SUP F∈{F. finite F ∧ F⊆part r}. sum g F)
  using finite
  by (simp add: Sup_upper)
also have ... ≤ B' r
  unfolding B'_def
  using s1 SUP_subset_mono by blast
finally have r * of_nat (card (part r)) ≤ B' r by assumption
thus ?thesis
  unfolding M_def
  using part_def finite by auto
next
case infinite
from r_finite[OF ⟨r : gS⟩] obtain r' where r': r = ennreal r'
  using ennreal_cases by auto
with infinite have r' > 0
  using ennreal_less_zero_iff_not_gr_zero by blast
obtain N::nat where N:N > B / r' and real N > 0 apply atomize_elim
  using reals_Archimedean2
  by (metis less_trans linorder_neqE_linordered_idom)
obtain F where finite F and card F = N and F ⊆ part r
  using infinite(1) infinite_arbitrarily_large by blast
from ⟨F ⊆ part r⟩ have F ⊆ S unfolding part_def by simp
have B < r * N
  unfolding r' ennreal_of_nat_eq_real_of_nat
  using N ⟨0 < r'⟩ ⟨B ≥ 0⟩ r'
  by (metis enn2real_ennreal enn2real_less_iff ennreal_less_top ennreal_mult' less_le mult_less_cancel_left pos nonzero_mult_div_cancel_left times_divide_eq_right)
also have r * N = (∑ x∈F. r)
  using ⟨card F = N⟩ by (simp add: mult.commute)
also have (∑ x∈F. r) = (∑ x∈F. g x)
  using ⟨F ⊆ part r⟩ part_def sum.cong subsetD by fastforce
also have (∑ x∈F. g x) ≤ (∑ x∈F. ennreal (norm (f x)))
  by (metis (mono_tags, lifting) ⟨g ≤ normf⟩ ⟨normf ≡ λx. ennreal (norm (f x))⟩ le_fun_def
    sum_mono)
also have (∑ x∈F. ennreal (norm (f x))) ≤ B
  using ⟨F ⊆ S⟩ ⟨finite F⟩ ⟨normf ≡ λx. ennreal (norm (f x))⟩ normf_B
by blast
finally have B < B by auto
thus ?thesis by simp
qed

have integralS M g = (∑ r ∈ gS. r * emeasure M (part r))
  unfolding simple_integral_def gS_def M_def part_def by simp

```



```

    also have ... ≤ (∑ r ∈ gS. B' r)
      by (simp add: emeasure_B' sum_mono)
    also have ... ≤ B
      using sumB' by blast
    finally show ?thesis by assumption
  qed
  hence int_leq_B: integralN M normf ≤ B
  unfolding nn_integral_def by (metis (no_types, lifting) SUP_least mem_Collect_eq)
  hence integralN M normf < ∞
    using le_less_trans by fastforce
  hence integrable M f
    unfolding M_def normf_def by (rule integrableI_bounded[rotated], simp)
  hence v1: f abs_summable_on S
    unfolding abs_summable_on_def M_def by simp

  have (λx. norm (f x)) abs_summable_on S
    using v1 Infinite_Set_Sum.abs_summable_on_norm_iff[where A = S and
f = f]
      by auto
  moreover have 0 ≤ norm (f x)
    if x ∈ S for x
      by simp
  moreover have (∫+ x. ennreal (norm (f x)) ∂count_space S) ≤ ennreal B
    using M_def ‹normf ≡ λx. ennreal (norm (f x))› int_leq_B by auto
  ultimately have ennreal (∑ax∈S. norm (f x)) ≤ ennreal B
    by (simp add: nn_integral_conv_infsetsum)
  hence v2: (∑ax∈S. norm (f x)) ≤ B
    by (subst ennreal_le_iff[symmetric], simp add: assms ‹B ≥ 0›)
  show ?thesis
    using v1 v2 by auto
  qed
  then show f abs_summable_on S
    by (metis abs_summable_on_finite assms empty_subsetI finite.emptyI sum_clauses(1))
  qed

```

lemma *infsetsum_nonneg_is_SUPREMUM_ennreal*:

```

  fixes f :: 'a ⇒ real
  assumes summable: f abs_summable_on A
    and fnn: ∧x. x∈A ⇒ f x ≥ 0
  shows ennreal (infsetsum f A) = (SUP F∈{F. finite F ∧ F ⊆ A}. (ennreal (sum
f F)))
  proof -
    have sum_F_A: sum f F ≤ infsetsum f A
      if F ∈ {F. finite F ∧ F ⊆ A}
      for F
    proof -
      from that have finite F and F ⊆ A by auto
      from ‹finite F› have sum f F = infsetsum f F by auto

```

```

also have ... ≤ infsetsum f A
proof (rule infsetsum_mono_neutral_left)
  show f abs_summable_on F
    by (simp add: ⟨finite F⟩)
  show f abs_summable_on A
    by (simp add: local.summable)
  show f x ≤ f x
    if x ∈ F
    for x :: 'a
    by simp
  show F ⊆ A
    by (simp add: ⟨F ⊆ A⟩)
  show 0 ≤ f x
    if x ∈ A - F
    for x :: 'a
    using that fnn by auto
qed
finally show ?thesis by assumption
qed
hence geq: ennreal (infsetsum f A) ≥ (SUP F∈{G. finite G ∧ G ⊆ A}. (ennreal
(sum f F)))
  by (meson SUP_least ennreal_leI)

define fe where fe x = ennreal (f x) for x

have sum_f_int: infsetsum f A = ∫+ x. fe x ∂(count_space A)
  unfolding infsetsum_def fe_def
proof (rule nn_integral_eq_integral [symmetric])
  show integrable (count_space A) f
    using abs_summable_on_def local.summable by blast
  show AE x in count_space A. 0 ≤ f x
    using fnn by auto
qed
also have ... = (SUP g ∈ {g. finite (g'A) ∧ g ≤ fe}. integralS (count_space A)
g)
  unfolding nn_integral_def simple_function_count_space by simp
also have ... ≤ (SUP F∈{F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
proof (rule Sup_least)
  fix x assume x ∈ integralS (count_space A) ‘ {g. finite (g'A) ∧ g ≤ fe}
  then obtain g where xg: x = integralS (count_space A) g and fin_gA: finite
(g'A)
    and g_fe: g ≤ fe by auto
  define F where F = {z:A. g z ≠ 0}
  hence F ⊆ A by simp

  have fin: finite {z:A. g z = t} if t ≠ 0 for t
  proof (rule ccontr)
    assume inf: infinite {z:A. g z = t}
    hence tgA: t ∈ g'A

```

```

    by (metis (mono_tags, lifting) image_eqI not_finite_existsD)
  have  $x = (\sum x \in g \text{ ` } A. x * \text{emeasure } (\text{count\_space } A) (g \text{ - ` } \{x\} \cap A))$ 
    unfolding  $xg \text{ simple\_integral\_def } \text{space\_count\_space}$  by simp
  also have  $\dots \geq (\sum x \in \{t\}. x * \text{emeasure } (\text{count\_space } A) (g \text{ - ` } \{x\} \cap A))$ 
(is  $\_ \geq \dots$ )
  proof (rule sum_mono2)
    show finite  $(g \text{ ` } A)$ 
      by (simp add: fin_gA)
    show  $\{t\} \subseteq g \text{ ` } A$ 
      by (simp add: tgA)
    show  $0 \leq b * \text{emeasure } (\text{count\_space } A) (g \text{ - ` } \{b\} \cap A)$ 
      if  $b \in g \text{ ` } A - \{t\}$ 
      for  $b :: \text{ennreal}$ 
      using that
      by simp
  qed
  also have  $\dots = t * \text{emeasure } (\text{count\_space } A) (g \text{ - ` } \{t\} \cap A)$ 
    by auto
  also have  $\dots = t * \infty$ 
  proof (subst  $\text{emeasure\_count\_space\_infinite}$ )
    show  $g \text{ - ` } \{t\} \cap A \subseteq A$ 
      by simp
    have  $\{a \in A. g \ a = t\} = \{a \in g \text{ - ` } \{t\}. a \in A\}$ 
      by auto
    thus infinite  $(g \text{ - ` } \{t\} \cap A)$ 
      by (metis (full_types) Int_def inf)
    show  $t * \infty = t * \infty$ 
      by simp
  qed
  also have  $\dots = \infty$  using  $\langle t \neq 0 \rangle$ 
    by (simp add:  $\text{ennreal\_mult\_eq\_top\_iff}$ )
  finally have  $x\_inf: x = \infty$ 
    using  $\text{neq\_top\_trans}$  by auto
  have  $x = \text{integral}^S (\text{count\_space } A) g$  by (fact  $xg$ )
  also have  $\dots = \text{integral}^N (\text{count\_space } A) g$ 
    by (simp add:  $\text{fin\_gA } \text{nn\_integral\_eq\_simple\_integral}$ )
  also have  $\dots \leq \text{integral}^N (\text{count\_space } A) fe$ 
    using  $g\_fe$ 
    by (simp add:  $\text{le\_funD } \text{nn\_integral\_mono}$ )
  also have  $\dots < \infty$ 
    by (metis  $\text{sum\_f\_int } \text{ennreal\_less\_top } \text{infinity\_ennreal\_def}$ )
  finally have  $x\_fin: x < \infty$  by simp
  from  $x\_inf \ x\_fin$  show False by simp
qed
have  $F: F = (\bigcup t \in g \text{ ` } A - \{0\}. \{z \in A. g \ z = t\})$ 
  unfolding  $F\_def$  by auto
hence finite  $F$ 
  unfolding  $F$  using  $\text{fin\_gA } \text{fin}$  by auto
have  $x = \text{integral}^N (\text{count\_space } A) g$ 

```

```

unfolding xg
  by (simp add: fin_gA nn_integral_eq_simple_integral)
also have ... = set_nn_integral (count_space UNIV) A g
  by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)
also have ... = set_nn_integral (count_space UNIV) F g
proof -
  have  $\forall a. g a * (if a \in \{a \in A. g a \neq 0\} then 1 else 0) = g a * (if a \in A$ 
then 1 else 0)
  by auto
  hence  $(\int^+ a. g a * (if a \in A then 1 else 0) \partial count\_space UNIV)$ 
    =  $(\int^+ a. g a * (if a \in \{a \in A. g a \neq 0\} then 1 else 0) \partial count\_space$ 
UNIV)
  by presburger
  thus ?thesis unfolding F_def indicator_def
  using mult.right_neutral mult_zero_right nn_integral_cong
  by (simp add: of_bool_def)
qed
also have ... = integralN (count_space F) g
  by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)
also have ... = sum g F
  using  $\langle finite F \rangle$  by (rule nn_integral_count_space_finite)
also have sum g F ≤ sum fe F
  using g_fe unfolding le_fun_def
  by (simp add: sum_mono)
also have ... ≤  $(SUP F \in \{G. finite G \wedge G \subseteq A\}. (sum fe F))$ 
  using  $\langle finite F \rangle \langle F \subseteq A \rangle$ 
  by (simp add: SUP_upper)
also have ... =  $(SUP F \in \{F. finite F \wedge F \subseteq A\}. (ennreal (sum f F)))$ 
proof (rule SUP_cong [OF refl])
  have finite x  $\implies x \subseteq A \implies (\sum x \in x. ennreal (f x)) = ennreal (sum f x)$ 
  for x
  by (metis fnn_subsetCE sum_ennreal)
  thus sum fe x = ennreal (sum f x)
  if  $x \in \{G. finite G \wedge G \subseteq A\}$ 
  for  $x :: 'a set$ 
  using that unfolding fe_def by auto
qed
finally show  $x \leq \dots$  by simp
qed
finally have leq: ennreal (infsetsum f A) ≤ (SUP F ∈ {F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
  by assumption
from leq geq show ?thesis by simp
qed

lemma infsetsum_nonneg_is_SUPREMUM_ereal:
  fixes  $f :: 'a \Rightarrow real$ 
  assumes summable: f abs_summable_on A
  and  $fnn: \bigwedge x. x \in A \implies f x \geq 0$ 

```

shows $\text{ereal} (\text{infsetsum } f A) = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal} (\text{sum } f F)))$
proof –
have $\text{ereal} (\text{infsetsum } f A) = \text{enn2ereal} (\text{ennreal} (\text{infsetsum } f A))$
by (*simp add: fnn infsetsum_nonneg*)
also have $\dots = \text{enn2ereal} (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. \text{ennreal} (\text{sum } f F))$
apply (*subst infsetsum_nonneg_is_SUPREMUM_ennreal*)
using fnn by (*auto simp add: local.summable*)
also have $\dots = (\text{SUP } F \in \{F. \text{finite } F \wedge F \subseteq A\}. (\text{ereal} (\text{sum } f F)))$
proof (*simp add: image_def Sup_ennreal.rep_eq*)
have $0 \leq \text{Sup} \{y. \exists x. (\exists xa. \text{finite } xa \wedge xa \subseteq A \wedge x = \text{ennreal} (\text{sum } f xa)) \wedge y = \text{enn2ereal } x\}$
by (*metis (mono_tags, lifting) Sup_upper empty_subsetI ennreal_0 finite.emptyI mem_Collect_eq sum.empty zero_ennreal.rep_eq*)
moreover have $(\exists x. (\exists y. \text{finite } y \wedge y \subseteq A \wedge x = \text{ennreal} (\text{sum } f y)) \wedge y = \text{enn2ereal } x) =$
 $(\exists x. \text{finite } x \wedge x \subseteq A \wedge y = \text{ereal} (\text{sum } f x))$ **for** y
proof –
have $(\exists x. (\exists y. \text{finite } y \wedge y \subseteq A \wedge x = \text{ennreal} (\text{sum } f y)) \wedge y = \text{enn2ereal } x) \longleftrightarrow$
 $(\exists X x. \text{finite } X \wedge X \subseteq A \wedge x = \text{ennreal} (\text{sum } f X) \wedge y = \text{enn2ereal } x)$
by blast
also have $\dots \longleftrightarrow (\exists X. \text{finite } X \wedge X \subseteq A \wedge y = \text{ereal} (\text{sum } f X))$
by (*rule arg_cong[of _ _ Ex]*)
(auto simp: fun_eq_iff intro!: enn2ereal_ennreal sum_nonneg enn2ereal_ennreal[symmetric] fnn)
finally show *?thesis* .
qed
hence $\text{Sup} \{y. \exists x. (\exists y. \text{finite } y \wedge y \subseteq A \wedge x = \text{ennreal} (\text{sum } f y)) \wedge y = \text{enn2ereal } x\} =$
 $\text{Sup} \{y. \exists x. \text{finite } x \wedge x \subseteq A \wedge y = \text{ereal} (\text{sum } f x)\}$
by simp
ultimately show $\text{max } 0 (\text{Sup} \{y. \exists x. (\exists xa. \text{finite } xa \wedge xa \subseteq A \wedge x = \text{ennreal} (\text{sum } f xa)) \wedge y = \text{enn2ereal } x\})$
 $= \text{Sup} \{y. \exists x. \text{finite } x \wedge x \subseteq A \wedge y = \text{ereal} (\text{sum } f x)\}$
by linarith
qed
finally show *?thesis*
by simp
qed

The following theorem relates (*abs_summable_on*) with *Infinite_Sum.abs_summable_on*. Note that while this theorem expresses an equivalence, the notion on the lhs is more general nonetheless because it applies to a wider range of types. (The rhs requires second-countable Banach spaces while the lhs is well-defined on arbitrary real vector spaces.)

lemma *abs_summable_equivalent*: $\langle \text{Infinite_Sum.abs_summable_on } f A \longleftrightarrow f \text{ abs_summable_on } A \rangle$

proof (*rule iffI*)
define n **where** $\langle n\ x = \text{norm}\ (f\ x)\rangle$ **for** x
assume $\langle n\ \text{summable_on}\ A\rangle$
then have $\langle \text{sum}\ n\ F \leq \text{infsum}\ n\ A\rangle$ **if** $\langle \text{finite}\ F\rangle$ **and** $\langle F \subseteq A\rangle$ **for** F
using that by (*auto simp flip: infsum_finite simp: n_def[abs_def] intro!: infsum_mono_neutral*)

then show $\langle f\ \text{abs_summable_on}\ A\rangle$
by (*auto intro!: abs_summable_finite_sumsI simp: n_def*)
next
define n **where** $\langle n\ x = \text{norm}\ (f\ x)\rangle$ **for** x
assume $\langle f\ \text{abs_summable_on}\ A\rangle$
then have $\langle n\ \text{abs_summable_on}\ A\rangle$
by (*simp add: $\langle f\ \text{abs_summable_on}\ A\rangle\ n_def$*)
then have $\langle \text{sum}\ n\ F \leq \text{infsetsum}\ n\ A\rangle$ **if** $\langle \text{finite}\ F\rangle$ **and** $\langle F \subseteq A\rangle$ **for** F
using that by (*auto simp flip: infsetsum_finite simp: n_def[abs_def] intro!: infsetsum_mono_neutral*)
then show $\langle n\ \text{summable_on}\ A\rangle$
apply (*rule_tac nonneg_bdd_above_summable_on*)
by (*auto simp add: n_def bdd_above_def*)
qed

lemma *infsetsum_infsum*:
assumes $f\ \text{abs_summable_on}\ A$
shows $\text{infsetsum}\ f\ A = \text{infsum}\ f\ A$
proof –
have *conv_sum_norm[simp]*: $(\lambda x. \text{norm}\ (f\ x))\ \text{summable_on}\ A$
using *abs_summable_equivalent assms by blast*
have $\text{norm}\ (\text{infsetsum}\ f\ A - \text{infsum}\ f\ A) \leq \varepsilon$ **if** $\varepsilon > 0$ **for** ε
proof –
define δ **where** $\delta = \varepsilon/2$
with that have [*simp*]: $\delta > 0$ **by** *simp*
obtain $F1$ **where** $F1A: F1 \subseteq A$ **and** *finF1*: *finite* $F1$ **and** *leq_eps*: $\text{infsetsum}\ (\lambda x. \text{norm}\ (f\ x))\ (A - F1) \leq \delta$
proof –
have *sum_SUP*: $\text{ereal}\ (\text{infsetsum}\ (\lambda x. \text{norm}\ (f\ x))\ A) = (\text{SUP}\ F \in \{F. \text{finite}\ F \wedge F \subseteq A\}. \text{ereal}\ (\text{sum}\ (\lambda x. \text{norm}\ (f\ x))\ F))$
(is $_ = ?\text{SUP}$ **)**
apply (*rule infsetsum_nonneg_is_SUPREMUM_ereal*)
using *assms by auto*

have $(\text{SUP}\ F \in \{F. \text{finite}\ F \wedge F \subseteq A\}. \text{ereal}\ (\sum_{x \in F}. \text{norm}\ (f\ x))) - \text{ereal}\ \delta$
 $< (\text{SUP}\ i \in \{F. \text{finite}\ F \wedge F \subseteq A\}. \text{ereal}\ (\sum_{x \in i}. \text{norm}\ (f\ x)))$
using $\langle \delta > 0 \rangle$
by (*metis diff_strict_left_mono diff_zero ereal_less_eq(3) ereal_minus(1) not_le sum_SUP*)
then obtain F **where** $F \in \{F. \text{finite}\ F \wedge F \subseteq A\}$ **and** $\text{ereal}\ (\text{sum}\ (\lambda x. \text{norm}\ (f\ x))\ F) > ?\text{SUP} - \text{ereal}\ (\delta)$
by (*meson less_SUP_iff*)

```

hence  $\text{sum } (\lambda x. \text{norm } (f x)) F > \text{infsetsum } (\lambda x. \text{norm } (f x)) A - (\delta)$ 
  unfolding  $\text{sum\_SUP[symmetric]}$  by auto
hence  $\delta > \text{infsetsum } (\lambda x. \text{norm } (f x)) (A-F)$ 
proof (subst infsetsum_Diff)
  show  $(\lambda x. \text{norm } (f x)) \text{abs\_summable\_on } A$ 
    if  $(\sum_{a \in A}. \text{norm } (f x)) - \delta < (\sum_{x \in F}. \text{norm } (f x))$ 
      using that
      by (simp add: assms)
    show  $F \subseteq A$ 
      if  $(\sum_{a \in A}. \text{norm } (f x)) - \delta < (\sum_{x \in F}. \text{norm } (f x))$ 
        using that  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle$  by blast
      show  $(\sum_{a \in A}. \text{norm } (f x)) - (\sum_{a \in F}. \text{norm } (f x)) < \delta$ 
        if  $(\sum_{a \in A}. \text{norm } (f x)) - \delta < (\sum_{x \in F}. \text{norm } (f x))$ 
          using that  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle$  by auto
    qed
  thus ?thesis using that
    apply atomize_elim
    using  $\langle F \in \{F. \text{finite } F \wedge F \subseteq A\} \rangle$  less_imp_le by blast
qed
obtain  $F2$  where  $F2A: F2 \subseteq A$  and  $\text{fin}F2: \text{finite } F2$ 
  and  $\text{dist}: \text{dist } (\text{sum } (\lambda x. \text{norm } (f x)) F2) (\text{infsum } (\lambda x. \text{norm } (f x)) A) \leq \delta$ 
  apply atomize_elim
  by (metis  $\langle 0 < \delta \rangle$  conv_sum_norm infsum_finite_approximation)
have  $\text{leq\_eps}': \text{infsum } (\lambda x. \text{norm } (f x)) (A-F2) \leq \delta$ 
  apply (subst infsum_Diff)
  using  $\text{fin}F2 F2A \text{dist}$  by (auto simp: dist_norm)
define  $F$  where  $F = F1 \cup F2$ 
have  $FA: F \subseteq A$  and  $\text{fin}F: \text{finite } F$ 
  unfolding  $F\_def$  using  $F1A F2A \text{fin}F1 \text{fin}F2$  by auto

have  $(\sum_{a \in A - (F1 \cup F2)}. \text{norm } (f x)) \leq (\sum_{a \in A - F1}. \text{norm } (f x))$ 
  apply (rule infsetsum_mono_neutral_left)
  using abs_summable_on_subset assms by fastforce+
hence  $\text{leq\_eps}: \text{infsetsum } (\lambda x. \text{norm } (f x)) (A-F) \leq \delta$ 
  unfolding  $F\_def$ 
  using  $\text{leq\_eps}$  by linarith
have  $\text{infsum } (\lambda x. \text{norm } (f x)) (A - (F1 \cup F2))$ 
   $\leq \text{infsum } (\lambda x. \text{norm } (f x)) (A - F2)$ 
  apply (rule infsum_mono_neutral)
  using  $\text{fin}F$  by (auto simp add: finF2 summable_on_cofin_subset F_def)
hence  $\text{leq\_eps}': \text{infsum } (\lambda x. \text{norm } (f x)) (A-F) \leq \delta$ 
  unfolding  $F\_def$ 
  by (rule order.trans[OF leq_eps'])
have  $\text{norm } (\text{infsetsum } f A - \text{infsetsum } f F) = \text{norm } (\text{infsetsum } f (A-F))$ 
  apply (subst infsetsum_Diff [symmetric])
  by (auto simp: FA assms)
also have  $\dots \leq \text{infsetsum } (\lambda x. \text{norm } (f x)) (A-F)$ 
  using  $\text{norm\_infsetsum\_bound}$  by blast

```

3420

```
also have ... ≤ δ
  using leq_eps by simp
finally have diff1: norm (infsetsum f A - infsetsum f F) ≤ δ
  by assumption
have norm (infsum f A - infsum f F) = norm (infsum f (A-F))
  apply (subst infsum_Diff [symmetric])
  by (auto simp: asms abs_summable_summable finF FA)
also have ... ≤ infsum (λx. norm (f x)) (A-F)
  by (simp add: finF summable_on_cofin_subset norm_infsum_bound)
also have ... ≤ δ
  using leq_eps' by simp
finally have diff2: norm (infsum f A - infsum f F) ≤ δ
  by assumption

have x1: infsetsum f F = infsum f F
  using finF by simp
have norm (infsetsum f A - infsum f A) ≤ norm (infsetsum f A - infsetsum
f F) + norm (infsum f A - infsum f F)
  apply (rule_tac norm_diff_triangle_le)
  apply auto
  by (simp_all add: x1 norm_minus_commute)
also have ... ≤ ε
  using diff1 diff2 δ_def by linarith
finally show ?thesis
  by assumption
qed
hence norm (infsetsum f A - infsum f A) = 0
  by (meson antisym_conv1 dense_ge norm_not_less_zero)
thus ?thesis
  by auto
qed
end
```

9.24 Faces, Extreme Points, Polytopes, Polyhedra etc

Ported from HOL Light by L C Paulson

```
theory Polytope
imports Cartesian_Euclidean_Space Path_Connected
begin
```

9.24.1 Faces of a (usually convex) set

```
definition face_of :: ['a::real_vector set, 'a set] ⇒ bool (infixr (face'_of) 50)
where
  T face_of S ←→
    T ⊆ S ∧ convex T ∧
```


$$(\forall a \in S. \forall b \in S. \forall x \in T. x \in \text{open_segment } a \ b \longrightarrow a \in T \wedge b \in T)$$

lemma *face_ofD*: $\llbracket T \text{ face_of } S; x \in \text{open_segment } a \ b; a \in S; b \in S; x \in T \rrbracket \Longrightarrow a \in T \wedge b \in T$
unfolding *face_of_def* **by** *blast*

lemma *face_of_translation_eq* [*simp*]:
 $((+) \ a \ ' \ T \ \text{face_of} \ (+) \ a \ ' \ S) \longleftrightarrow T \ \text{face_of} \ S$
proof –
have *: $\bigwedge a \ T \ S. T \ \text{face_of} \ S \Longrightarrow ((+) \ a \ ' \ T \ \text{face_of} \ (+) \ a \ ' \ S)$
by (*simp add: face_of_def*)
show ?thesis
by (*force simp: image_comp o_def dest: * [where a = -a] intro: **)
qed

lemma *face_of_linear_image*:
assumes *linear f inj f*
shows $(f \ ' \ c \ \text{face_of} \ f \ ' \ S) \longleftrightarrow c \ \text{face_of} \ S$
by (*simp add: face_of_def inj_image_subset_iff inj_image_mem_iff open_segment_linear_image assms*)

lemma *faces_of_linear_image*:
 $\llbracket \text{linear } f; \text{inj } f \rrbracket \Longrightarrow \{T. T \ \text{face_of} \ (f \ ' \ S)\} = (\text{image } f) \ ' \ \{T. T \ \text{face_of} \ S\}$
by (*smt (verit) Collect_cong face_of_def face_of_linear_image setcompr_eq_image subset_imageE*)

lemma *face_of_refl*: $\text{convex } S \Longrightarrow S \ \text{face_of} \ S$
by (*auto simp: face_of_def*)

lemma *face_of_refl_eq*: $S \ \text{face_of} \ S \longleftrightarrow \text{convex } S$
by (*auto simp: face_of_def*)

lemma *empty_face_of* [*iff*]: $\{\} \ \text{face_of} \ S$
by (*simp add: face_of_def*)

lemma *face_of_empty* [*simp*]: $S \ \text{face_of} \ \{\} \longleftrightarrow S = \{\}$
by (*meson empty_face_of face_of_def subset_empty*)

lemma *face_of_trans* [*trans*]: $\llbracket S \ \text{face_of} \ T; T \ \text{face_of} \ u \rrbracket \Longrightarrow S \ \text{face_of} \ u$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_face*: $T \ \text{face_of} \ S \Longrightarrow (f \ \text{face_of} \ T \longleftrightarrow f \ \text{face_of} \ S \wedge f \subseteq T)$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_subset*: $\llbracket F \ \text{face_of} \ S; F \subseteq T; T \subseteq S \rrbracket \Longrightarrow F \ \text{face_of} \ T$
unfolding *face_of_def* **by** (*safe; blast*)

lemma *face_of_slice*: $\llbracket F \ \text{face_of} \ S; \text{convex } T \rrbracket \Longrightarrow (F \cap T) \ \text{face_of} \ (S \cap T)$
unfolding *face_of_def* **by** (*blast intro: convex_Int*)

```

lemma face_of_Int:  $\llbracket t1 \text{ face\_of } S; t2 \text{ face\_of } S \rrbracket \implies (t1 \cap t2) \text{ face\_of } S$ 
  unfolding face_of_def by (blast intro: convex_Int)

lemma face_of_Inter:  $\llbracket A \neq \{\}; \bigwedge T. T \in A \implies T \text{ face\_of } S \rrbracket \implies (\bigcap A) \text{ face\_of } S$ 
  unfolding face_of_def by (blast intro: convex_Inter)

lemma face_of_Int_Int:  $\llbracket F \text{ face\_of } T; F' \text{ face\_of } t' \rrbracket \implies (F \cap F') \text{ face\_of } (T \cap t')$ 
  unfolding face_of_def by (blast intro: convex_Int)

lemma face_of_imp_subset:  $T \text{ face\_of } S \implies T \subseteq S$ 
  unfolding face_of_def by blast

proposition face_of_imp_eq_affine_Int:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $S$ : convex  $S$  and  $T$ :  $T \text{ face\_of } S$ 
  shows  $T = (\text{affine hull } T) \cap S$ 
proof -
  have convex  $T$  using  $T$  by (simp add: face_of_def)
  have *: False if  $x: x \in \text{affine hull } T$  and  $x \in S$   $x \notin T$  and  $y: y \in \text{rel\_interior } T$ 
for  $x y$ 
  proof -
    obtain  $e$  where  $e > 0$  and  $e$ :  $\text{cball } y \ e \cap \text{affine hull } T \subseteq T$ 
    using  $y$  by (auto simp: rel_interior_cball)
    have  $y \neq x$   $y \in S$   $y \in T$ 
    using face_of_imp_subset rel_interior_subset  $T$  that by blast+
    then have  $znc: \bigwedge u. \llbracket u \in \{0 <..< 1\}; (1 - u) *_R y + u *_R x \in T \rrbracket \implies \text{False}$ 
    using  $\langle x \in S \rangle \langle x \notin T \rangle \langle T \text{ face\_of } S \rangle$  unfolding face_of_def
    by (meson greaterThanLessThan_iff in_segment(2))
    define  $u$  where  $u \equiv \min (1/2) (e / \text{norm } (x - y))$ 
    have  $in01: u \in \{0 <..< 1\}$ 
    using  $\langle y \neq x \rangle \langle e > 0 \rangle$  by (simp add: u_def)
    have  $\text{norm } (u *_R y - u *_R x) \leq e$ 
    using  $\langle e > 0 \rangle$ 
    by (simp add: u_def norm_minus_commute min_mult_distrib_right flip:
scaleR_diff_right)
    then have  $\text{dist } y ((1 - u) *_R y + u *_R x) \leq e$ 
    by (simp add: dist_norm algebra_simps)
    then show False
    using  $znc$   $[OF in01 e [THEN subsetD]]$  by (simp add:  $\langle y \in T \rangle$  hull_inc
mem_affine  $x$ )
  qed
  show ?thesis
proof (rule subset_antisym)
  show  $T \subseteq \text{affine hull } T \cap S$ 
  using  $assms$  by (simp add: hull_subset face_of_imp_subset)
  show  $\text{affine hull } T \cap S \subseteq T$ 

```

```

    using * ⟨convex T⟩ rel_interior_eq_empty by fastforce
  qed
qed

lemma face_of_imp_closed:
  fixes S :: 'a::euclidean_space set
  assumes convex S closed S T face_of S shows closed T
  by (metis affine_affine_hull affine_closed closed_Int face_of_imp_eq_affine_Int
  assms)

lemma face_of_Int_supporting_hyperplane_le_strong:
  assumes convex (S ∩ {x. a · x = b}) and aleb:  $\bigwedge x. x \in S \implies a \cdot x \leq b$ 
  shows (S ∩ {x. a · x = b}) face_of S
proof -
  have *:  $a \cdot u = a \cdot x$  if  $x \in \text{open\_segment } u \ v \ u \in S \ v \in S$  and  $b = a \cdot x$ 
    for  $u \ v \ x$ 
  proof (rule antisym)
    show  $a \cdot u \leq a \cdot x$ 
      using aleb ⟨ $u \in S$ ⟩ ⟨ $b = a \cdot x$ ⟩ by blast
  next
    obtain  $\xi$  where  $b = a \cdot ((1 - \xi) *_{\mathbb{R}} u + \xi *_{\mathbb{R}} v)$   $0 < \xi \ \xi < 1$ 
      using ⟨ $b = a \cdot x$ ⟩ ⟨ $x \in \text{open\_segment } u \ v$ ⟩ in_segment
      by (auto simp: open_segment_image_interval split: if_split_asm)
    then have  $b + \xi * (a \cdot u) \leq a \cdot u + \xi * b$ 
      using aleb [OF ⟨ $v \in S$ ⟩] by (simp add: algebra_simps)
    then have  $(1 - \xi) * b \leq (1 - \xi) * (a \cdot u)$ 
      by (simp add: algebra_simps)
    then have  $b \leq a \cdot u$ 
      using ⟨ $\xi < 1$ ⟩ by auto
  with b show  $a \cdot x \leq a \cdot u$  by simp
  qed
show ?thesis
  using * open_segment_commute by (fastforce simp add: face_of_def assms)
qed

lemma face_of_Int_supporting_hyperplane_ge_strong:
   $\llbracket \text{convex}(S \cap \{x. a \cdot x = b\}); \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket$ 
 $\implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  using face_of_Int_supporting_hyperplane_le_strong [of S -a -b] by simp

lemma face_of_Int_supporting_hyperplane_le:
   $\llbracket \text{convex } S; \bigwedge x. x \in S \implies a \cdot x \leq b \rrbracket \implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  by (simp add: convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_le_strong)

lemma face_of_Int_supporting_hyperplane_ge:
   $\llbracket \text{convex } S; \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket \implies (S \cap \{x. a \cdot x = b\}) \text{ face\_of } S$ 
  by (simp add: convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_ge_strong)

lemma face_of_imp_convex:  $T \text{ face\_of } S \implies \text{convex } T$ 

```

3424

using *face_of_def* by *blast*

lemma *face_of_imp_compact*:

fixes $S :: 'a::\text{euclidean_space set}$

shows $\llbracket \text{convex } S; \text{ compact } S; T \text{ face_of } S \rrbracket \implies \text{compact } T$

by (meson bounded_subset compact_eq_bounded_closed face_of_imp_closed face_of_imp_subset)

lemma *face_of_Int_subface*:

$\llbracket A \cap B \text{ face_of } A; A \cap B \text{ face_of } B; C \text{ face_of } A; D \text{ face_of } B \rrbracket$

$\implies (C \cap D) \text{ face_of } C \wedge (C \cap D) \text{ face_of } D$

by (meson face_of_Int_Int face_of_face inf_le1 inf_le2)

lemma *subset_of_face_of*:

fixes $S :: 'a::\text{real_normed_vector set}$

assumes $T \text{ face_of } S \ u \subseteq S \ T \cap (\text{rel_interior } u) \neq \{\}$

shows $u \subseteq T$

proof

fix c

assume $c \in u$

obtain b where $b \in T \ b \in \text{rel_interior } u$ using *assms* by *auto*

then obtain e where $e > 0 \ b \in u$ and $e: \text{cball } b \ e \cap \text{affine hull } u \subseteq u$

by (*auto simp: rel_interior_cball*)

show $c \in T$

proof (*cases b=c*)

case *True* with $\langle b \in T \rangle$ show *?thesis* by *blast*

next

case *False*

define d where $d = b + (e / \text{norm}(b - c)) *_{\mathbb{R}} (b - c)$

have $d \in \text{cball } b \ e \cap \text{affine hull } u$

using $\langle e > 0 \rangle \langle b \in u \rangle \langle c \in u \rangle$

by (*simp add: d_def dist_norm hull_inc mem_affine_3_minus False*)

with e have $d \in u$ by *blast*

have $\text{norm } (b - c) + e > 0$ using $\langle e > 0 \rangle$

by (*metis add.commute le_less_trans less_add_same_cancel2 norm_ge_zero*)

then have [*simp*]: $d \neq c$ using *False scaleR_cancel_left* [of $1 + (e / \text{norm } (b - c)) \ b \ c$]

by (*simp add: algebra_simps d_def*) (*simp add: field_split_simps*)

have [*simp*]: $((e - e * e / (e + \text{norm } (b - c))) / \text{norm } (b - c)) = (e / (e + \text{norm } (b - c)))$

using *False nbc*

by (*simp add: divide_simps*) (*simp add: algebra_simps*)

have $b \in \text{open_segment } d \ c$

apply (*simp add: open_segment_image_interval*)

apply (*simp add: d_def algebra_simps*)

apply (*rule_tac x=e / (e + norm (b - c)) in image_eqI*)

using *False nbc* $\langle 0 < e \rangle$ by (*auto simp: algebra_simps*)

then have $d \in T \wedge c \in T$

by (meson $\langle b \in T \rangle \langle c \in u \rangle \langle d \in u \rangle$ *assms face_ofD subset_iff*)

then show ?thesis ..
qed
qed

lemma face_of_eq:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $T \text{ face_of } S \ U \text{ face_of } S \ (\text{rel_interior } T) \cap (\text{rel_interior } U) \neq \{\}$
shows $T = U$
using *assms*
unfolding *disjoint_iff_not_equal*
by (*metis IntI empty_iff face_of_imp_subset mem_rel_interior_ball subset_antisym subset_of_face_of*)

lemma face_of_disjoint_rel_interior:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $T \text{ face_of } S \ T \neq S$
shows $T \cap \text{rel_interior } S = \{\}$
by (*meson assms subset_of_face_of face_of_imp_subset order_refl subset_antisym*)

lemma face_of_disjoint_interior:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $T \text{ face_of } S \ T \neq S$
shows $T \cap \text{interior } S = \{\}$
using *assms face_of_disjoint_rel_interior interior_subset_rel_interior* by *fastforce*

lemma face_of_subset_rel_boundary:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $T \text{ face_of } S \ T \neq S$
shows $T \subseteq (S - \text{rel_interior } S)$
by (*meson DiffI assms disjoint_iff_not_equal face_of_disjoint_rel_interior face_of_imp_subset subset_iff*)

lemma face_of_subset_rel_frontier:
fixes $S :: 'a::\text{real_normed_vector_set}$
assumes $T \text{ face_of } S \ T \neq S$
shows $T \subseteq \text{rel_frontier } S$
using *assms closure_subset face_of_disjoint_rel_interior face_of_imp_subset rel_frontier_def* by *fastforce*

lemma face_of_aff_dim_lt:
fixes $S :: 'a::\text{euclidean_space_set}$
assumes $\text{convex } S \ T \text{ face_of } S \ T \neq S$
shows $\text{aff_dim } T < \text{aff_dim } S$
proof –
have $\text{aff_dim } T \leq \text{aff_dim } S$
by (*simp add: face_of_imp_subset aff_dim_subset assms*)
moreover have $\text{aff_dim } T \neq \text{aff_dim } S$
by (*metis aff_dim_empty assms convex_rel_frontier_aff_dim face_of_imp_convex*)

```

      face_of_subset_rel_frontier order_less_irrefl)
    ultimately show ?thesis
      by simp
  qed

```

```

lemma subset_of_face_of_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes T: T face_of S and convex S U  $\subseteq$  S and dis:  $\neg$  disjnt (affine hull T)
  (rel_interior U)
  shows U  $\subseteq$  T
proof (rule subset_of_face_of [OF T  $\langle$ U  $\subseteq$  S $\rangle$ ])
  show T  $\cap$  rel_interior U  $\neq$  {}
    using face_of_imp_eq_affine_Int [OF  $\langle$ convex S $\rangle$  T] rel_interior_subset dis
   $\langle$ U  $\subseteq$  S $\rangle$  disjnt_def
    by fastforce
  qed

```

```

lemma affine_hull_face_of_disjoint_rel_interior:
  fixes S :: 'a::euclidean_space set
  assumes convex S F face_of S F  $\neq$  S
  shows affine hull F  $\cap$  rel_interior S = {}
  by (meson antisym assms disjnt_def equalityD2 face_of_def subset_of_face_of_affine_hull)

```

```

lemma affine_diff_divide:
  assumes affine S k  $\neq$  0 k  $\neq$  1 and xy: x  $\in$  S y /R (1 - k)  $\in$  S
  shows (x - y) /R k  $\in$  S
proof -
  have inverse(k) *R (x - y) = (1 - inverse k) *R inverse(1 - k) *R y +
  inverse(k) *R x
  using assms
  by (simp add: algebra_simps) (simp add: scaleR_left_distrib [symmetric]
  field_split_simps)
  then show ?thesis
    using  $\langle$ affine S $\rangle$  xy by (auto simp: affine_alt)
  qed

```

```

proposition face_of_conic:
  assumes conic S f face_of S
  shows conic f
  unfolding conic_def
proof (intro strip)
  fix x and c::real
  assume x  $\in$  f and 0  $\leq$  c
  have f:  $\bigwedge$  a b x.  $\llbracket$ a  $\in$  S; b  $\in$  S; x  $\in$  f; x  $\in$  open_segment a b $\rrbracket$   $\implies$  a  $\in$  f  $\wedge$  b  $\in$  f
  using  $\langle$ f face_of S $\rangle$  face_ofD by blast
  show c *R x  $\in$  f
proof (cases x=0  $\vee$  c=1)
  case True

```

```

then show ?thesis
  using ⟨ $x \in f$ ⟩ by auto
next
  case False
  with ⟨ $0 \leq c$ ⟩ obtain  $d e$  where  $de: 0 \leq d \ 0 \leq e \ d < 1 \ 1 < e \ d < e \ (d = c \vee e = c)$ 
    apply (simp add: neq_iff)
    by (metis gt_ex less_eq_real_def order_less_le_trans zero_less_one)
  then obtain [simp]:  $c *_R x \in S \ e *_R x \in S \ \langle x \in S \rangle$ 
    using ⟨ $x \in f$ ⟩ assms conic_mul face_of_imp_subset by blast
  have  $x \in \text{open\_segment} \ (d *_R x) \ (e *_R x)$  if  $c *_R x \notin f$ 
    using  $de$  False that
    apply (simp add: in_segment)
    apply (rule_tac  $x = (1 - d) / (e - d)$  in  $exI$ )
    apply (simp add: field_simps)
    by (smt (verit, del_insts) add_divide_distrib divide_self scaleR_collapse)
  then show ?thesis
    using ⟨conic  $S$ ⟩  $f$  [of  $d *_R x \ e *_R x \ x$ ]  $de$  ⟨ $x \in f$ ⟩
    by (force simp: conic_def in_segment)
qed
qed

```

proposition face_of_convex_hulls:

```

assumes  $S$ : finite  $S \ T \subseteq S$  and disj: affine_hull  $T \cap$  convex_hull  $(S - T) = \{\}$ 
shows (convex_hull  $T$ ) face_of (convex_hull  $S$ )

```

proof –

```

have  $fin$ : finite  $T$  finite  $(S - T)$  using  $assms$ 
by (auto simp: finite_subset)
have  $*$ :  $x \in$  convex_hull  $T$ 
  if  $x$ :  $x \in$  convex_hull  $S$  and  $y$ :  $y \in$  convex_hull  $S$  and  $w$ :  $w \in$  convex_hull
 $T \ w \in$  open_segment  $x \ y$ 
  for  $x \ y \ w$ 

```

proof –

```

have  $waff$ :  $w \in$  affine_hull  $T$ 
using convex_hull_subset_affine_hull  $w$  by blast
obtain  $a \ b$  where  $a$ :  $\bigwedge i. i \in S \implies 0 \leq a \ i$  and  $asum$ :  $\text{sum } a \ S = 1$  and
 $aeqx$ :  $(\sum_{i \in S} a \ i *_R i) = x$ 
  and  $b$ :  $\bigwedge i. i \in S \implies 0 \leq b \ i$  and  $bsum$ :  $\text{sum } b \ S = 1$  and  $beqy$ :
 $(\sum_{i \in S} b \ i *_R i) = y$ 
using  $x \ y$  by (auto simp:  $assms$  convex_hull_finite)
obtain  $u$  where  $(1 - u) *_R x + u *_R y \in$  convex_hull  $T \ x \neq y$  and  $weq$ :  $w$ 
 $= (1 - u) *_R x + u *_R y$ 
  and  $u01$ :  $0 < u \ u < 1$ 
using  $w$  by (auto simp: open_segment_image_interval_split: if_split_asm)
define  $c$  where  $c \ i = (1 - u) * a \ i + u * b \ i$  for  $i$ 
have  $cge0$ :  $\bigwedge i. i \in S \implies 0 \leq c \ i$ 
using  $a \ b \ u01$  by (simp add:  $c\_def$ )
have  $sumc1$ :  $\text{sum } c \ S = 1$ 

```

```

    by (simp add: c_def sum.distrib sum_distrib_left [symmetric] asum bsum)
  have sumci_xy:  $(\sum_{i \in S} c \ i *_{\mathbb{R}} \ i) = (1 - u) *_{\mathbb{R}} \ x + u *_{\mathbb{R}} \ y$ 
    apply (simp add: c_def sum.distrib scaleR_left_distrib)
  by (simp only: scaleR_scaleR [symmetric] Real_Vector_Spaces.scaleR_right.sum
[symmetric] aeqx beqy)
  show ?thesis
  proof (cases sum c (S - T) = 0)
    case True
      have ci0:  $\bigwedge i. i \in (S - T) \implies c \ i = 0$ 
        using True cge0 fin(2) sum_nonneg_eq_0_iff by auto
      have a0:  $a \ i = 0$  if  $i \in (S - T)$  for  $i$ 
        using ci0 [OF that] u01 a [of i] b [of i] that
      by (simp add: c_def Groups.ordered_comm_monoid_add_class.add_nonneg_eq_0_iff)
      have sum a T = 1
        using assms by (metis sum.mono_neutral_cong_right a0 asum)
      moreover have  $(\sum_{x \in T} a \ x *_{\mathbb{R}} \ x) = x$ 
        using a0 assms by (auto simp: cge0 a aeqx [symmetric] sum.mono_neutral_right)
      ultimately show ?thesis
        using a assms(2) by (auto simp add: convex_hull_finite ⟨finite T⟩)
    next
      case False
      define k where  $k = \text{sum } c \ (S - T)$ 
      have  $k > 0$  using False
      unfolding k_def by (metis DiffD1 antisym_conv cge0 sum_nonneg not_less)
      have weq_sumsum:  $w = \text{sum } (\lambda x. c \ x *_{\mathbb{R}} \ x) \ T + \text{sum } (\lambda x. c \ x *_{\mathbb{R}} \ x) \ (S - T)$ 
        by (metis (no_types) add commute S(1) S(2) sum.subset_diff sumci_xy
weq)
      show ?thesis
      proof (cases k = 1)
        case True
          then have sum c T = 0
            by (simp add: S k_def sum_diff sumc1)
          then have sum c (S - T) = 1
            by (simp add: S sum_diff sumc1)
          moreover have ci0:  $\bigwedge i. i \in T \implies c \ i = 0$ 
            by (meson ⟨finite T⟩ ⟨sum c T = 0⟩ ⟨T ⊆ S⟩ cge0 sum_nonneg_eq_0_iff
subsetCE)
          then have  $(\sum_{i \in S - T} c \ i *_{\mathbb{R}} \ i) = w$ 
            by (simp add: weq_sumsum)
          ultimately have  $w \in \text{convex hull } (S - T)$ 
            using cge0 by (auto simp add: convex_hull_finite fin)
          then show ?thesis
            using disj waff by blast
        next
          case False
          then have sumcf:  $\text{sum } c \ T = 1 - k$ 
            by (simp add: S k_def sum_diff sumc1)
          have  $\bigwedge x. x \in T \implies 0 \leq \text{inverse } (1 - k) * c \ x$ 

```



```

  by (metis ‹ $T \subseteq S$ › cge0 inverse_nonnegative_iff_nonnegative mult_nonneg_nonneg
subsetD sum_nonneg sumcf)
  moreover have  $(\sum x \in T. \text{inverse } (1 - k) * c x) = 1$ 
  by (metis False eq_iff_diff_eq_0 mult.commute right_inverse sum_distrib_left
sumcf)
  ultimately have  $(\sum i \in T. c i *_{\mathbb{R}} i) /_{\mathbb{R}} (1 - k) \in \text{convex hull } T$ 
  apply (simp add: convex_hull_finite fin)
  by (metis (mono_tags, lifting) scaleR_right.sum scaleR_scaleR sum.cong)
  with ‹ $0 < k$ › have  $\text{inverse}(k) *_{\mathbb{R}} (w - \text{sum } (\lambda i. c i *_{\mathbb{R}} i) T) \in \text{affine hull}$ 
T
  by (simp add: affine_diff_divide [OF affine_affine_hull] False waff
convex_hull_subset_affine_hull [THEN subsetD])
  moreover have  $\text{inverse}(k) *_{\mathbb{R}} (w - \text{sum } (\lambda x. c x *_{\mathbb{R}} x) T) \in \text{convex hull}$ 
(S - T)
  apply (simp add: weq_sumsum convex_hull_finite fin)
  apply (rule_tac x= $\lambda i. \text{inverse } k * c i$  in exI)
  using ‹ $k > 0$ › cge0
  apply (auto simp: scaleR_right.sum simp flip: sum_distrib_left k_def)
  done
  ultimately show ?thesis
  using disj by blast
qed
qed
qed
have [simp]:  $\text{convex hull } T \subseteq \text{convex hull } S$ 
  by (simp add: ‹ $T \subseteq S$ › hull_mono)
show ?thesis
  using open_segment_commute by (auto simp: face_of_def intro: *)
qed

```

proposition *face_of_convex_hull_insert:*

assumes *finite* S $a \notin \text{affine hull } S$ and $T: T \text{ face_of convex hull } S$

shows $T \text{ face_of convex hull insert } a S$

proof –

have $\text{convex hull } S \text{ face_of convex hull insert } a S$

by (simp add: assms face_of_convex_hulls insert_Diff_if subset_insertI)

then show ?thesis

using $T \text{ face_of_trans}$ by blast

qed

proposition *face_of_affine_trivial:*

assumes *affine* S $T \text{ face_of } S$

shows $T = \{\}$ \vee $T = S$

proof (*rule ccontr, clarsimp*)

assume $T \neq \{\}$ $T \neq S$

then obtain a where $a \in T$ by auto

then have $a \in S$

using ‹ $T \text{ face_of } S$ › *face_of_imp_subset* by blast

have $S \subseteq T$

```

proof
  fix  $b$  assume  $b \in S$ 
  show  $b \in T$ 
  proof (cases  $a = b$ )
    case True with  $\langle a \in T \rangle$  show ?thesis by auto
  next
    case False
    then have  $a \in \text{open\_segment } (2 *_{\mathbb{R}} a - b) b$ 
    by (metis diff_add_cancel inverse_eq_divide midpoint_def midpoint_in_open_segment

      scaleR_2 scaleR_half_double)
    moreover have  $2 *_{\mathbb{R}} a - b \in S$ 
    by (rule mem_affine [OF <affine S> <a \in S> <b \in S>, of 2 -1, simplified])
    moreover note  $\langle b \in S \rangle \langle a \in T \rangle$ 
    ultimately show ?thesis
    by (rule face_ofD [OF <T face_of S>, THEN conjunct2])
  qed
qed
then show False
  using  $\langle T \neq S \rangle \langle T \text{ face\_of } S \rangle \text{face\_of\_imp\_subset}$  by blast
qed

```

lemma *face_of_affine_eq*:

affine S $\implies (T \text{ face_of } S \iff T = \{\} \vee T = S)$

using *affine_imp_convex face_of_affine_trivial face_of_refl* **by** *auto*

proposition *Inter_faces_finite_altbound*:

fixes $T :: 'a::\text{euclidean_space}$ *set set*

assumes *cfaI*: $\bigwedge c. c \in T \implies c \text{ face_of } S$

shows $\exists F'. \text{finite } F' \wedge F' \subseteq T \wedge \text{card } F' \leq \text{DIM}('a) + 2 \wedge \bigcap F' = \bigcap T$

proof (*cases* $\forall F'. \text{finite } F' \wedge F' \subseteq T \wedge \text{card } F' \leq \text{DIM}('a) + 2 \longrightarrow (\exists c. c \in T \wedge c \cap (\bigcap F') \subset (\bigcap F'))$)

case *True*

then **obtain** c **where** c :

$\bigwedge F'. \llbracket \text{finite } F'; F' \subseteq T; \text{card } F' \leq \text{DIM}('a) + 2 \rrbracket \implies c \in F' \cap (\bigcap F') \subset (\bigcap F')$

by *metis*

define d **where** $d \equiv \lambda n. ((\lambda r. \text{insert } (c \ r) \ r) \rightsquigarrow n) \{c\}$

note d_def [*simp*]

have $d\text{Suc}$: $\bigwedge n. d (\text{Suc } n) = \text{insert } (c \ (d \ n)) \ (d \ n)$

by *simp*

have $dn_notempty$: $d \ n \neq \{\}$ **for** n

by (*induction n*) *auto*

have dn_le_Suc : $d \ n \subseteq T \wedge \text{finite}(d \ n) \wedge \text{card}(d \ n) \leq \text{Suc } n$ **if** $n \leq \text{DIM}('a) + 2$ **for** n

using *that*

proof (*induction n*)

```

    case 0
    then show ?case by (simp add: c)
  next
    case (Suc n)
    then show ?case by (auto simp: c card_insert_if)
  qed
  have aff_dim_le: aff_dim( $\bigcap$  (d n))  $\leq$  DIM('a) - int n if n  $\leq$  DIM('a) + 2 for
n
  using that
  proof (induction n)
    case 0
    then show ?case
      by (simp add: aff_dim_le_DIM)
  next
    case (Suc n)
    have fs:  $\bigcap$  (d (Suc n)) face_of S
      by (meson Suc.prem cfaI dn_le_Suc dn_notempty face_of_Inter subsetCE)
    have condn: convex ( $\bigcap$  (d n))
      using Suc.prem nat_le_linear not_less_eq_eq
      by (blast intro: face_of_imp_convex cfaI convex_Inter dest: dn_le_Suc)
    have fdn:  $\bigcap$  (d (Suc n)) face_of  $\bigcap$  (d n)
      by (metis (no_types, lifting) Inter_anti_mono Suc.prem dSuc cfaI dn_le_Suc
dn_notempty face_of_Inter face_of_imp_subset face_of_subset subset_iff sub-
set_insertI)
    have ne:  $\bigcap$  (d (Suc n))  $\neq$   $\bigcap$  (d n)
      by (metis (no_types, lifting) Suc.prem Suc_leD c complete_lattice_class.Inf_insert
dSuc dn_le_Suc less_irrefl order.trans)
    have *:  $\bigwedge m::int. \bigwedge d. \bigwedge d':int. d < d' \wedge d' \leq m - n \implies d \leq m - \text{of\_nat}(n+1)$ 
      by arith
    have aff_dim ( $\bigcap$  (d (Suc n)))  $<$  aff_dim ( $\bigcap$  (d n))
      by (rule face_of_aff_dim_lt [OF condn fdn ne])
    moreover have aff_dim ( $\bigcap$  (d n))  $\leq$  int (DIM('a)) - int n
      using Suc by auto
    ultimately
    have aff_dim ( $\bigcap$  (d (Suc n)))  $\leq$  int (DIM('a)) - (n+1) by arith
    then show ?case by linarith
  qed
  have aff_dim ( $\bigcap$  (d (DIM('a) + 2)))  $\leq$  -2
    using aff_dim_le [OF order_refl] by simp
  with aff_dim_geq [of  $\bigcap$  (d (DIM('a) + 2))] show ?thesis
    using order.trans by fastforce
  next
    case False
    then show ?thesis by fastforce
  qed

lemma faces_of_translation:
  {F. F face_of (+) a ' S} = (image ((+) a)) ' {F. F face_of S}
proof -

```

3432

have $\bigwedge F. F \text{ face_of } (+) a \text{ ' } S \implies \exists G. G \text{ face_of } S \wedge F = (+) a \text{ ' } G$
by (metis face_of_imp_subset face_of_translation_eq subset_imageE)
then show ?thesis
by (auto simp: image_iff)
qed

proposition face_of_Times:

assumes $F \text{ face_of } S$ and $F' \text{ face_of } S'$

shows $(F \times F') \text{ face_of } (S \times S')$

proof –

have $F \times F' \subseteq S \times S'$

using assms [unfolded face_of_def] by blast

moreover

have convex $(F \times F')$

using assms [unfolded face_of_def] by (blast intro: convex_Times)

moreover

have $a \in F \wedge a' \in F' \wedge b \in F \wedge b' \in F'$

if $a \in S \wedge b \in S \wedge a' \in S' \wedge b' \in S' \wedge x \in F \times F' \wedge x \in \text{open_segment } (a, a') (b, b')$

for $a \ b \ a' \ b' \ x$

proof (cases $b=a \vee b'=a'$)

case True with that show ?thesis

using assms

by (force simp: in_segment dest: face_ofD)

next

case False with assms [unfolded face_of_def] that show ?thesis

by (blast dest!: open_segment_PairD)

qed

ultimately show ?thesis

unfolding face_of_def by blast

qed

corollary face_of_Times_decomp:

fixes $S :: 'a::\text{euclidean_space set}$ and $S' :: 'b::\text{euclidean_space set}$

shows $C \text{ face_of } (S \times S') \iff (\exists F F'. F \text{ face_of } S \wedge F' \text{ face_of } S' \wedge C = F \times F')$

(is ?lhs = ?rhs)

proof

assume $C: ?lhs$

show ?rhs

proof (cases $C = \{\}$)

case True then show ?thesis by auto

next

case False

have 1: $\text{fst ' } C \subseteq S$ and 2: $\text{snd ' } C \subseteq S'$

using $C \text{ face_of_imp_subset}$ by fastforce+

have convex C

using C by (metis face_of_imp_convex)

have conv: convex $(\text{fst ' } C)$ convex $(\text{snd ' } C)$

by (simp_all add: convex_C convex_linear_image linear_fst linear_snd)

```

have fstab: a ∈ fst ' C ∧ b ∈ fst ' C
  if a ∈ S b ∈ S x ∈ open_segment a b (x,x') ∈ C for a b x x'
proof -
  have *: (x,x') ∈ open_segment (a,x') (b,x')
    using that by (auto simp: in_segment)
  show ?thesis
    using face_ofD [OF C *] that face_of_imp_subset [OF C] by force
qed
have fst: fst ' C face_of S
  by (force simp: face_of_def 1 conv fstab)
have sndab: a' ∈ snd ' C ∧ b' ∈ snd ' C
  if a' ∈ S' b' ∈ S' x' ∈ open_segment a' b' (x,x') ∈ C for a' b' x x'
proof -
  have *: (x,x') ∈ open_segment (x,a') (x,b')
    using that by (auto simp: in_segment)
  show ?thesis
    using face_ofD [OF C *] that face_of_imp_subset [OF C] by force
qed
have snd: snd ' C face_of S'
  by (force simp: face_of_def 1 conv sndab)
have cc: rel_interior C ⊆ rel_interior (fst ' C) × rel_interior (snd ' C)
  by (force simp: face_of_Times rel_interior_Times conv fst snd ‹convex C›
linear_fst linear_snd rel_interior_convex_linear_image [symmetric])
have C = fst ' C × snd ' C
proof (rule face_of_eq [OF C])
  show fst ' C × snd ' C face_of S × S'
    by (simp add: face_of_Times rel_interior_Times conv fst snd)
  show rel_interior C ∩ rel_interior (fst ' C × snd ' C) ≠ {}
    using False rel_interior_eq_empty ‹convex C› cc
    by (auto simp: face_of_Times rel_interior_Times conv fst)
qed
with fst snd show ?thesis by metis
qed
qed (use face_of_Times in auto)

```

lemma face_of_Times_eq:

```

fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
shows (F × F') face_of (S × S') ↔ F = {} ∨ F' = {} ∨ F face_of S ∧ F'
face_of S'
by (auto simp: face_of_Times_decomp times_eq_iff)

```

lemma hyperplane_face_of_halfspace_le: {x. a · x = b} face_of {x. a · x ≤ b}

proof -

```

have {x. a · x ≤ b} ∩ {x. a · x = b} = {x. a · x = b}

```

by auto

```

with face_of_Int_supporting_hyperplane_le [OF convex_halfspace_le [of a b],
of a b]

```

```

show ?thesis by auto

```

qed

```

lemma hyperplane_face_of_halfspace_ge: {x. a · x = b} face_of {x. a · x ≥ b}
proof -
  have {x. a · x ≥ b} ∩ {x. a · x = b} = {x. a · x = b}
    by auto
  with face_of_Int_supporting_hyperplane_ge [OF convex_halfspace_ge [of b a],
of b a]
  show ?thesis by auto
qed

```

```

lemma face_of_halfspace_le:
  fixes a :: 'n::euclidean_space
  shows F face_of {x. a · x ≤ b} ↔ F = {} ∨ F = {x. a · x = b} ∨ F = {x.
a · x ≤ b}
  (is ?lhs = ?rhs)
proof (cases a = 0)
  case True then show ?thesis
    using face_of_affine_eq affine_UNIV by auto
  next
  case False
  then have ine: interior {x. a · x ≤ b} ≠ {}
    using halfspace_eq_empty_lt interior_halfspace_le by blast
  show ?thesis
  proof
    assume L: ?lhs
    have F face_of {x. a · x = b} if F ≠ {x. a · x ≤ b}
    proof -
      have F face_of rel_frontier {x. a · x ≤ b}
      proof (rule face_of_subset [OF L])
        show F ⊆ rel_frontier {x. a · x ≤ b}
          by (simp add: L face_of_subset_rel_frontier that)
      qed (force simp: rel_frontier_def closed_halfspace_le)
      then show ?thesis
        using False
        by (simp add: frontier_halfspace_le rel_frontier_nonempty_interior [OF
ine])
      qed
    with L show ?rhs
      using affine_hyperplane face_of_affine_eq by blast
  next
  assume ?rhs
  then show ?lhs
    by (metis convex_halfspace_le empty_face_of_face_of_refl hyperplane_face_of_halfspace_le)
  qed
qed

```

```

lemma face_of_halfspace_ge:
  fixes a :: 'n::euclidean_space
  shows F face_of {x. a · x ≥ b} ↔ F = {} ∨ F = {x. a · x = b} ∨ F = {x.

```

$a \cdot x \geq b$
using *face_of_halfspace_le* [of $F -a -b$] **by** *simp*

9.24.2 Exposed faces

That is, faces that are intersection with supporting hyperplane

definition *exposed_face_of* :: [$'a::\text{euclidean_space set}$, $'a \text{ set}$] \Rightarrow *bool*
 (**infixr** (*exposed'_face'_of*) 50)

where T *exposed_face_of* $S \longleftrightarrow$
 $T \text{ face_of } S \wedge (\exists a b. S \subseteq \{x. a \cdot x \leq b\} \wedge T = S \cap \{x. a \cdot x = b\})$

lemma *empty_exposed_face_of* [*iff*]: $\{\}$ *exposed_face_of* S

proof –

have $S \subseteq \{x. 0 \cdot x \leq 1\} \wedge \{\} = S \cap \{x. 0 \cdot x = 1\}$

by *force*

then show *?thesis*

using *exposed_face_of_def* **by** *blast*

qed

lemma *exposed_face_of_refl_eq* [*simp*]: S *exposed_face_of* $S \longleftrightarrow$ *convex* S

proof

assume S : *convex* S

have $S \subseteq \{x. 0 \cdot x \leq 0\} \wedge S = S \cap \{x. 0 \cdot x = 0\}$

by *auto*

with S **show** S *exposed_face_of* S

using *exposed_face_of_def* *face_of_refl_eq* **by** *blast*

qed (*simp add: exposed_face_of_def face_of_refl_eq*)

lemma *exposed_face_of_refl*: *convex* $S \Longrightarrow S$ *exposed_face_of* S

by *simp*

lemma *exposed_face_of*:

T *exposed_face_of* $S \longleftrightarrow$

$T \text{ face_of } S \wedge (T = \{\} \vee T = S \vee$

$(\exists a b. a \neq 0 \wedge S \subseteq \{x. a \cdot x \leq b\} \wedge T = S \cap \{x. a \cdot x = b\}))$

(**is** *?lhs = ?rhs*)

proof

show *?lhs* \Longrightarrow *?rhs*

by (*smt* (*verit*) *Collect_cong* *exposed_face_of_def* *hyperplane_eq_empty* *inf.absorb_iff1*
inf_bot_right *inner_zero_left*)

show *?rhs* \Longrightarrow *?lhs*

using *exposed_face_of_def* *face_of_imp_convex* **by** *fastforce*

qed

lemma *exposed_face_of_Int_supporting_hyperplane_le*:

$\llbracket \text{convex } S; \bigwedge x. x \in S \Longrightarrow a \cdot x \leq b \rrbracket \Longrightarrow (S \cap \{x. a \cdot x = b\})$ *exposed_face_of*
 S

by (*force* *simp: exposed_face_of_def face_of_Int_supporting_hyperplane_le*)

lemma *exposed_face_of_Int_supporting_hyperplane_ge*:

$\llbracket \text{convex } S; \bigwedge x. x \in S \implies a \cdot x \geq b \rrbracket \implies (S \cap \{x. a \cdot x = b\}) \text{ exposed_face_of } S$

using *exposed_face_of_Int_supporting_hyperplane_le* [of $S -a -b$] **by** *simp*

proposition *exposed_face_of_Int*:

assumes $T \text{ exposed_face_of } S$

and $U \text{ exposed_face_of } S$

shows $(T \cap U) \text{ exposed_face_of } S$

proof –

obtain $a \ b$ **where** $T: S \cap \{x. a \cdot x = b\} \text{ face_of } S$

and $S: S \subseteq \{x. a \cdot x \leq b\}$

and $teq: T = S \cap \{x. a \cdot x = b\}$

using *assms* **by** (*auto simp: exposed_face_of_def*)

obtain $a' \ b'$ **where** $U: S \cap \{x. a' \cdot x = b'\} \text{ face_of } S$

and $s': S \subseteq \{x. a' \cdot x \leq b'\}$

and $ueq: U = S \cap \{x. a' \cdot x = b'\}$

using *assms* **by** (*auto simp: exposed_face_of_def*)

have $tu: T \cap U \text{ face_of } S$

using $T \ teq \ U \ ueq$ **by** (*simp add: face_of_Int*)

have $ss: S \subseteq \{x. (a + a') \cdot x \leq b + b'\}$

using $S \ s'$ **by** (*force simp: inner_left_distrib*)

have $S \subseteq \{x. (a + a') \cdot x \leq b + b'\} \wedge T \cap U = S \cap \{x. (a + a') \cdot x = b + b'\}$

using $S \ s'$ **by** (*fastforce simp: ss inner_left_distrib teq ueq*)

then show *?thesis*

using *exposed_face_of_def tu* **by** *auto*

qed

proposition *exposed_face_of_Inter*:

fixes $P :: 'a::\text{euclidean_space} \text{ set set}$

assumes $P \neq \{\}$

and $\bigwedge T. T \in P \implies T \text{ exposed_face_of } S$

shows $\bigcap P \text{ exposed_face_of } S$

proof –

obtain Q **where** *finite* Q **and** $Q_{\text{sub}P}: Q \subseteq P \ \text{card } Q \leq \text{DIM}('a) + 2$ **and** $\text{Int}Q:$
 $\bigcap Q = \bigcap P$

using *Inter_faces_finite_altbound* [of $P \ S$] *assms* [*unfolded exposed_face_of*]
by *force*

show *?thesis*

proof (*cases* $Q = \{\}$)

case *True* **then show** *?thesis*

by (*metis IntQ Inter_UNIV_conv(2) assms(1) assms(2) ex_in_conv*)

next

case *False*

have $Q \subseteq \{T. T \text{ exposed_face_of } S\}$

using $Q_{\text{sub}P}$ *assms* **by** *blast*

moreover have $Q \subseteq \{T. T \text{ exposed_face_of } S\} \implies \bigcap Q \text{ exposed_face_of } S$

using $\langle \text{finite } Q \rangle$ *False*

by (*induction* Q *rule: finite_induct; use exposed_face_of_Int in fastforce*)


```

ultimately show ?thesis
  by (simp add: IntQ)
qed
qed

proposition exposed_face_of_sums:
assumes convex S and convex T
  and F exposed_face_of {x + y | x y. x ∈ S ∧ y ∈ T}
  (is F exposed_face_of ?ST)
obtains k l
  where k exposed_face_of S l exposed_face_of T
   $F = \{x + y \mid x y. x \in k \wedge y \in l\}$ 
proof (cases F = {})
  case True then show ?thesis
    using that by blast
next
  case False
show ?thesis
proof (cases F = ?ST)
  case True then show ?thesis
    using assms exposed_face_of_refl_eq that by blast
next
  case False
obtain p where p ∈ F using ⟨F ≠ {}⟩ by blast
moreover
obtain u z where T: ?ST ∩ {x. u · x = z} face_of ?ST
  and S: ?ST ⊆ {x. u · x ≤ z}
  and feq: F = ?ST ∩ {x. u · x = z}
  using assms by (auto simp: exposed_face_of_def)
ultimately obtain a0 b0
  where p: p = a0 + b0 and a0 ∈ S b0 ∈ T and z: u · p = z
  by auto
have lez: u · (x + y) ≤ z if x ∈ S y ∈ T for x y
  using S that by auto
have sef: S ∩ {x. u · x = u · a0} exposed_face_of S
proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF ⟨convex S⟩])
  show  $\bigwedge x. x \in S \implies u \cdot x \leq u \cdot a0$ 
  by (metis p z add_le_cancel_right inner_right_distrib lez [OF _ ⟨b0 ∈ T⟩])
qed
have tef: T ∩ {x. u · x = u · b0} exposed_face_of T
proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF ⟨convex T⟩])
  show  $\bigwedge x. x \in T \implies u \cdot x \leq u \cdot b0$ 
  by (metis p z add.commute add_le_cancel_right inner_right_distrib lez [OF ⟨a0 ∈ S⟩])
qed
have  $\{x + y \mid x y. x \in S \wedge u \cdot x = u \cdot a0 \wedge y \in T \wedge u \cdot y = u \cdot b0\} \subseteq F$ 
  by (auto simp: feq) (metis inner_right_distrib p z)
moreover have  $F \subseteq \{x + y \mid x y. x \in S \wedge u \cdot x = u \cdot a0 \wedge y \in T \wedge u \cdot y =$ 

```

3438

```

u · b0}
proof –
  have  $\bigwedge x y. \llbracket z = u \cdot (x + y); x \in S; y \in T \rrbracket$ 
     $\implies u \cdot x = u \cdot a0 \wedge u \cdot y = u \cdot b0$ 
  by (smt (verit, best) z p <a0 ∈ S> <b0 ∈ T> inner_right_distrib lez)
  then show ?thesis
  using feq by blast
qed
ultimately have  $F = \{x + y \mid x y. x \in S \cap \{x. u \cdot x = u \cdot a0\} \wedge y \in T \cap \{x. u \cdot x = u \cdot b0\}\}$ 
by blast
then show ?thesis
by (rule that [OF sef tef])
qed
qed

```

proposition *exposed_face_of_parallel:*

$$T \text{ exposed_face_of } S \iff$$

$$T \text{ face_of } S \wedge$$

$$(\exists a b. S \subseteq \{x. a \cdot x \leq b\} \wedge T = S \cap \{x. a \cdot x = b\} \wedge$$

$$(T \neq \{\} \longrightarrow T \neq S \longrightarrow a \neq 0) \wedge$$

$$(T \neq S \longrightarrow (\forall w \in \text{affine hull } S. (w + a) \in \text{affine hull } S)))$$

(is ?lhs = ?rhs)

proof

assume ?lhs **then show** ?rhs

proof (clarsimp simp: exposed_face_of_def)

fix a b

assume faceS: $S \cap \{x. a \cdot x = b\}$ face_of S **and** Ssub: $S \subseteq \{x. a \cdot x \leq b\}$

show $\exists c d. S \subseteq \{x. c \cdot x \leq d\} \wedge$

$S \cap \{x. a \cdot x = b\} = S \cap \{x. c \cdot x = d\} \wedge$

$(S \cap \{x. a \cdot x = b\} \neq \{\} \longrightarrow S \cap \{x. a \cdot x = b\} \neq S \longrightarrow c \neq 0) \wedge$

$(S \cap \{x. a \cdot x = b\} \neq S \longrightarrow (\forall w \in \text{affine hull } S. w + c \in \text{affine hull$

S))

proof (cases affine hull S $\cap \{x. -a \cdot x \leq -b\} = \{\} \vee \text{affine hull } S \subseteq \{x. -a \cdot x \leq -b\}$)

case True

then show ?thesis

proof

assume affine hull S $\cap \{x. -a \cdot x \leq -b\} = \{\}$

then show ?thesis

apply (rule_tac x=0 in exI)

apply (rule_tac x=1 in exI)

using hull_subset **by** fastforce

next

assume affine hull S $\subseteq \{x. -a \cdot x \leq -b\}$

then show ?thesis

apply (rule_tac x=0 in exI)

apply (rule_tac x=0 in exI)

using Ssub hull_subset **by** fastforce

```

qed
next
case False
then obtain a' b' where a' ≠ 0
  and le: affine_hull S ∩ {x. a' · x ≤ b'} = affine_hull S ∩ {x. - a · x ≤ - b}
  and eq: affine_hull S ∩ {x. a' · x = b'} = affine_hull S ∩ {x. - a · x = - b}
  and mem:  $\bigwedge w. w \in \text{affine\_hull } S \implies w + a' \in \text{affine\_hull } S$ 
  using affine_parallel_slice affine_affine_hull by metis
show ?thesis
proof (intro conjI impI allI ballI exI)
  have *:  $S \subseteq - (\text{affine\_hull } S \cap \{x. P\ x\}) \cup \text{affine\_hull } S \cap \{x. Q\ x\} \implies S$ 
 $\subseteq \{x. \neg P\ x \vee Q\ x\}$ 
  for P Q
  using hull_subset by fastforce
  have  $S \subseteq \{x. \neg (a' \cdot x \leq b') \vee a' \cdot x = b'\}$ 
  by (rule *) (use le eq Ssub in auto)
  then show  $S \subseteq \{x. - a' \cdot x \leq - b'\}$ 
  by auto
  show  $S \cap \{x. a \cdot x = b\} = S \cap \{x. - a' \cdot x = - b'\}$ 
  using eq hull_subset [of S affine] by force
  show  $\llbracket S \cap \{x. a \cdot x = b\} \neq \{\}; S \cap \{x. a \cdot x = b\} \neq S \rrbracket \implies - a' \neq 0$ 
  using  $\langle a' \neq 0 \rangle$  by auto
  show  $w + - a' \in \text{affine\_hull } S$ 
  if  $S \cap \{x. a \cdot x = b\} \neq S$   $w \in \text{affine\_hull } S$  for w
  proof -
    have  $w + 1 *_R (w - (w + a')) \in \text{affine\_hull } S$ 
    using affine_affine_hull mem mem_affine_3_minus that(2) by blast
    then show ?thesis by simp
  qed
qed
qed
qed
next
assume ?rhs then show ?lhs
  unfolding exposed_face_of_def by blast
qed

```

9.24.3 Extreme points of a set: its singleton faces

definition *extreme_point_of* :: $[a::\text{real_vector}, 'a\ \text{set}] \Rightarrow \text{bool}$
 $(\text{infixr } (\text{extreme_point_of})\ 50)$

where x *extreme_point_of* $S \iff$
 $x \in S \wedge (\forall a \in S. \forall b \in S. x \notin \text{open_segment } a\ b)$

lemma *extreme_point_of_stillconvex*:

$\text{convex } S \implies (x \text{ extreme_point_of } S \iff x \in S \wedge \text{convex}(S - \{x\}))$

by (fastforce simp add: convex_contains_segment extreme_point_of_def open_segment_def)

lemma *face_of_singleton*:

$\{x\}$ *face_of* $S \longleftrightarrow x$ *extreme_point_of* S
by (*fastforce simp add: extreme_point_of_def face_of_def*)

lemma *extreme_point_not_in_REL_INTERIOR*:

fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\llbracket x$ *extreme_point_of* $S; S \neq \{x\} \rrbracket \implies x \notin \text{rel_interior } S$
by (*metis disjoint_iff face_of_disjoint_rel_interior face_of_singleton insertI1*)

lemma *extreme_point_not_in_interior*:

fixes $S :: 'a::\{\text{real_normed_vector}, \text{perfect_space}\}$ *set*
assumes x *extreme_point_of* S **shows** $x \notin \text{interior } S$
using *assms extreme_point_not_in_REL_INTERIOR interior_subset_rel_interior*
by fastforce

lemma *extreme_point_of_face*:

F *face_of* $S \implies v$ *extreme_point_of* $F \longleftrightarrow v$ *extreme_point_of* $S \wedge v \in F$
by (*meson empty_subsetI face_of_face face_of_singleton insert_subset*)

lemma *extreme_point_of_convex_hull*:

x *extreme_point_of* (*convex hull* S) $\implies x \in S$
using *hull_minimal* [*of* S (*convex hull* S) - $\{x\}$ *convex*]
using *hull_subset* [*of* S *convex*]
by (*force simp add: extreme_point_of_stillconvex*)

proposition *extreme_points_of_convex_hull*:

$\{x. x$ *extreme_point_of* (*convex hull* $S\})\} \subseteq S$
using *extreme_point_of_convex_hull* **by auto**

lemma *extreme_point_of_empty* [*simp*]: $\neg (x$ *extreme_point_of* $\{\})$

by (*simp add: extreme_point_of_def*)

lemma *extreme_point_of_singleton* [*iff*]: x *extreme_point_of* $\{a\} \longleftrightarrow x = a$

using *extreme_point_of_stillconvex* **by auto**

lemma *extreme_point_of_translation_eq*:

$(a + x)$ *extreme_point_of* (*image* $(\lambda x. a + x)$ S) $\longleftrightarrow x$ *extreme_point_of* S
by (*auto simp: extreme_point_of_def*)

lemma *extreme_points_of_translation*:

$\{x. x$ *extreme_point_of* (*image* $(\lambda x. a + x)$ $S\}) =$
 $(\lambda x. a + x) ' \{x. x$ *extreme_point_of* $S\}$
using *extreme_point_of_translation_eq*
by auto (*metis (no_types, lifting) image_iff mem_Collect_eq minus_add_cancel*)

lemma *extreme_points_of_translation_subtract*:

$\{x. x$ *extreme_point_of* (*image* $(\lambda x. x - a)$ $S\}) =$
 $(\lambda x. x - a) ' \{x. x$ *extreme_point_of* $S\}$
using *extreme_points_of_translation* [*of* $- a$ S]
by simp

lemma *extreme_point_of_Int*:

$\llbracket x \text{ extreme_point_of } S; x \text{ extreme_point_of } T \rrbracket \Longrightarrow x \text{ extreme_point_of } (S \cap T)$
by (*simp add: extreme_point_of_def*)

lemma *extreme_point_of_Int_supporting_hyperplane_le*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \Longrightarrow a \cdot x \leq b \rrbracket \Longrightarrow c \text{ extreme_point_of } S$
by (*metis convex_singleton face_of_Int_supporting_hyperplane_le strong face_of_singleton*)

lemma *extreme_point_of_Int_supporting_hyperplane_ge*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \Longrightarrow a \cdot x \geq b \rrbracket \Longrightarrow c \text{ extreme_point_of } S$
using *extreme_point_of_Int_supporting_hyperplane_le* [*of S -a -b c*]
by *simp*

lemma *exposed_point_of_Int_supporting_hyperplane_le*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \Longrightarrow a \cdot x \leq b \rrbracket \Longrightarrow \{c\} \text{ exposed_face_of } S$
unfolding *exposed_face_of_def*
by (*force simp: face_of_singleton extreme_point_of_Int_supporting_hyperplane_le*)

lemma *exposed_point_of_Int_supporting_hyperplane_ge*:

$\llbracket S \cap \{x. a \cdot x = b\} = \{c\}; \bigwedge x. x \in S \Longrightarrow a \cdot x \geq b \rrbracket \Longrightarrow \{c\} \text{ exposed_face_of } S$
using *exposed_point_of_Int_supporting_hyperplane_le* [*of S -a -b c*]
by *simp*

lemma *extreme_point_of_convex_hull_insert*:

assumes *finite S a \notin convex hull S*
shows *a extreme_point_of (convex hull (insert a S))*
proof (*cases a \in S*)
case *False*
then show *?thesis*
using *face_of_convex_hulls* [*of insert a S {a}*] *assms*
by (*auto simp: face_of_singleton hull_same*)
qed (*use assms in <simp add: hull_inc>*)

lemma *extreme_point_of_conic*:

assumes *conic S and x: x extreme_point_of S*
shows *x = 0*
proof –
have *{x} face_of S*
by (*simp add: face_of_singleton x*)
then have *conic{x}*
using *assms(1) face_of_conic* **by** *blast*
then show *?thesis*
by (*force simp: conic_def*)
qed

9.24.4 Facets

definition *facet_of* :: [*a*::euclidean_space set, 'a set] \Rightarrow bool

(**infixr** (*facet'_of*) 50)

where $F \text{ facet_of } S \iff F \text{ face_of } S \wedge F \neq \{\} \wedge \text{aff_dim } F = \text{aff_dim } S - 1$

lemma *facet_of_empty* [*simp*]: $\neg S \text{ facet_of } \{\}$

by (*simp add: facet_of_def*)

lemma *facet_of_irrefl* [*simp*]: $\neg S \text{ facet_of } S$

by (*simp add: facet_of_def*)

lemma *facet_of_imp_face_of*: $F \text{ facet_of } S \implies F \text{ face_of } S$

by (*simp add: facet_of_def*)

lemma *facet_of_imp_subset*: $F \text{ facet_of } S \implies F \subseteq S$

by (*simp add: face_of_imp_subset facet_of_def*)

lemma *hyperplane_facet_of_halfspace_le*:

$a \neq 0 \implies \{x. a \cdot x = b\} \text{ facet_of } \{x. a \cdot x \leq b\}$

unfolding *facet_of_def hyperplane_eq_empty*

by (*auto simp: hyperplane_face_of_halfspace_ge hyperplane_face_of_halfspace_le Suc_leI of_nat_diff aff_dim_halfspace_le*)

lemma *hyperplane_facet_of_halfspace_ge*:

$a \neq 0 \implies \{x. a \cdot x = b\} \text{ facet_of } \{x. a \cdot x \geq b\}$

unfolding *facet_of_def hyperplane_eq_empty*

by (*auto simp: hyperplane_face_of_halfspace_le hyperplane_face_of_halfspace_ge Suc_leI of_nat_diff aff_dim_halfspace_ge*)

lemma *facet_of_halfspace_le*:

$F \text{ facet_of } \{x. a \cdot x \leq b\} \iff a \neq 0 \wedge F = \{x. a \cdot x = b\}$

(**is** ?lhs = ?rhs)

proof

assume *c*: ?lhs

with *c* *facet_of_irrefl* **show** ?rhs

by (*force simp: aff_dim_halfspace_le facet_of_def face_of_halfspace_le cong: conj_cong split: if_split_asm*)

next

assume ?rhs **then show** ?lhs

by (*simp add: hyperplane_facet_of_halfspace_le*)

qed

lemma *facet_of_halfspace_ge*:

$F \text{ facet_of } \{x. a \cdot x \geq b\} \iff a \neq 0 \wedge F = \{x. a \cdot x = b\}$

using *facet_of_halfspace_le* [*of F -a -b*] **by** *simp*

9.24.5 Edges: faces of affine dimension 1

definition $edge_of :: ['a::euclidean_space\ set, 'a\ set] \Rightarrow bool$ (**infixr** ($edge_of$) 50)

where $e\ edge_of\ S \longleftrightarrow e\ face_of\ S \wedge aff_dim\ e = 1$

lemma $edge_of_imp_subset$:

$S\ edge_of\ T \Longrightarrow S \subseteq T$

by ($simp\ add: edge_of_def\ face_of_imp_subset$)

9.24.6 Existence of extreme points

proposition $different_norm_3_collinear_points$:

fixes $a :: 'a::euclidean_space$

assumes $x \in open_segment\ a\ b$ $norm(a) = norm(b)$ $norm(x) = norm(b)$

shows $False$

proof –

obtain u **where** $norm\ ((1 - u) *_{\mathbb{R}} a + u *_{\mathbb{R}} b) = norm\ b$

and $a \neq b$

and $u01: 0 < u < 1$

using $assms$ **by** ($auto\ simp: open_segment_image_interval\ if_splits$)

then have $(1 - u) *_{\mathbb{R}} a \cdot (1 - u) *_{\mathbb{R}} a + ((1 - u) *_{\mathbb{R}} 2) *_{\mathbb{R}} a \cdot u *_{\mathbb{R}} b =$
 $(1 - u * u) *_{\mathbb{R}} (a \cdot a)$

using $assms$ **by** ($simp\ add: norm_eq\ algebra_simps\ inner_commute$)

then have $(1 - u) *_{\mathbb{R}} ((1 - u) *_{\mathbb{R}} a \cdot a + (2 * u) *_{\mathbb{R}} a \cdot b) =$
 $(1 - u) *_{\mathbb{R}} ((1 + u) *_{\mathbb{R}} (a \cdot a))$

by ($simp\ add: algebra_simps$)

then have $(1 - u) *_{\mathbb{R}} (a \cdot a) + (2 * u) *_{\mathbb{R}} (a \cdot b) = (1 + u) *_{\mathbb{R}} (a \cdot a)$

using $u01$ **by** $auto$

then have $a \cdot b = a \cdot a$

using $u01$ **by** ($simp\ add: algebra_simps$)

then have $a = b$

using $\langle norm(a) = norm(b) \rangle\ norm_eq\ vector_eq$ **by** $fastforce$

then show $?thesis$

using $\langle a \neq b \rangle$ **by** $force$

qed

proposition $extreme_point_exists_convex$:

fixes $S :: 'a::euclidean_space\ set$

assumes $compact\ S$ $convex\ S$ $S \neq \{\}$

obtains x **where** $x\ extreme_point_of\ S$

proof –

obtain x **where** $x \in S$ **and** $xsup: \bigwedge y. y \in S \Longrightarrow norm\ y \leq norm\ x$

using $distance_attains_sup$ [of S 0] $assms$ **by** $auto$

have $False$ **if** $a \in S$ $b \in S$ **and** $x: x \in open_segment\ a\ b$ **for** $a\ b$

proof –

have $noax: norm\ a \leq norm\ x$ **and** $nobx: norm\ b \leq norm\ x$ **using** $xsup$ **that**

by $auto$

have $a \neq b$

using $empty_iff\ open_segment_idem\ x$ **by** $auto$

```

  show False
  by (metis dist_0_norm dist_decreases_open_segment noax nobx not_le x)
qed
then show ?thesis
  by (meson <x ∈ S> extreme_point_of_def that)
qed

```

9.24.7 Krein-Milman, the weaker form

proposition *Krein_Milman*:

fixes $S :: 'a::euclidean_space\ set$

assumes $compact\ S\ convex\ S$

shows $S = closure(convex\ hull\ \{x.\ x\ extreme_point_of\ S\})$

proof (cases $S = \{\}$)

case *True* **then show** *?thesis* **by** *simp*

next

case *False*

have $closed\ S$

by (*simp add: <compact S> compact_imp_closed*)

have $closure(convex\ hull\ \{x.\ x\ extreme_point_of\ S\}) \subseteq S$

by (*simp add: <closed S> assms closure_minimal extreme_point_of_def hull_minimal*)

moreover have $u \in closure(convex\ hull\ \{x.\ x\ extreme_point_of\ S\})$

if $u \in S$ **for** u

proof (*rule ccontr*)

assume $u \notin closure(convex\ hull\ \{x.\ x\ extreme_point_of\ S\})$

then obtain $a\ b$ **where** $a \cdot u < b$

and $ab: \bigwedge x.\ x \in closure(convex\ hull\ \{x.\ x\ extreme_point_of\ S\}) \implies b < a \cdot x$

using *separating_hyperplane_closed_point* [*of closure(convex hull {x. x extreme_point_of S})*]

by *blast*

have $continuous_on\ S\ ((\cdot)\ a)$

by (*rule continuous_intros*)⁺

then obtain m **where** $m \in S$ **and** $m: \bigwedge y.\ y \in S \implies a \cdot m \leq a \cdot y$

using *continuous_attains_inf* [*of S λx. a · x*] *<compact S>* *<u ∈ S>*

by *auto*

define T **where** $T = S \cap \{x.\ a \cdot x = a \cdot m\}$

have $m \in T$

by (*simp add: T_def <m ∈ S>*)

moreover have $compact\ T$

by (*simp add: T_def compact_Int_closed* [*OF <compact S> closed_hyperplane*])

moreover have $convex\ T$

by (*simp add: T_def convex_Int* [*OF <convex S> convex_hyperplane*])

ultimately obtain v **where** $v: v\ extreme_point_of\ T$

using *extreme_point_exists_convex* [*of T*] **by** *auto*

then have $\{v\}\ face_of\ T$

by (*simp add: face_of_singleton*)

also have $T\ face_of\ S$

by (*simp add: T_def m face_of_Int_supporting_hyperplane_ge* [*OF <convex*


```

S>])
  finally have v extreme_point_of S
    by (simp add: face_of_singleton)
  then have b < a · v
    using closure_subset by (simp add: closure_hull hull_inc ab)
  then show False
    using ⟨a · u < b⟩ ⟨{v} face_of T⟩ face_of_imp_subset m T_def that by
fastforce
  qed
  ultimately show ?thesis
    by blast
qed

```

Now the sharper form.

```

lemma Krein_Milman_Minkowski_aux:
  fixes S :: 'a::euclidean_space set
  assumes n: dim S = n and S: compact S convex S 0 ∈ S
  shows 0 ∈ convex hull {x. x extreme_point_of S}
using n S
proof (induction n arbitrary: S rule: less_induct)
  case (less n S) show ?case
  proof (cases 0 ∈ rel_interior S)
    case True with Krein_Milman less.premis
    show ?thesis
      by (metis subsetD convex_convex_hull convex_rel_interior_closure rel_interior_subset)
  next
    case False
    have rel_interior S ≠ {}
      by (simp add: rel_interior_convex_nonempty_aux less)
    then obtain c where c: c ∈ rel_interior S by blast
    obtain a where a ≠ 0
      and le_ay:  $\bigwedge y. y \in S \implies a \cdot 0 \leq a \cdot y$ 
      and less_ay:  $\bigwedge y. y \in \text{rel\_interior } S \implies a \cdot 0 < a \cdot y$ 
      by (blast intro: supporting_hyperplane_rel_boundary intro!: less False)
    have face: S ∩ {x. a · x = 0} face_of S
      using face_of_Int_supporting_hyperplane_ge le_ay ⟨convex S⟩ by auto
    then have co: compact (S ∩ {x. a · x = 0}) convex (S ∩ {x. a · x = 0})
      using less.premis by (blast intro: face_of_imp_compact face_of_imp_convex)+
    have a · y = 0 if y ∈ span (S ∩ {x. a · x = 0}) for y
    proof -
      have y ∈ span {x. a · x = 0}
        by (metis inf.cobounded2 span_mono subsetCE that)
      then show ?thesis
        by (blast intro: span_induct [OF _ subspace_hyperplane])
    qed
    then have dim (S ∩ {x. a · x = 0}) < n
      by (metis (no_types) less_ay c subsetD dim_eq_span inf.strict_order_iff
        inf_le1 ⟨dim S = n⟩ not_le rel_interior_subset span_0 span_base)
    then have 0 ∈ convex hull {x. x extreme_point_of (S ∩ {x. a · x = 0})}

```

```

    by (rule less.IH) (auto simp: co less.prem)
  then show ?thesis
    by (metis (mono_tags, lifting) Collect_mono_iff face_extreme_point_of_face
        hull_mono subset_iff)
  qed
qed

```

theorem *Krein_Milman_Minkowski*:

```

  fixes S :: 'a::euclidean_space set
  assumes compact S convex S
  shows S = convex_hull {x. x extreme_point_of S}
proof
  show S ⊆ convex_hull {x. x extreme_point_of S}
  proof
    fix a assume [simp]: a ∈ S
    have 1: compact ((+) (- a) ' S)
    by (simp add: ⟨compact S⟩ compact_translation_subtract cong: image_cong_simp)
    have 2: convex ((+) (- a) ' S)
    by (simp add: ⟨convex S⟩ compact_translation_subtract)
    show a_inver: a ∈ convex_hull {x. x extreme_point_of S}
    using Krein_Milman_Minkowski_aux [OF refl 1 2]
        convex_hull_translation [of -a]
    by (auto simp: extreme_points_of_translation_subtract translation_assoc
        cong: image_cong_simp)
    qed
  next
    show convex_hull {x. x extreme_point_of S} ⊆ S
    using ⟨convex S⟩ extreme_point_of_stillconvex subset_hull by fastforce
  qed

```

9.24.8 Applying it to convex hulls of explicitly indicated finite sets

corollary *Krein_Milman_polytope*:

```

  fixes S :: 'a::euclidean_space set
  shows
    finite S
    ⇒ convex_hull S =
      convex_hull {x. x extreme_point_of (convex_hull S)}
  by (simp add: Krein_Milman_Minkowski finite_imp_compact_convex_hull)

```

lemma *extreme_points_of_convex_hull_eq*:

```

  fixes S :: 'a::euclidean_space set
  shows
    [[compact S; ∧ T. T ⊆ S ⇒ convex_hull T ≠ convex_hull S]]
    ⇒ {x. x extreme_point_of (convex_hull S)} = S
  by (metis (full_types) Krein_Milman_Minkowski compact_convex_hull
      convex_convex_hull extreme_points_of_convex_hull psubsetI)

```

```

lemma extreme_point_of_convex_hull_eq:
  fixes  $S :: 'a::euclidean\_space$  set
  shows
     $\llbracket compact\ S; \bigwedge T. T \subseteq S \implies convex\ hull\ T \neq convex\ hull\ S \rrbracket$ 
     $\implies (x\ extreme\_point\_of\ (convex\ hull\ S) \longleftrightarrow x \in S)$ 
using extreme_points_of_convex_hull_eq by auto

lemma extreme_point_of_convex_hull_convex_independent:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes compact S and S:  $\bigwedge a. a \in S \implies a \notin convex\ hull\ (S - \{a\})$ 
  shows  $(x\ extreme\_point\_of\ (convex\ hull\ S) \longleftrightarrow x \in S)$ 
proof -
  have  $convex\ hull\ T \neq convex\ hull\ S$  if  $T \subseteq S$  for  $T$ 
  proof -
    obtain  $a$  where  $T \subseteq S$   $a \in S$   $a \notin T$  using  $\langle T \subseteq S \rangle$  by blast
    then show ?thesis
    by (metis (full_types) Diff_eq_empty_iff Diff_insert0 S_hull_mono_hull_subset
insert_Diff_single subsetCE)
    qed
    then show ?thesis
    by (rule extreme_point_of_convex_hull_eq [OF  $\langle compact\ S \rangle$ ])
  qed

lemma extreme_point_of_convex_hull_affine_independent:
  fixes  $S :: 'a::euclidean\_space$  set
  shows
     $\neg affine\_dependent\ S$ 
     $\implies (x\ extreme\_point\_of\ (convex\ hull\ S) \longleftrightarrow x \in S)$ 
by (metis aff_independent_finite affine_dependent_def affine_hull_convex_hull
extreme_point_of_convex_hull_convex_independent_finite_imp_compact_hull_inc)

Elementary proofs exist, not requiring Euclidean spaces and all this development

lemma extreme_point_of_convex_hull_2:
  fixes  $x :: 'a::euclidean\_space$ 
  shows  $x\ extreme\_point\_of\ (convex\ hull\ \{a,b\}) \longleftrightarrow x = a \vee x = b$ 
  by (simp add: extreme_point_of_convex_hull_affine_independent)

lemma extreme_point_of_segment:
  fixes  $x :: 'a::euclidean\_space$ 
  shows  $x\ extreme\_point\_of\ closed\_segment\ a\ b \longleftrightarrow x = a \vee x = b$ 
  by (simp add: extreme_point_of_convex_hull_2 segment_convex_hull)

lemma face_of_convex_hull_subset:
  fixes  $S :: 'a::euclidean\_space$  set
  assumes compact S and T:  $T\ face\_of\ (convex\ hull\ S)$ 
  obtains  $S'$  where  $S' \subseteq S$   $T = convex\ hull\ S'$ 

```

proof

show $\{x. x \text{ extreme_point_of } T\} \subseteq S$
using $T \text{ extreme_point_of_convex_hull } \text{extreme_point_of_face}$ **by** *blast*
show $T = \text{convex hull } \{x. x \text{ extreme_point_of } T\}$
by (*metis Krein_Milman_Minkowski assms compact_convex_hull convex_convex_hull*
face_of_imp_compact face_of_imp_convex)

qed

lemma *face_of_convex_hull_aux*:

assumes $eq: x *_R p = u *_R a + v *_R b + w *_R c$
and $x: u + v + w = x \ x \neq 0$ **and** $S: \text{affine } S \ a \in S \ b \in S \ c \in S$
shows $p \in S$

proof –

have $p = (u *_R a + v *_R b + w *_R c) /_R x$
by (*metis* $\langle x \neq 0 \rangle$ *eq mult.commute right_inverse scaleR_one scaleR_scaleR*)
moreover **have** $\text{affine hull } \{a,b,c\} \subseteq S$
by (*simp add: S hull_minimal*)
moreover **have** $(u *_R a + v *_R b + w *_R c) /_R x \in \text{affine hull } \{a,b,c\}$
apply (*simp add: affine_hull_3*)
apply (*rule_tac x=u/x in exI*)
apply (*rule_tac x=v/x in exI*)
apply (*rule_tac x=w/x in exI*)
using x **apply** (*auto simp: field_split_simps*)
done

ultimately show *?thesis* **by** *force*

qed

proposition *face_of_convex_hull_insert_eq*:

fixes $a :: 'a :: \text{euclidean_space}$
assumes *finite S* **and** $a: a \notin \text{affine hull } S$
shows $(F \text{ face_of } (\text{convex hull } (\text{insert } a \ S))) \longleftrightarrow$
 $F \text{ face_of } (\text{convex hull } S) \vee$
 $(\exists F'. F' \text{ face_of } (\text{convex hull } S) \wedge F = \text{convex hull } (\text{insert } a \ F'))$
(is $F \text{ face_of } ?CAS \longleftrightarrow _)$

proof *safe*

assume $F: F \text{ face_of } ?CAS$

and $*$: $\nexists F'. F' \text{ face_of } \text{convex hull } S \wedge F = \text{convex hull } \text{insert } a \ F'$

obtain T **where** $T: T \subseteq \text{insert } a \ S$ **and** $F \text{ eq } T: F = \text{convex hull } T$

by (*metis* $F \langle \text{finite } S \rangle \text{ compact_insert_finite_imp_compact_face_of_convex_hull_subset}$)

show $F \text{ face_of } \text{convex hull } S$

proof (*cases* $a \in T$)

case *True*

have $F = \text{convex hull } \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$

proof

have $T \subseteq \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$

using $T \text{ hull_subset}$ **by** *fastforce*

then show $F \subseteq \text{convex hull } \text{insert } a \ (\text{convex hull } T \cap \text{convex hull } S)$

```

    by (simp add: FeqT hull_mono)
  show convex hull insert a (convex hull T  $\cap$  convex hull S)  $\subseteq$  F
    by (simp add: FeqT True hull_inc hull_minimal)
qed
moreover have convex hull T  $\cap$  convex hull S face_of convex hull S
  by (metis F FeqT convex_convex_hull face_of_slice hull_mono inf.absorb_iff2
subset_insertI)
ultimately show ?thesis
  using * by force
next
case False
then show ?thesis
  by (metis FeqT F T face_of_subset hull_mono subset_insert subset_insertI)
qed
next
assume F face_of convex hull S
show F face_of ?CAS
  by (simp add:  $\langle$ F face_of convex hull S $\rangle$  a face_of_convex_hull_insert  $\langle$ finite
S $\rangle$ )
next
fix F
assume F: F face_of convex hull S
show convex hull insert a F face_of ?CAS
proof (cases S = {})
case True
then show ?thesis
  using F face_of_affine_eq by auto
next
case False
have anote: a  $\notin$  convex hull S
  by (metis (no_types) a affine_hull_convex_hull hull_inc)
show ?thesis
proof (cases F = {})
case True show ?thesis
  using anote by (simp add:  $\langle$ F = {} $\rangle$   $\langle$ finite S $\rangle$  extreme_point_of_convex_hull_insert
face_of_singleton)
next
case False
have convex hull insert a F  $\subseteq$  ?CAS
  by (simp add: F a  $\langle$ finite S $\rangle$  convex_hull_subset face_of_convex_hull_insert
face_of_imp_subset hull_inc)
moreover
have  $(\exists y v. (1 - ub) *_R a + ub *_R b = (1 - v) *_R a + v *_R y \wedge$ 
 $0 \leq v \wedge v \leq 1 \wedge y \in F) \wedge$ 
 $(\exists x u. (1 - uc) *_R a + uc *_R c = (1 - u) *_R a + u *_R x \wedge$ 
 $0 \leq u \wedge u \leq 1 \wedge x \in F)$ 
if *:  $(1 - ux) *_R a + ux *_R x$ 
 $\in$  open_segment  $((1 - ub) *_R a + ub *_R b) ((1 - uc) *_R a + uc *_R$ 
c)

```

```

    and  $0 \leq ub$   $ub \leq 1$   $0 \leq uc$   $uc \leq 1$   $0 \leq ux$   $ux \leq 1$ 
    and  $b: b \in \text{convex hull } S$  and  $c: c \in \text{convex hull } S$  and  $x \in F$ 
  for  $b$   $c$   $ub$   $uc$   $ux$   $x$ 
  proof -
    have  $xah: x \in \text{affine hull } S$ 
    using  $F$   $\text{convex\_hull\_subset\_affine\_hull}$   $\text{face\_of\_imp\_subset}$   $\langle x \in F \rangle$  by
blast
    have  $ah: b \in \text{affine hull } S$   $c \in \text{affine hull } S$ 
    using  $b$   $c$   $\text{convex\_hull\_subset\_affine\_hull}$  by blast+
    obtain  $v$  where  $ne: (1 - ub) *_R a + ub *_R b \neq (1 - uc) *_R a + uc *_R c$ 
    and  $eq: (1 - ux) *_R a + ux *_R x =$ 
       $(1 - v) *_R ((1 - ub) *_R a + ub *_R b) + v *_R ((1 - uc) *_R a +$ 
 $uc *_R c)$ 
    and  $0 < v$   $v < 1$ 
    using * by (auto simp:  $\text{in\_segment}$ )
    then have  $0: ((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *_R a +$ 
       $(ux *_R x - (((1 - v) * ub) *_R b + (v * uc) *_R c)) = 0$ 
    by (auto simp:  $\text{algebra\_simps}$ )
    then have  $((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *_R a =$ 
       $((1 - v) * ub) *_R b + (v * uc) *_R c + (-ux) *_R x$ 
    by (auto simp:  $\text{algebra\_simps}$ )
    then have  $a \in \text{affine hull } S$  if  $1 - ux - ((1 - v) * (1 - ub) + v * (1 -$ 
 $uc)) \neq 0$ 
    by (rule  $\text{face\_of\_convex\_hull\_aux}$ ) (use  $b$   $c$   $xah$   $ah$  that in  $\langle \text{auto simp:}$ 
 $\text{algebra\_simps} \rangle$ )
    then have  $1 - ux - ((1 - v) * (1 - ub) + v * (1 - uc)) = 0$ 
    using  $a$  by blast
    with  $0$  have  $equx: (1 - v) * ub + v * uc = ux$ 
    and  $uxx: ux *_R x = (((1 - v) * ub) *_R b + (v * uc) *_R c)$ 
    by auto (auto simp:  $\text{algebra\_simps}$ )
    show ?thesis
    proof (cases  $uc = 0$ )
    case True
    then show ?thesis
    using  $equx$   $\langle 0 \leq ub \rangle$   $\langle ub \leq 1 \rangle$   $\langle v < 1 \rangle$   $uxx$   $\langle x \in F \rangle$  by force
    next
    case False
    show ?thesis
    proof (cases  $ub = 0$ )
    case True
    then show ?thesis
    using  $equx$   $\langle 0 \leq uc \rangle$   $\langle uc \leq 1 \rangle$   $\langle 0 < v \rangle$   $uxx$   $\langle x \in F \rangle$  by force
    next
    case False
    then have  $0 < ub$   $0 < uc$ 
    using  $\langle uc \neq 0 \rangle$   $\langle 0 \leq ub \rangle$   $\langle 0 \leq uc \rangle$  by auto
    then have  $(1 - v) * ub > 0$   $v * uc > 0$ 
    by ( $\text{simp\_all add:}$   $\langle 0 < uc \rangle$   $\langle 0 < v \rangle$   $\langle v < 1 \rangle$ )
    then have  $ux \neq 0$ 

```

```

    using equx  $\langle 0 < v \rangle$  by auto
  have  $b \in F \wedge c \in F$ 
  proof (cases  $b = c$ )
    case True
    then show ?thesis
    by (metis  $\langle ux \neq 0 \rangle$  equx real_vector.scale_cancel_left scaleR_add_left
    uxx  $\langle x \in F \rangle$ )
  next
    case False
    have  $x = (((1 - v) * ub) *_R b + (v * uc) *_R c) /_R ux$ 
    by (metis  $\langle ux \neq 0 \rangle$  uxx mult.commute right_inverse scaleR_one
    scaleR_scaleR)
    also have  $\dots = (1 - v * uc / ux) *_R b + (v * uc / ux) *_R c$ 
    using  $\langle ux \neq 0 \rangle$  equx apply (auto simp: field_split_simps)
    by (metis add.commute add_diff_eq add_divide_distrib diff_add_cancel
    scaleR_add_left)
    finally have  $x = (1 - v * uc / ux) *_R b + (v * uc / ux) *_R c$  .
    then have  $x \in \text{open\_segment } b \ c$ 
    apply (simp add: in_segment  $\langle b \neq c \rangle$ )
    apply (rule_tac  $x=(v * uc) / ux$  in exI)
    using  $\langle 0 \leq ux \rangle$   $\langle ux \neq 0 \rangle$   $\langle 0 < uc \rangle$   $\langle 0 < v \rangle$   $\langle 0 < ub \rangle$   $\langle v < 1 \rangle$  equx
    apply (force simp: field_split_simps)
    done
    then show ?thesis
    by (rule face_ofD [OF F  $\_ b \ c \ \langle x \in F \rangle$ ])
  qed
  with  $\langle 0 \leq ub \rangle$   $\langle ub \leq 1 \rangle$   $\langle 0 \leq uc \rangle$   $\langle uc \leq 1 \rangle$  show ?thesis by blast
qed
qed
qed
moreover have convex hull  $F = F$ 
  by (meson  $F$  convex_hull_eq face_of_imp_convex)
ultimately show ?thesis
  unfolding face_of_def by (fastforce simp: convex_hull_insert_alt  $\langle S \neq \{\} \rangle$ 
 $\langle F \neq \{\} \rangle$ )
qed
qed
qed

```

lemma *face_of_convex_hull_insert2*:

```

  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  assumes  $S$ : finite  $S$  and  $a$ :  $a \notin \text{affine hull } S$  and  $F$ :  $F$  face_of convex hull  $S$ 
  shows convex hull (insert  $a$   $F$ ) face_of convex hull (insert  $a$   $S$ )
  by (metis  $F$  face_of_convex_hull_insert_eq [OF S a])

```

proposition *face_of_convex_hull_affine_independent*:

```

  fixes  $S :: 'a :: \text{euclidean\_space}$  set
  assumes  $\neg$  affine_dependent  $S$ 
  shows ( $T$  face_of (convex hull  $S$ ))  $\longleftrightarrow$  ( $\exists c. c \subseteq S \wedge T = \text{convex hull } c$ )

```

3452

(is ?lhs = ?rhs)

proof

assume ?lhs

then show ?rhs

by (meson <T face_of convex hull S> aff_independent_finite assms face_of_convex_hull_subset finite_imp_compact)

next

assume ?rhs

then obtain C **where** $C \subseteq S$ **and** T: $T = \text{convex hull } C$

by blast

have affine_hull C \cap affine_hull (S - C) = {}

by (intro disjoint_affine_hull [OF assms <C \subseteq S>], auto)

then have affine_hull C \cap convex_hull (S - C) = {}

using convex_hull_subset_affine_hull **by** fastforce

then show ?lhs

by (metis face_of_convex_hulls <C \subseteq S> aff_independent_finite assms T)

qed

lemma facet_of_convex_hull_affine_independent:

fixes S :: 'a::euclidean_space set

assumes \neg affine_dependent S

shows T facet_of (convex_hull S) \longleftrightarrow

$T \neq \{\}$ \wedge ($\exists u. u \in S \wedge T = \text{convex hull } (S - \{u\})$)

(is ?lhs = ?rhs)

proof

assume ?lhs

then have T face_of (convex_hull S) $T \neq \{\}$

and afft: aff_dim T = aff_dim (convex_hull S) - 1

by (auto simp: facet_of_def)

then obtain c **where** $c \subseteq S$ **and** c: $T = \text{convex hull } c$

by (auto simp: face_of_convex_hull_affine_independent [OF assms])

then have affs: aff_dim S = aff_dim c + 1

by (metis aff_dim_convex_hull afft eq_diff_eq)

have \neg affine_dependent c

using <c \subseteq S> affine_dependent_subset assms **by** blast

with affs **have** card (S - c) = 1

by (smt (verit) <c \subseteq S> aff_dim_affine_independent aff_independent_finite assms card_Diff_subset

 card_mono of_nat_diff of_nat_eq_1_iff)

then obtain u **where** u: $u \in S - c$

by (metis DiffI <c \subseteq S> aff_independent_finite assms cancel_comm_monoid_add_class.diff_cancel card_Diff_subset subsetI subset_antisym zero_neq_one)

then have u: $S = \text{insert } u \ c$

by (metis Diff_subset <c \subseteq S> <card (S - c) = 1> card_1_singletonE double_diff_insert_Diff_insert_subset_singletonD)

have T = convex_hull (c - {u})

by (metis Diff_empty Diff_insert0 <T facet_of convex_hull S> c facet_of_irrefl insert_absorb u)

with <T $\neq \{\}$ > **show** ?rhs


```

    using c u by auto
next
assume ?rhs
then obtain u where  $T \neq \{\}$   $u \in S$  and  $u: T = \text{convex hull } (S - \{u\})$ 
  by (force simp: facet_of_def)
then have  $\neg S \subseteq \{u\}$ 
  using  $\langle T \neq \{\} \rangle u$  by auto
have  $\text{aff\_dim } (S - \{u\}) = \text{aff\_dim } S - 1$ 
  using  $\text{assms } \langle u \in S \rangle$ 
  unfolding affine_dependent_def
  by (metis add_diff_cancel_right' aff_dim_insert insert_Diff [of u S])
then have  $\text{aff\_dim } (\text{convex hull } (S - \{u\})) = \text{aff\_dim } (\text{convex hull } S) - 1$ 
  by (simp add: aff_dim_convex_hull)
then show ?lhs
  by (metis Diff_subset  $\langle T \neq \{\} \rangle$   $\text{assms face\_of\_convex\_hull\_affine\_independent}$ 
    facet_of_def u)
qed

```

lemma *facet_of_convex_hull_affine_independent_alt:*

```

  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes  $\neg \text{affine\_dependent } S$ 
  shows  $(T \text{ facet\_of } (\text{convex hull } S) \iff 2 \leq \text{card } S \wedge (\exists u. u \in S \wedge T = \text{convex hull } (S - \{u\})))$ 
    (is ?lhs = ?rhs)

```

proof

```

  assume L: ?lhs
  then obtain x where
     $x \in S$  and  $x: T = \text{convex hull } (S - \{x\})$  and  $\text{finite } S$ 
  using  $\text{assms facet\_of\_convex\_hull\_affine\_independent aff\_independent\_finite}$ 
  by blast
  moreover have  $\text{Suc } (\text{Suc } 0) \leq \text{card } S$ 
    using L  $x \langle x \in S \rangle \langle \text{finite } S \rangle$ 
    by (metis Suc_leI  $\text{assms card.remove convex\_hull\_eq\_empty card\_gt\_0\_iff}$ 
      facet_of_convex_hull_affine_independent finite_Diff not_less_eq_eq)
  ultimately show ?rhs
    by auto
next
assume ?rhs then show ?lhs
  using  $\text{assms}$ 
  by (auto simp: facet_of_convex_hull_affine_independent Set.subset_singleton_iff)
qed

```

lemma *segment_face_of:*

```

  assumes  $(\text{closed\_segment } a b) \text{ face\_of } S$ 
  shows  $a \text{ extreme\_point\_of } S \wedge b \text{ extreme\_point\_of } S$ 

```

proof –

```

  have  $as: \{a\} \text{ face\_of } S$ 
  by (metis (no_types)  $\text{assms convex\_hull\_singleton empty\_iff extreme\_point\_of\_convex\_hull\_insert}$ 
    face_of_face face_of_singleton finite.emptyI finite.insertI insert_absorb insert_iff)

```

```

segment_convex_hull)
  moreover have {b} face_of S
  proof -
    have  $b \in \text{convex hull } \{a\} \vee b \text{ extreme\_point\_of convex hull } \{b, a\}$ 
      by (meson extreme_point_of_convex_hull_insert finite.emptyI finite.insertI)
    moreover have  $\text{closed\_segment } a \ b = \text{convex hull } \{b, a\}$ 
      using closed_segment_commute segment_convex_hull by blast
    ultimately show ?thesis
      by (metis as_assms face_of_face convex_hull_singleton_empty_iff_face_of_singleton
insertE)
    qed
  ultimately show  $a \text{ extreme\_point\_of } S \ b \text{ extreme\_point\_of } S$ 
    using face_of_singleton by blast+
  qed

```

proposition *Krein_Milman_frontier*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes  $\text{convex } S \ \text{compact } S$ 
shows  $S = \text{convex hull } (\text{frontier } S)$ 
(is ?lhs = ?rhs)

```

proof

```

have ?lhs  $\subseteq \text{convex hull } \{x. x \text{ extreme\_point\_of } S\}$ 
  using Krein_Milman_Minkowski_assms by blast
also have  $\dots \subseteq ?rhs$ 
  proof (rule hull_mono)
    show  $\{x. x \text{ extreme\_point\_of } S\} \subseteq \text{frontier } S$ 
      using closure_subset
    by (auto simp: frontier_def extreme_point_not_in_interior extreme_point_of_def)
  qed
finally show ?lhs  $\subseteq ?rhs$  .

```

next

```

have ?rhs  $\subseteq \text{convex hull } S$ 
  by (metis Diff_subset <compact S> closure_closed compact_eq_bounded_closed
frontier_def hull_mono)
also have  $\dots \subseteq ?lhs$ 
  by (simp add: <convex S> hull_same)
finally show ?rhs  $\subseteq ?lhs$  .
qed

```

9.24.9 Polytopes

definition *polytope where*

$\text{polytope } S \equiv \exists v. \text{finite } v \wedge S = \text{convex hull } v$

lemma *polytope_translation_eq*: $\text{polytope } ((+) a \ 'S) \longleftrightarrow \text{polytope } S$

unfolding *polytope_def*

```

by (metis (no_types, opaque_lifting) add.left_inverse convex_hull_translation
finite_imageI image_add_0 translation_assoc)

```

lemma *polytope_linear_image*: $\llbracket \text{linear } f; \text{ polytope } p \rrbracket \implies \text{polytope}(\text{image } f p)$
unfolding *polytope_def* **using** *convex_hull_linear_image* **by** *blast*

lemma *polytope_empty*: *polytope* $\{\}$
using *convex_hull_empty* *polytope_def* **by** *blast*

lemma *polytope_convex_hull*: *finite* $S \implies \text{polytope}(\text{convex hull } S)$
using *polytope_def* **by** *auto*

lemma *polytope_Times*: $\llbracket \text{polytope } S; \text{ polytope } T \rrbracket \implies \text{polytope}(S \times T)$
unfolding *polytope_def*
by (*metis finite_cartesian_product convex_hull_Times*)

lemma *face_of_polytope_polytope*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
shows $\llbracket \text{polytope } S; F \text{ face_of } S \rrbracket \implies \text{polytope } F$
unfolding *polytope_def*
by (*meson face_of_convex_hull_subset finite_imp_compact finite_subset*)

lemma *finite_polytope_faces*:
fixes $S :: 'a::\text{euclidean_space}$ *set*
assumes *polytope* S
shows *finite* $\{F. F \text{ face_of } S\}$
proof –
obtain v **where** *finite* $v \ S = \text{convex hull } v$
using *assms polytope_def* **by** *auto*
have *finite* $((\text{hull}) \text{ convex } \{T. T \subseteq v\})$
by (*simp add: finite_v*)
moreover **have** $\{F. F \text{ face_of } S\} \subseteq ((\text{hull}) \text{ convex } \{T. T \subseteq v\})$
by (*metis (no_types, lifting) finite_v S = convex_hull_v face_of_convex_hull_subset finite_imp_compact image_eqI mem_Collect_eq subsetI*)
ultimately show *?thesis*
by (*blast intro: finite_subset*)
qed

lemma *finite_polytope_facets*:
assumes *polytope* S
shows *finite* $\{T. T \text{ facet_of } S\}$
by (*simp add: assms facet_of_def finite_polytope_faces*)

lemma *polytope_scaling*:
assumes *polytope* S **shows** *polytope* $(\text{image } (\lambda x. c *_{\mathbb{R}} x) S)$
by (*simp add: assms polytope_linear_image*)

lemma *polytope_imp_compact*:
fixes $S :: 'a::\text{real_normed_vector}$ *set*
shows *polytope* $S \implies \text{compact } S$
by (*metis finite_imp_compact_convex_hull polytope_def*)

lemma *polytope_imp_convex*: $\text{polytope } S \implies \text{convex } S$
by (*metis convex_convex_hull polytope_def*)

lemma *polytope_imp_closed*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{polytope } S \implies \text{closed } S$
by (*simp add: compact_imp_closed polytope_imp_compact*)

lemma *polytope_imp_bounded*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{polytope } S \implies \text{bounded } S$
by (*simp add: compact_imp_bounded polytope_imp_compact*)

lemma *polytope_interval*: $\text{polytope}(\text{cbox } a \ b)$
unfolding *polytope_def* **by** (*meson closed_interval_as_convex_hull*)

lemma *polytope_sing*: $\text{polytope } \{a\}$
using *polytope_def* **by** *force*

lemma *face_of_polytope_insert*:
 $\llbracket \text{polytope } S; a \notin \text{affine hull } S; F \text{ face_of } S \rrbracket \implies F \text{ face_of convex hull } (\text{insert } a \ S)$
by (*metis (no_types, lifting) affine_hull_convex_hull face_of_convex_hull_insert hull_insert polytope_def*)

proposition *face_of_polytope_insert2*:
fixes $a :: 'a :: \text{euclidean_space}$
assumes $\text{polytope } S \ a \notin \text{affine hull } S \ F \text{ face_of } S$
shows $\text{convex hull } (\text{insert } a \ F) \text{ face_of convex hull } (\text{insert } a \ S)$
proof –
obtain V **where** $\text{finite } V \ S = \text{convex hull } V$
using *assms* **by** (*auto simp: polytope_def*)
then have $\text{convex hull } (\text{insert } a \ F) \text{ face_of convex hull } (\text{insert } a \ V)$
using *affine_hull_convex_hull assms face_of_convex_hull_insert2* **by** *blast*
then show *?thesis*
by (*metis ⟨S = convex hull V⟩ hull_insert*)
qed

9.24.10 Polyhedra

definition *polyhedron* **where**

$$\begin{aligned} \text{polyhedron } S &\equiv \\ &\exists F. \text{finite } F \wedge \\ &S = \bigcap F \wedge \\ &(\forall h \in F. \exists a \ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}) \end{aligned}$$

lemma *polyhedron_Int* [*intro,simp*]:
 $\llbracket \text{polyhedron } S; \text{polyhedron } T \rrbracket \implies \text{polyhedron } (S \cap T)$

```

apply (clarsimp simp add: polyhedron_def)
subgoal for  $F\ G$ 
  by (rule_tac  $x=F \cup G$  in exI, auto)
done

```

```

lemma polyhedron_UNIV [iff]: polyhedron UNIV
  using polyhedron_def by auto

```

```

lemma polyhedron_Inter [intro,simp]:
   $\llbracket \text{finite } F; \bigwedge S. S \in F \implies \text{polyhedron } S \rrbracket \implies \text{polyhedron}(\bigcap F)$ 
  by (induction F rule: finite_induct) auto

```

```

lemma polyhedron_empty [iff]: polyhedron ( $\{\} :: 'a :: \text{euclidean\_space set}$ )
proof -
  define  $i::'a$  where ( $i \equiv \text{SOME } i. i \in \text{Basis}$ )
  have  $\exists a. a \neq 0 \wedge (\exists b. \{x. i \cdot x \leq -1\} = \{x. a \cdot x \leq b\})$ 
    by (rule_tac  $x=i$  in exI) (force simp: i_def SOME_Basis nonzero_Basis)
  moreover have  $\exists a b. a \neq 0 \wedge \{x. -i \cdot x \leq -1\} = \{x. a \cdot x \leq b\}$ 
    by (metis Basis_zero SOME_Basis i_def neg_0_equal_iff_equal)
  ultimately show ?thesis
    unfolding polyhedron_def
    by (rule_tac  $x=\{\{x. i \cdot x \leq -1\}, \{x. -i \cdot x \leq -1\}\}$  in exI) force
qed

```

```

lemma polyhedron_halfspace_le:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows polyhedron  $\{x. a \cdot x \leq b\}$ 
proof (cases  $a = 0$ )
  case True then show ?thesis by auto
next
  case False
  then show ?thesis
    unfolding polyhedron_def
    by (rule_tac  $x=\{\{x. a \cdot x \leq b\}\}$  in exI) auto
qed

```

```

lemma polyhedron_halfspace_ge:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows polyhedron  $\{x. a \cdot x \geq b\}$ 
  using polyhedron_halfspace_le [of  $-a -b$ ] by simp

```

```

lemma polyhedron_hyperplane:
  fixes  $a :: 'a :: \text{euclidean\_space}$ 
  shows polyhedron  $\{x. a \cdot x = b\}$ 
proof -
  have  $\{x. a \cdot x = b\} = \{x. a \cdot x \leq b\} \cap \{x. a \cdot x \geq b\}$ 
    by force
  then show ?thesis

```

by (simp add: polyhedron_halfspace_ge polyhedron_halfspace_le)
qed

lemma *affine_imp_polyhedron*:
fixes $S :: 'a :: euclidean_space\ set$
shows $affine\ S \implies polyhedron\ S$
by (metis affine_hull_finite_intersection_hyperplanes hull_same polyhedron_Inter polyhedron_hyperplane)

lemma *polyhedron_imp_closed*:
fixes $S :: 'a :: euclidean_space\ set$
shows $polyhedron\ S \implies closed\ S$
by (metis closed_Inter closed_halfspace_le polyhedron_def)

lemma *polyhedron_imp_convex*:
fixes $S :: 'a :: euclidean_space\ set$
shows $polyhedron\ S \implies convex\ S$
by (metis convex_Inter convex_halfspace_le polyhedron_def)

lemma *polyhedron_affine_hull*:
fixes $S :: 'a :: euclidean_space\ set$
shows $polyhedron(affine\ hull\ S)$
by (simp add: affine_imp_polyhedron)

9.24.11 Canonical polyhedron representation making facial structure explicit

proposition *polyhedron_Int_affine*:
fixes $S :: 'a :: euclidean_space\ set$
shows $polyhedron\ S \longleftrightarrow$
 $(\exists F. finite\ F \wedge S = (affine\ hull\ S) \cap \bigcap F \wedge$
 $(\forall h \in F. \exists a\ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}))$
by (metis hull_subset_inf.absorb_iff2 polyhedron_Int polyhedron_affine_hull polyhedron_def)

proposition *rel_interior_polyhedron_explicit*:
assumes $finite\ F$
and seq: $S = affine\ hull\ S \cap \bigcap F$
and faceq: $\bigwedge h. h \in F \implies a\ h \neq 0 \wedge h = \{x. a\ h \cdot x \leq b\ h\}$
and psub: $\bigwedge F'. F' \subset F \implies S \subset affine\ hull\ S \cap \bigcap F'$
shows $rel_interior\ S = \{x \in S. \forall h \in F. a\ h \cdot x < b\ h\}$

proof –

have rels: $\bigwedge x. x \in rel_interior\ S \implies x \in S$

by (meson IntE mem_rel_interior)

moreover have $a\ i \cdot x < b\ i$ if $x \in rel_interior\ S$ and $i \in F$ for $x\ i$

proof –

have fif: $F - \{i\} \subset F$

using $\langle i \in F \rangle Diff_insert_absorb\ Diff_subset\ set_insert\ psubsetI$ by blast

then have $S \subset affine\ hull\ S \cap \bigcap (F - \{i\})$

```

    by (rule psub)
  then obtain z where ssub:  $S \subseteq \bigcap (F - \{i\})$  and zint:  $z \in \bigcap (F - \{i\})$ 
    and  $z \notin S$  and zaff:  $z \in \text{affine hull } S$ 
  by auto
  have  $z \neq x$ 
    using  $\langle z \notin S \rangle$  rels x by blast
  have  $z \notin \text{affine hull } S \cap \bigcap F$ 
    using  $\langle z \notin S \rangle$  seq by auto
  then have aiz:  $a \cdot i \cdot z > b \cdot i$ 
    using faceq zint zaff by fastforce
  obtain e where  $e > 0$   $x \in S$  and e:  $\text{ball } x \ e \cap \text{affine hull } S \subseteq S$ 
    using x by (auto simp: mem_rel_interior_ball)
  then have ins:  $\bigwedge y. [\text{norm } (x - y) < e; y \in \text{affine hull } S] \implies y \in S$ 
    by (metis IntI subsetD dist_norm mem_ball)
  define  $\xi$  where  $\xi = \min (1/2) (e / 2 / \text{norm}(z - x))$ 
  have  $\text{norm } (\xi *_{\mathbb{R}} x - \xi *_{\mathbb{R}} z) = \text{norm } (\xi *_{\mathbb{R}} (x - z))$ 
    by (simp add:  $\xi\_def$  algebra_simps norm_mult)
  also have  $\dots = \xi * \text{norm } (x - z)$ 
    using  $\langle e > 0 \rangle$  by (simp add:  $\xi\_def$ )
  also have  $\dots < e$ 
    using  $\langle z \neq x \rangle \langle e > 0 \rangle$  by (simp add:  $\xi\_def$  min_def field_split_simps
norm_minus_commute)
  finally have lte:  $\text{norm } (\xi *_{\mathbb{R}} x - \xi *_{\mathbb{R}} z) < e$  .
  have  $\xi\_aff: \xi *_{\mathbb{R}} z + (1 - \xi) *_{\mathbb{R}} x \in \text{affine hull } S$ 
    by (simp add:  $\langle x \in S \rangle$  hull_inc mem_affine zaff)
  have  $\xi *_{\mathbb{R}} z + (1 - \xi) *_{\mathbb{R}} x \in S$ 
    using ins [ $OF \ \xi\_aff$ ] by (simp add: algebra_simps lte)
  then obtain l where  $l: 0 < l < 1$  and ls:  $(l *_{\mathbb{R}} z + (1 - l) *_{\mathbb{R}} x) \in S$ 
    using  $\langle e > 0 \rangle \langle z \neq x \rangle$ 
    by (rule_tac  $l = \xi$  in that) (auto simp:  $\xi\_def$ )
  then have i:  $l *_{\mathbb{R}} z + (1 - l) *_{\mathbb{R}} x \in i$ 
    using seq  $\langle i \in F \rangle$  by auto
  have  $b \cdot i \cdot l + (a \cdot i \cdot x) * (1 - l) < a \cdot i \cdot (l *_{\mathbb{R}} z + (1 - l) *_{\mathbb{R}} x)$ 
    using l by (simp add: algebra_simps aiz)
  also have  $\dots \leq b \cdot i$  using i l
    using faceq mem_Collect_eq  $\langle i \in F \rangle$  by blast
  finally have  $(a \cdot i \cdot x) * (1 - l) < b \cdot i * (1 - l)$ 
    by (simp add: algebra_simps)
  with l show ?thesis
    by simp
qed
moreover have  $x \in \text{rel\_interior } S$ 
  if  $x \in S$  and less:  $\bigwedge h. h \in F \implies a \cdot h \cdot x < b \cdot h$  for x
proof -
  have 1:  $\bigwedge h. h \in F \implies x \in \text{interior } h$ 
    by (metis interior_halfspace_le mem_Collect_eq less faceq)
  have 2:  $\bigwedge y. [\bigwedge h \in F. y \in \text{interior } h; y \in \text{affine hull } S] \implies y \in S$ 
    by (metis IntI Inter_iff subsetD interior_subset seq)
  show ?thesis

```

```

apply (simp add: rel_interior ⟨x ∈ S⟩)
apply (rule_tac x=⋂ h∈F. interior h in exI)
apply (auto simp: ⟨finite F⟩ open_INT 1 2)
done
qed
ultimately show ?thesis by blast
qed

lemma polyhedron_Int_affine_parallel:
fixes S :: 'a :: euclidean_space set
shows polyhedron S ↔
  (∃ F. finite F ∧
    S = (affine hull S) ∩ (⋂ F) ∧
    (∀ h ∈ F. ∃ a b. a ≠ 0 ∧ h = {x. a · x ≤ b} ∧
      (∀ x ∈ affine hull S. (x + a) ∈ affine hull S)))
(is ?lhs = ?rhs)
proof
assume ?lhs
then obtain F where finite F and seq: S = (affine hull S) ∩ ⋂ F
and faces: ⋀ h. h ∈ F ⟹ ∃ a b. a ≠ 0 ∧ h = {x. a · x ≤ b}
by (fastforce simp add: polyhedron_Int_affine)
then obtain a b where ab: ⋀ h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}
by metis
show ?rhs
proof -
have ∃ a' b'. a' ≠ 0 ∧
affine hull S ∩ {x. a' · x ≤ b'} = affine hull S ∩ h ∧
(∀ w ∈ affine hull S. (w + a') ∈ affine hull S)
if h ∈ F ¬(affine hull S ⊆ h) for h
proof -
have a h ≠ 0 and h = {x. a h · x ≤ b h} h ∩ ⋂ F = ⋂ F
using ⟨h ∈ F⟩ ab by auto
then have (affine hull S) ∩ {x. a h · x ≤ b h} ≠ {}
by (metis affine_hull_eq_empty inf.absorb_iff1 inf_assoc inf_bot_left seq
that(2))
moreover have ¬(affine hull S ⊆ {x. a h · x ≤ b h})
using ⟨h = {x. a h · x ≤ b h}⟩ that(2) by blast
ultimately show ?thesis
using affine_parallel_slice [of affine hull S]
by (metis ⟨h = {x. a h · x ≤ b h}⟩ affine_affine_hull)
qed
then obtain a b
where ab: ⋀ h. [h ∈ F; ¬(affine hull S ⊆ h)]
⟹ a h ≠ 0 ∧
affine hull S ∩ {x. a h · x ≤ b h} = affine hull S ∩ h ∧
(∀ w ∈ affine hull S. (w + a h) ∈ affine hull S)
by metis
let ?F = (λh. {x. a h · x ≤ b h}) ‘ {h ∈ F. ¬ affine hull S ⊆ h}

```



```

show ?thesis
proof (intro exI conjI)
  show finite ?F
    using ⟨finite F⟩ by force
  show  $S = \text{affine hull } S \cap \bigcap ?F$ 
    by (subst seq) (auto simp: ab INT_extend_simps)
qed (use ab in blast)
qed
next
  assume ?rhs then show ?lhs
    by (metis polyhedron_Int_affine)
qed

proposition polyhedron_Int_affine_parallel_minimal:
  fixes  $S :: 'a :: \text{euclidean\_space set}$ 
  shows  $\text{polyhedron } S \longleftrightarrow$ 
     $(\exists F. \text{finite } F \wedge$ 
       $S = (\text{affine hull } S) \cap (\bigcap F) \wedge$ 
       $(\forall h \in F. \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\} \wedge$ 
         $(\forall x \in \text{affine hull } S. (x + a) \in \text{affine hull } S)) \wedge$ 
       $(\forall F'. F' \subset F \longrightarrow S \subset (\text{affine hull } S) \cap (\bigcap F')))$ 
    (is ?lhs = ?rhs)

proof
  assume ?lhs
  then obtain f0
    where f0: finite f0
       $S = (\text{affine hull } S) \cap (\bigcap f0)$ 
      (is ?P f0)
       $\forall h \in f0. \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\} \wedge$ 
         $(\forall x \in \text{affine hull } S. (x + a) \in \text{affine hull } S)$ 
      (is ?Q f0)
    by (force simp: polyhedron_Int_affine_parallel)
  define n where  $n = (\text{LEAST } n. \exists F. \text{card } F = n \wedge \text{finite } F \wedge ?P F \wedge ?Q F)$ 
  have nf:  $\exists F. \text{card } F = n \wedge \text{finite } F \wedge ?P F \wedge ?Q F$ 
    apply (simp add: n_def)
    apply (rule LeastI [where k = card f0])
    using f0 apply auto
  done
  then obtain F where  $F: \text{card } F = n$  finite F and seq: ?P F and aff: ?Q F
    by blast
  then have  $\neg (\text{finite } g \wedge ?P g \wedge ?Q g)$  if  $\text{card } g < n$  for g
    using that by (auto simp: n_def dest!: not_less_Least)
  then have *:  $\neg (?P g \wedge ?Q g)$  if  $g \subset F$  for g
    using that ⟨finite F⟩ psubset_card_mono ⟨card F = n⟩
    by (metis finite_Int_inf.strict_order_iff)
  have 1:  $\bigwedge F'. F' \subset F \implies S \subseteq \text{affine hull } S \cap \bigcap F'$ 
    by (subst seq) blast
  have 2:  $S \neq \text{affine hull } S \cap \bigcap F'$  if  $F' \subset F$  for F'

```

3462

```

    using * [OF that] by (metis IntE aff inf.strict_order_iff that)
  show ?rhs
    by (metis ‹finite F› seq aff psubsetI 1 2)
next
  assume ?rhs then show ?lhs
    by (auto simp: polyhedron_Int_affine_parallel)
qed

```

lemma *polyhedron_Int_affine_minimal*:

fixes $S :: 'a :: euclidean_space$ set

shows $polyhedron\ S \longleftrightarrow$

$$\begin{aligned}
 & (\exists F. \text{finite } F \wedge S = (\text{affine hull } S) \cap \bigcap F \wedge \\
 & \quad (\forall h \in F. \exists a\ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}) \wedge \\
 & \quad (\forall F'. F' \subset F \longrightarrow S \subset (\text{affine hull } S) \cap \bigcap F'))
 \end{aligned}$$

by (*metis polyhedron_Int_affine polyhedron_Int_affine_parallel_minimal*)

proposition *facet_of_polyhedron_explicit*:

assumes *finite F*

and *seq*: $S = \text{affine hull } S \cap \bigcap F$

and *faceq*: $\bigwedge h. h \in F \implies a\ h \neq 0 \wedge h = \{x. a\ h \cdot x \leq b\ h\}$

and *psub*: $\bigwedge F'. F' \subset F \implies S \subset \text{affine hull } S \cap \bigcap F'$

shows C *facet_of S* $\longleftrightarrow (\exists h. h \in F \wedge C = S \cap \{x. a\ h \cdot x = b\ h\})$

proof (*cases S = {}*)

case *True* **with** *psub* **show** *?thesis* **by** *force*

next

case *False*

have *polyhedron S*

unfolding *polyhedron_Int_affine* **by** (*metis ‹finite F› faceq seq*)

then **have** *convex S*

by (*rule polyhedron_imp_convex*)

with *False rel_interior_eq_empty* **have** *rel_interior S ≠ {}* **by** *blast*

then **obtain** x **where** $x \in \text{rel_interior } S$ **by** *auto*

then **obtain** T **where** *open T* $x \in T$ $x \in S$ $T \cap \text{affine hull } S \subseteq S$

by (*force simp: mem_rel_interior*)

then **have** *zaff*: $x \in \text{affine hull } S$ **and** *zint*: $x \in \bigcap F$

using *seq hull_inc* **by** *auto*

have *rel_interior S* $= \{x \in S. \forall h \in F. a\ h \cdot x < b\ h\}$

by (*rule rel_interior_polyhedron_explicit [OF ‹finite F› seq faceq psub]*)

with $\langle x \in \text{rel_interior } S \rangle$

have [*simp*]: $\bigwedge h. h \in F \implies a\ h \cdot x < b\ h$ **by** *blast*

have *: $(S \cap \{x. a\ h \cdot x = b\ h\})$ *facet_of S* **if** $h \in F$ **for** h

proof –

have $S \subset \text{affine hull } S \cap \bigcap (F - \{h\})$

using *psub that* **by** (*metis Diff_disjoint Diff_subset insert_disjoint(2) psubsetI*)

then **obtain** z **where** *zaff*: $z \in \text{affine hull } S$ **and** *zint*: $z \in \bigcap (F - \{h\})$ **and** $z \notin S$

by *force*

```

then have  $z \neq x \wedge z \notin h$  using seq  $\langle x \in S \rangle$  by auto
have  $x \in h$  using that xint by auto
then have able:  $a h \cdot x \leq b h$ 
  using faceq that by blast
also have  $\dots < a h \cdot z$  using  $\langle z \notin h \rangle$  faceq [OF that] xint by auto
finally have xltz:  $a h \cdot x < a h \cdot z$  .
define l where  $l = (b h - a h \cdot x) / (a h \cdot z - a h \cdot x)$ 
define w where  $w = (1 - l) *_R x + l *_R z$ 
have  $0 < l < 1$ 
  using able xltz  $\langle b h < a h \cdot z \rangle \langle h \in F \rangle$ 
  by (auto simp: l_def field_split_simps)
have awlt:  $a i \cdot w < b i$  if  $i \in F$   $i \neq h$  for i
proof -
  have  $(1 - l) * (a i \cdot x) < (1 - l) * b i$ 
    by (simp add:  $\langle l < 1 \rangle \langle i \in F \rangle$ )
  moreover have  $l * (a i \cdot z) \leq l * b i$ 
  proof (rule mult_left_mono)
    show  $a i \cdot z \leq b i$ 
      by (metis DiffI Inter_iff empty_iff faceq insertE mem_Collect_eq that
zint)
  qed (use  $\langle 0 < l \rangle$  in auto)
  ultimately show ?thesis by (simp add: w_def algebra_simps)
qed
have weq:  $a h \cdot w = b h$ 
  using xltz unfolding w_def l_def
  by (simp add: algebra_simps) (simp add: field_simps)
let ?F =  $\{x. a h \cdot x = b h\}$ 
have faceS:  $S \cap ?F$  face_of S
proof (rule face_of_Int_supporting_hyperplane_le)
  show  $\bigwedge x. x \in S \implies a h \cdot x \leq b h$ 
    using faceq seq that by fastforce
qed fact
have  $w \in \text{affine hull } S$ 
  by (simp add: w_def mem_affine xaff zaff)
moreover have  $w \in \bigcap F$ 
  using  $\langle a h \cdot w = b h \rangle$  awlt faceq less_eq_real_def by blast
ultimately have  $w \in S$ 
  using seq by blast
with weq have ne:  $S \cap ?F \neq \{\}$  by blast
moreover have  $\text{affine hull } (S \cap ?F) = (\text{affine hull } S) \cap ?F$ 
proof
  show  $\text{affine hull } (S \cap ?F) \subseteq \text{affine hull } S \cap ?F$ 
  proof -
    have  $\text{affine hull } (S \cap ?F) \subseteq \text{affine hull } S$ 
      by (simp add: hull_mono)
    then show ?thesis
      by (simp add: affine_hyperplane subset_hull)
  qed
qed
next

```

```

show affine hull  $S \cap ?F \subseteq$  affine hull  $(S \cap ?F)$ 
proof
  fix  $y$ 
  assume  $y_{\text{aff}}: y \in$  affine hull  $S \cap \{y. a \cdot h \cdot y = b \cdot h\}$ 
  obtain  $T$  where  $0 < T$ 
    and  $T: \bigwedge j. \llbracket j \in F; j \neq h \rrbracket \implies T * (a \cdot j \cdot y - a \cdot j \cdot w) \leq b \cdot j - a \cdot j \cdot w$ 
  proof (cases  $F - \{h\} = \{\}$ )
    case True then show ?thesis
      by (rule_tac  $T=1$  in that) auto
  next
    case False
      then obtain  $h'$  where  $h': h' \in F - \{h\}$  by auto
      let ?body =  $(\lambda j. \text{if } 0 < a \cdot j \cdot y - a \cdot j \cdot w$ 
        then  $(b \cdot j - a \cdot j \cdot w) / (a \cdot j \cdot y - a \cdot j \cdot w)$  else 1) '  $(F - \{h\})$ 
      define inff where inff = Inf ?body
      from  $\langle \text{finite } F \rangle$  have finite ?body
        by blast
      moreover from  $h'$  have ?body  $\neq \{\}$ 
        by blast
      moreover have  $j > 0$  if  $j \in ?body$  for  $j$ 
      proof -
        from that obtain  $x$  where  $x \in F$  and  $x \neq h$  and  $*$ :  $j =$ 
          (if  $0 < a \cdot x \cdot y - a \cdot x \cdot w$ 
            then  $(b \cdot x - a \cdot x \cdot w) / (a \cdot x \cdot y - a \cdot x \cdot w)$  else 1)
          by blast
        with awlt [of  $x$ ] have  $a \cdot x \cdot w < b \cdot x$ 
          by simp
        with * show ?thesis
          by simp
      qed
      ultimately have  $0 < \text{inff}$ 
        by (simp_all add: finite_less_Inf_iff inff_def)
      moreover have inff  $*$   $(a \cdot j \cdot y - a \cdot j \cdot w) \leq b \cdot j - a \cdot j \cdot w$ 
        if  $j \in F$   $j \neq h$  for  $j$ 
      proof (cases  $a \cdot j \cdot w < a \cdot j \cdot y$ )
        case True
          then have inff  $\leq (b \cdot j - a \cdot j \cdot w) / (a \cdot j \cdot y - a \cdot j \cdot w)$ 
            unfolding inff_def
            using  $\langle \text{finite } F \rangle$  by (auto intro: cInf_le_finite simp add: that split:
              if_split_asm)
          then show ?thesis
            using  $\langle 0 < \text{inff} \rangle$  awlt [OF that] mult_strict_left_mono
            by (fastforce simp add: field_split_simps split: if_split_asm)
        next
          case False
            then show ?thesis
              with  $\langle 0 < \text{inff} \rangle$  have inff  $*$   $(a \cdot j \cdot y - a \cdot j \cdot w) \leq 0$ 
                by (simp add: mult_le_0_iff)
            also have  $\dots < b \cdot j - a \cdot j \cdot w$ 
              by (simp add: awlt that)

```

```

    finally show ?thesis by simp
  qed
  ultimately show ?thesis
    by (blast intro: that)
  qed
  define C where C = (1 - T) *R w + T *R y
  have (1 - T) *R w + T *R y ∈ j if j ∈ F for j
  proof (cases j = h)
    case True
    have (1 - T) *R w + T *R y ∈ {x. a h · x ≤ b h}
      using weq yaff by (auto simp: algebra_simps)
    with True faceq [OF that] show ?thesis by metis
  next
    case False
    with T that have (1 - T) *R w + T *R y ∈ {x. a j · x ≤ b j}
      by (simp add: algebra_simps)
    with faceq [OF that] show ?thesis by simp
  qed
  moreover have (1 - T) *R w + T *R y ∈ affine hull S
    using yaff ⟨w ∈ affine hull S⟩ affine_affine_hull affine_alt by blast
  ultimately have C ∈ S
    using seq by (force simp: C_def)
  moreover have a h · C = b h
    using yaff by (force simp: C_def algebra_simps weq)
  ultimately have caff: C ∈ affine hull (S ∩ {y. a h · y = b h})
    by (simp add: hull_inc)
  have waff: w ∈ affine hull (S ∩ {y. a h · y = b h})
    using ⟨w ∈ S⟩ weq by (blast intro: hull_inc)
  have yeq: y = (1 - inverse T) *R w + C /R T
    using ⟨0 < T⟩ by (simp add: C_def algebra_simps)
  show y ∈ affine hull (S ∩ {y. a h · y = b h})
    by (metis yeq affine_affine_hull [simplified affine_alt, rule_format, OF
waff caff])
  qed
  qed
  ultimately have aff_dim (affine hull (S ∩ ?F)) = aff_dim S - 1
    using ⟨b h < a h · z⟩ zaff by (force simp: aff_dim_affine_Int_hyperplane)
  then show ?thesis
    by (simp add: ne_faceS facet_of_def)
  qed
  show ?thesis
  proof
    show ∃h. h ∈ F ∧ C = S ∩ {x. a h · x = b h} ⇒ C facet_of S
      using * by blast
  next
    assume C facet_of S
    then have C face_of S convex C C ≠ {} and aff: aff_dim C = aff_dim S
      - 1
      by (auto simp: facet_of_def face_of_imp_convex)

```

```

then obtain  $x$  where  $x: x \in \text{rel\_interior } C$ 
  by (force simp: rel_interior_eq_empty)
then have  $x \in C$ 
  by (meson subsetD rel_interior_subset)
then have  $x \in S$ 
  using  $\langle C \text{ facet\_of } S \rangle$  facet_of_imp_subset by blast
have  $\text{rels: rel\_interior } S = \{x \in S. \forall h \in F. a \cdot h \cdot x < b \cdot h\}$ 
  by (rule rel_interior_polyhedron_explicit [OF assms])
have  $C \neq S$ 
  using  $\langle C \text{ facet\_of } S \rangle$  facet_of_irrefl by blast
then have  $x \notin \text{rel\_interior } S$ 
by (metis IntI empty_iff  $\langle x \in C \rangle$   $\langle C \neq S \rangle$   $\langle C \text{ facet\_of } S \rangle$  facet_of_disjoint_rel_interior)
with  $\text{rels } \langle x \in S \rangle$  obtain  $i$  where  $i \in F$  and  $i: a \cdot i \cdot x \geq b \cdot i$ 
  by force
have  $x \in \{u. a \cdot i \cdot u \leq b \cdot i\}$ 
  by (metis IntD2 InterE  $\langle i \in F \rangle$   $\langle x \in S \rangle$  faceq seq)
then have  $a \cdot i \cdot x \leq b \cdot i$  by simp
then have  $a \cdot i \cdot x = b \cdot i$  using  $i$  by auto
have  $C \subseteq S \cap \{x. a \cdot i \cdot x = b \cdot i\}$ 
proof (rule subset_of_face_of [of _ S])
  show  $S \cap \{x. a \cdot i \cdot x = b \cdot i\} \text{ face\_of } S$ 
    by (simp add: *  $\langle i \in F \rangle$  facet_of_imp_face_of)
  show  $C \subseteq S$ 
    by (simp add:  $\langle C \text{ facet\_of } S \rangle$  facet_of_imp_subset)
  show  $S \cap \{x. a \cdot i \cdot x = b \cdot i\} \cap \text{rel\_interior } C \neq \{\}$ 
    using  $\langle a \cdot i \cdot x = b \cdot i \rangle$   $\langle x \in S \rangle$   $x$  by blast
qed
then have  $\text{cfac: } C \text{ face\_of } (S \cap \{x. a \cdot i \cdot x = b \cdot i\})$ 
  by (meson  $\langle C \text{ facet\_of } S \rangle$  facet_of_subset_inf_le1)
have  $\text{con: convex } (S \cap \{x. a \cdot i \cdot x = b \cdot i\})$ 
  by (simp add:  $\langle \text{convex } S \rangle$  convex_Int convex_hyperplane)
show  $\exists h. h \in F \wedge C = S \cap \{x. a \cdot h \cdot x = b \cdot h\}$ 
  apply (rule_tac  $x=i$  in exI)
  by (metis (no_types)  $*$   $\langle i \in F \rangle$  affc facet_of_def less_irrefl facet_of_aff_dim_lt
[OF con cface])
qed
qed

```

lemma *facet_of_polyhedron_subset_explicit*:

fixes $S :: 'a :: \text{euclidean_space set}$

assumes *finite F*

and *seq: S = affine hull S $\cap \bigcap F$*

and *faceq: $\bigwedge h. h \in F \implies a \cdot h \neq 0 \wedge h = \{x. a \cdot h \cdot x \leq b \cdot h\}$*

and *psub: $\bigwedge F'. F' \subset F \implies S \subset \text{affine hull } S \cap \bigcap F'$*

and $C: C \text{ facet_of } S$ **and** $C \neq \{\}$ $C \neq S$

obtains h **where** $h \in F$ $C \subseteq S \cap \{x. a \cdot h \cdot x = b \cdot h\}$

proof –

have $C \subseteq S$ **using** $\langle C \text{ facet_of } S \rangle$

```

  by (simp add: face_of_imp_subset)
  have polyhedron S
  by (metis ⟨finite F⟩ faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
polyhedron_halfspace_le seq)
  then have convex S
  by (simp add: polyhedron_imp_convex)
  then have *: (S ∩ {x. a h · x = b h}) face_of S if h ∈ F for h
  using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
  have rel_interior C ≠ {}
  using C ⟨C ≠ {}⟩ face_of_imp_convex rel_interior_eq_empty by blast
  then obtain x where x ∈ rel_interior C by auto
  have rels: rel_interior S = {x ∈ S. ∀ h ∈ F. a h · x < b h}
  by (rule rel_interior_polyhedron_explicit [OF ⟨finite F⟩ seq faceq psub])
  then have xnot: x ∉ rel_interior S
  by (metis IntI ⟨x ∈ rel_interior C⟩ C ⟨C ≠ S⟩ contra_subsetD empty_iff
face_of_disjoint_rel_interior rel_interior_subset)
  then have x ∈ S
  using ⟨C ⊆ S⟩ ⟨x ∈ rel_interior C⟩ rel_interior_subset by auto
  then have xint: x ∈ ∩ F
  using seq by blast
  have F ≠ {} using assms
  by (metis affine_Int affine_Inter affine_affine_hull ex_in_conv face_of_affine_trivial)
  then obtain i where i ∈ F ∧ (a i · x < b i)
  using ⟨x ∈ S⟩ rels xnot by auto
  with xint have a i · x = b i
  by (metis eq_iff mem_Collect_eq not_le Inter_iff faceq)
  have face: S ∩ {x. a i · x = b i} face_of S
  by (simp add: * ⟨i ∈ F⟩)
  show ?thesis
  proof
    show C ⊆ S ∩ {x. a i · x = b i}
    using subset_of_face_of [OF face ⟨C ⊆ S⟩] ⟨a i · x = b i⟩ ⟨x ∈ rel_interior
C⟩ ⟨x ∈ S⟩ by blast
  qed fact
qed

```

Initial part of proof duplicates that above

proposition *face_of_polyhedron_explicit*:

fixes $S :: 'a :: euclidean_space$ set

assumes *finite F*

and *seq*: $S = \text{affine hull } S \cap \bigcap F$

and *faceq*: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$

and *psub*: $\bigwedge F'. F' \subset F \implies S \subset \text{affine hull } S \cap \bigcap F'$

and *C*: $C \text{ face_of } S$ and $C \neq \{\}$ $C \neq S$

shows $C = \bigcap \{S \cap \{x. a h \cdot x = b h\} \mid h. h \in F \wedge C \subseteq S \cap \{x. a h \cdot x = b h\}\}$

proof –

let $?ab = \lambda h. \{x. a h \cdot x = b h\}$

have $C \subseteq S$ using $\langle C \text{ face_of } S \rangle$

```

  by (simp add: face_of_imp_subset)
  have polyhedron S
  by (metis ⟨finite F⟩ faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
polyhedron_halfspace_le seq)
  then have convex S
  by (simp add: polyhedron_imp_convex)
  then have *: (S ∩ ?ab h) face_of S if h ∈ F for h
  using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
  have rel_interior C ≠ {}
  using C ⟨C ≠ {}⟩ face_of_imp_convex rel_interior_eq_empty by blast
  then obtain z where z: z ∈ rel_interior C by auto
  have rels: rel_interior S = {z ∈ S. ∀ h ∈ F. a h · z < b h}
  by (rule rel_interior_polyhedron_explicit [OF ⟨finite F⟩ seq faceq psub])
  then have xnot: z ∉ rel_interior S
  by (metis IntI ⟨z ∈ rel_interior C⟩ C ⟨C ≠ S⟩ contra_subsetD empty_iff
face_of_disjoint_rel_interior_rel_interior_subset)
  then have z ∈ S
  using ⟨C ⊆ S⟩ ⟨z ∈ rel_interior C⟩ rel_interior_subset by auto
  with seq have xint: z ∈ ∩ F by blast
  have open (∩ h ∈ {h ∈ F. a h · z < b h}. {w. a h · w < b h})
  by (auto simp: ⟨finite F⟩ open_halfspace_lt open_INT)
  then obtain e where 0 < e
  ball z e ⊆ (∩ h ∈ {h ∈ F. a h · z < b h}. {w. a h · w < b h})
  by (auto intro: openE [of _ z])
  then have e: ∧ h. [h ∈ F; a h · z < b h] ⇒ ball z e ⊆ {w. a h · w < b h}
  by blast
  have C ⊆ (S ∩ ?ab h) ↔ z ∈ S ∩ ?ab h if h ∈ F for h
  proof
  show z ∈ S ∩ ?ab h ⇒ C ⊆ S ∩ ?ab h
  by (metis * Collect_cong IntI ⟨C ⊆ S⟩ empty_iff subset_of_face_of that z)
  next
  show C ⊆ S ∩ ?ab h ⇒ z ∈ S ∩ ?ab h
  using ⟨z ∈ rel_interior C⟩ rel_interior_subset by force
  qed
  then have **: {S ∩ ?ab h | h. h ∈ F ∧ C ⊆ S ∧ C ⊆ ?ab h} =
  {S ∩ ?ab h | h. h ∈ F ∧ z ∈ S ∩ ?ab h}
  by blast
  have bsub: ball z e ∩ affine_hull (∩ {S ∩ ?ab h | h. h ∈ F ∧ a h · z = b h})
  ⊆ affine_hull S ∩ ∩ F ∩ ∩ {?ab h | h. h ∈ F ∧ a h · z = b h}
  if i ∈ F and i: a i · z = b i for i
  proof -
  have sub: ball z e ∩ ∩ {?ab h | h. h ∈ F ∧ a h · z = b h} ⊆ j
  if j ∈ F for j
  proof -
  have a j · z ≤ b j using faceq that xint by auto
  then consider a j · z < b j | a j · z = b j by linarith
  then have ∃ G. G ∈ {?ab h | h. h ∈ F ∧ a h · z = b h} ∧ ball z e ∩ G ⊆ j
  proof cases
  assume a j · z < b j

```



```

then have ball z e  $\cap$  {x. a i  $\cdot$  x = b i}  $\subseteq$  j
  using e [OF  $\langle j \in F \rangle$ ] faceq that
  by (fastforce simp: ball_def)
then show ?thesis
  by (rule_tac x={x. a i  $\cdot$  x = b i} in exI) (force simp:  $\langle i \in F \rangle$  i)
next
  assume eq: a j  $\cdot$  z = b j
  with faceq that show ?thesis
  by (rule_tac x={x. a j  $\cdot$  x = b j} in exI) (fastforce simp add:  $\langle j \in F \rangle$ )
qed
then show ?thesis by blast
qed
have 1: affine hull  $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}  $\subseteq$  affine hull S
  using that  $\langle z \in S \rangle$  by (intro hull_mono) auto
have 2: affine hull  $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}
   $\subseteq$   $\cap$  {?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}
  by (rule hull_minimal) (auto intro: affine_hyperplane)
have 3: ball z e  $\cap$   $\cap$  {?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}  $\subseteq$   $\cap$  F
  by (iprover intro: sub_Inter_greatest)
have *:  $\llbracket A \subseteq (B :: 'a \text{ set}); A \subseteq C; E \cap C \subseteq D \rrbracket \implies E \cap A \subseteq (B \cap D) \cap C$ 
  for A B C D E by blast
show ?thesis by (intro * 1 2 3)
qed
have  $\exists h. h \in F \wedge C \subseteq ?ab h$ 
  using assms
  by (metis face_of_polyhedron_subset_explicit [OF  $\langle \text{finite } F \rangle$  seq faceq psub]
le_inf_iff)
then have fac:  $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  C  $\subseteq$  S  $\cap$  ?ab h} face_of S
  using * by (force simp:  $\langle C \subseteq S \rangle$  intro: face_of_Inter)
have red:  $(\bigwedge a. P a \implies T \subseteq S \cap \cap \{F X |X. P X\}) \implies T \subseteq \cap \{S \cap F X |X::'a$ 
set. P X\} for P T F
  by blast
have ball z e  $\cap$  affine hull  $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}
   $\subseteq$   $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  a h  $\cdot$  z = b h}
  by (rule red) (metis seqbsub)
with  $\langle 0 < e \rangle$  have zinrel: z  $\in$  rel_interior
  ( $\cap$  {S  $\cap$  ?ab h |h. h  $\in$  F  $\wedge$  z  $\in$  S  $\wedge$  a h  $\cdot$  z = b h})
  by (auto simp: mem_rel_interior_ball  $\langle z \in S \rangle$ )
show ?thesis
  using z zinrel
  by (intro face_of_eq [OF C fac]) (force simp: **)
qed

```

9.24.12 More general corollaries from the explicit representation

corollary facet_of_polyhedron:

assumes polyhedron S and C facet_of S

obtains a b where $a \neq 0$ $S \subseteq \{x. a \cdot x \leq b\}$ $C = S \cap \{x. a \cdot x = b\}$

proof –

obtain F **where** *finite* F **and** *seq*: $S = \text{affine hull } S \cap \bigcap F$
and *faces*: $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$
and *min*: $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$
using *assms* **by** (*simp* *add*: *polyhedron_Int_affine_minimal*) *meson*
then obtain $a b$ **where** *ab*: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$
by *metis*
obtain i **where** $i \in F$ **and** C : $C = S \cap \{x. a i \cdot x = b i\}$
using *facet_of_polyhedron_explicit* [*OF* \langle *finite* F \rangle *seq* *ab* *min*] *assms*
by *force*
moreover have *ssub*: $S \subseteq \{x. a i \cdot x \leq b i\}$
using $\langle i \in F \rangle$ *ab* **by** (*subst* *seq*) *auto*
ultimately show *?thesis*
by (*rule_tac* $a = a i$ **and** $b = b i$ **in** *that*) (*simp_all* *add*: *ab*)

qed

corollary *face_of_polyhedron*:

assumes *polyhedron* S **and** C *face_of* S **and** $C \neq \{\}$ **and** $C \neq S$
shows $C = \bigcap \{F. F \text{ facet_of } S \wedge C \subseteq F\}$

proof –

obtain F **where** *finite* F **and** *seq*: $S = \text{affine hull } S \cap \bigcap F$
and *faces*: $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$
and *min*: $\bigwedge F'. F' \subset F \implies S \subset (\text{affine hull } S) \cap \bigcap F'$
using *assms* **by** (*simp* *add*: *polyhedron_Int_affine_minimal*) *meson*
then obtain $a b$ **where** *ab*: $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$
by *metis*
show *?thesis*
apply (*subst* *face_of_polyhedron_explicit* [*OF* \langle *finite* F \rangle *seq* *ab* *min*])
apply (*auto* *simp*: *assms* *facet_of_polyhedron_explicit* [*OF* \langle *finite* F \rangle *seq* *ab*
min] *cong*: *Collect_cong*)
done

qed

lemma *face_of_polyhedron_subset_facet*:

assumes *polyhedron* S **and** C *face_of* S **and** $C \neq \{\}$ **and** $C \neq S$
obtains F **where** F *facet_of* S $C \subseteq F$

using *face_of_polyhedron* *assms*

by (*metis* (*no_types*, *lifting*) *Inf_greatest_antisym_conv* *face_of_imp_subset*
mem_Collect_eq)

lemma *exposed_face_of_polyhedron*:

assumes *polyhedron* S

shows F *exposed_face_of* $S \iff F$ *face_of* S

proof

show F *exposed_face_of* $S \implies F$ *face_of* S

by (*simp* *add*: *exposed_face_of_def*)

next

assume F *face_of* S

```

show  $F$  exposed_face_of  $S$ 
proof (cases  $F = \{\}$   $\vee$   $F = S$ )
  case True then show ?thesis
    using  $\langle F$  face_of  $S \rangle$  exposed_face_of by blast
  next
  case False
  then have  $\{g. g$  facet_of  $S \wedge F \subseteq g\} \neq \{\}$ 
  by (metis Collect_empty_eq_bot  $\langle F$  face_of  $S \rangle$  assms empty_def face_of_polyhedron_subset_facet)
  moreover have  $\bigwedge T. \llbracket T$  facet_of  $S; F \subseteq T \rrbracket \implies T$  exposed_face_of  $S$ 
  by (metis assms exposed_face_of_facet_of_imp_face_of_facet_of_polyhedron)
  ultimately have  $\bigcap \{G. G$  facet_of  $S \wedge F \subseteq G\}$  exposed_face_of  $S$ 
  by (metis (no_types, lifting) mem_Collect_eq exposed_face_of_Inter)
  then show ?thesis
    using False  $\langle F$  face_of  $S \rangle$  assms face_of_polyhedron by fastforce
qed
qed

lemma face_of_polyhedron_polyhedron:
  fixes  $S :: 'a ::$  euclidean_space set
  assumes polyhedron  $S$  c face_of  $S$  shows polyhedron  $c$ 
by (metis assms face_of_imp_eq_affine_Int polyhedron_Int polyhedron_affine_hull
polyhedron_imp_convex)

lemma finite_polyhedron_faces:
  fixes  $S :: 'a ::$  euclidean_space set
  assumes polyhedron  $S$ 
  shows finite  $\{F. F$  face_of  $S\}$ 
proof –
  obtain  $F$  where finite  $F$  and seq:  $S =$  affine_hull  $S \cap \bigcap F$ 
    and faces:  $\bigwedge h. h \in F \implies \exists a b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$ 
    and min:  $\bigwedge F'. F' \subset F \implies S \subset (\text{affine\_hull } S) \cap \bigcap F'$ 
  using assms by (simp add: polyhedron_Int_affine_minimal) meson
  then obtain  $a b$  where ab:  $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$ 
  by metis
  have finite  $\{\bigcap \{S \cap \{x. a h \cdot x = b h\} \mid h. h \in F'\} \mid F'. F' \in \text{Pow } F\}$ 
  by (simp add:  $\langle$  finite  $F \rangle$ )
  moreover have  $\{F. F$  face_of  $S\} - \{\{\}, S\} \subseteq \{\bigcap \{S \cap \{x. a h \cdot x = b h\} \mid h.$ 
 $h \in F'\} \mid F'. F' \in \text{Pow } F\}$ 
  apply clarify
  apply (rename_tac  $c$ )
  apply (drule face_of_polyhedron_explicit [OF  $\langle$  finite  $F \rangle$  seq ab min, simplified],
simp_all)
  apply (rule_tac  $x = \{h \in F. c \subseteq S \cap \{x. a h \cdot x = b h\}\}$  in exI, auto)
  done
  ultimately show ?thesis
  by (meson finite.emptyI finite.insertI finite_Diff2 finite_subset)
qed

```

lemma *finite_polyhedron_exposed_faces*:

3472

$polyhedron\ S \implies finite\ \{F.\ F\ exposed_face_of\ S\}$
using *exposed_face_of_polyhedron finite_polyhedron_faces* **by** *fastforce*

lemma *finite_polyhedron_extreme_points*:

fixes $S :: 'a :: euclidean_space\ set$

assumes *polyhedron S* **shows** *finite {v. v extreme_point_of S}*

proof –

have *finite {v. {v} face_of S}*

using *assms* **by** (*intro finite_subset [OF _ finite_vimageI [OF finite_polyhedron_faces]]*,
auto)

then show *?thesis*

by (*simp add: face_of_singleton*)

qed

lemma *finite_polyhedron_facets*:

fixes $S :: 'a :: euclidean_space\ set$

shows *polyhedron S* $\implies finite\ \{F.\ F\ facet_of\ S\}$

unfolding *facet_of_def*

by (*blast intro: finite_subset [OF _ finite_polyhedron_faces]*)

proposition *rel_interior_of_polyhedron*:

fixes $S :: 'a :: euclidean_space\ set$

assumes *polyhedron S*

shows *rel_interior S = S - $\bigcup\{F.\ F\ facet_of\ S\}$*

proof –

obtain F **where** *finite F* **and** *seq: S = affine_hull S \cap $\bigcap F$*

and *faces: $\bigwedge h. h \in F \implies \exists a\ b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$*

and *min: $\bigwedge F'. F' \subset F \implies S \subset (affine_hull\ S) \cap \bigcap F'$*

using *assms* **by** (*simp add: polyhedron_Int_affine_minimal*) *meson*

then obtain $a\ b$ **where** *ab: $\bigwedge h. h \in F \implies a\ h \neq 0 \wedge h = \{x. a\ h \cdot x \leq b\ h\}$*

by *metis*

have *facet: $(c\ facet_of\ S) \iff (\exists h. h \in F \wedge c = S \cap \{x. a\ h \cdot x = b\ h\})$* **for** c

by (*rule facet_of_polyhedron_explicit [OF <finite F> seq ab min]*)

have *rel: $rel_interior\ S = \{x \in S. \forall h \in F. a\ h \cdot x < b\ h\}$*

by (*rule rel_interior_polyhedron_explicit [OF <finite F> seq ab min]*)

have *$a\ h \cdot x < b\ h$ if $x \in S\ h \in F$ and *xnot: $x \notin \bigcup\{F.\ F\ facet_of\ S\}$* **for** $x\ h$*

proof –

have $x \in \bigcap F$ **using** *seq* **that** **by** *force*

with $\langle h \in F \rangle\ ab$ **have** $a\ h \cdot x \leq b\ h$ **by** *auto*

then consider $a\ h \cdot x < b\ h \mid a\ h \cdot x = b\ h$ **by** *linarith*

then show *?thesis*

proof *cases*

case 1 **then show** *?thesis* .

next

case 2

have *Collect ((\in) x) \notin Collect ((\in) ($\bigcup\{A.\ A\ facet_of\ S\}))$*

using *xnot* **by** *fastforce*

then have $F \notin Collect ((\in) h)$

```

    using 2 ⟨ $x \in S$ ⟩ facet by blast
  with 2 that ⟨ $x \in \bigcap F$ ⟩ show ?thesis
    by blast
  qed
qed
moreover have  $\exists h \in F. a \cdot h \cdot x \geq b \cdot h$  if  $x \in \bigcup \{F. F \text{ facet\_of } S\}$  for  $x$ 
  using that by (force simp: facet)
ultimately show ?thesis
  by (force simp: rel)
qed

lemma rel_boundary_of_polyhedron:
  fixes  $S :: 'a :: euclidean\_space \text{ set}$ 
  assumes polyhedron  $S$ 
  shows  $S - \text{rel\_interior } S = \bigcup \{F. F \text{ facet\_of } S\}$ 
using facet_of_imp_subset by (fastforce simp add: rel_interior_of_polyhedron
  assms)

lemma rel_frontier_of_polyhedron:
  fixes  $S :: 'a :: euclidean\_space \text{ set}$ 
  assumes polyhedron  $S$ 
  shows  $\text{rel\_frontier } S = \bigcup \{F. F \text{ facet\_of } S\}$ 
by (simp add: assms rel_frontier_def polyhedron_imp_closed rel_boundary_of_polyhedron)

lemma rel_frontier_of_polyhedron_alt:
  fixes  $S :: 'a :: euclidean\_space \text{ set}$ 
  assumes polyhedron  $S$ 
  shows  $\text{rel\_frontier } S = \bigcup \{F. F \text{ face\_of } S \wedge F \neq S\}$ 
proof
  show  $\text{rel\_frontier } S \subseteq \bigcup \{F. F \text{ face\_of } S \wedge F \neq S\}$ 
    by (force simp: rel_frontier_of_polyhedron facet_of_def assms)
  qed (use face_of_subset_rel_frontier in fastforce)

A characterization of polyhedra as having finitely many faces

proposition polyhedron_eq_finite_exposed_faces:
  fixes  $S :: 'a :: euclidean\_space \text{ set}$ 
  shows  $\text{polyhedron } S \iff \text{closed } S \wedge \text{convex } S \wedge \text{finite } \{F. F \text{ exposed\_face\_of } S\}$ 
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (auto simp: polyhedron_imp_closed polyhedron_imp_convex finite_polyhedron_exposed_faces)
next
  assume ?rhs
  then have  $\text{closed } S \text{ convex } S$  and fin:  $\text{finite } \{F. F \text{ exposed\_face\_of } S\}$  by auto
  show ?lhs
  proof (cases  $S = \{\}$ )
    case True then show ?thesis by auto
  next

```

```

case False
define F where F = {h. h exposed_face_of S ∧ h ≠ {} ∧ h ≠ S}
have finite F by (simp add: fin F_def)
have hface: h face_of S
  and ∃ a b. a ≠ 0 ∧ S ⊆ {x. a · x ≤ b} ∧ h = S ∩ {x. a · x = b}
  if h ∈ F for h
  using exposed_face_of F_def that by blast+
then obtain a b where ab:
  ∧ h. h ∈ F ⇒ a h ≠ 0 ∧ S ⊆ {x. a h · x ≤ b h} ∧ h = S ∩ {x. a h · x = b
h}
  by metis
have *: False
if paff: p ∈ affine hull S and p ∉ S
and pint: p ∈ ⋂ {{x. a h · x ≤ b h} | h. h ∈ F} for p
proof -
have rel_interior S ≠ {}
  by (simp add: ⟨S ≠ {}⟩ ⟨convex S⟩ rel_interior_eq_empty)
then obtain c where c: c ∈ rel_interior S by auto
with rel_interior_subset have c ∈ S by blast
have ccp: closed_segment c p ⊆ affine hull S
  by (meson affine_affine_hull affine_imp_convex c closed_segment_subset
hull_subset paff rel_interior_subset subsetCE)
have oS: openin (top_of_set (closed_segment c p)) (closed_segment c p ∩
rel_interior S)
  by (force simp: openin_rel_interior openin_Int intro: openin_subtopology_Int_subset
[OF _ ccp])
obtain x where xcl: x ∈ closed_segment c p and x ∈ S and xnot: x ∉
rel_interior S
  using connected_openin [of closed_segment c p]
  apply simp
  apply (drule_tac x=closed_segment c p ∩ rel_interior S in spec)
  apply (drule mp [OF _ oS])
  apply (drule_tac x=closed_segment c p ∩ (− S) in spec)
  using rel_interior_subset ⟨closed S⟩ c ⟨p ∉ S⟩ apply blast
done
then obtain μ where 0 ≤ μ μ ≤ 1 and req: x = (1 − μ) *R c + μ *R p
  by (auto simp: in_segment)
show False
proof (cases μ=0 ∨ μ=1)
case True with req c xnot ⟨x ∈ S⟩ ⟨p ∉ S⟩
show False by auto
next
case False
then have xos: x ∈ open_segment c p
  using ⟨x ∈ S⟩ c open_segment_def that(2) xcl xnot by auto
have xclo: x ∈ closure S
  using ⟨x ∈ S⟩ closure_subset by blast
obtain d where d ≠ 0
  and dle: ∧ y. y ∈ closure S ⇒ d · x ≤ d · y

```

```

    and dless:  $\bigwedge y. y \in \text{rel\_interior } S \implies d \cdot x < d \cdot y$ 
  by (metis supporting_hyperplane_relative_frontier [OF ‹convex S› xclo
xnot])
  have sex:  $S \cap \{y. d \cdot y = d \cdot x\}$  exposed_face_of S
  by (simp add: ‹closed S› dle exposed_face_of_Int_supporting_hyperplane_ge
[OF ‹convex S›])
  have sne:  $S \cap \{y. d \cdot y = d \cdot x\} \neq \{\}$ 
  using ‹x ∈ S› by blast
  have sns:  $S \cap \{y. d \cdot y = d \cdot x\} \neq S$ 
  by (metis (mono_tags) Int_Collect c subsetD dless not_le order_refl
rel_interior_subset)
  obtain h where h ∈ F x ∈ h
  using F_def ‹x ∈ S› sex sns by blast
  have abface:  $\{y. a \cdot h \cdot y = b \cdot h\}$  face_of  $\{y. a \cdot h \cdot y \leq b \cdot h\}$ 
  using hyperplane_face_of_halfspace_le by blast
  then have c ∈ h
  using face_ofD [OF abface xos] ‹c ∈ S› ‹h ∈ F› ab pint ‹x ∈ h› by blast
  with c have h ∩ rel_interior S ≠  $\{\}$  by blast
  then show False
  using ‹h ∈ F› F_def face_of_disjoint_rel_interior hface by auto
qed
qed
let ?S' = affine_hull S ∩  $\bigcap \{\{x. a \cdot h \cdot x \leq b \cdot h\} \mid h. h \in F\}$ 
have S ⊆ ?S'
  using ab by (auto simp: hull_subset)
moreover have ?S' ⊆ S
  using * by blast
ultimately have S = ?S' ..
moreover have polyhedron ?S'
  by (force intro: polyhedron_affine_hull polyhedron_halfspace_le simp: ‹finite
F›)
ultimately show ?thesis
  by auto
qed
qed

corollary polyhedron_eq_finite_faces:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S  $\longleftrightarrow$  closed S ∧ convex S ∧ finite {F. F face_of S}
  (is ?lhs = ?rhs)

proof
  assume ?lhs
  then show ?rhs
  by (simp add: finite_polyhedron_faces polyhedron_imp_closed polyhedron_imp_convex)
next
  assume ?rhs
  then show ?lhs
  by (force simp: polyhedron_eq_finite_exposed_faces exposed_face_of intro:
finite_subset)

```

qed

lemma *polyhedron_linear_image_eq*:
fixes $h :: 'a :: euclidean_space \Rightarrow 'b :: euclidean_space$
assumes *linear h bij h*
shows $polyhedron (h \text{ ` } S) \longleftrightarrow polyhedron S$
proof –
have [*simp*]: *inj h using bij_is_inj assms by blast*
then have *injinj: inj_on ((\text{ ` } h) A for A*
by (*simp add: inj_on_def inj_image_eq_iff*)
{ fix P
have $\bigwedge x. P x \implies x \in (\text{ ` } h \text{ ` } \{f. P (h \text{ ` } f)\})$
using *bij_is_surj [OF \langle bij h \rangle]*
by (*metis image_eqI mem_Collect_eq subset_imageE top_greatest*)
then have $\{f. P f\} = (image h) \text{ ` } \{f. P (h \text{ ` } f)\}$
by force
}
then have $finite \{F. F \text{ face_of } h \text{ ` } S\} = finite \{F. F \text{ face_of } S\}$
using *\langle linear h \rangle*
by (*simp add: finite_image_iff injinjinj flip: face_of_linear_image [of h _ S]*)
then show *?thesis*
using *\langle linear h \rangle*
by (*simp add: polyhedron_eq_finite_faces closed_injective_linear_image_eq*)
qed

lemma *polyhedron_negations*:
fixes $S :: 'a :: euclidean_space \text{ set}$
shows $polyhedron S \implies polyhedron(image \text{ uminus } S)$
by (*subst polyhedron_linear_image_eq*) (*auto simp: bij_uminus intro!: linear_uminus*)

9.24.13 Relation between polytopes and polyhedra

proposition *polytope_eq_bounded_polyhedron*:
fixes $S :: 'a :: euclidean_space \text{ set}$
shows $polytope S \longleftrightarrow polyhedron S \wedge bounded S$
(is ?lhs = ?rhs)
proof
assume *?lhs*
then show *?rhs*
by (*simp add: finite_polytope_faces polyhedron_eq_finite_faces*
polytope_imp_closed polytope_imp_convex polytope_imp_bounded)
next
assume $R: ?rhs$
then have $finite \{v. v \text{ extreme_point_of } S\}$
by (*simp add: finite_polyhedron_extreme_points*)
moreover have $S = convex \text{ hull } \{v. v \text{ extreme_point_of } S\}$
using R **by** (*simp add: Krein_Milman_Minkowski compact_eq_bounded_closed*
polyhedron_imp_closed polyhedron_imp_convex)
ultimately show *?lhs*

unfolding *polytope_def* **by** *blast*
qed

lemma *polytope_Int*:
fixes $S :: 'a :: \text{euclidean_space set}$
shows $\llbracket \text{polytope } S; \text{polytope } T \rrbracket \implies \text{polytope}(S \cap T)$
by (*simp add: polytope_eq_bounded_polyhedron bounded_Int*)

lemma *polytope_Int_polyhedron*:
fixes $S :: 'a :: \text{euclidean_space set}$
shows $\llbracket \text{polytope } S; \text{polyhedron } T \rrbracket \implies \text{polytope}(S \cap T)$
by (*simp add: bounded_Int polytope_eq_bounded_polyhedron*)

lemma *polyhedron_Int_polytope*:
fixes $S :: 'a :: \text{euclidean_space set}$
shows $\llbracket \text{polyhedron } S; \text{polytope } T \rrbracket \implies \text{polytope}(S \cap T)$
by (*simp add: bounded_Int polytope_eq_bounded_polyhedron*)

lemma *polytope_imp_polyhedron*:
fixes $S :: 'a :: \text{euclidean_space set}$
shows $\text{polytope } S \implies \text{polyhedron } S$
by (*simp add: polytope_eq_bounded_polyhedron*)

lemma *polytope_facet_exists*:
fixes $p :: 'a :: \text{euclidean_space set}$
assumes $\text{polytope } p \ 0 < \text{aff_dim } p$
obtains F **where** $F \text{ facet_of } p$
proof (*cases p = {}*)
case *True* **with** *assms* **show** *?thesis* **by** *auto*
next
case *False*
then **obtain** v **where** $v \text{ extreme_point_of } p$
using *extreme_point_exists_convex*
by (*blast intro: <polytope p> polytope_imp_compact polytope_imp_convex*)
then
show *?thesis*
by (*metis face_of_polyhedron_subset_facet polytope_imp_polyhedron aff_dim_singleton_not_in_conv assms face_of_singleton less_irrefl singletonI that*)
qed

lemma *polyhedron_interval* [*iff*]: $\text{polyhedron}(\text{cbox } a \ b)$
by (*metis polytope_imp_polyhedron polytope_interval*)

lemma *polyhedron_convex_hull*:
fixes $S :: 'a :: \text{euclidean_space set}$
shows $\text{finite } S \implies \text{polyhedron}(\text{convex_hull } S)$
by (*simp add: polytope_convex_hull polytope_imp_polyhedron*)

9.24.14 Relative and absolute frontier of a polytope

lemma *rel_boundary_of_convex_hull*:

fixes $S :: 'a::\text{euclidean_space set}$

assumes $\neg \text{affine_dependent } S$

shows $(\text{convex hull } S) - \text{rel_interior}(\text{convex hull } S) = (\bigcup_{a \in S} \text{convex hull } (S - \{a\}))$

proof –

have *finite* S **by** (*metis* *assms* *aff_independent_finite*)

then consider $\text{card } S = 0 \mid \text{card } S = 1 \mid 2 \leq \text{card } S$ **by** *arith*

then show *?thesis*

proof *cases*

case 1 **then have** $S = \{\}$ **by** (*simp* *add*: $\langle \text{finite } S \rangle$)

then show *?thesis* **by** *simp*

next

case 2 **show** *?thesis*

by (*auto* *intro*: *card_1_singletonE* [*OF* $\langle \text{card } S = 1 \rangle$])

next

case 3

with *assms* **show** *?thesis*

by (*auto* *simp*: *polyhedron_convex_hull* *rel_boundary_of_polyhedron* *facet_of_convex_hull* *affine_independent_finite* S)

qed

qed

proposition *frontier_of_convex_hull*:

fixes $S :: 'a::\text{euclidean_space set}$

assumes $\text{card } S = \text{Suc } (\text{DIM } 'a)$

shows $\text{frontier}(\text{convex hull } S) = \bigcup \{ \text{convex hull } (S - \{a\}) \mid a. a \in S \}$

proof (*cases* *affine_dependent* S)

case *True*

have [*iff*]: *finite* S

using *assms* **using** *card.infinite* **by** *force*

then have *ccs*: *closed* $(\text{convex hull } S)$

by (*simp* *add*: *compact_imp_closed* *finite_imp_compact* *convex_hull*)

{ **fix** $x \ T$

assume *int* $(\text{card } T) \leq \text{aff_dim } S + 1$ *finite* T $T \subseteq S$ $x \in \text{convex hull } T$

then have $S \neq T$

using *True* $\langle \text{finite } S \rangle$ *aff_dim_le_card* *affine_independent_iff_card* **by** *fastforce*

then obtain a **where** $a \in S$ $a \notin T$

using $\langle T \subseteq S \rangle$ **by** *blast*

then have $\exists y \in S. x \in \text{convex hull } (S - \{y\})$

using *True* *affine_independent_iff_card* [*of* S]

by (*metis* (*no_types*, *opaque_lifting*) *Diff_eq_empty_iff* *Diff_insert0* $\langle a \notin T \rangle$ $\langle T \subseteq S \rangle$ $\langle x \in \text{convex hull } T \rangle$ *hull_mono* *insert_Diff_single* *subsetCE*)

} **note** $*$ = *this*

have 1 : $\text{convex hull } S \subseteq (\bigcup_{a \in S} \text{convex hull } (S - \{a\}))$

by (*subst* *caratheodory_aff_dim*) (*blast* *dest*: $*$)

have 2 : $\bigcup (\lambda a. \text{convex hull } (S - \{a\})) \text{ ` } S \subseteq \text{convex hull } S$

```

    by (rule Union_least) (metis (no_types, lifting) Diff_subset hull_mono
imageE)
  show ?thesis using True
    apply (simp add: segment_convex_hull_frontier_def)
    using interior_convex_hull_eq_empty [OF assms]
    apply (simp add: closure_closed [OF ccs])
    using 1 2 by auto
next
  case False
  then have frontier (convex hull S) = closure (convex hull S) - interior (convex
hull S)
    by (simp add: rel_boundary_of_convex_hull frontier_def)
  also have ... = (convex hull S) - rel_interior (convex hull S)
    by (metis False aff_independent_finite assms closure_convex_hull finite_imp_compact_convex_hull
hull_hull interior_convex_hull_eq_empty rel_interior_nonempty_interior)
  also have ... =  $\bigcup \{ \text{convex hull } (S - \{a\}) \mid a. a \in S \}$ 
    proof -
      have convex hull S - rel_interior (convex hull S) = rel_frontier (convex hull
S)
        by (simp add: False aff_independent_finite polyhedron_convex_hull rel_boundary_of_polyhedron
rel_frontier_of_polyhedron)
      then show ?thesis
        by (simp add: False rel_frontier_convex_hull_cases)
    qed
  finally show ?thesis .
qed

```

9.24.15 Special case of a triangle

proposition *frontier_of_triangle*:

fixes $a :: 'a::\text{euclidean_space}$

assumes $\text{DIM}('a) = 2$

shows $\text{frontier}(\text{convex hull } \{a,b,c\}) = \text{closed_segment } a \ b \cup \text{closed_segment } b \ c \cup \text{closed_segment } c \ a$
(is ?lhs = ?rhs)

proof (cases $b = a \vee c = a \vee c = b$)

case True then show ?thesis

by (auto simp: assms segment_convex_hull frontier_def empty_interior_convex_hull insert_commute card_insert_le_m1 hull_inc insert_absorb)

next

case False then have [simp]: $\text{card } \{a, b, c\} = \text{Suc } (\text{DIM}('a))$

by (simp add: card.insert_remove Set.insert_Diff_if assms)

show ?thesis

proof

show $?lhs \subseteq ?rhs$

using False

by (force simp: segment_convex_hull frontier_of_convex_hull insert_Diff_if insert_commute split: if_split_asm)

show $?rhs \subseteq ?lhs$

```

using False
apply (simp add: frontier_of_convex_hull segment_convex_hull)
apply (intro conjI subsetI)
apply (rule_tac X=convex_hull {a,b} in UnionI; force simp: Set.insert_Diff_if)
  apply (rule_tac X=convex_hull {b,c} in UnionI; force)
apply (rule_tac X=convex_hull {a,c} in UnionI; force simp: insert_commute
Set.insert_Diff_if)
  done
qed
qed

```

```

corollary inside_of_triangle:
  fixes a :: 'a::euclidean_space
  assumes DIM('a) = 2
  shows inside (closed_segment a b  $\cup$  closed_segment b c  $\cup$  closed_segment c
a) = interior(convex_hull {a,b,c})
by (metis assms frontier_of_triangle bounded_empty bounded_insert convex_convex_hull
inside_frontier_eq_interior bounded_convex_hull)

```

```

corollary interior_of_triangle:
  fixes a :: 'a::euclidean_space
  assumes DIM('a) = 2
  shows interior(convex_hull {a,b,c}) =
    convex_hull {a,b,c} - (closed_segment a b  $\cup$  closed_segment b c  $\cup$ 
closed_segment c a)
  using interior_subset
  by (force simp: frontier_of_triangle [OF assms, symmetric] frontier_def Diff_Diff_Int)

```

9.24.16 Subdividing a cell complex

```

lemma subdivide_interval:
  fixes x::real
  assumes a < |x - y| 0 < a
  obtains n where n  $\in \mathbb{Z}$  x < n * a  $\wedge$  n * a < y  $\vee$  y < n * a  $\wedge$  n * a < x
proof -
  consider a + x < y | a + y < x
  using assms by linarith
  then show ?thesis
  proof cases
  case 1
  let ?n = of_int (floor (x/a)) + 1
  have x: x < ?n * a
    by (meson <0 < a> divide_less_eq floor_eq_iff)
  have ?n * a  $\leq$  a + x
    using <a>0 by (simp add: distrib_right floor_divide_lower)
  also have ... < y
    by (rule 1)
  finally have ?n * a < y .
  with x show ?thesis

```

```

    using Ints_1 Ints_add Ints_of_int that by blast
  next
  case 2
  let ?n = of_int (floor (y/a)) + 1
  have y: y < ?n * a
    by (meson <0 < a> divide_less_eq floor_eq_iff)
  have ?n * a ≤ a + y
    using <a>0> by (simp add: distrib_right floor_divide_lower)
  also have ... < x
    by (rule 2)
  finally have ?n * a < x .
  then show ?thesis
    using Ints_1 Ints_add Ints_of_int that y by blast
qed
qed

```

lemma cell_subdivision_lemma:

```

assumes finite F
  and  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
  and  $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq d$ 
  and  $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face\_of } X$ 
  and finite I
shows  $\exists \mathcal{G}. \bigcup \mathcal{G} = \bigcup \mathcal{F} \wedge$ 
  finite  $\mathcal{G} \wedge$ 
   $(\forall C \in \mathcal{G}. \exists D. D \in \mathcal{F} \wedge C \subseteq D) \wedge$ 
   $(\forall C \in \mathcal{F}. \forall x \in C. \exists D. D \in \mathcal{G} \wedge x \in D \wedge D \subseteq C) \wedge$ 
   $(\forall X \in \mathcal{G}. \text{polytope } X) \wedge$ 
   $(\forall X \in \mathcal{G}. \text{aff\_dim } X \leq d) \wedge$ 
   $(\forall X \in \mathcal{G}. \forall Y \in \mathcal{G}. X \cap Y \text{ face\_of } X) \wedge$ 
   $(\forall X \in \mathcal{G}. \forall x \in X. \forall y \in X. \forall a b.$ 
     $(a, b) \in I \implies a \cdot x \leq b \wedge a \cdot y \leq b \vee$ 
     $a \cdot x \geq b \wedge a \cdot y \geq b)$ 

  using <finite I>
proof induction
  case empty
  then show ?case
    by (rule_tac x=F in exI) (auto simp: assms)
next
  case (insert ab I)
  then obtain G where eq:  $\bigcup \mathcal{G} = \bigcup \mathcal{F}$  and finite G
    and sub1:  $\bigwedge C. C \in \mathcal{G} \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
    and sub2:  $\bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{G} \wedge x \in D \wedge D$ 
 $\subseteq C$ 
    and poly:  $\bigwedge X. X \in \mathcal{G} \implies \text{polytope } X$ 
    and aff:  $\bigwedge X. X \in \mathcal{G} \implies \text{aff\_dim } X \leq d$ 
    and face:  $\bigwedge X Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \text{ face\_of } X$ 
    and I:  $\bigwedge X x y a b. \llbracket X \in \mathcal{G}; x \in X; y \in X; (a, b) \in I \rrbracket \implies$ 
     $a \cdot x \leq b \wedge a \cdot y \leq b \vee a \cdot x \geq b \wedge a \cdot y \geq b$ 
  by (auto simp: that)

```

```

obtain  $a\ b$  where  $ab = (a, b)$ 
by fastforce
let  $?G = (\lambda X. X \cap \{x. a \cdot x \leq b\}) \cup (\lambda X. X \cap \{x. a \cdot x \geq b\}) \cup \mathcal{G}$ 
have  $eqInt: (S \cap Collect\ P) \cap (T \cap Collect\ Q) = (S \cap T) \cap (Collect\ P \cap Collect\ Q)$ 
for  $S\ T :: 'a\ set$  and  $P\ Q$ 
by blast
show  $?case$ 
proof (intro conjI exI)
show  $\bigcup ?G = \bigcup \mathcal{F}$ 
by (force simp: eq [symmetric])
show finite  $?G$ 
using  $\langle finite\ \mathcal{G} \rangle$  by force
show  $\forall X \in ?G. polytope\ X$ 
by (force simp: poly\ polytope\_Int\ polyhedron\ polyhedron\_halfspace\_le\ polyhedron\_halfspace\_ge)
show  $\forall X \in ?G. aff\_dim\ X \leq d$ 
by (auto;metis order\_trans\ aff\ aff\_dim\_subset\ inf\_le1)
show  $\forall X \in ?G. \forall x \in X. \forall y \in X. \forall a\ b.$ 

$$(a, b) \in insert\ ab\ I \longrightarrow a \cdot x \leq b \wedge a \cdot y \leq b \vee$$


$$a \cdot x \geq b \wedge a \cdot y \geq b$$

using  $\langle ab = (a, b) \rangle I$  by fastforce
show  $\forall X \in ?G. \forall Y \in ?G. X \cap Y\ face\_of\ X$ 
by (auto simp: eqInt\ halfspace\_Int\ eq\ face\_of\_Int\_Int\ face\ face\_of\_halfspace\_le\ face\_of\_halfspace\_ge)
show  $\forall C \in ?G. \exists D. D \in \mathcal{F} \wedge C \subseteq D$ 
using sub1 by force
show  $\forall C \in \mathcal{F}. \forall x \in C. \exists D. D \in ?G \wedge x \in D \wedge D \subseteq C$ 
proof (intro ballI)
fix  $C\ z$ 
assume  $C \in \mathcal{F}\ z \in C$ 
with sub2 obtain  $D$  where  $D \in \mathcal{G}\ z \in D\ D \subseteq C$  by blast
have  $D \in \mathcal{G} \wedge z \in D \cap \{x. a \cdot x \leq b\} \wedge D \cap \{x. a \cdot x \leq b\} \subseteq C \vee$ 
 $D \in \mathcal{G} \wedge z \in D \cap \{x. a \cdot x \geq b\} \wedge D \cap \{x. a \cdot x \geq b\} \subseteq C$ 
using linorder\_class.linear [of  $a \cdot z\ b$ ]  $D$  by blast
then show  $\exists D. D \in ?G \wedge z \in D \wedge D \subseteq C$ 
by blast
qed
qed
qed

```

proposition *cell_complex_subdivision_exists:*

fixes $\mathcal{F} :: 'a::euclidean_space\ set\ set$

assumes $0 < e$ *finite* \mathcal{F}

and *poly:* $\bigwedge X. X \in \mathcal{F} \implies polytope\ X$

and *aff:* $\bigwedge X. X \in \mathcal{F} \implies aff_dim\ X \leq d$

and *face:* $\bigwedge X\ Y. [X \in \mathcal{F}; Y \in \mathcal{F}] \implies X \cap Y\ face_of\ X$

obtains \mathcal{F}' **where** *finite* $\mathcal{F}' \cup \mathcal{F}' = \bigcup \mathcal{F} \bigwedge X. X \in \mathcal{F}' \implies diameter\ X < e$
 $\bigwedge X. X \in \mathcal{F}' \implies polytope\ X \bigwedge X. X \in \mathcal{F}' \implies aff_dim\ X \leq d$

$$\begin{aligned} & \bigwedge X Y. \llbracket X \in \mathcal{F}'; Y \in \mathcal{F}' \rrbracket \implies X \cap Y \text{ face_of } X \\ & \bigwedge C. C \in \mathcal{F}' \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D \\ & \bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{F}' \wedge x \in D \wedge D \subseteq C \end{aligned}$$

proof –

have $\text{bounded}(\bigcup \mathcal{F})$

by (*simp add: <finite F> poly bounded_Union polytope_imp_bounded*)

then obtain B **where** $B > 0$ **and** $B: \bigwedge x. x \in \bigcup \mathcal{F} \implies \text{norm } x < B$

by (*meson bounded_pos_less*)

define C **where** $C \equiv \{z \in \mathbb{Z}. |z * e / 2 / \text{real } \text{DIM}('a)| \leq B\}$

define I **where** $I \equiv \bigcup i \in \text{Basis}. \bigcup j \in C. \{(i::'a, j * e / 2 / \text{real } \text{DIM}('a))\}$

have $C \subseteq \{x \in \mathbb{Z}. -B / (e / 2 / \text{real } \text{DIM}('a)) \leq x \wedge x \leq B / (e / 2 / \text{real } \text{DIM}('a))\}$

using $\langle 0 < e \rangle$ **by** (*auto simp: field_split_simps C_def*)

then have *finite C*

using *finite_int_segment finite_subset* **by** *blast*

then have *finite I*

by (*simp add: I_def*)

obtain \mathcal{F}' **where** $\text{eq}: \bigcup \mathcal{F}' = \bigcup \mathcal{F}$ **and** *finite F'*

and *poly: $\bigwedge X. X \in \mathcal{F}' \implies \text{polytope } X$*

and *aff: $\bigwedge X. X \in \mathcal{F}' \implies \text{aff_dim } X \leq d$*

and *face: $\bigwedge X Y. \llbracket X \in \mathcal{F}'; Y \in \mathcal{F}' \rrbracket \implies X \cap Y \text{ face_of } X$*

and $I: \bigwedge X x y a b. \llbracket X \in \mathcal{F}'; x \in X; y \in X; (a,b) \in I \rrbracket \implies$
 $a \cdot x \leq b \wedge a \cdot y \leq b \vee a \cdot x \geq b \wedge a \cdot y \geq b$

and *sub1: $\bigwedge C. C \in \mathcal{F}' \implies \exists D. D \in \mathcal{F} \wedge C \subseteq D$*

and *sub2: $\bigwedge C x. C \in \mathcal{F} \wedge x \in C \implies \exists D. D \in \mathcal{F}' \wedge x \in D \wedge D \subseteq C$*

apply (*rule exE [OF cell_subdivision_lemma]*)

using *assms <finite I>* **by** *auto*

show *?thesis*

proof (*rule_tac F'=F' in that*)

show *diameter X < e* **if** $X \in \mathcal{F}'$ **for** X

proof –

have *diameter X ≤ e/2*

proof (*rule diameter_le*)

show *norm (x - y) ≤ e / 2* **if** $x \in X$ $y \in X$ **for** x y

proof –

have *norm x < B* *norm y < B*

using $B \langle X \in \mathcal{F}' \rangle$ *eq that* **by** *blast+*

have $\text{norm } (x - y) \leq (\sum_{b \in \text{Basis}. |(x-y) \cdot b|})$

by (*rule norm_le_l1*)

also have $\dots \leq \text{of_nat } (\text{DIM}('a)) * (e / 2 / \text{real } \text{DIM}('a))$

proof (*rule sum_bounded_above*)

fix $i::'a$

assume $i \in \text{Basis}$

then have $I': \bigwedge z b. \llbracket z \in C; b = z * e / (2 * \text{real } \text{DIM}('a)) \rrbracket \implies i \cdot x$
 $\leq b \wedge i \cdot y \leq b \vee i \cdot x \geq b \wedge i \cdot y \geq b$

using $I[\text{of } X \ x \ y] \langle X \in \mathcal{F}' \rangle$ *that* **unfolding** I_def **by** *auto*

show $|(x - y) \cdot i| \leq e / 2 / \text{real } \text{DIM}('a)$

proof (*rule ccontr*)

assume $\neg |(x - y) \cdot i| \leq e / 2 / \text{real } \text{DIM}('a)$

```

then have xyi:  $|i \cdot x - i \cdot y| > e / 2 / \text{real DIM}('a)$ 
  by (simp add: inner_commute inner_diff_right)
obtain n where  $n \in \mathbb{Z}$  and  $n: i \cdot x < n * (e / 2 / \text{real DIM}('a)) \wedge$ 
 $n * (e / 2 / \text{real DIM}('a)) < i \cdot y \vee i \cdot y < n * (e / 2 / \text{real DIM}('a)) \wedge n * (e$ 
 $/ 2 / \text{real DIM}('a)) < i \cdot x$ 
  using subdivide_interval [OF xyi] DIM_positive  $\langle 0 < e \rangle$ 
  by (auto simp: zero_less_divide_iff)
have  $|i \cdot x| < B$ 
by (metis  $\langle i \in \text{Basis} \rangle \langle \text{norm } x < B \rangle$  inner_commute norm_bound_Basis_lt)
have  $|i \cdot y| < B$ 
by (metis  $\langle i \in \text{Basis} \rangle \langle \text{norm } y < B \rangle$  inner_commute norm_bound_Basis_lt)
have *:  $|n * e| \leq B * (2 * \text{real DIM}('a))$ 
  if  $|ix| < B$   $|iy| < B$ 
  and  $ix: ix * (2 * \text{real DIM}('a)) < n * e$ 
  and  $iy: n * e < iy * (2 * \text{real DIM}('a))$  for  $ix \ iy$ 
proof (rule abs_leI)
  have  $iy * (2 * \text{real DIM}('a)) \leq B * (2 * \text{real DIM}('a))$ 
  by (rule mult_right_mono) (use  $\langle |iy| < B \rangle$  in linarith)+
  then show  $n * e \leq B * (2 * \text{real DIM}('a))$ 
  using  $iy$  by linarith
next
  have  $- ix * (2 * \text{real DIM}('a)) \leq B * (2 * \text{real DIM}('a))$ 
  by (rule mult_right_mono) (use  $\langle |ix| < B \rangle$  in linarith)+
  then show  $-(n * e) \leq B * (2 * \text{real DIM}('a))$ 
  using  $ix$  by linarith
qed
have  $n \in C$ 
  using  $\langle n \in \mathbb{Z} \rangle n$  by (auto simp: C_def divide_simps intro: *  $\langle |i \cdot x|$ 
 $< B \rangle \langle |i \cdot y| < B \rangle$ )
show False
  using I' [OF  $\langle n \in C \rangle$  refl] n by auto
qed
qed
also have  $\dots = e / 2$ 
  by simp
finally show ?thesis .
qed
qed (use  $\langle 0 < e \rangle$  in force)
also have  $\dots < e$ 
  by (simp add:  $\langle 0 < e \rangle$ )
finally show ?thesis .
qed
qed (auto simp: eq_poly aff_face sub1 sub2  $\langle \text{finite } \mathcal{F}' \rangle$ )
qed

```

9.24.17 Simplexes

The notion of n-simplex for integer $- (1::'a) \leq n$

definition *simplex* :: $\text{int} \Rightarrow 'a::\text{euclidean_space set} \Rightarrow \text{bool}$ (**infix** *simplex* 50)

where n simplex $S \equiv \exists C. \neg \text{affine_dependent } C \wedge \text{int}(\text{card } C) = n + 1 \wedge S = \text{convex hull } C$

lemma *simplex*:

n simplex $S \longleftrightarrow (\exists C. \text{finite } C \wedge$
 $\neg \text{affine_dependent } C \wedge$
 $\text{int}(\text{card } C) = n + 1 \wedge$
 $S = \text{convex hull } C)$

by (*auto simp add: simplex_def intro: aff_independent_finite*)

lemma *simplex_convex_hull*:

$\neg \text{affine_dependent } C \wedge \text{int}(\text{card } C) = n + 1 \implies n$ simplex $(\text{convex hull } C)$

by (*auto simp add: simplex_def*)

lemma *convex_simplex*: n simplex $S \implies \text{convex } S$

by (*metis convex_convex_hull simplex_def*)

lemma *compact_simplex*: n simplex $S \implies \text{compact } S$

unfolding *simplex*

using *finite_imp_compact_convex_hull* **by** *blast*

lemma *closed_simplex*: n simplex $S \implies \text{closed } S$

by (*simp add: compact_imp_closed compact_simplex*)

lemma *simplex_imp_polytope*:

n simplex $S \implies \text{polytope } S$

unfolding *simplex_def polytope_def*

using *aff_independent_finite* **by** *blast*

lemma *simplex_imp_polyhedron*:

n simplex $S \implies \text{polyhedron } S$

by (*simp add: polytope_imp_polyhedron simplex_imp_polytope*)

lemma *simplex_dim_ge*: n simplex $S \implies -1 \leq n$

by (*metis (no_types, opaque_lifting) aff_dim_geq affine_independent_iff_card diff_add_cancel diff_diff_eq2 simplex_def*)

lemma *simplex_empty* [*simp*]: n simplex $\{\}$ $\longleftrightarrow n = -1$

proof

assume n simplex $\{\}$

then show $n = -1$

unfolding *simplex* **by** (*metis card.empty convex_hull_eq_empty diff_0 diff_eq_eq of_nat_0*)

next

assume $n = -1$ **then show** n simplex $\{\}$

by (*fastforce simp: simplex*)

qed

lemma *simplex_minus_1* [*simp*]: -1 simplex $S \longleftrightarrow S = \{\}$

3486

by (*metis simplex cancel comm monoid_add_class.diff_cancel card_0_eq diff_minus_eq_add of_nat_eq_0_iff simplex_empty*)

lemma *aff_dim_simplex*:

n simplex $S \implies \text{aff_dim } S = n$

by (*metis simplex add.commute add_diff_cancel_left' aff_dim_convex_hull affine_independent_iff_ca*)

lemma *zero_simplex_sing*: 0 simplex $\{a\}$

using *affine_independent_1 simplex_convex_hull* **by** *fastforce*

lemma *simplex_sing* [*simp*]: n simplex $\{a\} \longleftrightarrow n = 0$

using *aff_dim_simplex aff_dim_sing zero_simplex_sing* **by** *blast*

lemma *simplex_zero*: 0 simplex $S \longleftrightarrow (\exists a. S = \{a\})$

by (*metis aff_dim_eq_0 aff_dim_simplex simplex_sing*)

lemma *one_simplex_segment*: $a \neq b \implies 1$ simplex *closed_segment* a b

unfolding *simplex_def*

by (*rule_tac x={a,b} in exI*) (*auto simp: segment_convex_hull*)

lemma *simplex_segment_cases*:

(if $a = b$ then 0 else 1) simplex *closed_segment* a b

by (*auto simp: one_simplex_segment*)

lemma *simplex_segment*:

$\exists n. n$ simplex *closed_segment* a b

using *simplex_segment_cases* **by** *metis*

lemma *polytope_lowdim_imp_simplex*:

assumes *polytope* P $\text{aff_dim } P \leq 1$

obtains n **where** n simplex P

proof (*cases* $P = \{\}$)

case *True*

then show *?thesis*

by (*simp add: that*)

next

case *False*

then show *?thesis*

by (*metis assms compact_convex_collinear_segment collinear_aff_dim polytope_imp_compact polytope_imp_convex simplex_segment_cases that*)

qed

lemma *simplex_insert_dimplus1*:

fixes $n::\text{int}$

assumes n simplex S **and** $a \notin \text{affine hull } S$

shows $(n+1)$ simplex (*convex hull* (*insert* a S))

proof –

obtain C **where** C : *finite* $C \neg \text{affine_dependent } C \text{int}(\text{card } C) = n+1$ **and** S :

```

S = convex hull C
  using assms unfolding simplex by force
show ?thesis
  unfolding simplex
  proof (intro exI conjI)
    have aff_dim S = n
      using aff_dim_simplex assms(1) by blast
    moreover have a ∉ affine hull C
      using S a affine_hull_convex_hull by blast
    moreover have a ∉ C
      using S a hull_inc by fastforce
    ultimately show ¬ affine_dependent (insert a C)
      by (simp add: C S aff_dim_convex_hull aff_dim_insert affine_independent_iff_card)
  next
    have a ∉ C
      using S a hull_inc by fastforce
    then show int (card (insert a C)) = n + 1 + 1
      by (simp add: C)
  next
    show convex hull insert a S = convex hull (insert a C)
      by (simp add: S convex_hull_insert_segments)
  qed (use C in auto)
qed

```

9.24.18 Simplicial complexes and triangulations

definition *simplicial_complex* where

$$\begin{aligned}
\text{simplicial_complex } \mathcal{C} \equiv & \\
& \text{finite } \mathcal{C} \wedge \\
& (\forall S \in \mathcal{C}. \exists n. n \text{ simplex } S) \wedge \\
& (\forall F S. S \in \mathcal{C} \wedge F \text{ face_of } S \longrightarrow F \in \mathcal{C}) \wedge \\
& (\forall S S'. S \in \mathcal{C} \wedge S' \in \mathcal{C} \longrightarrow (S \cap S') \text{ face_of } S)
\end{aligned}$$

definition *triangulation* where

$$\begin{aligned}
\text{triangulation } \mathcal{T} \equiv & \\
& \text{finite } \mathcal{T} \wedge \\
& (\forall T \in \mathcal{T}. \exists n. n \text{ simplex } T) \wedge \\
& (\forall T T'. T \in \mathcal{T} \wedge T' \in \mathcal{T} \longrightarrow (T \cap T') \text{ face_of } T)
\end{aligned}$$

9.24.19 Refining a cell complex to a simplicial complex

proposition *convex_hull_insert_Int_eq*:

fixes $z :: 'a :: \text{euclidean_space}$
assumes $z: z \in \text{rel_interior } S$
and $T: T \subseteq \text{rel_frontier } S$
and $U: U \subseteq \text{rel_frontier } S$
and $\text{convex } S \text{ convex } T \text{ convex } U$
shows $\text{convex hull } (\text{insert } z T) \cap \text{convex hull } (\text{insert } z U) = \text{convex hull } (\text{insert } z (T \cap U))$
(is ?lhs = ?rhs)

```

proof
  show ?lhs  $\subseteq$  ?rhs
  proof (cases T={ }  $\vee$  U={ })
    case True then show ?thesis by auto
  next
    case False
    then have  $T \neq \{ \} \ U \neq \{ \}$  by auto
    have TU: convex (T  $\cap$  U)
      by (simp add:  $\langle$ convex T $\rangle$   $\langle$ convex U $\rangle$  convex_Int)
    have ( $\bigcup x \in T. \text{closed\_segment } z \ x$ )  $\cap$  ( $\bigcup x \in U. \text{closed\_segment } z \ x$ )
       $\subseteq$  (if T  $\cap$  U = { } then {z} else  $\bigcup ((\text{closed\_segment } z) \ ' (T \cap U))$ ) (is _
 $\subseteq$  ?IF)
    proof clarify
      fix x t u
      assume xt: x  $\in$  closed_segment z t
        and xu: x  $\in$  closed_segment z u
        and t  $\in$  T u  $\in$  U
      then have ne: t  $\neq$  z u  $\neq$  z
        using T U z unfolding rel_frontier_def by blast+
      show x  $\in$  ?IF
    proof (cases x = z)
      case True then show ?thesis by auto
    next
      case False
      have t: t  $\in$  closure S
        using T  $\langle$ t  $\in$  T $\rangle$  rel_frontier_def by auto
      have u: u  $\in$  closure S
        using U  $\langle$ u  $\in$  U $\rangle$  rel_frontier_def by auto
      show ?thesis
    proof (cases t = u)
      case True
        then show ?thesis
          using  $\langle$ t  $\in$  T $\rangle$   $\langle$ u  $\in$  U $\rangle$  xt by auto
      next
        case False
        have tnot: t  $\notin$  closed_segment u z
          proof -
            have t  $\in$  closure S - rel_interior S
              using T  $\langle$ t  $\in$  T $\rangle$  rel_frontier_def by blast
            then have t  $\notin$  open_segment z u
              by (meson DiffD2 rel_interior_closure_convex_segment [OF  $\langle$ convex
S $\rangle$  z u] subsetD)
            then show ?thesis
              by (simp add:  $\langle$ t  $\neq$  u $\rangle$   $\langle$ t  $\neq$  z $\rangle$  open_segment_commute open_segment_def)
          qed
        moreover have u  $\notin$  closed_segment z t
          using rel_interior_closure_convex_segment [OF  $\langle$ convex S $\rangle$  z t]  $\langle$ u  $\in$ 
U $\rangle$   $\langle$ u  $\neq$  z $\rangle$ 
          U [unfolded rel_frontier_def] tnot

```

```

    by (auto simp: closed_segment_eq_open)
  ultimately
  have  $\neg(\text{between } (t,u) z \mid \text{between } (u,z) t \mid \text{between } (z,t) u)$  if  $x \neq z$ 
    using that xt xu
    by (meson between_antisym between_mem_segment between_trans_2
ends_in_segment(2))
  then have  $\neg \text{collinear } \{t, z, u\}$  if  $x \neq z$ 
    by (auto simp: that collinear_between_cases between_commute)
  moreover have collinear  $\{t, z, x\}$ 
  by (metis closed_segment_commute collinear_2 collinear_closed_segment
collinear_triples ends_in_segment(1) insert_absorb insert_absorb2 xt)
  moreover have collinear  $\{z, x, u\}$ 
  by (metis closed_segment_commute collinear_2 collinear_closed_segment
collinear_triples ends_in_segment(1) insert_absorb insert_absorb2 xu)
  ultimately have False
    using collinear_3_trans [of t z x u]  $\langle x \neq z \rangle$  by blast
  then show ?thesis by metis
qed
qed
qed
then show ?thesis
  using False  $\langle \text{convex } T \rangle \langle \text{convex } U \rangle TU$ 
  by (simp add: convex_hull_insert_segments hull_same split: if_split_asm)
qed
show ?rhs  $\subseteq$  ?lhs
  by (metis inf_greatest hull_mono inf.cobounded1 inf.cobounded2 insert_mono)
qed

```

lemma simplicial_subdivision_aux:

```

  assumes finite  $\mathcal{M}$ 
    and  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq \text{of\_nat } n$ 
    and  $\bigwedge C F. \llbracket C \in \mathcal{M}; F \text{ face\_of } C \rrbracket \implies F \in \mathcal{M}$ 
    and  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  shows  $\exists \mathcal{T}. \text{simplicial\_complex } \mathcal{T} \wedge$ 
     $(\forall K \in \mathcal{T}. \text{aff\_dim } K \leq \text{of\_nat } n) \wedge$ 
     $\bigcup \mathcal{T} = \bigcup \mathcal{M} \wedge$ 
     $(\forall C \in \mathcal{M}. \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F) \wedge$ 
     $(\forall K \in \mathcal{T}. \exists C. C \in \mathcal{M} \wedge K \subseteq C)$ 
  using assms
proof (induction n arbitrary:  $\mathcal{M}$  rule: less_induct)
  case (less n)
  then have poly $\mathcal{M}$ :  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and aff $\mathcal{M}$ :  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq \text{of\_nat } n$ 
    and face $\mathcal{M}$ :  $\bigwedge C F. \llbracket C \in \mathcal{M}; F \text{ face\_of } C \rrbracket \implies F \in \mathcal{M}$ 
    and intface $\mathcal{M}$ :  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
    by metis+
  show ?case
  proof (cases  $n \leq 1$ )

```

```

case True
have  $\bigwedge s. \llbracket n \leq 1; s \in \mathcal{M} \rrbracket \implies \exists m. m \text{ simplex } s$ 
  using polyM affM by (force intro: polytope_lowdim_imp_simplex)
then show ?thesis
  unfolding simplicial_complex_def using True
  by (rule_tac x= $\mathcal{M}$  in exI) (auto simp: less.prems)
next
case False
define S where  $S \equiv \{C \in \mathcal{M}. \text{aff\_dim } C < n\}$ 
have finite S  $\wedge C. C \in S \implies \text{polytope } C \wedge C. C \in S \implies \text{aff\_dim } C \leq \text{int}$ 
(n - 1)
 $\bigwedge C1 C2. \llbracket C1 \in S; C2 \in S \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  using less.prems by (auto simp: S_def)
moreover have  $\S: \bigwedge C F. \llbracket C \in S; F \text{ face\_of } C \rrbracket \implies F \in S$ 
  using less.prems unfolding S_def
  by (metis (no_types, lifting) mem_Collect_eq aff_dim_subset face_of_imp_subset
less_le not_le)
ultimately obtain U where simplicial_complex U
  and aff_dimU:  $\bigwedge K. K \in U \implies \text{aff\_dim } K \leq \text{int } (n - 1)$ 
  and  $\bigcup U = \bigcup S$ 
  and finU:  $\bigwedge C. C \in S \implies \exists F. \text{finite } F \wedge F \subseteq U \wedge C = \bigcup F$ 
  and CU:  $\bigwedge K. K \in U \implies \exists C. C \in S \wedge K \subseteq C$ 
  using less.IH [of n-1 S] False by auto
then have finite U
  and simplU:  $\bigwedge S. S \in U \implies \exists n. n \text{ simplex } S$ 
  and faceU:  $\bigwedge F S. \llbracket S \in U; F \text{ face\_of } S \rrbracket \implies F \in U$ 
  and faceIU:  $\bigwedge S S'. \llbracket S \in U; S' \in U \rrbracket \implies (S \cap S') \text{ face\_of } S$ 
  by (auto simp: simplicial_complex_def)
define N where  $N \equiv \{C \in \mathcal{M}. \text{aff\_dim } C = n\}$ 
have finite N
  by (simp add: N_def less.prems(1))
have polyN:  $\bigwedge C. C \in N \implies \text{polytope } C$ 
  and convexN:  $\bigwedge C. C \in N \implies \text{convex } C$ 
  and closedN:  $\bigwedge C. C \in N \implies \text{closed } C$ 
  by (auto simp: N_def polyM polytope_imp_convex polytope_imp_closed)
have in_rel_interior: (SOME z. z  $\in$  rel_interior C)  $\in$  rel_interior C if C  $\in$ 
N for C
  using that polyM polytope_imp_convex rel_interior_aff_dim some_in_eq
by (fastforce simp: N_def)
have *:  $\exists T. \neg \text{affine\_dependent } T \wedge \text{card } T \leq n \wedge \text{aff\_dim } K < n \wedge K =$ 
convex hull T
  if K  $\in$  U for K
proof -
obtain r where r: r simplex K
  using  $\langle K \in U \rangle$  simplU by blast
have r = aff_dim K
  using  $\langle r \text{ simplex } K \rangle$  aff_dim_simplex by blast
with r
show ?thesis

```

```

    unfolding simplex_def
    using False ⟨ $\bigwedge K. K \in \mathcal{U} \implies \text{aff\_dim } K \leq \text{int } (n - 1)$ ⟩ that by fastforce
  qed
  have ahK_C_disjoint: affine_hull K  $\cap$  rel_interior C = {}
    if C  $\in$   $\mathcal{N}$  K  $\in$   $\mathcal{U}$  K  $\subseteq$  rel_frontier C for C K
  proof -
    have convex C closed C
      by (auto simp: convexN closedN ⟨C  $\in$   $\mathcal{N}$ ⟩)
    obtain F where F: F face_of C and F  $\neq$  C K  $\subseteq$  F
    proof -
      obtain L where L  $\in$   $\mathcal{S}$  K  $\subseteq$  L
        using ⟨K  $\in$   $\mathcal{U}$ ⟩ CU by blast
      have K  $\leq$  rel_frontier C
        by (simp add: ⟨K  $\subseteq$  rel_frontier C⟩)
      also have ...  $\leq$  C
        by (simp add: ⟨closed C⟩ rel_frontier_def subset_iff)
      finally have K  $\subseteq$  C .
      have L  $\cap$  C face_of C
        using N_def S_def ⟨C  $\in$   $\mathcal{N}$ ⟩ ⟨L  $\in$   $\mathcal{S}$ ⟩ intfaceM by (simp add:
inf_commute)
      moreover have L  $\cap$  C  $\neq$  C
        using ⟨C  $\in$   $\mathcal{N}$ ⟩ ⟨L  $\in$   $\mathcal{S}$ ⟩
        by (metis (mono_tags, lifting) N_def S_def intfaceM mem_Collect_eq
not_le order_refl §)
      moreover have K  $\subseteq$  L  $\cap$  C
        using ⟨C  $\in$   $\mathcal{N}$ ⟩ ⟨L  $\in$   $\mathcal{S}$ ⟩ ⟨K  $\subseteq$  C⟩ ⟨K  $\subseteq$  L⟩ by (auto simp: N_def S_def)
      ultimately show ?thesis using that by metis
    qed
    have affine_hull F  $\cap$  rel_interior C = {}
      by (rule affine_hull_face_of_disjoint_rel_interior [OF ⟨convex C⟩ F ⟨F
 $\neq$  C⟩])
    with hull_mono [OF ⟨K  $\subseteq$  F⟩]
    show affine_hull K  $\cap$  rel_interior C = {}
      by fastforce
  qed
  let ?T = ( $\bigcup$  C  $\in$   $\mathcal{N}$ .  $\bigcup$  K  $\in$   $\mathcal{U} \cap$  Pow (rel_frontier C).
    {convex_hull (insert (SOME z. z  $\in$  rel_interior C) K)})
  have  $\exists$  T. simplicial_complex T  $\wedge$ 
    ( $\forall$  K  $\in$  T. aff_dim K  $\leq$  of_nat n)  $\wedge$ 
    ( $\forall$  C  $\in$   $\mathcal{M}$ .  $\exists$  F. F  $\subseteq$  T  $\wedge$  C =  $\bigcup$  F)  $\wedge$ 
    ( $\forall$  K  $\in$  T.  $\exists$  C. C  $\in$   $\mathcal{M} \wedge$  K  $\subseteq$  C)
  proof (rule exI, intro conjI ballI)
    show simplicial_complex (U  $\cup$  ?T)
      unfolding simplicial_complex_def
    proof (intro conjI impI ballI allI)
      show finite (U  $\cup$  ?T)
        using ⟨finite U⟩ ⟨finite  $\mathcal{N}$ ⟩ by simp
      show  $\exists$  n. n simplex S if S  $\in$  U  $\cup$  ?T for S
        using that ahK_C_disjoint in_rel_interior simplU simplex_insert_dimplus1

```

by *fastforce*
show $F \in \mathcal{U} \cup ?\mathcal{T}$ **if** $S: S \in \mathcal{U} \cup ?\mathcal{T} \wedge F \text{ face_of } S$ **for** $F S$
proof –
have $F \in \mathcal{U}$ **if** $S \in \mathcal{U}$
using $S \text{ face}\mathcal{U}$ **that by** *blast*
moreover have $F \in \mathcal{U} \cup ?\mathcal{T}$
if $F \text{ face_of } S C \in \mathcal{N} K \in \mathcal{U}$ **and** $K \subseteq \text{rel_frontier } C$
and $S: S = \text{convex hull insert (SOME } z. z \in \text{rel_interior } C) K$ **for** C
 K
proof –
let $?z = \text{SOME } z. z \in \text{rel_interior } C$
have $?z \in \text{rel_interior } C$
by (*simp add: in_rel_interior* $\langle C \in \mathcal{N} \rangle$)
moreover
obtain I **where** $\neg \text{affine_dependent } I \text{ card } I \leq n \text{ aff_dim } K < \text{int } n K$
 $= \text{convex hull } I$
using $* [OF \langle K \in \mathcal{U} \rangle]$ **by** *auto*
ultimately have $?z \notin \text{affine hull } I$
using $ahK_C \text{ disjoint affine_hull_convex_hull}$ **that by** *blast*
have *compact* I *finite* I
by (*auto simp:* $\langle \neg \text{affine_dependent } I \rangle \text{aff_independent_finite finite_imp_compact}$)
moreover have $F \text{ face_of convex hull insert } ?z I$
by (*metis* $S \langle F \text{ face_of } S \rangle \langle K = \text{convex hull } I \rangle \text{convex_hull_eq_empty convex_hull_insert_segments hull_hull}$)
ultimately obtain J **where** $J: J \subseteq \text{insert } ?z I F = \text{convex hull } J$
using $\text{face_of_convex_hull_subset [of insert } ?z I F]$ **by** *auto*
show *?thesis*
proof (*cases* $?z \in J$)
case *True*
have $F \in (\bigcup K \in \mathcal{U} \cap \text{Pow (rel_frontier } C). \{\text{convex hull insert } ?z K\})$
proof
have $\text{convex hull } (J - \{?z\}) \text{ face_of } K$
by (*metis* $\text{True} \langle J \subseteq \text{insert } ?z I \rangle \langle K = \text{convex hull } I \rangle \langle \neg \text{affine_dependent } I \rangle \text{face_of_convex_hull_affine_independent subset_insert_iff}$)
then have $\text{convex hull } (J - \{?z\}) \in \mathcal{U}$
by (*rule* $\text{face}\mathcal{U} [OF \langle K \in \mathcal{U} \rangle]$)
moreover
have $\bigwedge x. x \in \text{convex hull } (J - \{?z\}) \implies x \in \text{rel_frontier } C$
by (*metis* $\text{True} \langle J \subseteq \text{insert } ?z I \rangle \langle K = \text{convex hull } I \rangle \text{subsetD hull_mono subset_insert_iff that(4)}$)
ultimately show $\text{convex hull } (J - \{?z\}) \in \mathcal{U} \cap \text{Pow (rel_frontier } C)$ **by** *auto*
let $?F = \text{convex hull insert } ?z (\text{convex hull } (J - \{?z\}))$
have $F \subseteq ?F$
by (*simp add:* $\langle F = \text{convex hull } J \rangle \text{hull_mono hull_subset subset_insert_iff}$)
moreover have $?F \subseteq F$
by (*metis* $\text{True} \langle F = \text{convex hull } J \rangle \text{hull_insert insert_Diff}$)


```

set_eq_subset)
  ultimately
  show  $F \in \{?F\}$  by auto
  qed
  with  $\langle C \in \mathcal{N} \rangle$  show ?thesis by auto
  next
  case False
  then have  $F \in \mathcal{U}$ 
  using face_of_convex_hull_affine_independent [OF  $\langle \neg \text{affine\_dependent}$ 
I>]
    by (metis  $J \langle K = \text{convex hull } I \rangle \text{faceU subset\_insert } \langle K \in \mathcal{U} \rangle$ )
  then show  $F \in \mathcal{U} \cup ?\mathcal{T}$ 
    by blast
  qed
  qed
  ultimately show ?thesis
    using that by auto
  qed
  have  $\S: X \cap Y \text{ face\_of } X \wedge X \cap Y \text{ face\_of } Y$ 
  if  $XY: X \in \mathcal{U} \ Y \in ?\mathcal{T}$  for  $X \ Y$ 
  proof -
  obtain  $C \ K$ 
    where  $C \in \mathcal{N} \ K \in \mathcal{U} \ K \subseteq \text{rel\_frontier } C$ 
    and  $Y: Y = \text{convex hull insert } (\text{SOME } z. z \in \text{rel\_interior } C) \ K$ 
    using  $XY$  by blast
  have convex  $C$ 
    by (simp add:  $\langle C \in \mathcal{N} \rangle \text{convexN}$ )
  have  $K \subseteq C$ 
    by (metis DiffE  $\langle C \in \mathcal{N} \rangle \langle K \subseteq \text{rel\_frontier } C \rangle \text{closedN closure\_closed}$ 
rel_frontier_def subset_iff)
  let  $?z = (\text{SOME } z. z \in \text{rel\_interior } C)$ 
  have  $z: ?z \in \text{rel\_interior } C$ 
    using  $\langle C \in \mathcal{N} \rangle \text{in\_rel\_interior}$  by blast
  obtain  $D$  where  $D \in \mathcal{S} \ X \subseteq D$ 
    using  $CU \langle X \in \mathcal{U} \rangle$  by blast
  have  $D \cap \text{rel\_interior } C = (C \cap D) \cap \text{rel\_interior } C$ 
    using rel_interior_subset by blast
  also have  $(C \cap D) \cap \text{rel\_interior } C = \{\}$ 
  proof (rule face_of_disjoint_rel_interior)
  show  $C \cap D \text{ face\_of } C$ 
    using  $\mathcal{N\_def} \ \mathcal{S\_def} \ \langle C \in \mathcal{N} \rangle \ \langle D \in \mathcal{S} \rangle \text{intfaceM}$  by blast
  show  $C \cap D \neq C$ 
    by (metis (mono_tags, lifting) Int_lower2  $\mathcal{N\_def} \ \mathcal{S\_def} \ \langle C \in \mathcal{N} \rangle \ \langle D$ 
 $\in \mathcal{S} \rangle \text{aff\_dim\_subset mem\_Collect\_eq not\_le}$ )
  qed
  finally have  $DC: D \cap \text{rel\_interior } C = \{\}$  .
  have eq:  $X \cap \text{convex hull } (\text{insert } ?z \ K) = X \cap \text{convex hull } K$ 
  proof (rule Int_convex_hull_insert_rel_exterior [OF  $\langle \text{convex } C \rangle \ \langle K \subseteq$ 
 $C \rangle \ z$ ])

```

```

    show disjnt X (rel_interior C)
      using DC by (meson ⟨X ⊆ D⟩ disjnt_def disjnt_subset1)
  qed
  obtain I where I: ¬ affine_dependent I
    and Keq: K = convex_hull I and [simp]: convex_hull K = K
    using * ⟨K ∈ U⟩ by force
  then have ?z ∉ affine_hull I
    using ahK_C_disjoint ⟨C ∈ N⟩ ⟨K ∈ U⟩ ⟨K ⊆ rel_frontier C⟩
  affine_hull_convex_hull z by blast
  have X ∩ K face_of K
    by (simp add: XY(1) ⟨K ∈ U⟩ faceIU inf_commute)
  also have ... face_of_convex_hull_insert ?z K
    by (metis I Keq ⟨?z ∉ affine_hull I⟩ aff_independent_finite_convex_hull_face_of_convex_hull_insert face_of_refl_hull_insert)
  finally have X ∩ K face_of_convex_hull_insert ?z K .
  then show ?thesis
    by (simp add: XY(1) Y ⟨K ∈ U⟩ eq_faceIU)
  qed

show S ∩ S' face_of S
  if S ∈ U ∪ ?T ∧ S' ∈ U ∪ ?T for S S'
  using that
  proof (elim conjE UnE)
    fix X Y
    assume X ∈ U and Y ∈ U
    then show X ∩ Y face_of X
      by (simp add: faceIU)
  next
    fix X Y
    assume XY: X ∈ U Y ∈ ?T
    then show X ∩ Y face_of X Y ∩ X face_of Y
      using § [OF XY] by (auto simp: Int_commute)
  next
    fix X Y
    assume XY: X ∈ ?T Y ∈ ?T
    show X ∩ Y face_of X
    proof -
      obtain C K D L
        where C ∈ N K ∈ U K ⊆ rel_frontier C
          and X: X = convex_hull_insert (SOME z. z ∈ rel_interior C) K
          and D ∈ N L ∈ U L ⊆ rel_frontier D
          and Y: Y = convex_hull_insert (SOME z. z ∈ rel_interior D) L
        using XY by blast
      let ?z = (SOME z. z ∈ rel_interior C)
      have z: ?z ∈ rel_interior C
        using ⟨C ∈ N⟩ in_rel_interior by blast
      have convex C
        by (simp add: ⟨C ∈ N⟩ convexN)
      have convex K

```

```

    using * ⟨K ∈ U⟩ by blast
  have convex L
    by (meson ⟨L ∈ U⟩ convex_simplex simplU)
  show ?thesis
  proof (cases D=C)
    case True
    then have L ⊆ rel_frontier C
      using ⟨L ⊆ rel_frontier D⟩ by auto
    have convex hull insert (SOME z. z ∈ rel_interior C) (K ∩ L) face_of
      convex hull insert (SOME z. z ∈ rel_interior C) K
      by (metis IntI ⟨C ∈ N⟩ ⟨K ∈ U⟩ ⟨K ⊆ rel_frontier C⟩ ⟨L
    ∈ U⟩ ahK_C_disjoint_empty_iff faceIU face_of_polytope_insert2 simplU sim-
    plex_imp_polytope z)
    then show ?thesis
      using True X Y ⟨K ⊆ rel_frontier C⟩ ⟨L ⊆ rel_frontier C⟩ ⟨convex
    C⟩ ⟨convex K⟩ ⟨convex L⟩ convex_hull_insert_Int_eq z by force
  next
    case False
    have convex D
      by (simp add: ⟨D ∈ N⟩ convexN)
    have K ⊆ C
      by (metis DiffE ⟨C ∈ N⟩ ⟨K ⊆ rel_frontier C⟩ closedN closure_closed
    rel_frontier_def subset_eq)
    have L ⊆ D
      by (metis DiffE ⟨D ∈ N⟩ ⟨L ⊆ rel_frontier D⟩ closedN closure_closed
    rel_frontier_def subset_eq)
    let ?w = (SOME w. w ∈ rel_interior D)
    have w: ?w ∈ rel_interior D
      using ⟨D ∈ N⟩ in_rel_interior by blast
    have C ∩ rel_interior D = (D ∩ C) ∩ rel_interior D
      using rel_interior_subset by blast
    also have (D ∩ C) ∩ rel_interior D = {}
    proof (rule face_of_disjoint_rel_interior)
      show D ∩ C face_of D
        using N_def ⟨C ∈ N⟩ ⟨D ∈ N⟩ intfaceM by blast
      have D ∈ M ∧ aff_dim D = int n
        using N_def ⟨D ∈ N⟩ by blast
      moreover have C ∈ M ∧ aff_dim C = int n
        using N_def ⟨C ∈ N⟩ by blast
      ultimately show D ∩ C ≠ D
        by (metis Int_commute False face_of_aff_dim_lt inf.idem inf_le1
    intfaceM not_le polyM polytope_imp_convex)
    qed
    finally have CD: C ∩ (rel_interior D) = {} .
    have zKC: (convex hull insert ?z K) ⊆ C
      by (metis ⟨K ⊆ C⟩ ⟨convex C⟩ in_mono insert_subsetI rel_interior_subset
    subset_hull z)
    have disjnt (convex hull insert (SOME z. z ∈ rel_interior C) K)
    (rel_interior D)

```

```

    using zKC CD by (force simp: disjnt_def)
  then have eq: convex hull (insert ?z K) ∩ convex hull (insert ?w L) =
    convex hull (insert ?z K) ∩ convex hull L
    by (rule Int_convex_hull_insert_rel_exterior [OF ⟨convex D⟩ ⟨L ⊆
D⟩ w])
  have ch_id: convex hull K = K convex hull L = L
    using * ⟨K ∈ U⟩ ⟨L ∈ U⟩ hull_same by auto
  have convex C
    by (simp add: ⟨C ∈ N⟩ convexN)
  have convex hull (insert ?z K) ∩ L = L ∩ convex hull (insert ?z K)
    by blast
  also have ... = convex hull K ∩ L
  proof (subst Int_convex_hull_insert_rel_exterior [OF ⟨convex C⟩ ⟨K
⊆ C⟩ z])
    have (C ∩ D) ∩ rel_interior C = {}
    proof (rule face_of_disjoint_rel_interior)
      show C ∩ D face_of C
        using N_def ⟨C ∈ N⟩ ⟨D ∈ N⟩ intfaceM by blast
      have D ∈ M aff_dim D = int n
        using N_def ⟨D ∈ N⟩ by fastforce+
      moreover have C ∈ M aff_dim C = int n
        using N_def ⟨C ∈ N⟩ by fastforce+
      ultimately have aff_dim D + - 1 * aff_dim C ≤ 0
        by fastforce
      then have ¬ C face_of D
        using False ⟨convex D⟩ face_of_aff_dim_lt by fastforce
      show C ∩ D ≠ C
        by (metis inf_commute ⟨C ∈ M⟩ ⟨D ∈ M⟩ ⟨¬ C face_of D⟩
intfaceM)
    qed
    then have D ∩ rel_interior C = {}
  by (metis inf.absorb_iff2 inf_assoc inf_sup_aci(1) rel_interior_subset)
  then show disjnt L (rel_interior C)
    by (meson ⟨L ⊆ D⟩ disjnt_def disjnt_subset1)
  next
  show L ∩ convex hull K = convex hull K ∩ L
    by force
  qed
  finally have chKL: convex hull (insert ?z K) ∩ L = convex hull K ∩
L .
  have convex hull insert ?z K ∩ convex hull L face_of K
    by (simp add: ⟨K ∈ U⟩ ⟨L ∈ U⟩ ch_id chKL faceIU)
  also have ... face_of convex hull insert ?z K
  proof -
    obtain I where I: ¬ affine_dependent I K = convex hull I
      using * [OF ⟨K ∈ U⟩] by auto
    then have ∧a. a ∉ rel_interior C ∨ a ∉ affine hull I
      using ahK_C_disjoint ⟨C ∈ N⟩ ⟨K ∈ U⟩ ⟨K ⊆ rel_frontier C⟩
affine_hull_convex_hull by blast

```

```

    then show ?thesis
    by (metis I ⟨convex K⟩ aff_independent_finite face_of_convex_hull_insert_eq
face_of_refl hull_insert z)
    qed
    finally have 1: convex_hull_insert ?z K ∩ convex_hull L face_of_convex
hull_insert ?z K .
    have convex_hull_insert ?z K ∩ convex_hull L face_of L
    by (metis ⟨K ∈ U⟩ ⟨L ∈ U⟩ chKL ch_id faceIU inf_commute)
    also have ... face_of_convex_hull_insert ?w L
    proof -
    obtain I where I: ¬ affine_dependent I L = convex_hull I
    using * [OF ⟨L ∈ U⟩] by auto
    then have ∧a. a ∉ rel_interior D ∨ a ∉ affine_hull I
    using ⟨D ∈ N⟩ ⟨L ∈ U⟩ ⟨L ⊆ rel_frontier D⟩ affine_hull_convex_hull
ahK_C_disjoint by blast
    then show ?thesis
    by (metis I ⟨convex L⟩ aff_independent_finite face_of_convex_hull_insert
face_of_refl hull_insert w)
    qed
    finally have 2: convex_hull_insert ?z K ∩ convex_hull L face_of_convex
hull_insert ?w L .
    show ?thesis
    by (simp add: X Y eq 1 2)
    qed
  qed
  qed
  qed
  show ∃ F ⊆ U ∪ ?T. C = ⋃ F if C ∈ M for C
  proof (cases C ∈ S)
  case True
  then show ?thesis
  by (meson UnCI finU subsetD subsetI)
  next
  case False
  then have C ∈ N
  by (simp add: N_def S_def affM less_le that)
  let ?z = SOME z. z ∈ rel_interior C
  have z: ?z ∈ rel_interior C
  using ⟨C ∈ N⟩ in_rel_interior by blast
  let ?F = ⋃ K ∈ U ∩ Pow (rel_frontier C). {convex_hull (insert ?z K)}
  have ?F ⊆ ?T
  using ⟨C ∈ N⟩ by blast
  moreover have C ⊆ ⋃ ?F
  proof
  fix x
  assume x ∈ C
  have convex C
  using ⟨C ∈ N⟩ convexN by blast
  have bounded C

```

```

    using ⟨C ∈ N⟩ by (simp add: polyM polytope_imp_bounded that)
  have polytope C
    using ⟨C ∈ N⟩ polyN by auto
  have ¬ (?z = x ∧ C = {?z})
    using ⟨C ∈ N⟩ aff_dim_sing [of ?z] ⟨¬ n ≤ 1⟩ by (force simp: N_def)
  then obtain y where y: y ∈ rel_frontier C and xzy: x ∈ closed_segment
    ?z y
    and sub: open_segment ?z y ⊆ rel_interior C
    by (blast intro: segment_to_rel_frontier [OF ⟨convex C⟩ ⟨bounded C⟩ z
  ⟨x ∈ C⟩])
  then obtain F where y ∈ F F face_of C F ≠ C
    by (auto simp: rel_frontier_of_polyhedron_alt [OF polytope_imp_polyhedron
  [OF ⟨polytope C⟩]])
  then obtain G where finite G G ⊆ U F = ⋃ G
    by (metis (mono_tags, lifting) S_def ⟨C ∈ M⟩ ⟨convex C⟩ affM faceM
  face_of_aff_dim_lt finU le_less_trans mem_Collect_eq not_less)
  then obtain K where y ∈ K K ∈ G
    using ⟨y ∈ F⟩ by blast
  moreover have x: x ∈ convex_hull {?z,y}
    using segment_convex_hull xzy by auto
  moreover have convex_hull {?z,y} ⊆ convex_hull insert ?z K
    by (metis (full_types) ⟨y ∈ K⟩ hull_mono empty_subsetI insertCI
  insert_subset)
  moreover have K ∈ U
    using ⟨K ∈ G⟩ ⟨G ⊆ U⟩ by blast
  moreover have K ⊆ rel_frontier C
    using ⟨F = ⋃ G⟩ ⟨F ≠ C⟩ ⟨F face_of C⟩ ⟨K ∈ G⟩ face_of_subset_rel_frontier
  by fastforce
  ultimately show x ∈ ⋃ ?F
    by force
qed
moreover
have convex_hull insert (SOME z. z ∈ rel_interior C) K ⊆ C
  if K ∈ U K ⊆ rel_frontier C for K
proof (rule hull_minimal)
  show insert (SOME z. z ∈ rel_interior C) K ⊆ C
    using that ⟨C ∈ N⟩ in_rel_interior rel_interior_subset
    by (force simp: closure_eq rel_frontier_def closedN)
  show convex C
    by (simp add: ⟨C ∈ N⟩ convexN)
qed
then have ⋃ ?F ⊆ C
  by auto
ultimately show ?thesis
  by blast
qed
have (∃ C. C ∈ M ∧ L ⊆ C) ∧ aff_dim L ≤ int n if L ∈ U ∪ ?T for L
  using that
proof

```

```

    assume  $L \in \mathcal{U}$ 
  then show ?thesis
    using  $CU\ S\_def$  * by fastforce
next
  assume  $L \in ?\mathcal{T}$ 
  then obtain  $C\ K$  where  $C \in \mathcal{N}$ 
    and  $L: L = \text{convex hull insert } (SOME\ z. z \in \text{rel\_interior } C)\ K$ 
    and  $K: K \in \mathcal{U}\ K \subseteq \text{rel\_frontier } C$ 
    by auto
  then have  $\text{convex hull } C = C$ 
    by (meson convexN convex_hull_eq)
  then have convex  $C$ 
    by (metis (no_types) convex_convex_hull)
  have  $\text{rel\_frontier } C \subseteq C$ 
    by (metis DiffE closedN  $\langle C \in \mathcal{N} \rangle$  closure_closed rel_frontier_def subsetI)
  have  $K \subseteq C$ 
    using  $K \langle \text{rel\_frontier } C \subseteq C \rangle$  by blast
  have  $C \in \mathcal{M}$ 
    using  $\mathcal{N}\_def \langle C \in \mathcal{N} \rangle$  by auto
  moreover have  $L \subseteq C$ 
    using  $K\ L \langle C \in \mathcal{N} \rangle$ 
    by (metis  $\langle K \subseteq C \rangle \langle \text{convex hull } C = C \rangle$  contra_subsetD hull_mono
in_rel_interior_insert_subset_rel_interior_subset)
  ultimately show ?thesis
    using  $\langle \text{rel\_frontier } C \subseteq C \rangle \langle L \subseteq C \rangle$  affM aff_dim_subset  $\langle C \in \mathcal{M} \rangle$ 
dual_order.trans by blast
  qed
  then show  $\exists C. C \in \mathcal{M} \wedge L \subseteq C$  aff_dim  $L \leq \text{int } n$  if  $L \in \mathcal{U} \cup ?\mathcal{T}$  for  $L$ 
    using that by auto
  qed
then show ?thesis
  apply (rule ex_forward, safe)
  apply (meson Union_iff subsetCE, fastforce)
  by (meson infinite_super simplicial_complex_def)
qed
qed

```

lemma simplicial_subdivision_of_cell_complex_lowdim:

```

  assumes finite  $\mathcal{M}$ 
    and poly:  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and face:  $\bigwedge C1\ C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
    and aff:  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C \leq d$ 
  obtains  $\mathcal{T}$  where simplicial_complex  $\mathcal{T} \bigwedge K. K \in \mathcal{T} \implies \text{aff\_dim } K \leq d$ 
     $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
     $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
     $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$ 

```

proof (cases $d \geq 0$)

case True

then obtain n **where** $n: d = \text{of_nat } n$
using `zero_le_imp_eq_int` **by** `blast`
have $\exists \mathcal{T}. \text{simplicial_complex } \mathcal{T} \wedge$
 $(\forall K \in \mathcal{T}. \text{aff_dim } K \leq \text{int } n) \wedge$
 $\bigcup \mathcal{T} = \bigcup (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\}) \wedge$
 $(\forall C \in \bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\}.$
 $\quad \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F) \wedge$
 $(\forall K \in \mathcal{T}. \exists C. C \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\}) \wedge K \subseteq C)$
proof (`rule simplicial_subdivision_aux`)
show `finite` $(\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$
using $\langle \text{finite } \mathcal{M} \rangle$ `poly polyhedron_eq finite_faces polytope_imp polyhedron`
by `fastforce`
show `polytope` F **if** $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$ **for** F
using `poly that face_of_polytope_polytope` **by** `blast`
show `aff_dim` $F \leq \text{int } n$ **if** $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$ **for** F
using `that`
by `clarify` (`metis` n `aff_dim_subset aff_face_of_imp_subset order_trans`)
show $F \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$
if $G \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$ **and** $F \text{ face_of } G$ **for** $F G$
using `that face_of_trans` **by** `blast`
next
fix $F1 F2$
assume $F1 \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$ **and** $F2 \in (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\})$
then obtain $C1 C2$ **where** $C1 \in \mathcal{M} C2 \in \mathcal{M}$ **and** $F: F1 \text{ face_of } C1 F2$
 $\text{face_of } C2$
by `auto`
show $F1 \cap F2 \text{ face_of } F1$
using `face_of_Int_subface` [`OF _ _ F`]
by (`metis` $\langle C1 \in \mathcal{M} \rangle \langle C2 \in \mathcal{M} \rangle$ `face_inf_commute`)
qed
moreover
have $\bigcup (\bigcup C \in \mathcal{M}. \{F. F \text{ face_of } C\}) = \bigcup \mathcal{M}$
using `face_of_imp_subset face` **by** `blast`
ultimately show `?thesis`
using `face_of_imp_subset n`
by (`fastforce intro!`: `that simp add: poly face_of_refl polytope_imp_convex`)
next
case `False`
then have $m1: \bigwedge C. C \in \mathcal{M} \implies \text{aff_dim } C = -1$
by (`metis` `aff aff_dim_empty_eq aff_dim_negative_iff dual_order.trans not_less`)
then have `faceM`: $\bigwedge F S. \llbracket S \in \mathcal{M}; F \text{ face_of } S \rrbracket \implies F \in \mathcal{M}$
by (`metis` `aff_dim_empty face_of_empty`)
show `?thesis`
proof
have $\bigwedge S. S \in \mathcal{M} \implies \exists n. n \text{ simplex } S$
by (`metis` (`no_types`) `m1 aff_dim_empty simplex_minus_1`)
then show `simplicial_complex` \mathcal{M}
by (`auto simp: simplicial_complex_def` $\langle \text{finite } \mathcal{M} \rangle$ `face intro: faceM`)

show $\text{aff_dim } K \leq d$ **if** $K \in \mathcal{M}$ **for** K
by (*simp add: that aff*)
show $\exists F. \text{finite } F \wedge F \subseteq \mathcal{M} \wedge C = \bigcup F$ **if** $C \in \mathcal{M}$ **for** C
using $\langle C \in \mathcal{M} \rangle \text{ equals0I}$ **by** *auto*
show $\exists C. C \in \mathcal{M} \wedge K \subseteq C$ **if** $K \in \mathcal{M}$ **for** K
using $\langle K \in \mathcal{M} \rangle$ **by** *blast*
qed *auto*
qed

proposition *simplicial_subdivision_of_cell_complex*:
assumes *finite* \mathcal{M}
and *poly*: $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$
and *face*: $\bigwedge C1\ C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face_of } C1$
obtains \mathcal{T} **where** *simplicial_complex* \mathcal{T}
 $\bigcup \mathcal{T} = \bigcup \mathcal{M}$
 $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
 $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$
by (*blast intro: simplicial_subdivision_of_cell_complex_lowdim [OF assms aff_dim_le_DIM]*)

corollary *fine_simplicial_subdivision_of_cell_complex*:
assumes $0 < e$ *finite* \mathcal{M}
and *poly*: $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$
and *face*: $\bigwedge C1\ C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face_of } C1$
obtains \mathcal{T} **where** *simplicial_complex* \mathcal{T}
 $\bigwedge K. K \in \mathcal{T} \implies \text{diameter } K < e$
 $\bigcup \mathcal{T} = \bigcup \mathcal{M}$
 $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
 $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$

proof –
obtain \mathcal{N} **where** \mathcal{N} : *finite* $\mathcal{N} \cup \mathcal{N} = \bigcup \mathcal{M}$
and *diapoly*: $\bigwedge X. X \in \mathcal{N} \implies \text{diameter } X < e \wedge X. X \in \mathcal{N} \implies \text{polytope } X$
 X
and $\bigwedge X\ Y. \llbracket X \in \mathcal{N}; Y \in \mathcal{N} \rrbracket \implies X \cap Y \text{ face_of } X$
and \mathcal{N} *covers*: $\bigwedge C\ x. C \in \mathcal{M} \wedge x \in C \implies \exists D. D \in \mathcal{N} \wedge x \in D \wedge D \subseteq C$
 $\subseteq C$
and \mathcal{N} *covered*: $\bigwedge C. C \in \mathcal{N} \implies \exists D. D \in \mathcal{M} \wedge C \subseteq D$
by (*blast intro: cell_complex_subdivision_exists [OF <0 < e> <finite M> poly aff_dim_le_DIM face]*)
then obtain \mathcal{T} **where** \mathcal{T} : *simplicial_complex* $\mathcal{T} \cup \mathcal{T} = \bigcup \mathcal{N}$
and \mathcal{T} *covers*: $\bigwedge C. C \in \mathcal{N} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
and \mathcal{T} *covered*: $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{N} \wedge K \subseteq C$
using *simplicial_subdivision_of_cell_complex* [*OF* $\langle \text{finite } \mathcal{N} \rangle$] **by** *metis*
show *?thesis*
proof
show *simplicial_complex* \mathcal{T}
by (*rule* \mathcal{T})
show *diameter* $K < e$ **if** $K \in \mathcal{T}$ **for** K
by (*metis le_less_trans diapoly* \mathcal{T} *covered diameter_subset polytope_imp_bounded that*)

```

show  $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
  by (simp add:  $\mathcal{N}(2)$   $\langle \bigcup \mathcal{T} = \bigcup \mathcal{N} \rangle$ )
show  $\exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$  if  $C \in \mathcal{M}$  for  $C$ 
proof -
  { fix  $x$ 
    assume  $x \in C$ 
    then obtain  $D$  where  $D \in \mathcal{T} \ x \in D \ D \subseteq C$ 
      using  $\mathcal{N}$  covers  $\langle C \in \mathcal{M} \rangle$   $\mathcal{T}$  covers by force
    then have  $\exists X \in \mathcal{T} \cap \text{Pow } C. x \in X$ 
      using  $\langle D \in \mathcal{T} \rangle \langle D \subseteq C \rangle \langle x \in D \rangle$  by blast
  }
  moreover
  have finite  $(\mathcal{T} \cap \text{Pow } C)$ 
    using  $\langle \text{simplicial\_complex } \mathcal{T} \rangle$  simplicial_complex_def by auto
  ultimately show ?thesis
    by (rule_tac  $x=(\mathcal{T} \cap \text{Pow } C)$  in exI) auto
qed
show  $\exists C. C \in \mathcal{M} \wedge K \subseteq C$  if  $K \in \mathcal{T}$  for  $K$ 
  by (meson  $\mathcal{N}$  covered  $\mathcal{T}$  covered order_trans that)
qed
qed

```

9.24.20 Some results on cell division with full-dimensional cells only

```

lemma convex_Union_fulldim_cells:
  assumes finite  $\mathcal{S}$  and clo:  $\bigwedge C. C \in \mathcal{S} \implies \text{closed } C$  and con:  $\bigwedge C. C \in \mathcal{S} \implies$ 
  convex  $C$ 
  and eq:  $\bigcup \mathcal{S} = U$  and convex  $U$ 
  shows  $\bigcup \{C \in \mathcal{S}. \text{aff\_dim } C = \text{aff\_dim } U\} = U$  (is ?lhs =  $U$ )
proof -
  have closed  $U$ 
    using  $\langle \text{finite } \mathcal{S} \rangle$  clo eq by blast
  have ?lhs  $\subseteq U$ 
    using eq by blast
  moreover have  $U \subseteq ?lhs$ 
  proof (cases  $\forall C \in \mathcal{S}. \text{aff\_dim } C = \text{aff\_dim } U$ )
    case True
      then show ?thesis
        using eq by blast
    next
      case False
        have closed ?lhs
          by (simp add:  $\langle \text{finite } \mathcal{S} \rangle$  clo closed_Union)
        moreover have  $U \subseteq \text{closure } ?lhs$ 
        proof -
          have  $U \subseteq \text{closure}(\bigcap \{U - C \mid C. C \in \mathcal{S} \wedge \text{aff\_dim } C < \text{aff\_dim } U\})$ 
          proof (rule Baire [OF  $\langle \text{closed } U \rangle$ ])
            show countable  $\{U - C \mid C. C \in \mathcal{S} \wedge \text{aff\_dim } C < \text{aff\_dim } U\}$ 

```

```

    using ⟨finite S⟩ uncountable_infinite by fastforce
  have  $\bigwedge C. C \in \mathcal{S} \implies \text{openin } (\text{top\_of\_set } U) (U - C)$ 
    by (metis Sup_upper clo closed_limpt closedin_limpt eq openin_diff
openin_subtopology_self)
  then show  $\text{openin } (\text{top\_of\_set } U) T \wedge U \subseteq \text{closure } T$ 
    if  $T \in \{U - C \mid C. C \in \mathcal{S} \wedge \text{aff\_dim } C < \text{aff\_dim } U\}$  for T
    using that dense_complement_convex_closed ⟨closed U⟩ ⟨convex U⟩ by
auto
  qed
  also have  $\dots \subseteq \text{closure } ?\text{lhs}$ 
  proof -
    obtain C where  $C \in \mathcal{S} \text{ aff\_dim } C < \text{aff\_dim } U$ 
      by (metis False Sup_upper aff_dim_subset eq eq_iff not_le)
    then have  $\exists X. X \in \mathcal{S} \wedge \text{aff\_dim } X = \text{aff\_dim } U \wedge x \in X$ 
      if  $\bigwedge V. (\exists C. V = U - C \wedge C \in \mathcal{S} \wedge \text{aff\_dim } C < \text{aff\_dim } U) \implies x \in V$  for x
      by (metis Diff_iff Sup_upper UnionE aff_dim_subset eq order_less_le
that)
    then show ?thesis
      by (auto intro!: closure_mono)
  qed
  finally show ?thesis .
  qed
  ultimately show ?thesis
    using closure_subset_eq by blast
  qed
  ultimately show ?thesis by blast
  qed

```

proposition *fine_triangular_subdivision_of_cell_complex:*

```

  assumes  $0 < e$  finite M
    and poly:  $\bigwedge C. C \in \mathcal{M} \implies \text{polytope } C$ 
    and aff:  $\bigwedge C. C \in \mathcal{M} \implies \text{aff\_dim } C = d$ 
    and face:  $\bigwedge C1 C2. \llbracket C1 \in \mathcal{M}; C2 \in \mathcal{M} \rrbracket \implies C1 \cap C2 \text{ face\_of } C1$ 
  obtains  $\mathcal{T}$  where triangulation  $\mathcal{T} \bigwedge k. k \in \mathcal{T} \implies \text{diameter } k < e$ 
     $\bigwedge k. k \in \mathcal{T} \implies \text{aff\_dim } k = d \bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
     $\bigwedge C. C \in \mathcal{M} \implies \exists f. \text{finite } f \wedge f \subseteq \mathcal{T} \wedge C = \bigcup f$ 
     $\bigwedge k. k \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge k \subseteq C$ 

```

proof –

```

  obtain  $\mathcal{T}$  where simplicial_complex  $\mathcal{T}$ 
    and dia $\mathcal{T}$ :  $\bigwedge K. K \in \mathcal{T} \implies \text{diameter } K < e$ 
    and  $\bigcup \mathcal{T} = \bigcup \mathcal{M}$ 
    and in $\mathcal{M}$ :  $\bigwedge C. C \in \mathcal{M} \implies \exists F. \text{finite } F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ 
    and in $\mathcal{T}$ :  $\bigwedge K. K \in \mathcal{T} \implies \exists C. C \in \mathcal{M} \wedge K \subseteq C$ 
    by (blast intro: fine_simplicial_subdivision_of_cell_complex [OF ⟨e > 0⟩
⟨finite M⟩ poly face])
  let ? $\mathcal{T} = \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$ 
  show thesis
  proof

```

```

show triangulation ? $\mathcal{T}$ 
  using  $\langle$ simplicial_complex  $\mathcal{T}\rangle$  by (auto simp: triangulation_def simplicial_complex_def)
show diameter  $L < e$  if  $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$  for  $L$ 
  using that by (auto simp: dia $\mathcal{T}$ )
show aff_dim  $L = d$  if  $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$  for  $L$ 
  using that by auto
show  $\exists F. \text{finite } F \wedge F \subseteq \{K \in \mathcal{T}. \text{aff\_dim } K = d\} \wedge C = \bigcup F$  if  $C \in \mathcal{M}$ 
for  $C$ 
proof –
  obtain  $F$  where finite  $F$   $F \subseteq \mathcal{T}$   $C = \bigcup F$ 
  using in $\mathcal{M}$  [ $OF \langle C \in \mathcal{M} \rangle$ ] by auto
  show ?thesis
  proof (intro exI conjI)
    show finite  $\{K \in F. \text{aff\_dim } K = d\}$ 
    by (simp add:  $\langle$ finite  $F\rangle$ )
    show  $\{K \in F. \text{aff\_dim } K = d\} \subseteq \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$ 
    using  $\langle F \subseteq \mathcal{T} \rangle$  by blast
    have  $d = \text{aff\_dim } C$ 
    by (simp add: aff that)
    moreover have  $\bigwedge K. K \in F \implies \text{closed } K \wedge \text{convex } K$ 
    using  $\langle$ simplicial_complex  $\mathcal{T}\rangle$   $\langle F \subseteq \mathcal{T} \rangle$ 
    unfolding simplicial_complex_def by (metis subsetCE  $\langle F \subseteq \mathcal{T} \rangle$  closed_simplex_convex_simplex)
    moreover have convex  $(\bigcup F)$ 
    using  $\langle C = \bigcup F \rangle$  poly polytope_imp_convex that by blast
    ultimately show  $C = \bigcup \{K \in F. \text{aff\_dim } K = d\}$ 
    by (simp add: convex_Union_fulldim_cells  $\langle C = \bigcup F \rangle$   $\langle$ finite  $F\rangle$ )
  qed
qed
then show  $\bigcup \{K \in \mathcal{T}. \text{aff\_dim } K = d\} = \bigcup \mathcal{M}$ 
by auto (meson in $\mathcal{T}$  subsetCE)
show  $\exists C. C \in \mathcal{M} \wedge L \subseteq C$ 
if  $L \in \{K \in \mathcal{T}. \text{aff\_dim } K = d\}$  for  $L$ 
using that by (auto simp: in $\mathcal{T}$ )
qed
qed

```

9.25 Finitely generated cone is polyhedral, and hence closed

```

proposition polyhedron_convex_cone_hull:
  fixes  $S :: 'a::\text{euclidean\_space}$  set
  assumes finite  $S$ 
  shows polyhedron(convex_cone hull  $S$ )
proof (cases  $S = \{\}$ )
  case True
  then show ?thesis

```

```

    by (simp add: affine_imp_polyhedron)
next
case False
then have polyhedron(convex hull (insert 0 S))
  by (simp add: assms polyhedron_convex_hull)
then obtain F a b where finite F
  and F: convex hull (insert 0 S) =  $\bigcap$  F
  and ab:  $\bigwedge h. h \in F \implies a h \neq 0 \wedge h = \{x. a h \cdot x \leq b h\}$ 
  unfolding polyhedron_def by metis
then have F  $\neq$  {}
  by (metis bounded_convex_hull finite_imp_bounded Inf_empty assms finite_insert
not_bounded_UNIV)
show ?thesis
  unfolding polyhedron_def
proof (intro exI conjI)
  show convex_cone hull S =  $\bigcap \{h \in F. b h = 0\}$  (is ?lhs = ?rhs)
proof
  show ?lhs  $\subseteq$  ?rhs
proof (rule hull_minimal)
  show S  $\subseteq$   $\bigcap \{h \in F. b h = 0\}$ 
  by (smt (verit, best) F InterE InterI hull_subset insert_subset mem_Collect_eq
subset_eq)
  have  $\bigwedge S. \llbracket S \in F; b S = 0 \rrbracket \implies$  convex_cone S
  by (metis ab convex_cone_halfspace_le)
  then show convex_cone ( $\bigcap \{h \in F. b h = 0\}$ )
  by (force intro: convex_cone_Inter)
qed
have x  $\in$  convex_cone hull S
  if x:  $\bigwedge h. \llbracket h \in F; b h = 0 \rrbracket \implies x \in h$  for x
proof -
  have  $\exists t. 0 < t \wedge (t *_R x) \in h$  if h  $\in$  F for h
  proof (cases b h = 0)
  case True
  then show ?thesis
  by (metis x linordered_field_no_ub mult_1 scaleR_one that zero_less_mult_iff)
next
case False
then have b h > 0
  by (smt (verit, del_insts) F InterE ab hull_subset inner_zero_right
insert_subset mem_Collect_eq that)
then have 0  $\in$  interior  $\{x. a h \cdot x \leq b h\}$ 
  by (simp add: ab that)
then have 0  $\in$  interior h
  using ab that by auto
then obtain  $\varepsilon$  where 0 <  $\varepsilon$  and  $\varepsilon$ : ball 0  $\varepsilon \subseteq$  h
  using mem_interior by blast
show ?thesis
proof (cases x=0)
  case True

```

```

    then show ?thesis
      using  $\varepsilon \langle 0 < \varepsilon \rangle$  by auto
    next
      case False
      with  $\varepsilon \langle 0 < \varepsilon \rangle$  show ?thesis
        by (rule_tac  $x = \varepsilon / (2 * \text{norm } x)$  in exI) (auto simp: divide_simps)
      qed
    qed
  then obtain  $t$  where  $t: \bigwedge h. h \in F \implies 0 < t h \wedge (t h *_R x) \in h$ 
    by metis
  then have  $\text{Inf } (t \text{ ' } F) *_R x /_R \text{Inf } (t \text{ ' } F) = x$ 
    by (smt (verit)  $\langle F \neq \{\} \rangle \langle \text{finite } F \rangle \text{divideR\_right finite\_imageI finite\_less\_Inf\_iff image\_iff image\_is\_empty}$ )
  moreover have  $\text{Inf } (t \text{ ' } F) *_R x /_R \text{Inf } (t \text{ ' } F) \in \text{convex\_cone hull } S$ 
  proof (rule conicD [OF conic_convex_cone_hull])
    have  $\text{Inf } (t \text{ ' } F) *_R x \in \bigcap F$ 
    proof clarify
      fix  $h$ 
      assume  $h \in F$ 
      have eq:  $\text{Inf } (t \text{ ' } F) *_R x = (1 - \text{Inf}(t \text{ ' } F) / t h) *_R 0 + (\text{Inf}(t \text{ ' } F) / t h) *_R t h *_R x$ 
      using  $\langle h \in F \rangle t$  by force
      show  $\text{Inf } (t \text{ ' } F) *_R x \in h$ 
      unfolding eq
      proof (rule convexD_alt)
        have  $h = \{x. a h \cdot x \leq b h\}$ 
        by (simp add:  $\langle h \in F \rangle ab$ )
        then show convex  $h$ 
        by (metis convex_halfspace_le)
      show  $0 \in h$ 
      by (metis F InterE  $\langle h \in F \rangle \text{hull\_subset insertCI subsetD}$ )
      show  $t h *_R x \in h$ 
      by (simp add:  $\langle h \in F \rangle t$ )
      show  $0 \leq \text{Inf } (t \text{ ' } F) / t h$ 
      by (metis  $\langle F \neq \{\} \rangle \langle h \in F \rangle \text{cINF\_greatest divide\_nonneg\_pos less\_eq\_real\_def } t$ )
      show  $\text{Inf } (t \text{ ' } F) / t h \leq 1$ 
      by (simp add:  $\langle \text{finite } F \rangle \langle h \in F \rangle \text{cInf\_le\_finite } t$ )
    qed
  qed
  moreover have  $\text{convex hull } (\text{insert } 0 S) \subseteq \text{convex\_cone hull } S$ 
  by (simp add: convex_cone_hull_contains_0 convex_convex_cone_hull hull_minimal hull_subset)
  ultimately show  $\text{Inf } (t \text{ ' } F) *_R x \in \text{convex\_cone hull } S$ 
  using F by blast
  show  $0 \leq \text{inverse } (\text{Inf } (t \text{ ' } F))$ 
  using  $t$  by (simp add:  $\langle F \neq \{\} \rangle \langle \text{finite } F \rangle \text{finite\_less\_Inf\_iff less\_eq\_real\_def}$ )
  qed
  ultimately show ?thesis

```

```

      by auto
    qed
  then show ?rhs  $\subseteq$  ?lhs
    by auto
  qed
  show  $\forall h \in \{h \in F. b \cdot h = 0\}. \exists a \cdot b. a \neq 0 \wedge h = \{x. a \cdot x \leq b\}$ 
    using ab by blast
  qed (auto simp: ‹finite F›)
qed

```

```

lemma closed_convex_cone_hull:
  fixes S :: 'a::euclidean_space set
  shows finite S  $\implies$  closed(convex_cone hull S)
  by (simp add: polyhedron_convex_cone_hull polyhedron_imp_closed)

```

```

lemma polyhedron_convex_cone_hull_polytope:
  fixes S :: 'a::euclidean_space set
  shows polytope S  $\implies$  polyhedron(convex_cone hull S)
  by (metis convex_cone_hull_separate hull_hull polyhedron_convex_cone_hull
  polytope_def)

```

```

lemma polyhedron_conic_hull_polytope:
  fixes S :: 'a::euclidean_space set
  shows polytope S  $\implies$  polyhedron(conic hull S)
  by (metis conic_hull_eq_empty convex_cone_hull_separate_nonempty hull_hull
  polyhedron_convex_cone_hull_polytope polyhedron_empty polytope_def)

```

```

lemma closed_conic_hull_strong:
  fixes S :: 'a::euclidean_space set
  shows  $0 \in \text{rel\_interior } S \vee \text{polytope } S \vee \text{compact } S \wedge \sim(0 \in S) \implies \text{closed}(\text{conic hull } S)$ 
  using closed_conic_hull polyhedron_conic_hull_polytope polyhedron_imp_closed
  by blast

```

end

9.26 Absolute Retracts, Absolute Neighbourhood Retracts and Euclidean Neighbourhood Retracts

```

theory Retracts
imports
  Brouwer_Fixpoint
  Continuous_Extension
begin

```

Absolute retracts (AR), absolute neighbourhood retracts (ANR) and also

Euclidean neighbourhood retracts (ENR). We define AR and ANR by specializing the standard definitions for a set to embedding in spaces of higher dimension.

John Harrison writes: "This turns out to be sufficient (since any set in \mathbb{R}^n can be embedded as a closed subset of a convex subset of \mathbb{R}^{n+1}) to derive the usual definitions, but we need to split them into two implications because of the lack of type quantifiers. Then ENR turns out to be equivalent to ANR plus local compactness."

definition $AR :: 'a::topological_space\ set \Rightarrow bool\ \mathbf{where}$

$AR\ S \equiv \forall U. \forall S'::('a * real)\ set.$

$S\ \text{homeomorphic}\ S' \wedge \text{closedin}\ (\text{top_of_set}\ U)\ S' \longrightarrow S'\ \text{retract_of}\ U$

definition $ANR :: 'a::topological_space\ set \Rightarrow bool\ \mathbf{where}$

$ANR\ S \equiv \forall U. \forall S'::('a * real)\ set.$

$S\ \text{homeomorphic}\ S' \wedge \text{closedin}\ (\text{top_of_set}\ U)\ S' \\ \longrightarrow (\exists T. \text{openin}\ (\text{top_of_set}\ U)\ T \wedge S'\ \text{retract_of}\ T)$

definition $ENR :: 'a::topological_space\ set \Rightarrow bool\ \mathbf{where}$

$ENR\ S \equiv \exists U. \text{open}\ U \wedge S\ \text{retract_of}\ U$

First, show that we do indeed get the "usual" properties of ARs and ANRs.

lemma $AR_imp_absolute_extensor:$

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes $AR\ S$ **and** $\text{contf}: \text{continuous_on}\ T\ f$ **and** $f\ ' T \subseteq S$

and $\text{cloUT}: \text{closedin}\ (\text{top_of_set}\ U)\ T$

obtains g **where** $\text{continuous_on}\ U\ g\ g\ ' U \subseteq S \wedge x. x \in T \Longrightarrow g\ x = f\ x$

proof –

have $\text{aff_dim}\ S < \text{int}\ (\text{DIM}\ ('b \times \text{real}))$

using $\text{aff_dim_le_DIM}\ [\text{of}\ S]$ **by** simp

then obtain C **and** $S' :: ('b * real)\ set$

where $C: \text{convex}\ C\ C \neq \{\}$

and $\text{cloCS}: \text{closedin}\ (\text{top_of_set}\ C)\ S'$

and $\text{hom}: S\ \text{homeomorphic}\ S'$

by $(\text{metis}\ \text{that}\ \text{homeomorphic_closedin_convex})$

then have $S'\ \text{retract_of}\ C$

using $\langle AR\ S \rangle$ **by** $(\text{simp}\ \text{add}: AR_def)$

then obtain r **where** $S' \subseteq C$ **and** $\text{contr}: \text{continuous_on}\ C\ r$

and $r\ ' C \subseteq S'$ **and** $\text{rid}: \bigwedge x. x \in S' \Longrightarrow r\ x = x$

by $(\text{auto}\ \text{simp}: \text{retraction_def}\ \text{retract_of_def})$

obtain $g\ h$ **where** $\text{homeomorphism}\ S\ S'\ g\ h$

using hom **by** $(\text{force}\ \text{simp}: \text{homeomorphic_def})$

then have $\text{continuous_on}\ (f\ ' T)\ g$

by $(\text{meson}\ \langle f\ ' T \subseteq S \rangle\ \text{continuous_on_subset}\ \text{homeomorphism_def})$

then have $\text{contgf}: \text{continuous_on}\ T\ (g \circ f)$

by $(\text{metis}\ \text{continuous_on_compose}\ \text{contf})$

have $gfTC: (g \circ f)\ ' T \subseteq C$

proof –

have $g\ ' S = S'$


```

    by (metis (no_types) ‹homeomorphism S S' g h› homeomorphism_def)
  with ‹S' ⊆ C› ‹f' ‹ T ⊆ S› show ?thesis by force
qed
obtain f' where f': continuous_on U f' f' ‹ U ⊆ C
      ∧ x. x ∈ T ⇒ f' x = (g ∘ f) x
  by (metis Dugundji [OF C cloUT contgf gfTC])
show ?thesis
proof (rule_tac g = h ∘ r ∘ f' in that)
  show continuous_on U (h ∘ r ∘ f')
  proof (intro continuous_on_compose f')
    show continuous_on (f' ‹ U) r
      using continuous_on_subset contr f' by blast
    show continuous_on (r ‹ f' ‹ U) h
      using ‹homeomorphism S S' g h› ‹f' ‹ U ⊆ C›
      unfolding homeomorphism_def
      by (metis ‹r ‹ C ⊆ S'› continuous_on_subset image_mono)
  qed
  show (h ∘ r ∘ f') ‹ U ⊆ S
    using ‹homeomorphism S S' g h› ‹r ‹ C ⊆ S'› ‹f' ‹ U ⊆ C›
    by (fastforce simp: homeomorphism_def)
  show ∧x. x ∈ T ⇒ (h ∘ r ∘ f') x = f x
    using ‹homeomorphism S S' g h› ‹f' ‹ T ⊆ S› f'
    by (auto simp: rid homeomorphism_def)
  qed
qed
qed

lemma AR_imp_absolute_retract:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes AR S S homeomorphic S'
    and clo: closedin (top_of_set U) S'
  shows S' retract_of U
proof -
  obtain g h where hom: homeomorphism S S' g h
    using assms by (force simp: homeomorphic_def)
  obtain h: continuous_on S' h h ‹ S' ⊆ S
    using hom homeomorphism_def by blast
  obtain h' where h': continuous_on U h' h' ‹ U ⊆ S
    and h'h: ∧x. x ∈ S' ⇒ h' x = h x
  by (blast intro: AR_imp_absolute_extensor [OF ‹AR S› h clo])
  have [simp]: S' ⊆ U using clo closedin_limpt by blast
  show ?thesis
proof (simp add: retraction_def retract_of_def, intro exI conjI)
  show continuous_on U (g ∘ h')
    by (meson continuous_on_compose continuous_on_subset h' hom homeo-
        morphism_cont1)
  show (g ∘ h') ∈ U → S'
    using h' by clarsimp (metis hom subsetD homeomorphism_def imageI)
  show ∀x∈S'. (g ∘ h') x = x
    by clarsimp (metis h'h hom homeomorphism_def)

```

3510

qed
qed

lemma *AR_imp_absolute_retract_UNIV*:
fixes $S :: 'a::\text{euclidean_space set}$ and $S' :: 'b::\text{euclidean_space set}$
assumes $AR\ S\ S\ \text{homeomorphic}\ S'\ \text{closed}\ S'$
shows S' retract_of UNIV
using *AR_imp_absolute_retract* *assms* by *fastforce*

lemma *absolute_extensor_imp_AR*:
fixes $S :: 'a::\text{euclidean_space set}$
assumes $\bigwedge f :: 'a * \text{real} \Rightarrow 'a.$
 $\bigwedge U\ T. \llbracket \text{continuous_on}\ T\ f; f\ ' T \subseteq S;$
 $\text{closedin}\ (\text{top_of_set}\ U)\ T \rrbracket$
 $\implies \exists g. \text{continuous_on}\ U\ g \wedge g\ ' U \subseteq S \wedge (\forall x \in T. g\ x = f\ x)$
shows $AR\ S$

proof (*clarsimp simp: AR_def*)
fix U and $T :: ('a * \text{real})\ \text{set}$
assume $S\ \text{homeomorphic}\ T$ and $\text{clo}: \text{closedin}\ (\text{top_of_set}\ U)\ T$
then obtain $g\ h$ where $\text{hom}: \text{homeomorphism}\ S\ T\ g\ h$
by (*force simp: homeomorphic_def*)
obtain $h: \text{continuous_on}\ T\ h\ h\ ' T \subseteq S$
using $\text{hom}\ \text{homeomorphism_def}$ by *blast*
obtain h' where $h': \text{continuous_on}\ U\ h'\ h'\ ' U \subseteq S$
and $h'h: \forall x \in T. h'\ x = h\ x$
using *assms* [*OF* $h\ \text{clo}$] by *blast*
have [*simp*]: $T \subseteq U$
using $\text{clo}\ \text{closedin_imp_subset}$ by *auto*
show T retract_of U
proof (*simp add: retraction_def retract_of_def, intro exI conjI*)
show $\text{continuous_on}\ U\ (g \circ h')$
by (*meson continuous_on_compose continuous_on_subset h' hom homeomorphism_cont1*)
show $(g \circ h') \in U \rightarrow T$
using h' by *clarsimp* (*metis hom subsetD homeomorphism_def imageI*)
show $\forall x \in T. (g \circ h')\ x = x$
by *clarsimp* (*metis h'h hom homeomorphism_def*)
qed
qed

lemma *AR_eq_absolute_extensor*:
fixes $S :: 'a::\text{euclidean_space set}$
shows $AR\ S \iff$
 $(\forall f :: 'a * \text{real} \Rightarrow 'a.$
 $\forall U\ T. \text{continuous_on}\ T\ f \implies f\ ' T \subseteq S \implies$
 $\text{closedin}\ (\text{top_of_set}\ U)\ T \implies$
 $(\exists g. \text{continuous_on}\ U\ g \wedge g\ ' U \subseteq S \wedge (\forall x \in T. g\ x = f\ x)))$
by (*metis* (*mono_tags, opaque_lifting*) *AR_imp_absolute_extensor absolute_extensor_imp_AR*)

```

lemma AR_imp_retract:
  fixes S :: 'a::euclidean_space set
  assumes AR S  $\wedge$  closedin (top_of_set U) S
  shows S retract_of U
using AR_imp_absolute_retract assms homeomorphic_refl by blast

lemma AR_homeomorphic_AR:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes AR T S homeomorphic T
  shows AR S
unfolding AR_def
by (metis assms AR_imp_absolute_retract homeomorphic_trans [of _ S] homeomorphic_sym)

lemma homeomorphic_AR_iff_AR:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  shows S homeomorphic T  $\implies$  AR S  $\longleftrightarrow$  AR T
by (metis AR_homeomorphic_AR homeomorphic_sym)

lemma ANR_imp_absolute_neighbourhood_extensor:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes ANR S and contf: continuous_on T f and f  $\in$  T  $\rightarrow$  S
  and cloUT: closedin (top_of_set U) T
  obtains V g where T  $\subseteq$  V openin (top_of_set U) V
  continuous_on V g
  g  $\in$  V  $\rightarrow$  S  $\wedge$  x. x  $\in$  T  $\implies$  g x = f x
proof -
  have aff_dim S < int (DIM('b  $\times$  real))
  using aff_dim_le_DIM [of S] by simp
  then obtain C and S' :: ('b * real) set
  where C: convex C C  $\neq$  {}
  and cloCS: closedin (top_of_set U) C S'
  and hom: S homeomorphic S'
  by (metis that homeomorphic_closedin_convex)
  then obtain D where opD: openin (top_of_set U) D and S' retract_of D
  using  $\langle$ ANR S $\rangle$  by (auto simp: ANR_def)
  then obtain r where S'  $\subseteq$  D and contr: continuous_on D r
  and r' D  $\subseteq$  S' and rid:  $\wedge$ x. x  $\in$  S'  $\implies$  r x = x
  by (auto simp: retraction_def retract_of_def)
  obtain g h where homgh: homeomorphism S S' g h
  using hom by (force simp: homeomorphic_def)
  have continuous_on (f ' T) g
  by (metis PiE assms(3) continuous_on_subset homeomorphism_cont1 homgh
  image_subset_iff)
  then have contgf: continuous_on T (g  $\circ$  f)
  by (intro continuous_on_compose contf)
  have gfTC: (g  $\circ$  f) ' T  $\subseteq$  C
  proof -

```

```

have g ' S = S'
  by (metis (no_types) homeomorphism_def homgh)
then show ?thesis
  by (metis PiE assms(3) cloCS closedin_def image_comp image_mono im-
age_subset_iff order.trans topspace_euclidean_subtopology)
qed
obtain f' where contf': continuous_on U f'
  and f' ' U  $\subseteq$  C
  and eq:  $\bigwedge x. x \in T \implies f' x = (g \circ f) x$ 
  by (metis Dugundji [OF C cloUT contgf gfTC])
show ?thesis
proof (rule_tac V = U  $\cap$  f' - ' D and g = h  $\circ$  r  $\circ$  f' in that)
  show T  $\subseteq$  U  $\cap$  f' - ' D
    using cloUT closedin_imp_subset  $\langle S' \subseteq D \rangle \langle f \in T \rightarrow S \rangle$  eq homeomor-
phism_image1 homgh
    by fastforce
  show ope: openin (top_of_set U) (U  $\cap$  f' - ' D)
  by (meson  $\langle f' ' U \subseteq C \rangle$  contf' continuous_openin_preimage image_subset_iff_funcset
opD)
  have conth: continuous_on (r ' f' ' (U  $\cap$  f' - ' D)) h
  proof (rule continuous_on_subset [of S'])
    show continuous_on S' h
      using homeomorphism_def homgh by blast
  qed (use  $\langle r ' D \subseteq S' \rangle$  in blast)
  show continuous_on (U  $\cap$  f' - ' D) (h  $\circ$  r  $\circ$  f')
    by (blast intro: continuous_on_compose conth continuous_on_subset [OF
contr] continuous_on_subset [OF contf'])
  show (h  $\circ$  r  $\circ$  f')  $\in$  (U  $\cap$  f' - ' D)  $\rightarrow$  S
    using  $\langle$ homeomorphism S S' g h $\rangle$   $\langle f' ' U \subseteq C \rangle$   $\langle r ' D \subseteq S' \rangle$ 
    by (auto simp: homeomorphism_def)
  show  $\bigwedge x. x \in T \implies (h \circ r \circ f') x = f x$ 
    using  $\langle$ homeomorphism S S' g h $\rangle$   $\langle f \in T \rightarrow S \rangle$  eq
    by (metis PiE comp_apply homeomorphism_def image_iff rid)
  qed
qed

```

corollary ANR_imp_absolute_neighbourhood_retract:

```

fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
assumes ANR S S homeomorphic S'
  and clo: closedin (top_of_set U) S'
obtains V where openin (top_of_set U) V S' retract_of V
proof -
  obtain g h where hom: homeomorphism S S' g h
    using assms by (force simp: homeomorphic_def)
  obtain h: continuous_on S' h h  $\in$  S'  $\rightarrow$  S
    using hom homeomorphism_def by blast
  from ANR_imp_absolute_neighbourhood_extensor [OF  $\langle$ ANR S $\rangle$  h clo]
  obtain V h' where S'  $\subseteq$  V and opUV: openin (top_of_set U) V

```

```

and h': continuous_on V h' h' ' V ⊆ S
and h'h:  $\bigwedge x. x \in S' \implies h' x = h x$ 
by (blast intro: ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR S› h
clo])
have S' retract_of V
proof (simp add: retraction_def retract_of_def, intro exI conjI ‹S' ⊆ V›)
  show continuous_on V (g ∘ h')
  by (meson continuous_on_compose continuous_on_subset h'(1) h'(2) hom
homeomorphism_cont1)
  show (g ∘ h') ∈ V → S'
  using h' by clarsimp (metis hom subsetD homeomorphism_def imageI)
  show  $\forall x \in S'. (g \circ h') x = x$ 
  by clarsimp (metis h'h hom homeomorphism_def)
qed
then show ?thesis
by (rule that [OF opUV])
qed

```

```

corollary ANR_imp_absolute_neighbourhood_retract_UNIV:
fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
assumes ANR S and hom: S homeomorphic S' and clo: closed S'
obtains V where open V S' retract_of V
using ANR_imp_absolute_neighbourhood_retract [OF ‹ANR S› hom]
by (metis clo closed_closedin open_openin subtopology_UNIV)

```

```

corollary neighbourhood_extension_into_ANR:
fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes contf: continuous_on S f and fim: f ∈ S → T and ANR T closed S
obtains V g where S ⊆ V open V continuous_on V g
  g ∈ V → T  $\bigwedge x. x \in S \implies g x = f x$ 
using ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR T› contf fim]
by (metis ‹closed S› closed_closedin open_openin subtopology_UNIV)

```

```

lemma absolute_neighbourhood_extensor_imp_ANR:
fixes S :: 'a::euclidean_space set
assumes  $\bigwedge f :: 'a * \text{real} \Rightarrow 'a.$ 
   $\bigwedge U T. \llbracket \text{continuous\_on } T f; f \in T \rightarrow S; \text{closedin (top\_of\_set } U) T \rrbracket$ 
   $\implies \exists V g. T \subseteq V \wedge \text{openin (top\_of\_set } U) V \wedge$ 
    continuous_on V g  $\wedge g \in V \rightarrow S \wedge (\forall x \in T. g x = f x)$ 
shows ANR S
proof (clarsimp simp: ANR_def)
fix U and T :: ('a * real) set
assume S homeomorphic T and clo: closedin (top_of_set U) T
then obtain g h where hom: homeomorphism S T g h
  by (force simp: homeomorphic_def)
obtain h: continuous_on T h h ∈ T → S
  using hom homeomorphism_def by blast
obtain V h' where T ⊆ V and opV: openin (top_of_set U) V

```

```

      and h': continuous_on V h' h' ∈ V → S
      and h'h: ∀ x ∈ T. h' x = h x
    using assms [OF h clo] by blast
  have [simp]: T ⊆ U
    using clo closedin_imp_subset by auto
  have T retract_of V
  proof (simp add: retraction_def retract_of_def, intro exI conjI ⟨T ⊆ V⟩)
    show continuous_on V (g ∘ h')
      by (meson continuous_on_compose continuous_on_subset h' hom homeo-
morph_def image_subset_iff_funcset)
    show (g ∘ h') ∈ V → T
      using h' hom homeomorphism_image1 by fastforce
    show ∀ x ∈ T. (g ∘ h') x = x
      by clarsimp (metis h'h hom homeomorphism_def)
  qed
  then show ∃ V. openin (top_of_set U) V ∧ T retract_of V
    using opV by blast
  qed

```

lemma *ANR_eq_absolute_neighbourhood_extensor*:

fixes $S :: 'a::euclidean_space\ set$

shows $ANR\ S \longleftrightarrow$

$(\forall f :: 'a * real \Rightarrow 'a.$

$\forall U\ T. continuous_on\ T\ f \longrightarrow f \in T \rightarrow S \longrightarrow$

$closedin\ (top_of_set\ U)\ T \longrightarrow$

$(\exists V\ g. T \subseteq V \wedge openin\ (top_of_set\ U)\ V \wedge$

$continuous_on\ V\ g \wedge g \in T \rightarrow S \wedge (\forall x \in T. g\ x = f\ x))$) (**is**

$_ = ?rhs$)

proof

assume $ANR\ S$ **then show** $?rhs$

by (*metis ANR_imp_absolute_neighbourhood_extensor*)

qed (*simp add: absolute_neighbourhood_extensor_imp_ANR*)

lemma *ANR_imp_neighbourhood_retract*:

fixes $S :: 'a::euclidean_space\ set$

assumes $ANR\ S\ closedin\ (top_of_set\ U)\ S$

obtains V **where** $openin\ (top_of_set\ U)\ V\ S\ retract_of\ V$

using *ANR_imp_absolute_neighbourhood_retract* *assms* *homeomorphic_refl* **by** *blast*

lemma *ANR_imp_absolute_closed_neighbourhood_retract*:

fixes $S :: 'a::euclidean_space\ set$ **and** $S' :: 'b::euclidean_space\ set$

assumes $ANR\ S\ S\ homeomorphic\ S'$ **and** $US': closedin\ (top_of_set\ U)\ S'$

obtains $V\ W$

where $openin\ (top_of_set\ U)\ V$

$closedin\ (top_of_set\ U)\ W$

$S' \subseteq V\ V \subseteq W\ S'\ retract_of\ W$

proof —

obtain Z **where** $openin\ (top_of_set\ U)\ Z$ **and** $S'Z: S'\ retract_of\ Z$

```

  by (blast intro: assms ANR_imp_absolute_neighbourhood_retract)
  then have UUZ: closedin (top_of_set U) (U - Z)
  by auto
  have S' ∩ (U - Z) = {}
  using ⟨S' retract_of Z⟩ closedin_retract closedin_subtopology by fastforce
  then obtain V W
  where openin (top_of_set U) V
  and openin (top_of_set U) W
  and S' ⊆ V U - Z ⊆ W V ∩ W = {}
  using separation_normal_local [OF US' UUZ] by auto
  moreover have S' retract_of U - W
  proof (rule retract_of_subset [OF S'Z])
  show S' ⊆ U - W
  using US' ⟨S' ⊆ V⟩ ⟨V ∩ W = {}⟩ closedin_subset by fastforce
  show U - W ⊆ Z
  using Diff_subset_conv ⟨U - Z ⊆ W⟩ by blast
  qed
  ultimately show ?thesis
  by (metis Diff_subset_conv Diff_triv Int_Diff_Un Int_absorb1 openin_closedin_eq
  that topspace_euclidean_subtopology)
  qed

```

lemma *ANR_imp_closed_neighbourhood_retract*:

```

  fixes S :: 'a::euclidean_space set
  assumes ANR S closedin (top_of_set U) S
  obtains V W where openin (top_of_set U) V
  closedin (top_of_set U) W
  S ⊆ V V ⊆ W S retract_of W
  by (meson ANR_imp_absolute_closed_neighbourhood_retract assms homeomor-
  phic_refl)

```

lemma *ANR_homeomorphic_ANR*:

```

  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes ANR T S homeomorphic T
  shows ANR S
  unfolding ANR_def
  by (metis assms ANR_imp_absolute_neighbourhood_retract homeomorphic_trans
  [of _ S] homeomorphic_sym)

```

lemma *homeomorphic_ANR_iff_ANR*:

```

  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  shows S homeomorphic T ⟹ ANR S ⟷ ANR T
  by (metis ANR_homeomorphic_ANR homeomorphic_sym)

```

9.26.1 Analogous properties of ENRs

lemma *ENR_imp_absolute_neighbourhood_retract*:

```

  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes ENR S and hom: S homeomorphic S'

```

```

    and  $S' \subseteq U$ 
  obtains  $V$  where  $openin (top\_of\_set U) V S'$   $retract\_of V$ 
proof -
  obtain  $X$  where  $open X S$   $retract\_of X$ 
  using  $\langle ENR S \rangle$  by (auto simp:  $ENR\_def$ )
  then obtain  $r$  where  $retraction X S r$ 
  by (auto simp:  $retract\_of\_def$ )
  have  $locally\_compact S'$ 
  using  $retract\_of\_locally\_compact$   $open\_imp\_locally\_compact$ 
     $homeomorphic\_local\_compactness \langle S retract\_of X \rangle \langle open X \rangle$   $hom$  by blast
  then obtain  $W$  where  $UW: openin (top\_of\_set U) W$ 
    and  $WS': closedin (top\_of\_set W) S'$ 
  apply (rule  $locally\_compact\_closedin\_open$ )
  by (meson  $Int\_lower2$   $assms(3)$   $closedin\_imp\_subset$   $closedin\_subset\_trans$ 
     $le\_inf\_iff\_openin\_open$ )
  obtain  $f g$  where  $hom: homeomorphism S S' f g$ 
  using  $assms$  by (force simp:  $homeomorphic\_def$ )
  have  $contg: continuous\_on S' g$ 
  using  $hom$   $homeomorphism\_def$  by blast
  moreover have  $g \text{ ` } S' \subseteq S$  by (metis  $hom$   $equalityE$   $homeomorphism\_def$ )
  ultimately obtain  $h$  where  $conth: continuous\_on W h$  and  $hg: \bigwedge x. x \in S' \implies h x = g x$ 
  using  $Tietze\_unbounded [of S' g W]$   $WS'$  by blast
  have  $W \subseteq U$  using  $UW$   $openin\_open$  by auto
  have  $S' \subseteq W$  using  $WS'$   $closedin\_closed$  by auto
  have  $him: \bigwedge x. x \in S' \implies h x \in X$ 
  by (metis (no\_types)  $\langle S retract\_of X \rangle hg$   $hom$   $homeomorphism\_def$   $image\_insert$ 
     $insert\_absorb$   $insert\_iff$   $retract\_of\_imp\_subset$   $subset\_eq$ )
  have  $S' retract\_of (W \cap h \text{ ` } X)$ 
  proof (simp add:  $retraction\_def$   $retract\_of\_def$ , intro  $exI$   $conjI$ )
    show  $S' \subseteq W$   $S' \subseteq h \text{ ` } X$ 
    using  $him$   $WS'$   $closedin\_imp\_subset$  by blast+
    show  $continuous\_on (W \cap h \text{ ` } X) (f \circ r \circ h)$ 
    proof (intro  $continuous\_on\_compose$ )
      show  $continuous\_on (W \cap h \text{ ` } X) h$ 
      by (meson  $conth$   $continuous\_on\_subset$   $inf\_le1$ )
      show  $continuous\_on (h \text{ ` } (W \cap h \text{ ` } X)) r$ 
    proof -
      have  $h \text{ ` } (W \cap h \text{ ` } X) \subseteq X$ 
      by blast
      then show  $continuous\_on (h \text{ ` } (W \cap h \text{ ` } X)) r$ 
      by (meson  $\langle retraction X S r \rangle$   $continuous\_on\_subset$   $retraction$ )
    qed
  qed
  show  $continuous\_on (r \text{ ` } h \text{ ` } (W \cap h \text{ ` } X)) f$ 
  proof (rule  $continuous\_on\_subset [of S]$ )
    show  $continuous\_on S f$ 
    using  $hom$   $homeomorphism\_def$  by blast
    show  $r \text{ ` } h \text{ ` } (W \cap h \text{ ` } X) \subseteq S$ 
    by (metis  $\langle retraction X S r \rangle$   $image\_mono$   $image\_subset\_iff\_subset\_vimage$ )
  qed

```



```

inf_le2 retraction)
  qed
  qed
  show  $(f \circ r \circ h) \in (W \cap h^{-1} X) \rightarrow S'$ 
    using <retraction X S r> hom
    by (auto simp: retraction_def homeomorphism_def)
  show  $\forall x \in S'. (f \circ r \circ h) x = x$ 
    using <retraction X S r> hom by (auto simp: retraction_def homeomor-
    phism_def hg)
  qed
  then show ?thesis
    using UW <open X> conth continuous_openin_preimage_eq openin_trans that
  by blast
  qed

```

```

corollary ENR_imp_absolute_neighbourhood_retract_UNIV:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $S' :: 'b::euclidean\_space\ set$ 
  assumes ENR S S homeomorphic S'
  obtains  $T'$  where open T' S' retract_of T'
  by (metis ENR_imp_absolute_neighbourhood_retract UNIV_I assms(1) assms(2)
  open_openin_subsetI subtopology_UNIV)

```

```

lemma ENR_homeomorphic_ENR:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes ENR T S homeomorphic T
  shows ENR S
  unfolding ENR_def
  by (meson ENR_imp_absolute_neighbourhood_retract_UNIV assms homeomor-
  phic_sym)

```

```

lemma homeomorphic_ENR_iff_ENR:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes S homeomorphic T
  shows  $ENR\ S \longleftrightarrow ENR\ T$ 
  by (meson ENR_homeomorphic_ENR assms homeomorphic_sym)

```

```

lemma ENR_translation:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  shows  $ENR(image\ (\lambda x. a + x)\ S) \longleftrightarrow ENR\ S$ 
  by (meson homeomorphic_sym homeomorphic_translation homeomorphic_ENR_iff_ENR)

```

```

lemma ENR_linear_image_eq:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes linear f inj f
  shows  $ENR\ (image\ f\ S) \longleftrightarrow ENR\ S$ 
  by (meson assms homeomorphic_ENR_iff_ENR linear_homeomorphic_image)

```

Some relations among the concepts. We also relate AR to being a retract of UNIV, which is often a more convenient proxy in the closed case.

3518

lemma *AR_imp_ANR*: $AR\ S \implies ANR\ S$
using *ANR_def AR_def* **by** *fastforce*

lemma *ENR_imp_ANR*:
fixes $S :: 'a::euclidean_space\ set$
shows $ENR\ S \implies ANR\ S$
by (*meson ANR_def ENR_imp_absolute_neighbourhood_retract closedin_imp_subset*)

lemma *ENR_ANR*:
fixes $S :: 'a::euclidean_space\ set$
shows $ENR\ S \longleftrightarrow ANR\ S \wedge locally\ compact\ S$

proof

assume $ENR\ S$

then have *locally compact S*

using *ENR_def open_imp_locally_compact retract_of_locally_compact* **by**

auto

then show $ANR\ S \wedge locally\ compact\ S$

using *ENR_imp_ANR* $\langle ENR\ S \rangle$ **by** *blast*

next

assume $ANR\ S \wedge locally\ compact\ S$

then have *ANR S locally compact S* **by** *auto*

then obtain $T :: ('a * real)\ set$ **where** *closed T S homeomorphic T*

using *locally_compact_homeomorphic_closed*

by (*metis DIM_prod DIM_real Suc_eq_plus1 lessI*)

then show $ENR\ S$

using $\langle ANR\ S \rangle$

by (*meson ANR_imp_absolute_neighbourhood_retract_UNIV ENR_def ENR_homeomorphic_ENR*)

qed

lemma *AR_ANR*:

fixes $S :: 'a::euclidean_space\ set$

shows $AR\ S \longleftrightarrow ANR\ S \wedge contractible\ S \wedge S \neq \{\}$

(**is** *?lhs = ?rhs*)

proof

assume *?lhs*

have *aff_dim S < int DIM('a × real)*

using *aff_dim_le_DIM* [of S] **by** *auto*

then obtain C and $S' :: ('a * real)\ set$

where *convex C C ≠ {} closedin (top_of_set C) S' S homeomorphic S'*

using *homeomorphic_closedin_convex* **by** *blast*

with $\langle AR\ S \rangle$ **have** *contractible S*

by (*meson AR_def convex_imp_contractible homeomorphic_contractible_eq retract_of_contractible*)

with $\langle AR\ S \rangle$ **show** *?rhs*

using *AR_imp_ANR AR_imp_retract* **by** *fastforce*

next

assume *?rhs*

then obtain a and $h :: real \times 'a \Rightarrow 'a$

```

  where conth: continuous_on ( $\{0..1\} \times S$ ) h
    and hS:  $h \text{ ' } (\{0..1\} \times S) \subseteq S$ 
    and [simp]:  $\bigwedge x. h(0, x) = x$ 
    and [simp]:  $\bigwedge x. h(1, x) = a$ 
    and ANR S S  $\neq \{\}$ 
  by (auto simp: contractible_def homotopic_with_def)
then have a  $\in S$ 
  by (metis all_not_in_conv atLeastAtMost_iff image_subset_iff mem_Sigma_iff
order_refl zero_le_one)
  have  $\exists g. \text{continuous\_on } W \ g \wedge g \in W \rightarrow S \wedge (\forall x \in T. g \ x = f \ x)$ 
    if  $f: \text{continuous\_on } T \ f \ f \in T \rightarrow S$ 
    and WT: closedin (top_of_set W) T
    for W T and f :: 'a  $\times$  real  $\Rightarrow$  'a
  proof -
    obtain U g
      where T  $\subseteq U$  and WU: openin (top_of_set W) U
      and contg: continuous_on U g
      and g  $\in U \rightarrow S$  and gf:  $\bigwedge x. x \in T \implies g \ x = f \ x$ 
      using iffD1 [OF ANR_eq_absolute_neighbourhood_extensor  $\langle$ ANR S $\rangle$ ,
rule_format, OF f WT]
      by auto
    have WWU: closedin (top_of_set W) (W - U)
      using WU closedin_diff by fastforce
    moreover have (W - U)  $\cap T = \{\}$ 
      using  $\langle T \subseteq U \rangle$  by auto
    ultimately obtain V V'
      where WV': openin (top_of_set W) V'
      and WV: openin (top_of_set W) V
      and W - U  $\subseteq V' \ T \subseteq V \ V' \cap V = \{\}$ 
      using separation_normal_local [of W W - U T] WT by blast
    then have WVT: T  $\cap$  (W - V) =  $\{\}$ 
      by auto
    have WWV: closedin (top_of_set W) (W - V)
      using WV closedin_diff by fastforce
    obtain j :: 'a  $\times$  real  $\Rightarrow$  real
      where contj: continuous_on W j
      and j:  $\bigwedge x. x \in W \implies j \ x \in \{0..1\}$ 
      and j0:  $\bigwedge x. x \in W - V \implies j \ x = 1$ 
      and j1:  $\bigwedge x. x \in T \implies j \ x = 0$ 
      by (rule Urysohn_local [OF WT WWV WVT, of 0 1::real]) (auto simp:
in_segment)
    have Weg: W = (W - V)  $\cup$  (W - V')
      using  $\langle V' \cap V = \{\} \rangle$  by force
    show ?thesis
  proof (intro conjI exI)
    have *: continuous_on (W - V') ( $\lambda x. h \ (j \ x, g \ x)$ )
    proof (rule continuous_on_compose2 [OF conth continuous_on_Pair])
      show continuous_on (W - V') j
        by (rule continuous_on_subset [OF contj Diff_subset])
    end
  end

```

```

    show continuous_on (W - V') g
      by (metis Diff_subset_conv ⟨W - U ⊆ V'⟩ contg continuous_on_subset
Un_commute)
    show (λx. (j x, g x)) ' (W - V') ⊆ {0..1} × S
      using j ⟨g ∈ U → S⟩ ⟨W - U ⊆ V'⟩ by fastforce
  qed
  show continuous_on W (λx. if x ∈ W - V then a else h (j x, g x))
  proof (subst Weq, rule continuous_on_cases_local)
    show continuous_on (W - V') (λx. h (j x, g x))
      using * by blast
  qed (use WWV WV' Weq j0 j1 in auto)
next
  have h (j (x, y), g (x, y)) ∈ S if (x, y) ∈ W (x, y) ∈ V for x y
  proof -
    have j(x, y) ∈ {0..1}
      using j that by blast
    moreover have g(x, y) ∈ S
      using ⟨V' ∩ V = {}⟩ ⟨W - U ⊆ V'⟩ ⟨g ∈ U → S⟩ that by fastforce
    ultimately show ?thesis
      using hS by blast
  qed
  with ⟨a ∈ S⟩ ⟨g ∈ U → S⟩
  show (λx. if x ∈ W - V then a else h (j x, g x)) ∈ W → S
    by auto
next
  show ∀x∈T. (if x ∈ W - V then a else h (j x, g x)) = f x
    using ⟨T ⊆ V⟩ by (auto simp: j0 j1 gf)
  qed
  then show ?lhs
    by (simp add: AR_eq_absolute_extensor image_subset_iff_funcset)
  qed

```

lemma ANR_retract_of_ANR:

```

  fixes S :: 'a::euclidean_space set
  assumes ANR T and ST: S retract_of T
  shows ANR S
proof (clarsimp simp add: ANR_eq_absolute_neighbourhood_extensor)
  fix f::'a × real ⇒ 'a and U W
  assume W: continuous_on W f f ∈ W → S closedin (top_of_set U) W
  then obtain r where S ⊆ T and r: continuous_on T r r ∈ T → S ∀x∈S. r
x = x continuous_on W f f ∈ W → S
      closedin (top_of_set U) W
  by (metis ST retract_of_def retraction_def)
  then have f ' W ⊆ T
    by blast
  with W obtain V g where V: W ⊆ V openin (top_of_set U) V continuous_on
V g g ∈ V → T ∀x∈W. g x = f x

```

```

by (smt (verit) ANR_imp_absolute_neighbourhood_extensor Pi_I assms(1)
funcset_mem image_subset_iff_funcset)
with r have continuous_on V (r ∘ g) ∧ (r ∘ g) ∈ V → S ∧ (∀ x ∈ W. (r ∘ g) x
= f x)
by (smt (verit, del_insts) Pi_iff_comp_apply continuous_on_compose contin-
uous_on_subset image_subset_iff_funcset)
then show ∃ V. W ⊆ V ∧ openin (top_of_set U) V ∧ (∃ g. continuous_on V
g ∧ g ∈ V → S ∧ (∀ x ∈ W. g x = f x))
by (meson V)
qed

```

lemma AR_retract_of_AR:

```

fixes S :: 'a::euclidean_space set
shows [[AR T; S retract_of T]] ⇒ AR S
using ANR_retract_of_ANR AR_ANR retract_of_contractible by fastforce

```

lemma ENR_retract_of_ENR:

```

[[ENR T; S retract_of T]] ⇒ ENR S
by (meson ENR_def retract_of_trans)

```

lemma retract_of_UNIV:

```

fixes S :: 'a::euclidean_space set
shows S retract_of UNIV ⇔ AR S ∧ closed S
by (metis AR_ANR AR_imp_retract ENR_def ENR_imp_ANR closed_UNIV
closed_closedin contractible_UNIV empty_not_UNIV open_UNIV retract_of_closed
retract_of_contractible retract_of_empty(1) subtopology_UNIV)

```

lemma compact_AR:

```

fixes S :: 'a::euclidean_space set
shows compact S ∧ AR S ⇔ compact S ∧ S retract_of UNIV
using compact_imp_closed retract_of_UNIV by blast

```

More properties of ARs, ANRs and ENRs

```

lemma not_AR_empty [simp]: ¬ AR({})
by (auto simp: AR_def)

```

```

lemma ENR_empty [simp]: ENR {}
by (simp add: ENR_def)

```

```

lemma ANR_empty [simp]: ANR ({} :: 'a::euclidean_space set)
by (simp add: ENR_imp_ANR)

```

lemma convex_imp_AR:

```

fixes S :: 'a::euclidean_space set
shows [[convex S; S ≠ {}]] ⇒ AR S
by (metis (mono_tags, lifting) Dugundji_absolute_extensor_imp_AR)

```

lemma convex_imp_ANR:

```

fixes S :: 'a::euclidean_space set

```

3522

shows $\text{convex } S \implies \text{ANR } S$
using *ANR_empty AR_imp_ANR convex_imp_AR by blast*

lemma *ENR_convex_closed*:
fixes $S :: 'a::\text{euclidean_space set}$
shows $\llbracket \text{closed } S; \text{convex } S \rrbracket \implies \text{ENR } S$
using *ENR_def ENR_empty convex_imp_AR retract_of_UNIV by blast*

lemma *AR_UNIV [simp]*: $\text{AR } (\text{UNIV} :: 'a::\text{euclidean_space set})$
using *retract_of_UNIV by auto*

lemma *ANR_UNIV [simp]*: $\text{ANR } (\text{UNIV} :: 'a::\text{euclidean_space set})$
by (*simp add: AR_imp_ANR*)

lemma *ENR_UNIV [simp]*: $\text{ENR } \text{UNIV}$
using *ENR_def by blast*

lemma *AR_singleton*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{AR } \{a\}$
using *retract_of_UNIV by blast*

lemma *ANR_singleton*:
fixes $a :: 'a::\text{euclidean_space}$
shows $\text{ANR } \{a\}$
by (*simp add: AR_imp_ANR AR_singleton*)

lemma *ENR_singleton*: $\text{ENR } \{a\}$
using *ENR_def by blast*

ARs closed under union

lemma *AR_closed_Un_local_aux*:
fixes $U :: 'a::\text{euclidean_space set}$
assumes $\text{closedin } (\text{top_of_set } U) S$
 $\text{closedin } (\text{top_of_set } U) T$
 $\text{AR } S \text{ AR } T \text{ AR}(S \cap T)$
shows $(S \cup T) \text{ retract_of } U$
proof –
have $S \cap T \neq \{\}$
using *assms AR_def by fastforce*
have $S \subseteq U \ T \subseteq U$
using *assms by (auto simp: closedin_imp_subset)*
define S' **where** $S' \equiv \{x \in U. \text{setdist } \{x\} S \leq \text{setdist } \{x\} T\}$
define T' **where** $T' \equiv \{x \in U. \text{setdist } \{x\} T \leq \text{setdist } \{x\} S\}$
define W **where** $W \equiv \{x \in U. \text{setdist } \{x\} S = \text{setdist } \{x\} T\}$
have US' : $\text{closedin } (\text{top_of_set } U) S'$
using *continuous_closedin_preimage [of U $\lambda x. \text{setdist } \{x\} S - \text{setdist } \{x\} T$ $\{..0\}$]*
by (*simp add: S'_def vimage_def Collect_conj_eq continuous_on_diff contin-*

```

uous_on_setdist)
  have UT': closedin (top_of_set U) T'
    using continuous_closedin_preimage [of U  $\lambda x. \text{setdist } \{x\} T - \text{setdist } \{x\} S$ 
    {..0}]
    by (simp add: T'_def vimage_def Collect_conj_eq continuous_on_diff con-
    tinuous_on_setdist)
  have S  $\subseteq$  S'
    using S'_def  $\langle S \subseteq U \rangle \text{setdist\_sing\_in\_set}$  by fastforce
  have T  $\subseteq$  T'
    using T'_def  $\langle T \subseteq U \rangle \text{setdist\_sing\_in\_set}$  by fastforce
  have S  $\cap$  T  $\subseteq$  W W  $\subseteq$  U
    using  $\langle S \subseteq U \rangle$  by (auto simp: W_def setdist_sing_in_set)
  have (S  $\cap$  T) retract_of W
  proof (rule AR_imp_absolute_retract [OF  $\langle AR(S \cap T) \rangle$ ])
    show S  $\cap$  T homeomorphic S  $\cap$  T
      by (simp add: homeomorphic_refl)
    show closedin (top_of_set W) (S  $\cap$  T)
      by (meson  $\langle S \cap T \subseteq W \rangle \langle W \subseteq U \rangle \text{assms closedin\_Int closedin\_subset\_trans}$ )
  qed
  then obtain r0
    where S  $\cap$  T  $\subseteq$  W and contr0: continuous_on W r0
    and r0 ' W  $\subseteq$  S  $\cap$  T
    and r0 [simp]:  $\bigwedge x. x \in S \cap T \implies r0 x = x$ 
    by (auto simp: retract_of_def retraction_def)
  have ST:  $x \in W \implies x \in S \longleftrightarrow x \in T$  for x
    using setdist_eq_0_closedin  $\langle S \cap T \neq \{\} \rangle \text{assms}$ 
    by (force simp: W_def setdist_sing_in_set)
  have S'  $\cap$  T' = W
    by (auto simp: S'_def T'_def W_def)
  then have cloUW: closedin (top_of_set U) W
    using closedin_Int US' UT' by blast
  define r where r  $\equiv \lambda x. \text{if } x \in W \text{ then } r0 x \text{ else } x$ 
  have contr: continuous_on (W  $\cup$  (S  $\cup$  T)) r
  unfolding r_def
  proof (rule continuous_on_cases_local [OF __ contr0 continuous_on_id])
    show closedin (top_of_set (W  $\cup$  (S  $\cup$  T))) W
      using  $\langle S \subseteq U \rangle \langle T \subseteq U \rangle \langle W \subseteq U \rangle \langle \text{closedin (top_of_set U) W} \rangle \text{closedin\_subset\_trans}$ 
    by fastforce
    show closedin (top_of_set (W  $\cup$  (S  $\cup$  T))) (S  $\cup$  T)
      by (meson  $\langle S \subseteq U \rangle \langle T \subseteq U \rangle \langle W \subseteq U \rangle \text{assms closedin\_Un closedin\_subset\_trans}$ 
      sup.bounded_iff sup.cobounded2)
    show  $\bigwedge x. x \in W \wedge x \notin W \vee x \in S \cup T \wedge x \in W \implies r0 x = x$ 
      by (auto simp: ST)
  qed
  have rim: r ' (W  $\cup$  S)  $\subseteq$  S r ' (W  $\cup$  T)  $\subseteq$  T
    using  $\langle r0 ' W \subseteq S \cap T \rangle r\_def$  by auto
  have cloUWS: closedin (top_of_set U) (W  $\cup$  S)
    by (simp add: cloUW assms closedin_Un)
  obtain g where contg: continuous_on U g

```

```

    and g ' U ⊆ S and gqr:  $\bigwedge x. x \in W \cup S \implies g x = r x$ 
  proof (rule AR_imp_absolute_extensor [OF <AR S> __ cloUWS])
    show continuous_on (W ∪ S) r
      using continuous_on_subset contr sup_assoc by blast
  qed (use rim in auto)
  have cloUWT: closedin (top_of_set U) (W ∪ T)
    by (simp add: cloUW assms closedin_Un)
  obtain h where conth: continuous_on U h
    and h ' U ⊆ T and heqr:  $\bigwedge x. x \in W \cup T \implies h x = r x$ 
  proof (rule AR_imp_absolute_extensor [OF <AR T> __ cloUWT])
    show continuous_on (W ∪ T) r
      using continuous_on_subset contr sup_assoc by blast
  qed (use rim in auto)
  have U: U = S' ∪ T'
    by (force simp: S'_def T'_def)
  have cont: continuous_on U ( $\lambda x. \text{if } x \in S' \text{ then } g x \text{ else } h x$ )
    unfolding U
    apply (rule continuous_on_cases_local)
    using US' UT' <S' ∩ T' = W> <U = S' ∪ T'>
      contg conth continuous_on_subset gqr heqr by auto
  have UST: ( $\lambda x. \text{if } x \in S' \text{ then } g x \text{ else } h x$ ) ' U ⊆ S ∪ T
    using <g ' U ⊆ S> <h ' U ⊆ T> by auto
  show ?thesis
    apply (simp add: retract_of_def retraction_def <S ⊆ U> <T ⊆ U>)
    apply (rule_tac x= $\lambda x. \text{if } x \in S' \text{ then } g x \text{ else } h x$  in exI)
    using ST UST <S ⊆ S'> <S' ∩ T' = W> <T ⊆ T'> cont gqr heqr r_def
    by (smt (verit, del_insts) IntI Pi_I Un_iff image_subset_iff r0 subsetD)
qed

```

lemma AR_closed_Un_local:

```

  fixes S :: 'a::euclidean_space set
  assumes STS: closedin (top_of_set (S ∪ T)) S
    and STT: closedin (top_of_set (S ∪ T)) T
    and AR S AR T AR(S ∩ T)
  shows AR(S ∪ T)
proof -
  have C retract_of U
    if hom: S ∪ T homeomorphic C and UC: closedin (top_of_set U) C
    for U and C :: ('a * real) set
  proof -
    obtain f g where hom: homeomorphism (S ∪ T) C f g
      using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C ∩ g - ' S)
      by (metis STS continuous_on_imp_closedin hom homeomorphism_def closedin_trans
        [OF _ UC])
    have UT: closedin (top_of_set U) (C ∩ g - ' T)
      by (metis STT continuous_on_closed hom homeomorphism_def closedin_trans
        [OF _ UC])
  qed

```



```

have homeomorphism (C ∩ g -' S) S g f
  using hom
  apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
  apply (rule_tac x=f x in image_eqI, auto)
  done
then have ARS: AR (C ∩ g -' S)
  using ⟨AR S⟩ homeomorphic_AR_iff_AR homeomorphic_def by blast
have homeomorphism (C ∩ g -' T) T g f
  using hom
  apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
  apply (rule_tac x=f x in image_eqI, auto)
  done
then have ART: AR (C ∩ g -' T)
  using ⟨AR T⟩ homeomorphic_AR_iff_AR homeomorphic_def by blast
have homeomorphism (C ∩ g -' S ∩ (C ∩ g -' T)) (S ∩ T) g f
  using hom
  apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
  apply (rule_tac x=f x in image_eqI, auto)
  done
then have ARI: AR ((C ∩ g -' S) ∩ (C ∩ g -' T))
  using ⟨AR (S ∩ T)⟩ homeomorphic_AR_iff_AR homeomorphic_def by blast
have C = (C ∩ g -' S) ∪ (C ∩ g -' T)
  using hom by (auto simp: homeomorphism_def)
then show ?thesis
  by (metis AR_closed_Un_local_aux [OF US UT ARS ART ARI])
qed
then show ?thesis
  by (force simp: AR_def)
qed

```

corollary AR_closed_Un:

```

fixes S :: 'a::euclidean_space set
shows [[closed S; closed T; AR S; AR T; AR (S ∩ T)] ⇒ AR (S ∪ T)
by (metis AR_closed_Un_local_aux closed_closedin retract_of_UNIV subtopology_UNIV)

```

ANRs closed under union

lemma ANR_closed_Un_local_aux:

```

fixes U :: 'a::euclidean_space set
assumes US: closedin (top_of_set U) S
  and UT: closedin (top_of_set U) T
  and ANR S ANR T ANR(S ∩ T)
obtains V where openin (top_of_set U) V (S ∪ T) retract_of V
proof (cases S = {} ∨ T = {})
case True with assms that show ?thesis
  by (metis ANR_imp_neighbourhood_retract Un_commute inf_bot_right sup_inf_absorb)
next
case False
then have [simp]: S ≠ {} T ≠ {} by auto

```

```

have S ⊆ U T ⊆ U
  using assms by (auto simp: closedin_imp_subset)
define S' where S' ≡ {x ∈ U. setdist {x} S ≤ setdist {x} T}
define T' where T' ≡ {x ∈ U. setdist {x} T ≤ setdist {x} S}
define W where W ≡ {x ∈ U. setdist {x} S = setdist {x} T}
have cloUS': closedin (top_of_set U) S'
  using continuous_closedin_preimage [of U λx. setdist {x} S - setdist {x} T
{..0}]
  by (simp add: S'_def vimage_def Collect_conj_eq continuous_on_diff contin-
uous_on_setdist)
have cloUT': closedin (top_of_set U) T'
  using continuous_closedin_preimage [of U λx. setdist {x} T - setdist {x} S
{..0}]
  by (simp add: T'_def vimage_def Collect_conj_eq continuous_on_diff con-
tinuous_on_setdist)
have S ⊆ S'
  using S'_def ⟨S ⊆ U⟩ setdist_sing_in_set by fastforce
have T ⊆ T'
  using T'_def ⟨T ⊆ U⟩ setdist_sing_in_set by fastforce
have S' ∪ T' = U
  by (auto simp: S'_def T'_def)
have W ⊆ S'
  by (simp add: Collect_mono S'_def W_def)
have W ⊆ T'
  by (simp add: Collect_mono T'_def W_def)
have ST_W: S ∩ T ⊆ W and W ⊆ U
  using ⟨S ⊆ U⟩ by (force simp: W_def setdist_sing_in_set)+
have S' ∩ T' = W
  by (auto simp: S'_def T'_def W_def)
then have cloUW: closedin (top_of_set U) W
  using closedin_Int cloUS' cloUT' by blast
obtain W' W0 where openin (top_of_set W) W'
  and cloWW0: closedin (top_of_set W) W0
  and S ∩ T ⊆ W' W' ⊆ W0
  and ret: (S ∩ T) retract_of W0
  by (meson ANR_imp_closed_neighbourhood_retract ST_W US UT ⟨W ⊆ U⟩
⟨ANR(S ∩ T)⟩ closedin_Int closedin_subset_trans)
then obtain U0 where opeUU0: openin (top_of_set U) U0
  and U0: S ∩ T ⊆ U0 U0 ∩ W ⊆ W0
  unfolding openin_open using ⟨W ⊆ U⟩ by blast
have W0 ⊆ U
  using ⟨W ⊆ U⟩ cloWW0 closedin_subset by fastforce
obtain r0
  where S ∩ T ⊆ W0 and contr0: continuous_on W0 r0 and r0 ∈ W0 → S
∩ T
  and r0 [simp]: ∧x. x ∈ S ∩ T ⇒ r0 x = x
  using ret by (force simp: retract_of_def retraction_def)
have ST: x ∈ W ⇒ x ∈ S ↔ x ∈ T for x
  using assms by (auto simp: W_def setdist_sing_in_set dest!: setdist_eq_0_closedin)

```

```

define r where  $r \equiv \lambda x. \text{if } x \in W0 \text{ then } r0\ x \text{ else } x$ 
have  $r \text{ ' } (W0 \cup S) \subseteq S \text{ ' } (W0 \cup T) \subseteq T$ 
  using  $\langle r0 \in W0 \rightarrow S \cap T \rangle$  r_def by auto
have contr: continuous_on (W0  $\cup$  (S  $\cup$  T)) r
unfolding r_def
proof (rule continuous_on_cases_local [OF __ contr0 continuous_on_id])
  show closedin (top_of_set (W0  $\cup$  (S  $\cup$  T))) W0
    using closedin_subset_trans [of U]
    by (metis le_sup_iff order_refl cloWW0 cloUW closedin_trans  $\langle W0 \subseteq U \rangle$ 
 $\langle S \subseteq U \rangle$   $\langle T \subseteq U \rangle$ )
  show closedin (top_of_set (W0  $\cup$  (S  $\cup$  T))) (S  $\cup$  T)
    by (meson  $\langle S \subseteq U \rangle$   $\langle T \subseteq U \rangle$   $\langle W0 \subseteq U \rangle$  assms closedin_Un closedin_subset_trans
sup.bounded_iff sup.cobounded2)
  show  $\bigwedge x. x \in W0 \wedge x \notin W0 \vee x \in S \cup T \wedge x \in W0 \implies r0\ x = x$ 
    using ST cloWW0 closedin_subset by fastforce
qed
have cloS'WS: closedin (top_of_set S') (W0  $\cup$  S)
  by (meson closedin_subset_trans US cloUS'  $\langle S \subseteq S' \rangle$   $\langle W \subseteq S' \rangle$  cloUW cloWW0
closedin_Un closedin_imp_subset closedin_trans)
obtain W1 g where W0  $\cup$  S  $\subseteq$  W1 and contg: continuous_on W1 g
  and opeSW1: openin (top_of_set S') W1
  and  $g \in W1 \rightarrow S$  and geqr:  $\bigwedge x. x \in W0 \cup S \implies g\ x = r\ x$ 
proof (rule ANR_imp_absolute_neighbourhood_extensor [OF  $\langle$ ANR S $\rangle$  __
cloS'WS])
  show continuous_on (W0  $\cup$  S) r
    using continuous_on_subset contr sup_assoc by blast
qed (use  $\langle r \text{ ' } (W0 \cup S) \subseteq S \rangle$  in auto)
have cloT'WT: closedin (top_of_set T') (W0  $\cup$  T)
  by (meson closedin_subset_trans UT cloUT'  $\langle T \subseteq T' \rangle$   $\langle W \subseteq T' \rangle$  cloUW
cloWW0
closedin_Un closedin_imp_subset closedin_trans)
obtain W2 h where W0  $\cup$  T  $\subseteq$  W2 and conth: continuous_on W2 h
  and opeSW2: openin (top_of_set T') W2
  and  $h \text{ ' } W2 \subseteq T$  and heqr:  $\bigwedge x. x \in W0 \cup T \implies h\ x = r\ x$ 
proof (rule ANR_imp_absolute_neighbourhood_extensor [OF  $\langle$ ANR T $\rangle$  __
cloT'WT])
  show continuous_on (W0  $\cup$  T) r
    using continuous_on_subset contr sup_assoc by blast
qed (use  $\langle r \text{ ' } (W0 \cup T) \subseteq T \rangle$  in auto)
have  $S' \cap T' = W$ 
  by (force simp: S'_def T'_def W_def)
obtain O1 O2 where O12: open O1 W1 = S'  $\cap$  O1 open O2 W2 = T'  $\cap$  O2
  using opeSW1 opeSW2 by (force simp: openin_open)
show ?thesis
proof
  have eq: W1 - (W - U0)  $\cup$  (W2 - (W - U0))
    = ((U - T')  $\cap$  O1  $\cup$  (U - S')  $\cap$  O2  $\cup$  U  $\cap$  O1  $\cap$  O2) - (W - U0)
  (is ?WW1  $\cup$  ?WW2 = ?rhs)

```

```

using ⟨ $U0 \cap W \subseteq W0$ ⟩ ⟨ $W0 \cup S \subseteq W1$ ⟩ ⟨ $W0 \cup T \subseteq W2$ ⟩
by (auto simp: ⟨ $S' \cup T' = U$ ⟩ [symmetric] ⟨ $S' \cap T' = W$ ⟩ [symmetric] ⟨ $W1$ 
=  $S' \cap O1$ ⟩ ⟨ $W2 = T' \cap O2$ ⟩)
show openin (top_of_set U) (?WW1 ∪ ?WW2)
by (simp add: eq ⟨open O1⟩ ⟨open O2⟩ cloUS' cloUT' cloUW closedin_diff
opeUU0 openin_Int_open openin_Un openin_diff)
obtain SU' where closed SU' S' = U ∩ SU'
using cloUS' by (auto simp add: closedin_closed)
moreover have ?WW1 = (?WW1 ∪ ?WW2) ∩ SU'
using ⟨S' = U ∩ SU'⟩ ⟨W1 = S' ∩ O1⟩ ⟨S' ∪ T' = U⟩ ⟨W2 = T' ∩ O2⟩
⟨S' ∩ T' = W⟩ ⟨W0 ∪ S ⊆ W1⟩ U0
by auto
ultimately have cloW1: closedin (top_of_set (W1 - (W - U0) ∪ (W2 -
(W - U0)))) (W1 - (W - U0))
by (metis closedin_closed_Int)
obtain TU' where closed TU' T' = U ∩ TU'
using cloUT' by (auto simp add: closedin_closed)
moreover have ?WW2 = (?WW1 ∪ ?WW2) ∩ TU'
using ⟨T' = U ∩ TU'⟩ ⟨W1 = S' ∩ O1⟩ ⟨S' ∪ T' = U⟩ ⟨W2 = T' ∩ O2⟩
⟨S' ∩ T' = W⟩ ⟨W0 ∪ T ⊆ W2⟩ U0
by auto
ultimately have cloW2: closedin (top_of_set (?WW1 ∪ ?WW2)) ?WW2
by (metis closedin_closed_Int)
let ?gh = λx. if x ∈ S' then g x else h x
have ∃r. continuous_on (?WW1 ∪ ?WW2) r ∧ r ' (?WW1 ∪ ?WW2) ⊆ S
∪ T ∧ (∀x ∈ S ∪ T. r x = x)
proof (intro exI conjI)
show ∀x ∈ S ∪ T. ?gh x = x
using ST ⟨S' ∩ T' = W⟩ geqr heqr O12
by (metis Int_iff Un_iff ⟨W0 ∪ S ⊆ W1⟩ ⟨W0 ∪ T ⊆ W2⟩ r0 r_def
sup.order_iff)
have ∧x. x ∈ ?WW1 ∧ x ∉ S' ∨ x ∈ ?WW2 ∧ x ∈ S' ⇒ g x = h x
using O12
by (metis (full_types) DiffD1 DiffD2 DiffI IntE IntI U0(2) UnCI ⟨S' ∩ T'
= W⟩ geqr heqr in_mono)
then show continuous_on (?WW1 ∪ ?WW2) ?gh
using continuous_on_cases_local [OF cloW1 cloW2 continuous_on_subset
[OF contg] continuous_on_subset [OF conth]]
by simp
show ?gh ' (?WW1 ∪ ?WW2) ⊆ S ∪ T
using ⟨W1 = S' ∩ O1⟩ ⟨W2 = T' ∩ O2⟩ ⟨S' ∩ T' = W⟩ ⟨g ∈ W1 → S⟩
⟨h ' W2 ⊆ T⟩ ⟨U0 ∩ W ⊆ W0⟩ ⟨W0 ∪ S ⊆ W1⟩
by (auto simp add: image_subset_iff)
qed
then show S ∪ T retract_of ?WW1 ∪ ?WW2
using ⟨W0 ∪ S ⊆ W1⟩ ⟨W0 ∪ T ⊆ W2⟩ ST opeUU0 U0
by (auto simp: retract_of_def retraction_def image_subset_iff_funcset)
qed
qed

```

```

lemma ANR_closed_Un_local:
  fixes S :: 'a::euclidean_space set
  assumes STS: closedin (top_of_set (S ∪ T)) S
    and STT: closedin (top_of_set (S ∪ T)) T
    and ANR S ANR T ANR(S ∩ T)
  shows ANR(S ∪ T)
proof -
  have ∃ T. openin (top_of_set U) T ∧ C retract_of T
    if hom: S ∪ T homeomorphic C and UC: closedin (top_of_set U) C
    for U and C :: ('a * real) set
  proof -
    obtain f g where hom: homeomorphism (S ∪ T) C f g
      using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C ∩ g -' S)
      by (metis STS UC closedin_trans continuous_on_imp_closedin hom homeo-
morphicism_def)
    have UT: closedin (top_of_set U) (C ∩ g -' T)
      by (metis STT UC closedin_trans continuous_on_imp_closedin hom homeo-
morphicism_def)
    have homeomorphism (C ∩ g -' S) S g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRS: ANR (C ∩ g -' S)
      using ⟨ANR S⟩ homeomorphic_ANR_iff_ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g -' T) T g f
      using hom apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRT: ANR (C ∩ g -' T)
      using ⟨ANR T⟩ homeomorphic_ANR_iff_ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g -' S ∩ (C ∩ g -' T)) (S ∩ T) g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRI: ANR ((C ∩ g -' S) ∩ (C ∩ g -' T))
      using ⟨ANR (S ∩ T)⟩ homeomorphic_ANR_iff_ANR homeomorphic_def
    by blast
    have C = (C ∩ g -' S) ∪ (C ∩ g -' T)
      using hom by (auto simp: homeomorphism_def)
    then show ?thesis
      by (metis ANR_closed_Un_local_aux [OF US UT ANRS ANRT ANRI])
  qed
  then show ?thesis
    by (auto simp: ANR_def)
qed

```

corollary ANR_closed_Un:

3530

fixes $S :: 'a::\text{euclidean_space set}$
shows $\llbracket \text{closed } S; \text{ closed } T; \text{ ANR } S; \text{ ANR } T; \text{ ANR } (S \cap T) \rrbracket \implies \text{ANR } (S \cup T)$
by (*simp add: ANR_closed_Un_local closedin_def diff_eq open_Compl openin_open_Int*)

lemma *ANR_openin:*

fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{ANR } T$ **and** $\text{opeTS: openin } (\text{top_of_set } T) S$
shows $\text{ANR } S$
proof (*clarsimp simp only: ANR_eq_absolute_neighbourhood_extensor*)
fix $f :: 'a \times \text{real} \Rightarrow 'a$ **and** $U C$
assume $\text{contf: continuous_on } C f$ **and** $\text{fim: } f \in C \rightarrow S$
and $\text{cloUC: closedin } (\text{top_of_set } U) C$
have $f \in C \rightarrow T$
using $\text{fim opeTS openin_imp_subset}$ **by** *blast*
obtain $W g$ **where** $C \subseteq W$
and $UW: \text{openin } (\text{top_of_set } U) W$
and $\text{contg: continuous_on } W g$
and $\text{gim: } g \in W \rightarrow T$
and $\text{geq: } \bigwedge x. x \in C \implies g x = f x$
using $\text{ANR_imp_absolute_neighbourhood_extensor } [OF \langle \text{ANR } T \rangle \text{ contf } \langle f \in C \rightarrow T \rangle \text{ cloUC}] \text{ fim}$ **by** *auto*
show $\exists V g. C \subseteq V \wedge \text{openin } (\text{top_of_set } U) V \wedge \text{continuous_on } V g \wedge g \in V \rightarrow S \wedge (\forall x \in C. g x = f x)$
proof (*intro exI conjI*)
show $C \subseteq W \cap g^{-1} S$
using $\langle C \subseteq W \rangle \text{ fim geq}$ **by** *blast*
show $\text{openin } (\text{top_of_set } U) (W \cap g^{-1} S)$
by (*metis (mono_tags, lifting) UW contg continuous_openin_preimage gim opeTS openin_trans*)
show $\text{continuous_on } (W \cap g^{-1} S) g$
by (*blast intro: continuous_on_subset [OF contg]*)
show $g \in (W \cap g^{-1} S) \rightarrow S$
using gim **by** *blast*
show $\forall x \in C. g x = f x$
using geq **by** *blast*
qed
qed

lemma *ENR_openin:*

fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{ENR } T$ $\text{openin } (\text{top_of_set } T) S$
shows $\text{ENR } S$
by (*meson ANR_openin ENR_ANR assms locally_open_subset*)

lemma *ANR_neighborhood_retract:*

fixes $S :: 'a::\text{euclidean_space set}$
assumes $\text{ANR } U$ $S \text{ retract_of } T$ $\text{openin } (\text{top_of_set } U) T$
shows $\text{ANR } S$
using $\text{ANR_openin ANR_retract_of_ANR assms}$ **by** *blast*

```

lemma ENR_neighborhood_retract:
  fixes S :: 'a::euclidean_space set
  assumes ENR U S retract_of T openin (top_of_set U) T
  shows ENR S
  using ENR_openin ENR_retract_of_ENR assms by blast

lemma ANR_rel_interior:
  fixes S :: 'a::euclidean_space set
  shows ANR S  $\implies$  ANR(rel_interior S)
  by (blast intro: ANR_openin openin_set_rel_interior)

lemma ANR_delete:
  fixes S :: 'a::euclidean_space set
  shows ANR S  $\implies$  ANR(S - {a})
  by (blast intro: ANR_openin openin_delete openin_subtopology_self)

lemma ENR_rel_interior:
  fixes S :: 'a::euclidean_space set
  shows ENR S  $\implies$  ENR(rel_interior S)
  by (blast intro: ENR_openin openin_set_rel_interior)

lemma ENR_delete:
  fixes S :: 'a::euclidean_space set
  shows ENR S  $\implies$  ENR(S - {a})
  by (blast intro: ENR_openin openin_delete openin_subtopology_self)

lemma open_imp_ENR: open S  $\implies$  ENR S
  using ENR_def by blast

lemma open_imp_ANR:
  fixes S :: 'a::euclidean_space set
  shows open S  $\implies$  ANR S
  by (simp add: ENR_imp_ANR open_imp_ENR)

lemma ANR_ball [iff]:
  fixes a :: 'a::euclidean_space
  shows ANR(ball a r)
  by (simp add: convex_imp_ANR)

lemma ENR_ball [iff]: ENR(ball a r)
  by (simp add: open_imp_ENR)

lemma AR_ball [simp]:
  fixes a :: 'a::euclidean_space
  shows AR(ball a r)  $\iff$  0 < r
  by (auto simp: AR_ANR convex_imp_contractible)

lemma ANR_cball [iff]:

```

3532

```
fixes a :: 'a::euclidean_space
shows ANR(cball a r)
by (simp add: convex_imp_ANR)
```

```
lemma ENR_cball:
fixes a :: 'a::euclidean_space
shows ENR(cball a r)
using ENR_convex_closed by blast
```

```
lemma AR_cball [simp]:
fixes a :: 'a::euclidean_space
shows AR(cball a r)  $\longleftrightarrow$   $0 \leq r$ 
by (auto simp: AR_ANR convex_imp_contractible)
```

```
lemma ANR_box [iff]:
fixes a :: 'a::euclidean_space
shows ANR(cbox a b) ANR(box a b)
by (auto simp: convex_imp_ANR open_imp_ANR)
```

```
lemma ENR_box [iff]:
fixes a :: 'a::euclidean_space
shows ENR(cbox a b) ENR(box a b)
by (simp_all add: ENR_convex_closed closed_cbox open_box open_imp_ENR)
```

```
lemma AR_box [simp]:
AR(cbox a b)  $\longleftrightarrow$   $cbox\ a\ b \neq \{\}$  AR(box a b)  $\longleftrightarrow$   $box\ a\ b \neq \{\}$ 
by (auto simp: AR_ANR convex_imp_contractible)
```

```
lemma ANR_interior:
fixes S :: 'a::euclidean_space set
shows ANR(interior S)
by (simp add: open_imp_ANR)
```

```
lemma ENR_interior:
fixes S :: 'a::euclidean_space set
shows ENR(interior S)
by (simp add: open_imp_ENR)
```

```
lemma AR_imp_contractible:
fixes S :: 'a::euclidean_space set
shows AR S  $\implies$  contractible S
by (simp add: AR_ANR)
```

```
lemma ENR_imp_locally_compact:
fixes S :: 'a::euclidean_space set
shows ENR S  $\implies$  locally compact S
by (simp add: ENR_ANR)
```

```
lemma ANR_imp_locally_path_connected:
```



```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $ANR\ S$ 
shows  $locally\ path\_connected\ S$ 
proof –
obtain  $U\ and\ T :: ('a \times real)\ set$ 
where  $convex\ U\ U \neq \{\}$ 
and  $UT: closedin\ (top\_of\_set\ U)\ T\ and\ S\ homeomorphic\ T$ 
proof ( $rule\ homeomorphic\_closedin\_convex$ )
show  $aff\_dim\ S < int\ DIM('a \times real)$ 
using  $aff\_dim\_le\_DIM\ [of\ S]\ by\ auto$ 
qed  $auto$ 
then have  $locally\ path\_connected\ T$ 
by ( $meson\ ANR\_imp\_absolute\_neighbourhood\_retract$ 
 $assms\ convex\_imp\_locally\_path\_connected\ locally\_open\_subset\ retract\_of\_locally\_path\_connected$ )
then have  $S: locally\ path\_connected\ S$ 
if  $openin\ (top\_of\_set\ U)\ V\ T\ retract\_of\ V\ U \neq \{\}$  for  $V$ 
using  $\langle S\ homeomorphic\ T \rangle\ homeomorphic\_locally\ homeomorphic\_path\_connectedness$ 
by blast
obtain  $Ta\ where\ (openin\ (top\_of\_set\ U)\ Ta \wedge T\ retract\_of\ Ta)$ 
using  $ANR\_def\ UT\ \langle S\ homeomorphic\ T \rangle\ assms\ by\ atomize\_elim\ (auto\ simp:$ 
 $choice)$ 
then show  $?thesis$ 
using  $S\ \langle U \neq \{\} \rangle\ by\ blast$ 
qed

```

lemma $ANR_imp_locally_connected:$

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $ANR\ S$ 
shows  $locally\ connected\ S$ 
using  $locally\_path\_connected\_imp\_locally\_connected\ ANR\_imp\_locally\_path\_connected$ 
 $assms\ by\ auto$ 

```

lemma $AR_imp_locally_path_connected:$

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $AR\ S$ 
shows  $locally\ path\_connected\ S$ 
by ( $simp\ add: ANR\_imp\_locally\_path\_connected\ AR\_imp\_ANR\ assms$ )

```

lemma $AR_imp_locally_connected:$

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $AR\ S$ 
shows  $locally\ connected\ S$ 
using  $ANR\_imp\_locally\_connected\ AR\_ANR\ assms\ by\ blast$ 

```

lemma $ENR_imp_locally_path_connected:$

```

fixes  $S :: 'a::euclidean\_space\ set$ 
assumes  $ENR\ S$ 
shows  $locally\ path\_connected\ S$ 
by ( $simp\ add: ANR\_imp\_locally\_path\_connected\ ENR\_imp\_ANR\ assms$ )

```

```

lemma ENR_imp_locally_connected:
  fixes S :: 'a::euclidean_space set
  assumes ENR S
  shows locally_connected S
using ANR_imp_locally_connected ENR_ANR assms by blast

lemma ANR_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes ANR S ANR T shows ANR(S × T)
proof (clarsimp simp only: ANR_eq_absolute_neighbourhood_extensor)
  fix f :: ('a × 'b) × real ⇒ 'a × 'b and U C
  assume continuous_on C f and fim: f ∈ C → S × T
  and cloUC: closedin (top_of_set U) C
  have contf1: continuous_on C (fst ∘ f)
  by (simp add: ⟨continuous_on C f⟩ continuous_on_fst)
  obtain W1 g where C ⊆ W1
  and UW1: openin (top_of_set U) W1
  and contg: continuous_on W1 g
  and gim: g ' W1 ⊆ S
  and geq: ∧x. x ∈ C ⇒ g x = (fst ∘ f) x
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ⟨ANR S⟩ contf1
  _ cloUC])
    show (fst ∘ f) ∈ C → S
    using fim by force
  qed auto
  have contf2: continuous_on C (snd ∘ f)
  by (simp add: ⟨continuous_on C f⟩ continuous_on_snd)
  obtain W2 h where C ⊆ W2
  and UW2: openin (top_of_set U) W2
  and conth: continuous_on W2 h
  and him: h ∈ W2 → T
  and heq: ∧x. x ∈ C ⇒ h x = (snd ∘ f) x
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ⟨ANR T⟩ contf2
  _ cloUC])
    show (snd ∘ f) ∈ C → T
    using fim by force
  qed auto
  show ∃ V g. C ⊆ V ∧
  openin (top_of_set U) V ∧
  continuous_on V g ∧ g ∈ V → S × T ∧ (∀ x ∈ C. g x = f x)
  proof (intro exI conjI)
    show C ⊆ W1 ∩ W2
    by (simp add: ⟨C ⊆ W1⟩ ⟨C ⊆ W2⟩)
    show openin (top_of_set U) (W1 ∩ W2)
    by (simp add: UW1 UW2 openin_Int)
    show continuous_on (W1 ∩ W2) (λx. (g x, h x))
    by (metis (no_types) contg conth continuous_on_Pair continuous_on_subset
  inf_commute inf_le1)
  end
end

```

```

  show  $(\lambda x. (g\ x, h\ x)) \in (W1 \cap W2) \rightarrow S \times T$ 
    using gim him by blast
  show  $(\forall x \in C. (g\ x, h\ x) = f\ x)$ 
    using geq heq by auto
qed
qed

lemma AR_Times:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes AR S AR T shows  $AR(S \times T)$ 
  using assms by (simp add: AR_ANR ANR_Times contractible_Times)

```

9.26.2 More advanced properties of ANRs and ENRs

```

lemma ENR_rel_frontier_convex:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes bounded S convex S
  shows  $ENR(\text{rel\_frontier } S)$ 
proof (cases S = {})
  case True then show ?thesis
    by simp
next
  case False
  with assms have  $\text{rel\_interior } S \neq \{\}$ 
    by (simp add: rel_interior_eq_empty)
  then obtain  $a$  where  $a \in \text{rel\_interior } S$ 
    by auto
  have  $\text{ah}S: \text{affine hull } S - \{a\} \subseteq \{x. \text{closest\_point } (\text{affine hull } S)\ x \neq a\}$ 
    by (auto simp: closest_point_self)
  have  $\text{rel\_frontier } S \text{ retract\_of } \text{affine hull } S - \{a\}$ 
    by (simp add: assms a rel_frontier_retract_of_punctured_affine_hull)
  also have  $\dots \text{ retract\_of } \{x. \text{closest\_point } (\text{affine hull } S)\ x \neq a\}$ 
    unfolding retract_of_def retraction_def ahS
    apply (rule_tac x=closest_point (affine hull S) in exI)
    apply (auto simp: False closest_point_self affine_imp_convex closest_point_in_set
      continuous_on_closest_point)
    done
  finally have  $\text{rel\_frontier } S \text{ retract\_of } \{x. \text{closest\_point } (\text{affine hull } S)\ x \neq a\}$ .
  moreover have  $\text{openin } (\text{top\_of\_set } UNIV) (UNIV \cap \text{closest\_point } (\text{affine hull } S) - \{-\{a\}\})$ 
    by (intro continuous_openin_preimage_gen) (auto simp: False affine_imp_convex
      continuous_on_closest_point)
  ultimately show ?thesis
    by (meson ENR_convex_closed ENR_delete ENR_retract_of_ENR <rel_frontier
      S retract_of affine hull S - {a}>
      closed_affine_hull convex_affine_hull)
qed

```

```

lemma ANR_rel_frontier_convex:

```

```

fixes  $S :: 'a::\text{euclidean\_space set}$ 
assumes  $\text{bounded } S \text{ convex } S$ 
shows  $\text{ANR}(\text{rel\_frontier } S)$ 
by (simp add: ENR_imp_ANR ENR_rel_frontier_convex assms)

lemma  $\text{ENR\_closedin\_Un\_local}$ :
fixes  $S :: 'a::\text{euclidean\_space set}$ 
shows  $\llbracket \text{ENR } S; \text{ENR } T; \text{ENR}(S \cap T);$ 
 $\text{closedin } (\text{top\_of\_set } (S \cup T)) S; \text{closedin } (\text{top\_of\_set } (S \cup T)) T \rrbracket$ 
 $\implies \text{ENR}(S \cup T)$ 
by (simp add: ENR_ANR ANR_closed_Un_local locally_compact_closedin_Un)

lemma  $\text{ENR\_closed\_Un}$ :
fixes  $S :: 'a::\text{euclidean\_space set}$ 
shows  $\llbracket \text{closed } S; \text{closed } T; \text{ENR } S; \text{ENR } T; \text{ENR}(S \cap T) \rrbracket \implies \text{ENR}(S \cup T)$ 
by (auto simp: closed_subset ENR_closedin_Un_local)

lemma  $\text{absolute\_retract\_Un}$ :
fixes  $S :: 'a::\text{euclidean\_space set}$ 
shows  $\llbracket S \text{ retract\_of } \text{UNIV}; T \text{ retract\_of } \text{UNIV}; (S \cap T) \text{ retract\_of } \text{UNIV} \rrbracket$ 
 $\implies (S \cup T) \text{ retract\_of } \text{UNIV}$ 
by (meson AR_closed_Un_local_aux closed_subset retract_of_UNIV retract_of_imp_subset)

lemma  $\text{retract\_from\_Un\_Int}$ :
fixes  $S :: 'a::\text{euclidean\_space set}$ 
assumes  $\text{clS}: \text{closedin } (\text{top\_of\_set } (S \cup T)) S$ 
and  $\text{clT}: \text{closedin } (\text{top\_of\_set } (S \cup T)) T$ 
and  $\text{Un}: (S \cup T) \text{ retract\_of } U$  and  $\text{Int}: (S \cap T) \text{ retract\_of } T$ 
shows  $S \text{ retract\_of } U$ 
proof –
obtain  $r$  where  $r: \text{continuous\_on } T \text{ } r \text{ } r \text{ } 'T \subseteq S \cap T \forall x \in S \cap T. r \text{ } x = x$ 
using  $\text{Int}$  by (auto simp: retraction_def retract_of_def)
have  $S \text{ retract\_of } S \cup T$ 
unfolding  $\text{retraction\_def retract\_of\_def}$ 
proof (intro exI conjI)
show  $\text{continuous\_on } (S \cup T) (\lambda x. \text{if } x \in S \text{ then } x \text{ else } r \text{ } x)$ 
using  $r$  by (intro continuous_on_cases_local [OF clS clT] auto)
qed (use r in auto)
also have  $\dots \text{ retract\_of } U$ 
by (rule Un)
finally show  $?thesis$  .
qed

lemma  $\text{AR\_from\_Un\_Int\_local}$ :
fixes  $S :: 'a::\text{euclidean\_space set}$ 
assumes  $\text{clS}: \text{closedin } (\text{top\_of\_set } (S \cup T)) S$ 
and  $\text{clT}: \text{closedin } (\text{top\_of\_set } (S \cup T)) T$ 
and  $\text{Un}: \text{AR}(S \cup T)$  and  $\text{Int}: \text{AR}(S \cap T)$ 
shows  $\text{AR } S$ 

```

by (meson AR_imp_retract AR_retract_of_AR Un assms closedin_closed_subset local.Int

retract_from_Un_Int retract_of_refl sup_ge2)

lemma AR_from_Un_Int_local':

fixes $S :: 'a::euclidean_space$ set

assumes closedin (top_of_set (S \cup T)) S

and closedin (top_of_set (S \cup T)) T

and AR(S \cup T) AR(S \cap T)

shows AR T

using AR_from_Un_Int_local [of T S] assms by (simp add: Un_commute Int_commute)

lemma AR_from_Un_Int:

fixes $S :: 'a::euclidean_space$ set

assumes clo: closed S closed T and Un: AR(S \cup T) and Int: AR(S \cap T)

shows AR S

by (metis AR_from_Un_Int_local [OF __ Un Int] Un_commute clo closed_closedin closedin_closed_subset inf_sup_absorb subtopology_UNIV top_greatest)

lemma ANR_from_Un_Int_local:

fixes $S :: 'a::euclidean_space$ set

assumes clS: closedin (top_of_set (S \cup T)) S

and clT: closedin (top_of_set (S \cup T)) T

and Un: ANR(S \cup T) and Int: ANR(S \cap T)

shows ANR S

proof -

obtain V where clo: closedin (top_of_set (S \cup T)) (S \cap T)

and ope: openin (top_of_set (S \cup T)) V

and ret: S \cap T retract_of V

using ANR_imp_neighbourhood_retract [OF Int] by (metis clS clT closedin_Int)

then obtain r where r: continuous_on V r and rim: r ' V \subseteq S \cap T and req:

$\forall x \in S \cap T. r x = x$

by (auto simp: retraction_def retract_of_def)

have Vsub: V \subseteq S \cup T

by (meson ope openin_contains_cball)

have Vsup: S \cap T \subseteq V

by (simp add: retract_of_imp_subset ret)

then have eq: S \cup V = ((S \cup T) - T) \cup V

by auto

have eq': S \cup V = S \cup (V \cap T)

using Vsub by blast

have continuous_on (S \cup V \cap T) ($\lambda x. \text{if } x \in S \text{ then } x \text{ else } r x$)

proof (rule continuous_on_cases_local)

show closedin (top_of_set (S \cup V \cap T)) S

using clS closedin_subset_trans inf.boundedE by blast

show closedin (top_of_set (S \cup V \cap T)) (V \cap T)

using clT Vsup by (auto simp: closedin_closed)

show continuous_on (V \cap T) r

```

    by (meson Int_lower1 continuous_on_subset r)
  qed (use req continuous_on_id in auto)
  with rim have S retract_of S ∪ V
    unfolding retraction_def retract_of_def using eq' by fastforce
  then show ?thesis
    using ANR_neighborhood_retract [OF Un]
    using ⟨S ∪ V = S ∪ T - T ∪ V⟩ clT ope by fastforce
qed

```

lemma *ANR_from_Un_Int:*

```

  fixes S :: 'a::euclidean_space set
  assumes clo: closed S closed T and Un: ANR(S ∪ T) and Int: ANR(S ∩ T)
  shows ANR S
  by (metis ANR_from_Un_Int_local [OF __ Un Int] Un_commute clo closed_closedin
closedin_closed_subset inf_sup_absorb subtopology_UNIV top_greatest)

```

lemma *ANR_finite_Union_convex_closed:*

```

  fixes T :: 'a::euclidean_space set set
  assumes T: finite T and clo:  $\bigwedge C. C \in T \implies \text{closed } C$  and con:  $\bigwedge C. C \in T \implies \text{convex } C$ 
  shows ANR( $\bigcup T$ )
  proof -
    have ANR( $\bigcup T$ ) if card T < n for n
    using assms that
    proof (induction n arbitrary: T)
      case 0 then show ?case by simp
    next
      case (Suc n)
      have ANR( $\bigcup U$ ) if finite U U  $\subseteq$  T for U
      using that
      proof (induction U)
        case empty
        then show ?case by simp
      next
        case (insert C U)
        have ANR (C  $\cup$   $\bigcup U$ )
        proof (rule ANR_closed_Un)
          show ANR (C  $\cap$   $\bigcup U$ )
            unfolding Int_Union
          proof (rule Suc)
            show finite (( $\cap$ ) C ' U)
              by (simp add: insert.hyps(1))
            show  $\bigwedge Ca. Ca \in (\cap) C ' U \implies \text{closed } Ca$ 
              by (metis (no_types, opaque_lifting) Suc.prem(2) closed_Int subsetD
imageE insert.prem insertI1 insertI2)
            show  $\bigwedge Ca. Ca \in (\cap) C ' U \implies \text{convex } Ca$ 
              by (metis (mono_tags, lifting) Suc.prem(3) convex_Int imageE insert.prem insert_subset subsetCE)
            show card (( $\cap$ ) C ' U) < n
          end
        end
      end
    end
  end

```

```

proof –
  have  $\text{card } \mathcal{T} \leq n$ 
    by (meson Suc.prems(4) not_less not_less_eq)
  then show ?thesis
    by (metis Suc.prems(1) card_image_le card_seteq insert.hyps insert.prems insert_subset le_trans not_less)
  qed
qed
show closed ( $\bigcup \mathcal{U}$ )
  using Suc.prems(2) insert.hyps(1) insert.prems by blast
qed (use Suc.prems convex_imp_ANR insert.prems insert.IH in auto)
then show ?case
  by simp
qed
then show ?case
  using Suc.prems(1) by blast
qed
then show ?thesis
  by blast
qed

```

lemma *finite_imp_ANR*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes finite  $S$ 
shows ANR  $S$ 
proof –
  have ANR( $\bigcup x \in S. \{x\}$ )
    by (blast intro: ANR_finite_Union_convex_closed assms)
  then show ?thesis
    by simp
qed

```

lemma *ANR_insert*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
assumes ANR  $S$  closed  $S$ 
shows ANR(insert  $a$   $S$ )
by (metis ANR_closed_Un ANR_empty ANR_singleton Diff_disjoint Diff_insert_absorb
assms closed_singleton insert_absorb insert_is_Un)

```

lemma *ANR_path_component_ANR*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
shows ANR  $S \implies \text{ANR}(\text{path\_component\_set } S \ x)$ 
using ANR_imp_locally_path_connected ANR_openin openin_path_component_locally_path_connected
by blast

```

lemma *ANR_connected_component_ANR*:

```

fixes  $S :: 'a::\text{euclidean\_space}$  set
shows ANR  $S \implies \text{ANR}(\text{connected\_component\_set } S \ x)$ 

```

by (*metis ANR_openin openin_connected_component_locally_connected ANR_imp_locally_connected*)

lemma *ANR_component_ANR*:

fixes $S :: 'a::\text{euclidean_space set}$

assumes $ANR\ S\ c \in \text{components } S$

shows $ANR\ c$

by (*metis ANR_connected_component_ANR assms componentsE*)

9.26.3 Original ANR material, now for ENRs

lemma *ENR_bounded*:

fixes $S :: 'a::\text{euclidean_space set}$

assumes $\text{bounded } S$

shows $ENR\ S \longleftrightarrow (\exists U. \text{open } U \wedge \text{bounded } U \wedge S \text{ retract_of } U)$

(**is** $?lhs = ?rhs$)

proof

obtain r **where** $0 < r$ **and** $r: S \subseteq \text{ball } 0\ r$

using *bounded_subset_ballD assms* **by** *blast*

assume $?lhs$

then show $?rhs$

by (*meson ENR_def Elementary_Metric_Spaces.open_ball bounded_Int bounded_ball inf_le2 le_inf_iff*)

open_Int r retract_of_imp_subset retract_of_subset)

next

assume $?rhs$

then show $?lhs$

using *ENR_def* **by** *blast*

qed

lemma *absolute_retract_imp_AR_gen*:

fixes $S :: 'a::\text{euclidean_space set}$ **and** $S' :: 'b::\text{euclidean_space set}$

assumes $S \text{ retract_of } T \text{ convex } T\ T \neq \{\}$ $S \text{ homeomorphic } S' \text{ closedin } (\text{top_of_set } U) S'$

shows $S' \text{ retract_of } U$

proof –

have $AR\ T$

by (*simp add: assms convex_imp_AR*)

then have $AR\ S$

using *AR_retract_of_AR assms* **by** *auto*

then show $?thesis$

using *assms AR_imp_absolute_retract* **by** *metis*

qed

lemma *absolute_retract_imp_AR*:

fixes $S :: 'a::\text{euclidean_space set}$ **and** $S' :: 'b::\text{euclidean_space set}$

assumes $S \text{ retract_of } UNIV\ S \text{ homeomorphic } S' \text{ closed } S'$

shows $S' \text{ retract_of } UNIV$

using *AR_imp_absolute_retract_UNIV assms retract_of_UNIV* **by** *blast*

lemma *homeomorphic_compact_arness*:

fixes $S :: 'a::\text{euclidean_space set}$ **and** $S' :: 'b::\text{euclidean_space set}$
assumes S *homeomorphic* S'
shows $\text{compact } S \wedge S \text{ retract_of UNIV} \longleftrightarrow \text{compact } S' \wedge S' \text{ retract_of UNIV}$
using *assms homeomorphic_compactness*
by (*metis compact_AR homeomorphic_AR_iff_AR*)

lemma *absolute_retract_from_Un_Int*:

fixes $S :: 'a::\text{euclidean_space set}$
assumes $(S \cup T) \text{ retract_of UNIV}$ $(S \cap T) \text{ retract_of UNIV}$ *closed* S *closed* T
shows $S \text{ retract_of UNIV}$
using *AR_from_Un_Int assms retract_of_UNIV* **by** *auto*

lemma *ENR_from_Un_Int_gen*:

fixes $S :: 'a::\text{euclidean_space set}$
assumes *closedin* (*top_of_set* $(S \cup T)$) S *closedin* (*top_of_set* $(S \cup T)$) T
 $\text{ENR}(S \cup T)$ $\text{ENR}(S \cap T)$
shows $\text{ENR } S$
by (*meson ANR_from_Un_Int_local ANR_imp_neighbourhood_retract ENR_ANR ENR_neighborhood_retract assms*)

lemma *ENR_from_Un_Int*:

fixes $S :: 'a::\text{euclidean_space set}$
assumes *closed* S *closed* T $\text{ENR}(S \cup T)$ $\text{ENR}(S \cap T)$
shows $\text{ENR } S$
by (*meson ENR_from_Un_Int_gen assms closed_subset sup_ge1 sup_ge2*)

lemma *ENR_finite_Union_convex_closed*:

fixes $\mathcal{T} :: 'a::\text{euclidean_space set set}$
assumes T *finite* \mathcal{T} **and** *clo*: $\bigwedge C. C \in \mathcal{T} \implies \text{closed } C$ **and** *con*: $\bigwedge C. C \in \mathcal{T} \implies \text{convex } C$
shows $\text{ENR}(\bigcup \mathcal{T})$
by (*simp add: ENR_ANR ANR_finite_Union_convex_closed T clo closed_Union closed_imp_locally_compact con*)

lemma *finite_imp_ENR*:

fixes $S :: 'a::\text{euclidean_space set}$
shows *finite* $S \implies \text{ENR } S$
by (*simp add: ENR_ANR finite_imp_ANR finite_imp_closed closed_imp_locally_compact*)

lemma *ENR_insert*:

fixes $S :: 'a::\text{euclidean_space set}$
assumes *closed* S $\text{ENR } S$
shows $\text{ENR}(\text{insert } a \ S)$

proof –

have $\text{ENR } (\{a\} \cup S)$
by (*metis ANR_insert ENR_ANR Un_commute Un_insert_right assms closed_imp_locally_compact*)

3542

closed insert sup bot right)

then show *?thesis*

by *auto*

qed

lemma *ENR_path_component_ENR*:

fixes *S :: 'a::euclidean_space set*

assumes *ENR S*

shows *ENR(path_component_set S x)*

by (*metis ANR_imp_locally_path_connected ENR_empty ENR_imp_ANR ENR_openin assms*

locally_path_connected_2 openin_subtopology_self path_component_eq_empty)

9.26.4 Finally, spheres are ANRs and ENRs

lemma *absolute_retract_homeomorphic_convex_compact*:

fixes *S :: 'a::euclidean_space set* **and** *U :: 'b::euclidean_space set*

assumes *S homeomorphic U S ≠ {} S ⊆ T convex U compact U*

shows *S retract_of T*

by (*metis UNIV_I assms compact_AR convex_imp_AR homeomorphic_AR_iff_AR homeomorphic_compactness homeomorphic_empty(1) retract_of_subset subsetI*)

lemma *frontier_retract_of_punctured_universe*:

fixes *S :: 'a::euclidean_space set*

assumes *convex S bounded S a ∈ interior S*

shows (*frontier S*) *retract_of (- {a})*

using *rel_frontier_retract_of_punctured_affine_hull*

by (*metis Compl_eq_Diff_UNIV affine_hull_nonempty_interior assms empty_iff rel_frontier_frontier rel_interior_nonempty_interior*)

lemma *sphere_retract_of_punctured_universe_gen*:

fixes *a :: 'a::euclidean_space*

assumes *b ∈ ball a r*

shows *sphere a r retract_of (- {b})*

proof –

have *frontier (cball a r) retract_of (- {b})*

using *assms frontier_retract_of_punctured_universe interior_cball* **by** *blast*

then show *?thesis*

by *simp*

qed

lemma *sphere_retract_of_punctured_universe*:

fixes *a :: 'a::euclidean_space*

assumes *0 < r*

shows *sphere a r retract_of (- {a})*

by (*simp add: assms sphere_retract_of_punctured_universe_gen*)

lemma *ENR_sphere*:

fixes *a :: 'a::euclidean_space*

```

  shows ENR(sphere a r)
proof (cases 0 < r)
  case True
  then have sphere a r retract_of -{a}
    by (simp add: sphere_retract_of_punctured_universe)
  with open_delete show ?thesis
    by (auto simp: ENR_def)
next
  case False
  then show ?thesis
    using finite_imp_ENR
    by (metis finite_insert infinite_imp_nonempty less_linear sphere_eq_empty
sphere_trivial)
qed

```

```

corollary ANR_sphere:
  fixes a :: 'a::euclidean_space
  shows ANR(sphere a r)
  by (simp add: ENR_imp_ANR ENR_sphere)

```

9.26.5 Spheres are connected, etc

```

lemma locally_path_connected_sphere_gen:
  fixes S :: 'a::euclidean_space set
  assumes bounded S and convex S
  shows locally_path_connected (rel_frontier S)
proof (cases rel_interior S = {})
  case True
  with assms show ?thesis
    by (simp add: rel_interior_eq_empty)
next
  case False
  then obtain a where a: a ∈ rel_interior S
    by blast
  show ?thesis
  proof (rule retract_of_locally_path_connected)
    show locally_path_connected (affine hull S - {a})
      by (meson convex_affine_hull convex_imp_locally_path_connected locally_open_subset
openin_delete openin_subtopology_self)
    show rel_frontier S retract_of affine hull S - {a}
      using a assms rel_frontier_retract_of_punctured_affine_hull by blast
  qed
qed

```

```

lemma locally_connected_sphere_gen:
  fixes S :: 'a::euclidean_space set
  assumes bounded S and convex S
  shows locally_connected (rel_frontier S)
  by (simp add: ANR_imp_locally_connected ANR_rel_frontier_convex assms)

```

```

lemma locally_path_connected_sphere:
  fixes a :: 'a::euclidean_space
  shows locally_path_connected (sphere a r)
  using ENR_imp_locally_path_connected ENR_sphere by blast

```

```

lemma locally_connected_sphere:
  fixes a :: 'a::euclidean_space
  shows locally_connected(sphere a r)
  using ANR_imp_locally_connected ANR_sphere by blast

```

9.26.6 Borsuk homotopy extension theorem

It's only this late so we can use the concept of retraction, saying that the domain sets or range set are ENRs.

```

theorem Borsuk_homotopy_extension_homotopic:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes cloTS: closedin (top_of_set T) S
    and anr: (ANR S  $\wedge$  ANR T)  $\vee$  ANR U
    and contf: continuous_on T f
    and f  $\in$  T  $\rightarrow$  U
    and homotopic_with_canon ( $\lambda$ x. True) S U f g
  obtains g' where homotopic_with_canon ( $\lambda$ x. True) T U f g'
    continuous_on T g' image g' T  $\subseteq$  U
     $\wedge$ x. x  $\in$  S  $\implies$  g' x = g x

```

proof –

```

  have S  $\subseteq$  T using assms closedin_imp_subset by blast
  obtain h where conth: continuous_on ({0..1}  $\times$  S) h
    and him: h  $\in$  ({0..1}  $\times$  S)  $\rightarrow$  U
    and [simp]:  $\wedge$ x. h(0, x) = f x  $\wedge$ x. h(1::real, x) = g x
    using assms by (fastforce simp: homotopic_with_def)
  define h' where h'  $\equiv$   $\lambda$ z. if snd z  $\in$  S then h z else (f  $\circ$  snd) z
  define B where B  $\equiv$  {0::real}  $\times$  T  $\cup$  {0..1}  $\times$  S
  have clo0T: closedin (top_of_set ({0..1}  $\times$  T)) ({0::real}  $\times$  T)
    by (simp add: Abstract_Topology.closedin_Times)
  moreover have cloT1S: closedin (top_of_set ({0..1}  $\times$  T)) ({0..1}  $\times$  S)
    by (simp add: Abstract_Topology.closedin_Times assms)
  ultimately have clo0TB: closedin (top_of_set ({0..1}  $\times$  T)) B
    by (auto simp: B_def)
  have cloBS: closedin (top_of_set B) ({0..1}  $\times$  S)
    by (metis (no_types) Un_subset_iff B_def closedin_subset_trans [OF cloT1S]
      clo0TB closedin_imp_subset closedin_self)
  moreover have cloBT: closedin (top_of_set B) ({0}  $\times$  T)
    using  $\langle$ S  $\subseteq$  T $\rangle$  closedin_subset_trans [OF clo0T]
    by (metis B_def Un_upper1 clo0TB closedin_closed inf_le1)
  moreover have continuous_on ({0}  $\times$  T) (f  $\circ$  snd)
  proof (rule continuous_intros)+
    show continuous_on (snd  $\circ$  {0}  $\times$  T) f
      by (simp add: contf)

```

```

qed
ultimately have continuous_on ( $\{0..1\} \times S \cup \{0\} \times T$ ) ( $\lambda x$ . if snd  $x \in S$  then
h  $x$  else ( $f \circ \text{snd}$ )  $x$ )
  by (auto intro!: continuous_on_cases_local conth simp: B_def Un_commute
[of  $\{0\} \times T$ ])
then have conth': continuous_on  $B$   $h'$ 
  by (simp add: h'_def B_def Un_commute [of  $\{0\} \times T$ ])
have image  $h' B \subseteq U$ 
  using  $\langle f \in T \rightarrow U \rangle$  him by (auto simp: h'_def B_def)
obtain  $V k$  where  $B \subseteq V$  and opeTV: openin (top_of_set ( $\{0..1\} \times T$ ))  $V$ 
  and contk: continuous_on  $V k$  and kim:  $k \in V \rightarrow U$ 
  and keq:  $\bigwedge x. x \in B \implies k x = h' x$ 
using anr
proof
  assume ST: ANR  $S \wedge \text{ANR } T$ 
  have eq: ( $\{0\} \times T \cap \{0..1\} \times S$ ) =  $\{0::\text{real}\} \times S$ 
    using  $\langle S \subseteq T \rangle$  by auto
  have ANR B
    unfolding B_def
  proof (rule ANR_closed_Un_local)
    show closedin (top_of_set ( $\{0\} \times T \cup \{0..1\} \times S$ )) ( $\{0::\text{real}\} \times T$ )
      by (metis cloBT B_def)
    show closedin (top_of_set ( $\{0\} \times T \cup \{0..1\} \times S$ )) ( $\{0..1::\text{real}\} \times S$ )
      by (metis Un_commute cloBS B_def)
  qed (simp_all add: ANR_Times convex_imp_ANR ANR_singleton ST eq)
  note  $Vk = \text{that}$ 
  have  $*$ : thesis if openin (top_of_set ( $\{0..1::\text{real}\} \times T$ ))  $V$ 
    retraction  $V B r$  for  $V r$ 
  proof –
    have continuous_on  $V$  ( $h' \circ r$ )
      using conth' continuous_on_compose retractionE that(2) by blast
    moreover have ( $h' \circ r$ ) ' $V \subseteq U$ '
      by (metis  $\langle h' ' B \subseteq U \rangle$  image_comp retractionE that(2))
    ultimately show ?thesis
    using  $Vk$  [of  $V h' \circ r$ ] by (metis comp_apply retraction image_subset_iff_funcset
that)
  qed
  show thesis
  by (meson  $*$  ANR_imp_neighbourhood_retract  $\langle \text{ANR } B \rangle$  clo0TB retract_of_def)
next
  assume ANR U
  with ANR_imp_absolute_neighbourhood_extensor  $\langle h' ' B \subseteq U \rangle$  clo0TB conth'
image_subset_iff_funcset that
  show ?thesis
  by (smt (verit) Pi_I funcset_mem)
qed
define  $S'$  where  $S' \equiv \{x. \exists u::\text{real}. u \in \{0..1\} \wedge (u, x::'a) \in \{0..1\} \times T - V\}$ 
have closedin (top_of_set  $T$ )  $S'$ 
  unfolding  $S'_\text{def}$  using closedin_self opeTV

```

```

    by (blast intro: closedin_compact_projection)
  have S'_def: S' = {x.  $\exists u::real. (u, x)::'a \in \{0..1\} \times T - V$ }
    by (auto simp: S'_def)
  have cloTS': closedin (top_of_set T) S'
    using S'_def <closedin (top_of_set T) S'> by blast
  have S  $\cap$  S' = {}
    using S'_def B_def <B  $\subseteq$  V> by force
  obtain a :: 'a  $\Rightarrow$  real where conta: continuous_on T a
    and  $\bigwedge x. x \in T \Rightarrow a x \in \text{closed\_segment } 1 \ 0$ 
    and a1:  $\bigwedge x. x \in S \Rightarrow a x = 1$ 
    and a0:  $\bigwedge x. x \in S' \Rightarrow a x = 0$ 
  by (rule Urysohn_local [OF cloTS cloTS' <S  $\cap$  S' = {}>, of 1 0], blast)
  then have ain:  $\bigwedge x. x \in T \Rightarrow a x \in \{0..1\}$ 
    using closed_segment_eq_real_ivl by auto
  have inV: (u * a t, t)  $\in$  V if t  $\in$  T 0  $\leq$  u u  $\leq$  1 for t u
  proof (rule ccontr)
    assume (u * a t, t)  $\notin$  V
    with ain [OF <t  $\in$  T>] have a t = 0
      apply simp
      by (metis (no_types, lifting) a0 DiffI S'_def SigmaI atLeastAtMost_iff
        mem_Collect_eq mult_le_one mult_nonneg_nonneg that)
    show False
      using B_def <(u * a t, t)  $\notin$  V> <B  $\subseteq$  V> <a t = 0> that by auto
  qed
  show ?thesis
  proof
    show hom: homotopic_with_canon ( $\lambda x. \text{True}$ ) T U f ( $\lambda x. k (a x, x)$ )
    proof (simp add: homotopic_with, intro exI conjI)
      show continuous_on ( $\{0..1\} \times T$ ) (k  $\circ$  ( $\lambda z. (\text{fst } z *_{\mathbb{R}} (a \circ \text{snd}) z, \text{snd } z)$ ))
        apply (intro continuous_on_compose continuous_intros)
      apply (force intro: inV continuous_on_subset [OF contk] continuous_on_subset
        [OF conta])+
      done
      show (k  $\circ$  ( $\lambda z. (\text{fst } z *_{\mathbb{R}} (a \circ \text{snd}) z, \text{snd } z)$ )) ' ( $\{0..1\} \times T$ )  $\subseteq$  U
        using inV kim by auto
      show  $\forall x \in T. (k \circ (\lambda z. (\text{fst } z *_{\mathbb{R}} (a \circ \text{snd}) z, \text{snd } z))) (0, x) = f x$ 
        by (simp add: B_def h'_def keq)
      show  $\forall x \in T. (k \circ (\lambda z. (\text{fst } z *_{\mathbb{R}} (a \circ \text{snd}) z, \text{snd } z))) (1, x) = k (a x, x)$ 
        by auto
    qed
    show continuous_on T ( $\lambda x. k (a x, x)$ )
      using homotopic_with_imp_continuous_maps [OF hom] by auto
    show ( $\lambda x. k (a x, x)$ ) ' T  $\subseteq$  U
    proof clarify
      fix t
      assume t  $\in$  T
      show k (a t, t)  $\in$  U
        by (metis <t  $\in$  T> image_subset_iff inV kim not_one_le_zero linear_mult_cancel_right1
          image_subset_iff_funcset)
    qed
  qed

```

```

qed
show  $\bigwedge x. x \in S \implies k(a\ x, x) = g\ x$ 
  by (simp add: B_def a1 h'_def keq)
qed
qed

```

corollary *nullhomotopic_into_ANR_extension:*

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes closed S
  and conf: continuous_on S f
  and ANR T
  and fim: f ' S  $\subseteq$  T
  and S  $\neq$  {}
shows  $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True})\ S\ T\ f\ (\lambda x. c)) \longleftrightarrow$ 
   $(\exists g. \text{continuous\_on UNIV } g \wedge \text{range } g \subseteq T \wedge (\forall x \in S. g\ x = f\ x))$ 
(is ?lhs = ?rhs)

```

proof

```

assume ?lhs
then obtain c where c: homotopic_with_canon  $(\lambda x. \text{True})\ S\ T\ (\lambda x. c)\ f$ 
  by (blast intro: homotopic_with_symD)
have closedin (top_of_set UNIV) S
  using <closed S> closed_closedin by fastforce
then obtain g where continuous_on UNIV g range g  $\subseteq$  T
   $\bigwedge x. x \in S \implies g\ x = f\ x$ 
proof (rule Borsuk_homotopy_extension_homotopic)
  show  $(\lambda x. c) \in \text{UNIV} \rightarrow T$ 
    using <S  $\neq$  {}> c homotopic_with_imp_subset1 by fastforce
qed (use assms c in auto)
then show ?rhs by blast
next
assume ?rhs
then obtain g where continuous_on UNIV g range g  $\subseteq$  T  $\bigwedge x. x \in S \implies g\ x =$ 
f x
  by blast
then obtain c where homotopic_with_canon  $(\lambda h. \text{True})\ \text{UNIV}\ T\ g\ (\lambda x. c)$ 
  using nullhomotopic_from_contractible [of UNIV g T] contractible_UNIV by
blast
then have homotopic_with_canon  $(\lambda x. \text{True})\ S\ T\ g\ (\lambda x. c)$ 
  by (simp add: homotopic_from_subtopology)
then show ?lhs
  by (force elim: homotopic_with_eq [of _ _ _ g  $\lambda x. c$ ] simp: < $\bigwedge x. x \in S \implies$ 
g x = f x>)
qed

```

corollary *nullhomotopic_into_rel_frontier_extension:*

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes closed S
  and conf: continuous_on S f

```

and *convex* T *bounded* T
and *fm*: $f \text{ ' } S \subseteq \text{rel_frontier } T$
and $S \neq \{\}$
shows $(\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{rel_frontier } T) f (\lambda x. c)) \longleftrightarrow$
 $(\exists g. \text{continuous_on UNIV } g \wedge \text{range } g \subseteq \text{rel_frontier } T \wedge (\forall x \in S. g \ x = f \ x))$
by (*simp add*: *nullhomotopic_into_ANR_extension* *assms ANR_rel_frontier_convex*)

corollary *nullhomotopic_into_sphere_extension*:

fixes $f :: 'a :: \text{euclidean_space} \Rightarrow 'b :: \text{euclidean_space}$
assumes *closed* S **and** *contf*: *continuous_on* S f
and $S \neq \{\}$ **and** *fm*: $f \text{ ' } S \subseteq \text{sphere } a \ r$
shows $((\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{sphere } a \ r) f (\lambda x. c)) \longleftrightarrow$
 $(\exists g. \text{continuous_on UNIV } g \wedge \text{range } g \subseteq \text{sphere } a \ r \wedge (\forall x \in S. g \ x = f \ x)))$
(is ?lhs = ?rhs)
proof (*cases* $r = 0$)
case *True* **with** *fm* **show** *?thesis*
by (*metis ANR_sphere* $\langle \text{closed } S \rangle \langle S \neq \{\} \rangle$ *contfnullhomotopic_into_ANR_extension*)
next
case *False*
then have *eq*: $\text{sphere } a \ r = \text{rel_frontier } (\text{cball } a \ r)$ **by** *simp*
show *?thesis*
using *fm nullhomotopic_into_rel_frontier_extension* [*OF* $\langle \text{closed } S \rangle$ *contf*
convex_cball bounded_cball]
by (*simp add*: $\langle S \neq \{\} \rangle$ *eq*)
qed

proposition *Borsuk_map_essential_bounded_component*:

fixes $a :: 'a :: \text{euclidean_space}$
assumes *compact* S **and** $a \notin S$
shows *bounded* $(\text{connected_component_set } (- S) a) \longleftrightarrow$
 $\neg(\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1)$
 $(\lambda x. \text{inverse}(\text{norm}(x - a)) *_R (x - a)) (\lambda x. c))$
(is ?lhs = ?rhs)
proof (*cases* $S = \{\}$)
case *True* **then show** *?thesis*
by (*simp add*: *homotopic_on_emptyI*)
next
case *False*
have *closed* S *bounded* S
using $\langle \text{compact } S \rangle$ *compact_eq_bounded_closed* **by** *auto*
have *s01*: $(\lambda x. (x - a) /_R \text{norm } (x - a)) \text{ ' } S \subseteq \text{sphere } 0 \ 1$
using $\langle a \notin S \rangle$ **by** *clarsimp* (*metis dist_eq_0_iff dist_norm mult.commute*
right_inverse)
have *aincc*: $a \in \text{connected_component_set } (- S) a$
by (*simp add*: $\langle a \notin S \rangle$)
obtain r **where** $r > 0$ **and** $r: S \subseteq \text{ball } 0 \ r$


```

    using bounded_subset_ballD ‹bounded S› by blast
  have  $\neg ?rhs \longleftrightarrow \neg ?lhs$ 
  proof
    assume notr:  $\neg ?rhs$ 
    have nog:  $\nexists g. \text{continuous\_on } (S \cup \text{connected\_component\_set } (- S) a) g \wedge$ 
       $g \text{ ' } (S \cup \text{connected\_component\_set } (- S) a) \subseteq \text{sphere } 0 \ 1 \wedge$ 
       $(\forall x \in S. g \ x = (x - a) /_R \text{norm } (x - a))$ 
    if bounded (connected_component_set (- S) a)
    using non_extensible_Borsuk_map [OF ‹compact S› componentsI _ aincc]
  ‹a  $\notin S$ › that by auto
    obtain g where range g  $\subseteq \text{sphere } 0 \ 1$  continuous_on UNIV g
       $\bigwedge x. x \in S \implies g \ x = (x - a) /_R \text{norm } (x - a)$ 
    using notr
    by (auto simp: nullhomotopic_into_sphere_extension
      [OF ‹closed S› continuous_on_Borsuk_map [OF ‹a  $\notin S$ › False s01]])
    with ‹a  $\notin S$ › show  $\neg ?lhs$ 
    by (metis UNIV_I continuous_on_subset image_subset_iff nog subsetI)
  next
    assume  $\neg ?lhs$ 
    then obtain b where b:  $b \in \text{connected\_component\_set } (- S) a$  and  $r \leq \text{norm}$ 
    b
      using bounded_iff_linear by blast
    then have bnot:  $b \notin \text{ball } 0 \ r$ 
      by simp
    have homotopic_with_canon  $(\lambda x. \text{True}) \ S \ (\text{sphere } 0 \ 1) \ (\lambda x. (x - a) /_R \text{norm}$ 
     $(x - a))$ 
       $(\lambda x. (x - b) /_R \text{norm } (x - b))$ 
    proof -
      have path_component  $(- S) a \ b$ 
      by (metis (full_types) ‹closed S› b mem_Collect_eq open_CompI open_path_connected_component)
      then show ?thesis
        using Borsuk_maps_homotopic_in_path_component by blast
    qed
  moreover
    obtain c where homotopic_with_canon  $(\lambda x. \text{True}) \ (\text{ball } 0 \ r) \ (\text{sphere } 0 \ 1)$ 
       $(\lambda x. \text{inverse } (\text{norm } (x - b)) *_R (x - b)) \ (\lambda x. c)$ 
    proof (rule nullhomotopic_from_contractible)
      show contractible (ball (0::'a) r)
        by (metis convex_imp_contractible convex_ball)
      show continuous_on (ball 0 r)  $(\lambda x. \text{inverse}(\text{norm } (x - b)) *_R (x - b))$ 
        by (rule continuous_on_Borsuk_map [OF bnot])
      show  $(\lambda x. (x - b) /_R \text{norm } (x - b)) \in \text{ball } 0 \ r \rightarrow \text{sphere } 0 \ 1$ 
        using bnot Borsuk_map_into_sphere by blast
    qed blast
    ultimately have homotopic_with_canon  $(\lambda x. \text{True}) \ S \ (\text{sphere } 0 \ 1) \ (\lambda x. (x -$ 
     $a) /_R \text{norm } (x - a)) \ (\lambda x. c)$ 
      by (meson homotopic_with_subset_left homotopic_with_trans r)
    then show  $\neg ?rhs$ 
      by blast
  end

```

3550

qed
then show *?thesis* **by** *blast*
qed

lemma *homotopic_Borsuk_maps_in_bounded_component*:

fixes $a :: 'a :: \text{euclidean_space}$

assumes *compact* S **and** $a \notin S$ **and** $b \notin S$

and *boc*: *bounded (connected_component_set (- S) a)*

and *hom*: *homotopic_with_canon ($\lambda x. \text{True}$) S (sphere 0 1)*

$(\lambda x. (x - a) /_R \text{norm } (x - a))$

$(\lambda x. (x - b) /_R \text{norm } (x - b))$

shows *connected_component (- S) a b*

proof (*rule ccontr*)

assume *notcc*: $\neg \text{connected_component } (- S) a b$

let $?T = S \cup \text{connected_component_set } (- S) a$

have $\exists g. \text{continuous_on } (S \cup \text{connected_component_set } (- S) a) g \wedge$

$g \in (S \cup \text{connected_component_set } (- S) a) \rightarrow \text{sphere } 0 \ 1 \wedge$

$(\forall x \in S. g \ x = (x - a) /_R \text{norm } (x - a))$

using *non_extensible_Borsuk_map [OF <compact S> _ boc] <a \notin S>*

by (*simp add: componentsI*)

moreover obtain g **where** *continuous_on (S \cup connected_component_set (- S) a) g*

$g \ ' (S \cup \text{connected_component_set } (- S) a) \subseteq \text{sphere } 0 \ 1$

$\wedge x. x \in S \implies g \ x = (x - a) /_R \text{norm } (x - a)$

proof (*rule Borsuk_homotopy_extension_homotopic*)

show *closedin (top_of_set ?T) S*

by (*simp add: <compact S> closed_subset compact_imp_closed*)

show *continuous_on ?T ($\lambda x. (x - b) /_R \text{norm } (x - b)$)*

by (*simp add: <b \notin S> notcc continuous_on_Borsuk_map*)

show $(\lambda x. (x - b) /_R \text{norm } (x - b)) \in ?T \rightarrow \text{sphere } 0 \ 1$

by (*simp add: <b \notin S> notcc Borsuk_map_into_sphere*)

show *homotopic_with_canon ($\lambda x. \text{True}$) S (sphere 0 1)*

$(\lambda x. (x - b) /_R \text{norm } (x - b)) (\lambda x. (x - a) /_R \text{norm } (x - a))$

by (*simp add: hom_homotopic_with_symD*)

qed (*auto simp: ANR_sphere intro: that*)

ultimately show *False* **by** *blast*

qed

lemma *Borsuk_maps_homotopic_in_connected_component_eq*:

fixes $a :: 'a :: \text{euclidean_space}$

assumes S : *compact* S $a \notin S$ $b \notin S$ **and** 2 : $2 \leq \text{DIM}(a)$

shows *(homotopic_with_canon ($\lambda x. \text{True}$) S (sphere 0 1)*

$(\lambda x. (x - a) /_R \text{norm } (x - a))$

$(\lambda x. (x - b) /_R \text{norm } (x - b)) \longleftrightarrow$

connected_component (- S) a b)

(**is** *?lhs = ?rhs*)

proof

assume L : *?lhs*

```

show ?rhs
proof (cases bounded(connected_component_set (- S) a))
  case True
    show ?thesis
    by (rule homotopic_Borsuk_maps_in_bounded_component [OF S True L])
  next
    case not_bo_a: False
    show ?thesis
    proof (cases bounded(connected_component_set (- S) b))
      case True
        show ?thesis
        using homotopic_Borsuk_maps_in_bounded_component [OF S]
        by (simp add: L True assms connected_component_sym homotopic_Borsuk_maps_in_bounded_component
homotopic_with_sym)
      next
        case False
        then show ?thesis
        using cobounded_unique_unbounded_component [of -S a b] ⟨compact S⟩
    not_bo_a
    by (auto simp: compact_eq_bounded_closed assms connected_component_eq_eq)
    qed
  qed
next
  assume R: ?rhs
  then have path_component (- S) a b
  using assms(1) compact_eq_bounded_closed open_Comp open_path_connected_component_set
by fastforce
  then show ?lhs
  by (simp add: Borsuk_maps_homotopic_in_path_component)
qed

```

9.26.7 More extension theorems

```

lemma extension_from_clopen:
  assumes ope: openin (top_of_set S) T
  and clo: closedin (top_of_set S) T
  and contf: continuous_on T f and fim: f ' T ⊆ U and null: U = {} ⇒ S
= {}
  obtains g where continuous_on S g g ' S ⊆ U ∧ x. x ∈ T ⇒ g x = f x
proof (cases U = {})
  case True
    then show ?thesis
    by (simp add: null that)
  next
    case False
    then obtain a where a ∈ U
    by auto
    let ?g = λx. if x ∈ T then f x else a
    have Seq: S = T ∪ (S - T)

```

```

    using clo closedin_imp_subset by fastforce
  show ?thesis
  proof
    have continuous_on (T ∪ (S - T)) ?g
      using Seq clo ope by (intro continuous_on_cases_local) (auto simp: contf)
    with Seq show continuous_on S ?g
      by metis
    show ?g ' S ⊆ U
      using ⟨a ∈ U⟩ fim by auto
    show ∧x. x ∈ T ⇒ ?g x = f x
      by auto
  qed
qed

```

```

lemma extension_from_component:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes S: locally_connected S ∨ compact S and ANR U
  and C: C ∈ components S and contf: continuous_on C f and fim: f ∈ C →
  U
  obtains g where continuous_on S g g ∈ S → U ∧x. x ∈ C ⇒ g x = f x
  proof -
    obtain T g where ope: openin (top_of_set S) T
      and clo: closedin (top_of_set S) T
      and C ⊆ T and contg: continuous_on T g and gim: g ∈ T → U
      and gf: ∧x. x ∈ C ⇒ g x = f x
    using S
  proof
    assume locally_connected S
    show ?thesis
      by (metis C ⟨locally_connected S⟩ openin_components_locally_connected
        closedin_component contf fim order_refl that)
    next
      assume compact S
      then obtain W g where C ⊆ W and opeW: openin (top_of_set S) W
        and contg: continuous_on W g
        and gim: g ∈ W → U and gf: ∧x. x ∈ C ⇒ g x = f x
      using ANR_imp_absolute_neighbourhood_extensor [of U C f S] C ⟨ANR U⟩
        closedin_component contf fim by blast
      then obtain V where open V and V: W = S ∩ V
        by (auto simp: openin_open)
      moreover have locally_compact S
        by (simp add: ⟨compact S⟩ closed_imp_locally_compact compact_imp_closed)
      ultimately obtain K where opeK: openin (top_of_set S) K and compact K
        C ⊆ K K ⊆ V
      by (metis C Int_subset_iff ⟨C ⊆ W⟩ ⟨compact S⟩ compact_components
        Sura_Bura_clopen_subset)
    show ?thesis
  proof

```

```

  show closedin (top_of_set S) K
  by (meson <compact K> <compact S> closedin_compact_eq opeK openin_imp_subset)
  show continuous_on K g
    by (metis Int_subset_iff V <K ⊆ V> contg continuous_on_subset opeK
openin_subtopology subset_eq)
  show g ∈ K → U
    using V <K ⊆ V> gim opeK openin_imp_subset by fastforce
  qed (use opeK gf <C ⊆ K> in auto)
  qed
  obtain h where continuous_on S h h ∈ S → U ∧ x. x ∈ T ⇒ h x = g x
    using extension_from_clopen
  by (metis C bot.extremum_uniqueI clo contg gim fim image_is_empty_in_components_nonempty
ope_image_subset_iff_funcset)
  then show ?thesis
    by (metis <C ⊆ T> gf subset_eq that)
  qed

```

lemma *tube_lemma*:

```

  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S and S: S ≠ {} (λx. (x,a)) ' S ⊆ U
    and ope: openin (top_of_set (S × T)) U
  obtains V where openin (top_of_set T) V a ∈ V S × V ⊆ U
  proof -
    let ?W = {y. ∃x. x ∈ S ∧ (x, y) ∈ (S × T - U)}
    have U ⊆ S × T closedin (top_of_set (S × T)) (S × T - U)
      using ope by (auto simp: openin_closedin_eq)
    then have closedin (top_of_set T) ?W
      using <compact S> closedin_compact_projection by blast
    moreover have a ∈ T - ?W
      using <U ⊆ S × T> S by auto
    moreover have S × (T - ?W) ⊆ U
      by auto
    ultimately show ?thesis
      by (metis (no_types, lifting) Sigma_cong closedin_def that topspace_euclidean_subtopology)
  qed

```

lemma *tube_lemma_gen*:

```

  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S S ≠ {} T ⊆ T' S × T ⊆ U
    and ope: openin (top_of_set (S × T')) U
  obtains V where openin (top_of_set T') V T ⊆ V S × V ⊆ U
  proof -
    have ∧x. x ∈ T ⇒ ∃V. openin (top_of_set T') V ∧ x ∈ V ∧ S × V ⊆ U
      using assms by (auto intro: tube_lemma [OF <compact S>])
    then obtain F where F: ∧x. x ∈ T ⇒ openin (top_of_set T') (F x) ∧ x ∈
F x ∧ S × F x ⊆ U
      by metis
    show ?thesis

```

```

proof
  show openin (top_of_set  $T'$ ) ( $\bigcup(F' T)$ )
    using  $F$  by blast
  show  $T \subseteq \bigcup(F' T)$ 
    using  $F$  by blast
  show  $S \times \bigcup(F' T) \subseteq U$ 
    using  $F$  by auto
qed
qed

proposition homotopic_neighbourhood_extension:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes contf: continuous_on  $S f$  and fim:  $f' S \subseteq U$ 
    and contg: continuous_on  $S g$  and gim:  $g' S \subseteq U$ 
    and clo: closedin (top_of_set  $S$ )  $T$ 
    and ANR  $U$  and hom: homotopic_with_canon  $(\lambda x. True) T U f g$ 
  obtains  $V$  where  $T \subseteq V$  openin (top_of_set  $S$ )  $V$ 
    homotopic_with_canon  $(\lambda x. True) V U f g$ 

proof –
  have  $T \subseteq S$ 
    using clo closedin_imp_subset by blast
  obtain  $h$  where conth: continuous_on  $(\{0..1\} \times T) h$ 
    and him:  $h' (\{0..1\} \times T) \subseteq U$ 
    and h0:  $\bigwedge x. h(0, x) = f x$  and h1:  $\bigwedge x. h(1, x) = g x$ 
    using hom by (auto simp: homotopic_with_def)
  define  $h'$  where  $h' \equiv \lambda z. \text{if fst } z \in \{0\} \text{ then } f(\text{snd } z)$ 
     $\text{else if fst } z \in \{1\} \text{ then } g(\text{snd } z)$ 
     $\text{else } h z$ 
  let  $?S0 = \{0::real\} \times S$  and  $?S1 = \{1::real\} \times S$ 
  have continuous_on  $(?S0 \cup (?S1 \cup \{0..1\} \times T)) h'$ 
    unfolding h'_def
  proof (intro continuous_on_cases_local)
    show closedin (top_of_set  $(?S0 \cup (?S1 \cup \{0..1\} \times T))$ )  $?S0$ 
      closedin (top_of_set  $(?S1 \cup \{0..1\} \times T)$ )  $?S1$ 
      using  $\langle T \subseteq S \rangle$  by (force intro: closedin_Times closedin_subset_trans [of
 $\{0..1\} \times S$ ])+
    show closedin (top_of_set  $(?S0 \cup (?S1 \cup \{0..1\} \times T))$ )  $(?S1 \cup \{0..1\} \times T)$ 
      closedin (top_of_set  $(?S1 \cup \{0..1\} \times T)$ )  $(\{0..1\} \times T)$ 
      using  $\langle T \subseteq S \rangle$  by (force intro: clo closedin_Times closedin_subset_trans [of
 $\{0..1\} \times S$ ])+
    show continuous_on  $(?S0)$   $(\lambda x. f(\text{snd } x))$ 
      by (intro continuous_intros continuous_on_compose2 [OF contf]) auto
    show continuous_on  $(?S1)$   $(\lambda x. g(\text{snd } x))$ 
      by (intro continuous_intros continuous_on_compose2 [OF contg]) auto
  qed (use h0 h1 conth in auto)
  then have continuous_on  $(\{0,1\} \times S \cup (\{0..1\} \times T)) h'$ 
    by (metis Sigma_Un_distrib1 Un_assoc insert_is_Un)
  moreover have  $h' ' (\{0,1\} \times S \cup \{0..1\} \times T) \subseteq U$ 
    using fm gim him  $\langle T \subseteq S \rangle$  unfolding h'_def by force

```

```

moreover have closedin (top_of_set ({0..1::real} × S)) ({0,1} × S ∪ {0..1::real}
× T)
  by (intro closedin_Times closedin_Un clo) (simp_all add: closed_subset)
ultimately
obtain W k where W: ({0,1} × S) ∪ ({0..1} × T) ⊆ W
  and opeW: openin (top_of_set ({0..1} × S)) W
  and contk: continuous_on W k
  and kim: k ∈ W → U
  and kh': ∧x. x ∈ ({0,1} × S) ∪ ({0..1} × T) ⇒ k x = h' x
  by (metis ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR U›, of
({0,1} × S) ∪ ({0..1} × T) h' {0..1} × S] image_subset_iff_funcset)
obtain T' where opeT': openin (top_of_set S) T'
  and T ⊆ T' and TW: {0..1} × T' ⊆ W
  using tube_lemma_gen [of {0..1::real} T S W] W ‹T ⊆ S› opeW by auto
moreover have homotopic_with_canon (λx. True) T' U f g
proof (simp add: homotopic_with, intro exI conjI)
  show continuous_on ({0..1} × T') k
    using TW continuous_on_subset contk by auto
  show k '({0..1} × T') ⊆ U
    using TW kim by fastforce
  have T' ⊆ S
    by (meson opeT' subsetD openin_imp_subset)
  then show ∀x∈T'. k (0, x) = f x ∀x∈T'. k (1, x) = g x
    by (auto simp: kh' h'_def)
qed
ultimately show ?thesis
  by (blast intro: that)
qed

```

Homotopy on a union of closed-open sets.

```

proposition homotopic_on_clopen_Union:
  fixes F :: 'a::euclidean_space set set
  assumes ∧S. S ∈ F ⇒ closedin (top_of_set (∪F)) S
    and ∧S. S ∈ F ⇒ openin (top_of_set (∪F)) S
    and ∧S. S ∈ F ⇒ homotopic_with_canon (λx. True) S T f g
  shows homotopic_with_canon (λx. True) (∪F) T f g
proof –
  obtain V where V ⊆ F countable V and eqU: ∪V = ∪F
    using Lindelof_openin_assms by blast
  show ?thesis
proof (cases V = {})
  case True
    then show ?thesis
      by (metis Union_empty eqU homotopic_with_canon_on_empty)
  next
  case False
    then obtain V :: nat ⇒ 'a set where V: range V = V
      using range_from_nat_into ‹countable V› by metis
    with ‹V ⊆ F› have clo: ∧n. closedin (top_of_set (∪F)) (V n)

```

```

    and ope:  $\bigwedge n. \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) (V n)$ 
    and hom:  $\bigwedge n. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (V n) T f g$ 
  using assms by auto
  then obtain h where conth:  $\bigwedge n. \text{continuous\_on } (\{0..1\} \times V n) (h n)$ 
    and him:  $\bigwedge n. h n ' (\{0..1\} \times V n) \subseteq T$ 
    and h0:  $\bigwedge n. \bigwedge x. x \in V n \implies h n (0, x) = f x$ 
    and h1:  $\bigwedge n. \bigwedge x. x \in V n \implies h n (1, x) = g x$ 
  by (simp add: homotopic_with) metis
  have wop:  $b \in V x \implies \exists k. b \in V k \wedge (\forall j < k. b \notin V j)$  for b x
    using nat_less_induct [where P =  $\lambda i. b \notin V i$ ] by meson
  obtain  $\zeta$  where cont:  $\text{continuous\_on } (\{0..1\} \times \bigcup (V ' UNIV)) \zeta$ 
    and eq:  $\bigwedge x i. \llbracket x \in \{0..1\} \times \bigcup (V ' UNIV) \cap \{0..1\} \times (V i - (\bigcup_{m < i} V m)) \rrbracket \implies \zeta x = h i x$ 
  proof (rule pasting_lemma_exists)
    let ?X =  $\text{top\_of\_set } (\{0..1\} \times \bigcup (\text{range } V))$ 
    show  $\text{topspace } ?X \subseteq (\bigcup i. \{0..1\} \times (V i - (\bigcup_{m < i} V m)))$ 
      by (force simp: Ball_def dest: wop)
    show  $\text{openin } (\text{top\_of\_set } (\{0..1\} \times \bigcup (V ' UNIV)))$ 
      ( $\{0..1\} \times (V i - (\bigcup_{m < i} V m))$ ) for i
    proof (intro openin_Times openin_subtopology_self openin_diff)
      show  $\text{openin } (\text{top\_of\_set } (\bigcup (V ' UNIV))) (V i)$ 
        using ope V eqU by auto
      show  $\text{closedin } (\text{top\_of\_set } (\bigcup (V ' UNIV))) (\bigcup_{m < i} V m)$ 
        using V clo eqU by (force intro: closedin_Union)
    qed
    show  $\text{continuous\_map } (\text{subtopology } ?X (\{0..1\} \times (V i - \bigcup (V ' \{..<i\}))))$ 
      euclidean (h i) for i
      by (auto simp add: subtopology_subtopology intro!: continuous_on_subset
        [OF conth])
    show  $\bigwedge i j x. x \in \text{topspace } ?X \cap \{0..1\} \times (V i - (\bigcup_{m < i} V m)) \cap \{0..1\}$ 
       $\times (V j - (\bigcup_{m < j} V m))$ 
         $\implies h i x = h j x$ 
      by clarsimp (metis lessThan_iff linorder_neqE_nat)
  qed auto
  show ?thesis
  proof (simp add: homotopic_with eqU [symmetric], intro exI conjI ballI)
    show  $\text{continuous\_on } (\{0..1\} \times \bigcup \mathcal{V}) \zeta$ 
      using V eqU by (blast intro!: continuous_on_subset [OF cont])
    show  $\zeta ' (\{0..1\} \times \bigcup \mathcal{V}) \subseteq T$ 
  proof clarsimp
    fix t :: real and y :: 'a and X :: 'a set
    assume  $y \in X$   $X \in \mathcal{V}$  and  $t: 0 \leq t \leq 1$ 
    then obtain k where  $y \in V k$  and  $j: \forall j < k. y \notin V j$ 
      by (metis image_iff V wop)
    with him t show  $\zeta(t, y) \in T$ 
      by (subst eq) force+
  qed
  fix X y
  assume  $X \in \mathcal{V}$   $y \in X$ 

```



```

    then obtain  $k$  where  $y \in V k$  and  $j: \forall j < k. y \notin V j$ 
      by (metis image_iff V wop)
    then show  $\zeta(0, y) = f y$  and  $\zeta(1, y) = g y$ 
      by (subst eq [where  $i=k$ ]; force simp: h0 h1)+
  qed
qed
qed

lemma homotopic_on_components_eq:
  fixes  $S :: 'a :: euclidean\_space$  set and  $T :: 'b :: euclidean\_space$  set
  assumes  $S$ : locally connected  $S \vee$  compact  $S$  and ANR  $T$ 
  shows homotopic_with_canon  $(\lambda x. True) S T f g \longleftrightarrow$ 
    (continuous_on  $S f \wedge f ' S \subseteq T \wedge$  continuous_on  $S g \wedge g ' S \subseteq T) \wedge$ 
    ( $\forall C \in$  components  $S. homotopic\_with\_canon (\lambda x. True) C T f g$ )
  (is ?lhs  $\longleftrightarrow$  ?C  $\wedge$  ?rhs)
proof -
  have continuous_on  $S f f ' S \subseteq T$  continuous_on  $S g g ' S \subseteq T$  if ?lhs
    using homotopic_with_imp_continuous homotopic_with_imp_subset1 homotopic_with_imp_subset2 that by blast+
  moreover have ?lhs  $\longleftrightarrow$  ?rhs
    if contf: continuous_on  $S f$  and fim:  $f ' S \subseteq T$  and contg: continuous_on  $S g$ 
and gim:  $g ' S \subseteq T$ 
  proof
    assume ?lhs
    with that show ?rhs
      by (simp add: homotopic_with_subset_left_in_components_subset)
  next
    assume  $R$ : ?rhs
    have  $\exists U. C \subseteq U \wedge$  closedin (top_of_set  $S$ )  $U \wedge$ 
      openin (top_of_set  $S$ )  $U \wedge$ 
      homotopic_with_canon  $(\lambda x. True) U T f g$  if  $C$ :  $C \in$  components  $S$ 
  for  $C$ 
  proof -
    have  $C \subseteq S$ 
      by (simp add: in_components_subset that)
    show ?thesis
      using  $S$ 
    proof
      assume locally connected  $S$ 
      show ?thesis
        proof (intro exI conjI)
          show closedin (top_of_set  $S$ )  $C$ 
            by (simp add: closedin_component that)
          show openin (top_of_set  $S$ )  $C$ 
            by (simp add: locally_connected  $S$ ) openin_components_locally_connected
        that)
      show homotopic_with_canon  $(\lambda x. True) C T f g$ 
        by (simp add:  $R$  that)
    qed auto
  end
end

```

```

next
  assume compact S
  have hom: homotopic_with_canon (λx. True) C T f g
    using R that by blast
  obtain U where C ⊆ U and opeU: openin (top_of_set S) U
    and hom: homotopic_with_canon (λx. True) U T f g
    using homotopic_neighbourhood_extension [OF contf fim contg gim _
  ⟨ANR T⟩ hom]
    ⟨C ∈ components S⟩ closedin_component by blast
  then obtain V where open V and V: U = S ∩ V
    by (auto simp: openin_open)
  moreover have locally_compact S
  by (simp add: ⟨compact S⟩ closed_imp_locally_compact compact_imp_closed)
  ultimately obtain K where opeK: openin (top_of_set S) K and compact
  K C ⊆ K K ⊆ V
    by (metis C Int_subset_iff Sura_Bura_clopen_subset ⟨C ⊆ U⟩ ⟨compact
  S⟩ compact_components)
  show ?thesis
  proof (intro exI conjI)
    show closedin (top_of_set S) K
      by (meson ⟨compact K⟩ ⟨compact S⟩ closedin_compact_eq opeK
  openin_imp_subset)
    show homotopic_with_canon (λx. True) K T f g
      using V ⟨K ⊆ V⟩ hom homotopic_with_subset_left opeK openin_imp_subset
  by fastforce
    qed (use opeK ⟨C ⊆ K⟩ in auto)
  qed
  qed
  then obtain φ where φ: ⋀ C. C ∈ components S ⇒ C ⊆ φ C
    and cloφ: ⋀ C. C ∈ components S ⇒ closedin (top_of_set S) (φ
  C)
    and opeφ: ⋀ C. C ∈ components S ⇒ openin (top_of_set S) (φ
  C)
    and homφ: ⋀ C. C ∈ components S ⇒ homotopic_with_canon
  (λx. True) (φ C) T f g
    by metis
  have Seq: S = ⋃ (φ ` components S)
  proof
    show S ⊆ ⋃ (φ ` components S)
      by (metis Sup_mono Union_components φ imageI)
    show ⋃ (φ ` components S) ⊆ S
      using opeφ openin_imp_subset by fastforce
    qed
  show ?lhs
    apply (subst Seq)
    using Seq cloφ opeφ homφ by (intro homotopic_on_clopen_Union) auto
  qed
  ultimately show ?thesis by blast
  qed

```

lemma *cohomotopically_trivial_on_components*:

fixes $S :: 'a :: \text{euclidean_space set}$ **and** $T :: 'b :: \text{euclidean_space set}$

assumes S : *locally connected* $S \vee$ *compact* S **and** *ANR* T

shows

$(\forall f g. \text{continuous_on } S f \longrightarrow f \in S \rightarrow T \longrightarrow \text{continuous_on } S g \longrightarrow g \in S \rightarrow T \longrightarrow$

$\text{homotopic_with_canon } (\lambda x. \text{True}) S T f g)$

\longleftrightarrow

$(\forall C \in \text{components } S.$

$\forall f g. \text{continuous_on } C f \longrightarrow f \in C \rightarrow T \longrightarrow \text{continuous_on } C g \longrightarrow g \in C \rightarrow T \longrightarrow$

$\text{homotopic_with_canon } (\lambda x. \text{True}) C T f g)$

(is $?lhs = ?rhs)$

proof

assume $L[\text{rule_format}]$: $?lhs$

show $?rhs$

proof *clarify*

fix $C f g$

assume contf : *continuous_on* $C f$ **and** fim : $f \in C \rightarrow T$

and contg : *continuous_on* $C g$ **and** gim : $g \in C \rightarrow T$ **and** C : $C \in \text{components } S$

obtain f' **where** contf' : *continuous_on* $S f'$ **and** $f'im$: $f' \in S \rightarrow T$ **and** $f'f$: $\bigwedge x. x \in C \implies f' x = f x$

using *extension_from_component* $[OF S \langle ANR T \rangle C \text{contf } \text{fim}]$ **by** *metis*

obtain g' **where** contg' : *continuous_on* $S g'$ **and** $g'im$: $g' \in S \rightarrow T$ **and** $g'g$: $\bigwedge x. x \in C \implies g' x = g x$

using *extension_from_component* $[OF S \langle ANR T \rangle C \text{contg } \text{gim}]$ **by** *metis*

have *homotopic_with_canon* $(\lambda x. \text{True}) C T f' g'$

using $L [OF \text{contf}' f'im \text{contg}' g'im]$ *homotopic_with_subset_left* C *in_components_subset*

by *fastforce*

then show *homotopic_with_canon* $(\lambda x. \text{True}) C T f g$

using $f'f g'g$ *homotopic_with_eq* **by** *force*

qed

next

assume $R[\text{rule_format}]$: $?rhs$

show $?lhs$

proof *clarify*

fix $f g$

assume contf : *continuous_on* $S f$ **and** fim : $f \in S \rightarrow T$

and contg : *continuous_on* $S g$ **and** gim : $g \in S \rightarrow T$

moreover have *homotopic_with_canon* $(\lambda x. \text{True}) C T f g$ **if** $C \in \text{components } S$ **for** C

using $R [OF \text{that}]$ *contf contg continuous_on_subset fim gim in_components_subset* **by** *smt* (*verit*, *del_insts*) Pi_anti_mono *subsetD* *that*)

ultimately show *homotopic_with_canon* $(\lambda x. \text{True}) S T f g$

by (*subst homotopic_on_components_eq* $[OF S \langle ANR T \rangle]$) *auto*

qed

qed

9.26.8 The complement of a set and path-connectedness

Complement in dimension $N > 1$ of set homeomorphic to any interval in any dimension is (path-)connected. This naively generalizes the argument in Ryuji Maehara's paper "The Jordan curve theorem via the Brouwer fixed point theorem", American Mathematical Monthly 1984.

lemma *unbounded_components_complement_absolute_retract:*

fixes $S :: 'a::euclidean_space\ set$

assumes $C: C \in components(- S)$ **and** $S: compact\ S\ AR\ S$

shows $\neg bounded\ C$

proof –

obtain y **where** $y: C = connected_component_set\ (-\ S)\ y$ **and** $y \notin S$

using C **by** (*auto simp: components_def*)

have $open(- S)$

using S **by** (*simp add: closed_open_compact_eq_bounded_closed*)

have $S\ retract_of\ UNIV$

using $S\ compact_AR$ **by** *blast*

then obtain r **where** $contr: continuous_on\ UNIV\ r$ **and** $ontor: range\ r \subseteq S$

and $r: \bigwedge x. x \in S \implies r\ x = x$

by (*auto simp: retract_of_def retraction_def*)

show *?thesis*

proof

assume $bounded\ C$

have $connected_component_set\ (-\ S)\ y \subseteq S$

proof (*rule frontier_subset_retraction*)

show $bounded\ (connected_component_set\ (-\ S)\ y)$

using $\langle bounded\ C \rangle\ y$ **by** *blast*

show $frontier\ (connected_component_set\ (-\ S)\ y) \subseteq S$

using $C\ \langle compact\ S \rangle\ compact_eq_bounded_closed\ frontier_of_components_closed_complement$

y **by** *blast*

show $continuous_on\ (closure\ (connected_component_set\ (-\ S)\ y))\ r$

by (*blast intro: continuous_on_subset [OF contr]*)

qed (*use ontor r in auto*)

with $\langle y \notin S \rangle$ **show** *False* **by** *force*

qed

qed

lemma *connected_complement_absolute_retract:*

fixes $S :: 'a::euclidean_space\ set$

assumes $S: compact\ S\ AR\ S$ **and** $2: 2 \leq DIM('a)$

shows $connected(- S)$

proof –

have $S\ retract_of\ UNIV$

using $S\ compact_AR$ **by** *blast*

show *?thesis*

proof (*clarsimp simp: connected_iff_connected_component_eq*)

```

have  $\neg$  bounded (connected_component_set (- S) x) if  $x \notin S$  for x
by (meson Compl_iff assms componentsI that unbounded_components_complement_absolute_retract)
then show connected_component_set (- S) x = connected_component_set
(- S) y
if  $x \notin S$   $y \notin S$  for x y
using cobounded_unique_unbounded_component [OF_2]
by (metis <compact S> compact_imp_bounded double_compl that)
qed
qed

```

```

lemma path_connected_complement_absolute_retract:
fixes S :: 'a::euclidean_space set
assumes compact S AR S  $2 \leq \text{DIM}('a)$ 
shows path_connected(- S)
using connected_complement_absolute_retract [OF assms]
using <compact S> compact_eq_bounded_closed connected_open_path_connected
by blast

```

```

theorem connected_complement_homeomorphic_convex_compact:
fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
assumes hom: S homeomorphic T and T: convex T compact T and 2:  $2 \leq$ 
DIM('a)
shows connected(- S)
proof (cases S = {})
case True
then show ?thesis
by (simp add: connected_UNIV)
next
case False
show ?thesis
proof (rule connected_complement_absolute_retract)
show compact S
using <compact T> hom homeomorphic_compactness by auto
show AR S
by (meson AR_ANR_False <convex T> convex_imp_ANR convex_imp_contractible
hom homeomorphic_ANR_iff_ANR homeomorphic_contractible_eq)
qed (rule 2)
qed

```

```

corollary path_connected_complement_homeomorphic_convex_compact:
fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
assumes hom: S homeomorphic T convex T compact T  $2 \leq \text{DIM}('a)$ 
shows path_connected(- S)
using connected_complement_homeomorphic_convex_compact [OF assms]
using <compact T> compact_eq_bounded_closed connected_open_path_connected
hom homeomorphic_compactness by blast

```

```

lemma path_connected_complement_homeomorphic_interval:
fixes S :: 'a::euclidean_space set

```

3562

```
  assumes  $S$  homeomorphic cbox  $a$   $b$   $2 \leq DIM('a)$   
  shows path_connected( $-S$ )  
  using assms compact_cbox convex_box(1) path_connected_complement_homeomorphic_convex_comp  
  by blast
```

```
lemma connected_complement_homeomorphic_interval:  
  fixes  $S :: 'a::euclidean\_space$  set  
  assumes  $S$  homeomorphic cbox  $a$   $b$   $2 \leq DIM('a)$   
  shows connected( $-S$ )  
  using assms path_connected_complement_homeomorphic_interval path_connected_imp_connected  
  by blast
```

end

9.27 Extending Continous Maps, Invariance of Domain, etc

Ported from HOL Light (moretop.ml) by L C Paulson

```
theory Further_Topology  
  imports Weierstrass_Theorems Polytope Complex_Transcendental Equivalence_Lebesgue_Henstock_I  
  Retracts  
begin
```

9.27.1 A map from a sphere to a higher dimensional sphere is nullhomotopic

```
lemma spheremap_lemma1:  
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'a::euclidean\_space$   
  assumes subspace  $S$  subspace  $T$  and dimST:  $dim\ S < dim\ T$   
    and  $S \subseteq T$   
    and diff_f:  $f$  differentiable_on sphere  $0$   $1 \cap S$   
  shows  $f' (sphere\ 0\ 1 \cap S) \neq sphere\ 0\ 1 \cap T$   
proof  
  assume fm:  $f' (sphere\ 0\ 1 \cap S) = sphere\ 0\ 1 \cap T$   
  have inS:  $\bigwedge x. \llbracket x \in S; x \neq 0 \rrbracket \Longrightarrow (x /_R norm\ x) \in S$   
    using subspace_mul  $\langle$ subspace  $S\rangle$  by blast  
  have subS01:  $(\lambda x. x /_R norm\ x)' (S - \{0\}) \subseteq sphere\ 0\ 1 \cap S$   
    using  $\langle$ subspace  $S\rangle$  subspace_mul by fastforce  
  then have diff_f':  $f$  differentiable_on  $(\lambda x. x /_R norm\ x)' (S - \{0\})$   
    by (rule differentiable_on_subset [OF diff_f])  
  define  $g$  where  $g \equiv \lambda x. norm\ x *_R f(inverse(norm\ x) *_R x)$   
  have gdifff:  $g$  differentiable_on  $S - \{0\}$   
    unfolding g_def  
    by (rule diff_f' derivative_intros differentiable_on_compose [where  $f=f$ ] |  
force)  
  have geq:  $g' (S - \{0\}) = T - \{0\}$   
  proof  
    have  $\bigwedge u. \llbracket u \in S; norm\ u *_R f(u /_R norm\ u) \notin T \rrbracket \Longrightarrow u = 0$ 
```

```

    by (metis (mono_tags, lifting) DiffI subS01 subspace_mul [OF ‹subspace T›]
    fim image_subset_iff inf_le2 singletonD)
  then have  $g \in (S - \{0\}) \rightarrow T$ 
    using  $g\_def$  by blast
  moreover have  $g \in (S - \{0\}) \rightarrow UNIV - \{0\}$ 
  proof (clarsimp simp:  $g\_def$ )
    fix  $y$ 
    assume  $y \in S$  and  $f0: f (y /_R norm y) = 0$ 
    then have  $y \neq 0 \implies y /_R norm y \in sphere\ 0\ 1 \cap S$ 
      by (auto simp: subspace_mul [OF ‹subspace S›])
    then show  $y = 0$ 
    by (metis fim f0 Int_iff image_iff mem_sphere_0 norm_eq_zero zero_neq_one)
  qed
  ultimately show  $g \text{ ' } (S - \{0\}) \subseteq T - \{0\}$ 
    by auto
next
have *:  $sphere\ 0\ 1 \cap T \subseteq f \text{ ' } (sphere\ 0\ 1 \cap S)$ 
  using fim by (simp add: image_subset_iff)
have  $x \in (\lambda x. norm\ x *_R f (x /_R norm\ x)) \text{ ' } (S - \{0\})$ 
  if  $x \in T$   $x \neq 0$  for  $x$ 
proof -
  have  $x /_R norm\ x \in T$ 
    using ‹subspace T› subspace_mul that by blast
  then obtain  $u$  where  $u: f\ u \in T$   $x /_R norm\ x = f\ u$   $norm\ u = 1$   $u \in S$ 
    using * [THEN subsetD, of  $x /_R norm\ x$ ] ‹ $x \neq 0$ › by auto
  with that have [simp]:  $norm\ x *_R f\ u = x$ 
    by (metis divideR_right norm_eq_zero)
  moreover have  $norm\ x *_R u \in S - \{0\}$ 
    using ‹subspace S› subspace_scale that(2)  $u$  by auto
  with  $u$  show ?thesis
    by (simp add: image_eqI [where  $x=norm\ x *_R\ u$ ])
  qed
then have  $T - \{0\} \subseteq (\lambda x. norm\ x *_R f (x /_R norm\ x)) \text{ ' } (S - \{0\})$ 
  by force
then show  $T - \{0\} \subseteq g \text{ ' } (S - \{0\})$ 
  by (simp add:  $g\_def$ )
qed
define  $T'$  where  $T' \equiv \{y. \forall x \in T. orthogonal\ x\ y\}$ 
have subspace  $T'$ 
  by (simp add: subspace_orthogonal_to_vectors  $T'_def$ )
have  $dim\_eq: dim\ T' + dim\ T = DIM('a)$ 
  using  $dim\_subspace\_orthogonal\_to\_vectors$  [of  $T\ UNIV$ ] ‹subspace T›
  by (simp add:  $T'_def$ )
have  $\exists v1\ v2. v1 \in span\ T \wedge (\forall w \in span\ T. orthogonal\ v2\ w) \wedge x = v1 + v2$ 
for  $x$ 
  by (force intro: orthogonal_subspace_decomp_exists [of  $T\ x$ ])
then obtain  $p1\ p2$  where  $p1span: p1\ x \in span\ T$ 
  and  $\bigwedge w. w \in span\ T \implies orthogonal\ (p2\ x)\ w$ 
  and  $eq: p1\ x + p2\ x = x$  for  $x$ 

```

```

    by metis
  then have p1:  $\bigwedge z. p1\ z \in T$  and ortho:  $\bigwedge w. w \in T \implies \text{orthogonal } (p2\ x)\ w$ 
  for x
    using span_eq_iff ‹subspace T› by blast+
  then have p2:  $\bigwedge z. p2\ z \in T'$ 
    by (simp add: T'_def orthogonal_commute)
  have p12_eq:  $\bigwedge x\ y. \llbracket x \in T; y \in T' \rrbracket \implies p1(x + y) = x \wedge p2(x + y) = y$ 
  proof (rule orthogonal_subspace_decomp_unique [OF eq_p1span, where T=T'])
    show  $\bigwedge x\ y. \llbracket x \in T; y \in T' \rrbracket \implies p2(x + y) \in \text{span } T'$ 
      using span_eq_iff p2 ‹subspace T'› by blast
    show  $\bigwedge a\ b. \llbracket a \in T; b \in T' \rrbracket \implies \text{orthogonal } a\ b$ 
      using T'_def by blast
  qed (auto simp: span_base)
  then have  $\bigwedge c\ x. p1(c *R x) = c *R p1\ x \wedge p2(c *R x) = c *R p2\ x$ 
  proof -
    fix c :: real and x :: 'a
    have f1:  $c *R x = c *R p1\ x + c *R p2\ x$ 
      by (metis eq_pth_6)
    have f2:  $c *R p2\ x \in T'$ 
      by (simp add: ‹subspace T'› p2_subspace_scale)
    have  $c *R p1\ x \in T$ 
      by (metis (full_types) assms(2) p1span span_eq_iff subspace_scale)
    then show  $p1(c *R x) = c *R p1\ x \wedge p2(c *R x) = c *R p2\ x$ 
      using f2 f1 p12_eq by presburger
  qed
  moreover have lin_add:  $\bigwedge x\ y. p1(x + y) = p1\ x + p1\ y \wedge p2(x + y) = p2\ x + p2\ y$ 
  proof (rule orthogonal_subspace_decomp_unique [OF _ p1span, where T=T'])
    show  $\bigwedge x\ y. p1(x + y) + p2(x + y) = p1\ x + p1\ y + (p2\ x + p2\ y)$ 
      by (simp add: add.assoc add.left_commute eq)
    show  $\bigwedge a\ b. \llbracket a \in T; b \in T' \rrbracket \implies \text{orthogonal } a\ b$ 
      using T'_def by blast
  qed (auto simp: p1span p2 span_base span_add)
  ultimately have linear_p1 linear_p2
    by unfold_locales auto
  have g differentiable_on p1 ‹ $\{x + y \mid x\ y. x \in S - \{0\} \wedge y \in T'\}$ ›
    using p12_eq ‹ $S \subseteq T$ › by (force intro: differentiable_on_subset [OF gdiff])
  then have  $(\lambda z. g(p1\ z))$  differentiable_on  $\{x + y \mid x\ y. x \in S - \{0\} \wedge y \in T'\}$ 
    by (rule differentiable_on_compose [OF linear_imp_differentiable_on [OF ‹linear p1›]])
  then have diff:  $(\lambda x. g(p1\ x) + p2\ x)$  differentiable_on  $\{x + y \mid x\ y. x \in S - \{0\} \wedge y \in T'\}$ 
    by (intro derivative_intros linear_imp_differentiable_on [OF ‹linear p2›])
  have  $\dim \{x + y \mid x\ y. x \in S - \{0\} \wedge y \in T'\} \leq \dim \{x + y \mid x\ y. x \in S \wedge y \in T'\}$ 
    by (blast intro: dim_subset)
  also have ... =  $\dim S + \dim T' - \dim (S \cap T')$ 
    using dim_sums_Int [OF ‹subspace S› ‹subspace T'›]
    by (simp add: algebra_simps)

```



```

also have ... < DIM('a)
  using dimST dim_eq by auto
finally have neg: negligible {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}
  by (rule negligible_lowdim)
have negligible ((λx. g (p1 x) + p2 x) ' {x + y | x y. x ∈ S - {0} ∧ y ∈ T'})
  by (rule negligible_differentiable_image_negligible [OF order_refl neg diff])
then have negligible {x + y | x y. x ∈ g ' (S - {0}) ∧ y ∈ T'}
proof (rule negligible_subset)
  have [[t' ∈ T'; s ∈ S; s ≠ 0]]
    ⇒ g s + t' ∈ (λx. g (p1 x) + p2 x) '
      {x + t' | x t'. x ∈ S ∧ x ≠ 0 ∧ t' ∈ T'} for t' s
  using ⟨S ⊆ T⟩ p12_eq by (rule_tac x=s + t' in image_eqI) auto
then show {x + y | x y. x ∈ g ' (S - {0}) ∧ y ∈ T'}
  ⊆ (λx. g (p1 x) + p2 x) ' {x + y | x y. x ∈ S - {0} ∧ y ∈ T'}
  by auto
qed
moreover have - T' ⊆ {x + y | x y. x ∈ g ' (S - {0}) ∧ y ∈ T'}
proof clarsimp
  fix z assume z ∉ T'
  show ∃x y. z = x + y ∧ x ∈ g ' (S - {0}) ∧ y ∈ T'
  by (metis Diff_iff ⟨z ∉ T'⟩ add.left_neutral eq geq p1 p2 singletonD)
qed
ultimately have negligible (-T')
  using negligible_subset by blast
moreover have negligible T'
  using negligible_lowdim
  by (metis add commute assms(3) diff_add_inverse2 diff_self_eq_0 dim_eq
le_add1 le_antisym linordered_semidom_class.add_diff_inverse not_less0)
ultimately have negligible (-T' ∪ T')
  by (metis negligible_Un_eq)
then show False
  using negligible_Un_eq non_negligible_UNIV by simp
qed

```

lemma *spheremap_lemma2*:

```

fixes f :: 'a::euclidean_space ⇒ 'a::euclidean_space
assumes ST: subspace S subspace T dim S < dim T
  and S ⊆ T
  and contf: continuous_on (sphere 0 1 ∩ S) f
  and fim: f ∈ (sphere 0 1 ∩ S) → sphere 0 1 ∩ T
shows ∃c. homotopic_with_canon (λx. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩
T) f (λx. c)
proof -
  have [simp]: ∧x. [[norm x = 1; x ∈ S]] ⇒ norm (f x) = 1
  using fim by auto
  have compact (sphere 0 1 ∩ S)
  by (simp add: ⟨subspace S⟩ closed_subspace compact_Int_closed)
  then obtain g where pfg: polynomial_function g and gim: g ∈ (sphere 0 1 ∩

```

```

S) → T
  and g12:  $\bigwedge x. x \in \text{sphere } 0 \ 1 \cap S \implies \text{norm}(f \ x - g \ x) < 1/2$ 
  using Stone_Weierstrass_polynomial_function_subspace [OF _ contf _ ‹sub-
space T›, of 1/2] fim
  by (auto simp: image_subset_iff_funcset)
  have gnz:  $g \ x \neq 0$  if  $x \in \text{sphere } 0 \ 1 \cap S$  for  $x$ 
  using g12 that by fastforce
  have diffg:  $g$  differentiable_on sphere 0 1  $\cap S$ 
  by (metis pfg differentiable_on_polynomial_function)
  define h where  $h \equiv \lambda x. \text{inverse}(\text{norm}(g \ x)) *_{\mathbb{R}} g \ x$ 
  have h:  $x \in \text{sphere } 0 \ 1 \cap S \implies h \ x \in \text{sphere } 0 \ 1 \cap T$  for  $x$ 
  unfolding h_def using ‹subspace T› gim gnz subspace_mul by fastforce
  have diffh:  $h$  differentiable_on sphere 0 1  $\cap S$ 
  unfolding h_def using gnz
  by (fastforce intro: derivative_intros diffg differentiable_on_compose [OF diffg])
  have homfg: homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere 0 1  $\cap S$ ) (T - {0}) f g
  proof (rule homotopic_with_linear [OF contf])
    show continuous_on (sphere 0 1  $\cap S$ ) g
    using pfg by (simp add: differentiable_imp_continuous_on diffg)
  next
  have non0fg:  $0 \notin \text{closed\_segment}(f \ x)(g \ x)$  if  $\text{norm } x = 1$   $x \in S$  for  $x$ 
  proof -
    have  $f \ x \in \text{sphere } 0 \ 1$ 
    using fim that by (simp add: image_subset_iff)
    moreover have  $\text{norm}(f \ x - g \ x) < 1/2$ 
    using g12 that by auto
    ultimately show ?thesis
    by (auto simp: norm_minus_commute dest: segment_bound)
  qed
  show closed_segment (f x) (g x)  $\subseteq T - \{0\}$  if  $x \in \text{sphere } 0 \ 1 \cap S$  for  $x$ 
  proof -
    have convex T
    by (simp add: ‹subspace T› subspace_imp_convex)
    then have convex hull {f x, g x}  $\subseteq T$ 
    by (metis IntD2 PiE closed_segment_subset fim gim segment_convex_hull
that)
    then show ?thesis
    using that non0fg segment_convex_hull by fastforce
  qed
  obtain d where  $d: d \in (\text{sphere } 0 \ 1 \cap T) - h \ ` (\text{sphere } 0 \ 1 \cap S)$ 
  using h spheremap_lemma1 [OF ST ‹S  $\subseteq T$ › diffh] by force
  then have non0hd:  $0 \notin \text{closed\_segment}(h \ x)(-d)$  if  $\text{norm } x = 1$   $x \in S$  for  $x$ 
  using midpoint_between [of 0 h x -d] that h [of x]
  by (auto simp: between_mem_segment midpoint_def)
  have conth: continuous_on (sphere 0 1  $\cap S$ ) h
  using differentiable_imp_continuous_on diffh by blast
  have hom_hd: homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere 0 1  $\cap S$ ) (T - {0})
h ( $\lambda x. -d$ )

```

```

proof (rule homotopic_with_linear [OF conth continuous_on_const])
  fix x
  assume x: x ∈ sphere 0 1 ∩ S
  have convex hull {h x, - d} ⊆ T
  proof (rule hull_minimal)
    show {h x, - d} ⊆ T
    using h d x by (force simp: subspace_neg [OF ‹subspace T›])
  qed (simp add: subspace_imp_convex [OF ‹subspace T›])
  with x segment_convex_hull show closed_segment (h x) (- d) ⊆ T - {0}
  by (auto simp add: subset_Diff_insert non0hd)
qed
have conT0: continuous_on (T - {0}) (λy. inverse(norm y) *R y)
  by (intro continuous_intros) auto
have sub0T: (λy. y /R norm y) ∈ (T - {0}) → sphere 0 1 ∩ T
  by (fastforce simp: assms(2) subspace_mul)
obtain c where homhc: homotopic_with_canon (λz. True) (sphere 0 1 ∩ S)
(sphere 0 1 ∩ T) h (λx. c)
proof
  show homotopic_with_canon (λz. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) h
(λx. - d)
  using d
  by (force simp: h_def
intro: homotopic_with_eq homotopic_with_compose_continuous_left [OF
hom_hd conT0 sub0T])
qed
have homotopic_with_canon (λx. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) f h
by (force simp: h_def
intro: homotopic_with_eq homotopic_with_compose_continuous_left
[OF homfg conT0 sub0T])
then show ?thesis
by (metis homotopic_with_trans [OF _ homhc])
qed

```

lemma spheremap_lemma3:

```

assumes bounded S convex S subspace U and affSU: aff_dim S ≤ dim U
obtains T where subspace T T ⊆ U S ≠ {} ⇒ aff_dim T = aff_dim S
(rel_frontier S) homeomorphic (sphere 0 1 ∩ T)
proof (cases S = {})
  case True
  with ‹subspace U› subspace_0 show ?thesis
  by (rule_tac T = {0} in that) auto
  next
  case False
  then obtain a where a ∈ S
  by auto
  then have affS: aff_dim S = int (dim ((λx. -a+x) ‘ S))
  by (metis hull_inc aff_dim_eq_dim)
  with affSU have dim ((λx. -a+x) ‘ S) ≤ dim U

```

```

    by linarith
    with choose_subspace_of_subspace
    obtain T where subspace T T ⊆ span U and dimT: dim T = dim ((λx. -a+x)
    ' S) .
    show ?thesis
    proof (rule that [OF ⟨subspace T⟩])
      show T ⊆ U
        using span_eq_iff ⟨subspace U⟩ ⟨T ⊆ span U⟩ by blast
      show aff_dim T = aff_dim S
        using dimT ⟨subspace T⟩ affS aff_dim_subspace by fastforce
      show rel_frontier S homeomorphic sphere 0 1 ∩ T
      proof -
        have aff_dim (ball 0 1 ∩ T) = aff_dim (T)
        by (metis IntI interior_ball ⟨subspace T⟩ aff_dim_convex_Int_nonempty_interior
        centre_in_ball empty_iff inf_commute subspace_0 subspace_imp_convex_zero_less_one)
        then have affS_eq: aff_dim S = aff_dim (ball 0 1 ∩ T)
          using ⟨aff_dim T = aff_dim S⟩ by simp
        have rel_frontier S homeomorphic rel_frontier(ball 0 1 ∩ T)
          using homeomorphic_rel_frontiers_convex_bounded_sets [OF ⟨convex S⟩
        ⟨bounded S⟩]
        by (simp add: ⟨subspace T⟩ affS_eq assms bounded_Int convex_Int
        homeomorphic_rel_frontiers_convex_bounded_sets subspace_imp_convex)
        also have ... = frontier (ball 0 1) ∩ T
        proof (rule convex_affine_rel_frontier_Int [OF convex_ball])
          show affine T
            by (simp add: ⟨subspace T⟩ subspace_imp_affine)
          show interior (ball 0 1) ∩ T ≠ {}
            using ⟨subspace T⟩ subspace_0 by force
        qed
        also have ... = sphere 0 1 ∩ T
          by auto
        finally show ?thesis .
      qed
    qed
  qed

```

```

proposition inessential_spheremap_lowdim_gen:
  fixes f :: 'M::euclidean_space ⇒ 'a::euclidean_space
  assumes convex S bounded S convex T bounded T
    and affST: aff_dim S < aff_dim T
    and contf: continuous_on (rel_frontier S) f
    and fim: f ∈ (rel_frontier S) → rel_frontier T
  obtains c where homotopic_with_canon (λz. True) (rel_frontier S) (rel_frontier
  T) f (λx. c)
proof (cases S = {})
  case True
  then show ?thesis
    using homotopic_with_canon_on_empty that by auto

```

```

next
  case False
  then show ?thesis
  proof (cases T = {})
    case True
    then show ?thesis
      by (smt (verit, best) False affST aff_dim_negative_iff)
  next
  case False
  obtain T':: 'a set
    where subspace T' and affT': aff_dim T' = aff_dim T
      and homT: rel_frontier T homeomorphic sphere 0 1  $\cap$  T'
      using  $\langle T \neq \{\} \rangle$  spheremap_lemma3 [OF  $\langle$ bounded T $\rangle$   $\langle$ convex T $\rangle$  subspace_UNIV, where 'b='a]
      by (force simp add: aff_dim_le_DIM)
  with homeomorphic_imp_homotopy_eqv
  have relT: sphere 0 1  $\cap$  T' homotopy_eqv rel_frontier T
    using homotopy_equivalent_space_sym by blast
  have aff_dim S  $\leq$  int (dim T')
    using affT'  $\langle$ subspace T' $\rangle$  affST aff_dim_subspace by force
  with spheremap_lemma3 [OF  $\langle$ bounded S $\rangle$   $\langle$ convex S $\rangle$   $\langle$ subspace T' $\rangle$ ]  $\langle$ S  $\neq$   $\{\}$  $\rangle$ 
  obtain S':: 'a set where subspace S' S'  $\subseteq$  T'
    and affS': aff_dim S' = aff_dim S
    and homT: rel_frontier S homeomorphic sphere 0 1  $\cap$  S'
      by metis
  with homeomorphic_imp_homotopy_eqv
  have relS: sphere 0 1  $\cap$  S' homotopy_eqv rel_frontier S
    using homotopy_equivalent_space_sym by blast
  have dimST': dim S' < dim T'
    by (metis  $\langle$ S'  $\subseteq$  T' $\rangle$   $\langle$ subspace S' $\rangle$   $\langle$ subspace T' $\rangle$  affS' affST affT' less_irrefl not_le subspace_dim_equal)
  have  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{rel\_frontier } S) (\text{rel\_frontier } T) f (\lambda x. c)$ 
    using spheremap_lemma2 homotopy_eqv_cohomotopic_triviality_null[OF relS]
    using homotopy_eqv_homotopic_triviality_null_imp [OF relT contf fim]
    by (metis  $\langle$ S'  $\subseteq$  T' $\rangle$   $\langle$ subspace S' $\rangle$   $\langle$ subspace T' $\rangle$  dimST' image_subset_iff_funcset)
  with that show ?thesis by blast
qed
qed

lemma inessential_spheremap_lowdim:
  fixes f :: 'M::euclidean_space  $\Rightarrow$  'a::euclidean_space
  assumes
    DIM('M) < DIM('a) and f: continuous_on (sphere a r) f f  $\in$  (sphere a r)  $\rightarrow$  (sphere b s)
  obtains c where homotopic_with_canon ( $\lambda x. \text{True}$ ) (sphere a r) (sphere b s) f ( $\lambda x. c$ )
  proof (cases s  $\leq$  0)

```

```

case True then show ?thesis
  by (meson nullhomotopic_into_contractible f contractible_sphere that)
next
case False
show ?thesis
proof (cases r ≤ 0)
  case True then show ?thesis
    by (meson f nullhomotopic_from_contractible contractible_sphere that)
  next
  case False
  with ⟨ $\neg s \leq 0$ ⟩ have  $r > 0 \ s > 0$  by auto
  show thesis
    using inessential_spheremap_lowdim_gen [of cball a r cball b s f]
    using ⟨ $0 < r$ ⟩ ⟨ $0 < s$ ⟩ assms(1) that by (auto simp add: f aff_dim_cball)
qed
qed

```

9.27.2 Some technical lemmas about extending maps from cell complexes

```

lemma extending_maps_Union_aux:
  assumes fin: finite F
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
    and  $\bigwedge S T. \llbracket S \in \mathcal{F}; T \in \mathcal{F}; S \neq T \rrbracket \implies S \cap T \subseteq K$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \exists g. \text{continuous\_on } S \ g \wedge g \in S \rightarrow T \wedge (\forall x \in S \cap K. g \ x = h \ x)$ 
  shows  $\exists g. \text{continuous\_on } (\bigcup \mathcal{F}) \ g \wedge g \in (\bigcup \mathcal{F}) \rightarrow T \wedge (\forall x \in \bigcup \mathcal{F} \cap K. g \ x = h \ x)$ 
  using assms
  proof (induction F)
    case empty show ?case by simp
  next
    case (insert S F)
    then obtain f where contf: continuous_on S f and fm: f ∈ S → T and feq: ∀ x ∈ S ∩ K. f x = h x
      by (metis funcset_image insert_iff)
    obtain g where contg: continuous_on (∪ F) g and gm: g ∈ (∪ F) → T and
      geq: ∀ x ∈ ∪ F ∩ K. g x = h x
      using insert by auto
    have fg: f x = g x if x ∈ T T ∈ F x ∈ S for x T
    proof –
      have  $T \cap S \subseteq K \vee S = T$ 
      using that by (metis (no_types) insert.prem(2) insertCI)
    then show ?thesis
      using UnionI feq geq ⟨S ∉ F⟩ subsetD that by fastforce
    qed
  moreover have continuous_on (S ∪ ∪ F) (λx. if x ∈ S then f x else g x)
    by (auto simp: insert closed_Union contf contg intro: fg continuous_on_cases)
  moreover have  $S \cup \bigcup \mathcal{F} = \bigcup (\text{insert } S \ \mathcal{F})$ 

```

by auto
ultimately show ?case
by (smt (verit) Int_iff Pi_iff UnE feq fim geq gim)
qed

lemma extending_maps_Union:

assumes fin: finite \mathcal{F}
and $\bigwedge S. S \in \mathcal{F} \implies \exists g. \text{continuous_on } S \ g \wedge g \in S \rightarrow T \wedge (\forall x \in S \cap K. g \ x = h \ x)$
and $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$
and $K: \bigwedge X \ Y. [X \in \mathcal{F}; Y \in \mathcal{F}; \neg X \subseteq Y; \neg Y \subseteq X] \implies X \cap Y \subseteq K$
shows $\exists g. \text{continuous_on } (\bigcup \mathcal{F}) \ g \wedge g \in (\bigcup \mathcal{F}) \rightarrow T \wedge (\forall x \in \bigcup \mathcal{F} \cap K. g \ x = h \ x)$
proof –
have $\bigwedge S \ T. [S \in \mathcal{F}; \forall U \in \mathcal{F}. \neg S \subset U; T \in \mathcal{F}; \forall U \in \mathcal{F}. \neg T \subset U; S \neq T] \implies S \cap T \subseteq K$
by (metis K psubsetI)
then show ?thesis
apply (simp flip: Union_maximal_sets [OF fin])
apply (rule extending_maps_Union_aux, simp_all add: Union_maximal_sets [OF fin] assms)
done
qed

lemma extend_map_lemma:

assumes finite \mathcal{F} $\mathcal{G} \subseteq \mathcal{F}$ convex T bounded T
and poly: $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$
and aff: $\bigwedge X. X \in \mathcal{F} - \mathcal{G} \implies \text{aff_dim } X < \text{aff_dim } T$
and face: $\bigwedge S \ T. [S \in \mathcal{F}; T \in \mathcal{F}] \implies (S \cap T) \text{ face_of } S$
and contf: $\text{continuous_on } (\bigcup \mathcal{G}) \ f$ and $\text{fim: } f \in (\bigcup \mathcal{G}) \rightarrow \text{rel_frontier } T$
obtains g where $\text{continuous_on } (\bigcup \mathcal{F}) \ g \wedge g \in (\bigcup \mathcal{F}) \rightarrow \text{rel_frontier } T \wedge \lambda x. x \in \bigcup \mathcal{G} \implies g \ x = f \ x$
proof (cases $\mathcal{F} - \mathcal{G} = \{\}$)
case True
show ?thesis
using True assms(2) contf fim that by force
next
case False
then have $0 \leq \text{aff_dim } T$
by (metis aff aff_dim_empty aff_dim_geq aff_dim_negative_iff all_not_in_conv not_less)
then obtain $i::\text{nat}$ where $i: \text{int } i = \text{aff_dim } T$
by (metis nonneg_eq_int)
have Union_empty_eq: $\bigcup \{D. D = \{\} \wedge P \ D\} = \{\}$ for $P :: 'a \text{ set} \implies \text{bool}$
by auto
have face': $\bigwedge S \ T. [S \in \mathcal{F}; T \in \mathcal{F}] \implies (S \cap T) \text{ face_of } S \wedge (S \cap T) \text{ face_of } T$
by (metis face inf_commute)
have extendf: $\exists g. \text{continuous_on } (\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D \text{ face_of } C \wedge \text{aff_dim } D < i\})) \ g \wedge$

```

      g ' (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < i})) ⊆
rel_frontier T ∧
      (∀ x ∈ ⋃ G. g x = f x)
    if i ≤ aff_dim T for i::nat
  using that
  proof (induction i)
    case 0
    show ?case
      using 0 contf_fm by (auto simp add: Union_empty_eq)
    next
    case (Suc p)
    with ⟨bounded T⟩ have rel_frontier T ≠ {}
      by (auto simp: rel_frontier_eq_empty affine_bounded_eq_lowdim [of T])
    then obtain t where t: t ∈ rel_frontier T by auto
    have ple: int p ≤ aff_dim T using Suc.prem by force
    obtain h where conth: continuous_on (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧
aff_dim D < p})) h
      and him: h ' (⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}))
        ⊆ rel_frontier T
      and heq: ∧ x. x ∈ ⋃ G ⇒ h x = f x
    using Suc.IH [OF ple] by auto
    let ?Faces = {D. ∃ C ∈ F. D face_of C ∧ aff_dim D ≤ p}
    have extendh: ∃ g. continuous_on D g ∧
      g ∈ D → rel_frontier T ∧
      (∀ x ∈ D ∩ ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D <
p}). g x = h x)
    if D: D ∈ G ∪ ?Faces for D
  proof (cases D ⊆ ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}))
    case True
    have continuous_on D h
      using True conth continuous_on_subset by blast
    moreover have h ∈ D → rel_frontier T
      using True him by blast
    ultimately show ?thesis
      by blast
  next
  case False
  case False
  note notDsub = False
  show ?thesis
  proof (cases ∃ a. D = {a})
    case True
    then obtain a where D = {a} by auto
    with notDsub t show ?thesis
      by (rule_tac x=λx. t in exI) simp
  next
  case False
  have D ≠ {} using notDsub by auto
  have Dnotin: D ∉ G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}
    using notDsub by auto

```



```

then have  $D \notin \mathcal{G}$  by simp
have  $D \in ?Faces - \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < p\}$ 
  using Dnotin that by auto
then obtain  $C$  where  $C \in \mathcal{F}$   $D \text{ face\_of } C$  and affD:  $\text{aff\_dim } D = \text{int } p$ 
  by auto
then have bounded  $D$ 
  using face_of_polytope_polytope poly polytope_imp_bounded by blast
then have [simp]:  $\neg \text{affine } D$ 
  using affine_bounded_eq_trivial False  $\langle D \neq \{\} \rangle \langle \text{bounded } D \rangle$  by blast
have  $\{F. F \text{ facet\_of } D\} \subseteq \{E. E \text{ face\_of } C \wedge \text{aff\_dim } E < \text{int } p\}$ 
  by clarify (metis  $\langle D \text{ face\_of } C \rangle$  affD eq_iff_face_of_trans facet_of_def
zle_diff1_eq)
moreover have polyhedron  $D$ 
  using  $\langle C \in \mathcal{F} \rangle \langle D \text{ face\_of } C \rangle$  face_of_polytope_polytope poly poly-
tope_imp_polyhedron by auto
ultimately have rel_sub:  $\text{rel\_frontier } D \subseteq \bigcup \{E. E \text{ face\_of } C \wedge \text{aff\_dim}$ 
 $E < p\}$ 
  by (simp add: rel_frontier_of_polyhedron Union_mono)
then have him_rel:  $h \in \text{rel\_frontier } D \rightarrow \text{rel\_frontier } T$ 
  using  $\langle C \in \mathcal{F} \rangle$  him by blast
have convex  $D$ 
  by (simp add:  $\langle \text{polyhedron } D \rangle$  polyhedron_imp_convex)
have affD_lessT:  $\text{aff\_dim } D < \text{aff\_dim } T$ 
  using Suc.premis affD by linarith
have contDh: continuous_on (rel_frontier  $D$ )  $h$ 
  using  $\langle C \in \mathcal{F} \rangle$  rel_sub by (blast intro: continuous_on_subset [OF conth])
  then have *:  $(\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{rel\_frontier } D)$ 
 $(\text{rel\_frontier } T) h (\lambda x. c)) =$ 
 $(\exists g. \text{continuous\_on UNIV } g \wedge \text{range } g \subseteq \text{rel\_frontier } T \wedge$ 
 $(\forall x \in \text{rel\_frontier } D. g x = h x))$ 
  by (simp add: assms image_subset_iff_funcset rel_frontier_eq_empty
him_rel nullhomotopic_into_rel_frontier_extension [OF closed_rel_frontier])
  have  $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) (\text{rel\_frontier } D) (\text{rel\_frontier}$ 
 $T) h (\lambda x. c)$ 
  by (metis inessential_spheremap_lowdim_gen
[OF  $\langle \text{convex } D \rangle \langle \text{bounded } D \rangle \langle \text{convex } T \rangle \langle \text{bounded } T \rangle$  affD_lessT
contDh him_rel])
then obtain  $g$  where contg: continuous_on UNIV  $g$ 
  and gim:  $\text{range } g \subseteq \text{rel\_frontier } T$ 
  and gh:  $\bigwedge x. x \in \text{rel\_frontier } D \implies g x = h x$ 
  by (metis *)
have  $D \cap E \subseteq \text{rel\_frontier } D$ 
  if  $E \in \mathcal{G} \cup \{D. \exists \mathcal{F} ((\text{face\_of}) D) \wedge \text{aff\_dim } D < \text{int } p\}$  for  $E$ 
proof (rule face_of_subset_rel_frontier)
show  $D \cap E \text{ face\_of } D$ 
  using that
proof safe
assume  $E \in \mathcal{G}$ 
then show  $D \cap E \text{ face\_of } D$ 

```

```

    by (meson ⟨C ∈ F⟩ ⟨D face_of C⟩ assms(2) face' face_of_Int_subface
face_of_refl_eq poly polytope_imp_convex subsetD)
  next
    fix x
    assume aff_dim E < int p x ∈ F E face_of x
    then show D ∩ E face_of D
      by (meson ⟨C ∈ F⟩ ⟨D face_of C⟩ face' face_of_Int_subface that)
    qed
    show D ∩ E ≠ D
      using that notDsub by auto
    qed
    moreover have continuous_on D g
      using contg continuous_on_subset by blast
    ultimately show ?thesis
      by (rule_tac x=g in exI) (use gh gim in fastforce)
    qed
  qed
  have intle: i < 1 + int j ⟷ i ≤ int j for i j
    by auto
  have finite G
    using ⟨finite F⟩ ⟨G ⊆ F⟩ rev_finite_subset by blast
  moreover have finite (?Faces)
  proof -
    have §: finite (⋃ {{D. D face_of C} | C. C ∈ F})
      by (auto simp: ⟨finite F⟩ finite_polytope_faces poly)
    show ?thesis
      by (auto intro: finite_subset [OF _ §])
    qed
  ultimately have fin: finite (G ∪ ?Faces)
    by simp
  have clo: closed S if S ∈ G ∪ ?Faces for S
    using that ⟨G ⊆ F⟩ face_of_polytope_polytope poly polytope_imp_closed by
blast
  have K: X ∩ Y ⊆ ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < int p})
    if X ∈ G ∪ ?Faces Y ∈ G ∪ ?Faces ¬ Y ⊆ X for X Y
  proof -
    have ff: X ∩ Y face_of X ∧ X ∩ Y face_of Y
      if XY: X face_of D Y face_of E and DE: D ∈ F E ∈ F for D E
      by (rule face_of_Int_subface [OF _ _ XY]) (auto simp: face' DE)
    show ?thesis
      using that
      apply clarsimp
      by (smt (verit) IntI face_of_aff_dim_lt face_of_imp_convex face_of_trans
ff inf_commute)
    qed
  obtain g where continuous_on (⋃ (G ∪ ?Faces)) g
    g ' ⋃ (G ∪ ?Faces) ⊆ rel_frontier T
    (∀ x ∈ ⋃ (G ∪ ?Faces) ∩
      ⋃ (G ∪ {D. ∃ C ∈ F. D face_of C ∧ aff_dim D < p}). g x =

```

```

h x)
  by (rule exE [OF extending_maps_Union [OF fin_extendh clo K]], blast+)
  then show ?case
    by (simp add: intle local.heq [symmetric], blast)
qed
have eq:  $\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < i\}) = \bigcup \mathcal{F}$ 
proof
  show  $\bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < \text{int } i\}) \subseteq \bigcup \mathcal{F}$ 
    using  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  face_of_imp_subset by fastforce
  show  $\bigcup \mathcal{F} \subseteq \bigcup (\mathcal{G} \cup \{D. \exists C \in \mathcal{F}. D \text{ face\_of } C \wedge \text{aff\_dim } D < i\})$ 
    using face by (intro Union_mono) (fastforce simp: aff i)
qed
have int i  $\leq$  aff_dim T by (simp add: i)
then show ?thesis
  using extendf [of i] that unfolding eq by fastforce
qed

lemma extend_map_lemma_cofinite0:
  assumes finite  $\mathcal{F}$ 
    and pairwise  $(\lambda S T. S \cap T \subseteq K)$   $\mathcal{F}$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \exists a g. a \notin U \wedge \text{continuous\_on } (S - \{a\}) g \wedge g \in (S - \{a\}) \rightarrow T \wedge (\forall x \in S \cap K. g x = h x)$ 
    and  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
  shows  $\exists C g. \text{finite } C \wedge \text{disjnt } C U \wedge \text{card } C \leq \text{card } \mathcal{F} \wedge$ 
     $\text{continuous\_on } (\bigcup \mathcal{F} - C) g \wedge g \in (\bigcup \mathcal{F} - C) \rightarrow T$ 
     $\wedge (\forall x \in (\bigcup \mathcal{F} - C) \cap K. g x = h x)$ 
  using assms
proof induction
  case empty then show ?case
    by force
next
  case (insert X  $\mathcal{F}$ )
  then have closed X and clo:  $\bigwedge X. X \in \mathcal{F} \implies \text{closed } X$ 
    and  $\mathcal{F}: \bigwedge S. S \in \mathcal{F} \implies \exists a g. a \notin U \wedge \text{continuous\_on } (S - \{a\}) g \wedge g \in (S - \{a\}) \rightarrow T \wedge (\forall x \in S \cap K. g x = h x)$ 
    and pwX:  $\bigwedge Y. Y \in \mathcal{F} \wedge Y \neq X \implies X \cap Y \subseteq K \wedge Y \cap X \subseteq K$ 
    and pwF: pairwise  $(\lambda S T. S \cap T \subseteq K)$   $\mathcal{F}$ 
  by (simp_all add: pairwise_insert)
  obtain C g where C: finite C disjnt C U card C  $\leq$  card  $\mathcal{F}$ 
    and contg: continuous_on  $(\bigcup \mathcal{F} - C) g$ 
    and gim:  $g \in (\bigcup \mathcal{F} - C) \rightarrow T$ 
    and gh:  $\bigwedge x. x \in (\bigcup \mathcal{F} - C) \cap K \implies g x = h x$ 
  using insert.IH [OF pwF  $\mathcal{F}$  clo] by auto
  obtain a f where a  $\notin$  U
    and contf: continuous_on  $(X - \{a\}) f$ 
    and fim:  $f \in (X - \{a\}) \rightarrow T$ 
    and fh:  $(\forall x \in X \cap K. f x = h x)$ 
  using insert.premis by (meson insertI1)
  show ?case

```

```

proof (intro exI conjI)
  show finite (insert a C)
    by (simp add: C)
  show disjnt (insert a C) U
    using C ⟨a ∉ U⟩ by simp
  show card (insert a C) ≤ card (insert X F)
    by (simp add: C card_insert_if insert.hyps le_SucI)
  have closed (⋃F)
    using clo insert.hyps by blast
  have continuous_on (X - insert a C) f
    using contf by (force simp: elim: continuous_on_subset)
  moreover have continuous_on (⋃ F - insert a C) g
    using contg by (force simp: elim: continuous_on_subset)
  ultimately
  have continuous_on (X - insert a C ∪ (⋃F - insert a C)) (λx. if x ∈ X
then f x else g x)
    apply (intro continuous_on_cases_local; simp add: closedin_closed)
    using ⟨closed X⟩ apply blast
    using ⟨closed (⋃F)⟩ apply blast
    using fh gh insert.hyps pwX by fastforce
  then show continuous_on (⋃(insert X F) - insert a C) (λa. if a ∈ X then f
a else g a)
    by (blast intro: continuous_on_subset)
  show ∀ x ∈ (⋃(insert X F) - insert a C) ∩ K. (if x ∈ X then f x else g x) = h
x
    using gh by (auto simp: fh)
  show (λa. if a ∈ X then f a else g a) ∈ (⋃(insert X F) - insert a C) → T
    using fim gim Pi_iff by fastforce
qed
qed

```

lemma extend_map_lemma_cofinite1:

```

assumes finite F
  and F: ∧X. X ∈ F ⇒ ∃ a g. a ∉ U ∧ continuous_on (X - {a}) g ∧ g ∈ (X
- {a}) → T ∧ (∀ x ∈ X ∩ K. g x = h x)
  and clo: ∧X. X ∈ F ⇒ closed X
  and K: ∧X Y. [X ∈ F; Y ∈ F; ¬ X ⊆ Y; ¬ Y ⊆ X] ⇒ X ∩ Y ⊆ K
obtains C g where finite C disjnt C U card C ≤ card F continuous_on (⋃F
- C) g
  g ∈ (⋃F - C) → T
  ∧x. x ∈ (⋃F - C) ∩ K ⇒ g x = h x

```

proof -

```

let ?F = {X ∈ F. ∀ Y ∈ F. ¬ X ⊆ Y}
have [simp]: ⋃ ?F = ⋃ F
  by (simp add: Union_maximal_sets assms)
have fin: finite ?F
  by (force intro: finite_subset [OF _ ⟨finite F⟩])
have pw: pairwise (λ S T. S ∩ T ⊆ K) ?F

```

```

  by (simp add: pairwise_def) (metis K psubsetI)
  have card {X ∈ F. ∀ Y ∈ F. ¬ X ⊂ Y} ≤ card F
  by (simp add: ⟨finite F⟩ card_mono)
  moreover
  obtain C g where finite C ∧ disjnt C U ∧ card C ≤ card ?F ∧
    continuous_on (∪ ?F - C) g ∧ g ∈ (∪ ?F - C) → T
    ∧ (∀ x ∈ (∪ ?F - C) ∩ K. g x = h x)
  using extend_map_lemma_cofinite0 [OF fin pw, of U T h] by (fastforce intro!:
clo F)
  ultimately show ?thesis
  by (rule_tac C=C and g=g in that) auto
qed

```

lemma extend_map_lemma_cofinite:

```

  assumes finite F G ⊆ F and T: convex T bounded T
  and poly: ∧X. X ∈ F ⇒ polytope X
  and conf: continuous_on (∪ G) f and fim: f ∈ (∪ G) → rel_frontier T
  and face: ∧X Y. [X ∈ F; Y ∈ F] ⇒ (X ∩ Y) face_of X
  and aff: ∧X. X ∈ F - G ⇒ aff_dim X ≤ aff_dim T
  obtains C g where
    finite C disjnt C (∪ G) card C ≤ card F continuous_on (∪ F - C) g
    g ∈ (∪ F - C) → rel_frontier T ∧ x. x ∈ ∪ G ⇒ g x = f x
  proof -
    define H where H ≡ G ∪ {D. ∃ C ∈ F - G. D face_of C ∧ aff_dim D <
aff_dim T}
    have finite G
    using assms finite_subset by blast
    have *: finite (∪ {D. D face_of C} | C. C ∈ F)
    using finite_polytope_faces poly ⟨finite F⟩ by force
    then have finite H
    by (auto simp: H_def ⟨finite G⟩ intro: finite_subset [OF _ *])
    have face': ∧S T. [S ∈ F; T ∈ F] ⇒ (S ∩ T) face_of S ∧ (S ∩ T) face_of T
    by (metis face inf_commute)
    have *: ∧X Y. [X ∈ H; Y ∈ H] ⇒ X ∩ Y face_of X
    by (simp add: H_def) (smt (verit) ⟨G ⊆ F⟩ DiffE face' face_of_Int_subface
in_mono inf.idem)
    obtain h where conth: continuous_on (∪ H) h and him: h ∈ (∪ H) → rel_frontier
T
    and hf: ∧x. x ∈ ∪ G ⇒ h x = f x
  proof (rule extend_map_lemma [OF ⟨finite H⟩ [unfolded H_def] Un_upper1
T])
    show ∧X. [X ∈ G ∪ {D. ∃ C ∈ F - G. D face_of C ∧ aff_dim D < aff_dim
T}] ⇒ polytope X
    using ⟨G ⊆ F⟩ face_of_polytope_polytope poly by fastforce
  qed (use * H_def conf fim in auto)
  have bounded (∪ G)
  using ⟨finite G⟩ ⟨G ⊆ F⟩ poly polytope_imp_bounded by blast
  then have ∪ G ≠ UNIV

```

```

    by auto
  then obtain a where a: a  $\notin \bigcup \mathcal{G}$ 
    by blast
  have  $\mathcal{F}$ :  $\exists a g. a \notin \bigcup \mathcal{G} \wedge \text{continuous\_on } (D - \{a\}) g \wedge$ 
       $g \in (D - \{a\}) \rightarrow \text{rel\_frontier } T \wedge (\forall x \in D \cap \bigcup \mathcal{H}. g x = h x)$ 
    if  $D \in \mathcal{F}$  for  $D$ 
  proof (cases  $D \subseteq \bigcup \mathcal{H}$ )
  case True
  then have  $h \in (D - \{a\}) \rightarrow \text{rel\_frontier } T \text{ continuous\_on } (D - \{a\}) h$ 
    using him by (blast intro!:  $\langle a \notin \bigcup \mathcal{G} \rangle \text{continuous\_on\_subset } [OF \text{ conth}])+$ 
  then show ?thesis
    using a by blast
  next
  case False
  note  $D\_not\_subset = False$ 
  show ?thesis
  proof (cases  $D \in \mathcal{G}$ )
  case True
  with  $D\_not\_subset$  show ?thesis
    by (auto simp:  $\mathcal{H\_def}$ )
  next
  case False
  then have  $\text{aff}D$ :  $\text{aff\_dim } D \leq \text{aff\_dim } T$ 
    by (simp add:  $\langle D \in \mathcal{F} \rangle \text{aff}$ )
  show ?thesis
  proof (cases  $\text{rel\_interior } D = \{\}$ )
  case True
  with  $\langle D \in \mathcal{F} \rangle \text{poly } a$  show ?thesis
    by (force simp:  $\text{rel\_interior\_eq\_empty polytope\_imp\_convex}$ )
  next
  case False
  then obtain b where  $\text{brel}D$ :  $b \in \text{rel\_interior } D$ 
    by blast
  have  $\text{polyhedron } D$ 
    by (simp add:  $\text{poly polytope\_imp\_polyhedron that}$ )
  have  $\text{rel\_frontier } D \text{ retract\_of affine hull } D - \{b\}$ 
    by (simp add:  $\text{rel\_frontier\_retract\_of\_punctured\_affine\_hull poly polytope\_imp\_bounded polytope\_imp\_convex that brel}D$ )
  then obtain r where  $\text{rel}D$ :  $\text{rel\_frontier } D \subseteq \text{affine hull } D - \{b\}$ 
    and  $\text{contr}$ :  $\text{continuous\_on } (\text{affine hull } D - \{b\}) r$ 
    and  $\text{rim}$ :  $r \in (\text{affine hull } D - \{b\}) \rightarrow \text{rel\_frontier } D$ 
    and  $\text{rid}$ :  $\bigwedge x. x \in \text{rel\_frontier } D \implies r x = x$ 
    by (auto simp:  $\text{retract\_of\_def retraction\_def}$ )
  show ?thesis
  proof (intro  $\text{exI conjI ballI}$ )
  show  $b \notin \bigcup \mathcal{G}$ 
  proof clarify
  fix  $E$ 
  assume  $b \in E E \in \mathcal{G}$ 

```

```

    then have  $E \cap D$  face_of  $E \wedge E \cap D$  face_of  $D$ 
      using  $\langle \mathcal{G} \subseteq \mathcal{F} \rangle$  face' that by auto
  with face_of_subset_rel_frontier  $\langle E \in \mathcal{G} \rangle \langle b \in E \rangle$  brelD rel_interior_subset
[of  $D$ ]
    D_not_subset rel_frontier_def  $\mathcal{H}$ _def
  show False
    by blast
  qed
  have  $r^{-1}(D - \{b\}) \subseteq r^{-1}(\text{affine hull } D - \{b\})$ 
    by (simp add: Diff_mono hull_subset image_mono)
  also have  $\dots \subseteq \text{rel\_frontier } D$ 
    using rim by auto
  also have  $\dots \subseteq \bigcup \{E. E \text{ face\_of } D \wedge \text{aff\_dim } E < \text{aff\_dim } T\}$ 
    using affD
    by (force simp: rel_frontier_of_polyhedron [OF  $\langle \text{polyhedron } D \rangle$ ]
facet_of_def)
  also have  $\dots \subseteq \bigcup (\mathcal{H})$ 
    using D_not_subset  $\mathcal{H}$ _def that by fastforce
  finally have rsub:  $r^{-1}(D - \{b\}) \subseteq \bigcup (\mathcal{H})$  .
  show continuous_on  $(D - \{b\})$   $(h \circ r)$ 
  proof (rule continuous_on_compose)
    show continuous_on  $(D - \{b\})$   $r$ 
      by (meson Diff_mono continuous_on_subset contr hull_subset order_refl)
    show continuous_on  $(r^{-1}(D - \{b\}))$   $h$ 
      by (simp add: Diff_mono hull_subset continuous_on_subset [OF conth
rsub])
  qed
  show  $(h \circ r) \in (D - \{b\}) \rightarrow \text{rel\_frontier } T$ 
    using brelD him rsub by fastforce
  show  $(h \circ r) x = h x$  if  $x: x \in D \cap \bigcup \mathcal{H}$  for  $x$ 
  proof -
    consider  $A$  where  $x \in D \ A \in \mathcal{G} \ x \in A$ 
      |  $A \ B$  where  $x \in D \ A \text{ face\_of } B \ B \in \mathcal{F} \ B \notin \mathcal{G} \ \text{aff\_dim } A < \text{aff\_dim}$ 
 $T \ x \in A$ 
    using  $x$  by (auto simp:  $\mathcal{H}$ _def)
    then have xrel:  $x \in \text{rel\_frontier } D$ 
  proof cases
    case 1 show ?thesis
  proof (rule face_of_subset_rel_frontier [THEN subsetD])
    show  $D \cap A$  face_of  $D$ 
      using  $\langle A \in \mathcal{G} \rangle \langle \mathcal{G} \subseteq \mathcal{F} \rangle$  face  $\langle D \in \mathcal{F} \rangle$  by blast
    show  $D \cap A \neq D$ 
      using  $\langle A \in \mathcal{G} \rangle$  D_not_subset  $\mathcal{H}$ _def by blast
  qed (auto simp: 1)
  next
    case 2 show ?thesis
  proof (rule face_of_subset_rel_frontier [THEN subsetD])
    have  $D$  face_of  $D$ 

```

```

    by (simp add: ⟨polyhedron D⟩ polyhedron_imp_convex face_of_refl)
  then show  $D \cap A$  face_of D
    by (meson 2(2) 2(3) ⟨D ∈  $\mathcal{F}$ ⟩ face' face_of_Int_Int face_of_face)
  show  $D \cap A \neq D$ 
    using 2 D_not_subset  $\mathcal{H}$ _def by blast
  qed (auto simp: 2)
  qed
  show ?thesis
    by (simp add: rid xrel)
  qed
  qed
  qed
  have clo:  $\bigwedge S. S \in \mathcal{F} \implies \text{closed } S$ 
    by (simp add: poly polytope_imp_closed)
  obtain C g where finite C disjnt C  $(\bigcup \mathcal{G})$  card C  $\leq$  card  $\mathcal{F}$  continuous_on  $(\bigcup \mathcal{F} - C)$  g
    g ∈  $(\bigcup \mathcal{F} - C) \rightarrow \text{rel\_frontier } T$ 
    and gh:  $\bigwedge x. x \in (\bigcup \mathcal{F} - C) \cap \bigcup \mathcal{H} \implies g x = h x$ 
  proof (rule extend_map_lemma_cofinite1 [OF ⟨finite  $\mathcal{F}$ ⟩  $\mathcal{F}$  clo])
    show  $X \cap Y \subseteq \bigcup \mathcal{H}$  if XY:  $X \in \mathcal{F} Y \in \mathcal{F}$  and  $\neg X \subseteq Y \neg Y \subseteq X$  for  $X Y$ 
    proof (cases  $X \in \mathcal{G}$ )
      case True
        then show ?thesis
          by (auto simp:  $\mathcal{H}$ _def)
      next
        case False
          have  $X \cap Y \neq X$ 
            using ⟨ $\neg X \subseteq Y$ ⟩ by blast
          with XY
            show ?thesis
              by (clarsimp simp:  $\mathcal{H}$ _def)
              (metis Diff_iff Int_iff aff antisym_conv face face_of_aff_dim_lt
                face_of_refl not_le poly polytope_imp_convex)
    qed
  qed (blast)+
  with ⟨ $\mathcal{G} \subseteq \mathcal{F}$ ⟩ show ?thesis
    by (rule_tac C=C and g=g in that) (auto simp: disjnt_def hf [symmetric]
 $\mathcal{H}$ _def intro!: gh)
  qed

```

The next two proofs are similar

```

theorem extend_map_cell_complex_to_sphere:
  assumes finite  $\mathcal{F}$  and S:  $S \subseteq \bigcup \mathcal{F}$  closed S and T: convex T bounded T
    and poly:  $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$ 
    and aff:  $\bigwedge X. X \in \mathcal{F} \implies \text{aff\_dim } X < \text{aff\_dim } T$ 
    and face:  $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face\_of } X$ 

```



```

and contf: continuous_on S f and fim:  $f \in S \rightarrow \text{rel\_frontier } T$ 
obtains g where continuous_on  $(\bigcup \mathcal{F})$  g
 $g \in (\bigcup \mathcal{F}) \rightarrow \text{rel\_frontier } T \wedge x. x \in S \implies g \ x = f \ x$ 
proof -
  obtain V g where  $S \subseteq V$  open V continuous_on V g and gim:  $g \in V \rightarrow$ 
rel\_frontier T and gf:  $\wedge x. x \in S \implies g \ x = f \ x$ 
  using neighbourhood_extension_into_ANR [OF contf fim  $\langle \text{closed } S \rangle$ ] ANR_rel_frontier_convex
T by blast
  have compact S
  by (meson assms compact_Union poly polytope_imp_compact seq_compact_closed_subset
seq_compact_eq_compact)
  then obtain d where  $d > 0$  and  $d: \wedge x \ y. \llbracket x \in S; y \in - \ V \rrbracket \implies d \leq \text{dist } x \ y$ 
  using separate_compact_closed [of  $S - V$ ]  $\langle \text{open } V \rangle$   $\langle S \subseteq V \rangle$  by force
  obtain  $\mathcal{G}$  where finite  $\mathcal{G} \cup \mathcal{G} = \bigcup \mathcal{F}$ 
    and diaG:  $\wedge X. X \in \mathcal{G} \implies \text{diameter } X < d$ 
    and polyG:  $\wedge X. X \in \mathcal{G} \implies \text{polytope } X$ 
    and affG:  $\wedge X. X \in \mathcal{G} \implies \text{aff\_dim } X \leq \text{aff\_dim } T - 1$ 
    and faceG:  $\wedge X \ Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \ \text{face\_of } X$ 
  proof (rule cell_complex_subdivision_exists [OF  $\langle d > 0 \rangle$   $\langle \text{finite } \mathcal{F} \rangle$  poly_face])
    show  $\wedge X. X \in \mathcal{F} \implies \text{aff\_dim } X \leq \text{aff\_dim } T - 1$ 
    by (simp add: aff)
  qed auto
  obtain h where conth: continuous_on  $(\bigcup \mathcal{G})$  h and him:  $h \ \langle \bigcup \mathcal{G} \subseteq \text{rel\_frontier } T$ 
T and hg:  $\wedge x. x \in \bigcup (\mathcal{G} \cap \text{Pow } V) \implies h \ x = g \ x$ 
  proof (rule extend_map_lemma [of  $\mathcal{G} \ \mathcal{G} \cap \text{Pow } V \ T \ g$ ])
    show continuous_on  $(\bigcup (\mathcal{G} \cap \text{Pow } V))$  g
    by (metis Union_Int_subset Union_Pow_eq  $\langle \text{continuous\_on } V \ g \rangle$  continuous_on_subset
le_inf_iff)
  qed (use  $\langle \text{finite } \mathcal{G} \rangle$  T polyG affG faceG gim image_subset_iff_funcset in auto)
  show ?thesis
  proof
    show continuous_on  $(\bigcup \mathcal{F})$  h
    using  $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle$  conth by auto
    show  $h \in \bigcup \mathcal{F} \rightarrow \text{rel\_frontier } T$ 
    using  $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle$  him by auto
    show  $h \ x = f \ x$  if  $x \in S$  for x
    proof -
      have  $x \in \bigcup \mathcal{G}$ 
      using  $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle$   $\langle S \subseteq \bigcup \mathcal{F} \rangle$  that by auto
      then obtain X where  $x \in X$   $X \in \mathcal{G}$  by blast
      then have diameter X  $< d$  bounded X
      by (auto simp: diaG  $\langle X \in \mathcal{G} \rangle$  polyG polytope_imp_bounded)
      then have  $X \subseteq V$  using d [OF  $\langle x \in S \rangle$ ] diameter_bounded_bound [OF
 $\langle \text{bounded } X \rangle$   $\langle x \in X \rangle$ ]
      by fastforce
      have  $h \ x = g \ x$ 
      using  $\langle X \in \mathcal{G} \rangle$   $\langle X \subseteq V \rangle$   $\langle x \in X \rangle$  hg by auto
      also have  $\dots = f \ x$ 
      by (simp add: gf that)
    qed

```

finally show $h x = f x$.
 qed
 qed
 qed

theorem *extend_map_cell_complex_to_sphere_cofinite:*

assumes *finite* \mathcal{F} **and** $S: S \subseteq \bigcup \mathcal{F}$ *closed* S **and** $T: \text{convex } T$ *bounded* T
and *poly*: $\bigwedge X. X \in \mathcal{F} \implies \text{polytope } X$
and *aff*: $\bigwedge X. X \in \mathcal{F} \implies \text{aff_dim } X \leq \text{aff_dim } T$
and *face*: $\bigwedge X Y. \llbracket X \in \mathcal{F}; Y \in \mathcal{F} \rrbracket \implies (X \cap Y) \text{ face_of } X$
and *contf*: *continuous_on* $S f$ **and** *fim*: $f \in S \rightarrow \text{rel_frontier } T$
obtains $C g$ **where** *finite* C *disjnt* $C S$ *continuous_on* $(\bigcup \mathcal{F} - C) g$
 $g \in (\bigcup \mathcal{F} - C) \rightarrow \text{rel_frontier } T \bigwedge x. x \in S \implies g x = f x$
proof –
obtain $V g$ **where** $S \subseteq V$ *open* V *continuous_on* $V g$ **and** *gim*: $g \in V \rightarrow$
 $\text{rel_frontier } T$ **and** *gf*: $\bigwedge x. x \in S \implies g x = f x$
using *neighbourhood_extension_into_ANR* [*OF* *contf* *fim* $\langle \text{closed } S \rangle$] *ANR_rel_frontier_convex*
 T **by** *blast*
have *compact* S
by (*meson* *assms* *compact_Union* *poly* *polytope_imp_compact* *seq_compact_closed_subset*
seq_compact_eq_compact)
then obtain d **where** $d > 0$ **and** $d: \bigwedge x y. \llbracket x \in S; y \in - V \rrbracket \implies d \leq \text{dist } x y$
using *separate_compact_closed* [*of* $S - V$] $\langle \text{open } V \rangle \langle S \subseteq V \rangle$ **by** *force*
obtain \mathcal{G} **where** *finite* \mathcal{G} $\bigcup \mathcal{G} = \bigcup \mathcal{F}$
and *diaG*: $\bigwedge X. X \in \mathcal{G} \implies \text{diameter } X < d$
and *polyG*: $\bigwedge X. X \in \mathcal{G} \implies \text{polytope } X$
and *affG*: $\bigwedge X. X \in \mathcal{G} \implies \text{aff_dim } X \leq \text{aff_dim } T$
and *faceG*: $\bigwedge X Y. \llbracket X \in \mathcal{G}; Y \in \mathcal{G} \rrbracket \implies X \cap Y \text{ face_of } X$
by (*rule* *cell_complex_subdivision_exists* [*OF* $\langle d > 0 \rangle \langle \text{finite } \mathcal{F} \rangle$ *poly* *aff* *face*])
auto
obtain $C h$ **where** *finite* C **and** *dis*: *disjnt* $C (\bigcup (\mathcal{G} \cap \text{Pow } V))$
and *card*: $\text{card } C \leq \text{card } \mathcal{G}$ **and** *conth*: *continuous_on* $(\bigcup \mathcal{G} - C) h$
and *him*: $h \in (\bigcup \mathcal{G} - C) \rightarrow \text{rel_frontier } T$
and *hg*: $\bigwedge x. x \in \bigcup (\mathcal{G} \cap \text{Pow } V) \implies h x = g x$
proof (*rule* *extend_map_lemma_cofinite* [*of* \mathcal{G} $\mathcal{G} \cap \text{Pow } V T g$])
show *continuous_on* $(\bigcup (\mathcal{G} \cap \text{Pow } V)) g$
by (*metis* *Union_Int_subset_Union_Pow_eq* $\langle \text{continuous_on } V g \rangle$ *contin-*
ous_on_subset_le_inf_iff)
show $g \in \bigcup (\mathcal{G} \cap \text{Pow } V) \rightarrow \text{rel_frontier } T$
using *gim* **by** *force*
qed (*auto* *intro*: $\langle \text{finite } \mathcal{G} \rangle T$ *polyG* *affG* *dest*: *faceG*)
have $S \subseteq \bigcup (\mathcal{G} \cap \text{Pow } V)$
proof
fix x
assume $x \in S$
then have $x \in \bigcup \mathcal{G}$
using $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle \langle S \subseteq \bigcup \mathcal{F} \rangle$ **by** *auto*
then obtain X **where** $x \in X$ $X \in \mathcal{G}$ **by** *blast*

```

then have diameter  $X < d$  bounded  $X$ 
  by (auto simp: diaG  $\langle X \in \mathcal{G} \rangle$  polyG polytope_imp_bounded)
  then have  $X \subseteq V$  using  $d$  [OF  $\langle x \in S \rangle$ ] diameter_bounded_bound [OF
 $\langle$ bounded  $X \rangle \langle x \in X \rangle$ ]
  by fastforce
  then show  $x \in \bigcup (\mathcal{G} \cap \text{Pow } V)$ 
    using  $\langle X \in \mathcal{G} \rangle \langle x \in X \rangle$  by blast
qed
then show ?thesis
  by (metis PowI Union_Pow_eq  $\langle \bigcup \mathcal{G} = \bigcup \mathcal{F} \rangle \langle$ finite  $C \rangle$  conth dis dis-
jnt_Union2 gf hg him subsetD that)
qed

```

9.27.3 Special cases and corollaries involving spheres

proposition extend_map_affine_to_sphere_cofinite_simple:

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$

assumes compact S convex U bounded U

and $\text{aff_dim } T \leq \text{aff_dim } U$

and $S \subseteq T$ **and** $\text{contf: continuous_on } S f$

and $\text{fim: } f \in S \rightarrow \text{rel_frontier } U$

obtains $K g$ **where** finite K $K \subseteq T$ disjoint $K S$ continuous_on $(T - K) g$

$g \in (T - K) \rightarrow \text{rel_frontier } U$

$\bigwedge x. x \in S \implies g x = f x$

proof –

have $\exists K g. \text{finite } K \wedge \text{disjnt } K S \wedge \text{continuous_on } (T - K) g \wedge$

$g \in (T - K) \rightarrow \text{rel_frontier } U \wedge (\forall x \in S. g x = f x)$

if affine $T S \subseteq T$ **and** $\text{aff: aff_dim } T \leq \text{aff_dim } U$ **for** T

proof (cases $S = \{\}$)

case True

show ?thesis

proof (cases $\text{rel_frontier } U = \{\}$)

case True

with \langle bounded $U \rangle$ **have** $\text{aff_dim } U \leq 0$

using affine_bounded_eq_lowdim $\text{rel_frontier_eq_empty}$ **by** auto

with aff **have** $\text{aff_dim } T \leq 0$ **by** auto

then obtain a **where** $T \subseteq \{a\}$

using \langle affine $T \rangle$ affine_bounded_eq_lowdim affine_bounded_eq_trivial **by**
auto

then show ?thesis

using $\langle S = \{\} \rangle$ fim

by (metis Diff_cancel contf disjnt_empty2 finite.emptyI finite_insert fi-
nite_subset)

next

case False

then obtain a **where** $a \in \text{rel_frontier } U$

by auto

then show ?thesis

using continuous_on_const [of a] $\langle S = \{\} \rangle$ **by** force

```

qed
next
case False
have bounded S
  by (simp add: ⟨compact S⟩ compact_imp_bounded)
then obtain b where b: S ⊆ cbox (-b) b
  using bounded_subset_cbox_symmetric by blast
define bbox where bbox ≡ cbox (-(b+One)) (b+One)
have cbox (-b) b ⊆ bbox
  by (auto simp: bbox_def algebra_simps intro!: subset_box_imp)
with b ⟨S ⊆ T⟩ have S ⊆ bbox ∩ T
  by auto
then have Ssub: S ⊆ ⋃{bbox ∩ T}
  by auto
then have aff_dim (bbox ∩ T) ≤ aff_dim U
  by (metis aff_dim_subset inf_commute inf_le1 order_trans)
obtain K g where K: finite K disjnt K S
  and contg: continuous_on (⋃{bbox ∩ T} - K) g
  and gim: g ∈ (⋃{bbox ∩ T} - K) → rel_frontier U
  and gf: ∧x. x ∈ S ⇒ g x = f x
proof (rule extend_map_cell_complex_to_sphere_cofinite
  [OF _ Ssub _ ⟨convex U⟩ ⟨bounded U⟩ _ _ _ contf fim])
  show closed S
    using ⟨compact S⟩ compact_eq_bounded_closed by auto
  show poly: ∧X. X ∈ {bbox ∩ T} ⇒ polytope X
  by (simp add: polytope_Int_polyhedron bbox_def polytope_interval affine_imp_polyhedron
    ⟨affine T⟩)
  show ∧X Y. [X ∈ {bbox ∩ T}; Y ∈ {bbox ∩ T}] ⇒ X ∩ Y face_of X
    by (simp add: poly_face_of_refl polytope_imp_convex)
  show ∧X. X ∈ {bbox ∩ T} ⇒ aff_dim X ≤ aff_dim U
    by (simp add: ⟨aff_dim (bbox ∩ T) ≤ aff_dim U⟩)
qed auto
define fro where fro ≡ λd. frontier(cbox (-(b + d *R One)) (b + d *R One))
obtain d where d12: 1/2 ≤ d d ≤ 1 and dd: disjnt K (fro d)
proof (rule disjoint_family_elem_disjnt [OF _ ⟨finite K⟩])
  show infinite {1/2..1::real}
    by (simp add: infinite_Icc)
  have dis1: disjnt (fro x) (fro y) if x < y for x y
    by (auto simp: algebra_simps that subset_box_imp disjnt_Diff1 frontier_def
      fro_def)
  then show disjoint_family_on fro {1/2..1}
    by (auto simp: disjoint_family_on_def disjnt_def neq_iff)
qed auto
define c where c ≡ b + d *R One
have csub: cbox (-b) b ⊆ cbox (-c) c cbox (-b) b ⊆ cbox (-c) c cbox (-c)
c ⊆ bbox
  using d12 by (auto simp: algebra_simps subset_box_imp c_def bbox_def)
have clo_cbT: closed (cbox (-c) c ∩ T)
  by (simp add: affine_closed closed_Int closed_cbox ⟨affine T⟩)

```

```

have cpT_ne: cbox (- c) c ∩ T ≠ {}
  using ⟨S ≠ {}⟩ b csub(2) ⟨S ⊆ T⟩ by fastforce
have closest_point (cbox (- c) c ∩ T) x ∉ K if x ∈ T x ∉ K for x
proof (cases x ∈ cbox (-c) c)
  case True with that show ?thesis
    by (simp add: closest_point_self)
next
case False
have int_ne: interior (cbox (-c) c) ∩ T ≠ {}
  using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b ⟨cbox (- b) b ⊆ box (- c) c⟩ by force
have convex T
  by (meson ⟨affine T⟩ affine_imp_convex)
then have x ∈ affine hull (cbox (- c) c ∩ T)
  by (metis Int_commute Int_iff ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ csub(1) ⟨x ∈ T⟩
affine_hull_convex_Int_nonempty_interior_all_not_in_conv b hull_inc inf.orderE
interior_cbox)
  then have x ∈ affine hull (cbox (- c) c ∩ T) - rel_interior (cbox (- c) c
∩ T)
  by (meson DiffI False Int_iff rel_interior_subset subsetCE)
  then have closest_point (cbox (- c) c ∩ T) x ∈ rel_frontier (cbox (- c) c
∩ T)
  by (rule closest_point_in_rel_frontier [OF clo_cbT cpT_ne])
moreover have (rel_frontier (cbox (- c) c ∩ T)) ⊆ fro d
  by (subst convex_affine_rel_frontier_Int [OF _ ⟨affine T⟩ int_ne]) (auto
simp: fro_def c_def)
ultimately show ?thesis
  using dd by (force simp: disjnt_def)
qed
then have cpt_subset: closest_point (cbox (- c) c ∩ T) ‘ (T - K) ⊆ ⋃ {bbox
∩ T} - K
  using closest_point_in_set [OF clo_cbT cpT_ne] csub(3) by force
show ?thesis
proof (intro conjI ballI exI)
  have continuous_on (T - K) (closest_point (cbox (- c) c ∩ T))
  proof (rule continuous_on_closest_point)
    show convex (cbox (- c) c ∩ T)
      by (simp add: affine_imp_convex convex_Int ⟨affine T⟩)
    show closed (cbox (- c) c ∩ T)
      using clo_cbT by blast
    show cbox (- c) c ∩ T ≠ {}
      using ⟨S ≠ {}⟩ csub(2) b that by auto
  qed
  then show continuous_on (T - K) (g ∘ closest_point (cbox (- c) c ∩ T))
  by (metis continuous_on_compose continuous_on_subset [OF contg cpt_subset])
  have (g ∘ closest_point (cbox (- c) c ∩ T)) ‘ (T - K) ⊆ g ‘ (⋃ {bbox ∩ T}
- K)
  by (metis image_comp image_mono cpt_subset)
  also have ... ⊆ rel_frontier U
  using gim by blast

```

```

finally show (g ◦ closest_point (cbox (- c) c ∩ T)) ∈ (T - K) → rel_frontier
U
  by blast
show (g ◦ closest_point (cbox (- c) c ∩ T)) x = f x if x ∈ S for x
proof -
  have (g ◦ closest_point (cbox (- c) c ∩ T)) x = g x
    unfolding o_def
    by (metis IntI ⟨S ⊆ T⟩ b cbsub(2) closest_point_self subset_eq that)
  also have ... = f x
    by (simp add: that gf)
  finally show ?thesis .
qed
qed (auto simp: K)
qed
then obtain K g where finite K disjoint K S
  and contg: continuous_on (affine hull T - K) g
  and gim: g ∈ (affine hull T - K) → rel_frontier U
  and gf: ∧x. x ∈ S ⇒ g x = f x
  by (metis aff affine_affine_hull aff_dim_affine_hull
    order_trans [OF ⟨S ⊆ T⟩ hull_subset [of T affine]])
then obtain K g where finite K disjoint K S
  and contg: continuous_on (T - K) g
  and gim: g ∈ (T - K) → rel_frontier U
  and gf: ∧x. x ∈ S ⇒ g x = f x
  by (rule_tac K=K and g=g in that) (auto simp: hull_inc elim: continu-
ous_on_subset)
  then show ?thesis
  by (rule_tac K=K ∩ T and g=g in that) (auto simp: disjoint_iff Diff_Int
contg)
qed

```

9.27.4 Extending maps to spheres

```

lemma extend_map_affine_to_sphere1:
  fixes f :: 'a::euclidean_space ⇒ 'b::topological_space
  assumes finite K affine U and contf: continuous_on (U - K) f
  and fim: f ∈ (U - K) → T
  and comps: ∧C. [C ∈ components(U - S); C ∩ K ≠ {}] ⇒ C ∩ L ≠ {}
  and clo: closedin (top_of_set U) S and K: disjoint K S K ⊆ U
  obtains g where continuous_on (U - L) g g ∈ (U - L) → T ∧x. x ∈ S ⇒
g x = f x
proof (cases K = {})
  case True
  then show ?thesis
  by (metis DiffD1 Diff_empty Diff_subset PiE Pi_I contf continuous_on_subset
fim that)
next
  case False
  have S ⊆ U

```

```

    using clo closedin_limpt by blast
  then have  $(U - S) \cap K \neq \{\}$ 
    by (metis Diff_triv False Int_Diff K disjnt_def inf.absorb_iff2 inf_commute)
  then have  $\bigcup (\text{components } (U - S)) \cap K \neq \{\}$ 
    using Union_components by simp
  then obtain C0 where C0:  $C0 \in \text{components } (U - S) \ C0 \cap K \neq \{\}$ 
    by blast
  have convex U
    by (simp add: affine_imp_convex ⟨affine U⟩)
  then have locally_connected U
    by (rule convex_imp_locally_connected)
  have  $\exists a \ g. \ a \in C \wedge a \in L \wedge \text{continuous\_on } (S \cup (C - \{a\})) \ g \wedge$ 
     $g \text{ ' } (S \cup (C - \{a\})) \subseteq T \wedge (\forall x \in S. \ g \ x = f \ x)$ 
    if C:  $C \in \text{components } (U - S)$  and CK:  $C \cap K \neq \{\}$  for C
  proof -
    have  $C \subseteq U - S \ C \cap L \neq \{\}$ 
      by (simp_all add: in_components_subset comps that)
    then obtain a where a:  $a \in C \ a \in L$  by auto
    have opeUC:  $\text{openin } (\text{top\_of\_set } U) \ C$ 
      by (metis C ⟨locally_connected U⟩ clo closedin_def locally_connected_open_component
topspace_euclidean_subtopology)
    then obtain d where  $C \subseteq U \ 0 < d$  and d:  $\text{cball } a \ d \cap U \subseteq C$ 
      using openin_contains_cball by (metis ⟨a ∈ C⟩)
    then have  $\text{ball } a \ d \cap U \subseteq C$ 
      by auto
    obtain h k where homhk:  $\text{homeomorphism } (S \cup C) \ (S \cup C) \ h \ k$ 
      and subC:  $\{x. (\neg (h \ x = x \wedge k \ x = x))\} \subseteq C$ 
      and bou:  $\text{bounded } \{x. (\neg (h \ x = x \wedge k \ x = x))\}$ 
      and hin:  $\bigwedge x. \ x \in C \cap K \implies h \ x \in \text{ball } a \ d \cap U$ 
    proof (rule homeomorphism_grouping_points_exists_gen [of C ball a d ∩ U
C ∩ K S ∪ C])
      show  $\text{openin } (\text{top\_of\_set } C) \ (\text{ball } a \ d \cap U)$ 
        by (metis open_ball ⟨C ⊆ U⟩ ⟨ball a d ∩ U ⊆ C⟩ inf.absorb_iff2 inf.orderE
inf_assoc open_openin openin_subtopology)
      show  $\text{openin } (\text{top\_of\_set } (\text{affine hull } C)) \ C$ 
        by (metis ⟨a ∈ C⟩ ⟨openin (top_of_set U) C⟩ affine_hull_eq affine_hull_openin
all_not_in_conv ⟨affine U⟩)
      show  $\text{ball } a \ d \cap U \neq \{\}$ 
        using ⟨0 < d⟩ ⟨C ⊆ U⟩ ⟨a ∈ C⟩ by force
      show finite (C ∩ K)
        by (simp add: ⟨finite K⟩)
      show  $S \cup C \subseteq \text{affine hull } C$ 
        by (metis ⟨S ⊆ U⟩ ⟨a ∈ C⟩ affine_hull_eq affine_hull_openin assms(2)
empty_iff hull_subset le_sup_iff opeUC)
      show connected C
        by (metis C in_components_connected)
    qed auto
  have a_BU:  $a \in \text{ball } a \ d \cap U$ 
    using ⟨0 < d⟩ ⟨C ⊆ U⟩ ⟨a ∈ C⟩ by auto

```

```

have rel_frontier (cball a d ∩ U) retract_of (affine hull (cball a d ∩ U) -
{a})
proof (rule rel_frontier_retract_of_punctured_affine_hull)
  show bounded (cball a d ∩ U) convex (cball a d ∩ U)
    by (auto simp: ⟨convex U⟩ convex_Int)
  show a ∈ rel_interior (cball a d ∩ U)
    by (metis ⟨affine U⟩ convex_cball_empty_iff_interior_cball a_BU rel_interior_convex_Int_affine)
qed
moreover have rel_frontier (cball a d ∩ U) = frontier (cball a d) ∩ U
  by (metis a_BU ⟨affine U⟩ convex_affine_rel_frontier_Int convex_cball
equals0D interior_cball)
moreover have affine hull (cball a d ∩ U) = U
  by (metis ⟨convex U⟩ a_BU affine_hull_convex_Int_nonempty_interior
affine_hull_eq ⟨affine U⟩ equals0D inf commute interior_cball)
ultimately have frontier (cball a d) ∩ U retract_of (U - {a})
  by metis
then obtain r where contr: continuous_on (U - {a}) r
  and rim: r ∈ (U - {a}) → sphere a d r ∈ (U - {a}) → U
  and req: ∧x. x ∈ sphere a d ∩ U ⇒ r x = x
  using ⟨affine U⟩ by (force simp: retract_of_def retraction_def hull_same)
define j where j ≡ λx. if x ∈ ball a d then r x else x
have kj: ∧x. x ∈ S ⇒ k (j x) = x
  using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨ball a d ∩ U ⊆ C⟩ j_def subC by auto
have Uaeq: U - {a} = (cball a d - {a}) ∩ U ∪ (U - ball a d)
  using ⟨0 < d⟩ by auto
have jim: j ‘ (S ∪ (C - {a})) ⊆ (S ∪ C) - ball a d
proof clarify
  fix y assume y ∈ S ∪ (C - {a})
  then have y ∈ U - {a}
    using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨a ∈ C⟩ by auto
  then have r y ∈ sphere a d
    using rim by auto
  then show j y ∈ S ∪ C - ball a d
    unfolding j_def
    using ⟨y ∈ S ∪ (C - {a})⟩ ⟨y ∈ U - {a}⟩ d rim(2) by auto
qed
have contj: continuous_on (U - {a}) j
  unfolding j_def Uaeq
proof (intro continuous_on_cases_local continuous_on_id, simp_all add: req
closedin_closed Uaeq [symmetric])
  show ∃ T. closed T ∧ (cball a d - {a}) ∩ U = (U - {a}) ∩ T
    using affine_closed ⟨affine U⟩ by (rule_tac x=(cball a d) ∩ U in exI) blast
  show ∃ T. closed T ∧ U - ball a d = (U - {a}) ∩ T
    using ⟨0 < d⟩ ⟨affine U⟩
    by (rule_tac x=U - ball a d in exI) (force simp: affine_closed)
  show continuous_on ((cball a d - {a}) ∩ U) r
    by (force intro: continuous_on_subset [OF contr])
qed
have fT: x ∈ U - K ⇒ f x ∈ T for x

```



```

    using fim by blast
  show ?thesis
  proof (intro conjI exI)
    show continuous_on (S ∪ (C - {a})) (f ∘ k ∘ j)
    proof (intro continuous_on_compose)
      have S ∪ (C - {a}) ⊆ U - {a}
        using ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨a ∈ C⟩ by force
      then show continuous_on (S ∪ (C - {a})) j
        by (rule continuous_on_subset [OF contj])
      have j '(S ∪ (C - {a})) ⊆ S ∪ C
        using jim ⟨C ⊆ U - S⟩ ⟨S ⊆ U⟩ ⟨ball a d ∩ U ⊆ C⟩ j_def by blast
      then show continuous_on (j '(S ∪ (C - {a}))) k
        by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homhk]])
      show continuous_on (k '(j '(S ∪ (C - {a})))) f
    proof (clarify intro!: continuous_on_subset [OF contf])
      fix y assume y ∈ S ∪ (C - {a})
      have ky: k y ∈ S ∪ C
        using homeomorphism_image2 [OF homhk] ⟨y ∈ S ∪ (C - {a})⟩ by
blast
      have jy: j y ∈ S ∪ C - ball a d
        using Un_iff ⟨y ∈ S ∪ (C - {a})⟩ jim by auto
      have k (j y) ∈ U
        using ⟨C ⊆ U⟩ ⟨S ⊆ U⟩ homeomorphism_image2 [OF homhk] jy by
blast
      moreover have k (j y) ∉ K
        using K unfolding disjnt_iff
      by (metis DiffE Int_iff Un_iff hin homeomorphism_def homhk image_eqI
jy)
      ultimately show k (j y) ∈ U - K
        by blast
    qed
  qed
  have ST: ∧x. x ∈ S ⟹ (f ∘ k ∘ j) x ∈ T
  proof (simp add: kj)
    show ∧x. x ∈ S ⟹ f x ∈ T
      using K ⟨S ⊆ U⟩ fT unfolding disjnt_iff by auto
  qed
  moreover have (f ∘ k ∘ j) x ∈ T if x ∈ C x ≠ a x ∉ S for x
  proof -
    have rx: r x ∈ sphere a d
      using ⟨C ⊆ U⟩ rim that by fastforce
    have jj: j x ∈ S ∪ C - ball a d
      using jim that by blast
    have k (j x) = j x ⟶ k (j x) ∈ C ∨ j x ∈ C
      by (metis Diff_iff Int_iff Un_iff ⟨S ⊆ U⟩ subsetD d j_def jj rx sphere_cball
that(1))
    then have kj: k (j x) ∈ C
      using homeomorphism_apply2 [OF homhk, of j x] ⟨C ⊆ U⟩ ⟨S ⊆ U⟩ a
rx

```

```

    by (metis (mono_tags, lifting) Diff_iff subsetD jj mem_Collect_eq subC)
  then show ?thesis
    by (metis DiffE DiffI IntD1 IntI ⟨C ⊆ U⟩ comp_apply fT hin homeomor-
    phism_apply2 homhk jj kj subset_eq)
  qed
  ultimately show (f ∘ k ∘ j) ‘(S ∪ (C - {a})) ⊆ T
    by force
  show ∀x∈S. (f ∘ k ∘ j) x = f x using kj by simp
  qed (auto simp: a)
  qed
  then obtain a h where
    ah: ∧C. [C ∈ components (U - S); C ∩ K ≠ {}]
      ⇒ a C ∈ C ∧ a C ∈ L ∧ continuous_on (S ∪ (C - {a C})) (h C) ∧
      h C ‘(S ∪ (C - {a C})) ⊆ T ∧ (∀x ∈ S. h C x = f x)
    using that by metis
  define F where F ≡ {C ∈ components (U - S). C ∩ K ≠ {}}
  define G where G ≡ {C ∈ components (U - S). C ∩ K = {}}
  define UF where UF ≡ (∪ C∈F. C - {a C})
  have C0 ∈ F
    by (auto simp: F_def C0)
  have finite F
  proof (subst finite_image_iff [of λC. C ∩ K F, symmetric])
    show inj_on (λC. C ∩ K) F
      unfolding F_def inj_on_def
      using components_nonoverlap by blast
    show finite ((λC. C ∩ K) ‘ F)
      unfolding F_def
      by (rule finite_subset [of _ Pow K]) (auto simp: ⟨finite K⟩)
  qed
  obtain g where contg: continuous_on (S ∪ UF) g
    and gh: ∧x i. [i ∈ F; x ∈ (S ∪ UF) ∩ (S ∪ (i - {a i}))]
      ⇒ g x = h i x
  proof (rule pasting_lemma_exists_closed [OF ⟨finite F⟩])
    let ?X = top_of_set (S ∪ UF)
    show topspace ?X ⊆ (∪ C∈F. S ∪ (C - {a C}))
      using ⟨C0 ∈ F⟩ by (force simp: UF_def)
    show closedin (top_of_set (S ∪ UF)) (S ∪ (C - {a C}))
      if C ∈ F for C
    proof (rule closedin_closed_subset [of U S ∪ C])
      have C ∈ components (U - S)
        using F_def that by blast
      then show closedin (top_of_set U) (S ∪ C)
        by (rule closedin_Un_complement_component [OF ⟨locally connected U⟩
        clo])
    next
    have x = a C' if C' ∈ F x ∈ C' x ∉ U for x C'
    proof -
      have ∀A. x ∈ ∪ A ∨ C' ∉ A
        using ⟨x ∈ C'⟩ by blast

```

```

    with that show  $x = a \ C'$ 
      by (metis (lifting) DiffD1 F_def Union_components mem_Collect_eq)
    qed
  then show  $S \cup UF \subseteq U$ 
    using  $\langle S \subseteq U \rangle$  by (force simp: UF_def)
  next
    show  $S \cup (C - \{a \ C'\}) = (S \cup C) \cap (S \cup UF)$ 
      using F_def UF_def components_nonoverlap that by auto
    qed
  show continuous_map (subtopology ?X (S  $\cup$  (C' - {a C'}))) euclidean (h C')
if C'  $\in$  F for C'
  proof -
    have C':  $C' \in \text{components } (U - S) \ C' \cap K \neq \{\}$ 
      using F_def that by blast+
    show ?thesis
      using ah [OF C'] by (auto simp: F_def subtopology_subtopology intro:
continuous_on_subset)
    qed
    show  $\bigwedge i \ j \ x. \llbracket i \in F; j \in F; \llbracket x \in \text{topspace } ?X \cap (S \cup (i - \{a \ i\})) \cap (S \cup (j - \{a \ j\})) \rrbracket \rrbracket \implies h \ i \ x = h \ j \ x$ 
      using components_eq by (fastforce simp: components_eq F_def ah)
    qed auto
  have SU':  $S \cup \bigcup G \cup (S \cup UF) \subseteq U$ 
    using  $\langle S \subseteq U \rangle$  in_components_subset by (auto simp: F_def G_def UF_def)
  have clo1: closedin (top_of_set (S  $\cup$   $\bigcup$  G  $\cup$  (S  $\cup$  UF))) (S  $\cup$   $\bigcup$  G)
  proof (rule closedin_closed_subset [OF _ SU'])
    have *:  $\bigwedge C. C \in F \implies \text{openin } (\text{top\_of\_set } U) \ C$ 
      unfolding F_def
      by (metis (no_types, lifting)  $\langle \text{locally connected } U \rangle$  clo_closedin_def locally_connected_open_component mem_Collect_eq topspace_euclidean_subtopology)
    show closedin (top_of_set U) (U - UF)
      unfolding UF_def by (force intro: openin_delete *)
    have  $(\bigcup_{x \in F} x - \{a \ x\}) \cap S = \{\} \cup G \subseteq U$ 
      using in_components_subset by (auto simp: F_def G_def)
    moreover have  $\bigcup G \cap UF = \{\}$ 
      using components_nonoverlap by (fastforce simp: F_def G_def UF_def)
    ultimately show  $S \cup \bigcup G = (U - UF) \cap (S \cup \bigcup G \cup (S \cup UF))$ 
      using UF_def  $\langle S \subseteq U \rangle$  by auto
    qed
  have clo2: closedin (top_of_set (S  $\cup$   $\bigcup$  G  $\cup$  (S  $\cup$  UF))) (S  $\cup$  UF)
  proof (rule closedin_closed_subset [OF _ SU'])
    show closedin (top_of_set U)  $(\bigcup_{C \in F} S \cup C)$ 
      proof (rule closedin_Union)
        show  $\bigwedge T. T \in (\bigcup) S \ ' F \implies \text{closedin } (\text{top\_of\_set } U) \ T$ 
          using F_def  $\langle \text{locally connected } U \rangle$  clo_closedin_Un_complement_component
        by blast
      qed
    qed
  have SU:  $S \cup UF = (\bigcup_{C \in F} S \cup C) \cap (S \cup \bigcup G \cup (S \cup UF))$ 

```

```

proof
  show  $\bigcup ((\bigcup S) \setminus F) \cap (S \cup \bigcup G \cup (S \cup UF)) \subseteq S \cup UF$ 
    using components_eq [of  $\_ U-S$ ]
    by (auto simp add: F_def G_def UF_def disjoint_iff_not_equal)
  qed (use UF_def  $\langle C0 \in F \rangle$  in blast)
qed
have SUG:  $S \cup \bigcup G \subseteq U - K$ 
  using  $\langle S \subseteq U \rangle$  K in_components_subset [of  $\_ U-S$ ] by (force simp: G_def
disjnt_iff)
then have contf': continuous_on  $(S \cup \bigcup G)$  f
  by (rule continuous_on_subset [OF contf])
have contg': continuous_on  $(S \cup UF)$  g
  by (simp add: contg)
have  $\bigwedge x. \llbracket S \subseteq U; x \in S \rrbracket \implies f\ x = g\ x$ 
  by (subst gh) (auto simp: ah C0 intro: \langle C0 \in F \rangle)
then have f_eq_g:  $\bigwedge x. x \in S \cup UF \wedge x \in S \cup \bigcup G \implies f\ x = g\ x$ 
  using  $\langle S \subseteq U \rangle$  components_eq [of  $\_ U-S$ ] by (fastforce simp add: F_def
G_def UF_def)
have cont: continuous_on  $(S \cup \bigcup G \cup (S \cup UF))$   $(\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f\ x$ 
else } g\ x)
  by (blast intro: continuous_on_cases_local [OF clo1 clo2 contf' contg' f_eq_g,
of  $\lambda x. x \in S \cup \bigcup G$ ])
show ?thesis
proof
have UF:  $\bigcup F - L \subseteq UF$ 
  unfolding F_def UF_def using ah by blast
have  $U - S - L = \bigcup (\text{components } (U - S)) - L$ 
  by simp
also have  $\dots = \bigcup F \cup \bigcup G - L$ 
  unfolding F_def G_def by blast
also have  $\dots \subseteq UF \cup \bigcup G$ 
  using UF by blast
finally have  $U - L \subseteq S \cup \bigcup G \cup (S \cup UF)$ 
  by blast
then show continuous_on  $(U - L)$   $(\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f\ x \text{ else } g\ x)$ 
  by (rule continuous_on_subset [OF cont])
have  $((U - L) \cap \{x. x \notin S \wedge (\forall xa \in G. x \notin xa)\}) \subseteq ((U - L) \cap (-S \cap UF))$ 
  using  $\langle U - L \subseteq S \cup \bigcup G \cup (S \cup UF) \rangle$  by auto
moreover have  $g \setminus ((U - L) \cap (-S \cap UF)) \subseteq T$ 
proof -
have  $g\ x \in T$  if  $x \in U$   $x \notin L$   $x \notin S$   $C \in F$   $x \in C$   $x \neq a$  for  $x\ C$ 
proof (subst gh)
  show  $x \in (S \cup UF) \cap (S \cup (C - \{a\}))$ 
    using that by (auto simp: UF_def)
  show  $h\ C\ x \in T$ 
    using ah that by (fastforce simp add: F_def)
qed (rule that)
then show ?thesis
  by (force simp: UF_def)

```

```

qed
ultimately have  $g \text{ ' } ((U - L) \cap \{x. x \notin S \wedge (\forall xa \in G. x \notin xa)\}) \subseteq T$ 
  using image_mono order_trans by blast
moreover have  $f \text{ ' } ((U - L) \cap (S \cup \bigcup G)) \subseteq T$ 
  using fim SUG by blast
ultimately show  $(\lambda x. \text{if } x \in S \cup \bigcup G \text{ then } f x \text{ else } g x) \in (U - L) \rightarrow T$ 
  by force
show  $\bigwedge x. x \in S \implies (\text{if } x \in S \cup \bigcup G \text{ then } f x \text{ else } g x) = f x$ 
  by (simp add: F_def G_def)
qed
qed

lemma extend_map_affine_to_sphere2:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes compact S convex U bounded U affine T S  $\subseteq T$ 
    and affTU: aff_dim T  $\leq$  aff_dim U
    and contf: continuous_on S f
    and fim: f  $\in S \rightarrow \text{rel\_frontier } U$ 
    and ovlap:  $\bigwedge C. C \in \text{components}(T - S) \implies C \cap L \neq \{\}$ 
  obtains  $K g$  where finite K K  $\subseteq L$   $K \subseteq T$  disjnt K S
    continuous_on (T - K) g  $g \in (T - K) \rightarrow \text{rel\_frontier } U$ 
     $\bigwedge x. x \in S \implies g x = f x$ 

proof -
  obtain  $K g$  where finite K K  $\subseteq T$  disjnt K S
    and contg: continuous_on (T - K) g
    and gim: g  $\in (T - K) \rightarrow \text{rel\_frontier } U$ 
    and gf:  $\bigwedge x. x \in S \implies g x = f x$ 
  using assms extend_map_affine_to_sphere_cofinite_simple by metis
  have  $\exists y C. C \in \text{components}(T - S) \wedge x \in C \wedge y \in C \wedge y \in L$  if  $x \in K$  for  $x$ 
  proof -
    have  $x \in T - S$ 
    using  $\langle K \subseteq T \rangle \langle \text{disjnt } K S \rangle$  disjnt_def that by fastforce
    then obtain  $C$  where  $C \in \text{components}(T - S)$   $x \in C$ 
    by (metis UnionE Union_components)
    with ovlap [of C] show ?thesis
    by blast
  qed
  then obtain  $\xi$  where  $\xi: \bigwedge x. x \in K \implies \exists C. C \in \text{components}(T - S) \wedge x \in C \wedge \xi x \in C \wedge \xi x \in L$ 
  by metis
  obtain  $h$  where conth: continuous_on (T -  $\xi \text{ ' } K$ ) h
    and him: h  $\in (T - \xi \text{ ' } K) \rightarrow \text{rel\_frontier } U$ 
    and hg:  $\bigwedge x. x \in S \implies h x = g x$ 
  proof (rule extend_map_affine_to_sphere1 [OF  $\langle \text{finite } K \rangle \langle \text{affine } T \rangle$  contg gim,
    of  $S \xi \text{ ' } K$ ])
    show cloTS: closedin (top_of_set T) S
    by (simp add:  $\langle \text{compact } S \rangle \langle S \subseteq T \rangle$  closed_subset compact_imp_closed)
    show  $\bigwedge C. \llbracket C \in \text{components}(T - S); C \cap K \neq \{\} \rrbracket \implies C \cap \xi \text{ ' } K \neq \{\}$ 

```

```

    using  $\xi$  components_eq by blast
qed (use K in auto)
show ?thesis
proof
  show *:  $\xi \text{ ' } K \subseteq L$ 
    using  $\xi$  by blast
  show finite ( $\xi \text{ ' } K$ )
    by (simp add: K)
  show  $\xi \text{ ' } K \subseteq T$ 
    by clarify (meson  $\xi$  Diff_iff contra_subsetD in_components_subset)
  show continuous_on ( $T - \xi \text{ ' } K$ ) h
    by (rule conth)
  show disjnt ( $\xi \text{ ' } K$ ) S
    using K  $\xi$  in_components_subset by (fastforce simp: disjnt_def)
qed (simp_all add: him hg gf)
qed

```

proposition *extend_map_affine_to_sphere_cofinite_gen:*

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes SUT: compact S convex U bounded U affine T S  $\subseteq$  T
  and aff: aff_dim T  $\leq$  aff_dim U
  and conf: continuous_on S f
  and fim: f  $\in$  S  $\rightarrow$  rel_frontier U
  and dis:  $\bigwedge C. \llbracket C \in \text{components}(T - S); \text{bounded } C \rrbracket \implies C \cap L \neq \{\}$ 
obtains K g where finite K K  $\subseteq$  L K  $\subseteq$  T disjnt K S continuous_on (T - K)
g

```

$$g \in (T - K) \rightarrow \text{rel_frontier } U$$

$$\bigwedge x. x \in S \implies g x = f x$$

```

proof (cases S = {})
case True
  show ?thesis
proof (cases rel_frontier U = {})
case True
  with aff have aff_dim T  $\leq$  0
  using affine_bounded_eq_lowdim <bounded U> order_trans
  by (auto simp add: rel_frontier_eq_empty)
  with aff_dim_geq [of T] consider aff_dim T = -1 | aff_dim T = 0
  by linarith
  then show ?thesis
proof cases
  assume aff_dim T = -1
  then have T = {}
  by (simp add: aff_dim_empty)
  then show ?thesis
  by (rule_tac K={ } in that) auto
next
  assume aff_dim T = 0
  then obtain a where T = {a}

```

```

    using aff_dim_eq_0 by blast
  then have a ∈ L
    using dis [of {a}] ⟨S = {}⟩ by (auto simp: in_components_self)
  with ⟨S = {}⟩ ⟨T = {a}⟩ show ?thesis
    by (rule_tac K={a} and g=f in that) auto
qed
next
case False
then obtain y where y ∈ rel_frontier U
  by auto
with ⟨S = {}⟩ show ?thesis
  by (rule_tac K={} and g=λx. y in that) (auto)
qed
next
case False
have bounded S
  by (simp add: assms compact_imp_bounded)
then obtain b where b: S ⊆ cbox (-b) b
  using bounded_subset_cbox_symmetric by blast
define LU where LU ≡ L ∪ (⋃ {C ∈ components (T - S). ¬bounded C} -
cbox (-(b+One)) (b+One))
obtain K g where finite K K ⊆ LU K ⊆ T disjnt K S
  and contg: continuous_on (T - K) g
  and gim: g ∈ (T - K) → rel_frontier U
  and gf: ∧x. x ∈ S ⇒ g x = f x
proof (rule extend_map_affine_to_sphere2 [OF SUT aff contf fim])
  show C ∩ LU ≠ {} if C ∈ components (T - S) for C
  proof (cases bounded C)
    case True
    with dis that show ?thesis
      unfolding LU_def by fastforce
  next
  case False
  then have ¬ bounded (⋃{C ∈ components (T - S). ¬ bounded C})
    by (metis (no_types, lifting) Sup_upper bounded_subset mem_Collect_eq
that)
  then show ?thesis
    by (simp add: LU_def disjoint_iff) (meson False bounded_cbox bounded_subset
subset_iff that)
  qed
qed blast
have *: False if x ∈ cbox (- b - m *R One) (b + m *R One)
  x ∉ cbox (- b - n *R One) (b + n *R One)
  0 ≤ m m < n n ≤ 1 for m n x
  using that by (auto simp: mem_box algebra_simps)
have disjoint_family_on (λd. frontier (cbox (- b - d *R One) (b + d *R One)))
{1 / 2..1}
  by (auto simp: disjoint_family_on_def neg_iff_frontier_def dest: *)
then obtain d where d12: 1/2 ≤ d d ≤ 1

```

```

    and ddis: disjnt K (frontier (cbox (-(b + d *R One)) (b + d *R
One)))
    using disjoint_family_elem_disjnt [of {1/2..1::real} K λd. frontier (cbox
(-(b + d *R One)) (b + d *R One))]
    by (auto simp: ⟨finite K⟩)
    define c where c ≡ b + d *R One
    have cbsub: cbox (-b) b ⊆ box (-c) c
      cbox (-b) b ⊆ cbox (-c) c
      cbox (-c) c ⊆ cbox (-(b+One)) (b+One)
    using d12 by (simp_all add: subset_box c_def inner_diff_left inner_left_distrib)
    have clo_cT: closed (cbox (-c) c ∩ T)
    using affine_closed ⟨affine T⟩ by blast
    have cT_ne: cbox (-c) c ∩ T ≠ {}
    using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b cbsub by fastforce
    have S_sub_cc: S ⊆ cbox (-c) c
    using ⟨cbox (-b) b ⊆ cbox (-c) c⟩ b by auto
    show ?thesis
    proof
      show finite (K ∩ cbox (-(b+One)) (b+One))
      using ⟨finite K⟩ by blast
      show K ∩ cbox (-(b + One)) (b + One) ⊆ L
      using ⟨K ⊆ LU⟩ by (auto simp: LU_def)
      show K ∩ cbox (-(b + One)) (b + One) ⊆ T
      using ⟨K ⊆ T⟩ by auto
      show disjnt (K ∩ cbox (-(b + One)) (b + One)) S
      using ⟨disjnt K S⟩ by (simp add: disjnt_def disjoint_eq_subset_Cmpl
inf.coboundedI1)
      have cloTK: closest_point (cbox (-c) c ∩ T) x ∈ T - K
      if x ∈ T and Knot: x ∈ K ⟶ x ∉ cbox (-b - One) (b + One) for
x
    proof (cases x ∈ cbox (-c) c)
      case True
      with ⟨x ∈ T⟩ show ?thesis
      using cbsub(3) Knot by (force simp: closest_point_self)
    next
      case False
      have clo_in_rf: closest_point (cbox (-c) c ∩ T) x ∈ rel_frontier (cbox (-
c) c ∩ T)
      proof (intro closest_point_in_rel_frontier [OF clo_cT cT_ne] DiffI notI)
      have T ∩ interior (cbox (-c) c) ≠ {}
      using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b cbsub(1) by fastforce
      then show x ∈ affine_hull (cbox (-c) c ∩ T)
      by (simp add: Int_commute affine_hull_affine_Int_nonempty_interior
⟨affine T⟩ hull_inc that(1))
    next
      show False if x ∈ rel_interior (cbox (-c) c ∩ T)
      proof -
      have interior (cbox (-c) c) ∩ T ≠ {}
      using ⟨S ≠ {}⟩ ⟨S ⊆ T⟩ b cbsub(1) by fastforce

```



```

then have affine_hull (T ∩ cbox (- c) c) = T
  using affine_hull_convex_Int_nonempty_interior [of T cbox (- c) c]
  by (simp add: affine_imp_convex ⟨affine T⟩ inf_commute)
then show ?thesis
  by (meson subsetD le_inf_iff rel_interior_subset that False)
qed
qed
have closest_point (cbox (- c) c ∩ T) x ∉ K
proof
  assume inK: closest_point (cbox (- c) c ∩ T) x ∈ K
  have  $\bigwedge x. x \in K \implies x \notin \text{frontier } (\text{cbox } (- (b + d *R \text{One})) (b + d *R \text{One}))$ 
  by (metis ddis_disjnt_iff)
  then show False
  by (metis DiffI Int_iff ⟨affine T⟩ cT_ne c_def clo_cT clo_in_rf closest_point_in_set
    convex_affine_rel_frontier_Int convex_box(1) empty_iff frontier_cbox_inK interior_cbox)
qed
then show ?thesis
  using cT_ne clo_cT closest_point_in_set by blast
qed
have convex (cbox (- c) c ∩ T)
  by (simp add: affine_imp_convex assms(4) convex_Int)
then show continuous_on (T - K ∩ cbox (- (b + One)) (b + One)) (g ∘ closest_point (cbox (-c) c ∩ T))
  using cloTK clo_cT cT_ne
  by (intro continuous_on_compose continuous_on_closest_point continuous_on_subset [OF contg]; force)
have g (closest_point (cbox (- c) c ∩ T) x) ∈ rel_frontier U
  if x ∈ T x ∈ K  $\longrightarrow$  x ∉ cbox (- b - One) (b + One) for x
  using cloTK gim that by auto
then show (g ∘ closest_point (cbox (- c) c ∩ T)) ∈ (T - K ∩ cbox (- (b + One)) (b + One))
   $\longrightarrow$  rel_frontier U
  by force
show  $\bigwedge x. x \in S \implies (g \circ \text{closest\_point } (\text{cbox } (- c) c \cap T)) x = f x$ 
  by simp (metis (mono_tags, lifting) IntI ⟨S ⊆ T⟩ cT_ne clo_cT closest_point_refl gf_subsetD S_sub_cc)
qed
qed

```

corollary extend_map_affine_to_sphere_cofinite:

fixes f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space

assumes SUT: compact S affine T S \subseteq T

and aff: aff_dim T \leq DIM('b) **and** 0 \leq r

and contf: continuous_on S f

and fim: f ∈ S \rightarrow sphere a r

and $dis: \bigwedge C. \llbracket C \in \text{components}(T - S); \text{bounded } C \rrbracket \implies C \cap L \neq \{\}$
obtains $K\ g$ **where** $finite\ K\ K \subseteq L\ K \subseteq T\ \text{disjnt } K\ S\ \text{continuous_on } (T - K)$
 g
 $g \in (T - K) \rightarrow \text{sphere } a\ r \bigwedge x. x \in S \implies g\ x = f\ x$
proof (*cases* $r = 0$)
case *True*
with *fm* **show** *?thesis*
by (*rule_tac* $K = \{\}$ **and** $g = \lambda x. a$ **in** *that*) (*auto*)
next
case *False*
show *thesis*
proof (*rule* *extend_map_affine_to_sphere_cofinite_gen*
 $[OF\ \langle compact\ S \rangle\ \text{convex_cball}\ \text{bounded_cball}\ \langle affine\ T \rangle\ \langle S \subseteq T \rangle\ _contf]$)
have $0 < r$
using *assms* *False* **by** *auto*
then **show** $aff_dim\ T \leq aff_dim\ (\text{cball } a\ r)$
by (*simp* *add: aff_dim_cball*)
show $f \in S \rightarrow \text{rel_frontier } (\text{cball } a\ r)$
by (*simp* *add: False fm*)
qed (*use* *dis* *False* **that** **in** *auto*)
qed

corollary *extend_map_UNIV_to_sphere_cofinite*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes $DIM('a) \leq DIM('b)$ **and** $0 \leq r$
and *compact* S
and *continuous_on* $S\ f$
and $f \in S \rightarrow \text{sphere } a\ r$
and $\bigwedge C. \llbracket C \in \text{components}(- S); \text{bounded } C \rrbracket \implies C \cap L \neq \{\}$
obtains $K\ g$ **where** $finite\ K\ K \subseteq L\ \text{disjnt } K\ S\ \text{continuous_on } (- K)\ g$
 $g \in (- K) \rightarrow \text{sphere } a\ r \bigwedge x. x \in S \implies g\ x = f\ x$
using *assms* *extend_map_affine_to_sphere_cofinite* [*OF* $\langle compact\ S \rangle$ *affine_UNIV*
subset_UNIV]
by (*metis* *Compl_eq_Diff_UNIV* *aff_dim_UNIV* *of_nat_le_iff*)

corollary *extend_map_UNIV_to_sphere_no_bounded_component*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *aff*: $DIM('a) \leq DIM('b)$ **and** $0 \leq r$
and *SUT*: *compact* S
and *contf*: *continuous_on* $S\ f$
and *fm*: $f \in S \rightarrow \text{sphere } a\ r$
and *dis*: $\bigwedge C. C \in \text{components}(- S) \implies \neg \text{bounded } C$
obtains g **where** *continuous_on* $UNIV\ g\ g \in UNIV \rightarrow \text{sphere } a\ r \bigwedge x. x \in S$
 $\implies g\ x = f\ x$
using *extend_map_UNIV_to_sphere_cofinite* [*OF* *aff* $\langle 0 \leq r \rangle\ \langle compact\ S \rangle$ *contf*
fm, *of* $\{\}$]
by (*metis* *Compl_empty_eq* *dis_subset_empty*)

theorem *Borsuk_separation_theorem_gen*:

fixes $S :: 'a::\text{euclidean_space set}$

assumes *compact S*

shows $(\forall c \in \text{components}(- S). \neg \text{bounded } c) \longleftrightarrow$

$(\forall f. \text{continuous_on } S f \wedge f \in S \rightarrow \text{sphere } (0::'a) 1$

$\rightarrow (\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{sphere } 0 1) f (\lambda x.$

$c)))$

(is ?lhs = ?rhs)

proof

assume L [*rule_format*]: *?lhs*

show *?rhs*

proof *clarify*

fix $f :: 'a \Rightarrow 'a$

assume *contf*: *continuous_on S f* **and** *fm*: $f \in S \rightarrow \text{sphere } 0 1$

obtain g **where** *contg*: *continuous_on UNIV g* **and** *gim*: $g \in \text{UNIV} \rightarrow \text{sphere } 0 1$

and *gf*: $\bigwedge x. x \in S \implies g x = f x$

by (*rule extend_map_UNIV_to_sphere_no_bounded_component [OF _ _ <compact S> contf fm L]*) *auto*

then obtain c **where** c : *homotopic_with_canon* $(\lambda h. \text{True}) \text{UNIV } (\text{sphere } 0 1) g (\lambda x. c)$

by (*metis contractible_UNIV nullhomotopic_from_contractible*)

then show $\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{sphere } 0 1) f (\lambda x. c)$

by (*metis assms compact_imp_closed contf contg contractible_empty fm gf gim nullhomotopic_from_contractible nullhomotopic_into_sphere_extension image_subset_iff_funcset*)

qed

next

assume R [*rule_format*]: *?rhs*

show *?lhs*

unfolding *components_def*

proof *clarify*

fix a

assume $a \notin S$ **and** a : *bounded (connected_component_set (- S) a)*

have $\forall x \in S. \text{norm } (x - a) \neq 0$

using $\langle a \notin S \rangle$ **by** *auto*

then have *cont*: *continuous_on S* $(\lambda x. \text{inverse}(\text{norm}(x - a)) *_{\mathbb{R}} (x - a))$

by (*intro continuous_intros*)

have *im*: $(\lambda x. \text{inverse}(\text{norm}(x - a)) *_{\mathbb{R}} (x - a)) ' S \subseteq \text{sphere } 0 1$

by *clarsimp* (*metis* $\langle a \notin S \rangle \text{eq_iff_diff_eq_0 left_inverse norm_eq_zero}$)

show *False*

using R *cont im Borsuk_map_essential_bounded_component [OF <compact S> <a \notin S>] a* **by** *blast*

qed

qed

corollary *Borsuk_separation_theorem*:

fixes $S :: 'a::\text{euclidean_space set}$

3600

```

assumes compact S and 2: 2 ≤ DIM('a)
shows connected(- S) ↔
  (∀ f. continuous_on S f ∧ f ∈ S → sphere (0::'a) 1
    → (∃ c. homotopic_with_canon (λx. True) S (sphere 0 1) f (λx.
c)))
  (is ?lhs = ?rhs)

```

```

proof
assume L: ?lhs
show ?rhs
proof (cases S = {})
  case True
    then show ?thesis
      using homotopic_with_canon_on_empty by blast
  next
    case False
      then have (∀ c ∈ components (- S). ¬ bounded c)
        by (metis L assms(1) bounded_empty cobounded_imp_unbounded compact_imp_bounded_in_components_maximal_order_refl)
      then show ?thesis
        by (simp add: Borsuk_separation_theorem_gen [OF ⟨compact S⟩])
      qed
  next
    assume R: ?rhs
    then have ∀ c ∈ components (- S). ¬ bounded c ⇒ connected (- S)
      by (metis 2 assms(1) cobounded_has_bounded_component compact_imp_bounded_double_complement)
    with R show ?lhs
      by (simp add: Borsuk_separation_theorem_gen [OF ⟨compact S⟩])
    qed

```

```

lemma homotopy_eqv_separation:
  fixes S :: 'a::euclidean_space set and T :: 'a set
  assumes S homotopy_eqv T and compact S and compact T
  shows connected(- S) ↔ connected(- T)
proof -
  consider DIM('a) = 1 | 2 ≤ DIM('a)
  by (metis DIM_ge_Suc0 One_nat_def Suc_1 dual_order.antisym not_less_eq_eq)
  then show ?thesis
  proof cases
    case 1
      then show ?thesis
        using bounded_connected_Compl_1 compact_imp_bounded homotopy_eqv_empty1 homotopy_eqv_empty2 assms by metis
    next
      case 2
        with assms show ?thesis
          by (simp add: Borsuk_separation_theorem homotopy_eqv_cohomotopic_triviality_null)
        qed
  qed

```

qed

proposition *Jordan_Brouwer_separation:*

fixes $S :: 'a::\text{euclidean_space set}$ and $a::'a$
 assumes $\text{hom}: S$ homeomorphic sphere a r and $0 < r$
 shows $\neg \text{connected}(\neg S)$

proof –

have $\neg \text{sphere } a \ r \cap \text{ball } a \ r \neq \{\}$
 using $\langle 0 < r \rangle$ by (simp add: Int_absorb1 subset_eq)

moreover

have eq: $\neg \text{sphere } a \ r - \text{ball } a \ r = - \text{cball } a \ r$
 by auto

have $\neg \text{cball } a \ r \neq \{\}$

proof –

have $\text{frontier } (\text{cball } a \ r) \neq \{\}$

using $\langle 0 < r \rangle$ by auto

then show ?thesis

by (metis frontier_complement frontier_empty)

qed

with eq have $\neg \text{sphere } a \ r - \text{ball } a \ r \neq \{\}$

by auto

moreover

have $\text{connected } (\neg S) = \text{connected } (\neg \text{sphere } a \ r)$

by (meson hom compact_sphere homeomorphic_compactness homeomorphic_imp_homotopy_eqv
 homotopy_eqv_separation)

ultimately show ?thesis

using $\text{connected_Int_frontier}$ [of $\neg \text{sphere } a \ r \ \text{ball } a \ r$] by (auto simp: $\langle 0 < r \rangle$)

qed

proposition *Jordan_Brouwer_frontier:*

fixes $S :: 'a::\text{euclidean_space set}$ and $a::'a$

assumes $S: S$ homeomorphic sphere a r and $T: T \in \text{components}(\neg S)$ and $2: 2 \leq \text{DIM}('a)$

shows $\text{frontier } T = S$

proof (cases r rule: linorder_cases)

assume $r < 0$

with $S \ T$ show ?thesis by auto

next

assume $r = 0$

with $S \ T \ \text{card_eq_SucD}$ obtain b where $S = \{b\}$

by (auto simp: homeomorphic_finite [of $\{a\} \ S$])

have $\text{components } (\neg \{b\}) = \{-\{b\}\}$

using $T \ \langle S = \{b\} \rangle$ by (auto simp: components_eq_sing_iff connected_punctured_universe
 2)

with T show ?thesis

by (metis $\langle S = \{b\} \rangle \ \text{cball_trivial} \ \text{frontier_cball} \ \text{frontier_complement} \ \text{singletonD} \ \text{sphere_trivial}$)

```

next
  assume  $r > 0$ 
  have compact S
    using homeomorphic_compactness compact_sphere S by blast
  show ?thesis
  proof (rule frontier_minimal_separating_closed)
    show closed S
      using  $\langle \text{compact } S \rangle$  compact_eq_bounded_closed by blast
    show  $\neg$  connected ( $- S$ )
      using Jordan_Brouwer_separation S  $\langle 0 < r \rangle$  by blast
    obtain f g where hom: homeomorphism S (sphere a r) f g
      using S by (auto simp: homeomorphic_def)
    show connected ( $- T$ ) if closed T  $T \subset S$  for T
    proof -
      have  $f \in T \rightarrow \text{sphere } a \ r$ 
        using  $\langle T \subset S \rangle$  hom homeomorphism_image1 by blast
      moreover have  $f ' T \neq \text{sphere } a \ r$ 
        using  $\langle T \subset S \rangle$  hom
      by (metis homeomorphism_image2 homeomorphism_of_subsets order_refl
psubsetE)
      ultimately have  $f ' T \subset \text{sphere } a \ r$  by blast
      then have connected ( $- f ' T$ )
        by (rule psubset_sphere_Compl_connected [OF_  $\langle 0 < r \rangle$  2])
      moreover have compact T
        using  $\langle \text{compact } S \rangle$  bounded_subset compact_eq_bounded_closed that by
blast
      moreover then have compact (f ' T)
        by (meson compact_continuous_image continuous_on_subset hom homeo-
morphism_def psubsetE  $\langle T \subset S \rangle$ )
      moreover have T homotopy_eqv f ' T
        by (meson hom homeomorphic_def homeomorphic_imp_homotopy_eqv
homeomorphism_of_subsets order_refl psubsetE that(2))
      ultimately show ?thesis
        using homotopy_eqv_separation [of T f'T] by blast
    qed
  qed (rule T)
qed

```

proposition *Jordan_Brouwer_nonseparation:*

fixes $S :: 'a::\text{euclidean_space}$ set and $a::'a$

assumes S : S homeomorphic sphere $a \ r$ and $T \subset S$ and 2: $2 \leq \text{DIM}('a)$

shows connected($- T$)

proof -

have *: connected($C \cup (S - T)$) if $C \in \text{components}(- S)$ for C

proof (rule connected_intermediate_closure)

show connected C

using in_components_connected that by auto

have $S = \text{frontier } C$

using 2 Jordan_Brouwer_frontier S that by blast

```

  with closure_subset show  $C \cup (S - T) \subseteq \text{closure } C$ 
  by (auto simp: frontier_def)
qed auto
have components( $- S$ )  $\neq \{\}$ 
by (metis S bounded_empty cobounded_imp_unbounded compact_eq_bounded_closed
compact_sphere
components_eq_empty homeomorphic_compactness)
then have  $- T = (\bigcup C \in \text{components}(- S). C \cup (S - T))$ 
using Union_components [of  $-S$ ]  $\langle T \subset S \rangle$  by auto
moreover have connected ...
using  $\langle T \subset S \rangle$  by (intro connected_Union) (auto simp: *)
ultimately show ?thesis
by simp
qed

```

9.27.5 Invariance of domain and corollaries

lemma invariance_of_domain_ball:

```

fixes f :: 'a  $\Rightarrow$  'a::euclidean_space
assumes contf: continuous_on (cball a r) f and 0 < r
and inj: inj_on f (cball a r)
shows open(f ' ball a r)
proof (cases DIM('a) = 1)
case True
obtain h::'a $\Rightarrow$ real and k
where linear h linear k h ' UNIV = UNIV k ' UNIV = UNIV
 $\wedge x. \text{norm}(h x) = \text{norm } x \wedge x. \text{norm}(k x) = \text{norm } x$ 
and kh:  $\wedge x. k(h x) = x$  and  $\wedge x. h(k x) = x$ 
proof (rule isomorphisms_UNIV_UNIV)
show DIM('a) = DIM(real)
using True by force
qed (metis UNIV_I UNIV_eq_I imageI)
have cont: continuous_on S h continuous_on T k for S T
by (simp_all add:  $\langle \text{linear } h \rangle \langle \text{linear } k \rangle$  linear_continuous_on linear_linear)
have continuous_on (h ' cball a r) (h  $\circ$  f  $\circ$  k)
by (intro continuous_on_compose cont continuous_on_subset [OF contf])
(auto simp: kh)
moreover have is_interval (h ' cball a r)
by (simp add: is_interval_connected_1  $\langle \text{linear } h \rangle$  linear_continuous_on
linear_linear connected_continuous_image)
moreover have inj_on (h  $\circ$  f  $\circ$  k) (h ' cball a r)
using inj by (simp add: inj_on_def) (metis  $\langle \wedge x. k (h x) = x \rangle$ )
ultimately have *:  $\wedge T. [\text{open } T; T \subseteq h ' \text{cball } a r] \implies \text{open } ((h \circ f \circ k) ' T)$ 
using injective_eq_1d_open_map_UNIV by blast
have open ((h  $\circ$  f  $\circ$  k) ' (h ' ball a r))
by (rule *) (auto simp:  $\langle \text{linear } h \rangle \langle \text{range } h = \text{UNIV} \rangle$  open_surjective_linear_image)
then have open ((h  $\circ$  f) ' ball a r)
by (simp add: image_comp  $\langle \wedge x. k (h x) = x \rangle$  cong: image_cong)

```

```

then show ?thesis
  unfolding image_comp [symmetric]
  by (metis open_bijjective_linear_image_eq ‹linear h› kh ‹range h = UNIV›
bijI inj_on_def)
next
  case False
  then have 2: DIM('a) ≥ 2
    by (metis DIM_ge_Suc0 One_nat_def Suc_1 antisym not_less_eq_eq)
  have fmsub: f ' ball a r ⊆ - f ' sphere a r
    using inj by clarsimp (metis inj_onD less_eq_real_def mem_cball order_less_irrefl)
  have hom: f ' sphere a r homeomorphic sphere a r
    by (meson compact_sphere contf continuous_on_subset homeomorphic_compact
homeomorphic_sym inj inj_on_subset sphere_cball)
  then have nconn: ¬ connected (- f ' sphere a r)
    by (rule Jordan_Brouwer_separation) (auto simp: ‹0 < r›)
  have bounded (f ' sphere a r)
    by (meson compact_imp_bounded compact_continuous_image_eq compact_sphere
contf inj sphere_cball)
  then obtain C where C: C ∈ components (- f ' sphere a r) and bounded C
    using cobounded_has_bounded_component [OF nconn] 2 by auto
  moreover have f ' (ball a r) = C
proof
  have C ≠ {}
    by (rule in_components_nonempty [OF C])
  show C ⊆ f ' ball a r
proof (rule ccontr)
  assume nonsub: ¬ C ⊆ f ' ball a r
  have - f ' cball a r ⊆ C
proof (rule components_maximal [OF C])
  have f ' cball a r homeomorphic cball a r
    using compact_cball contf homeomorphic_compact homeomorphic_sym
inj by blast
  then show connected (- f ' cball a r)
    by (auto intro: connected_complement_homeomorphic_convex_compact
2)
  show - f ' cball a r ⊆ - f ' sphere a r
    by auto
  then show C ∩ - f ' cball a r ≠ {}
    using ‹C ≠ {}› in_components_subset [OF C] nonsub
    using image_iff by fastforce
qed
  then have bounded (- f ' cball a r)
    using bounded_subset ‹bounded C› by auto
  then have ¬ bounded (f ' cball a r)
    using cobounded_imp_unbounded by blast
  then show False
    using compact_continuous_image [OF contf] compact_cball compact_imp_bounded
by blast
qed

```



```

with ⟨ $C \neq \{\}$ ⟩ have  $C \cap f^{-1} \text{ball } a \ r \neq \{\}$ 
  by (simp add: inf.absorb_iff1)
then show  $f^{-1} \text{ball } a \ r \subseteq C$ 
  by (metis components_maximal [OF C_fimsub] connected_continuous_image
ball_subset_cball connected_ball contf continuous_on_subset)
qed
moreover have open (−  $f^{-1} \text{sphere } a \ r$ )
  using hom_compact_eq_bounded_closed compact_sphere homeomorphic_compactness
by blast
ultimately show ?thesis
  using open_components by blast
qed

```

Proved by L. E. J. Brouwer (1912)

```

theorem invariance_of_domain:
  fixes  $f :: 'a \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes continuous_on S f open S inj_on f S
  shows open( $f^{-1} S$ )
  unfolding open_subopen [of  $f^{-1} S$ ]
proof clarify
  fix a
  assume  $a \in S$ 
  obtain  $\delta$  where  $\delta > 0$  and  $\delta: \text{cball } a \ \delta \subseteq S$ 
    using ⟨open S⟩ ⟨ $a \in S$ ⟩ open_contains_cball_eq by blast
  show  $\exists T. \text{open } T \wedge f a \in T \wedge T \subseteq f^{-1} S$ 
  proof (intro exI conjI)
    show open ( $f^{-1} (\text{ball } a \ \delta)$ )
      by (meson  $\delta < 0 < \delta$ ) assms continuous_on_subset inj_on_subset invariance_of_domain_ball)
    show  $f a \in f^{-1} \text{ball } a \ \delta$ 
      by (simp add: ⟨ $0 < \delta$ ⟩)
    show  $f^{-1} \text{ball } a \ \delta \subseteq f^{-1} S$ 
      using  $\delta \text{ball\_subset\_cball}$  by blast
  qed
qed

```

```

lemma inv_of_domain_ss0:
  fixes  $f :: 'a \Rightarrow 'a::\text{euclidean\_space}$ 
  assumes contf: continuous_on U f and injf: inj_on f U and fim:  $f \in U \rightarrow S$ 
    and subspace S and dimS:  $\dim S = \text{DIM}('b::\text{euclidean\_space})$ 
    and ope:  $\text{openin } (\text{top\_of\_set } S) U$ 
  shows  $\text{openin } (\text{top\_of\_set } S) (f^{-1} U)$ 
proof −
  have  $U \subseteq S$ 
    using ope  $\text{openin\_imp\_subset}$  by blast
  have (UNIV::'b set) homeomorphic S
    by (simp add: ⟨subspace S⟩ dimS homeomorphic_subspaces)
  then obtain  $h \ k$  where  $\text{homhk}: \text{homeomorphism } (\text{UNIV}::'b \text{ set}) S \ h \ k$ 
    using homeomorphic_def by blast

```

```

have homkh: homeomorphism S (k ' S) k h
  using homkh homeomorphism_image2 homeomorphism_sym by fastforce
have open ((k o f o h) ' k ' U)
proof (rule invariance_of_domain)
  show continuous_on (k ' U) (k o f o h)
  proof (intro continuous_intros)
    show continuous_on (k ' U) h
    by (meson continuous_on_subset [OF homeomorphism_cont1 [OF homkh]])
top_greatest)
  have h ' k ' U ⊆ U
    by (metis ⟨U ⊆ S⟩ dual_order.eq_iff homeomorphism_image2 homeomor-
phism_of_subsets homkh)
  then show continuous_on (h ' k ' U) f
    by (rule continuous_on_subset [OF contf])
  have f ' h ' k ' U ⊆ S
    using ⟨h ' k ' U ⊆ U⟩ fim by blast
  then show continuous_on (f ' h ' k ' U) k
    by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homkh]])
qed
have ope_iff: ⋀ T. open T ⟷ openin (top_of_set (k ' S)) T
  using homkh homeomorphism_image2 open_openin by fastforce
show open (k ' U)
  by (simp add: ope_iff homeomorphism_imp_open_map [OF homkh ope])
show inj_on (k o f o h) (k ' U)
  unfolding inj_on_def
  by (smt (verit, ccfv_threshold) PiE ⟨U ⊆ S⟩ assms(3) comp_apply homeo-
morphism_def homkh imageE inj_on_def injf subset_eq)
qed
moreover
have eq: f ' U = h ' (k o f o h o k) ' U
  unfolding image_comp [symmetric] using ⟨U ⊆ S⟩ fim
  by (metis homeomorphism_image2 homeomorphism_of_subsets homkh homkh
image_subset_iff_funcset top_greatest)
ultimately show ?thesis
  by (metis (no_types, opaque_lifting) homeomorphism_imp_open_map homkh
image_comp open_openin subtopology_UNIV)
qed

lemma inv_of_domain_ss1:
  fixes f :: 'a ⇒ 'a::euclidean_space
  assumes contf: continuous_on U f and injf: inj_on f U and fim: f ∈ U → S
  and subspace S
  and ope: openin (top_of_set S) U
  shows openin (top_of_set S) (f ' U)
proof -
  define S' where S' ≡ {y. ∀ x ∈ S. orthogonal x y}
  have subspace S'
    by (simp add: S'_def subspace_orthogonal_to_vectors)
  define g where g ≡ λz::'a*'a. ((f o fst)z, snd z)

```

```

have openin (top_of_set (S × S')) (g ' (U × S'))
proof (rule inv_of_domain_ss0)
  show continuous_on (U × S') g
    unfolding g_def
    by (auto intro!: continuous_intros continuous_on_compose2 [OF contf con-
tinuous_on_fst])
  show g ∈ (U × S') → S × S'
    using fim by (auto simp: g_def)
  show inj_on g (U × S')
    using injf by (auto simp: g_def inj_on_def)
  show subspace (S × S')
    by (simp add: ⟨subspace S'⟩ ⟨subspace S⟩ subspace_Times)
  show openin (top_of_set (S × S')) (U × S')
    by (simp add: openin_Times [OF ope])
  have dim (S × S') = dim S + dim S'
    by (simp add: ⟨subspace S'⟩ ⟨subspace S⟩ dim_Times)
  also have ... = DIM('a)
    using dim_subspace_orthogonal_to_vectors [OF ⟨subspace S⟩ subspace_UNIV]
    by (simp add: add commute S'_def)
  finally show dim (S × S') = DIM('a) .
qed
moreover have g ' (U × S') = f ' U × S'
  by (auto simp: g_def image_iff)
moreover have 0 ∈ S'
  using ⟨subspace S'⟩ subspace_affine by blast
ultimately show ?thesis
  by (auto simp: openin_Times_eq)
qed

```

corollary invariance_of_domain_subspaces:

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes ope: openin (top_of_set U) S
  and subspace U subspace V and VU: dim V ≤ dim U
  and contf: continuous_on S f and fim: f ∈ S → V
  and injf: inj_on f S
shows openin (top_of_set V) (f ' S)
proof -
  obtain V' where subspace V' V' ⊆ U dim V' = dim V
  using choose_subspace_of_subspace [OF VU]
  by (metis span_eq_iff ⟨subspace U⟩)
  then have V homeomorphic V'
    by (simp add: ⟨subspace V⟩ homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism V V' h k
    using homeomorphic_def by blast
  have eq: f ' S = k ' (h o f) ' S
  proof -
    have k ' h ' f ' S = f ' S
      by (meson equalityD2 fim funcset_image homeomorphism_image2 homeo-

```

```

morphism_of_subsets homhk)
  then show ?thesis
    by (simp add: image_comp)
qed
show ?thesis
  unfolding eq
proof (rule homeomorphism_imp_open_map)
  show homkh: homeomorphism V' V k h
    by (simp add: homeomorphism_symD homkh)
  have hfV': (h ∘ f) ' S ⊆ V'
    using fim homeomorphism_image1 homkh by fastforce
  moreover have openin (top_of_set U) ((h ∘ f) ' S)
  proof (rule inv_of_domain_ss1)
    show continuous_on S (h ∘ f)
      by (meson contf continuous_on_compose continuous_on_subset fim func-
set_image homeomorphism_cont2 homkh)
    show inj_on (h ∘ f) S
      by (smt (verit, ccfv_SIG) Pi_iff comp_apply fim homeomorphism_apply2
homkh inj_on_def injf)
    show h ∘ f ∈ S → U
      using ⟨V' ⊆ U⟩ hfV' by blast
    qed (auto simp: assms)
  ultimately show openin (top_of_set V') ((h ∘ f) ' S)
    using openin_subset_trans ⟨V' ⊆ U⟩ by force
  qed
qed

corollary invariance_of_dimension_subspaces:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and subspace U subspace V
    and contf: continuous_on S f and fim: f ∈ S → V
    and injf: inj_on f S and S ≠ {}
  shows dim U ≤ dim V
proof -
  have False if dim V < dim U
  proof -
    obtain T where subspace T T ⊆ U dim T = dim V
      using choose_subspace_of_subspace [of dim V U]
    by (metis ⟨dim V < dim U⟩ assms(2) order.strict_implies_order span_eq_iff)
    then have V homeomorphic T
      by (simp add: ⟨subspace V⟩ homeomorphic_subspaces)
    then obtain h k where homkh: homeomorphism V T h k
      using homeomorphic_def by blast
    have continuous_on S (h ∘ f)
      by (meson contf continuous_on_compose continuous_on_subset fim homeo-
morphism_def homkh image_subset_iff_funcset)
    moreover have (h ∘ f) ' S ⊆ U
      using ⟨T ⊆ U⟩ fim homeomorphism_image1 homkh by fastforce

```

```

moreover have inj_on (h ∘ f) S
  unfolding inj_on_def
  by (metis Pi_iff comp_apply fim_homeomorphism_def homhk inj_on_def
injf)
ultimately have ope_hf: openin (top_of_set U) ((h ∘ f) ‘ S)
  using invariance_of_domain_subspaces [OF ope ‹subspace U› ‹subspace U›]
by blast
have (h ∘ f) ‘ S ⊆ T
  using fim_homeomorphism_image1 homhk by fastforce
then have dim ((h ∘ f) ‘ S) ≤ dim T
  by (rule dim_subset)
also have dim ((h ∘ f) ‘ S) = dim U
  using ‹S ≠ {}› ‹subspace U›
  by (blast intro: dim_openin ope_hf)
finally show False
  using ‹dim V < dim U› ‹dim T = dim V› by simp
qed
then show ?thesis
  using not_less by blast
qed

```

corollary invariance_of_domain_affine_sets:

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes ope: openin (top_of_set U) S
  and aff: affine U affine V aff_dim V ≤ aff_dim U
  and contf: continuous_on S f and fim: f ∈ S → V
  and injf: inj_on f S
shows openin (top_of_set V) (f ‘ S)
proof (cases S = {})
case True
  then show ?thesis by auto
next
case False
obtain a b where a ∈ S a ∈ U b ∈ V
  using False fim ope openin_contains_cball by fastforce
have openin (top_of_set ((+) (- b) ‘ V)) (((+) (- b) ∘ f ∘ (+) a) ‘ (+) (- a) ‘ S)
proof (rule invariance_of_domain_subspaces)
  show openin (top_of_set ((+) (- a) ‘ U)) ((+) (- a) ‘ S)
  by (metis ope_homeomorphism_imp_open_map_homeomorphism_translation
translation_galois)
  show subspace ((+) (- a) ‘ U)
  by (simp add: ‹a ∈ U› affine_diffs_subspace_subtract ‹affine U› cong:
image_cong_simp)
  show subspace ((+) (- b) ‘ V)
  by (simp add: ‹b ∈ V› affine_diffs_subspace_subtract ‹affine V› cong:
image_cong_simp)
  show dim ((+) (- b) ‘ V) ≤ dim ((+) (- a) ‘ U)
  by (metis ‹a ∈ U› ‹b ∈ V› aff_dim_eq_dim_affine_hull_eq_aff_of_nat_le_iff)

```

```

show continuous_on ((+) (- a) ' S) ((+) (- b) ◦ f ◦ (+) a)
  by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
show (+) (- b) ◦ f ◦ (+) a ∈ (+) (- a) ' S → (+) (- b) ' V
  using fm by auto
show inj_on ((+) (- b) ◦ f ◦ (+) a) ((+) (- a) ' S)
  by (auto simp: inj_on_def) (meson inj_onD injf)
qed
then show ?thesis
  by (metis (no_types, lifting) homeomorphism_imp_open_map homeomor-
phism_translation image_comp translation_galois)
qed

```

corollary *invariance_of_dimension_affine_sets:*

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes ope: openin (top_of_set U) S
  and aff: affine U affine V
  and contf: continuous_on S f and fm: f ∈ S → V
  and injf: inj_on f S and S ≠ {}
shows aff_dim U ≤ aff_dim V
proof -
  obtain a b where a ∈ S a ∈ U b ∈ V
  using ⟨S ≠ {}⟩ fm ope openin_contains_cball by fastforce
  have dim ((+) (- a) ' U) ≤ dim ((+) (- b) ' V)
  proof (rule invariance_of_dimension_subspaces)
    show openin (top_of_set ((+) (- a) ' U)) ((+) (- a) ' S)
    by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
    show subspace ((+) (- a) ' U)
    by (simp add: ⟨a ∈ U⟩ affine_diffs_subspace_subtract ⟨affine U⟩ cong:
image_cong_simp)
    show subspace ((+) (- b) ' V)
    by (simp add: ⟨b ∈ V⟩ affine_diffs_subspace_subtract ⟨affine V⟩ cong:
image_cong_simp)
    show continuous_on ((+) (- a) ' S) ((+) (- b) ◦ f ◦ (+) a)
    by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
    show (+) (- b) ◦ f ◦ (+) a ∈ (+) (- a) ' S → (+) (- b) ' V
    using fm by auto
    show inj_on ((+) (- b) ◦ f ◦ (+) a) ((+) (- a) ' S)
    by (auto simp: inj_on_def) (meson inj_onD injf)
  qed (use ⟨S ≠ {}⟩ in auto)
then show ?thesis
  by (metis ⟨a ∈ U⟩ ⟨b ∈ V⟩ aff_dim_eq_dim affine_hull_eq aff_of_nat_le_iff)
qed

```

corollary *invariance_of_dimension:*

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
assumes contf: continuous_on S f and open S

```

```

    and injf: inj_on f S and S ≠ {}
  shows DIM('a) ≤ DIM('b)
  using invariance_of_dimension_subspaces [of UNIV S UNIV f] assms
  by auto

```

```

corollary continuous_injective_image_subspace_dim_le:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace S subspace T
    and contf: continuous_on S f and fim: f ∈ S → T
    and injf: inj_on f S
  shows dim S ≤ dim T
  using invariance_of_dimension_subspaces [of S S _ f] assms by (auto simp:
subspace_affine)

```

```

lemma invariance_of_dimension_convex_domain:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes convex S
    and contf: continuous_on S f and fim: f ∈ S → affine_hull T
    and injf: inj_on f S
  shows aff_dim S ≤ aff_dim T
proof (cases S = {})
  case True
    then show ?thesis by (simp add: aff_dim_geq)
  next
  case False
    have aff_dim (affine_hull S) ≤ aff_dim (affine_hull T)
    proof (rule invariance_of_dimension_affine_sets)
      show openin (top_of_set (affine_hull S)) (rel_interior S)
        by (simp add: openin_rel_interior)
      show continuous_on (rel_interior S) f
        using contf continuous_on_subset rel_interior_subset by blast
      show f ∈ rel_interior S → affine_hull T
        using fim rel_interior_subset by blast
      show inj_on f (rel_interior S)
        using inj_on_subset injf rel_interior_subset by blast
      show rel_interior S ≠ {}
        by (simp add: False ⟨convex S⟩ rel_interior_eq_empty)
    qed auto
    then show ?thesis
      by simp
  qed

```

```

lemma homeomorphic_convex_sets_le:
  assumes convex S S homeomorphic T
  shows aff_dim S ≤ aff_dim T
proof –
  obtain h k where homhk: homeomorphism S T h k

```

```

    using homeomorphic_def assms by blast
  show ?thesis
  proof (rule invariance_of_dimension_convex_domain [OF <convex S>])
    show continuous_on S h
      using homeomorphism_def homhk by blast
    show  $h \in S \rightarrow \text{affine hull } T$ 
      using homeomorphism_image1 homhk hull_subset by fastforce
    show inj_on h S
      by (meson homeomorphism_apply1 homhk inj_on_inverseI)
  qed
qed

lemma homeomorphic_convex_sets:
  assumes convex S convex T S homeomorphic T
  shows  $\text{aff\_dim } S = \text{aff\_dim } T$ 
  by (meson assms dual_order.antisym homeomorphic_convex_sets_le homeomorphic_sym)

lemma homeomorphic_convex_compact_sets_eq:
  assumes convex S compact S convex T compact T
  shows  $S \text{ homeomorphic } T \longleftrightarrow \text{aff\_dim } S = \text{aff\_dim } T$ 
  by (meson assms homeomorphic_convex_compact_sets homeomorphic_convex_sets)

lemma invariance_of_domain_gen:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes open S continuous_on S f inj_on f S  $\text{DIM}('b) \leq \text{DIM}('a)$ 
  shows open( $f \text{ ` } S$ )
  using invariance_of_domain_subspaces [of UNIV S UNIV f] assms by auto

lemma injective_into_1d_imp_open_map_UNIV:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes open T continuous_on S f inj_on f S  $T \subseteq S$ 
  shows open ( $f \text{ ` } T$ )
  proof -
    have  $\text{DIM}(\text{real}) \leq \text{DIM}('a)$ 
      by simp
    then show ?thesis
      using invariance_of_domain_gen assms continuous_on_subset subset_inj_on
  by metis
qed

lemma continuous_on_inverse_open:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes open S continuous_on S f  $\text{DIM}('b) \leq \text{DIM}('a)$  and gf:  $\bigwedge x. x \in S \implies g(f\ x) = x$ 
  shows continuous_on ( $f \text{ ` } S$ ) g
  proof (clarsimp simp add: continuous_openin_preimage_eq)
    fix T :: 'a set
    assume open T

```



```

have eq:  $f^{-1} S \cap g^{-1} T = f^{-1} (S \cap T)$ 
  by (auto simp: gf)
have open (f-1 S)
  by (rule invariance_of_domain_gen) (use assms inj_on_inverseI in auto)
moreover have open (f-1 (S ∩ T))
  using assms
  by (metis ⟨open T⟩ continuous_on_subset inj_onI inj_on_subset invariance_of_domain_gen openin_open openin_open_eq)
ultimately show openin (top_of_set (f-1 S)) (f-1 S ∩ g-1 T)
  unfolding eq by (auto intro: open_openin_trans)
qed

```

```

lemma invariance_of_domain_homeomorphism:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  obtains g where homeomorphism S (f-1 S) f g
proof
  show homeomorphism S (f-1 S) f (inv_into S f)
  by (simp add: assms continuous_on_inverse_open homeomorphism_def)
qed

```

```

corollary invariance_of_domain_homeomorphic:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  shows S homeomorphic (f-1 S)
  using invariance_of_domain_homeomorphism [OF assms]
  by (meson homeomorphic_def)

```

```

lemma continuous_image_subset_interior:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes continuous_on S f inj_on f S DIM('b) ≤ DIM('a)
  shows f ∈ (interior S) → interior(f-1 S)
proof -
  have open (f-1 interior S)
  using assms
  by (intro invariance_of_domain_gen) (auto simp: subset_inj_on interior_subset continuous_on_subset)
  then show ?thesis
  by (simp add: image_mono interiorI interior_subset)
qed

```

```

lemma homeomorphic_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and dimeq: DIM('a) = DIM('b)
  shows (interior S) homeomorphic (interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g

```

3614

```

assume  $S: \forall x \in S. f x \in T \wedge g (f x) = x$  and  $T: \forall y \in T. g y \in S \wedge f (g y) = y$ 
and  $contf: \text{continuous\_on } S f$  and  $contg: \text{continuous\_on } T g$ 
then have  $fST: f ' S = T$  and  $gTS: g ' T = S$  and  $inj\_on f S$   $inj\_on g T$ 
by (auto simp: inj_on_def intro: rev_image_eqI) metis+
have  $fm: f \in \text{interior } S \rightarrow \text{interior } T$ 
using continuous_image_subset_interior [OF contf <inj_on f S>] dimeq fST
by simp
have  $gm: g \in \text{interior } T \rightarrow \text{interior } S$ 
using continuous_image_subset_interior [OF contg <inj_on g T>] dimeq gTS
by simp
show homeomorphism (interior S) (interior T)  $f g$ 
unfolding homeomorphism_def
proof (intro conjI ballI)
show  $\bigwedge x. x \in \text{interior } S \implies g (f x) = x$ 
by (meson <\forall x \in S. f x \in T \wedge g (f x) = x> subsetD interior_subset)
have  $\text{interior } T \subseteq f ' \text{interior } S$ 
proof
fix  $x$  assume  $x \in \text{interior } T$ 
then have  $g x \in \text{interior } S$ 
using  $gm$  by blast
then show  $x \in f ' \text{interior } S$ 
by (metis T <x \in interior T> image_iff interior_subset subsetCE)
qed
then show  $f ' \text{interior } S = \text{interior } T$ 
using  $fm$  by blast
show continuous_on (interior S)  $f$ 
by (metis interior_subset continuous_on_subset contf)
show  $\bigwedge y. y \in \text{interior } T \implies f (g y) = y$ 
by (meson T subsetD interior_subset)
have  $\text{interior } S \subseteq g ' \text{interior } T$ 
proof
fix  $x$  assume  $x \in \text{interior } S$ 
then have  $f x \in \text{interior } T$ 
using  $fm$  by blast
then show  $x \in g ' \text{interior } T$ 
by (metis S <x \in interior S> image_iff interior_subset subsetCE)
qed
then show  $g ' \text{interior } T = \text{interior } S$ 
using  $gm$  by blast
show continuous_on (interior T)  $g$ 
by (metis interior_subset continuous_on_subset contg)
qed
qed

```

lemma *homeomorphic_open_imp_same_dimension*:

```

fixes  $S :: 'a::\text{euclidean\_space set}$  and  $T :: 'b::\text{euclidean\_space set}$ 
assumes  $S$  homeomorphic  $T$  open S  $S \neq \{\}$  open T  $T \neq \{\}$ 
shows  $\text{DIM}('a) = \text{DIM}('b)$ 
using assms

```

```

apply (simp add: homeomorphic_minimal)
apply (rule order_antisym; metis inj_onI invariance_of_dimension)
done

```

proposition *homeomorphic_interiors:*

```

fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $S\ homeomorphic\ T$   $interior\ S = \{\}$   $\longleftrightarrow$   $interior\ T = \{\}$ 
shows  $(interior\ S)\ homeomorphic\ (interior\ T)$ 
proof (cases interior  $T = \{\}$ )
  case False
    then have  $DIM('a) = DIM('b)$ 
      using assms
      apply (simp add: homeomorphic_minimal)
      apply (rule order_antisym; metis continuous_on_subset inj_onI inj_on_subset
interior_subset invariance_of_dimension open_interior)
    done
    then show ?thesis
      by (rule homeomorphic_interiors_same_dimension [OF  $\langle S\ homeomorphic\ T \rangle$ ])
qed (use assms in auto)

```

lemma *homeomorphic_frontiers_same_dimension:*

```

fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
assumes  $S\ homeomorphic\ T$   $closed\ S\ closed\ T$  and dimeq:  $DIM('a) = DIM('b)$ 
shows  $(frontier\ S)\ homeomorphic\ (frontier\ T)$ 
using assms [unfolded homeomorphic_minimal]
unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix  $f\ g$ 
  assume  $S: \forall x \in S. f\ x \in T \wedge g\ (f\ x) = x$  and  $T: \forall y \in T. g\ y \in S \wedge f\ (g\ y) = y$ 
    and contf: continuous_on  $S\ f$  and contg: continuous_on  $T\ g$ 
  then have  $fST: f\ 'S = T$  and  $gTS: g\ 'T = S$  and inj_on  $f\ S\ inj\_on\ g\ T$ 
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have  $g \in interior\ T \rightarrow interior\ S$ 
    using continuous_image_subset_interior [OF contg  $\langle inj\_on\ g\ T \rangle$ ] dimeq  $gTS$ 
by simp
  then have fm:  $f\ 'frontier\ S \subseteq frontier\ T$ 
    unfolding frontier_def using Pi_mem  $S\ assms$  by fastforce
  have  $f \in interior\ S \rightarrow interior\ T$ 
    using continuous_image_subset_interior [OF contf  $\langle inj\_on\ f\ S \rangle$ ] dimeq  $fST$ 
by simp
  then have gm:  $g\ 'frontier\ T \subseteq frontier\ S$ 
    unfolding frontier_def using Pi_mem  $T\ assms$  by fastforce
  show homeomorphism  $(frontier\ S)\ (frontier\ T)\ f\ g$ 
    unfolding homeomorphism_def
proof (intro conjI ballI)
    show gf:  $\bigwedge x. x \in frontier\ S \implies g\ (f\ x) = x$ 
      by (simp add:  $S\ assms(2)\ frontier\_def$ )
    show fg:  $\bigwedge y. y \in frontier\ T \implies f\ (g\ y) = y$ 
      by (simp add:  $T\ assms(3)\ frontier\_def$ )

```

```

have frontier T ⊆ f ` frontier S
proof
  fix x assume x ∈ frontier T
  then have g x ∈ frontier S
    using gim by blast
  then show x ∈ f ` frontier S
    by (metis fg ⟨x ∈ frontier T⟩ imageI)
qed
then show f ` frontier S = frontier T
  using fim by blast
show continuous_on (frontier S) f
  by (metis Diff_subset assms(2) closure_eq contf continuous_on_subset frontier_def)
have frontier S ⊆ g ` frontier T
proof
  fix x assume x ∈ frontier S
  then have f x ∈ frontier T
    using fim by blast
  then show x ∈ g ` frontier T
    by (metis gf ⟨x ∈ frontier S⟩ imageI)
qed
then show g ` frontier T = frontier S
  using gim by blast
show continuous_on (frontier T) g
  by (metis Diff_subset assms(3) closure_closed contg continuous_on_subset frontier_def)
qed
qed

```

lemma *homeomorphic_frontiers*:

```

fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
assumes S homeomorphic T closed S closed T
      interior S = {} ⟷ interior T = {}
shows (frontier S) homeomorphic (frontier T)
proof (cases interior T = {})
case True
  then show ?thesis
    by (metis Diff_empty assms closure_eq frontier_def)
next
case False
  then have DIM('a) = DIM('b)
    using assms homeomorphic_interiors homeomorphic_open_imp_same_dimension
  by blast
  then show ?thesis
    using assms homeomorphic_frontiers_same_dimension by blast
qed

```

lemma *continuous_image_subset_rel_interior*:

```

fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space

```

```

assumes contf: continuous_on S f and inj: inj_on f S and fm:  $f \in S \rightarrow T$ 
and TS: aff_dim T  $\leq$  aff_dim S
shows  $f \in (\text{rel\_interior } S) \rightarrow \text{rel\_interior}(f \text{ ` } S)$ 
unfolding image_subset_iff_funcset [symmetric]
proof (rule rel_interior_maximal)
show  $f \text{ ` } \text{rel\_interior } S \subseteq f \text{ ` } S$ 
by (simp add: image_mono rel_interior_subset)
show openin (top_of_set (affine_hull  $f \text{ ` } S$ )) ( $f \text{ ` } \text{rel\_interior } S$ )
proof (rule invariance_of_domain_affine_sets)
show openin (top_of_set (affine_hull S)) (rel_interior S)
by (simp add: openin_rel_interior)
show aff_dim (affine_hull  $f \text{ ` } S$ )  $\leq$  aff_dim (affine_hull S)
by (metis TS aff_dim_affine_hull aff_dim_subset_fm image_subset_iff_funcset
order_trans)
show  $f \in \text{rel\_interior } S \rightarrow \text{affine\_hull } f \text{ ` } S$ 
using  $\langle f \text{ ` } \text{rel\_interior } S \subseteq f \text{ ` } S \rangle$  hull_subset by fastforce
show continuous_on (rel_interior S) f
using contf continuous_on_subset_rel_interior_subset by blast
show inj_on f (rel_interior S)
using inj_on_subset_inj rel_interior_subset by blast
qed auto
qed

```

lemma *homeomorphic_rel_interiors_same_dimension*:

```

fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
assumes S homeomorphic T and aff: aff_dim S = aff_dim T
shows (rel_interior S) homeomorphic (rel_interior T)
using assms [unfolded homeomorphic_minimal]
unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
fix f g
assume S:  $\forall x \in S. f \ x \in T \wedge g \ (f \ x) = x$  and T:  $\forall y \in T. g \ y \in S \wedge f \ (g \ y) = y$ 
and contf: continuous_on S f and contg: continuous_on T g
then have fST:  $f \text{ ` } S = T$  and gTS:  $g \text{ ` } T = S$  and inj_on f S inj_on g T
by (auto simp: inj_on_def intro: rev_image_eqI) metis+
have fm:  $f \in \text{rel\_interior } S \rightarrow \text{rel\_interior } T$ 
by (smt (verit, best) PiE Pi_I S  $\langle \text{inj\_on } f \ S \rangle$  aff contf continuous_image_subset_rel_interior
fST)
have gm:  $g \in \text{rel\_interior } T \rightarrow \text{rel\_interior } S$ 
by (metis T  $\langle \text{inj\_on } g \ T \rangle$  aff contg continuous_image_subset_rel_interior
dual_order.refl_funcsetI gTS)
show homeomorphism (rel_interior S) (rel_interior T) f g
unfolding homeomorphism_def
proof (intro conjI ballI)
show gf:  $\bigwedge x. x \in \text{rel\_interior } S \implies g \ (f \ x) = x$ 
using S rel_interior_subset by blast
show fg:  $\bigwedge y. y \in \text{rel\_interior } T \implies f \ (g \ y) = y$ 
using T mem_rel_interior_ball by blast
have  $\text{rel\_interior } T \subseteq f \text{ ` } \text{rel\_interior } S$ 

```

```

proof
  fix  $x$  assume  $x \in \text{rel\_interior } T$ 
  then have  $g x \in \text{rel\_interior } S$ 
    using  $gim$  by  $blast$ 
  then show  $x \in f \text{ ' rel\_interior } S$ 
    by ( $metis fg \langle x \in \text{rel\_interior } T \rangle \text{ imageI}$ )
qed
moreover have  $f \text{ ' rel\_interior } S \subseteq \text{rel\_interior } T$ 
  using  $fim$  by  $blast$ 
ultimately show  $f \text{ ' rel\_interior } S = \text{rel\_interior } T$ 
  by  $blast$ 
show  $\text{continuous\_on } (\text{rel\_interior } S) f$ 
  using  $contf \text{ continuous\_on\_subset } \text{rel\_interior\_subset}$  by  $blast$ 
have  $\text{rel\_interior } S \subseteq g \text{ ' rel\_interior } T$ 
proof
  fix  $x$  assume  $x \in \text{rel\_interior } S$ 
  then have  $f x \in \text{rel\_interior } T$ 
    using  $fim$  by  $blast$ 
  then show  $x \in g \text{ ' rel\_interior } T$ 
    by ( $metis gf \langle x \in \text{rel\_interior } S \rangle \text{ imageI}$ )
qed
then show  $g \text{ ' rel\_interior } T = \text{rel\_interior } S$ 
  using  $gim$  by  $blast$ 
show  $\text{continuous\_on } (\text{rel\_interior } T) g$ 
  using  $contg \text{ continuous\_on\_subset } \text{rel\_interior\_subset}$  by  $blast$ 
qed
qed

```

lemma $\text{homeomorphic_aff_dim_le}$:

fixes $S :: 'a::\text{euclidean_space set}$

assumes $S \text{ homeomorphic } T \text{ rel_interior } S \neq \{\}$

shows $\text{aff_dim } (\text{affine hull } S) \leq \text{aff_dim } (\text{affine hull } T)$

proof –

obtain $f g$

where $S: \forall x \in S. f x \in T \wedge g (f x) = x$ **and** $T: \forall y \in T. g y \in S \wedge f (g y) = y$

and $contf: \text{continuous_on } S f$ **and** $contg: \text{continuous_on } T g$

using $assms$ [$\text{unfolded homeomorphic_minimal}$] **by** $auto$

show $?thesis$

proof ($\text{rule invariance_of_dimension_affine_sets}$)

show $\text{continuous_on } (\text{rel_interior } S) f$

using $contf \text{ continuous_on_subset } \text{rel_interior_subset}$ **by** $blast$

show $f \in \text{rel_interior } S \rightarrow \text{affine hull } T$

by ($\text{simp add: } S \text{ hull_inc mem_rel_interior_ball}$)

show $\text{inj_on } f (\text{rel_interior } S)$

by ($metis S \text{ inj_on_inverseI inj_on_subset } \text{rel_interior_subset}$)

qed ($\text{simp_all add: openin_rel_interior } assms$)

qed

```

lemma homeomorphic_rel_interiors:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$   $rel\_interior\ S = \{\} \longleftrightarrow rel\_interior\ T = \{\}$ 
  shows  $(rel\_interior\ S)\ homeomorphic\ (rel\_interior\ T)$ 
proof (cases  $rel\_interior\ T = \{\}$ )
  case True
  with assms show ?thesis by auto
next
  case False
  have  $aff\_dim\ (affine\ hull\ S) \leq aff\_dim\ (affine\ hull\ T)$ 
  using False assms homeomorphic\_aff\_dim\_le by blast
  moreover have  $aff\_dim\ (affine\ hull\ T) \leq aff\_dim\ (affine\ hull\ S)$ 
  using False assms(1) homeomorphic\_aff\_dim\_le homeomorphic\_sym by auto
  ultimately have  $aff\_dim\ S = aff\_dim\ T$  by force
  then show ?thesis
  by (rule homeomorphic\_rel\_interiors\_same\_dimension [OF  $\langle S\ homeomorphic\ T \rangle$ ])
qed

```

```

lemma homeomorphic_rel_boundaries_same_dimension:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$  and  $aff: aff\_dim\ S = aff\_dim\ T$ 
  shows  $(S - rel\_interior\ S)\ homeomorphic\ (T - rel\_interior\ T)$ 
  using assms [unfolded homeomorphic\_minimal]
  unfolding homeomorphic\_def
proof (clarify elim!: ex\_forward)
  fix  $f\ g$ 
  assume  $S: \forall x \in S. f\ x \in T \wedge g\ (f\ x) = x$  and  $T: \forall y \in T. g\ y \in S \wedge f\ (g\ y) = y$ 
  and contf: continuous\_on  $S\ f$  and contg: continuous\_on  $T\ g$ 
  then have  $fST: f\ 'S = T$  and  $gTS: g\ 'T = S$  and inj\_on  $f\ S$  inj\_on  $g\ T$ 
  by (auto simp: inj\_on\_def intro: rev\_image\_eqI) metis+
  have fim:  $f \in rel\_interior\ S \rightarrow rel\_interior\ T$ 
  by (metis  $\langle inj\_on\ f\ S \rangle\ aff\ contf\ continuous\_image\_subset\_rel\_interior\ dual\_order.refl$ 
fST\ image\_subset\_iff\_funcset)
  have gim:  $g \in rel\_interior\ T \rightarrow rel\_interior\ S$ 
  by (metis  $\langle inj\_on\ g\ T \rangle\ aff\ contg\ continuous\_image\_subset\_rel\_interior\ dual\_order.refl$ 
gTS\ image\_subset\_iff\_funcset)
  show homeomorphism  $(S - rel\_interior\ S)\ (T - rel\_interior\ T)\ f\ g$ 
  unfolding homeomorphism\_def
proof (intro conjI ballI)
  show gf:  $\bigwedge x. x \in S - rel\_interior\ S \implies g\ (f\ x) = x$ 
  using  $S\ rel\_interior\_subset$  by blast
  show fg:  $\bigwedge y. y \in T - rel\_interior\ T \implies f\ (g\ y) = y$ 
  using  $T\ mem\_rel\_interior\_ball$  by blast
  show  $f\ '(S - rel\_interior\ S) = T - rel\_interior\ T$ 
  using  $S\ fST\ fim\ gim\ image\_subset\_iff\_funcset$  by fastforce
  show continuous\_on  $(S - rel\_interior\ S)\ f$ 
  using contf\ continuous\_on\_subset\ rel\_interior\_subset by blast

```

3620

```

show  $g \text{ ' } (T - \text{rel\_interior } T) = S - \text{rel\_interior } S$ 
  using  $T \text{ gTS gim fim image\_subset\_iff\_funcset}$  by fastforce
show  $\text{continuous\_on } (T - \text{rel\_interior } T) \text{ g}$ 
  using  $\text{contg continuous\_on\_subset rel\_interior\_subset}$  by blast
qed
qed

```

lemma *homeomorphic_rel_boundaries*:

```

fixes  $S :: 'a::\text{euclidean\_space set}$  and  $T :: 'b::\text{euclidean\_space set}$ 
assumes  $S \text{ homeomorphic } T \text{ rel\_interior } S = \{\} \longleftrightarrow \text{rel\_interior } T = \{\}$ 
shows  $(S - \text{rel\_interior } S) \text{ homeomorphic } (T - \text{rel\_interior } T)$ 
proof ( $\text{cases rel\_interior } T = \{\}$ )
  case True
    with  $\text{assms}$  show ?thesis by auto
  next
    case False
      obtain  $f \text{ g}$ 
        where  $S: \forall x \in S. f \ x \in T \wedge g \ (f \ x) = x$  and  $T: \forall y \in T. g \ y \in S \wedge f \ (g \ y) = y$ 
          and  $\text{contf: continuous\_on } S \ f$  and  $\text{contg: continuous\_on } T \ g$ 
          using  $\text{assms}$  by (auto simp: homeomorphic_minimal)
        have  $\text{aff\_dim } (\text{affine hull } S) \leq \text{aff\_dim } (\text{affine hull } T)$ 
          using False  $\text{assms homeomorphic\_aff\_dim\_le}$  by blast
        moreover have  $\text{aff\_dim } (\text{affine hull } T) \leq \text{aff\_dim } (\text{affine hull } S)$ 
          by (meson False assms(1) homeomorphic\_aff\_dim\_le homeomorphic_sym)
        ultimately have  $\text{aff\_dim } S = \text{aff\_dim } T$  by force
        then show ?thesis
          by (rule homeomorphic_rel_boundaries\_same\_dimension [OF ⟨S homeomor-  
phic T⟩])
      qed

```

proposition *uniformly_continuous_homeomorphism_UNIV_trivial*:

```

fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'a$ 
assumes  $\text{contf: uniformly\_continuous\_on } S \ f$  and  $\text{hom: homeomorphism } S$ 
 $\text{UNIV } f \ g$ 
shows  $S = \text{UNIV}$ 
proof ( $\text{cases } S = \{\}$ )
  case True
    then show ?thesis
      by (metis UNIV_I hom empty_iff homeomorphism_def image_eqI)
  next
    case False
      have  $\text{inj } g$ 
        by (metis UNIV_I hom homeomorphism_apply2 injI)
      then have  $\text{open } (g \text{ ' } \text{UNIV})$ 
        by (blast intro: invariance_of_domain hom homeomorphism_cont2)
      then have  $\text{open } S$ 
        using  $\text{hom homeomorphism\_image2}$  by blast
      moreover have  $\text{complete } S$ 
        unfolding  $\text{complete\_def}$ 

```



```

proof clarify
  fix  $\sigma$ 
  assume  $\sigma: \forall n. \sigma n \in S$  and Cauchy  $\sigma$ 
  have Cauchy  $(f \circ \sigma)$ 
    using uniformly_continuous_imp_Cauchy_continuous  $\langle$ Cauchy  $\sigma$  $\rangle$  contf
    unfolding Cauchy_continuous_on_def by blast
  then obtain  $l$  where  $(f \circ \sigma) \longrightarrow l$ 
    by  $($ auto simp: convergent_eq_Cauchy [symmetric] $)$ 
  show  $\exists l \in S. \sigma \longrightarrow l$ 
proof
  show  $g l \in S$ 
    using hom_homeomorphism_image2 by blast
  have  $(g \circ (f \circ \sigma)) \longrightarrow g l$ 
    by  $($ meson UNIV_I  $\langle$  $(f \circ \sigma) \longrightarrow l$  $\rangle$  continuous_on_sequentially_hom
homeomorphism_cont2 $)$ 
  then show  $\sigma \longrightarrow g l$ 
proof -
  have  $\forall n. \sigma n = (g \circ (f \circ \sigma)) n$ 
    by  $($ metis (no_types)  $\sigma$  comp_eq_dest_lhs hom_homeomorphism_apply1 $)$ 
  then show ?thesis
    by  $($ metis (no_types) LIMSEQ_iff  $\langle$  $(g \circ (f \circ \sigma)) \longrightarrow g l$  $\rangle$  $)$ 
  qed
qed
qed
then have closed  $S$ 
  by  $($ simp add: complete_eq_closed $)$ 
ultimately show ?thesis
  using clopen [of S] False by simp
qed

```

9.27.6 Formulation of loop homotopy in terms of maps out of type complex

```

lemma homotopic_circlemaps_imp_homotopic_loops:
  assumes homotopic_with_canon  $(\lambda h. True)$   $($ sphere  $0\ 1$  $)$   $S\ f\ g$ 
  shows homotopic_loops  $S$   $(f \circ \exp \circ (\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i))$ 
     $(g \circ \exp \circ (\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i))$ 
proof -
  have homotopic_with_canon  $(\lambda f. True)$   $\{z. \text{cmod } z = 1\}$   $S\ f\ g$ 
    using assms by  $($ auto simp: sphere_def $)$ 
  moreover have continuous_on  $\{0..1\}$   $(\exp \circ (\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i))$ 
    by  $($ intro continuous_intros $)$ 
  moreover have  $(\exp \circ (\lambda t. 2 * \text{of\_real } \pi * \text{of\_real } t * i)) \text{ ' } \{0..1\} \subseteq \{z. \text{cmod } z = 1\}$ 
    by  $($ auto simp: norm_mult $)$ 
  ultimately
  show ?thesis
    apply  $($ simp add: homotopic_loops_def comp_assoc $)$ 

```

```

    apply (rule homotopic_with_compose_continuous_right)
    apply (auto simp: pathstart_def pathfinish_def)
  done
qed

lemma homotopic_loops_imp_homotopic_circlemaps:
  assumes homotopic_loops S p q
  shows homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0 1) S
    (p  $\circ$  ( $\lambda z. (\text{Arg}2\pi z / (2 * \pi)$ )))
    (q  $\circ$  ( $\lambda z. (\text{Arg}2\pi z / (2 * \pi)$ )))

proof -
  obtain h where conth: continuous_on ( $\{0..1\} \times \{0..1\}$ ) h
    and him: h  $\in$  ( $\{0..1\} \times \{0..1\}$ )  $\rightarrow$  S
    and h0: ( $\forall x. h (0, x) = p x$ )
    and h1: ( $\forall x. h (1, x) = q x$ )
    and h01: ( $\forall t \in \{0..1\}. h (t, 1) = h (t, 0)$ )

  using assms
  by (auto simp: homotopic_loops_def sphere_def homotopic_with_def pathstart_def pathfinish_def)
  define j where j  $\equiv$   $\lambda z. \text{if } 0 \leq \text{Im } (\text{snd } z) \text{ then } h (\text{fst } z, \text{Arg}2\pi (\text{snd } z) / (2 * \pi)) \text{ else } h (\text{fst } z, 1 - \text{Arg}2\pi (\text{cnj } (\text{snd } z)) / (2 * \pi))$ 
  have Arg2pi_eq:  $1 - \text{Arg}2\pi (\text{cnj } y) / (2 * \pi) = \text{Arg}2\pi y / (2 * \pi) \vee \text{Arg}2\pi y = 0 \wedge \text{Arg}2\pi (\text{cnj } y) = 0$  if cmod y = 1 for y
    using that Arg2pi_eq_0_pi Arg2pi_eq_pi by (force simp: Arg2pi_cnj_field_split_simps)
  show ?thesis
  proof (simp add: homotopic_with; intro conjI ballI exI)
    show continuous_on ( $\{0..1\} \times \text{sphere } 0 \ 1$ ) ( $\lambda w. h (\text{fst } w, \text{Arg}2\pi (\text{snd } w) / (2 * \pi))$ )
  proof (rule continuous_on_eq)
    show j:  $j x = h (\text{fst } x, \text{Arg}2\pi (\text{snd } x) / (2 * \pi))$  if  $x \in \{0..1\} \times \text{sphere } 0 \ 1$ 
    for x
      using Arg2pi_eq that h01 by (force simp: j_def)
    have eq:  $S = S \cap (\text{UNIV} \times \{z. 0 \leq \text{Im } z\}) \cup S \cap (\text{UNIV} \times \{z. \text{Im } z \leq 0\})$  for S :: (real*complex)set
      by auto
    have §:  $\text{Arg}2\pi z \leq 2 * \pi$  for z
      by (simp add: Arg2pi_order_le_less)
    have c1: continuous_on ( $\{0..1\} \times \text{sphere } 0 \ 1 \cap \text{UNIV} \times \{z. 0 \leq \text{Im } z\}$ ) ( $\lambda x. h (\text{fst } x, \text{Arg}2\pi (\text{snd } x) / (2 * \pi))$ )
      apply (intro continuous_intros continuous_on_compose2 [OF conth] continuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
      by (auto simp: Arg2pi §)
    have c2: continuous_on ( $\{0..1\} \times \text{sphere } 0 \ 1 \cap \text{UNIV} \times \{z. \text{Im } z \leq 0\}$ ) ( $\lambda x. h (\text{fst } x, 1 - \text{Arg}2\pi (\text{cnj } (\text{snd } x)) / (2 * \pi))$ )
      apply (intro continuous_intros continuous_on_compose2 [OF conth] continuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
      by (auto simp: Arg2pi §)
    show continuous_on ( $\{0..1\} \times \text{sphere } 0 \ 1$ ) j
  end
end

```

```

apply (simp add: j_def)
apply (subst eq)
apply (rule continuous_on_cases_local)
using Arg2pi_eq h01
  by (force simp add: eq [symmetric] closedin_closed_Int closed_Times
closed_halfspace_Im_le closed_halfspace_Im_ge c1 c2)+
qed
have ( $\lambda w. h (fst w, Arg2pi (snd w) / (2 * pi))$ ) ‘ $(\{0..1\} \times sphere\ 0\ 1) \subseteq h$  ‘
( $\{0..1\} \times \{0..1\}$ )
  by (auto simp: Arg2pi_ge_0 Arg2pi_lt_2pi less_imp_le)
also have  $\dots \subseteq S$ 
  using him by blast
finally show ( $\lambda w. h (fst w, Arg2pi (snd w) / (2 * pi))$ ) ‘ $(\{0..1\} \times sphere\ 0$ 
1)  $\subseteq S$  .
qed (auto simp: h0 h1)
qed

```

lemma simply_connected_homotopic_loops:

```

  simply_connected S  $\longleftrightarrow$ 
  ( $\forall p\ q. homotopic\_loops\ S\ p\ p \wedge homotopic\_loops\ S\ q\ q \longrightarrow homotopic\_loops$ 
S p q)
unfolding simply_connected_def using homotopic_loops_refl by metis

```

lemma simply_connected_eq_homotopic_circlemaps1:

```

fixes f :: complex  $\Rightarrow$  'a::topological_space and g :: complex  $\Rightarrow$  'a
assumes S: simply_connected S
  and contf: continuous_on (sphere 0 1) f and fim: f  $\in$  (sphere 0 1)  $\rightarrow$  S
  and contg: continuous_on (sphere 0 1) g and gim: g  $\in$  (sphere 0 1)  $\rightarrow$  S
shows homotopic_with_canon ( $\lambda h. True$ ) (sphere 0 1) S f g
proof –
  let ?h = ( $\lambda t. complex\_of\_real\ (2 * pi * t) * i$ )
  have homotopic_loops S (f  $\circ$  exp  $\circ$  ?h) (f  $\circ$  exp  $\circ$  ?h)  $\wedge$  homotopic_loops S (g
 $\circ$  exp  $\circ$  ?h) (g  $\circ$  exp  $\circ$  ?h)
  by (simp add: homotopic_circlemaps_imp_homotopic_loops contf fim contg
gim image_subset_iff_funcset)
  then have homotopic_loops S (f  $\circ$  exp  $\circ$  ?h) (g  $\circ$  exp  $\circ$  ?h)
  using S simply_connected_homotopic_loops by blast
  then show ?thesis
  apply (rule homotopic_with_eq [OF homotopic_loops_imp_homotopic_circlemaps])
  apply (auto simp: o_def complex_norm_eq_1_exp mult.commute)
  done
qed

```

lemma simply_connected_eq_homotopic_circlemaps2a:

```

fixes h :: complex  $\Rightarrow$  'a::topological_space
assumes conth: continuous_on (sphere 0 1) h and him: h  $\in$  sphere 0 1  $\rightarrow$  S
and hom:  $\bigwedge f\ g::complex \Rightarrow$  'a.
   $\llbracket$ continuous_on (sphere 0 1) f; f  $\in$  (sphere 0 1)  $\rightarrow$  S;

```

$\text{continuous_on } (\text{sphere } 0 \ 1) \ g; g \in (\text{sphere } 0 \ 1) \rightarrow S$
 $\implies \text{homotopic_with_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ f \ g$
shows $\exists a. \text{homotopic_with_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ h \ (\lambda x. a)$
by (*metis conth continuous_on_const him hom image_subset_iff image_subset_iff_funcset*)

lemma *simply_connected_eq_homotopic_circlemaps2b*:

fixes $S :: 'a::\text{real_normed_vector_set}$

assumes $\bigwedge f g::\text{complex} \Rightarrow 'a.$

$\llbracket \text{continuous_on } (\text{sphere } 0 \ 1) \ f; f \in (\text{sphere } 0 \ 1) \rightarrow S;$

$\text{continuous_on } (\text{sphere } 0 \ 1) \ g; g \in (\text{sphere } 0 \ 1) \rightarrow S \rrbracket$

$\implies \text{homotopic_with_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ f \ g$

shows *path_connected* S

proof (*clarsimp simp add: path_connected_eq_homotopic_points*)

fix $a \ b$

assume $a \in S \ b \in S$

then show *homotopic_loops* $S \ (\text{linepath } a \ a) \ (\text{linepath } b \ b)$

using *homotopic_circlemaps_imp_homotopic_loops* [*OF assms* [*of* $\lambda x. a \ \lambda x.$

b]]

by (*auto simp: o_def linepath_def*)

qed

lemma *simply_connected_eq_homotopic_circlemaps3*:

fixes $h :: \text{complex} \Rightarrow 'a::\text{real_normed_vector}$

assumes *path_connected* S

and *hom*: $\bigwedge f::\text{complex} \Rightarrow 'a.$

$\llbracket \text{continuous_on } (\text{sphere } 0 \ 1) \ f; f \ (\text{sphere } 0 \ 1) \subseteq S \rrbracket$

$\implies \exists a. \text{homotopic_with_canon } (\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S \ f \ (\lambda x. a)$

shows *simply_connected* S

proof (*clarsimp simp add: simply_connected_eq_contractible_loop_some assms*)

fix p

assume $p: \text{path } p \ \text{path_image } p \subseteq S \ \text{pathfinish } p = \text{pathstart } p$

then have *homotopic_loops* $S \ p \ p$

by (*simp add: homotopic_loops_refl*)

then obtain a **where** *homp*: *homotopic_with_canon* $(\lambda h. \text{True}) (\text{sphere } 0 \ 1) \ S$
 $(p \circ (\lambda z. \text{Arg}2\pi \ z / (2 * \pi))) (\lambda x. a)$

by (*metis homotopic_with_imp_subset2 homotopic_loops_imp_homotopic_circlemaps*
homotopic_with_imp_continuous hom)

show $\exists a. a \in S \wedge \text{homotopic_loops } S \ p \ (\text{linepath } a \ a)$

proof (*intro exI conjI*)

show $a \in S$

using *homotopic_with_imp_subset2* [*OF homp*]

by (*metis dist_0_norm image_subset_iff mem_sphere norm_one*)

have *teq*: $\bigwedge t. \llbracket 0 \leq t; t \leq 1 \rrbracket$

$\implies t = \text{Arg}2\pi \ (\exp (2 * \text{of_real } \pi * \text{of_real } t * i)) / (2 * \pi) \vee t=1$

$\wedge \text{Arg}2\pi \ (\exp (2 * \text{of_real } \pi * \text{of_real } t * i)) = 0$

using *Arg2pi_of_real* [*of 1*] **by** (*force simp: Arg2pi_exp*)

have *homotopic_loops* $S \ p \ (p \circ (\lambda z. \text{Arg}2\pi \ z / (2 * \pi))) \circ \exp \circ (\lambda t. 2 * \text{complex_of_real } \pi * \text{complex_of_real } t * i)$

using $p \ \text{teq}$ **by** (*fastforce simp: pathfinish_def pathstart_def intro: homo-*

```

topic_loops_eq [OF p]
  then show homotopic_loops S p (linepath a a)
  by (simp add: linepath_refl homotopic_loops_trans [OF homotopic_circlemaps_imp_homotopic_loops
[OF homp, simplified K_record_comp]])
qed
qed

```

proposition *simply_connected_eq_homotopic_circlemaps:*

```

fixes S :: 'a::real_normed_vector_set
shows simply_connected S  $\longleftrightarrow$ 
  ( $\forall f g::\text{complex} \Rightarrow 'a.$ 
    continuous_on (sphere 0 1) f  $\wedge$  f  $\in$  (sphere 0 1)  $\rightarrow$  S  $\wedge$ 
    continuous_on (sphere 0 1) g  $\wedge$  g  $\in$  (sphere 0 1)  $\rightarrow$  S
     $\rightarrow$  homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0 1) S f g)
by (metis image_subset_iff_funcset simply_connected_eq_homotopic_circlemaps1
simply_connected_eq_homotopic_circlemaps2a
simply_connected_eq_homotopic_circlemaps2b simply_connected_eq_homotopic_circlemaps3)

```

proposition *simply_connected_eq_contractible_circlemap:*

```

fixes S :: 'a::real_normed_vector_set
shows simply_connected S  $\longleftrightarrow$ 
  path_connected S  $\wedge$ 
  ( $\forall f::\text{complex} \Rightarrow 'a.$ 
    continuous_on (sphere 0 1) f  $\wedge$  f '(sphere 0 1)  $\subseteq$  S
     $\rightarrow$  ( $\exists a. \text{homotopic\_with\_canon} (\lambda h. \text{True}) (\text{sphere } 0 \ 1) S f (\lambda x. a))$ )
by (metis image_subset_iff_funcset simply_connected_eq_homotopic_circlemaps
simply_connected_eq_homotopic_circlemaps2a
simply_connected_eq_homotopic_circlemaps3 simply_connected_imp_path_connected)

```

corollary *homotopy_eqv_simple_connectedness:*

```

fixes S :: 'a::real_normed_vector_set and T :: 'b::real_normed_vector_set
shows S homotopy_eqv T  $\implies$  simply_connected S  $\longleftrightarrow$  simply_connected T
by (simp add: simply_connected_eq_homotopic_circlemaps homotopy_eqv_homotopic_triviality
image_subset_iff_funcset)

```

9.27.7 Homeomorphism of simple closed curves to circles

proposition *homeomorphic_simple_path_image_circle:*

```

fixes a :: complex and  $\gamma :: \text{real} \Rightarrow 'a::t2\_space$ 
assumes simple_path  $\gamma$  and loop: pathfinish  $\gamma = \text{pathstart } \gamma$  and  $0 < r$ 
shows (path_image  $\gamma$ ) homeomorphic sphere a r

```

proof –

```

have homotopic_loops (path_image  $\gamma$ )  $\gamma$   $\gamma$ 
  by (simp add: assms homotopic_loops_refl simple_path_imp_path)
then have hom: homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0 1) (path_image
 $\gamma$ )
  ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ )
by (rule homotopic_loops_imp_homotopic_circlemaps)

```

```

have  $\exists g.$  homeomorphism (sphere 0 1) (path_image  $\gamma$ ) ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) g
proof (rule homeomorphism_compact)
  show continuous_on (sphere 0 1) ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ )
    using hom homotopic_with_imp_continuous by blast
  show inj_on ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) (sphere 0 1)
proof
  fix x y
  assume xy: x  $\in$  sphere 0 1 y  $\in$  sphere 0 1
    and eq: ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) x = ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) y
  then have ( $\text{Arg}2\pi x / (2*\pi)$ ) = ( $\text{Arg}2\pi y / (2*\pi)$ )
  proof -
    have ( $\text{Arg}2\pi x / (2*\pi)$ )  $\in$  {0..1} ( $\text{Arg}2\pi y / (2*\pi)$ )  $\in$  {0..1}
      using Arg2pi_ge_0 Arg2pi_lt_2pi dual_order.strict_iff_order by fastforce+
    with eq show ?thesis
      using  $\langle$ simple_path  $\gamma\rangle$  Arg2pi_lt_2pi
      unfolding simple_path_def loop_free_def o_def
      by (metis eq_divide_eq_1 not_less_iff_gr_or_eq)
    qed
  with xy show x = y
    by (metis is_Arg_def Arg2pi Arg2pi_0 dist_0_norm divide_cancel_right dual_order.strict_iff_order mem_sphere)
  qed
  have  $\bigwedge z. \text{cmod } z = 1 \implies \exists x \in \{0..1\}. \gamma (\text{Arg}2\pi z / (2*\pi)) = \gamma x$ 
    by (metis Arg2pi_ge_0 Arg2pi_lt_2pi atLeastAtMost_iff divide_less_eq_1 less_eq_real_def zero_less_mult_iff pi_gt_zero zero_le_divide_iff zero_less_numeral)
  moreover have  $\exists z \in \text{sphere } 0 \ 1. \gamma x = \gamma (\text{Arg}2\pi z / (2*\pi))$  if  $0 \leq x \leq 1$ 
for x
  proof (cases x=1)
    case True
      with Arg2pi_of_real [of 1] loop show ?thesis
      by (rule_tac x=1 in bexI) (auto simp: pathfinish_def pathstart_def  $\langle 0 \leq x \rangle$ )
    next
      case False
      then have *: ( $\text{Arg}2\pi (\exp (i*(2* \text{of\_real } \pi * \text{of\_real } x))) / (2*\pi)$ ) = x
        using that by (auto simp: Arg2pi_exp field_split_simps)
      show ?thesis
      by (rule_tac x=exp(i * of_real(2*\pi*x)) in bexI) (auto simp: *)
    qed
  ultimately show ( $\gamma \circ (\lambda z. \text{Arg}2\pi z / (2*\pi))$ ) ' sphere 0 1 = path_image  $\gamma$ 
    by (auto simp: path_image_def image_iff)
  qed auto
  then have path_image  $\gamma$  homeomorphic sphere (0::complex) 1
    using homeomorphic_def homeomorphic_sym by blast
  also have ... homeomorphic sphere a r
    by (simp add: assms homeomorphic_spheres)
  finally show ?thesis .

```

qed

lemma *homeomorphic_simple_path_images*:

fixes $\gamma 1 :: \text{real} \Rightarrow 'a::t2_space$ **and** $\gamma 2 :: \text{real} \Rightarrow 'b::t2_space$
assumes *simple_path* $\gamma 1$ **and** *loop*: *pathfinish* $\gamma 1 = \text{pathstart } \gamma 1$
assumes *simple_path* $\gamma 2$ **and** *loop*: *pathfinish* $\gamma 2 = \text{pathstart } \gamma 2$
shows (*path_image* $\gamma 1$) *homeomorphic* (*path_image* $\gamma 2$)
by (*meson* *assms* *homeomorphic_simple_path_image_circle* *homeomorphic_sym*
homeomorphic_trans *loop* *pi_gt_zero*)

9.27.8 Dimension-based conditions for various homeomorphisms

lemma *homeomorphic_subspaces_eq*:

fixes $S :: 'a::euclidean_space$ *set* **and** $T :: 'b::euclidean_space$ *set*
assumes *subspace* S *subspace* T
shows S *homeomorphic* $T \longleftrightarrow \dim S = \dim T$

proof

assume S *homeomorphic* T
then obtain $f g$ **where** *hom*: *homeomorphism* $S T f g$
using *homeomorphic_def* **by** *blast*
show $\dim S = \dim T$
by (*metis* $\langle S$ *homeomorphic* $T \rangle$ *aff_dim_subspace* *assms* *homeomorphic_convex_sets*
of_nat_eq_iff *subspace_imp_convex*)
next
assume $\dim S = \dim T$
then show S *homeomorphic* T
by (*simp* *add*: *assms* *homeomorphic_subspaces*)

qed

lemma *homeomorphic_affine_sets_eq*:

fixes $S :: 'a::euclidean_space$ *set* **and** $T :: 'b::euclidean_space$ *set*
assumes *affine* S *affine* T
shows S *homeomorphic* $T \longleftrightarrow \text{aff_dim } S = \text{aff_dim } T$

proof (*cases* $S = \{\}$ \vee $T = \{\}$)

case *True*

then show *?thesis*
using *assms* *homeomorphic_affine_sets* **by** *force*

next

case *False*

then obtain $a b$ **where** $a \in S$ $b \in T$

by *blast*

then have *subspace* $((+) (- a) ' S)$ *subspace* $((+) (- b) ' T)$

using *affine_diffs_subspace* *assms* **by** *blast+*

then show *?thesis*

by (*metis* *affine_imp_convex* *assms* *homeomorphic_affine_sets* *homeomorphic_convex_sets*)

qed

lemma *homeomorphic_hyperplanes_eq*:
fixes $a :: 'a::\text{euclidean_space}$ **and** $c :: 'b::\text{euclidean_space}$
assumes $a \neq 0$ $c \neq 0$
shows $\{x. a \cdot x = b\}$ *homeomorphic* $\{x. c \cdot x = d\} \longleftrightarrow \text{DIM}('a) = \text{DIM}('b)$
proof –
have $\text{DIM}('a) - \text{Suc } 0 = \text{DIM}('b) - \text{Suc } 0 \implies \text{DIM}('a) = \text{DIM}('b)$
by (*metis DIM_positive Suc_pred*)
then show *?thesis*
by (*auto simp: homeomorphic_affine_sets_eq affine_hyperplane assms*)
qed

lemma *homeomorphic_UNIV_UNIV*:
shows $(\text{UNIV}::'a \text{ set})$ *homeomorphic* $(\text{UNIV}::'b \text{ set}) \longleftrightarrow$
 $\text{DIM}('a::\text{euclidean_space}) = \text{DIM}('b::\text{euclidean_space})$
by (*simp add: homeomorphic_subspaces_eq*)

lemma *simply_connected_sphere_gen*:
assumes *convex S bounded S* **and** $3 \leq \text{aff_dim } S$
shows *simply_connected(rel_frontier S)*
proof –
have *pa: path_connected (rel_frontier S)*
using *assms* **by** (*simp add: path_connected_sphere_gen*)
show *?thesis*
proof (*clarsimp simp add: simply_connected_eq contractible_circlemap pa*)
fix f
assume $f: \text{continuous_on } (\text{sphere } (0::\text{complex}) \ 1)$ $f f ' \text{sphere } 0 \ 1 \subseteq \text{rel_frontier } S$
have $\text{eq: sphere } (0::\text{complex}) \ 1 = \text{rel_frontier}(\text{cball } 0 \ 1)$
by *simp*
have *convex (cball (0::complex) 1)*
by (*rule convex_cball*)
then obtain c **where** *homotopic_with_canon* $(\lambda z. \text{True})$ $(\text{sphere } (0::\text{complex}) \ 1)$ $(\text{rel_frontier } S)$ f $(\lambda x. c)$
apply (*rule inessential_spheremap_lowdim_gen [OF__ bounded_cball <convex S> <bounded S>, where f=f]*)
using $f \ 3$ **by** (*auto simp: aff_dim_cball*)
then show $\exists a. \text{homotopic_with_canon } (\lambda h. \text{True})$ $(\text{sphere } 0 \ 1)$ $(\text{rel_frontier } S)$ f $(\lambda x. a)$
by *blast*
qed
qed

9.27.9 more invariance of domain

proposition *invariance_of_domain_sphere_affine_set_gen*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *contf: continuous_on S f* **and** *inj: inj_on f S* **and** *fim: f ∈ S → T*
and *U: bounded U convex U*
and *affine T* **and** *affTU: aff_dim T < aff_dim U*


```

    and ope: openin (top_of_set (rel_frontier U)) S
  shows openin (top_of_set T) (f ' S)
proof (cases rel_frontier U = {})
  case True
  then show ?thesis
    using ope openin_subset by force
next
  case False
  obtain b c where b: b ∈ rel_frontier U and c: c ∈ rel_frontier U and b ≠ c
    using ‹bounded U› rel_frontier_not_sing [of U] subset_singletonD False by
fastforce
  obtain V :: 'a set where affine V and affV: aff_dim V = aff_dim U - 1
  proof (rule choose_affine_subset [OF affine_UNIV])
    show - 1 ≤ aff_dim U - 1
      by (metis aff_dim_empty aff_dim_geq aff_dim_negative_iff affTU diff_0
diff_right_mono not_le)
    show aff_dim U - 1 ≤ aff_dim (UNIV::'a set)
      by (metis aff_dim_UNIV aff_dim_le_DIM le_cases not_le zle_diff1_eq)
  qed auto
  have SU: S ⊆ rel_frontier U
    using ope openin_imp_subset by auto
  have homb: rel_frontier U - {b} homeomorphic V
  and homc: rel_frontier U - {c} homeomorphic V
    using homeomorphic_punctured_sphere_affine_gen [of U _ V]
  by (simp_all add: ‹affine V› affV U b c)
  then obtain g h j k
    where gh: homeomorphism (rel_frontier U - {b}) V g h
      and jk: homeomorphism (rel_frontier U - {c}) V j k
    by (auto simp: homeomorphic_def)
  with SU have hgsub: (h ' g ' (S - {b})) ⊆ S and kjsub: (k ' j ' (S - {c})) ⊆ S
  by (simp_all add: homeomorphism_def subset_eq)
  have [simp]: aff_dim T ≤ aff_dim V
  by (simp add: affTU affV)
  have openin (top_of_set T) ((f ∘ h) ' g ' (S - {b}))
  proof (rule invariance_of_domain_affine_sets [OF _ ‹affine V›])
    have openin (top_of_set (rel_frontier U - {b})) (S - {b})
      by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl)
    then show openin (top_of_set V) (g ' (S - {b}))
      by (rule homeomorphism_imp_open_map [OF gh])
    show continuous_on (g ' (S - {b})) (f ∘ h)
      proof (rule continuous_on_compose)
        show continuous_on (g ' (S - {b})) h
          by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
gh set_eq_subset)
      qed (use contf continuous_on_subset hgsub in blast)
    show inj_on (f ∘ h) (g ' (S - {b}))
      by (smt (verit, del_insts) SU homeomorphism_def inj_on_def injf gh Diff_iff
comp_apply imageE subset_iff)

```

```

    show  $f \circ h \in g^{-1}(S - \{b\}) \rightarrow T$ 
      using fm hgsub by fastforce
  qed (auto simp: assms)
  moreover
  have openin (top_of_set T) ((f ∘ k)-1 (j-1 (S - {c})))
  proof (rule invariance_of_domain_affine_sets [OF_ <affine V>])
    show openin (top_of_set V) (j-1 (S - {c}))
      by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl homeomorphism_imp_open_map [OF jk])
    show continuous_on (j-1 (S - {c})) (f ∘ k)
      proof (rule continuous_on_compose)
        show continuous_on (j-1 (S - {c})) k
          by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
jk set_eq_subset)
        qed (use contf continuous_on_subset kjsub in blast)
      show inj_on (f ∘ k) (j-1 (S - {c}))
        by (smt (verit, del_insts) SU homeomorphism_def inj_on_def injf jk Diff_iff
comp_apply imageE subset_iff)
      show  $f \circ k \in j^{-1}(S - \{c\}) \rightarrow T$ 
        using fm kjsub by fastforce
      qed (auto simp: assms)
    ultimately have openin (top_of_set T) ((f ∘ h)-1 g-1 (S - {b}) ∪ ((f ∘ k)-1 j-1
(S - {c})))
      by (rule openin_Un)
    moreover have (f ∘ h)-1 g-1 (S - {b}) = f-1 (S - {b})
    proof -
      have  $h^{-1} g^{-1} (S - \{b\}) = (S - \{b\})$ 
        by (meson Diff_mono Diff_subset SU gh homeomorphism_def homeomor-
phism_of_subsets subset_singleton_iff)
      then show ?thesis
        by (metis image_comp)
    qed
    moreover have (f ∘ k)-1 j-1 (S - {c}) = f-1 (S - {c})
    proof -
      have  $k^{-1} j^{-1} (S - \{c\}) = (S - \{c\})$ 
        using homeomorphism_apply1 [OF jk] SU
        by (meson Diff_mono homeomorphism_def homeomorphism_of_subsets jk
subset_refl)
      then show ?thesis
        by (metis image_comp)
    qed
    moreover have  $f^{-1}(S - \{b\}) \cup f^{-1}(S - \{c\}) = f^{-1}(S)$ 
      using <b ≠ c> by blast
    ultimately show ?thesis
      by simp
  qed

```

lemma *invariance_of_domain_sphere_affine_set*:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes contf: continuous_on S f and injf: inj_on f S and fim: f  $\in$  S  $\rightarrow$  T
and r  $\neq$  0 affine T and affTU: aff_dim T < DIM('a)
and ope: openin (top_of_set (sphere a r)) S
shows openin (top_of_set T) (f ' S)
proof (cases sphere a r = {})
  case True
  then show ?thesis
    using ope openin_subset by force
  next
  case False
  show ?thesis
  proof (rule invariance_of_domain_sphere_affine_set_gen [OF contf injf fim
bounded_cball convex_cball <affine T>])
    show aff_dim T < aff_dim (cball a r)
      by (metis False affTU aff_dim_cball assms(4) linorder_cases sphere_empty)
    show openin (top_of_set (rel_frontier (cball a r))) S
      by (simp add: <r  $\neq$  0> ope)
  qed
qed

```

lemma no_embedding_sphere_lowdim:

```

fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
assumes contf: continuous_on (sphere a r) f and injf: inj_on f (sphere a r)
and r > 0
shows DIM('a)  $\leq$  DIM('b)
proof -
  have False if DIM('a) > DIM('b)
  proof -
    have compact (f ' sphere a r)
      using compact_continuous_image
      by (simp add: compact_continuous_image contf)
    then have  $\neg$  open (f ' sphere a r)
      using compact_open
    by (metis assms(3) image_is_empty_not_less_iff_gr_or_eq sphere_eq_empty)
    then have r=0
      by (metis Pi_I UNIV_I aff_dim_UNIV affine_UNIV contf injf invariance_of_domain_sphere_affine_set
of_nat_less_iff open_openin openin_subtopology_self subtopology_UNIV that)
    with <r>0 show False by auto
  qed
  then show ?thesis
    using not_less by blast
qed

```

lemma simply_connected_sphere:

```

fixes a :: 'a::euclidean_space
assumes 3  $\leq$  DIM('a)

```

3632

```
    shows simply_connected(sphere a r)
  proof (cases rule: linorder_cases [of r 0])
    case less
    then show ?thesis by simp
  next
    case equal
    then show ?thesis by (auto simp: convex_imp_simply_connected)
  next
    case greater
    then show ?thesis
      using simply_connected_sphere_gen [of cball a r] assms
      by (simp add: aff_dim_cball)
qed
```

lemma simply_connected_sphere_eq:

```
  fixes a :: 'a::euclidean_space
  shows simply_connected(sphere a r)  $\longleftrightarrow$   $3 \leq \text{DIM}('a) \vee r \leq 0$  (is ?lhs =
  ?rhs)
  proof (cases r  $\leq$  0)
    case True
    have simply_connected (sphere a r)
      using True less_eq_real_def by (auto intro: convex_imp_simply_connected)
    with True show ?thesis by auto
  next
    case False
    show ?thesis
    proof
      assume L: ?lhs
      have False if DIM('a) = 1  $\vee$  DIM('a) = 2
        using that
      proof
        assume DIM('a) = 1
        with L show False
          using connected_sphere_eq_simply_connected_imp_connected
          by (metis False Suc_1 not_less_eq_eq order_refl)
      next
        assume DIM('a) = 2
        then have sphere a r homeomorphic sphere (0::complex) 1
          by (metis DIM_complex False homeomorphic_spheres_gen not_less zero_less_one)
        then have simply_connected(sphere (0::complex) 1)
          using L homeomorphic_simply_connected_eq by blast
        then obtain a::complex where homotopic_with_canon ( $\lambda h. \text{True}$ ) (sphere 0
        1) (sphere 0 1) id ( $\lambda x. a$ )
          by (metis continuous_on_id' id_apply image_id subset_refl simply_connected_eq_contractible_cin)
        then show False
          using contractible_sphere contractible_def not_one_le_zero by blast
      qed
    with False show ?rhs
    by (metis DIM_ge_Suc0 One_nat_def Suc_1 not_less_eq_eq numeral_3_eq_3)
```

```

order_antisym_conv)
next
  assume ?rhs
  with False show ?lhs by (simp add: simply_connected_sphere)
qed
qed

```

```

lemma simply_connected_punctured_universe_eq:
  fixes a :: 'a::euclidean_space
  shows simply_connected(- {a})  $\longleftrightarrow$   $3 \leq \text{DIM}('a)$ 
proof -
  have [simp]:  $a \in \text{rel\_interior}(\text{cball } a \ 1)$ 
  by (simp add: rel_interior_nonempty_interior)
  have [simp]:  $\text{affine\_hull } \text{cball } a \ 1 - \{a\} = -\{a\}$ 
  by (metis Compl_eq_Diff_UNIV aff_dim_cball aff_dim_lt_full_not_less_iff_gr_or_eq
  zero_less_one)
  have sphere_a_1_homotopy_eqv - {a}
  using homotopy_eqv_rel_frontier_punctured_affine_hull [of cball a 1 a] by
  auto
  then have simply_connected(- {a})  $\longleftrightarrow$  simply_connected(sphere a 1)
  using homotopy_eqv_simple_connectedness by blast
  also have ...  $\longleftrightarrow$   $3 \leq \text{DIM}('a)$ 
  by (simp add: simply_connected_sphere_eq)
  finally show ?thesis .
qed

```

```

lemma not_simply_connected_circle:
  fixes a :: complex
  shows  $0 < r \implies \neg \text{simply\_connected}(\text{sphere } a \ r)$ 
by (simp add: simply_connected_sphere_eq)

```

```

proposition simply_connected_punctured_convex:
  fixes a :: 'a::euclidean_space
  assumes convex S and  $\exists: 3 \leq \text{aff\_dim } S$ 
  shows simply_connected(S - {a})
proof (cases  $a \in \text{rel\_interior } S$ )
case True
  then obtain e where  $a \in S$   $0 < e$  and  $e: \text{cball } a \ e \cap \text{affine\_hull } S \subseteq S$ 
  by (auto simp: rel_interior_cball)
  have con: convex (cball a e  $\cap$  affine_hull S)
  by (simp add: convex_Int)
  have bo: bounded (cball a e  $\cap$  affine_hull S)
  by (simp add: bounded_Int)
  have affine_hull_S  $\cap$  interior (cball a e)  $\neq \{\}$ 
  using  $\langle 0 < e \rangle \langle a \in S \rangle$  hull_subset by fastforce
  then have  $3 \leq \text{aff\_dim}(\text{affine\_hull } S \cap \text{cball } a \ e)$ 
  by (simp add:  $\exists \text{ aff\_dim\_convex\_Int\_nonempty\_interior}$  [OF convex_affine_hull])

```

```

also have ... = aff_dim (cball a e ∩ affine hull S)
  by (simp add: Int_commute)
finally have 3 ≤ aff_dim (cball a e ∩ affine hull S) .
moreover have rel_frontier (cball a e ∩ affine hull S) homotopy_eqv S - {a}
proof (rule homotopy_eqv_rel_frontier_punctured_convex)
  show a ∈ rel_interior (cball a e ∩ affine hull S)
    by (meson IntI Int_mono ⟨a ∈ S⟩ ⟨0 < e⟩ e ⟨cball a e ∩ affine hull S
  ⊆ S⟩ ball_subset_cball centre_in_cball dual_order.strict_implies_order hull_inc
  hull_mono mem_rel_interior_ball)
  have closed (cball a e ∩ affine hull S)
    by blast
  then show rel_frontier (cball a e ∩ affine hull S) ⊆ S
    by (metis Diff_subset closure_closed dual_order.trans e rel_frontier_def)
  show S ⊆ affine hull (cball a e ∩ affine hull S)
    by (metis (no_types, lifting) IntI ⟨a ∈ S⟩ ⟨0 < e⟩ affine_hull_convex_Int_nonempty_interior
  centre_in_ball convex_affine_hull_empty_iff_hull_subset_inf_commute interior_cball
  subsetCE subsetI)
  qed (auto simp: assms con bo)
ultimately show ?thesis
  using homotopy_eqv_simple_connectedness simply_connected_sphere_gen [OF
  con bo]
  by blast
next
case False
then have rel_interior S ⊆ S - {a}
  by (simp add: False rel_interior_subset subset_Diff_insert)
moreover have S - {a} ⊆ closure S
  by (meson Diff_subset closure_subset subset_trans)
ultimately show ?thesis
  by (metis contractible_imp_simply_connected contractible_convex_tweak_boundary_points
  [OF ⟨convex S⟩])
qed

corollary simply_connected_punctured_universe:
  fixes a :: 'a::euclidean_space
  assumes 3 ≤ DIM('a)
  shows simply_connected(- {a})
proof -
  have [simp]: affine hull cball a 1 = UNIV
    by (simp add: aff_dim_cball affine_hull_UNIV)
  have a ∈ rel_interior (cball a 1)
    by (simp add: rel_interior_interior)
  then
  have simply_connected (rel_frontier (cball a 1)) = simply_connected (affine hull
  cball a 1 - {a})
    using homotopy_eqv_rel_frontier_punctured_affine_hull homotopy_eqv_simple_connectedness
  by blast
  then show ?thesis
    using simply_connected_sphere [of a 1, OF assms] by (auto simp: Compl_eq_Diff_UNIV)

```

qed

9.27.10 The power, squaring and exponential functions as covering maps

proposition *covering_space_power_punctured_plane:*

assumes $0 < n$

shows *covering_space* $(-\{0\})$ $(\lambda z::\text{complex}. z^n)$ $(-\{0\})$

proof –

consider $n = 1 \mid 2 \leq n$ using *assms* by *linarith*

then obtain e where $0 < e$

and $e: \bigwedge w z. \text{cmod}(w - z) < e * \text{cmod } z \implies (w^n = z^n \iff w = z)$

proof *cases*

assume $n = 1$ then show *?thesis*

by (*rule_tac* $e=1$ in *that*) *auto*

next

assume $2 \leq n$

have *eq_if_pow_eq*:

$w = z$ if *lt*: $\text{cmod}(w - z) < 2 * \sin(\pi / \text{real } n) * \text{cmod } z$

and *eq*: $w^n = z^n$ for $w z$

proof (*cases* $z = 0$)

case *True* with *eq* *assms* show *?thesis* by (*auto simp: power_0_left*)

next

case *False*

then have $z \neq 0$ by *auto*

have $(w/z)^n = 1$

by (*metis False divide_self_if eq power_divide power_one*)

then obtain $j::\text{nat}$ where $j: w / z = \exp(2 * \text{of_real } \pi * i * j / n)$ and $j < n$

using *Suc_leI* *assms* $\langle 2 \leq n \rangle$ *complex_roots_unity* [*THEN eqset_imp_iff*, *of n w/z*]

by *force*

have $\text{cmod}(w/z - 1) < 2 * \sin(\pi / \text{real } n)$

using *lt* *assms* $\langle z \neq 0 \rangle$ by (*simp add: field_split_simps norm_divide*)

then have $\text{cmod}(\exp(i * \text{of_real}(2 * \pi * j / n)) - 1) < 2 * \sin(\pi / \text{real } n)$

by (*simp add: j field_simps*)

then have $2 * |\sin((2 * \pi * j / n) / 2)| < 2 * \sin(\pi / \text{real } n)$

by (*simp only: dist_exp_i_1*)

then have *sin_less*: $\sin(\pi * j / n) < \sin(\pi / \text{real } n)$

by (*simp add: field_simps*)

then have $w / z = 1$

proof (*cases* $j = 0$)

case *True* then show *?thesis* by (*auto simp: j*)

next

case *False*

then have $\sin(\pi / \text{real } n) \leq \sin(\pi * j / n)$

proof (*cases* $j / n \leq 1/2$)

```

      case True
      show ?thesis
      using ⟨j ≠ 0 ⟩ ⟨j < n⟩ True
      by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of_ 0])
    next
    case False
    then have seq: sin(pi * j / n) = sin(pi * (n - j) / n)
      using ⟨j < n⟩ by (simp add: algebra_simps diff_divide_distrib
of_nat_diff)
    show ?thesis
      unfolding seq
      using ⟨j < n⟩ False
      by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of_ 0])
    qed
    with sin_less show ?thesis by force
  qed
  then show ?thesis by simp
qed
show ?thesis
proof
  show 0 < 2 * sin (pi / real n)
    by (force simp: ⟨2 ≤ n⟩ sin_pi_divide_n_gt_0)
  qed (meson eq_if_pow_eq)
qed
have zn1: continuous_on (- {0}) (λz::complex. z^n)
  by (rule continuous_intros)+
have zn2: (λz::complex. z^n) '(- {0}) = - {0}
  using assms by (auto simp: image_def elim: exists_complex_root_nonzero
[where n = n])
have zn3: ∃ T. z^n ∈ T ∧ open T ∧ 0 ∉ T ∧
  (∃ v. ⋃ v = -{0} ∩ (λz. z^n) -' T ∧
  (∀ u ∈ v. open u ∧ 0 ∉ u) ∧
  pairwise disjnt v ∧
  (∀ u ∈ v. ∃ (homeomorphism u T (λz. z^n))))
  if z ≠ 0 for z::complex
proof -
  define d where d ≡ min (1/2) (e/4) * norm z
  have 0 < d
    by (simp add: d_def ⟨0 < e⟩ ⟨z ≠ 0⟩)
  have iff_x_eq_y: x^n = y^n ↔ x = y
    if eq: w^n = z^n and x: x ∈ ball w d and y: y ∈ ball w d for w x y
  proof -
    have [simp]: norm z = norm w using that
      by (simp add: assms power_eq_imp_eq_norm)
    show ?thesis
    proof (cases w = 0)
      case True with ⟨z ≠ 0⟩ assms eq

```



```

    show ?thesis by (auto simp: power_0_left)
  next
    case False
    have cmod (x - y) < 2*d
      using x y
      by (simp add: dist_norm [symmetric]) (metis dist_commute mult_2
dist_triangle_less_add)
    also have ... ≤ 2 * e / 4 * norm w
      using ‹e > 0› by (simp add: d_def min_mult_distrib_right)
    also have ... = e * (cmod w / 2)
      by simp
    also have ... ≤ e * cmod y
    proof (rule mult_left_mono)
      have cmod (w - y) < cmod w / 2 ⇒ cmod w / 2 ≤ cmod y
        by (metis (no_types) dist_0_norm dist_norm norm_triangle_half_l
not_le order_less_irrefl)
      then show cmod w / 2 ≤ cmod y
        using y by (simp add: dist_norm d_def min_mult_distrib_right)
    qed (use ‹e > 0› in auto)
    finally have cmod (x - y) < e * cmod y .
    then show ?thesis by (rule e)
  qed
qed
then have inj: inj_on (λw. w^n) (ball z d)
  by (simp add: inj_on_def)
have cont: continuous_on (ball z d) (λw. w ^ n)
  by (intro continuous_intros)
have noncon: ¬ (λw::complex. w^n) constant_on UNIV
  by (metis UNIV_I assms constant_on_def power_one zero_neq_one zero_power)
have im_eq: (λw. w^n) ' ball z' d = (λw. w^n) ' ball z d
  if z': z'^n = z^n for z'
proof -
  have nz': norm z' = norm z using that assms power_eq_imp_eq_norm by
blast
  have (w ∈ (λw. w^n) ' ball z' d) = (w ∈ (λw. w^n) ' ball z d) for w
  proof (cases w=0)
    case True with assms show ?thesis
      by (simp add: image_def ball_def nz')
  next
    case False
    have z' ≠ 0 using ‹z ≠ 0› nz' by force
    have 1: (z*x / z')^n = x^n if x ≠ 0 for x
      using z' that by (simp add: field_simps ‹z ≠ 0›)
    have 2: cmod (z - z * x / z') = cmod (z' - x) if x ≠ 0 for x
    proof -
      have cmod (z - z * x / z') = cmod z * cmod (1 - x / z')
      by (metis (no_types) ab_semigroup_mult_class.mult_ac(1) divide_complex_def
mult.right_neutral norm_mult_right_diff_distrib')
      also have ... = cmod z' * cmod (1 - x / z')

```

```

      by (simp add: nz')
    also have ... = cmod (z' - x)
      by (simp add: ⟨z' ≠ 0⟩ diff_divide_eq_iff norm_divide)
    finally show ?thesis .
  qed
  have 3: (z'*x / z) ^ n = x ^ n if x ≠ 0 for x
    using z' that by (simp add: field_simps ⟨z ≠ 0⟩)
  have 4: cmod (z' - z' * x / z) = cmod (z - x) if x ≠ 0 for x
  proof -
    have cmod (z * (1 - x * inverse z)) = cmod (z - x)
      by (metis ⟨z ≠ 0⟩ diff_divide_distrib divide_complex_def divide_self_if
nonzero_eq_divide_eq semiring_normalization_rules(7))
    then show ?thesis
      by (metis (no_types) mult.assoc divide_complex_def mult.right_neutral
norm_mult nz' right_diff_distrib')
    qed
    show ?thesis
      by (simp add: set_eq_iff image_def ball_def) (metis 1 2 3 4 diff_zero
dist_norm nz')
    qed
  then show ?thesis by blast
  qed
  have ex_ball: ∃ B. (∃ z'. B = ball z' d ∧ z' ^ n = z ^ n) ∧ x ∈ B
    if x ≠ 0 and eq: x ^ n = w ^ n and dzw: dist z w < d for x w
  proof -
    have w ≠ 0 by (metis assms power_eq_0_iff that(1) that(2))
    have [simp]: cmod x = cmod w
      using assms power_eq_imp_eq_norm eq by blast
    have [simp]: cmod (x * z / w - x) = cmod (z - w)
    proof -
      have cmod (x * z / w - x) = cmod x * cmod (z / w - 1)
        by (metis (no_types) mult.right_neutral norm_mult right_diff_distrib'
times_divide_eq_right)
      also have ... = cmod w * cmod (z / w - 1)
        by simp
      also have ... = cmod (z - w)
        by (simp add: ⟨w ≠ 0⟩ divide_diff_eq_iff nonzero_norm_divide)
      finally show ?thesis .
    qed
    show ?thesis
  proof (intro exI conjI)
    show (z / w * x) ^ n = z ^ n
      by (metis ⟨w ≠ 0⟩ eq_nonzero_eq_divide_eq power_mult_distrib)
    show x ∈ ball (z / w * x) d
      using ⟨d > 0⟩ that
      by (simp add: ball_eq_ball_iff ⟨z ≠ 0⟩ ⟨w ≠ 0⟩ field_simps) (simp add:
dist_norm)
    qed auto

```

qed

show ?thesis

proof (rule exI, intro conjI)

show $z \hat{\ } n \in (\lambda w. w \hat{\ } n) \text{ ' ball } z \ d$

using $\langle d > 0 \rangle$ by simp

show $open ((\lambda w. w \hat{\ } n) \text{ ' ball } z \ d)$

by (rule invariance_of_domain [OF cont open_ball inj])

show $0 \notin (\lambda w. w \hat{\ } n) \text{ ' ball } z \ d$

using $\langle z \neq 0 \rangle$ assms by (force simp: d_def)

show $\exists v. \bigcup v = - \{0\} \cap (\lambda z. z \hat{\ } n) - (\lambda w. w \hat{\ } n) \text{ ' ball } z \ d \wedge$

$(\forall u \in v. open \ u \wedge 0 \notin u) \wedge$

$disjoint \ v \wedge$

$(\forall u \in v. \exists x (homeomorphism \ u \ ((\lambda w. w \hat{\ } n) \text{ ' ball } z \ d) (\lambda z. z \hat{\ } n)))$

proof (rule exI, intro ballI conjI)

show $\bigcup \{ball \ z' \ d \mid z'. z' \hat{\ } n = z \hat{\ } n\} = - \{0\} \cap (\lambda z. z \hat{\ } n) - (\lambda w. w \hat{\ } n) \text{ ' ball } z \ d$ (is ?l = ?r)

proof

have $\bigwedge z'. cmod \ z' < d \implies z' \hat{\ } n \neq z \hat{\ } n$

by (auto simp add: assms d_def power_eq_imp_eq_norm that)

then show ?l \subseteq ?r

by auto (metis im_eq image_eqI mem_ball)

show ?r \subseteq ?l

by auto (meson ex_ball)

qed

show $\bigwedge u. u \in \{ball \ z' \ d \mid z'. z' \hat{\ } n = z \hat{\ } n\} \implies 0 \notin u$

by (force simp add: assms d_def power_eq_imp_eq_norm that)

show $disjoint \ \{ball \ z' \ d \mid z'. z' \hat{\ } n = z \hat{\ } n\}$

proof (clarsimp simp add: pairwise_def disjnt_iff)

fix $\xi \ \zeta \ x$

assume $\xi \hat{\ } n = z \hat{\ } n \ \zeta \hat{\ } n = z \hat{\ } n \ ball \ \xi \ d \neq ball \ \zeta \ d$

and $dist \ \xi \ x < d \ dist \ \zeta \ x < d$

then have $dist \ \xi \ \zeta < d + d$

using $dist_triangle_less_add$ by blast

then have $cmod \ (\xi - \zeta) < 2 * d$

by (simp add: dist_norm)

also have $\dots \leq e * cmod \ z$

using $mult_right_mono \ \langle 0 < e \rangle$ that by (auto simp: d_def)

finally have $cmod \ (\xi - \zeta) < e * cmod \ z$.

with e have $\xi = \zeta$

by (metis $\langle \xi \hat{\ } n = z \hat{\ } n \rangle \langle \zeta \hat{\ } n = z \hat{\ } n \rangle$ assms power_eq_imp_eq_norm)

then show False

using $\langle ball \ \xi \ d \neq ball \ \zeta \ d \rangle$ by blast

qed

show $\exists x (homeomorphism \ u \ ((\lambda w. w \hat{\ } n) \text{ ' ball } z \ d) (\lambda z. z \hat{\ } n))$

if $u \in \{ball \ z' \ d \mid z'. z' \hat{\ } n = z \hat{\ } n\}$ for u

proof (rule invariance_of_domain_homeomorphism [of $u \ \lambda z. z \hat{\ } n$])

show $open \ u$

```

    using that by auto
  show continuous_on u (λz. z ^ n)
  by (intro continuous_intros)
  show inj_on (λz. z ^ n) u
  using that by (auto simp: iff_x_eq_y inj_on_def)
  show ∧g. homeomorphism u ((λz. z ^ n) ' u) (λz. z ^ n) g ⇒ Ex
(homeomorphism u ((λw. w ^ n) ' ball z d) (λz. z ^ n))
  using im_eq that by clarify metis
qed auto
qed auto
qed
qed
show ?thesis
  using assms
  apply (simp add: covering_space_def zn1 zn2)
  apply (subst zn2 [symmetric])
  apply (simp add: openin_open_eq open_Cmpl zn3)
  done
qed

```

corollary *covering_space_square_punctured_plane:*
covering_space (− {0}) (λz::complex. z²) (− {0})
by (*simp add: covering_space_power_punctured_plane*)

proposition *covering_space_exp_punctured_plane:*
covering_space UNIV (λz::complex. exp z) (− {0})
proof (*simp add: covering_space_def, intro conjI ballI*)
 show continuous_on UNIV (λz::complex. exp z)
 by (*rule continuous_on_exp [OF continuous_on_id]*)
 show range exp = − {0::complex}
 by auto (*metis exp_Ln range_eqI*)
 show ∃ T. z ∈ T ∧ openin (top_of_set (− {0})) T ∧
 (∃ v. ∪ v = exp − ' T ∧ (∀ u ∈ v. open u) ∧ disjoint v ∧
 (∀ u ∈ v. ∃ q. homeomorphism u T exp q))
 if z ∈ − {0::complex} **for** z
proof −
 have z ≠ 0
 using that by auto
 have ball (Ln z) 1 ⊆ ball (Ln z) pi
 using pi_ge_two by (*simp add: ball_subset_ball_iff*)
 then have inj_exp: inj_on exp (ball (Ln z) 1)
 using inj_on_exp_pi inj_on_subset by blast
 define twopi where twopi ≡ λn. of_real (2 * of_int n * pi) * i
 define V where V ≡ range (λn. (λx. x + twopi n) ' (ball (Ln z) 1))
 have exp_eq': exp w = exp z ⇔ (∃ n::int. w = z + twopi n) **for** z w
 by (*simp add: exp_eq twopi_def*)
 show ?thesis
proof (*intro exI conjI*)

```

show  $z \in \exp \text{ ` } (\text{ball}(Ln\ z)\ 1)$ 
  by (metis  $\langle z \neq 0 \rangle$  centre_in_ball exp_Ln rev_image_eqI zero_less_one)
have open  $(- \{0::\text{complex}\})$ 
  by blast
with inj_exp show openin (top_of_set  $(- \{0\})$ ) (exp ` ball  $(Ln\ z)\ 1$ )
  by (auto simp: openin_open_eq invariance_of_domain continuous_on_exp
[OF continuous_on_id])
show  $UV: \bigcup \mathcal{V} = \exp \text{ ` } \exp \text{ ` } \text{ball}(Ln\ z)\ 1$ 
by (force simp: V_def twopi_def Complex_Transcendental.exp_eq_image_iff)
show  $\forall V \in \mathcal{V}. \text{open } V$ 
by (auto simp: V_def inj_on_def continuous_intros invariance_of_domain)
have  $2 \leq \text{cmod}(twopi\ m - twopi\ n)$  if  $m \neq n$  for  $m\ n$ 
proof -
  have  $1 \leq \text{abs}(m - n)$ 
    using that by linarith
  then have  $1 \leq \text{cmod}(of\_int\ m - of\_int\ n) * 1$ 
    by (metis mult.right_neutral norm_of_int of_int_1_le_iff of_int_abs
of_int_diff)
  also have  $\dots \leq \text{cmod}(of\_int\ m - of\_int\ n) * of\_real\ pi$ 
    using pi_ge_two
    by (intro mult_left_mono) auto
  also have  $\dots \leq \text{cmod}((of\_int\ m - of\_int\ n) * of\_real\ pi * i)$ 
    by (simp add: norm_mult)
  also have  $\dots \leq \text{cmod}(of\_int\ m * of\_real\ pi * i - of\_int\ n * of\_real\ pi * i)$ 
    by (simp add: algebra_simps)
  finally have  $1 \leq \text{cmod}(of\_int\ m * of\_real\ pi * i - of\_int\ n * of\_real\ pi$ 
 $* i)$  .
  then have  $2 * 1 \leq \text{cmod}(2 * (of\_int\ m * of\_real\ pi * i - of\_int\ n * of\_real\ pi * i))$ 
    by (metis mult_le_cancel_left_pos norm_mult_numeral1 zero_less_numeral)
  then show ?thesis
    by (simp add: algebra_simps twopi_def)
qed
then show disjoint  $\mathcal{V}$ 
  unfolding  $\mathcal{V}_\text{def}$  pairwise_def disjnt_iff
    by (smt (verit, best) add commute add_diff_cancel_left' add_diff_eq
dist_commute dist_complex_def dist_triangle imageE mem_ball)
show  $\forall u \in \mathcal{V}. \exists q. \text{homeomorphism } u (\exp \text{ ` } \text{ball}(Ln\ z)\ 1) \exp\ q$ 
proof
  fix  $u$ 
  assume  $u \in \mathcal{V}$ 
  then obtain  $n$  where  $u = (\lambda x. x + twopi\ n) \text{ ` } (\text{ball}(Ln\ z)\ 1)$ 
    by (auto simp: V_def)
  have compact  $(\text{cball}(Ln\ z)\ 1)$ 
    by simp
  moreover have continuous_on  $(\text{cball}(Ln\ z)\ 1) \exp$ 
    by (rule continuous_on_exp [OF continuous_on_id])
  moreover have inj_on  $\exp(\text{cball}(Ln\ z)\ 1)$ 
    using pi_ge_two inj_on_subset [OF inj_on_exp_pi [of  $Ln\ z$ ]]

```

```

    by (simp add: subset_iff)
  ultimately obtain  $\gamma$  where hom: homeomorphism (cball (Ln z) 1) (exp '
cball (Ln z) 1) exp  $\gamma$ 
    using homeomorphism_compact by blast
  have eq1: exp ' u = exp ' ball (Ln z) 1
    by (smt (verit) n exp_eq' image_cong image_image)
  have  $\gamma$ exp:  $\gamma$  (exp x) + twopi n = x if  $x \in u$  for x
  proof -
    have exp x = exp (x - twopi n)
      using exp_eq' by auto
    then have  $\gamma$  (exp x) =  $\gamma$  (exp (x - twopi n))
      by simp
    also have ... = x - twopi n
      using  $\langle x \in u \rangle$  by (auto simp: n intro: homeomorphism_apply1 [OF hom])
    finally show ?thesis
      by simp
  qed
  have exp2n: exp ( $\gamma$  (exp x) + twopi n) = exp x if dist (Ln z) x < 1 for x
    by (metis  $\gamma$ exp exp_eq' imageI mem_ball n that)
  have continuous_on (exp ' ball (Ln z) 1)  $\gamma$ 
    by (meson ball_subset_cball continuous_on_subset hom homeomor-
phism_cont2 image_mono)
  then have cont: continuous_on (exp ' ball (Ln z) 1) ( $\lambda x. \gamma$  x + twopi n)
    by (intro continuous_intros)
  have homeomorphism u (exp ' ball (Ln z) 1) exp (( $\lambda x. x$  + twopi n)  $\circ$   $\gamma$ )
    unfolding homeomorphism_def
  apply (intro conjI ballI eq1 continuous_on_exp [OF continuous_on_id])
  apply (auto simp:  $\gamma$ exp exp2n cont n)
  apply (force simp: image_iff homeomorphism_apply1 [OF hom])+
  done
  then show  $\exists q. \text{homeomorphism } u \text{ (exp ' ball (Ln z) 1) exp } q$  by metis
  qed
  qed
  qed
  qed

```

9.27.11 Hence the Borsukian results about mappings into circles

lemma *inessential_eq_continuous_logarithm*:

fixes $f :: 'a::\text{real_normed_vector} \Rightarrow \text{complex}$

shows $(\exists a. \text{homotopic_with_canon } (\lambda h. \text{True}) S \{-\{0\}\} f (\lambda t. a)) \longleftrightarrow$

$(\exists g. \text{continuous_on } S g \wedge (\forall x \in S. f x = \text{exp}(g x)))$

(is ?lhs \longleftrightarrow ?rhs)

proof

assume ?lhs **thus** ?rhs

by (metis covering_space_lift_inessential_function covering_space_exp_punctured_plane)

next

assume ?rhs

```

then obtain  $g$  where  $contg$ :  $continuous\_on\ S\ g$  and  $f$ :  $\bigwedge x. x \in S \implies f\ x = exp(g\ x)$ 
by metis
obtain  $a$  where  $homotopic\_with\_canon\ (\lambda h. True)\ S\ (-\ \{of\_real\ 0\})\ (exp\ \circ\ g)\ (\lambda x. a)$ 
proof (rule nullhomotopic\_through\_contractible [OF contg contractible_UNIV])
  show  $continuous\_on\ (UNIV::complex\ set)\ exp$ 
  by (intro continuous_intros)
  show  $exp \in UNIV \rightarrow -\{of\_real\ 0\}$ 
  by auto
qed force
then have  $homotopic\_with\_canon\ (\lambda h. True)\ S\ (-\ \{0\})\ f\ (\lambda t. a)$ 
using  $f$  homotopic\_with\_eq by fastforce
then show ?lhs ..
qed

```

```

corollary inessential_imp_continuous_logarithm_circle:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow complex$ 
  assumes  $homotopic\_with\_canon\ (\lambda h. True)\ S\ (sphere\ 0\ 1)\ f\ (\lambda t. a)$ 
  obtains  $g$  where  $continuous\_on\ S\ g$  and  $\bigwedge x. x \in S \implies f\ x = exp(g\ x)$ 
proof -
  have  $homotopic\_with\_canon\ (\lambda h. True)\ S\ (-\ \{0\})\ f\ (\lambda t. a)$ 
  using assms homotopic\_with\_subset\_right by fastforce
  then show ?thesis
  by (metis inessential_eq_continuous_logarithm that)
qed

```

```

lemma inessential_eq_continuous_logarithm_circle:
  fixes  $f :: 'a::real\_normed\_vector \Rightarrow complex$ 
  shows  $(\exists a. homotopic\_with\_canon\ (\lambda h. True)\ S\ (sphere\ 0\ 1)\ f\ (\lambda t. a)) \longleftrightarrow$ 
     $(\exists g. continuous\_on\ S\ g \wedge (\forall x \in S. f\ x = exp(i * of\_real(g\ x))))$ 
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume L: ?lhs
  then obtain  $g$  where  $contg$ :  $continuous\_on\ S\ g$  and  $g$ :  $\bigwedge x. x \in S \implies f\ x = exp(g\ x)$ 
  using inessential_imp_continuous_logarithm_circle by blast
  have  $f \in S \rightarrow sphere\ 0\ 1$ 
  by (metis L image_subset_iff_funcset homotopic\_with\_imp_subset1)
  then have  $\bigwedge x. x \in S \implies Re\ (g\ x) = 0$ 
  using  $g$  by (simp add: Pi_iff)
  then show ?rhs
  by (rule_tac  $x=Im \circ g$  in exI) (auto simp: Euler\ g\ intro: contg\ continuous_intros)
next
  assume ?rhs
  then obtain  $g$  where  $contg$ :  $continuous\_on\ S\ g$  and  $g$ :  $\bigwedge x. x \in S \implies f\ x = exp(i * of\_real(g\ x))$ 

```

by *metis*
obtain *a* where *homotopic_with_canon* ($\lambda h. \text{True}$) *S* (*sphere 0 1*) ((*exp* \circ ($\lambda z. i * z$)) \circ (*of_real* \circ *g*)) ($\lambda x. a$)
proof (*rule nullhomotopic_through_contractible*)
show *continuous_on* *S* (*complex_of_real* \circ *g*)
by (*intro conjI contg continuous_intros*)
show (*complex_of_real* \circ *g*) $\in S \rightarrow \mathbb{R}$
by *auto*
show *continuous_on* \mathbb{R} (*exp* \circ ($*$)*i*)
by (*intro continuous_intros*)
show (*exp* \circ ($*$)*i*) $\in \mathbb{R} \rightarrow \text{sphere } 0 \ 1$
by (*auto simp: complex_is_Real_iff*)
qed (*auto simp: convex_Reals convex_imp_contractible*)
moreover have $\bigwedge x. x \in S \implies (\text{exp} \circ (*i)) \circ (\text{complex_of_real} \circ g) x = f x$
by (*simp add: g*)
ultimately have *homotopic_with_canon* ($\lambda h. \text{True}$) *S* (*sphere 0 1*) *f* ($\lambda t. a$)
using *homotopic_with_eq* **by** *force*
then *show ?lhs ..*
qed

proposition *homotopic_with_sphere_times*:
fixes *f* :: '*a*::*real_normed_vector* \Rightarrow *complex*
assumes *hom*: *homotopic_with_canon* ($\lambda x. \text{True}$) *S* (*sphere 0 1*) *f g* **and** *conth*:
continuous_on *S* *h*
and *hin*: $\bigwedge x. x \in S \implies h x \in \text{sphere } 0 \ 1$
shows *homotopic_with_canon* ($\lambda x. \text{True}$) *S* (*sphere 0 1*) ($\lambda x. f x * h x$) ($\lambda x. g x * h x$)
proof –
obtain *k* where *contk*: *continuous_on* ($\{0..1\}::\text{real}\} \times S$) *k*
and *kim*: $k \in (\{0..1\} \times S) \rightarrow \text{sphere } 0 \ 1$
and *k0*: $\bigwedge x. k(0, x) = f x$
and *k1*: $\bigwedge x. k(1, x) = g x$
using *hom* **by** (*auto simp: homotopic_with_def*)
show *?thesis*
apply (*simp add: homotopic_with*)
apply (*rule_tac x= $\lambda z. k z * (h \circ \text{snd})z$ in exI*)
using *kim hin* **by** (*fastforce simp: conth norm_mult k0 k1 intro!: contk continuous_intros*)
qed

proposition *homotopic_circlemaps_divide*:
fixes *f* :: '*a*::*real_normed_vector* \Rightarrow *complex*
shows *homotopic_with_canon* ($\lambda x. \text{True}$) *S* (*sphere 0 1*) *f g* \longleftrightarrow
continuous_on *S* *f* $\wedge f \in S \rightarrow \text{sphere } 0 \ 1 \wedge$
continuous_on *S* *g* $\wedge g \in S \rightarrow \text{sphere } 0 \ 1 \wedge$
 $(\exists c. \text{homotopic_with_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) (\lambda x. f x / g x)$
 $(\lambda x. c))$
proof –
have *homotopic_with_canon* ($\lambda x. \text{True}$) *S* (*sphere 0 1*) ($\lambda x. f x / g x$) ($\lambda x. 1$)


```

    if homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. c$ )
  for  $c$ 
  proof -
    have  $S = \{\}$   $\vee$  path_component (sphere 0 1) 1  $c$ 
      using homotopic_with_imp_subset2 [OF that] path_connected_sphere [of
0::complex 1]
      by (auto simp: path_connected_component)
    with subtopology_empty_iff_trivial
    have homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1) ( $\lambda x. 1$ ) ( $\lambda x. c$ )
      by (force simp add: homotopic_constant_maps)
    then show ?thesis
      using homotopic_with_symD homotopic_with_trans that by blast
  qed
  then have *: ( $\exists c. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S (\text{sphere } 0 \ 1) (\lambda x. f x / g x) (\lambda x. c)$ )  $\longleftrightarrow$ 
    homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
  by auto
  have homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1)  $f g \longleftrightarrow$ 
    continuous_on  $S f \wedge f \in S \rightarrow \text{sphere } 0 \ 1 \wedge$ 
    continuous_on  $S g \wedge g \in S \rightarrow \text{sphere } 0 \ 1 \wedge$ 
    homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
    (is ?lhs  $\longleftrightarrow$  ?rhs)
  proof
    assume  $L: ?lhs$ 
    have geq1 [simp]:  $\bigwedge x. x \in S \implies \text{cmod } (g x) = 1$ 
      using homotopic_with_imp_subset2 [OF  $L$ ]
      by (simp add: image_subset_iff)
    have cont: continuous_on  $S$  (inverse  $\circ g$ )
    proof (rule continuous_intros)
      show continuous_on  $S g$ 
        using homotopic_with_imp_continuous [OF  $L$ ] by blast
      show continuous_on ( $g^{-1} S$ ) inverse
        by (rule continuous_on_subset [of sphere 0 1, OF continuous_on_inverse])
    auto
  qed
  have [simp]:  $\bigwedge x. x \in S \implies g x \neq 0$ 
    using geq1 by fastforce
  have homotopic_with_canon ( $\lambda x. \text{True}$ )  $S$  (sphere 0 1) ( $\lambda x. f x / g x$ ) ( $\lambda x. 1$ )
    using homotopic_with_eq [OF homotopic_with_sphere_times [OF  $L$  cont]]
    by (auto simp: divide_inverse norm_inverse)
  with  $L$  show ?rhs
  by (simp add: homotopic_with_imp_continuous homotopic_with_imp_funspace1)
next
  assume ?rhs then show ?lhs
    by (elim conjE homotopic_with_eq [OF homotopic_with_sphere_times];
force)
  qed
  then show ?thesis

```

by (*simp add: **)
qed

9.27.12 Upper and lower hemicontinuous functions

And relation in the case of preimage map to open and closed maps, and fact that upper and lower hemicontinuity together imply continuity in the sense of the Hausdorff metric (at points where the function gives a bounded and nonempty set).

Many similar proofs below.

lemma *upper_hemicontinuous*:

assumes $\bigwedge x. x \in S \implies f x \subseteq T$
shows $((\forall U. \text{openin } (\text{top_of_set } T) U \longrightarrow \text{openin } (\text{top_of_set } S) \{x \in S. f x \subseteq U\}) \longleftrightarrow (\forall U. \text{closedin } (\text{top_of_set } T) U \longrightarrow \text{closedin } (\text{top_of_set } S) \{x \in S. f x \cap U \neq \{\}\}))$
(is ?lhs = ?rhs)

proof (*intro iffI allI impI*)

fix U

assume $*$ [*rule_format*]: *?lhs* **and** $\text{closedin } (\text{top_of_set } T) U$

then have $\text{openin } (\text{top_of_set } T) (T - U)$

by (*simp add: openin_diff*)

then have $\text{openin } (\text{top_of_set } S) \{x \in S. f x \subseteq T - U\}$

using $*$ [*of T-U*] **by** *blast*

moreover have $S - \{x \in S. f x \subseteq T - U\} = \{x \in S. f x \cap U \neq \{\}\}$

using *assms* **by** *blast*

ultimately show $\text{closedin } (\text{top_of_set } S) \{x \in S. f x \cap U \neq \{\}\}$

by (*simp add: openin_closedin_eq*)

next

fix U

assume $*$ [*rule_format*]: *?rhs* **and** $\text{openin } (\text{top_of_set } T) U$

then have $\text{closedin } (\text{top_of_set } T) (T - U)$

by (*simp add: closedin_diff*)

then have $\text{closedin } (\text{top_of_set } S) \{x \in S. f x \cap (T - U) \neq \{\}\}$

using $*$ [*of T-U*] **by** *blast*

moreover have $\{x \in S. f x \cap (T - U) \neq \{\}\} = S - \{x \in S. f x \subseteq U\}$

using *assms* **by** *auto*

ultimately show $\text{openin } (\text{top_of_set } S) \{x \in S. f x \subseteq U\}$

by (*simp add: openin_closedin_eq*)

qed

lemma *lower_hemicontinuous*:

assumes $\bigwedge x. x \in S \implies f x \subseteq T$

shows $((\forall U. \text{closedin } (\text{top_of_set } T) U \longrightarrow \text{closedin } (\text{top_of_set } S) \{x \in S. f x \subseteq U\}) \longleftrightarrow (\forall U. \text{openin } (\text{top_of_set } T) U \longrightarrow \text{openin } (\text{top_of_set } S) \{x \in S. f x \cap U \neq \{\}\}))$
(is ?lhs = ?rhs)

```

proof (intro iffI allI impI)
  fix U
  assume * [rule_format]: ?lhs and openin (top_of_set T) U
  then have closedin (top_of_set T) (T - U)
    by (simp add: closedin_diff)
  then have closedin (top_of_set S) {x ∈ S. f x ⊆ T - U}
    using * [of T - U] by blast
  moreover have {x ∈ S. f x ⊆ T - U} = S - {x ∈ S. f x ∩ U ≠ {}}
    using assms by auto
  ultimately show openin (top_of_set S) {x ∈ S. f x ∩ U ≠ {}}
    by (simp add: openin_closedin_eq)
next
  fix U
  assume * [rule_format]: ?rhs and closedin (top_of_set T) U
  then have openin (top_of_set T) (T - U)
    by (simp add: openin_diff)
  then have openin (top_of_set S) {x ∈ S. f x ∩ (T - U) ≠ {}}
    using * [of T - U] by blast
  moreover have S - {x ∈ S. f x ∩ (T - U) ≠ {}} = {x ∈ S. f x ⊆ U}
    using assms by blast
  ultimately show closedin (top_of_set S) {x ∈ S. f x ⊆ U}
    by (simp add: openin_closedin_eq)
qed

lemma open_map_iff_lower_hemicontinuous_preimage:
  assumes f ∈ S → T
  shows ((∀ U. openin (top_of_set S) U
    → openin (top_of_set T) (f ' U)) ↔
    (∀ U. closedin (top_of_set S) U
    → closedin (top_of_set T) {y ∈ T. {x. x ∈ S ∧ f x = y} ⊆ U}))
  (is ?lhs = ?rhs)
proof (intro iffI allI impI)
  fix U
  assume * [rule_format]: ?lhs and closedin (top_of_set S) U
  then have openin (top_of_set S) (S - U)
    by (simp add: openin_diff)
  then have openin (top_of_set T) (f ' (S - U))
    using * [of S - U] by blast
  moreover have T - (f ' (S - U)) = {y ∈ T. {x ∈ S. f x = y} ⊆ U}
    using assms by blast
  ultimately show closedin (top_of_set T) {y ∈ T. {x ∈ S. f x = y} ⊆ U}
    by (simp add: openin_closedin_eq)
next
  fix U
  assume * [rule_format]: ?rhs and openSU: openin (top_of_set S) U
  then have closedin (top_of_set S) (S - U)
    by (simp add: closedin_diff)
  then have closedin (top_of_set T) {y ∈ T. {x ∈ S. f x = y} ⊆ S - U}
    using * [of S - U] by blast

```

moreover have $\{y \in T. \{x \in S. f x = y\} \subseteq S - U\} = T - (f \text{ ' } U)$
using *assms openin_imp_subset [OF opeSU]* **by** *auto*
ultimately show *openin (top_of_set T) (f \text{ ' } U)*
using *assms openin_imp_subset [OF opeSU]* **by** *(force simp: openin_closedin_eq)*
qed

lemma *closed_map_iff_upper_hemicontinuous_preimage:*

assumes $f \in S \rightarrow T$
shows $((\forall U. \text{closedin } (\text{top_of_set } S) U \rightarrow \text{closedin } (\text{top_of_set } T) (f \text{ ' } U)) \leftrightarrow$
 $(\forall U. \text{openin } (\text{top_of_set } S) U \rightarrow \text{openin } (\text{top_of_set } T) \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}))$
(is ?lhs = ?rhs)
proof *(intro iffI allI impI)*
fix U
assume $* [rule_format]: ?lhs$ **and** $opeSU: \text{openin } (\text{top_of_set } S) U$
then have $\text{closedin } (\text{top_of_set } S) (S - U)$
by *(simp add: closedin_diff)*
then have $\text{closedin } (\text{top_of_set } T) (f \text{ ' } (S - U))$
using $* [of S-U]$ **by** *blast*
moreover have $f \text{ ' } (S - U) = T - \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}$
using *assms openin_imp_subset [OF opeSU]* **by** *auto*
ultimately show $\text{openin } (\text{top_of_set } T) \{y \in T. \{x. x \in S \wedge f x = y\} \subseteq U\}$
using *assms openin_imp_subset [OF opeSU]* **by** *(force simp: openin_closedin_eq)*
next
fix U
assume $* [rule_format]: ?rhs$ **and** $cloSU: \text{closedin } (\text{top_of_set } S) U$
then have $\text{openin } (\text{top_of_set } S) (S - U)$
by *(simp add: openin_diff)*
then have $\text{openin } (\text{top_of_set } T) \{y \in T. \{x \in S. f x = y\} \subseteq S - U\}$
using $* [of S-U]$ **by** *blast*
moreover have $(f \text{ ' } U) = T - \{y \in T. \{x \in S. f x = y\} \subseteq S - U\}$
using *assms closedin_imp_subset [OF cloSU]* **by** *auto*
ultimately show $\text{closedin } (\text{top_of_set } T) (f \text{ ' } U)$
by *(simp add: openin_closedin_eq)*
qed

proposition *upper_lower_hemicontinuous_explicit:*

fixes $T :: ('b::\{\text{real_normed_vector, heine_borel}\}) \text{ set}$
assumes $fST: \bigwedge x. x \in S \implies f x \subseteq T$
and $ope: \bigwedge U. \text{openin } (\text{top_of_set } T) U \implies \text{openin } (\text{top_of_set } S) \{x \in S. f x \subseteq U\}$
and $clo: \bigwedge U. \text{closedin } (\text{top_of_set } T) U \implies \text{closedin } (\text{top_of_set } S) \{x \in S. f x \subseteq U\}$
and $x \in S \ 0 < e$ **and** $bofx: \text{bounded}(f x)$ **and** $fx_ne: f x \neq \{\}$
obtains d **where** $0 < d$
 $\bigwedge x'. \llbracket x' \in S; \text{dist } x x' < d \rrbracket$
 $\implies (\forall y \in f x. \exists y'. y' \in f x' \wedge \text{dist } y y' < e) \wedge$
 $(\forall y' \in f x'. \exists y. y \in f x \wedge \text{dist } y' y < e)$

proof –

```

have openin (top_of_set T) (T ∩ (⋃ a∈f x. ⋃ b∈ball 0 e. {a + b}))
  by (auto simp: open_sums openin_open_Int)
with ope have openin (top_of_set S)
  {u ∈ S. f u ⊆ T ∩ (⋃ a∈f x. ⋃ b∈ball 0 e. {a + b})} by blast
with <0 < e> <x ∈ S> obtain d1 where d1 > 0 and
  d1: ⋀ x'. [x' ∈ S; dist x' x < d1] ⇒ f x' ⊆ T ∧ f x' ⊆ (⋃ a ∈ f x. ⋃ b ∈
ball 0 e. {a + b})
  by (force simp: openin_euclidean_subtopology_iff dest: fST)
have oo: ⋀ U. openin (top_of_set T) U ⇒
  openin (top_of_set S) {x ∈ S. f x ∩ U ≠ {}}
  using lower_hemicontinuous_fST clo by blast
have compact (closure(f x))
  by (simp add: bofx)
moreover have closure(f x) ⊆ (⋃ a ∈ f x. ball a (e/2))
  using <0 < e> by (force simp: closure_approachable simp del: divide_const_simps)
ultimately obtain C where C ⊆ f x finite C closure(f x) ⊆ (⋃ a ∈ C. ball a
(e/2))
  by (meson compactE finite_subset_image Elementary_Metric_Spaces.open_ball
compactE_image)
then have fx_cover: f x ⊆ (⋃ a ∈ C. ball a (e/2))
  by (meson closure_subset order_trans)
with fx_ne have C ≠ {}
  by blast
have xin: x ∈ (⋂ a ∈ C. {x ∈ S. f x ∩ T ∩ ball a (e/2) ≠ {}})
  using <x ∈ S> <0 < e> fST <C ⊆ f x> by force
have openin (top_of_set S) {x ∈ S. f x ∩ (T ∩ ball a (e/2)) ≠ {}} for a
  by (simp add: openin_open_Int oo)
then have openin (top_of_set S) (⋂ a ∈ C. {x ∈ S. f x ∩ T ∩ ball a (e/2) ≠
{}})
  by (simp add: Int_assoc openin_INT2 [OF <finite C> <C ≠ {}>])
with xin obtain d2 where d2 > 0
  and d2: ⋀ u v. [u ∈ S; dist u x < d2; v ∈ C] ⇒ f u ∩ T ∩ ball v
(e/2) ≠ {}
  unfolding openin_euclidean_subtopology_iff using xin by fastforce
show ?thesis
proof (intro that conjI ballI)
  show 0 < min d1 d2
    using <0 < d1> <0 < d2> by linarith
next
  fix x' y
  assume x' ∈ S dist x x' < min d1 d2 y ∈ f x
  then have dd2: dist x' x < d2
    by (auto simp: dist_commute)
  obtain a where a ∈ C y ∈ ball a (e/2)
  using fx_cover <y ∈ f x> by auto
  then show ∃ y'. y' ∈ f x' ∧ dist y y' < e
    using d2 [OF <x' ∈ S> dd2] dist_triangle_half_r by fastforce
next

```

```

fix  $x' y'$ 
assume  $x' \in S \text{ dist } x x' < \min d1 d2 \ y' \in f x'$ 
then have  $\text{dist } x' x < d1$ 
  by (auto simp: dist_commute)
then have  $y' \in (\bigcup a \in f x. \bigcup b \in \text{ball } 0 \ e. \{a + b\})$ 
  using  $d1 \ [OF \ \langle x' \in S \rangle] \ \langle y' \in f x' \rangle$  by force
then show  $\exists y. y \in f x \wedge \text{dist } y' y < e$ 
  by clarsimp (metis add_diff_cancel_left' dist_norm)
qed
qed

```

9.27.13 Complex logs exist on various "well-behaved" sets

```

lemma continuous_logarithm_on_contractible:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$ 
  assumes  $\text{continuous\_on } S \ f \ \text{contractible } S \ \wedge z. z \in S \Longrightarrow f z \neq 0$ 
  obtains  $g$  where  $\text{continuous\_on } S \ g \ \wedge x. x \in S \Longrightarrow f x = \exp(g x)$ 
proof -
  obtain  $c$  where  $\text{hom: homotopic\_with\_canon } (\lambda h. \text{True}) \ S \ (-\{0\}) \ f \ (\lambda x. c)$ 
  using nullhomotopic_from_contractible assms
  by (metis imageE subset_Cmpl_singleton image_subset_iff_funcset)
  then show ?thesis
  by (metis inessential_eq_continuous_logarithm that)
qed

```

```

lemma continuous_logarithm_on_simply_connected:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$ 
  assumes  $\text{contf: continuous\_on } S \ f$  and  $S: \text{simply\_connected } S \ \text{locally\_path\_connected } S$ 
  and  $f: \wedge z. z \in S \Longrightarrow f z \neq 0$ 
  obtains  $g$  where  $\text{continuous\_on } S \ g \ \wedge x. x \in S \Longrightarrow f x = \exp(g x)$ 
  using covering_space_lift [OF covering_space_exp_punctured_plane S contf]
  by (metis (full_types) f imageE subset_Cmpl_singleton image_subset_iff_funcset)

```

```

lemma continuous_logarithm_on_cball:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$ 
  assumes  $\text{continuous\_on } (\text{cball } a \ r) \ f$  and  $\wedge z. z \in \text{cball } a \ r \Longrightarrow f z \neq 0$ 
  obtains  $h$  where  $\text{continuous\_on } (\text{cball } a \ r) \ h \ \wedge z. z \in \text{cball } a \ r \Longrightarrow f z = \exp(h z)$ 
  using assms continuous_logarithm_on_contractible convex_imp_contractible by blast

```

```

lemma continuous_logarithm_on_ball:
  fixes  $f :: 'a::\text{real\_normed\_vector} \Rightarrow \text{complex}$ 
  assumes  $\text{continuous\_on } (\text{ball } a \ r) \ f$  and  $\wedge z. z \in \text{ball } a \ r \Longrightarrow f z \neq 0$ 
  obtains  $h$  where  $\text{continuous\_on } (\text{ball } a \ r) \ h \ \wedge z. z \in \text{ball } a \ r \Longrightarrow f z = \exp(h z)$ 
  using assms continuous_logarithm_on_contractible convex_imp_contractible by blast

```

```

lemma continuous_sqrt_on_contractible:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  complex
  assumes continuous_on S f contractible S
    and  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where continuous_on S g  $\bigwedge x. x \in S \implies f x = (g x) ^ 2$ 
proof -
  obtain g where contg: continuous_on S g and feq:  $\bigwedge x. x \in S \implies f x = \exp(g x)$ 
  using continuous_logarithm_on_contractible [OF assms] by blast
  show ?thesis
  proof
    show continuous_on S ( $\lambda z. \exp (g z / 2)$ )
      by (rule continuous_on_compose2 [of UNIV exp]; intro continuous_intros
contg subset_UNIV) auto
    show  $\bigwedge x. x \in S \implies f x = (\exp (g x / 2))^2$ 
      by (metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
  qed
qed

```

```

lemma continuous_sqrt_on_simply_connected:
  fixes f :: 'a::real_normed_vector  $\Rightarrow$  complex
  assumes contf: continuous_on S f and S: simply_connected S locally_path_connected S
    and f:  $\bigwedge z. z \in S \implies f z \neq 0$ 
  obtains g where continuous_on S g  $\bigwedge x. x \in S \implies f x = (g x) ^ 2$ 
proof -
  obtain g where contg: continuous_on S g and feq:  $\bigwedge x. x \in S \implies f x = \exp(g x)$ 
  using continuous_logarithm_on_simply_connected [OF assms] by blast
  show ?thesis
  proof
    show continuous_on S ( $\lambda z. \exp (g z / 2)$ )
      by (rule continuous_on_compose2 [of UNIV exp]; intro continuous_intros
contg subset_UNIV) auto
    show  $\bigwedge x. x \in S \implies f x = (\exp (g x / 2))^2$ 
      by (metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
  qed
qed

```

9.27.14 Another simple case where sphere maps are nullhomotopic

```

lemma inessential_spheremap_2_aux:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  complex
  assumes 2:  $2 < DIM('a)$  and contf: continuous_on (sphere a r) f
    and fim:  $f \in (\text{sphere } a \ r) \rightarrow (\text{sphere } 0 \ 1)$ 

```

```

obtains c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere 0 1) f
( $\lambda x. c$ )
proof –
  obtain g where contg: continuous_on (sphere a r) g
    and feq:  $\bigwedge x. x \in \text{sphere } a \ r \implies f \ x = \exp(g \ x)$ 
  proof (rule continuous_logarithm_on_simply_connected [OF contf])
    show simply_connected (sphere a r)
      using 2 by (simp add: simply_connected_sphere_eq)
    show locally_path_connected (sphere a r)
      by (simp add: locally_path_connected_sphere)
    show  $\bigwedge z. z \in \text{sphere } a \ r \implies f \ z \neq 0$ 
      using fm by force
  qed auto
  have  $\exists g. \text{continuous\_on } (\text{sphere } a \ r) \ g \wedge (\forall x \in \text{sphere } a \ r. f \ x = \exp (i * \text{complex\_of\_real } (g \ x)))$ 
(plex_of_real (g x))
  proof (intro exI conjI)
    show continuous_on (sphere a r) (Im  $\circ$  g)
      by (intro contg continuous_intros continuous_on_compose)
    show  $\forall x \in \text{sphere } a \ r. f \ x = \exp (i * \text{complex\_of\_real } ((\text{Im} \circ g) \ x))$ 
      using exp_eq_polar feq fm norm_exp_eq_Re
      by (auto simp flip: image_subset_iff_funcset)
  qed
  with inessential_eq_continuous_logarithm_circle that show ?thesis
    by metis
qed

lemma inessential_spheremap_2:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes a2:  $2 < \text{DIM}('a)$  and b2:  $\text{DIM}('b) = 2$ 
  and contf: continuous_on (sphere a r) f and fm:  $f \in (\text{sphere } a \ r) \rightarrow (\text{sphere } b \ s)$ 
  obtains c where homotopic_with_canon ( $\lambda z. \text{True}$ ) (sphere a r) (sphere b s) f
( $\lambda x. c$ )
proof (cases s  $\leq$  0)
  case True
    then show ?thesis
      using contf contractible_sphere fm nullhomotopic_into_contractible that by
blast
  next
    case False
    then have sphere b s homeomorphic sphere (0::complex) 1
      using assms by (simp add: homeomorphic_spheres_gen)
    then obtain h k where hk: homeomorphism (sphere b s) (sphere (0::complex) 1) h k
      by (auto simp: homeomorphic_def)
    then have conth: continuous_on (sphere b s) h
      and contk: continuous_on (sphere 0 1) k
      and him:  $h \in \text{sphere } b \ s \rightarrow \text{sphere } 0 \ 1$ 
      and kim:  $k \in \text{sphere } 0 \ 1 \rightarrow \text{sphere } b \ s$ 

```



```

  by (force simp: homeomorphism_def)+
  obtain c where homotopic_with_canon ( $\lambda z. True$ ) (sphere a r) (sphere 0 1) (h
 $\circ f$ ) ( $\lambda x. c$ )
  proof (rule inessential_spheremap_2_aux [OF a2])
    show continuous_on (sphere a r) (h  $\circ f$ )
    by (meson contf conth continuous_on_compose continuous_on_subset fim
image_subset_iff_funcset)
    show (h  $\circ f$ )  $\in$  sphere a r  $\rightarrow$  sphere 0 1
    using fim him by force
  qed auto
  then have homotopic_with_canon ( $\lambda f. True$ ) (sphere a r) (sphere b s) (k  $\circ$  (h
 $\circ f$ )) (k  $\circ$  ( $\lambda x. c$ ))
  by (rule homotopic_with_compose_continuous_left [OF _ contk kim])
  moreover have  $\bigwedge x. r = \text{dist } a \ x \implies f \ x = k \ (h \ (f \ x))$ 
  by (metis fim hk homeomorphism_def image_subset_iff mem_sphere im-
age_subset_iff_funcset)
  ultimately have homotopic_with_canon ( $\lambda z. True$ ) (sphere a r) (sphere b s) f
( $\lambda x. k \ c$ )
  by (auto intro: homotopic_with_eq)
  then show ?thesis
  by (metis that)
qed

```

9.27.15 Holomorphic logarithms and square roots

lemma *g_imp_holomorphic_log*:

assumes *holf*: *f* holomorphic_on *S*

and *contg*: continuous_on *S* *g* and *feq*: $\bigwedge x. x \in S \implies f \ x = \exp \ (g \ x)$

and *fnz*: $\bigwedge z. z \in S \implies f \ z \neq 0$

obtains *g* where *g* holomorphic_on *S* $\bigwedge z. z \in S \implies f \ z = \exp(g \ z)$

proof –

have *contf*: continuous_on *S* *f*

by (simp add: holf holomorphic_on_imp_continuous_on)

have *g* field_differentiable at *z* within *S* if *f* field_differentiable at *z* within *S* *z* \in *S* for *z*

proof –

obtain *f'* where *f'*: $((\lambda y. (f \ y - f \ z) / (y - z)) \longrightarrow f')$ (at *z* within *S*)

using $\langle f \ \text{field_differentiable at } z \ \text{within } S \rangle$ by (auto simp: field_differentiable_def has_field_derivative_iff)

then have *ee*: $((\lambda x. (\exp(g \ x) - \exp(g \ z)) / (x - z)) \longrightarrow f')$ (at *z* within *S*)

by (simp add: feq $\langle z \in S \rangle$ Lim_transform_within [OF _ zero_less_one])

have $((\lambda y. \text{if } y = g \ z \ \text{then } \exp \ (g \ z) \ \text{else } (\exp \ y - \exp \ (g \ z)) / (y - g \ z)) \circ g) \longrightarrow \exp \ (g \ z)$
(at *z* within *S*)

proof (rule tendsto_compose_at)

show $(g \longrightarrow g \ z)$ (at *z* within *S*)

using *contg* continuous_on $\langle z \in S \rangle$ by blast

show $(\lambda y. \text{if } y = g \ z \ \text{then } \exp \ (g \ z) \ \text{else } (\exp \ y - \exp \ (g \ z)) / (y - g \ z)) - g \ z \rightarrow \exp \ (g \ z)$

```

    by (simp add: LIM_offset_zero_iff DERIV_D cong: if_cong Lim_cong_within)
    qed auto
  then have dd: (( $\lambda x$ . if  $g\ x = g\ z$  then  $\exp(g\ z)$  else  $(\exp(g\ x) - \exp(g\ z)) / (g\ x - g\ z)$ )  $\longrightarrow \exp(g\ z)$ ) (at  $z$  within  $S$ )
    by (simp add: o_def)
  have continuous (at  $z$  within  $S$ )  $g$ 
    using contg continuous_on_eq_continuous_within  $\langle z \in S \rangle$  by blast
  then have ( $\forall_F x$  in at  $z$  within  $S$ .  $\text{dist}(g\ x)\ (g\ z) < 2 * \pi$ )
    by (simp add: continuous_within tendsto_iff)
  then have  $\forall_F x$  in at  $z$  within  $S$ .  $\exp(g\ x) = \exp(g\ z) \longrightarrow g\ x \neq g\ z \longrightarrow x = z$ 
    by (rule eventually_mono) (auto simp: exp_eq dist_norm norm_mult)
  then have (( $\lambda y$ .  $(g\ y - g\ z) / (y - z)$ )  $\longrightarrow f' / \exp(g\ z)$ ) (at  $z$  within  $S$ )
    by (auto intro!: Lim_transform_eventually [OF tendsto_divide [OF ee dd]])
  then show ?thesis
    by (auto simp: field_differentiable_def has_field_derivative_iff)
  qed
  then have  $g$  holomorphic_on  $S$ 
    using holf holomorphic_on_def by auto
  then show ?thesis
    using feq that by auto
  qed

```

lemma *contractible_imp_holomorphic_log*:

```

  assumes holf:  $f$  holomorphic_on  $S$ 
    and  $S$ : contractible  $S$ 
    and fnz:  $\bigwedge z. z \in S \implies f\ z \neq 0$ 
  obtains  $g$  where  $g$  holomorphic_on  $S$   $\bigwedge z. z \in S \implies f\ z = \exp(g\ z)$ 
  proof -
    have contf: continuous_on  $S$   $f$ 
      by (simp add: holf holomorphic_on_imp_continuous_on)
    obtain  $g$  where contg: continuous_on  $S$   $g$  and feq:  $\bigwedge x. x \in S \implies f\ x = \exp(g\ x)$ 
      by (metis continuous_logarithm_on_contractible [OF contf  $S$  fnz])
    then show thesis
      using fnz  $g$ _imp_holomorphic_log holf that by blast
  qed

```

lemma *simply_connected_imp_holomorphic_log*:

```

  assumes holf:  $f$  holomorphic_on  $S$ 
    and  $S$ : simply_connected  $S$  locally_path_connected  $S$ 
    and fnz:  $\bigwedge z. z \in S \implies f\ z \neq 0$ 
  obtains  $g$  where  $g$  holomorphic_on  $S$   $\bigwedge z. z \in S \implies f\ z = \exp(g\ z)$ 
  proof -
    have contf: continuous_on  $S$   $f$ 
      by (simp add: holf holomorphic_on_imp_continuous_on)
    obtain  $g$  where contg: continuous_on  $S$   $g$  and feq:  $\bigwedge x. x \in S \implies f\ x = \exp(g\ x)$ 
      by (metis continuous_logarithm_on_simply_connected [OF contf  $S$  fnz])
  qed

```

then show *thesis*
using *fnz g_imp_holomorphic_log holf that by blast*
qed

lemma *contractible_imp_holomorphic_sqrt*:
assumes *holf: f holomorphic_on S*
and *S: contractible S*
and *fnz: $\bigwedge z. z \in S \implies f z \neq 0$*
obtains *g where g holomorphic_on S $\bigwedge z. z \in S \implies f z = g z^2$*
proof –
obtain *g where holg: g holomorphic_on S and feq: $\bigwedge z. z \in S \implies f z = \exp(g z)$*
using *contractible_imp_holomorphic_log [OF assms] by blast*
show *?thesis*
proof
show *$\exp \circ (\lambda z. z / 2) \circ g$ holomorphic_on S*
by *(intro holomorphic_on_compose holg holomorphic_intros) auto*
show *$\bigwedge z. z \in S \implies f z = ((\exp \circ (\lambda z. z / 2) \circ g) z)^2$*
by *(simp add: feq flip: exp_double)*
qed
qed

lemma *simply_connected_imp_holomorphic_sqrt*:
assumes *holf: f holomorphic_on S*
and *S: simply_connected S locally_path_connected S*
and *fnz: $\bigwedge z. z \in S \implies f z \neq 0$*
obtains *g where g holomorphic_on S $\bigwedge z. z \in S \implies f z = g z^2$*
proof –
obtain *g where holg: g holomorphic_on S and feq: $\bigwedge z. z \in S \implies f z = \exp(g z)$*
using *simply_connected_imp_holomorphic_log [OF assms] by blast*
show *?thesis*
proof
show *$\exp \circ (\lambda z. z / 2) \circ g$ holomorphic_on S*
by *(intro holomorphic_on_compose holg holomorphic_intros) auto*
show *$\bigwedge z. z \in S \implies f z = ((\exp \circ (\lambda z. z / 2) \circ g) z)^2$*
by *(simp add: feq flip: exp_double)*
qed
qed

Related theorems about holomorphic inverse cosines.

lemma *contractible_imp_holomorphic_arccos*:
assumes *holf: f holomorphic_on S and S: contractible S*
and *non1: $\bigwedge z. z \in S \implies f z \neq 1 \wedge f z \neq -1$*
obtains *g where g holomorphic_on S $\bigwedge z. z \in S \implies f z = \cos(g z)$*
proof –
have *hol1f: $(\lambda z. 1 - f z^2)$ holomorphic_on S*
by *(intro holomorphic_intros holf)*
obtain *g where holg: g holomorphic_on S and eq: $\bigwedge z. z \in S \implies 1 - (f z)^2 =$*

```

(g z)2
  using contractible_imp_holomorphic_sqrt [OF hol1f S]
  by (metis eq_iff_diff_eq_0 non1 power2_eq_1_iff)
  have holfg: (λz. f z + i*g z) holomorphic_on S
  by (intro holf holg holomorphic_intros)
  have ∧z. z ∈ S ⇒ f z + i*g z ≠ 0
  by (metis Arccos_body_lemma eq add commute add.inverse_unique complex_i_mult_minus
power2_csqrt power2_eq_iff)
  then obtain h where holh: h holomorphic_on S and fgeq: ∧z. z ∈ S ⇒ f z
+ i*g z = exp (h z)
  using contractible_imp_holomorphic_log [OF holfg S] by metis
  show ?thesis
  proof
  show (λz. -i*h z) holomorphic_on S
  by (intro holh holomorphic_intros)
  show f z = cos (- i*h z) if z ∈ S for z
  proof -
  have (f z + i*g z)*(f z - i*g z) = 1
  using that eq by (auto simp: algebra_simps power2_eq_square)
  then have f z - i*g z = inverse (f z + i*g z)
  using inverse_unique by force
  also have ... = exp (- h z)
  by (simp add: exp_minus fgeq that)
  finally have f z = exp (- h z) + i*g z
  by (simp add: diff_eq_eq)
  with that show ?thesis
  by (simp add: cos_exp_eq flip: fgeq)
  qed
  qed
  qed

```

lemma *contractible_imp_holomorphic_arccos_bounded*:

```

  assumes holf: f holomorphic_on S and S: contractible S and a ∈ S
  and non1: ∧z. z ∈ S ⇒ f z ≠ 1 ∧ f z ≠ -1
  obtains g where g holomorphic_on S norm(g a) ≤ pi + norm(f a) ∧z. z ∈ S
  ⇒ f z = cos(g z)
  proof -
  obtain g where holg: g holomorphic_on S and feq: ∧z. z ∈ S ⇒ f z = cos (g
z)
  using contractible_imp_holomorphic_arccos [OF holf S non1] by blast
  obtain b where cos b = f a norm b ≤ pi + norm (f a)
  using cos_Arccos norm_Arccos_bounded by blast
  then have cos b = cos (g a)
  by (simp add: ⟨a ∈ S⟩ feq)
  then consider n where n ∈ ℤ b = g a + of_real(2*n*pi) | n where n ∈ ℤ b
= -g a + of_real(2*n*pi)
  by (auto simp: complex_cos_eq)
  then show ?thesis

```

```

proof cases
  case 1
  show ?thesis
  proof
    show ( $\lambda z. g z + \text{of\_real}(2*n*pi)$ ) holomorphic_on S
      by (intro holomorphic_intros holg)
    show  $\text{cmod } (g a + \text{of\_real}(2*n*pi)) \leq pi + \text{cmod } (f a)$ 
      using 1  $\langle \text{cmod } b \leq pi + \text{cmod } (f a) \rangle$  by blast
    show  $\bigwedge z. z \in S \implies f z = \cos (g z + \text{complex\_of\_real } (2*n*pi))$ 
      by (metis  $\langle n \in \mathbb{Z} \rangle$  complex_cos_eq feq)
  qed
next
  case 2
  show ?thesis
  proof
    show ( $\lambda z. -g z + \text{of\_real}(2*n*pi)$ ) holomorphic_on S
      by (intro holomorphic_intros holg)
    show  $\text{cmod } (-g a + \text{of\_real}(2*n*pi)) \leq pi + \text{cmod } (f a)$ 
      using 2  $\langle \text{cmod } b \leq pi + \text{cmod } (f a) \rangle$  by blast
    show  $\bigwedge z. z \in S \implies f z = \cos (-g z + \text{complex\_of\_real } (2*n*pi))$ 
      by (metis  $\langle n \in \mathbb{Z} \rangle$  complex_cos_eq feq)
  qed
qed

```

9.27.16 The "Borsukian" property of sets

This doesn't have a standard name. Kuratowski uses "contractible with respect to $[S^1]$ " while Whyburn uses "property b". It's closely related to unicoherence.

definition *Borsukian where*

$$\begin{aligned} \text{Borsukian } S \equiv & \\ & \forall f. \text{continuous_on } S f \wedge f \in S \rightarrow (- \{0::\text{complex}\}) \\ & \rightarrow (\exists a. \text{homotopic_with_canon } (\lambda h. \text{True}) S (- \{0\}) f (\lambda x. a)) \end{aligned}$$

lemma *Borsukian_retraction_gen:*

```

assumes Borsukian S continuous_on S h  $h ' S = T$ 
  continuous_on T k  $k \in T \rightarrow S \bigwedge y. y \in T \implies h(k y) = y$ 
shows Borsukian T

```

proof –

```

interpret R: Retracts S h T k
  using assms by (simp add: image_subset_iff_funcset Retracts.intro)
show ?thesis
  using assms
  apply (clarsimp simp add: Borsukian_def)
  apply (rule R.cohomotopically_trivial_retraction_null_gen [OF TrueI TrueI
refl, of  $-\{0\}$ ], auto)
  done
qed

```

lemma *retract_of_Borsukian*: $\llbracket \text{Borsukian } T; S \text{ retract_of } T \rrbracket \implies \text{Borsukian } S$
by (*smt* (*verit*) *Borsukian_retraction_gen* *retract_of_def* *retraction* *retraction_def* *retraction_subset* *image_subset_iff_funcset*)

lemma *homeomorphic_Borsukian*: $\llbracket \text{Borsukian } S; S \text{ homeomorphic } T \rrbracket \implies \text{Borsukian } T$
using *Borsukian_retraction_gen* *order_refl*
by (*fastforce* *simp* *add*: *homeomorphism_def* *homeomorphic_def*)

lemma *homeomorphic_Borsukian_eq*:
 $S \text{ homeomorphic } T \implies \text{Borsukian } S \longleftrightarrow \text{Borsukian } T$
by (*meson* *homeomorphic_Borsukian* *homeomorphic_sym*)

lemma *Borsukian_translation*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{Borsukian } (\text{image } (\lambda x. a + x) S) \longleftrightarrow \text{Borsukian } S$
using *homeomorphic_Borsukian_eq* *homeomorphic_translation* **by** *blast*

lemma *Borsukian_injective_linear_image*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *linear* f *inj* f
shows $\text{Borsukian}(f \text{ ` } S) \longleftrightarrow \text{Borsukian } S$
using *assms* *homeomorphic_Borsukian_eq* *linear_homeomorphic_image* **by** *blast*

lemma *homotopy_eqv_Borsukianness*:
fixes $S :: 'a::\text{real_normed_vector_set}$
and $T :: 'b::\text{real_normed_vector_set}$
assumes $S \text{ homotopy_eqv } T$
shows $(\text{Borsukian } S \longleftrightarrow \text{Borsukian } T)$
by (*meson* *Borsukian_def* *assms* *homotopy_eqv_cohomotopic_triviality_null* *image_subset_iff_funcset*)

lemma *Borsukian_alt*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows
 $\text{Borsukian } S \longleftrightarrow$
 $(\forall f g. \text{continuous_on } S f \wedge f \in S \rightarrow -\{0\} \wedge$
 $\text{continuous_on } S g \wedge g \in S \rightarrow -\{0\}$
 $\longrightarrow \text{homotopic_with_canon } (\lambda h. \text{True}) S (-\{0::\text{complex}\}) f g)$
unfolding *Borsukian_def* *homotopic_triviality*
by (*force* *simp* *add*: *path_connected_punctured_universe*)

lemma *Borsukian_continuous_logarithm*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{Borsukian } S \longleftrightarrow$
 $(\forall f. \text{continuous_on } S f \wedge f \in S \rightarrow (-\{0::\text{complex}\})$
 $\longrightarrow (\exists g. \text{continuous_on } S g \wedge (\forall x \in S. f x = \text{exp}(g x))))$
by (*simp* *add*: *Borsukian_def* *inessential_eq_continuous_logarithm*)

```

lemma Borsukian_continuous_logarithm_circle:
  fixes S :: 'a::real_normed_vector set
  shows Borsukian S  $\longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } S f \wedge f \in S \rightarrow \text{sphere } (0::\text{complex}) 1$ 
       $\longrightarrow (\exists g. \text{continuous\_on } S g \wedge (\forall x \in S. f x = \text{exp}(g x)))$ )
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (force simp: Borsukian_continuous_logarithm)
next
  assume RHS [rule_format]: ?rhs
  show ?lhs
  proof (clarify simp: Borsukian_continuous_logarithm Pi_iff)
    fix f :: 'a  $\Rightarrow$  complex
    assume contf: continuous_on S f and 0:  $\forall i \in S. f i \neq 0$ 
    then have continuous_on S ( $\lambda x. f x / \text{complex\_of\_real } (\text{cmod } (f x))$ )
      by (intro continuous_intros) auto
    moreover have ( $\lambda x. f x / \text{complex\_of\_real } (\text{cmod } (f x)) \in S \rightarrow \text{sphere } 0 1$ )
      using 0 by (auto simp: norm_divide)
    ultimately obtain g where contg: continuous_on S g
      and fg:  $\forall x \in S. f x / \text{complex\_of\_real } (\text{cmod } (f x)) = \text{exp}(g x)$ 
    using RHS [of  $\lambda x. f x / \text{of\_real}(\text{norm}(f x))$ ] by auto
    show  $\exists g. \text{continuous\_on } S g \wedge (\forall x \in S. f x = \text{exp } (g x))$ 
  proof (intro exI ballI conjI)
    show continuous_on S ( $\lambda x. (Ln \circ \text{of\_real} \circ \text{norm} \circ f)x + g x$ )
      by (intro continuous_intros contf contg conjI) (use 0 in auto)
    show  $f x = \text{exp } ((Ln \circ \text{complex\_of\_real} \circ \text{cmod} \circ f) x + g x)$  if  $x \in S$  for  $x$ 
      using 0 that
      apply (simp add: exp_add)
    by (metis div_by_0 exp_Ln exp_not_eq_zero fg mult.commute nonzero_eq_divide_eq)
  qed
qed
qed
qed

```

```

lemma Borsukian_continuous_logarithm_circle_real:
  fixes S :: 'a::real_normed_vector set
  shows Borsukian S  $\longleftrightarrow$ 
    ( $\forall f. \text{continuous\_on } S f \wedge f \in S \rightarrow \text{sphere } (0::\text{complex}) 1$ 
       $\longrightarrow (\exists g. \text{continuous\_on } S (\text{complex\_of\_real} \circ g) \wedge (\forall x \in S. f x =$ 
 $\text{exp}(i * \text{of\_real}(g x))))$ )
  (is ?lhs = ?rhs)
proof
  assume LHS: ?lhs
  show ?rhs
  proof (clarify)
    fix f :: 'a  $\Rightarrow$  complex
    assume continuous_on S f and f01:  $f \in S \rightarrow \text{sphere } 0 1$ 

```

```

then obtain  $g$  where  $contg$ :  $continuous\_on\ S\ g$  and  $\bigwedge x. x \in S \implies f\ x = exp(g\ x)$ 
using  $LHS$  by ( $auto\ simp$ :  $Borsukian\_continuous\_logarithm\_circle$ )
then have  $\forall x \in S. f\ x = exp\ (i * complex\_of\_real\ ((Im \circ g)\ x))$ 
using  $f01\ exp\_eq\_polar\ norm\_exp\_eq\_Re$  by ( $fastforce\ simp$ :  $Pi\_iff$ )
then show  $\exists g. continuous\_on\ S\ (complex\_of\_real \circ g) \wedge (\forall x \in S. f\ x = exp\ (i * complex\_of\_real\ (g\ x)))$ 
by ( $rule\_tac\ x=Im \circ g$  in  $exI$ ) ( $force\ intro$ :  $continuous\_intros\ contg$ )
qed
next
assume  $RHS$  [ $rule\_format$ ]:  $?rhs$ 
show  $?lhs$ 
proof ( $clarsimp\ simp$ :  $Borsukian\_continuous\_logarithm\_circle$ )
fix  $f :: 'a \Rightarrow complex$ 
assume  $continuous\_on\ S\ f$  and  $f01$ :  $f \in S \rightarrow sphere\ 0\ 1$ 
then obtain  $g$  where  $contg$ :  $continuous\_on\ S\ (complex\_of\_real \circ g)$  and  $\bigwedge x. x \in S \implies f\ x = exp\ (i * of\_real(g\ x))$ 
by ( $metis\ RHS$ )
then show  $\exists g. continuous\_on\ S\ g \wedge (\forall x \in S. f\ x = exp\ (g\ x))$ 
by ( $rule\_tac\ x=\lambda x. i * of\_real(g\ x)$  in  $exI$ ) ( $auto\ simp$ :  $continuous\_intros\ contg$ )
qed
qed

```

lemma $Borsukian_circle$:

```

fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $Borsukian\ S \iff$ 
  ( $\forall f. continuous\_on\ S\ f \wedge f \in S \rightarrow sphere\ (0::complex)\ 1$ 
     $\longrightarrow (\exists a. homotopic\_with\_canon\ (\lambda h. True)\ S\ (sphere\ (0::complex)\ 1)$ 
       $f\ (\lambda x. a)))$ 
by ( $simp\ add$ :  $inessential\_eq\_continuous\_logarithm\_circle\ Borsukian\_continuous\_logarithm\_circle\_real$ )

```

lemma $contractible_imp_Borsukian$: $contractible\ S \implies Borsukian\ S$

by ($meson\ Borsukian_def\ nullhomotopic_from_contractible\ image_subset_iff_funcset$)

lemma $simply_connected_imp_Borsukian$:

```

fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $\llbracket simply\_connected\ S; locally\ path\_connected\ S \rrbracket \implies Borsukian\ S$ 
by ( $smt\ (verit, del\_insts)\ continuous\_logarithm\_on\_simply\_connected\ Borsukian\_continuous\_logarithm\_circle\_real$ 
   $PiE\ mem\_sphere\_0\ norm\_eq\_zero\ zero\_neq\_one$ )

```

lemma $starlike_imp_Borsukian$:

```

fixes  $S :: 'a::real\_normed\_vector\ set$ 
shows  $starlike\ S \implies Borsukian\ S$ 
by ( $simp\ add$ :  $contractible\_imp\_Borsukian\ starlike\_imp\_contractible$ )

```

lemma $Borsukian_empty$: $Borsukian\ \{\}$

by ($auto\ simp$: $contractible_imp_Borsukian$)

lemma *Borsukian_UNIV*: *Borsukian* (*UNIV* :: 'a::real_normed_vector set)
by (*auto simp: contractible_imp_Borsukian*)

lemma *convex_imp_Borsukian*:
fixes *S* :: 'a::real_normed_vector set
shows *convex S* \implies *Borsukian S*
by (*meson Borsukian_def convex_imp_contractible nullhomotopic_from_contractible*)

proposition *Borsukian_sphere*:
fixes *a* :: 'a::euclidean_space
shows $3 \leq \text{DIM}('a) \implies \text{Borsukian}(\text{sphere } a \ r)$
using *ENR_sphere*
by (*blast intro: simply_connected_imp_Borsukian ENR_imp_locally_path_connected simply_connected_sphere*)

lemma *Borsukian_Un_lemma*:
fixes *S* :: 'a::real_normed_vector set
assumes *BS*: *Borsukian S* **and** *BT*: *Borsukian T* **and** *ST*: *connected(S \cap T)*
and *: $\bigwedge f g :: 'a \Rightarrow \text{complex}$
 $\llbracket \text{continuous_on } S \ f; \text{continuous_on } T \ g; \bigwedge x. x \in S \wedge x \in T \implies f \ x = g \ x \rrbracket$
 $\implies \text{continuous_on} (S \cup T) (\lambda x. \text{if } x \in S \text{ then } f \ x \text{ else } g \ x)$
shows *Borsukian(S \cup T)*
proof (*clarsimp simp add: Borsukian_continuous_logarithm Pi_iff*)
fix *f* :: 'a \Rightarrow complex
assume *contf*: *continuous_on (S \cup T) f* **and** *0*: $\forall i \in S \cup T. f \ i \neq 0$
then have *contfS*: *continuous_on S f* **and** *contfT*: *continuous_on T f*
using *continuous_on_subset* **by** *auto*
have $\llbracket \text{continuous_on } S \ f; f \in S \rightarrow -\{0\} \rrbracket \implies \exists g. \text{continuous_on } S \ g \wedge (\forall x \in S. f \ x = \exp(g \ x))$
using *BS* **by** (*auto simp: Borsukian_continuous_logarithm*)
then obtain *g* **where** *contg*: *continuous_on S g* **and** *fg*: $\bigwedge x. x \in S \implies f \ x = \exp(g \ x)$
using *0 contfS* **by** *force*
have $\llbracket \text{continuous_on } T \ f; f \in T \rightarrow -\{0\} \rrbracket \implies \exists g. \text{continuous_on } T \ g \wedge (\forall x \in T. f \ x = \exp(g \ x))$
using *BT* **by** (*auto simp: Borsukian_continuous_logarithm*)
then obtain *h* **where** *conth*: *continuous_on T h* **and** *fh*: $\bigwedge x. x \in T \implies f \ x = \exp(h \ x)$
using *0 contfT* **by** *force*
show $\exists g. \text{continuous_on} (S \cup T) \ g \wedge (\forall x \in S \cup T. f \ x = \exp(g \ x))$
proof (*cases S \cap T = {}*)
case *True*
show *?thesis*
proof (*intro exI conjI*)
show *continuous_on (S \cup T) g* $\wedge (\forall x \in S \cup T. f \ x = \exp(g \ x))$
using *True* * [*OF contg conth*]
by (*meson disjoint_iff*)
show $\forall x \in S \cup T. f \ x = \exp(g \ x)$ *(if* *x* \in *S* *then* *g x* *else* *h x)*

3662

```

    using fg fh by auto
  qed
next
case False
have (λx. g x - h x) constant_on S ∩ T
proof (rule continuous_discrete_range_constant [OF ST])
  show continuous_on (S ∩ T) (λx. g x - h x)
  by (metis contg conth continuous_on_diff continuous_on_subset inf_le1
inf_le2)
  show ∃ e > 0. ∀ y. y ∈ S ∩ T ∧ g y - h y ≠ g x - h x → e ≤ cmod (g y -
h y - (g x - h x))
  if x ∈ S ∩ T for x
  proof -
    have g y - g x = h y - h x
      if y ∈ S y ∈ T cmod (g y - g x - (h y - h x)) < 2 * pi for y
    proof (rule exp_complex_eqI)
      have |Im (g y - g x) - Im (h y - h x)| ≤ cmod (g y - g x - (h y - h
x))
        by (metis abs_Im_le_cmod minus_complex.simps(2))
      then show |Im (g y - g x) - Im (h y - h x)| < 2 * pi
        using that by linarith
      have exp (g x) = exp (h x) exp (g y) = exp (h y)
        using fg fh that ⟨x ∈ S ∩ T⟩ by fastforce+
      then show exp (g y - g x) = exp (h y - h x)
        by (simp add: exp_diff)
    qed
  then show ?thesis
    by (rule_tac x=2*pi in exI) (fastforce simp add: algebra_simps)
  qed
qed
then obtain a where a: ∧x. x ∈ S ∩ T ⇒ g x - h x = a
  by (auto simp: constant_on_def)
with False have exp a = 1
  by (metis IntI disjoint_iff_not_equal divide_self_if exp_diff exp_not_eq_zero
fg fh)
with a show ?thesis
  apply (rule_tac x=λx. if x ∈ S then g x else a + h x in exI)
  apply (intro * contg conth continuous_intros conjI)
  apply (auto simp: algebra_simps fg fh exp_add)
done
qed
qed

```

proposition *Borsukian_open_Un*:

fixes $S :: 'a::real_normed_vector\ set$

assumes $opeS: openin (top_of_set (S \cup T)) S$

and $opeT: openin (top_of_set (S \cup T)) T$

and $BS: Borsukian\ S$ and $BT: Borsukian\ T$ and $ST: connected(S \cap T)$

shows $Borsukian(S \cup T)$

by (force intro: Borsukian_Un_lemma [OF BS BT ST] continuous_on_cases_local_open [OF opeS opeT])

lemma Borsukian_closed_Un:

fixes $S :: 'a::real_normed_vector_set$

assumes $cloS: closedin (top_of_set (S \cup T)) S$

and $cloT: closedin (top_of_set (S \cup T)) T$

and $BS: Borsukian S$ and $BT: Borsukian T$ and $ST: connected(S \cap T)$

shows Borsukian($S \cup T$)

by (force intro: Borsukian_Un_lemma [OF BS BT ST] continuous_on_cases_local [OF cloS cloT])

lemma Borsukian_separation_compact:

fixes $S :: complex\ set$

assumes compact S

shows Borsukian $S \longleftrightarrow connected(- S)$

by (simp add: Borsuk_separation_theorem Borsukian_circle assms)

lemma Borsukian_monotone_image_compact:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes Borsukian S and $contf: continuous_on S f$ and $fim: f ' S = T$

and compact S and $conn: \bigwedge y. y \in T \implies connected \{x. x \in S \wedge f x = y\}$

shows Borsukian T

proof (clarsimp simp: Borsukian_continuous_logarithm Pi_iff)

fix $g :: 'b \Rightarrow complex$

assume $contg: continuous_on T g$ and $0: \forall i \in T. g i \neq 0$

have $continuous_on S (g \circ f)$

using $contf contg continuous_on_compose fim$ by blast

moreover have $(g \circ f) ' S \subseteq -\{0\}$

using $fim 0$ by auto

ultimately obtain h where $conth: continuous_on S h$ and $gfh: \bigwedge x. x \in S \implies (g \circ f) x = exp(h x)$

using $\langle Borsukian S \rangle$ by (auto simp: Borsukian_continuous_logarithm)

have $\bigwedge y. \exists x. y \in T \longrightarrow x \in S \wedge f x = y$

using fim by auto

then obtain f' where $f': \bigwedge y. y \in T \longrightarrow f' y \in S \wedge f (f' y) = y$

by metis

have $*$: $(\lambda x. h x - h(f' y)) constant_on \{x. x \in S \wedge f x = y\}$ if $y \in T$ for y

proof (rule continuous_discrete_range_constant [OF conn [OF that], of $\lambda x. h x - h(f' y)$], simp_all add: algebra_simps)

show $continuous_on \{x \in S. f x = y\} (\lambda x. h x - h(f' y))$

by (intro continuous_intros continuous_on_subset [OF conth]) auto

show $\exists e > 0. \forall u. u \in S \wedge f u = y \wedge h u \neq h x \longrightarrow e \leq cmod (h u - h x)$

if $x \in S \wedge f x = y$ for x

proof -

have $h u = h x$ if $u \in S f u = y cmod (h u - h x) < 2 * pi$ for u

proof (rule exp_complex_eqI)

have $|Im (h u) - Im (h x)| \leq cmod (h u - h x)$

by (metis abs_Im_le_cmod minus_complex.simps(2))

```

    then show  $|Im (h u) - Im (h x)| < 2 * pi$ 
      using that by linarith
    show  $exp (h u) = exp (h x)$ 
      by (simp add: gfh [symmetric] x that)
  qed
  then show ?thesis
    by (rule_tac x=2*pi in exI) (fastforce simp add: algebra_simps)
  qed
  qed
  show  $\exists h. continuous\_on T h \wedge (\forall x \in T. g x = exp (h x))$ 
  proof (intro exI conjI)
    show continuous_on T (h o f')
  proof (rule continuous_from_closed_graph [of h ' S])
    show compact (h ' S)
      by (simp add: <compact S> compact_continuous_image conth)
    show  $(h \circ f') \in T \rightarrow h ' S$ 
      by (auto simp: f')
    have  $h x = h (f' (f x))$  if  $x \in S$  for  $x$ 
      using * [of f x] fm that unfolding constant_on_def by clarsimp (metis f'
imageI right_minus_eq)
    moreover have  $\bigwedge x. x \in T \implies \exists u. u \in S \wedge x = f u \wedge h (f' x) = h u$ 
      using f' by fastforce
    ultimately
      have eq:  $((\lambda x. (x, (h \circ f') x)) ' T) =$ 
 $\{p. \exists x. x \in S \wedge (x, p) \in (S \times UNIV) \cap ((\lambda z. snd z - ((f \circ fst) z,$ 
 $(h \circ fst) z)) - \{0\})\}$ 
      using fm by (auto simp: image_iff)
    moreover have closed ...
  apply (intro closed_compact_projection [OF <compact S>] continuous_closed_preimage
continuous_intros continuous_on_subset [OF contf] continu-
ous_on_subset [OF conth])
    by (auto simp: <compact S> closed_Times compact_imp_closed)
  ultimately show closed ((\lambda x. (x, (h o f') x)) ' T)
    by simp
  qed
  qed
  qed (use f' gfh in fastforce)
  qed

```

lemma *Borsukian_open_map_image_compact*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes *Borsukian S and contf: continuous_on S f and fim: f ' S = T and compact S*

and $ope: \bigwedge U. openin (top_of_set S) U$
 $\implies openin (top_of_set T) (f ' U)$

shows *Borsukian T*

proof (*clarsimp simp add: Borsukian_continuous_logarithm_circle_real*)

fix $g :: 'b \Rightarrow complex$

assume *contg: continuous_on T g and gim: g \in T \rightarrow sphere 0 1*

```

have continuous_on S (g o f)
  using contf contg continuous_on_compose fim by blast
moreover have (g o f) ∈ S → sphere 0 1
  using fim gim by auto
ultimately obtain h where cont_cxh: continuous_on S (complex_of_real o h)
  and gfh:  $\bigwedge x. x \in S \implies (g \circ f) x = \exp(i * \text{of\_real}(h x))$ 
  using ‹Borsukian S› Borsukian_continuous_logarithm_circle_real by metis
then have conth: continuous_on S h
  by simp
have  $\exists x. x \in S \wedge f x = y \wedge (\forall x' \in S. f x' = y \longrightarrow h x \leq h x')$  if  $y \in T$  for  $y$ 
proof -
  have 1: compact (h ‹ {x ∈ S. f x = y} ›)
  proof (rule compact_continuous_image)
    show continuous_on {x ∈ S. f x = y} h
      by (rule continuous_on_subset [OF conth]) auto
    have compact (S ∩ f ‹ {y} ›)
      using that proper_map_from_compact [OF contf ‹compact S›] fim
      by force
    then show compact {x ∈ S. f x = y}
      by (auto simp: vimage_def Int_def)
  qed
  have 2: h ‹ {x ∈ S. f x = y} › ≠ {}
    using fim that by auto
  have  $\exists s \in h ‹ {x \in S. f x = y} \rangle. \forall t \in h ‹ {x \in S. f x = y} \rangle. s \leq t$ 
    using compact_attains_inf [OF 1 2] by blast
  then show ?thesis by auto
qed
then obtain k where kTS:  $\bigwedge y. y \in T \implies k y \in S$ 
  and fk:  $\bigwedge y. y \in T \implies f (k y) = y$ 
  and hlc:  $\bigwedge x' y. \llbracket y \in T; x' \in S; f x' = y \rrbracket \implies h (k y) \leq h x'$ 
  by metis
have continuous_on T (h o k)
proof (clarsimp simp add: continuous_on_iff)
  fix y and e::real
  assume  $y \in T \ 0 < e$ 
  moreover have uniformly_continuous_on S (complex_of_real o h)
    using ‹compact S› cont_cxh compact_uniformly_continuous by blast
  ultimately obtain d where  $0 < d$ 
    and d:  $\bigwedge x x'. \llbracket x \in S; x' \in S; \text{dist } x' x < d \rrbracket \implies \text{dist } (h x') (h x) < e$ 
    by (force simp: uniformly_continuous_on_def)
  obtain  $\delta$  where  $0 < \delta$  and  $\delta$ :
     $\bigwedge x'. \llbracket x' \in T; \text{dist } y x' < \delta \rrbracket$ 
     $\implies (\forall v \in \{z \in S. f z = y\}. \exists v'. v' \in \{z \in S. f z = x'\} \wedge \text{dist } v v' <$ 
d)  $\wedge$ 
     $(\forall v' \in \{z \in S. f z = x'\}. \exists v. v \in \{z \in S. f z = y\} \wedge \text{dist } v' v < d)$ 
  proof (rule upper_lower_hemicontinuous_explicit [of T  $\lambda y. \{z \in S. f z = y\}$ 
S])
    show  $\bigwedge U. \text{openin } (\text{top\_of\_set } S) U$ 
       $\implies \text{openin } (\text{top\_of\_set } T) \{x \in T. \{z \in S. f z = x\} \subseteq U\}$ 

```

```

    using closed_map_iff_upper_hemicontinuous_preimage [of f S T] fim contf
  <compact S>
    using Abstract_Topology_2.continuous_imp_closed_map by blast
    show  $\bigwedge U. \text{closedin } (\text{top\_of\_set } S) U \implies$ 
       $\text{closedin } (\text{top\_of\_set } T) \{x \in T. \{z \in S. f z = x\} \subseteq U\}$ 
    using ope open_map_iff_lower_hemicontinuous_preimage [of f S T] fim
  [THEN equalityD1]
    by blast
    show bounded  $\{z \in S. f z = y\}$ 
      by (metis (no_types, lifting) compact_imp_bounded [OF <compact S>]
        bounded_subset mem_Collect_eq subsetI)
    qed (use <y ∈ T> <0 < d> fk kTS in <force+>)
    have  $\text{dist } (h (k y')) (h (k y)) < e$  if  $y' \in T \text{ dist } y y' < \delta$  for  $y'$ 
    proof -
      have k1:  $k y \in S f (k y) = y$  and k2:  $k y' \in S f (k y') = y'$ 
      by (auto simp: <y ∈ T> <y' ∈ T> kTS fk)
      have 1:  $\bigwedge v. \llbracket v \in S; f v = y \rrbracket \implies \exists v'. v' \in \{z \in S. f z = y'\} \wedge \text{dist } v v' < d$ 
      and 2:  $\bigwedge v'. \llbracket v' \in S; f v' = y' \rrbracket \implies \exists v. v \in \{z \in S. f z = y\} \wedge \text{dist } v' v < d$ 
      using  $\delta$  [OF that] by auto
      then obtain  $w' w$  where  $w' \in S f w' = y' \text{ dist } (k y) w' < d$ 
      and  $w \in S f w = y \text{ dist } (k y') w < d$ 
      using 1 [OF k1] 2 [OF k2] by auto
      then show ?thesis
      using  $d$  [of w k y]  $d$  [of w' k y] k1 k2 <y' ∈ T> <y ∈ T> hle
      by (fastforce simp: dist_norm abs_diff_less_iff algebra_simps)
    qed
    then show  $\exists d > 0. \forall x' \in T. \text{dist } x' y < d \implies \text{dist } (h (k x')) (h (k y)) < e$ 
      using <0 <  $\delta$ > by (auto simp: dist_commute)
    qed
    then show  $\exists h. \text{continuous\_on } T h \wedge (\forall x \in T. g x = \text{exp } (i * \text{complex\_of\_real } (h x)))$ 
      using fk gfh kTS by force
    qed

```

If two points are separated by a closed set, there's a minimal one.

proposition *closed_irreducible_separator:*

fixes $a :: 'a::\text{real_normed_vector}$

assumes $\text{closed } S$ and $ab: \neg \text{connected_component } (- S) a b$

obtains T where $T \subseteq S \text{ closed } T T \neq \{\}$ $\neg \text{connected_component } (- T) a b$
 $\bigwedge U. U \subset T \implies \text{connected_component } (- U) a b$

proof (cases $a \in S \vee b \in S$)

case *True*

then show ?thesis

proof

assume *: $a \in S$

show ?thesis

proof

show $\{a\} \subseteq S$

using * by blast

```

    show  $\neg$  connected_component ( $\neg$  {a}) a b
      using connected_component_in by auto
    show  $\bigwedge U. U \subseteq \{a\} \implies$  connected_component ( $\neg$  U) a b
      by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
less_le_not_le subset_singletonD)
    qed auto
  next
    assume *:  $b \in S$ 
    show ?thesis
    proof
      show  $\{b\} \subseteq S$ 
        using * by blast
      show  $\neg$  connected_component ( $\neg$  {b}) a b
        using connected_component_in by auto
      show  $\bigwedge U. U \subseteq \{b\} \implies$  connected_component ( $\neg$  U) a b
        by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
less_le_not_le subset_singletonD)
      qed auto
    qed
  next
    case False
    define A where  $A \equiv$  connected_component_set ( $\neg$  S) a
    define B where  $B \equiv$  connected_component_set ( $\neg$  (closure A)) b
    have  $a \in A$ 
      using False A_def by auto
    have  $b \in B$ 
      unfolding A_def B_def closure_Un_frontier
      using ab False  $\langle$ closed S $\rangle$  frontier_complement frontier_of_connected_component_subset
frontier_subset_closed by force
    have  $\text{frontier } B \subseteq \text{frontier } (\text{closure } A)$  b
      using B_def by blast
    also have frsub:  $\dots \subseteq \text{frontier } A$ 
    proof -
      have  $\bigwedge A. \text{closure } (\neg \text{closure } (\neg A)) \subseteq \text{closure } A$ 
        by (metis (no_types) closure_mono closure_subset compl_le_compl_iff double_compl)
      then show ?thesis
        by (metis (no_types) closure_closure double_compl frontier_closures frontier_of_connected_component_subset le_inf_iff subset_trans)
    qed
    finally have frBA:  $\text{frontier } B \subseteq \text{frontier } A$  .
    show ?thesis
    proof
      show  $\text{frontier } B \subseteq S$ 
      proof -
        have  $\text{frontier } S \subseteq S$ 
          by (simp add:  $\langle$ closed S $\rangle$  frontier_subset_closed)
        then show ?thesis
          using frsub frontier_complement frontier_of_connected_component_subset

```

```

      unfolding A_def B_def by blast
    qed
  show closed (frontier B)
    by simp
  show ¬ connected_component (− frontier B) a b
    unfolding connected_component_def
  proof clarify
    fix T
    assume connected T and TB: T ⊆ − frontier B and a ∈ T and b ∈ T
    have a ∉ B
      by (metis A_def B_def ComplD ⟨a ∈ A⟩ assms(1) closed_open connected_component_subset_in_closure_connected_component subsetD)
    have T ∩ B ≠ {}
      using ⟨b ∈ B⟩ ⟨b ∈ T⟩ by blast
    moreover have T − B ≠ {}
      using ⟨a ∉ B⟩ ⟨a ∈ T⟩ by blast
    ultimately show False
      using connected_Int_frontier [of T B] TB ⟨connected T⟩ by blast
    qed
  moreover have connected_component (− frontier B) a b if frontier B = {}
    using connected_component_eq_UNIV that by auto
  ultimately show frontier B ≠ {}
    by blast
  show connected_component (− U) a b if U ⊂ frontier B for U
  proof −
    obtain p where Usub: U ⊆ frontier B and p: p ∈ frontier B p ∉ U
      using ⟨U ⊂ frontier B⟩ by blast
    show ?thesis
      unfolding connected_component_def
    proof (intro exI conjI)
      have connected ((insert p A) ∪ (insert p B))
        proof (rule connected_Un)
          show connected (insert p A)
            by (metis A_def IntD1 frBA ⟨p ∈ frontier B⟩ closure_insert closure_subset connected_connected_component connected_intermediate_closure frontier_closures insert_absorb subsetCE subset_insertI)
          show connected (insert p B)
            by (metis B_def IntD1 ⟨p ∈ frontier B⟩ closure_insert closure_subset connected_connected_component connected_intermediate_closure frontier_closures insert_absorb subset_insertI)
        qed blast
      then show connected (insert p (B ∪ A))
        by (simp add: sup commute)
      have A ⊆ − U
        using A_def Usub ⟨frontier B ⊆ S⟩ connected_component_subset by
      fastforce
      moreover have B ⊆ − U
        using B_def Usub connected_component_subset frBA frontier_closures
      by fastforce
    end
  end

```



```

      ultimately show insert p (B ∪ A) ⊆ - U
      using p by auto
    qed (auto simp: ⟨a ∈ A⟩ ⟨b ∈ B⟩)
  qed
  qed
  qed

lemma frontier_minimal_separating_closed_pointwise:
  fixes S :: 'a::real_normed_vector set
  assumes S: closed S a ∉ S and nconn: ¬ connected_component (- S) a b
    and conn: ⋀T. [[closed T; T ⊂ S]] ⇒ connected_component (- T) a b
  shows frontier(connected_component_set (- S) a) = S (is ?F = S)
proof -
  have ?F ⊆ S
    by (simp add: S componentsI frontier_of_components_closed_complement)
  moreover have False if ?F ⊂ S
  proof -
    have connected_component (- ?F) a b
      by (simp add: conn that)
    then obtain T where connected T T ⊆ - ?F a ∈ T b ∈ T
      by (auto simp: connected_component_def)
    moreover have T ∩ ?F ≠ {}
    proof (rule connected_Int_frontier [OF ⟨connected T⟩])
      show T ∩ connected_component_set (- S) a ≠ {}
        using ⟨a ∉ S⟩ ⟨a ∈ T⟩ by fastforce
      show T - connected_component_set (- S) a ≠ {}
        using ⟨b ∈ T⟩ nconn by blast
    qed
  qed
  ultimately show ?thesis
    by blast
  qed
  ultimately show ?thesis
    by blast
  qed

```

9.27.17 Unicoherence (closed)

definition *unicoherent* where

```

unicoherent U ≡
  ∀ S T. connected S ∧ connected T ∧ S ∪ T = U ∧
    closedin (top_of_set U) S ∧ closedin (top_of_set U) T
  → connected (S ∩ T)

```

lemma *unicoherentI* [intro?]:

```

  assumes ⋀S T. [[connected S; connected T; U = S ∪ T; closedin (top_of_set
  U) S; closedin (top_of_set U) T]]
    ⇒ connected (S ∩ T)
  shows unicoherent U
  using assms unfolding unicoherent_def by blast

```

lemma *unicoherentD*:

assumes *unicoherent* U *connected* S *connected* T $U = S \cup T$ *closedin* (*top_of_set* U) S *closedin* (*top_of_set* U) T
shows *connected* ($S \cap T$)
using *assms* **unfolding** *unicoherent_def* **by** *blast*

proposition *homeomorphic_unicoherent*:

assumes ST : S *homeomorphic* T **and** S : *unicoherent* S
shows *unicoherent* T

proof –

obtain f g **where** gf : $\bigwedge x. x \in S \implies g(fx) = x$ **and** fim : $T = f' S$ **and** $gfim$:
 $g' f' S = S$

and $contf$: *continuous_on* S f **and** $contg$: *continuous_on* ($f' S$) g

using ST **by** (*auto simp: homeomorphic_def homeomorphism_def*)

show *?thesis*

proof

fix U V

assume *connected* U *connected* V **and** T : $T = U \cup V$

and $cloU$: *closedin* (*top_of_set* T) U

and $cloV$: *closedin* (*top_of_set* T) V

have $f \in (g' U \cap g' V) \rightarrow U$ $f \in (g' U \cap g' V) \rightarrow V$

using gf fim T **by** *auto* (*metis UnCI image_iff*)**+**

moreover **have** $U \cap V \subseteq f' (g' U \cap g' V)$

using gf fim **by** (*force simp: image_iff*) T

ultimately **have** $U \cap V = f' (g' U \cap g' V)$ **by** *blast*

moreover **have** *connected* ($f' (g' U \cap g' V)$)

proof (*rule connected_continuous_image*)

show *continuous_on* ($g' U \cap g' V$) f

using T fim $gfim$ **by** (*metis Un_upper1 contf continuous_on_subset image_mono inf_le1*)

show *connected* ($g' U \cap g' V$)

proof (*intro conjI unicoherentD [OF S]*)

show *connected* ($g' U$) *connected* ($g' V$)

using \langle *connected* U \rangle $cloU$ \langle *connected* V \rangle $cloV$

by (*metis Topological_Spaces.connected_continuous_image closedin_imp_subset contg continuous_on_subset fim*)**+**

show $S = g' U \cup g' V$

using T fim $gfim$ **by** *auto*

have hom : *homeomorphism* T S g f

by (*simp add: contf contg fim gf gfim homeomorphism_def*)

have *closedin* (*top_of_set* T) U *closedin* (*top_of_set* T) V

by (*simp_all add: cloU cloV*)

then **show** *closedin* (*top_of_set* S) ($g' U$)

closedin (*top_of_set* S) ($g' V$)

by (*blast intro: homeomorphism_imp_closed_map [OF hom]*)**+**

qed

qed

ultimately **show** *connected* ($U \cap V$) **by** *metis*

qed
qed

lemma *homeomorphic_unicoherent_eq*:
 S *homeomorphic* $T \implies (\text{unicoherent } S \longleftrightarrow \text{unicoherent } T)$
by (*meson homeomorphic_sym homeomorphic_unicoherent*)

lemma *unicoherent_translation*:
fixes $S :: 'a::\text{real_normed_vector_set}$
shows $\text{unicoherent } (\text{image } (\lambda x. a + x) S) \longleftrightarrow \text{unicoherent } S$
using *homeomorphic_translation homeomorphic_unicoherent_eq* **by** *blast*

lemma *unicoherent_injective_linear_image*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *linear f inj f*
shows $(\text{unicoherent}(f ` S) \longleftrightarrow \text{unicoherent } S)$
using *assms homeomorphic_unicoherent_eq linear_homeomorphic_image* **by**
blast

lemma *Borsukian_imp_unicoherent*:
fixes $U :: 'a::\text{euclidean_space}$ *set*
assumes *Borsukian U* **shows** *unicoherent U*
unfolding *unicoherent_def*
proof *clarify*
fix $S T$
assume *connected S connected T* $U = S \cup T$
and $\text{clo}S: \text{closedin } (\text{top_of_set } (S \cup T)) S$
and $\text{clo}T: \text{closedin } (\text{top_of_set } (S \cup T)) T$
show *connected (S \cap T)*
unfolding *connected_closedin_eq*
proof *clarify*
fix $V W$
assume $\text{closedin } (\text{top_of_set } (S \cap T)) V$
and $\text{closedin } (\text{top_of_set } (S \cap T)) W$
and $VW: V \cup W = S \cap T$ $V \cap W = \{\}$ **and** $V \neq \{\}$ $W \neq \{\}$
then **have** $\text{clo}V: \text{closedin } (\text{top_of_set } U) V$ **and** $\text{clo}W: \text{closedin } (\text{top_of_set } U) W$
using $\langle U = S \cup T \rangle$ *cloS cloT closedin_trans* **by** *blast+*
obtain q **where** *contq: continuous_on U q*
and $q01: \bigwedge x. x \in U \implies q x \in \{0..1::\text{real}\}$
and $qV: \bigwedge x. x \in V \implies q x = 0$ **and** $qW: \bigwedge x. x \in W \implies q x = 1$
by (*rule Urysohn_local [OF cloV cloW $\langle V \cap W = \{\} \rangle$, of 0 1]*)
(fastforce simp: closed_segment_eq_real_ivl)
let $?h = \lambda x. \text{if } x \in S \text{ then } \exp(\text{pi} * i * q x) \text{ else } 1 / \exp(\text{pi} * i * q x)$
have $\text{eq}ST: \exp(\text{pi} * i * q x) = 1 / \exp(\text{pi} * i * q x)$ **if** $x \in S \cap T$ **for** x
proof $-$

```

have  $x \in V \cup W$ 
  using that  $\langle V \cup W = S \cap T \rangle$  by blast
with  $qV$   $qW$  show ?thesis by force
qed
obtain  $g$  where  $contg$ : continuous_on  $U$   $g$ 
and  $circle$ :  $g \in U \rightarrow sphere\ 0\ 1$ 
and  $S$ :  $\bigwedge x. x \in S \implies g\ x = exp(pi * i * q\ x)$ 
and  $T$ :  $\bigwedge x. x \in T \implies g\ x = 1 / exp(pi * i * q\ x)$ 
proof
show continuous_on  $U$  ? $h$ 
  unfolding  $\langle U = S \cup T \rangle$ 
proof (rule continuous_on_cases_local [OF cloS cloT])
  show continuous_on  $S$  ( $\lambda x. exp\ (pi * i * q\ x)$ )
  proof (intro continuous_intros)
    show continuous_on  $S$   $q$ 
    using  $\langle U = S \cup T \rangle$  continuous_on_subset  $contg$  by blast
  qed
  show continuous_on  $T$  ( $\lambda x. 1 / exp\ (pi * i * q\ x)$ )
  proof (intro continuous_intros)
    show continuous_on  $T$   $q$ 
    using  $\langle U = S \cup T \rangle$  continuous_on_subset  $contg$  by auto
  qed auto
  qed (use eqST in auto)
qed (use eqST in  $\langle auto\ simp: norm\_divide \rangle$ )
then obtain  $h$  where  $conth$ : continuous_on  $U$   $h$  and  $heq$ :  $\bigwedge x. x \in U \implies g$ 
 $x = exp\ (h\ x)$ 
  by (metis Borsukian_continuous_logarithm_circle assms)
obtain  $v$   $w$  where  $v \in V$   $w \in W$ 
using  $\langle V \neq \{\} \rangle$   $\langle W \neq \{\} \rangle$  by blast
then have  $vw$ :  $v \in S \cap T$   $w \in S \cap T$ 
using  $VW$  by auto
have  $iff$ :  $2 * pi \leq cmod\ (2 * of\_int\ m * of\_real\ pi * i - 2 * of\_int\ n * of\_real\ pi * i)$ 
 $\longleftrightarrow 1 \leq abs\ (m - n)$  for  $m\ n$ 
proof -
have  $2 * pi \leq cmod\ (2 * of\_int\ m * of\_real\ pi * i - 2 * of\_int\ n * of\_real\ pi * i)$ 
 $\longleftrightarrow 2 * pi \leq cmod\ ((2 * pi * i) * (of\_int\ m - of\_int\ n))$ 
by (simp add: algebra_simps)
also have ...  $\longleftrightarrow 2 * pi \leq 2 * pi * cmod\ (of\_int\ m - of\_int\ n)$ 
by (simp add: norm_mult)
also have ...  $\longleftrightarrow 1 \leq abs\ (m - n)$ 
by simp (metis norm_of_int_of_int_1_le_iff_of_int_abs_of_int_diff)
finally show ?thesis .
qed
have *:  $\exists n::int. h\ x - (pi * i * q\ x) = (of\_int(2*n) * pi) * i$  if  $x \in S$  for  $x$ 
using that  $S$   $\langle U = S \cup T \rangle$   $heq\ exp\_eq$  [symmetric] by (simp add: algebra_simps)
moreover have ( $\lambda x. h\ x - (pi * i * q\ x)$ ) constant_on  $S$ 

```

```

proof (rule continuous_discrete_range_constant [OF ‹connected S›])
  have continuous_on S h continuous_on S q
    using ‹U = S ∪ T› continuous_on_subset conth contq by blast+
  then show continuous_on S (λx. h x - (pi * i * q x))
    by (intro continuous_intros)
  have 2*pi ≤ cmod (h y - (pi * i * q y) - (h x - (pi * i * q x)))
    if x ∈ S y ∈ S and ne: h y - (pi * i * q y) ≠ h x - (pi * i * q x) for x y
    using * [OF ‹x ∈ S›] * [OF ‹y ∈ S›] ne by (auto simp: iff)
  then show ∧x. x ∈ S ⇒
    ∃ e>0. ∀ y. y ∈ S ∧ h y - (pi * i * q y) ≠ h x - (pi * i * q x) ⇒
      e ≤ cmod (h y - (pi * i * q y) - (h x - (pi * i * q x)))
    by (rule_tac x=2*pi in exI) auto
qed
ultimately
obtain m where m: ∧x. x ∈ S ⇒ h x - (pi * i * q x) = (of_int(2*m) * pi)
* i
  using vw by (force simp: constant_on_def)
have *: ∃ n::int. h x = - (pi * i * q x) + (of_int(2*n) * pi) * i if x ∈ T for x
  unfolding exp_eq [symmetric]
    using that T ‹U = S ∪ T› by (simp add: exp_minus field_simps heq
[symmetric])
moreover have (λx. h x + (pi * i * q x)) constant_on T
proof (rule continuous_discrete_range_constant [OF ‹connected T›])
  have continuous_on T h continuous_on T q
    using ‹U = S ∪ T› continuous_on_subset conth contq by blast+
  then show continuous_on T (λx. h x + (pi * i * q x))
    by (intro continuous_intros)
  have 2*pi ≤ cmod (h y + (pi * i * q y) - (h x + (pi * i * q x)))
    if x ∈ T y ∈ T and ne: h y + (pi * i * q y) ≠ h x + (pi * i * q x) for x y
    using * [OF ‹x ∈ T›] * [OF ‹y ∈ T›] ne by (auto simp: iff)
  then show ∧x. x ∈ T ⇒
    ∃ e>0. ∀ y. y ∈ T ∧ h y + (pi * i * q y) ≠ h x + (pi * i * q x) ⇒
      e ≤ cmod (h y + (pi * i * q y) - (h x + (pi * i * q x)))
    by (rule_tac x=2*pi in exI) auto
qed
ultimately
obtain n where n: ∧x. x ∈ T ⇒ h x + (pi * i * q x) = (of_int(2*n) * pi)
* i
  using vw by (force simp: constant_on_def)
show False
  using m [of v] m [of w] n [of v] n [of w] vw
    by (auto simp: algebra_simps ‹v ∈ V› ‹w ∈ W› qV qW)
qed
qed

```

corollary contractible_imp_unicoherent:

fixes U :: 'a::euclidean_space set

assumes contractible U **shows** unicoherent U

by (simp add: Borsukian_imp_unicoherent assms contractible_imp_Borsukian)

corollary *convex_imp_unicoherent*:

fixes $U :: 'a::\text{euclidean_space set}$

assumes *convex* U **shows** *unicoherent* U

by (simp add: Borsukian_imp_unicoherent assms convex_imp_Borsukian)

If the type class constraint can be relaxed, I don't know how!

corollary *unicoherent_UNIV*: *unicoherent* ($UNIV :: 'a :: \text{euclidean_space set}$)

by (simp add: convex_imp_unicoherent)

lemma *unicoherent_monotone_image_compact*:

fixes $T :: 'b :: t2_space set$

assumes S : *unicoherent* S *compact* S **and** *contf*: *continuous_on* $S f$ **and** *fm*: f
' $S = T$

and *conn*: $\bigwedge y. y \in T \implies \text{connected } (S \cap f^{-1} \{y\})$

shows *unicoherent* T

proof

fix $U V$

assume UV : *connected* U *connected* V $T = U \cup V$

and *cloU*: *closedin* (*top_of_set* T) U

and *cloV*: *closedin* (*top_of_set* T) V

moreover **have** *compact* T

using $\langle \text{compact } S \rangle$ *compact_continuous_image* *contf* *fm* **by** *blast*

ultimately **have** *closed* U *closed* V

by (*auto* *simp*: *closedin_closed_eq* *compact_imp_closed*)

let $?SUV = (S \cap f^{-1} U) \cap (S \cap f^{-1} V)$

have UV_eq : $f^{-1} ?SUV = U \cap V$

using $\langle T = U \cup V \rangle$ *fm* **by** *force+*

have *connected* ($f^{-1} ?SUV$)

proof (*rule* *connected_continuous_image*)

show *continuous_on* $?SUV f$

by (*meson* *contf* *continuous_on_subset_inf_le1*)

show *connected* $?SUV$

proof (*rule* *unicoherentD* [*OF* $\langle \text{unicoherent } S \rangle$, *of* $S \cap f^{-1} U$ $S \cap f^{-1} V$])

have $\bigwedge C. \text{closedin } (\text{top_of_set } S) C \implies \text{closedin } (\text{top_of_set } T) (f^{-1} C)$

by (*metis* $\langle \text{compact } S \rangle$ *closed_subset* *closedin_compact* *closedin_imp_subset*
compact_continuous_image *compact_imp_closed* *contf* *continuous_on_subset* *fm*
image_mono)

then **show** *connected* ($S \cap f^{-1} U$) *connected* ($S \cap f^{-1} V$)

using UV **by** (*auto* *simp*: *conn* *intro*: *connected_closed_monotone_preimage*
[*OF* *contf* *fm*])

show $S = (S \cap f^{-1} U) \cup (S \cap f^{-1} V)$

using UV *fm* **by** *blast*

show *closedin* (*top_of_set* S) ($S \cap f^{-1} U$)

closedin (*top_of_set* S) ($S \cap f^{-1} V$)

by (*auto* *simp*: *continuous_on_imp_closedin* *cloU* *cloV* *contf* *fm*)

qed

```

qed
with UV_eq show connected (U ∩ V)
  by simp
qed

```

9.27.18 Several common variants of unicoherence

lemma *connected_frontier_simple*:

```

fixes S :: 'a :: euclidean_space set
assumes connected S connected (− S) shows connected(frontier S)
unfolding frontier_closures
by (rule unicoherentD [OF unicoherent_UNIV]; simp add: assms connected_imp_connected_closure
flip: closure_Un)

```

lemma *connected_frontier_component_complement*:

```

fixes S :: 'a :: euclidean_space set
assumes connected S C ∈ components(− S) shows connected(frontier C)
by (meson assms component_complement_connected connected_frontier_simple
in_components_connected)

```

lemma *connected_frontier_disjoint*:

```

fixes S :: 'a :: euclidean_space set
assumes connected S connected T disjnt S T and ST: frontier S ⊆ frontier T
shows connected(frontier S)

```

proof (cases S = UNIV)

```

case True then show ?thesis
  by simp

```

next

```

case False

```

```

then have −S ≠ {}

```

```

  by blast

```

```

then obtain C where C: C ∈ components(− S) and T ⊆ C

```

```

  by (metis ComplI disjnt_iff subsetI exists_component_superset ⟨disjnt S T⟩
⟨connected T⟩)

```

```

moreover have frontier S = frontier C

```

```

proof −

```

```

  have frontier C ⊆ frontier S

```

```

    using C frontier_complement frontier_of_components_subset by blast

```

```

moreover have x ∈ frontier C if x ∈ frontier S for x

```

```

proof −

```

```

  have x ∈ closure C

```

```

    using that unfolding frontier_def

```

```

    by (metis (no_types) Diff_eq ST ⟨T ⊆ C⟩ closure_mono contra_subsetD
frontier_def le_inf_iff that)

```

```

moreover have x ∉ interior C

```

```

    using that unfolding frontier_def

```

```

    by (metis C Compl_eq_Diff_UNIV Diff_iff subsetD in_components_subset
interior_diff interior_mono)

```

```

ultimately show ?thesis

```

```

      by (auto simp: frontier_def)
    qed
  ultimately show ?thesis
    by blast
  qed
  ultimately show ?thesis
    using ⟨connected S⟩ connected_frontier_component_complement by auto
  qed

```

9.27.19 Some separation results

```

lemma separation_by_component_closed_pointwise:
  fixes S :: 'a :: euclidean_space set
  assumes closed S  $\neg$  connected_component (– S) a b
  obtains C where C  $\in$  components S  $\neg$  connected_component(– C) a b
  proof (cases a  $\in$  S  $\vee$  b  $\in$  S)
  case True
  then show ?thesis
    using connected_component_in componentsI that by fastforce
  next
  case False
  obtain T where T  $\subseteq$  S closed T T  $\neq$  {}
    and nab:  $\neg$  connected_component (– T) a b
    and conn:  $\bigwedge U. U \subseteq T \implies$  connected_component (– U) a b
    using closed_irreducible_separator [OF assms] by metis
  moreover have connected T
  proof –
  have ab: frontier(connected_component_set (– T) a) = T frontier(connected_component_set
(– T) b) = T
    using frontier_minimal_separating_closed_pointwise
    by (metis False ⟨T  $\subseteq$  S⟩ ⟨closed T⟩ connected_component_sym conn connected_component_eq_empty connected_component_intermediate_subset empty_subsetI nab)+
  have connected (frontier (connected_component_set (– T) a))
  proof (rule connected_frontier_disjoint)
  show disjnt (connected_component_set (– T) a) (connected_component_set
(– T) b)
    unfolding disjnt_iff
    by (metis connected_component_eq connected_component_eq_empty connected_component_idemp mem_Collect_eq nab)
  show frontier (connected_component_set (– T) a)  $\subseteq$  frontier (connected_component_set
(– T) b)
    by (simp add: ab)
  qed auto
  with ab ⟨closed T⟩ show ?thesis
    by simp
  qed
  ultimately obtain C where C  $\in$  components S T  $\subseteq$  C
    using exists_component_superset [of T S] by blast

```



```

then show ?thesis
  by (meson Compl_anti_mono connected_component_of_subset nab that)
qed

```

lemma *separation_by_component_closed*:

```

fixes  $S :: 'a :: euclidean\_space$  set
assumes  $closed\ S \neg\ connected(-\ S)$ 
obtains  $C$  where  $C \in components\ S \neg\ connected(-\ C)$ 
proof -
  obtain  $x\ y$  where  $closed\ S\ x \notin S\ y \notin S$  and  $\neg\ connected\_component\ (-\ S)\ x\ y$ 
    using assms by (auto simp: connected_iff_connected_component)
  then obtain  $C$  where  $C \in components\ S \neg\ connected\_component(-\ C)\ x\ y$ 
    using separation_by_component_closed_pointwise by metis
  then show thesis
    by (metis Compl_iff  $\langle x \notin S \rangle \langle y \notin S \rangle\ connected\_component\_eq\_self\ in\_components\_subset$ 
mem_Collect_eq subsetD that)
qed

```

lemma *separation_by_Un_closed_pointwise*:

```

fixes  $S :: 'a :: euclidean\_space$  set
assumes  $ST: closed\ S\ closed\ T\ S \cap T = \{\}$ 
  and  $conS: connected\_component\ (-\ S)\ a\ b$  and  $conT: connected\_component$ 
 $(-\ T)\ a\ b$ 
  shows  $connected\_component\ (-\ (S \cup T))\ a\ b$ 
proof (rule ccontr)
  have  $a \notin S\ b \notin S\ a \notin T\ b \notin T$ 
    using  $conS\ conT\ connected\_component\_in$  by auto
  assume  $\neg\ connected\_component\ (-\ (S \cup T))\ a\ b$ 
  then obtain  $C$  where  $C \in components\ (S \cup T)$  and  $C: \neg\ connected\_component(-\ C)\ a\ b$ 
    using separation_by_component_closed_pointwise assms by blast
  then have  $C \subseteq S \vee C \subseteq T$ 
  proof -
    have  $connected\ C\ C \subseteq S \cup T$ 
      using  $\langle C \in components\ (S \cup T) \rangle\ in\_components\_subset$  by (blast elim: componentsE)+
    moreover then have  $C \cap T = \{\} \vee C \cap S = \{\}$ 
      by (metis Int_empty_right ST inf.commute connected_closed)
    ultimately show ?thesis
      by blast
  qed
then show False
  by (meson Compl_anti_mono C conS conT connected_component_of_subset)
qed

```

lemma *separation_by_Un_closed*:

```

fixes  $S :: 'a :: euclidean\_space$  set
assumes  $ST: closed\ S\ closed\ T\ S \cap T = \{\}$  and  $conS: connected(-\ S)$  and

```

```

conT: connected(- T)
  shows connected(- (S ∪ T))
  using assms separation_by_Un_closed_pointwise
  by (fastforce simp add: connected_iff_connected_component)

lemma open_unicoherent_UNIV:
  fixes S :: 'a :: euclidean_space set
  assumes open S open T connected S connected T S ∪ T = UNIV
  shows connected(S ∩ T)
proof -
  have connected(- (-S ∪ -T))
  by (metis closed_Compl compl_sup compl_top_eq double_compl separation_by_Un_closed
  assms)
  then show ?thesis
  by simp
qed

lemma separation_by_component_open_aux:
  fixes S :: 'a :: euclidean_space set
  assumes ST: closed S closed T S ∩ T = {}
  and S ≠ {} T ≠ {}
  obtains C where C ∈ components(-(S ∪ T)) C ≠ {} frontier C ∩ S ≠ {}
  frontier C ∩ T ≠ {}
proof (rule ccontr)
  let ?S = S ∪ ⋃{C ∈ components(-(S ∪ T)). frontier C ⊆ S}
  let ?T = T ∪ ⋃{C ∈ components(-(S ∪ T)). frontier C ⊆ T}
  assume ¬thesis
  with that have *: frontier C ∩ S = {} ∨ frontier C ∩ T = {}
  if C: C ∈ components(-(S ∪ T)) C ≠ {} for C
  using C by blast
  have ∃A B::'a set. closed A ∧ closed B ∧ UNIV ⊆ A ∪ B ∧ A ∩ B = {} ∧ A
  ≠ {} ∧ B ≠ {}
proof (intro exI conjI)
  have frontier (⋃{C ∈ components(- S ∩ - T). frontier C ⊆ S}) ⊆ S
  using subset_trans [OF frontier_Union_subset_closure]
  by (metis (no_types, lifting) SUP_least ‹closed S› closure_minimal mem_Collect_eq)
  then have frontier ?S ⊆ S
  by (simp add: frontier_subset_eq assms subset_trans [OF frontier_Un_subset])
  then show closed ?S
  using frontier_subset_eq by fastforce
  have frontier (⋃{C ∈ components(- S ∩ - T). frontier C ⊆ T}) ⊆ T
  using subset_trans [OF frontier_Union_subset_closure]
  by (metis (no_types, lifting) SUP_least ‹closed T› closure_minimal mem_Collect_eq)
  then have frontier ?T ⊆ T
  by (simp add: frontier_subset_eq assms subset_trans [OF frontier_Un_subset])
  then show closed ?T
  using frontier_subset_eq by fastforce
  have UNIV ⊆ (S ∪ T) ∪ ⋃(components(-(S ∪ T)))
  using Union_components by blast

```

```

also have ...  $\subseteq$  ?S  $\cup$  ?T
proof -
  have C  $\in$  components  $(-(S \cup T)) \wedge$  frontier C  $\subseteq$  S  $\vee$ 
    C  $\in$  components  $(-(S \cup T)) \wedge$  frontier C  $\subseteq$  T
  if C  $\in$  components  $(-(S \cup T))$  C  $\neq$  {} for C
  using * [OF that] that
  by clarify (metis (no_types, lifting) UnE <closed S> <closed T> closed_Un
disjoint_iff_not_equal frontier_of_components_closed_complement subsetCE)
  then show ?thesis
  by blast
qed
finally show UNIV  $\subseteq$  ?S  $\cup$  ?T .
have  $\bigcup\{C \in$  components  $(-(S \cup T)).$  frontier C  $\subseteq$  S $\} \cup$ 
   $\bigcup\{C \in$  components  $(-(S \cup T)).$  frontier C  $\subseteq$  T $\} \subseteq$   $-(S \cup T)$ 
  using in_components_subset by fastforce
moreover have  $\bigcup\{C \in$  components  $(-(S \cup T)).$  frontier C  $\subseteq$  S $\} \cap$ 
   $\bigcup\{C \in$  components  $(-(S \cup T)).$  frontier C  $\subseteq$  T $\} =$  {}
proof -
  have C  $\cap$  C' = {} if C  $\in$  components  $(-(S \cup T))$  frontier C  $\subseteq$  S
    C'  $\in$  components  $(-(S \cup T))$  frontier C'  $\subseteq$  T for C C'
proof -
  have NUN:  $-(S \cap -T) \neq$  UNIV
  using <T  $\neq$  {}> by blast
  have C  $\neq$  C'
proof
  assume C = C'
  with that have frontier C'  $\subseteq$  S  $\cap$  T
  by simp
  also have ... = {}
  using <S  $\cap$  T = {}> by blast
  finally have C' = {}  $\vee$  C' = UNIV
  using frontier_eq_empty by auto
  then show False
  using <C = C'> NUN that by (force simp: dest: in_components_nonempty
in_components_subset)
qed
with that show ?thesis
  by (simp add: components_nonoverlap [of _  $-(S \cup T)$ ])
qed
then show ?thesis
  by blast
qed
ultimately show ?S  $\cap$  ?T = {}
  using ST by blast
show ?S  $\neq$  {} ?T  $\neq$  {}
  using <S  $\neq$  {}> <T  $\neq$  {}> by blast+
qed
then show False
  by (metis Compl_disjoint connected_UNIV compl_bot_eq compl_unique

```

3680

connected_closedD inf_sup_absorb sup_compl_top_left1 top.extremum_uniqueI
qed

proposition *separation_by_component_open:*

fixes $S :: 'a :: \text{euclidean_space set}$

assumes *open S and non:* $\neg \text{connected}(-S)$

obtains C **where** $C \in \text{components } S \neg \text{connected}(-C)$

proof $-$

obtain $T U$

where *closed T closed U and* $TU: T \cup U = -S \ T \cap U = \{\} \ T \neq \{\} \ U \neq \{\}$

using *assms by (auto simp: connected_closed_set closed_def)*

then obtain C **where** $C: C \in \text{components}(-(T \cup U)) \ C \neq \{\}$

and *frontier C \cap T \neq {} frontier C \cap U \neq {}*

using *separation_by_component_open_aux [OF $\langle \text{closed } T \rangle \langle \text{closed } U \rangle \langle T \cap U = \{\} \rangle$]* **by force**

show *thesis*

proof

show $C \in \text{components } S$

using $C(1) \ TU(1)$ **by auto**

show $\neg \text{connected}(-C)$

proof

assume $\text{connected}(-C)$

then have $\text{connected}(\text{frontier } C)$

using *connected_frontier_simple [of C] $\langle C \in \text{components } S \rangle \text{in_components_connected}$*
by blast

then show *False*

unfolding *connected_closed*

by (*metis C(1) TU(2) $\langle \text{closed } T \rangle \langle \text{closed } U \rangle \langle \text{frontier } C \cap T \neq \{\} \rangle \langle \text{frontier } C \cap U \neq \{\} \rangle \text{closed_Un_frontier_of_components_closed_complement inf_bot_right inf_commute}$*)

qed

qed

qed

lemma *separation_by_Un_open:*

fixes $S :: 'a :: \text{euclidean_space set}$

assumes *open S open T S \cap T = {} and cS: connected(-S) and cT: connected(-T)*

shows $\text{connected}(-(S \cup T))$

using *assms uncoherent_UNIV unfolding uncoherent_def* **by force**

lemma *nonseparation_by_component_eq:*

fixes $S :: 'a :: \text{euclidean_space set}$

assumes *open S \vee closed S*

shows $((\forall C \in \text{components } S. \text{connected}(-C)) \longleftrightarrow \text{connected}(-S))$

by (*metis assms component_complement_connected double_complement separa-*

tion_by_component_closed_separation_by_component_open)

Another interesting equivalent of an inessential mapping into C-0

proposition *inessential_eq_extensible*:

fixes $f :: 'a::euclidean_space \Rightarrow complex$

assumes $closed\ S$

shows $(\exists a. homotopic_with_canon\ (\lambda h. True)\ S\ (-\{0\})\ f\ (\lambda t. a)) \longleftrightarrow$
 $(\exists g. continuous_on\ UNIV\ g \wedge (\forall x \in S. g\ x = f\ x) \wedge (\forall x. g\ x \neq 0))$
(is $?lhs = ?rhs$ **)**

proof

assume $?lhs$

then obtain a **where** $a: homotopic_with_canon\ (\lambda h. True)\ S\ (-\{0\})\ f\ (\lambda t. a)$

..

show $?rhs$

proof ($cases\ S = \{\}$)

case $True$

with a **show** $?thesis$ **by force**

next

case $False$

have $anr: ANR\ (-\{0::complex\})$

by ($simp\ add: ANR_delete_open_Compl_open_imp_ANR$)

obtain g **where** $contg: continuous_on\ UNIV\ g$ **and** $gim: g \in UNIV \rightarrow -\{0\}$

and $gf: \bigwedge x. x \in S \implies g\ x = f\ x$

proof ($rule\ Borsuk_homotopy_extension_homotopic\ [OF\ ___ continuous_on_const$
 $___ homotopic_with_symD\ [OF\ a]]$)

show $closedin\ (top_of_set\ UNIV)\ S$

using $assms$ **by auto**

show $(\lambda t. a) \in UNIV \rightarrow -\{0\}$

using a $homotopic_with_imp_subset2\ False$ **by blast**

qed ($use\ anr\ that\ in\ \langle force+\rangle$)

then show $?thesis$

by force

qed

next

assume $?rhs$

then obtain g **where** $contg: continuous_on\ UNIV\ g$

and $gf: \bigwedge x. x \in S \implies g\ x = f\ x$ **and** $non0: \bigwedge x. g\ x \neq 0$

by metis

obtain $h\ k::'a \Rightarrow 'a$ **where** $hk: homeomorphism\ (ball\ 0\ 1)\ UNIV\ h\ k$

using $homeomorphic_ball01_UNIV\ homeomorphic_def$ **by blast**

then have $continuous_on\ (ball\ 0\ 1)\ (g \circ h)$

by ($meson\ contg\ continuous_on_compose\ continuous_on_subset\ homeomor-$
 $phism_cont1\ top_greatest$)

then obtain j **where** $contj: continuous_on\ (ball\ 0\ 1)\ j$

and $j: \bigwedge z. z \in ball\ 0\ 1 \implies exp(j\ z) = (g \circ h)\ z$

by ($metis\ (mono_tags,\ opaque_lifting)\ continuous_logarithm_on_ball\ comp_apply$
 $non0$)

have $[simp]: \bigwedge x. x \in S \implies h\ (k\ x) = x$

using $hk\ homeomorphism_apply2$ **by blast**

```

have  $\exists \zeta. \text{continuous\_on } S \zeta \wedge (\forall x \in S. f x = \text{exp } (\zeta x))$ 
proof (intro exI conjI ballI)
  show  $\text{continuous\_on } S (j \circ k)$ 
  proof (rule continuous_on_compose)
    show  $\text{continuous\_on } S k$ 
    by (meson continuous_on_subset hk homeomorphism_cont2 top_greatest)
    show  $\text{continuous\_on } (k ' S) j$ 
    by (auto intro: continuous_on_subset [OF contj] simp flip: homeomorphism_image2 [OF hk])
  qed
  show  $f x = \text{exp } ((j \circ k) x)$  if  $x \in S$  for  $x$ 
  by (metis UNIV_I comp_apply gf hk homeomorphism_def image_eqI j that)
qed
then show ?lhs
  by (simp add: inessential_eq_continuous_logarithm)
qed

```

lemma *inessential_on_clopen_Union*:

```

fixes  $\mathcal{F} :: 'a::\text{euclidean\_space set set}$ 
assumes  $T: \text{path\_connected } T$ 
  and  $\bigwedge S. S \in \mathcal{F} \implies \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
  and  $\bigwedge S. S \in \mathcal{F} \implies \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
  and  $\text{hom}: \bigwedge S. S \in \mathcal{F} \implies \exists a. \text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. a)$ 
a)
  obtains  $a$  where  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) (\bigcup \mathcal{F}) T f (\lambda x. a)$ 
proof (cases  $\bigcup \mathcal{F} = \{\}$ )
  case True
  with that show ?thesis
  using homotopic_with_canon_on_empty by fastforce
next
  case False
  then obtain  $C$  where  $C \in \mathcal{F} \ C \neq \{\}$ 
  by blast
  then obtain  $a$  where  $\text{clo}: \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) C$ 
  and  $\text{ope}: \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) C$ 
  and  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) C T f (\lambda x. a)$ 
  using assms by blast
  with  $\langle C \neq \{\} \rangle$  have  $f \in C \rightarrow T a \in T$ 
  using homotopic_with_imp_subset1 homotopic_with_imp_subset2 by blast+
  have  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) (\bigcup \mathcal{F}) T f (\lambda x. a)$ 
proof (rule homotopic_on_clopen_Union)
  show  $\bigwedge S. S \in \mathcal{F} \implies \text{closedin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
  and  $\bigwedge S. S \in \mathcal{F} \implies \text{openin } (\text{top\_of\_set } (\bigcup \mathcal{F})) S$ 
  by (simp_all add: assms)
  show  $\text{homotopic\_with\_canon } (\lambda x. \text{True}) S T f (\lambda x. a)$  if  $S \in \mathcal{F}$  for  $S$ 
proof (cases  $S = \{\}$ )
  case False
  then obtain  $b$  where  $b \in S$ 
  by blast

```

```

obtain  $c$  where  $c$ : homotopic_with_canon  $(\lambda x. \text{True})$   $S$   $T$   $f$   $(\lambda x. c)$ 
  using  $\langle S \in \mathcal{F} \rangle$  hom by blast
then have  $c \in T$ 
  using  $\langle b \in S \rangle$  homotopic_with_imp_subset2 by blast
then have homotopic_with_canon  $(\lambda x. \text{True})$   $S$   $T$   $(\lambda x. a)$   $(\lambda x. c)$ 
using  $T \langle a \in T \rangle$  by (simp add: homotopic_constant_maps path_connected_component)
then show ?thesis
  using  $c$  homotopic_with_symD homotopic_with_trans by blast
qed (simp add: homotopic_on_empty)
qed
then show ?thesis ..
qed

```

proposition *Janiszewski_dual*:

```

fixes  $S :: \text{complex set}$ 
assumes compact S compact T connected S connected T connected( $-(S \cup T)$ )
shows connected( $S \cap T$ )
by (meson Borsukian_imp_unicoherent Borsukian_separation_compact assms
closed_subset compact_Un
compact_imp_closed sup_ge1 sup_ge2 unicoherentD)

```

end

9.28 The Jordan Curve Theorem and Applications

theory *Jordan_Curve*

```

imports Arcwise_Connected Further_Topology
begin

```

9.28.1 Janiszewski's theorem

lemma *Janiszewski_weak*:

```

fixes  $a b :: \text{complex}$ 
assumes compact S compact T and conST: connected( $S \cap T$ )
  and  $ccS$ : connected_component ( $- S$ )  $a$   $b$  and  $ccT$ : connected_component
( $- T$ )  $a$   $b$ 

```

```

shows connected_component ( $-(S \cup T)$ )  $a$   $b$ 

```

proof –

```

have [simp]:  $a \notin S$   $a \notin T$   $b \notin S$   $b \notin T$ 
by (meson ComplD ccS ccT connected_component_in)+
have  $clo$ : closedin (top_of_set ( $S \cup T$ ))  $S$  closedin (top_of_set ( $S \cup T$ ))  $T$ 
by (simp_all add: assms closed_subset compact_imp_closed)
obtain  $g$  where contg: continuous_on S g
  and  $g$ :  $\bigwedge x. x \in S \implies \exp(i * \text{of\_real } (g x)) = (x - a) /_R \text{cmod } (x -$ 
 $a) / ((x - b) /_R \text{cmod } (x - b))$ 
using  $ccS$   $\langle \text{compact } S \rangle$ 
apply (simp add: Borsuk_maps_homotopic_in_connected_component_eq [symmetric])
apply (subst (asm) homotopic_circlemaps_divide)
apply (auto simp: inessential_eq_continuous_logarithm_circle)

```

```

done
obtain h where conth: continuous_on T h
  and h:  $\bigwedge x. x \in T \implies \exp (i^* \text{ of\_real } (h x)) = (x - a) /_R \text{ cmod } (x - a) / ((x - b) /_R \text{ cmod } (x - b))$ 
  using ccT ⟨compact T⟩
  apply (simp add: Borsuk_maps_homotopic_in_connected_component_eq [symmetric])
  apply (subst (asm) homotopic_circlemaps_divide)
  apply (auto simp: inessential_eq_continuous_logarithm_circle)
done
have continuous_on (S ∪ T) (λx. (x - a) /_R cmod (x - a)) continuous_on (S ∪ T) (λx. (x - b) /_R cmod (x - b))
  by (intro continuous_intros; force)+
moreover have (λx. (x - a) /_R cmod (x - a)) ‘(S ∪ T) ⊆ sphere 0 1 (λx. (x - b) /_R cmod (x - b))’ ‘(S ∪ T) ⊆ sphere 0 1’
  by (auto simp: divide_simps)
moreover have  $\exists g. \text{ continuous\_on } (S \cup T) g \wedge (\forall x \in S \cup T. (x - a) /_R \text{ cmod } (x - a) / ((x - b) /_R \text{ cmod } (x - b)) = \exp (i^* \text{ complex\_of\_real } (g x)))$ 
  proof (cases S ∩ T = {})
  case True
  then have continuous_on (S ∪ T) (λx. if x ∈ S then g x else h x)
    using continuous_on_cases_local [OF clo contg conth]
    by (meson disjoint_iff)
  then show ?thesis
    by (rule_tac x=(λx. if x ∈ S then g x else h x) in exI) (auto simp: g h)
  case False
  then have diffpi:  $\exists n. g x = h x + 2^* \text{ of\_int } n^* \text{ pi}$  if  $x \in S \cap T$  for x
    proof -
    have  $\exp (i^* \text{ of\_real } (g x)) = \exp (i^* \text{ of\_real } (h x))$ 
      using that by (simp add: g h)
    then obtain n where  $\text{ complex\_of\_real } (g x) = \text{ complex\_of\_real } (h x) + 2^* \text{ of\_int } n^* \text{ complex\_of\_real } \text{ pi}$ 
      apply (simp add: exp_eq)
      by (metis complex_i_not_zero distrib_left mult.commute mult_cancel_left)
    then show ?thesis
      using of_real_eq_iff by (fastforce intro!: exI [where x=n])
    qed
  have contgh: continuous_on (S ∩ T) (λx. g x - h x)
    by (intro continuous_intros continuous_on_subset [OF contg] continuous_on_subset [OF conth]) auto
  moreover have disc:
     $\exists e > 0. \forall y. y \in S \cap T \wedge g y - h y \neq g x - h x \implies e \leq \text{ norm } ((g y - h y) - (g x - h x))$ 
    if  $x \in S \cap T$  for x
  proof -
  obtain nx where  $nx: g x = h x + 2^* \text{ of\_int } nx^* \text{ pi}$ 
    using ⟨x ∈ S ∩ T⟩ diffpi by blast
  have  $2^* \text{ pi} \leq \text{ norm } (g y - h y - (g x - h x))$  if  $y: y \in S \cap T$  and  $neq: g y$ 

```


– $h y \neq g x - h x$ for y

proof –

obtain ny **where** $ny: g y = h y + 2 * \text{of_int } ny * \pi$

using $\langle y \in S \cap T \rangle$ **diffpi** **by** *blast*

{ **assume** $nx \neq ny$

then have $1 \leq |\text{real_of_int } ny - \text{real_of_int } nx|$

by *linarith*

then have $(2 * \pi) * 1 \leq (2 * \pi) * |\text{real_of_int } ny - \text{real_of_int } nx|$

by *simp*

also have $\dots = |2 * \text{real_of_int } ny * \pi - 2 * \text{real_of_int } nx * \pi|$

by (*simp add: algebra_simps abs_if*)

finally have $2 * \pi \leq |2 * \text{real_of_int } ny * \pi - 2 * \text{real_of_int } nx * \pi|$ **by**

simp

}

with *neq* **show** *?thesis*

by (*simp add: nx ny*)

qed

then show *?thesis*

by (*rule_tac x=2*\pi in exI*) *auto*

qed

ultimately have $(\lambda x. g x - h x)$ *constant_on* $S \cap T$

using *continuous_discrete_range_constant* [*OF conST contgh*] **by** *blast*

then obtain z **where** $z: \bigwedge x. x \in S \cap T \implies g x - h x = z$

by (*auto simp: constant_on_def*)

obtain w **where** $\exp(i * \text{of_real}(h w)) = \exp(i * \text{of_real}(z + h w))$

using *disc z False*

by *auto* (*metis diff_add_cancel g h of_real_add*)

then have [*simp*]: $\exp(i * \text{of_real } z) = 1$

by (*metis cis_conv_exp cis_mult exp_not_eq_zero mult_cancel_right1*)

show *?thesis*

proof (*intro exI conjI*)

show *continuous_on* $(S \cup T)$ $(\lambda x. \text{if } x \in S \text{ then } g x \text{ else } z + h x)$

by (*intro continuous_intros continuous_on_cases_local* [*OF clo contg*]

contg) (*use z in force*)

qed (*auto simp: g h algebra_simps exp_add*)

qed

ultimately have *homotopic_with_canon* $(\lambda x. \text{True})$ $(S \cup T)$ (*sphere 0 1*)

$(\lambda x. (x - a) /_{\mathbb{R}} \text{cmod } (x - a))$ $(\lambda x. (x - b) /_{\mathbb{R}} \text{cmod } (x - b))$

by (*subst homotopic_circlemaps_divide*) (*auto simp: inessential_eq_continuous_logarithm_circle*)

moreover have *compact* $(S \cup T)$

using *assms* **by** *blast*

ultimately show *?thesis*

using *assms Borsuk_maps_homotopic_in_connected_component_eq* **by** *fast-*

force

qed

theorem *Janiszewski*:

fixes $a b :: \text{complex}$

```

assumes compact  $S$  closed  $T$  and conST: connected  $(S \cap T)$ 
and ccS: connected_component  $(- S)$   $a b$  and ccT: connected_component
 $(- T)$   $a b$ 
shows connected_component  $(- (S \cup T))$   $a b$ 
proof -
have path_component  $(- T)$   $a b$ 
by (simp add: <closed  $T$ > ccT open_Cmpl open_path_connected_component)
then obtain  $g$  where  $g$ : path  $g$  path_image  $g \subseteq - T$  pathstart  $g = a$  pathfinish
 $g = b$ 
by (auto simp: path_component_def)
then obtain  $C$  where  $C$ : compact  $C$  connected  $C$   $a \in C$   $b \in C$   $C \cap T = \{\}$ 
by fastforce
obtain  $r$  where  $0 < r$  and  $r$ :  $C \cup S \subseteq \text{ball } 0 r$ 
by (metis <compact  $C$ > <compact  $S$ > bounded_Un compact_imp_bounded bounded_subset_ballD)
have connected_component  $(- (S \cup (T \cap \text{cball } 0 r \cup \text{sphere } 0 r)))$   $a b$ 
proof (rule Janiszewski_weak [OF <compact  $S$ >])
show comT': compact  $((T \cap \text{cball } 0 r) \cup \text{sphere } 0 r)$ 
by (simp add: <closed  $T$ > closed_Int_compact compact_Un)
have  $S \cap (T \cap \text{cball } 0 r \cup \text{sphere } 0 r) = S \cap T$ 
using  $r$  by auto
with conST show connected  $(S \cap (T \cap \text{cball } 0 r \cup \text{sphere } 0 r))$ 
by simp
show connected_component  $(- (T \cap \text{cball } 0 r \cup \text{sphere } 0 r))$   $a b$ 
using conST  $C$   $r$ 
apply (simp add: connected_component_def)
apply (rule_tac  $x=C$  in exI)
by auto
qed (simp add: ccS)
then obtain  $U$  where  $U$ : connected  $U$   $U \subseteq - S$   $U \subseteq - T \cup - \text{cball } 0 r$   $U \subseteq$ 
 $- \text{sphere } 0 r$   $a \in U$   $b \in U$ 
by (auto simp: connected_component_def)
show ?thesis
unfolding connected_component_def
proof (intro exI conjI)
show  $U \subseteq - (S \cup T)$ 
using  $U$   $r$  < $0 < r$ > < $a \in C$ > connected_Int_frontier [of  $U$   $\text{cball } 0 r$ ]
apply simp
by (metis ball_subset_cball compl_inf_disjoint_eq_subset_Cmpl disjoint_iff_not_equal
inf.orderE inf_sup_aci(3) subsetCE)
qed (auto simp:  $U$ )
qed

```

lemma Janiszewski_connected:

fixes S :: complex set

assumes ST: compact S closed T connected $(S \cap T)$

and notST: connected $(- S)$ connected $(- T)$

shows connected $(- (S \cup T))$

using Janiszewski [OF ST] **by** (metis IntD1 IntD2 notST compl_sup connected_iff_connected_compon

9.28.2 The Jordan Curve theorem

lemma *exists_double_arc*:

fixes $g :: \text{real} \Rightarrow 'a::\text{real_normed_vector}$

assumes *simple_path* g *pathfinish* $g = \text{pathstart } g$ $a \in \text{path_image } g$ $b \in \text{path_image } g$ $a \neq b$

obtains u d **where** *arc* u *arc* d *pathstart* $u = a$ *pathfinish* $u = b$
pathstart $d = b$ *pathfinish* $d = a$
 $(\text{path_image } u) \cap (\text{path_image } d) = \{a, b\}$
 $(\text{path_image } u) \cup (\text{path_image } d) = \text{path_image } g$

proof –

obtain u **where** $0 \leq u \leq 1$ $g \ u = a$

using *assms* **by** (*auto simp: path_image_def*)

define h **where** $h \equiv \text{shiftpath } u \ g$

have *simple_path* h

using $\langle \text{simple_path } g \rangle$ *simple_path_shiftpath* $\langle 0 \leq u \rangle$ $\langle u \leq 1 \rangle$ *assms*(2) *h_def*

by *blast*

have *pathstart* $h = g \ u$

by (*simp add: $\langle u \leq 1 \rangle$ h_def pathstart_shiftpath*)

have *pathfinish* $h = g \ u$

by (*simp add: $\langle 0 \leq u \rangle$ assms h_def pathfinish_shiftpath*)

have *pihg*: *path_image* $h = \text{path_image } g$

by (*simp add: $\langle 0 \leq u \rangle$ $\langle u \leq 1 \rangle$ assms h_def path_image_shiftpath*)

then obtain v **where** $0 \leq v \leq 1$ $h \ v = b$

using *assms* **by** (*metis (mono_tags, lifting) atLeastAtMost_iff imageE path_image_def*)

show *?thesis*

proof

show *arc* (*subpath* 0 v h)

by (*metis (no_types) $\langle \text{pathstart } h = g \ u \rangle$ $\langle \text{simple_path } h \rangle$ *arc_simple_path_subpath* $\langle a \neq b \rangle$ *atLeastAtMost_iff zero_le_one order_refl pathstart_def* $u(3)$ v)*

show *arc* (*subpath* v 1 h)

by (*metis (no_types) $\langle \text{pathfinish } h = g \ u \rangle$ $\langle \text{simple_path } h \rangle$ *arc_simple_path_subpath* $\langle a \neq b \rangle$ *atLeastAtMost_iff zero_le_one order_refl pathfinish_def* $u(3)$ v)*

show *pathstart* (*subpath* 0 v h) = a

by (*metis $\langle \text{pathstart } h = g \ u \rangle$ pathstart_def pathstart_subpath* $u(3)$)

show *pathfinish* (*subpath* 0 v h) = b *pathstart* (*subpath* v 1 h) = b

by (*simp_all add: $v(3)$*)

show *pathfinish* (*subpath* v 1 h) = a

by (*metis $\langle \text{pathfinish } h = g \ u \rangle$ pathfinish_def pathfinish_subpath* $u(3)$)

show *path_image* (*subpath* 0 v h) \cap *path_image* (*subpath* v 1 h) = $\{a, b\}$

proof

have *loop_free* h

using $\langle \text{simple_path } h \rangle$ *simple_path_def* **by** *blast*

then show *path_image* (*subpath* 0 v h) \cap *path_image* (*subpath* v 1 h) $\subseteq \{a, b\}$

b}

using v $\langle \text{pathfinish } (\text{subpath } v \ 1 \ h) = a \rangle$

apply (*clarsimp simp add: loop_free_def path_image_subpath Ball_def*)

by (*smt (verit)*)

show $\{a, b\} \subseteq \text{path_image } (\text{subpath } 0 \ v \ h) \cap \text{path_image } (\text{subpath } v \ 1 \ h)$

using v $\langle \text{pathstart } (\text{subpath } 0 \ v \ h) = a \rangle$ $\langle \text{pathfinish } (\text{subpath } v \ 1 \ h) = a \rangle$

3688

```
      by (auto simp: path_image_subpath image_iff Bex_def)
    qed
  show path_image (subpath 0 v h)  $\cup$  path_image (subpath v 1 h) = path_image
  g
    using v path_image_subpath pihg path_image_def
    by (metis (full_types) image_Un ivl_disj_un_two_touch(4))
  qed
qed
```

theorem *Jordan_curve*:

fixes $c :: \text{real} \Rightarrow \text{complex}$

assumes *simple_path c and loop: pathfinish c = pathstart c*

obtains *inner outer where*

inner $\neq \{\}$ open inner connected inner

outer $\neq \{\}$ open outer connected outer

bounded inner \neg bounded outer inner \cap outer = $\{\}$

inner \cup outer = $-$ path_image c

frontier inner = path_image c

frontier outer = path_image c

proof $-$

have *path c*

by (*simp add: assms simple_path_imp_path*)

have *hom: (path_image c) homeomorphic (sphere(0::complex) 1)*

by (*simp add: assms homeomorphic_simple_path_image_circle*)

with *Jordan_Brouwer_separation* **have** \neg *connected $(- (path_image\ c))$*

by *fastforce*

then obtain *inner where inner: inner \in components $(- (path_image\ c))$ and bounded inner*

using *cobounded_has_bounded_component [of $- (path_image\ c)$]*

using $\langle \neg$ *connected $(- (path_image\ c))$ \rangle *simple_path c* *bounded_simple_path_image**

by *force*

obtain *outer where outer: outer \in components $(- (path_image\ c))$ and \neg bounded outer*

using *cobounded_unbounded_components [of $- (path_image\ c)$]*

using \langle *path c* *bounded_path_image* **by** *auto*

show *?thesis*

proof

show *inner $\neq \{\}$*

using *inner_in_components_nonempty* **by** *auto*

show *open inner*

by (*meson \langle simple_path c \rangle compact_imp_closed compact_simple_path_image inner open_Comp open_components*)

show *connected inner*

using *in_components_connected inner* **by** *blast*

show *outer $\neq \{\}$*

using *outer_in_components_nonempty* **by** *auto*

show *open outer*

by (*meson \langle simple_path c \rangle compact_imp_closed compact_simple_path_image*

```

outer open_Compl open_components)
  show connected outer
    using in_components_connected outer by blast
  show inner_outer: inner  $\cap$  outer = {}
  by (meson  $\langle \neg$  bounded outer  $\rangle$   $\langle$  bounded inner  $\rangle$   $\langle$  connected outer  $\rangle$  bounded_subset
  components_maximal in_components_subset inner outer)
  show fro_inner: frontier inner = path_image c
    by (simp add: Jordan_Brouwer_frontier [OF hom inner])
  show fro_outer: frontier outer = path_image c
    by (simp add: Jordan_Brouwer_frontier [OF hom outer])
  have False if m: middle  $\in$  components ( $-$  path_image c) and middle  $\neq$  inner
  middle  $\neq$  outer for middle
  proof -
    have frontier middle = path_image c
      by (simp add: Jordan_Brouwer_frontier [OF hom] that)
    obtain middle: open middle connected middle middle  $\neq$  {}
      by (metis fro_inner frontier_closed in_components_maximal m open_Compl
  open_components)
    obtain a0 b0 where a0  $\in$  path_image c b0  $\in$  path_image c a0  $\neq$  b0
      using simple_path_image_uncountable [OF  $\langle$  simple_path c  $\rangle$ ]
    by (metis Diff_cancel countable_Diff_eq countable_empty insert_iff subsetI
  subset_singleton_iff)
    obtain a b g where ab: a  $\in$  path_image c b  $\in$  path_image c a  $\neq$  b
      and g: arc g pathstart g = a pathfinish g = b
      and pag_sub: path_image g  $-$  {a,b}  $\subseteq$  middle
    proof (rule dense_accessible_frontier_point_pairs [OF  $\langle$  open middle  $\rangle$   $\langle$  con-
  nected middle  $\rangle$ , of path_image c  $\cap$  ball a0 (dist a0 b0) path_image c  $\cap$  ball b0
  (dist a0 b0)])
      show openin (top_of_set (frontier middle)) (path_image c  $\cap$  ball a0 (dist
  a0 b0))
        openin (top_of_set (frontier middle)) (path_image c  $\cap$  ball b0 (dist a0
  b0))
        by (simp_all add:  $\langle$  frontier middle = path_image c  $\rangle$  openin_open_Int)
      show path_image c  $\cap$  ball a0 (dist a0 b0)  $\neq$  path_image c  $\cap$  ball b0 (dist
  a0 b0)
        using  $\langle$  a0  $\neq$  b0  $\rangle$   $\langle$  b0  $\in$  path_image c  $\rangle$  by auto
        show path_image c  $\cap$  ball a0 (dist a0 b0)  $\neq$  {}
          using  $\langle$  a0  $\in$  path_image c  $\rangle$   $\langle$  a0  $\neq$  b0  $\rangle$  by auto
        show path_image c  $\cap$  ball b0 (dist a0 b0)  $\neq$  {}
          using  $\langle$  b0  $\in$  path_image c  $\rangle$   $\langle$  a0  $\neq$  b0  $\rangle$  by auto
    qed (use arc_distinct_ends arc_imp_simple_path simple_path_endless that
  in fastforce)
    obtain u d where arc u arc d
      and pathstart u = a pathfinish u = b pathstart d = b pathfinish d
  = a
      and ud_ab: (path_image u)  $\cap$  (path_image d) = {a,b}
      and ud_Un: (path_image u)  $\cup$  (path_image d) = path_image c
    using exists_double_arc [OF assms ab] by blast
    obtain x y where x  $\in$  inner y  $\in$  outer

```

```

using ⟨inner ≠ {}⟩ ⟨outer ≠ {}⟩ by auto
have inner ∩ middle = {} middle ∩ outer = {}
using components_nonoverlap inner outer m that by blast+
have connected_component (¬ (path_image u ∪ path_image g ∪ (path_image
d ∪ path_image g))) x y
proof (rule Janiszewski)
show compact (path_image u ∪ path_image g)
by (simp add: ⟨arc g⟩ ⟨arc u⟩ compact_Un compact_arc_image)
show closed (path_image d ∪ path_image g)
by (simp add: ⟨arc d⟩ ⟨arc g⟩ closed_Un closed_arc_image)
show connected ((path_image u ∪ path_image g) ∩ (path_image d ∪
path_image g))
using ud_ab
by (metis Un_insert_left g connected_arc_image insert_absorb pathfin-
ish_in_path_image pathstart_in_path_image sup_bot_left sup_commute sup_inf_distrib1)
show connected_component (¬ (path_image u ∪ path_image g)) x y
unfolding connected_component_def
proof (intro exI conjI)
have connected ((inner ∪ (path_image c - path_image u)) ∪ (outer ∪
(path_image c - path_image u)))
proof (rule connected_Un)
show connected (inner ∪ (path_image c - path_image u))
using connected_intermediate_closure [OF ⟨connected inner⟩]
by (metis Diff_subset closure_Un_frontier dual_order.refl fro_inner
sup_mono sup_ge1)
show connected (outer ∪ (path_image c - path_image u))
using connected_intermediate_closure [OF ⟨connected outer⟩]
by (simp add: Diff_eq closure_Un_frontier fro_outer sup_inf_distrib1)
have (inner ∩ outer) ∪ (path_image c - path_image u) ≠ {}
using ⟨arc d⟩ ⟨pathfinish d = a⟩ ⟨pathstart d = b⟩ arc_imp_simple_path
nonempty_simple_path_endless ud_Un ud_ab by fastforce
then show (inner ∪ (path_image c - path_image u)) ∩ (outer ∪
(path_image c - path_image u)) ≠ {}
by auto
qed
then show connected (inner ∪ outer ∪ (path_image c - path_image u))
by (metis sup.right_idem sup_assoc sup_commute)
have inner ⊆ - path_image u outer ⊆ - path_image u
using in_components_subset inner outer ud_Un by auto
moreover have inner ⊆ - path_image g outer ⊆ - path_image g
using ⟨inner ∩ middle = {}⟩ ⟨inner ⊆ - path_image u⟩
using ⟨middle ∩ outer = {}⟩ ⟨outer ⊆ - path_image u⟩ pag_sub ud_ab
by fastforce+
moreover have path_image c - path_image u ⊆ - path_image g
using in_components_subset m pag_sub ud_ab by fastforce
ultimately show inner ∪ outer ∪ (path_image c - path_image u) ⊆ -
(path_image u ∪ path_image g)
by force
show x ∈ inner ∪ outer ∪ (path_image c - path_image u)

```

```

    by (auto simp: ⟨x ∈ inner⟩)
  show y ∈ inner ∪ outer ∪ (path_image c - path_image u)
    by (auto simp: ⟨y ∈ outer⟩)
qed
show connected_component (-(path_image d ∪ path_image g)) x y
  unfolding connected_component_def
  proof (intro exI conjI)
    have connected ((inner ∪ (path_image c - path_image d)) ∪ (outer ∪
(path_image c - path_image d)))
    proof (rule connected_Un)
      show connected (inner ∪ (path_image c - path_image d))
        using connected_intermediate_closure [OF ⟨connected inner⟩] fro_inner
        by (simp add: closure_Un_frontier sup.coboundedI2)
      show connected (outer ∪ (path_image c - path_image d))
        using connected_intermediate_closure [OF ⟨connected outer⟩]
        by (simp add: closure_Un_frontier fro_outer sup.coboundedI2)
      have (inner ∩ outer) ∪ (path_image c - path_image d) ≠ {}
        using ⟨arc u⟩ ⟨pathfinish u = b⟩ ⟨pathstart u = a⟩ arc_imp_simple_path
nonempty_simple_path_endless ud_Un ud_ab by fastforce
      then show (inner ∪ (path_image c - path_image d)) ∩ (outer ∪
(path_image c - path_image d)) ≠ {}
        by auto
    qed
  then show connected (inner ∪ outer ∪ (path_image c - path_image d))
    by (metis sup.right_idem sup_assoc sup_commute)
  have inner ⊆ - path_image d outer ⊆ - path_image d
    using in_components_subset inner outer ud_Un by auto
  moreover have inner ⊆ - path_image g outer ⊆ - path_image g
    using ⟨inner ∩ middle = {}⟩ ⟨inner ⊆ - path_image d⟩
    using ⟨middle ∩ outer = {}⟩ ⟨outer ⊆ - path_image d⟩ pag_sub ud_ab
by fastforce+
  moreover have path_image c - path_image d ⊆ - path_image g
    using in_components_subset m pag_sub ud_ab by fastforce
  ultimately show inner ∪ outer ∪ (path_image c - path_image d) ⊆ -
(path_image d ∪ path_image g)
    by force
  show x ∈ inner ∪ outer ∪ (path_image c - path_image d)
    by (auto simp: ⟨x ∈ inner⟩)
  show y ∈ inner ∪ outer ∪ (path_image c - path_image d)
    by (auto simp: ⟨y ∈ outer⟩)
qed
qed
then have connected_component (-(path_image u ∪ path_image d ∪
path_image g)) x y
  by (simp add: Un_ac)
moreover have ¬(connected_component (-(path_image c)) x y)
  by (metis (no_types, lifting) ⟨¬ bounded outer⟩ ⟨bounded inner⟩ ⟨x ∈ inner⟩
⟨y ∈ outer⟩ componentsE connected_component_eq inner mem_Collect_eq outer)
ultimately show False

```

```

    by (auto simp: ud_Un [symmetric] connected_component_def)
  qed
  then have components (– path_image c) = {inner, outer}
    using inner outer by blast
  then have Union (components (– path_image c)) = inner ∪ outer
    by simp
  then show inner ∪ outer = – path_image c
    by auto
  qed (auto simp: ⟨bounded inner⟩ ⟨¬ bounded outer⟩)
qed

```

corollary *Jordan_disconnected:*

```

  fixes c :: real ⇒ complex
  assumes simple_path c pathfinish c = pathstart c
  shows ¬ connected(– path_image c)
  using Jordan_curve [OF assms]
  by (metis Jordan_Brouwer_separation assms homeomorphic_simple_path_image_circle
  zero_less_one)

```

corollary *Jordan_inside_outside:*

```

  fixes c :: real ⇒ complex
  assumes simple_path c pathfinish c = pathstart c
  shows inside(path_image c) ≠ {} ∧
    open(inside(path_image c)) ∧
    connected(inside(path_image c)) ∧
    outside(path_image c) ≠ {} ∧
    open(outside(path_image c)) ∧
    connected(outside(path_image c)) ∧
    bounded(inside(path_image c)) ∧
    ¬ bounded(outside(path_image c)) ∧
    inside(path_image c) ∩ outside(path_image c) = {} ∧
    inside(path_image c) ∪ outside(path_image c) =
    – path_image c ∧
    frontier(inside(path_image c)) = path_image c ∧
    frontier(outside(path_image c)) = path_image c

```

proof –

```

  obtain inner outer
  where *: inner ≠ {} open inner connected inner
    outer ≠ {} open outer connected outer
    bounded inner ¬ bounded outer inner ∩ outer = {}
    inner ∪ outer = – path_image c
    frontier inner = path_image c
    frontier outer = path_image c
  using Jordan_curve [OF assms] by blast
  then have inner: inside(path_image c) = inner
  by (metis dual_order.antisym inside_subset interior_eq interior_inside_frontier)
  have outer: outside(path_image c) = outer

```



```

    using ⟨inner ∪ outer = - path_image c⟩ ⟨inside (path_image c) = inner⟩
      outside_inside ⟨inner ∩ outer = {}⟩ by auto
  show ?thesis
    using * by (auto simp: inner outer)
qed

```

Triple-curve or "theta-curve" theorem

Proof that there is no fourth component taken from Kuratowski's Topology vol 2, para 61, II.

theorem *split_inside_simple_closed_curve*:

```

  fixes c :: real ⇒ complex
  assumes simple_path c1 and c1: pathstart c1 = a pathfinish c1 = b
    and simple_path c2 and c2: pathstart c2 = a pathfinish c2 = b
    and simple_path c and c: pathstart c = a pathfinish c = b
    and a ≠ b
    and c1c2: path_image c1 ∩ path_image c2 = {a,b}
    and c1c: path_image c1 ∩ path_image c = {a,b}
    and c2c: path_image c2 ∩ path_image c = {a,b}
    and ne_12: path_image c ∩ inside(path_image c1 ∪ path_image c2) ≠ {}
  obtains inside(path_image c1 ∪ path_image c) ∩ inside(path_image c2 ∪
    path_image c) = {}
    inside(path_image c1 ∪ path_image c) ∪ inside(path_image c2 ∪
    path_image c) ∪
    (path_image c - {a,b}) = inside(path_image c1 ∪ path_image c2)

```

proof –

```

  let ?Θ = path_image c let ?Θ1 = path_image c1 let ?Θ2 = path_image c2
  have sp: simple_path (c1 +++ reversepath c2) simple_path (c1 +++ re-
    versepath c) simple_path (c2 +++ reversepath c)
  using assms by (auto simp: simple_path_join_loop_eq arc_simple_path sim-
    ple_path_reversepath)
  then have op_in12: open (inside (?Θ1 ∪ ?Θ2))
    and op_out12: open (outside (?Θ1 ∪ ?Θ2))
    and op_in1c: open (inside (?Θ1 ∪ ?Θ))
    and op_in2c: open (inside (?Θ2 ∪ ?Θ))
    and op_out1c: open (outside (?Θ1 ∪ ?Θ))
    and op_out2c: open (outside (?Θ2 ∪ ?Θ))
    and co_in1c: connected (inside (?Θ1 ∪ ?Θ))
    and co_in2c: connected (inside (?Θ2 ∪ ?Θ))
    and co_out12c: connected (outside (?Θ1 ∪ ?Θ2))
    and co_out1c: connected (outside (?Θ1 ∪ ?Θ))
    and co_out2c: connected (outside (?Θ2 ∪ ?Θ))
    and pa_c: ?Θ - {pathstart c, pathfinish c} ⊆ - ?Θ1
      ?Θ - {pathstart c, pathfinish c} ⊆ - ?Θ2
    and pa_c1: ?Θ1 - {pathstart c1, pathfinish c1} ⊆ - ?Θ2
      ?Θ1 - {pathstart c1, pathfinish c1} ⊆ - ?Θ
    and pa_c2: ?Θ2 - {pathstart c2, pathfinish c2} ⊆ - ?Θ1
      ?Θ2 - {pathstart c2, pathfinish c2} ⊆ - ?Θ
    and co_c: connected(?Θ - {pathstart c, pathfinish c})

```

```

and co_c1: connected( $?\Theta 1 - \{\text{pathstart } c1, \text{pathfinish } c1\}$ )
and co_c2: connected( $?\Theta 2 - \{\text{pathstart } c2, \text{pathfinish } c2\}$ )
and fr_in: frontier(inside( $?\Theta 1 \cup ?\Theta 2$ )) =  $?\Theta 1 \cup ?\Theta 2$ 
      frontier(inside( $?\Theta 2 \cup ?\Theta$ )) =  $?\Theta 2 \cup ?\Theta$ 
      frontier(inside( $?\Theta 1 \cup ?\Theta$ )) =  $?\Theta 1 \cup ?\Theta$ 
and fr_out: frontier(outside( $?\Theta 1 \cup ?\Theta 2$ )) =  $?\Theta 1 \cup ?\Theta 2$ 
      frontier(outside( $?\Theta 2 \cup ?\Theta$ )) =  $?\Theta 2 \cup ?\Theta$ 
      frontier(outside( $?\Theta 1 \cup ?\Theta$ )) =  $?\Theta 1 \cup ?\Theta$ 
using Jordan_inside_outside [of c1 +++ reversepath c2]
using Jordan_inside_outside [of c1 +++ reversepath c]
using Jordan_inside_outside [of c2 +++ reversepath c] assms
      apply (simp_all add: path_image_join closed_Un closed_simple_path_image
open_inside open_outside)
      apply (blast | metis connected_simple_path_endless)+
    done
have inout_12: inside ( $?\Theta 1 \cup ?\Theta 2$ )  $\cap$  ( $?\Theta - \{\text{pathstart } c, \text{pathfinish } c\}$ )  $\neq \{\}$ 
by (metis (no_types, lifting) c c1c ne_12 Diff_Int_distrib Diff_empty Int_empty_right
Int_left_commute inf_sup_absorb inf_sup_aci(1) inside_no_overlap)
have pi_disjoint:  $?\Theta \cap \text{outside}(\Theta 1 \cup \Theta 2) = \{\}$ 
proof (rule ccontr)
  assume  $?\Theta \cap \text{outside}(\Theta 1 \cup \Theta 2) \neq \{\}$ 
  then show False
    using connectedD [OF co_c, of inside( $?\Theta 1 \cup ?\Theta 2$ ) outside( $?\Theta 1 \cup ?\Theta 2$ )]
    using c c1c2 pa_c op_in12 op_out12 inout_12
    apply clarsimp
    by (smt (verit, ccfv_threshold) Diff_Int_distrib Diff_cancel Diff_empty
Int_assoc inf_sup_absorb inf_sup_aci(1) outside_no_overlap)
  qed
have out_sub12: outside( $?\Theta 1 \cup ?\Theta 2$ )  $\subseteq$  outside( $?\Theta 1 \cup ?\Theta$ ) outside( $?\Theta 1 \cup$ 
 $?\Theta 2$ )  $\subseteq$  outside( $?\Theta 2 \cup ?\Theta$ )
by (metis Un_commute pi_disjoint outside_Un_outside_Un)+
have pa1_disj_in2:  $?\Theta 1 \cap \text{inside}(\Theta 2 \cup \Theta) = \{\}$ 
proof (rule ccontr)
  assume ne:  $?\Theta 1 \cap \text{inside}(\Theta 2 \cup \Theta) \neq \{\}$ 
  have 1: inside ( $?\Theta \cup ?\Theta 2$ )  $\cap$   $?\Theta = \{\}$ 
  by (metis (no_types) Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci(3)
inside_no_overlap)
  have 2: outside ( $?\Theta \cup ?\Theta 2$ )  $\cap$   $?\Theta = \{\}$ 
  by (metis (no_types) Int_empty_right Int_left_commute inf_sup_absorb
outside_no_overlap)
  have path_image c1  $\cap$  outside (path_image c2  $\cup$  path_image c) =  $\{\}$ 
  using connectedD [OF co_c1, of inside( $?\Theta 2 \cup ?\Theta$ ) outside( $?\Theta 2 \cup ?\Theta$ )]
  pa_c1 op_in2c op_out2c ne c1 c2c 1 2 by (auto simp: inf_sup_aci)
then have outside ( $?\Theta 2 \cup ?\Theta$ )  $\subseteq$  outside ( $?\Theta 1 \cup ?\Theta 2$ )
  by (metis outside_Un_outside_Un sup_commute)
with out_sub12
have outside( $?\Theta 1 \cup ?\Theta 2$ ) = outside( $?\Theta 2 \cup ?\Theta$ ) by blast
then have frontier(outside( $?\Theta 1 \cup ?\Theta 2$ )) = frontier(outside( $?\Theta 2 \cup ?\Theta$ ))
  by simp

```

```

    then show False
      using inout_12 pi_disjoint c c1c c2c fr_out by auto
    qed
    have pa2_disj_in1:  $?\Theta 2 \cap \text{inside} (? \Theta 1 \cup ? \Theta) = \{\}$ 
    proof (rule ccontr)
      assume ne:  $? \Theta 2 \cap \text{inside} (? \Theta 1 \cup ? \Theta) \neq \{\}$ 
      have 1:  $\text{inside} (? \Theta \cup ? \Theta 1) \cap ? \Theta = \{\}$ 
      by (metis (no_types) Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci(3)
inside_no_overlap)
      have 2:  $\text{outside} (? \Theta \cup ? \Theta 1) \cap ? \Theta = \{\}$ 
      by (metis (no_types) Int_empty_right Int_left_commute inf_sup_absorb
outside_no_overlap)
      have  $\text{outside} (? \Theta 1 \cup ? \Theta) \subseteq \text{outside} (? \Theta 1 \cup ? \Theta 2)$ 
      apply (rule outside_Un_outside_Un)
      using connectedD [OF co_c2, of inside (? \Theta 1 \cup ? \Theta) outside (? \Theta 1 \cup ? \Theta)]
      pa_c2 op_in1c op_out1c ne c2 c1c 1 2 by (auto simp: inf_sup_aci)
    with out_sub12
    have  $\text{outside} (? \Theta 1 \cup ? \Theta 2) = \text{outside} (? \Theta 1 \cup ? \Theta)$ 
      by blast
    then have  $\text{frontier}(\text{outside} (? \Theta 1 \cup ? \Theta 2)) = \text{frontier}(\text{outside} (? \Theta 1 \cup ? \Theta))$ 
      by simp
    then show False
      using inout_12 pi_disjoint c c1c c2c fr_out by auto
    qed
    have in_sub_in1:  $\text{inside} (? \Theta 1 \cup ? \Theta) \subseteq \text{inside} (? \Theta 1 \cup ? \Theta 2)$ 
      using pa2_disj_in1 out_sub12 by (auto simp: inside_outside)
    have in_sub_in2:  $\text{inside} (? \Theta 2 \cup ? \Theta) \subseteq \text{inside} (? \Theta 1 \cup ? \Theta 2)$ 
      using pa1_disj_in2 out_sub12 by (auto simp: inside_outside)
    have in_sub_out12:  $\text{inside} (? \Theta 1 \cup ? \Theta) \subseteq \text{outside} (? \Theta 2 \cup ? \Theta)$ 
    proof
      fix x
      assume x:  $x \in \text{inside} (? \Theta 1 \cup ? \Theta)$ 
      then have xnot:  $x \notin ? \Theta$ 
      by (simp add: inside_def)
      obtain z where zim:  $z \in ? \Theta 1$  and zout:  $z \in \text{outside} (? \Theta 2 \cup ? \Theta)$ 
      unfolding outside_inside
      using nonempty_simple_path_endless [OF  $\langle \text{simple\_path } c1 \rangle$ ] c1 c1c c1c2
pa1_disj_in2 by auto
      obtain e where  $e > 0$  and e:  $\text{ball } z \ e \subseteq \text{outside} (? \Theta 2 \cup ? \Theta)$ 
      using zout op_out2c open_contains_ball_eq by blast
      have  $z \in \text{frontier} (\text{inside} (? \Theta 1 \cup ? \Theta))$ 
      using zim by (auto simp: fr_in)
      then obtain w where w1:  $w \in \text{inside} (? \Theta 1 \cup ? \Theta)$  and dwz:  $\text{dist } w \ z < e$ 
      using zim  $\langle e > 0 \rangle$  by (auto simp: frontier_def closure_approachable)
      then have w2:  $w \in \text{outside} (? \Theta 2 \cup ? \Theta)$ 
      by (metis e dist_commute mem_ball subsetCE)
      then have connected_component  $(- ? \Theta 2 \cap - ? \Theta) \ z \ w$ 
      unfolding connected_component_def
      by (metis co_out2c compl_sup inside_Un_outside sup_ge2 zout)
    
```

```

moreover have connected_component ( $- \ ?\Theta 2 \cap - \ ?\Theta$ )  $w \ x$ 
  unfolding connected_component_def
  using pa2_disj_in1 co_in1c x w1 union_with_outside by fastforce
ultimately have eq: connected_component_set ( $- \ ?\Theta 2 \cap - \ ?\Theta$ )  $x =$ 
  connected_component_set ( $- \ ?\Theta 2 \cap - \ ?\Theta$ )  $z$ 
  by (metis (mono_tags, lifting) connected_component_eq mem_Collect_eq)
show  $x \in \text{outside} \ (\ ?\Theta 2 \cup \ ?\Theta)$ 
  using zout x pa2_disj_in1 by (auto simp: outside_def eq xnot)
qed
have in_sub_out21: inside( $\ ?\Theta 2 \cup \ ?\Theta$ )  $\subseteq$  outside( $\ ?\Theta 1 \cup \ ?\Theta$ )
proof
  fix  $x$ 
  assume  $x: x \in \text{inside} \ (\ ?\Theta 2 \cup \ ?\Theta)$ 
  then have xnot: x  $\notin \ ?\Theta$ 
    by (simp add: inside_def)
  obtain  $z$  where zim: z  $\in \ ?\Theta 2$  and zout: z  $\in \ \text{outside}(\ ?\Theta 1 \cup \ ?\Theta)$ 
    unfolding outside_inside
    using nonempty_simple_path_endless [OF  $\langle \text{simple\_path } c2 \rangle$ ] c1c2 c2 c2c
pa2_disj_in1 by auto
  obtain  $e$  where  $e > 0$  and  $e: \text{ball } z \ e \subseteq \ \text{outside}(\ ?\Theta 1 \cup \ ?\Theta)$ 
    using zout op_out1c open_contains_ball_eq by blast
  have  $z \in \text{frontier} \ (\ \text{inside} \ (\ ?\Theta 2 \cup \ ?\Theta))$ 
    using zim by (auto simp: fr_in)
  then obtain  $w$  where  $w2: w \in \text{inside} \ (\ ?\Theta 2 \cup \ ?\Theta)$  and dwz: dist  $w \ z < e$ 
    using zim  $\langle e > 0 \rangle$  by (auto simp: frontier_def closure_approachable)
  then have  $w1: w \in \text{outside} \ (\ ?\Theta 1 \cup \ ?\Theta)$ 
    by (metis  $e$  dist_commute mem_ball subsetCE)
  then have connected_component ( $- \ ?\Theta 1 \cap - \ ?\Theta$ )  $z \ w$ 
    unfolding connected_component_def
    by (metis Compl_Un co_out1c inside_Un_outside sup_ge2 zout)
  moreover have connected_component ( $- \ ?\Theta 1 \cap - \ ?\Theta$ )  $w \ x$ 
    unfolding connected_component_def
    using pa1_disj_in2 co_in2c x w2 union_with_outside by fastforce
  ultimately have eq: connected_component_set ( $- \ ?\Theta 1 \cap - \ ?\Theta$ )  $x =$ 
  connected_component_set ( $- \ ?\Theta 1 \cap - \ ?\Theta$ )  $z$ 
    by (metis (no_types, lifting) connected_component_eq mem_Collect_eq)
  show  $x \in \text{outside} \ (\ ?\Theta 1 \cup \ ?\Theta)$ 
    using zout x pa1_disj_in2 by (auto simp: outside_def eq xnot)
qed
show ?thesis
proof
  show inside ( $\ ?\Theta 1 \cup \ ?\Theta$ )  $\cap$  inside ( $\ ?\Theta 2 \cup \ ?\Theta$ )  $= \ \{\}$ 
    by (metis Int_Un_distrib in_sub_out12 bot_eq_sup_iff disjoint_eq_subset_Compl
outside_inside)
  have  $*$ : outside ( $\ ?\Theta 1 \cup \ ?\Theta$ )  $\cap$  outside ( $\ ?\Theta 2 \cup \ ?\Theta$ )  $\subseteq$  outside ( $\ ?\Theta 1 \cup \ ?\Theta 2$ )
  proof (rule components_maximal)
    show out_in: outside ( $\ ?\Theta 1 \cup \ ?\Theta 2$ )  $\in$  components ( $- \ (\ ?\Theta 1 \cup \ ?\Theta 2)$ )
      unfolding outside_in_components co_out12c
      using co_out12c fr_out(1) by force

```

```

have conn_U: connected (– (closure (inside (?Θ1 ∪ ?Θ)) ∪ closure (inside
(?Θ2 ∪ ?Θ))))
proof (rule Janiszewski_connected, simp_all)
  show bounded (inside (?Θ1 ∪ ?Θ))
  by (simp add: ‹simple_path c1› ‹simple_path c› bounded_inside bounded_simple_path_image)
  have if1: – (inside (?Θ1 ∪ ?Θ) ∪ frontier (inside (?Θ1 ∪ ?Θ))) = – ?Θ1
∩ – ?Θ ∩ – inside (?Θ1 ∪ ?Θ)
    by (metis (no_types, lifting) Int_commute Jordan_inside_outside c
c1 compl_sup_path_image_join path_image_reversepath pathfinish_join pathfin-
ish_reversepath pathstart_join pathstart_reversepath sp(2) closure_Un_frontier
fr_out(3))
  then show connected (– closure (inside (?Θ1 ∪ ?Θ)))
    by (metis Compl_Un outside_inside co_out1c closure_Un_frontier)
  have if2: – (inside (?Θ2 ∪ ?Θ) ∪ frontier (inside (?Θ2 ∪ ?Θ))) = – ?Θ2
∩ – ?Θ ∩ – inside (?Θ2 ∪ ?Θ)
    by (metis (no_types, lifting) Int_commute Jordan_inside_outside c
c2 compl_sup_path_image_join path_image_reversepath pathfinish_join pathfin-
ish_reversepath pathstart_join pathstart_reversepath sp(3) closure_Un_frontier
fr_out(2))
  then show connected (– closure (inside (?Θ2 ∪ ?Θ)))
    by (metis Compl_Un outside_inside co_out2c closure_Un_frontier)
  have connected(?Θ)
    by (metis ‹simple_path c› connected_simple_path_image)
  moreover
  have closure (inside (?Θ1 ∪ ?Θ)) ∩ closure (inside (?Θ2 ∪ ?Θ)) = ?Θ
(is ?lhs = ?rhs)
  proof
    show ?lhs ⊆ ?rhs
    proof clarify
      fix x
      assume x: x ∈ closure (inside (?Θ1 ∪ ?Θ)) x ∈ closure (inside (?Θ2 ∪
?Θ))
      then have x ∉ inside (?Θ1 ∪ ?Θ)
      by (meson closure_iff_nhds_not_empty_in_sub_out12 inside_Int_outside
op_in1c)
      with fr_in x show x ∈ ?Θ
      by (metis c1c c1c2 closure_Un_frontier pa1_disj_in2 Int_iff Un_iff
insert_disjoint(2) insert_subset subsetI subset_antisym)
    qed
    show ?rhs ⊆ ?lhs
    using if1 if2 closure_Un_frontier by fastforce
  qed
  ultimately
  show connected (closure (inside (?Θ1 ∪ ?Θ)) ∩ closure (inside (?Θ2 ∪
?Θ)))
    by auto
  qed
  show connected (outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ))
    using fr_in conn_U by (simp add: closure_Un_frontier outside_inside

```

```

Un_commute)
  show outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ) ⊆ - (?Θ1 ∪ ?Θ2)
  by clarify (metis Diff_Cmpl Diff_iff Un_iff inf_sup_absorb outside_inside)
  show outside (?Θ1 ∪ ?Θ2) ∩
    (outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ)) ≠ {}
  by (metis Int_assoc out_in inf.orderE out_sub12(1) out_sub12(2) out-
side_in_components)
  qed
  show inside (?Θ1 ∪ ?Θ) ∪ inside (?Θ2 ∪ ?Θ) ∪ (?Θ - {a, b}) = inside (?Θ1
∪ ?Θ2)
  (is ?lhs = ?rhs)
  proof
  have path_image c - {a, b} ⊆ inside (path_image c1 ∪ path_image c2)
  using c1c c2c inside_outside pi_disjoint by fastforce
  then show ?lhs ⊆ ?rhs
  by (simp add: in_sub_in1 in_sub_in2)
  have inside (?Θ1 ∪ ?Θ2) ⊆ inside (?Θ1 ∪ ?Θ) ∪ inside (?Θ2 ∪ ?Θ) ∪ (?Θ)
  using Cmpl_anti_mono [OF *] by (force simp: inside_outside)
  moreover have inside (?Θ1 ∪ ?Θ2) ⊆ -{a,b}
  using c1 union_with_outside by fastforce
  ultimately show ?rhs ⊆ ?lhs by auto
  qed
  qed
  qed
end

```

9.29 Polynomial Functions: Extremal Behaviour and Root Counts

```

theory Poly_Roots
imports Complex_Main
begin

```

9.29.1 Basics about polynomial functions: extremal behaviour and root counts

```

lemma sub_polyfun:
  fixes x :: 'a::{comm_ring,monoid_mult}
  shows (∑ i≤n. a i * xi) - (∑ i≤n. a i * yi) =
    (x - y) * (∑ j<n. ∑ k= Suc j..n. a k * y(k - Suc j) * xj)
  proof -
  have (∑ i≤n. a i * xi) - (∑ i≤n. a i * yi) =
    (∑ i≤n. a i * (xi - yi))
  by (simp add: algebra_simps sum_subtractf [symmetric])
  also have ... = (∑ i≤n. a i * (x - y) * (∑ j<i. y(i - Suc j) * xj))
  by (simp add: power_diff_sumr2 ac_simps)
  also have ... = (x - y) * (∑ i≤n. (∑ j<i. a i * y(i - Suc j) * xj))

```

by (simp add: sum_distrib_left ac_simps)
 also have ... = $(x - y) * (\sum j < n. (\sum i = \text{Suc } j..n. a \ i * y^{i - \text{Suc } j} * x^j))$
 by (simp add: sum.nested_swap')
 finally show ?thesis .
 qed

lemma sub_polyfun_alt:

fixes $x :: 'a :: \{\text{comm_ring, monoid_mult}\}$
 shows $(\sum i \leq n. a \ i * x^i) - (\sum i \leq n. a \ i * y^i) =$
 $(x - y) * (\sum j < n. \sum k < n - j. a \ (j + k + 1) * y^k * x^j)$
 proof -
 { fix j
 have $(\sum k = \text{Suc } j..n. a \ k * y^{k - \text{Suc } j} * x^j) =$
 $(\sum k < n - j. a \ (\text{Suc } (j + k)) * y^k * x^j)$
 by (rule sum.reindex_bij_witness[where $i = \lambda i. i + \text{Suc } j$ and $j = \lambda i. i - \text{Suc } j$]) auto }
 then show ?thesis
 by (simp add: sub_polyfun)
 qed

lemma polyfun_linear_factor:

fixes $a :: 'a :: \{\text{comm_ring, monoid_mult}\}$
 shows $\exists b. \forall z. (\sum i \leq n. c \ i * z^i) =$
 $(z - a) * (\sum i < n. b \ i * z^i) + (\sum i \leq n. c \ i * a^i)$
 proof -
 { fix z
 have $(\sum i \leq n. c \ i * z^i) - (\sum i \leq n. c \ i * a^i) =$
 $(z - a) * (\sum j < n. (\sum k = \text{Suc } j..n. c \ k * a^{k - \text{Suc } j}) * z^j)$
 by (simp add: sub_polyfun sum_distrib_right)
 then have $(\sum i \leq n. c \ i * z^i) =$
 $(z - a) * (\sum j < n. (\sum k = \text{Suc } j..n. c \ k * a^{k - \text{Suc } j}) * z^j)$
 $+ (\sum i \leq n. c \ i * a^i)$
 by (simp add: algebra_simps) }
 then show ?thesis
 by (intro exI allI)
 qed

lemma polyfun_linear_factor_root:

fixes $a :: 'a :: \{\text{comm_ring, monoid_mult}\}$
 assumes $(\sum i \leq n. c \ i * a^i) = 0$
 shows $\exists b. \forall z. (\sum i \leq n. c \ i * z^i) = (z - a) * (\sum i < n. b \ i * z^i)$
 using polyfun_linear_factor [of $c \ n \ a$] assms
 by simp

lemma adhoc_norm_triangle: $a + \text{norm}(y) \leq b \implies \text{norm}(x) \leq a \implies \text{norm}(x + y) \leq b$

by (metis norm_triangle_mono order.trans order_refl)

proposition polyfun_extremal_lemma:

3700

```

fixes c :: nat => 'a::real_normed_div_algebra
assumes e > 0
shows  $\exists M. \forall z. M \leq \text{norm } z \longrightarrow \text{norm}(\sum_{i \leq n}. c \ i * z^{\wedge} i) \leq e * \text{norm}(z)^{\wedge} \text{Suc } n$ 
proof (induction n)
  case 0
  show ?case
  by (rule exI [where x=norm (c 0) / e]) (auto simp: mult.commute pos_divide_le_eq
  assms)
next
  case (Suc n)
  then obtain M where M:  $\forall z. M \leq \text{norm } z \longrightarrow \text{norm}(\sum_{i \leq n}. c \ i * z^{\wedge} i) \leq e$ 
  * norm z ^ Suc n ..
  show ?case
  proof (rule exI [where x=max 1 (max M ((e + norm(c(Suc n))) / e)], clarify)
  fix z::'a
  assume max 1 (max M ((e + norm (c (Suc n))) / e))  $\leq$  norm z
  then have norm1:  $0 < \text{norm } z \ M \leq \text{norm } z (e + \text{norm} (c (Suc n))) / e \leq$ 
  norm z
  by auto
  then have norm2:  $(e + \text{norm} (c (Suc n))) \leq e * \text{norm } z (\text{norm } z * \text{norm } z^{\wedge} n) > 0$ 
  apply (metis assms less_divide_eq mult.commute not_le)
  using norm1 apply (metis mult_pos_pos zero_less_power)
  done
  have  $e * (\text{norm } z * \text{norm } z^{\wedge} n) + \text{norm} (c (Suc n) * (z * z^{\wedge} n)) =$ 
   $(e + \text{norm} (c (Suc n))) * (\text{norm } z * \text{norm } z^{\wedge} n)$ 
  by (simp add: norm_mult norm_power algebra_simps)
  also have ...  $\leq (e * \text{norm } z) * (\text{norm } z * \text{norm } z^{\wedge} n)$ 
  using norm2
  using assms mult_mono by fastforce
  also have ...  $= e * (\text{norm } z * (\text{norm } z * \text{norm } z^{\wedge} n))$ 
  by (simp add: algebra_simps)
  finally have  $e * (\text{norm } z * \text{norm } z^{\wedge} n) + \text{norm} (c (Suc n) * (z * z^{\wedge} n))$ 
   $\leq e * (\text{norm } z * (\text{norm } z * \text{norm } z^{\wedge} n)) .$ 
  then show norm  $(\sum_{i \leq \text{Suc } n}. c \ i * z^{\wedge} i) \leq e * \text{norm } z^{\wedge} \text{Suc} (\text{Suc } n)$  using
  M norm1
  by (drule_tac x=z in spec) (auto simp: intro!: adhoc_norm_triangle)
  qed
qed

lemma norm_lemma_xy: assumes  $|b| + 1 \leq \text{norm}(y) - a \ \text{norm}(x) \leq a$  shows
 $b \leq \text{norm}(x + y)$ 
proof -
  have  $b \leq \text{norm } y - \text{norm } x$ 
  using assms by linarith
  then show ?thesis
  by (metis (no_types) add.commute norm_diff_ineq order_trans)
qed

```


proposition *polyfun_extremal*:

fixes $c :: \text{nat} \Rightarrow 'a::\text{real_normed_div_algebra}$

assumes $\exists k. k \neq 0 \wedge k \leq n \wedge c\ k \neq 0$

shows *eventually* $(\lambda z. \text{norm}(\sum_{i \leq n}. c\ i * z^i) \geq B)$ *at_infinity*

using *assms*

proof (*induction n*)

case 0 **then show** *?case*

by *simp*

next

case (*Suc n*)

show *?case*

proof (*cases c (Suc n) = 0*)

case True

with *Suc* **show** *?thesis*

by *auto* (*metis diff_is_0_eq diffs0_imp_equal less_Suc_eq_le not_less_eq*)

next

case False

with *polyfun_extremal_lemma* [*of norm(c (Suc n)) / 2 c n*]

obtain *M* **where** $M: \bigwedge z. M \leq \text{norm } z \implies$

$\text{norm}(\sum_{i \leq n}. c\ i * z^i) \leq \text{norm}(c(Suc\ n)) / 2 * \text{norm } z ^{Suc\ n}$

by *auto*

show *?thesis*

unfolding *eventually_at_infinity*

proof (*rule exI* [**where** $x = \max 1 ((|B| + 1) / (\text{norm}(c(Suc\ n)) / 2))$], *clarsimp*)

fix $z::'a$

assume *les*: $M \leq \text{norm } z \wedge 1 \leq \text{norm } z \wedge (|B| * 2 + 2) / \text{norm}(c(Suc\ n)) \leq \text{norm } z$

then have $|B| * 2 + 2 \leq \text{norm } z * \text{norm}(c(Suc\ n))$

by (*metis False pos_divide_le_eq zero_less_norm_iff*)

then have $|B| * 2 + 2 \leq \text{norm } z ^{Suc\ n} * \text{norm}(c(Suc\ n))$

by (*metis* $\langle 1 \leq \text{norm } z \rangle$ *order.trans mult_right_mono norm_ge_zero self_le_power zero_less_Suc*)

then show $B \leq \text{norm}((\sum_{i \leq n}. c\ i * z^i) + c(Suc\ n) * (z * z ^ n))$ **using** *M les*

apply *auto*

apply (*rule norm_lemma_xy* [**where** $a = \text{norm}(c(Suc\ n)) * \text{norm } z ^{Suc\ n} / 2$])

apply (*simp_all add: norm_mult norm_power*)

done

qed

qed

qed

proposition *polyfun_rootbound*:

fixes $c :: \text{nat} \Rightarrow 'a::\{\text{comm_ring, real_normed_div_algebra}\}$

assumes $\exists k. k \leq n \wedge c\ k \neq 0$

shows *finite* $\{z. (\sum_{i \leq n}. c\ i * z^i) = 0\} \wedge \text{card } \{z. (\sum_{i \leq n}. c\ i * z^i) = 0\}$

3702

```

< n
using assms
proof (induction n arbitrary: c)
  case (Suc n) show ?case
  proof (cases {z. (∑ i≤Suc n. c i * zi) = 0} = {})
    case False
    then obtain a where a: (∑ i≤Suc n. c i * ai) = 0
    by auto
    from polyfun_linear_factor_root [OF this]
    obtain b where ∧z. (∑ i≤Suc n. c i * zi) = (z - a) * (∑ i< Suc n. b i *
zi)
    by auto
    then have b: ∧z. (∑ i≤Suc n. c i * zi) = (z - a) * (∑ i≤n. b i * zi)
    by (metis lessThan_Suc_atMost)
    then have ins_ab: {z. (∑ i≤Suc n. c i * zi) = 0} = insert a {z. (∑ i≤n. b i
* zi) = 0}
    by auto
    have c0: c 0 = - (a * b 0) using b [of 0]
    by simp
    then have extr_prem: ¬ (∃ k≤n. b k ≠ 0) ⇒ ∃ k. k ≠ 0 ∧ k ≤ Suc n ∧ c k
≠ 0
    by (metis Suc.prem le0 minus_zero mult_zero_right)
    have ∃ k≤n. b k ≠ 0
    apply (rule ccontr)
    using polyfun_extremal [OF extr_prem, of 1]
    apply (auto simp: eventually_at_infinity b simp del: sum.atMost_Suc)
    apply (drule_tac x=of_real ba in spec, simp)
    done
    then show ?thesis using Suc.IH [of b] ins_ab
    by (auto simp: card_insert_if)
  qed simp
qed simp

```

corollary

```

fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
assumes ∃ k. k ≤ n ∧ c k ≠ 0
  shows polyfun_rootbound_finite: finite {z. (∑ i≤n. c i * zi) = 0}
  and polyfun_rootbound_card: card {z. (∑ i≤n. c i * zi) = 0} ≤ n
using polyfun_rootbound [OF assms] by auto

```

proposition polyfun_finite_roots:

```

fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
  shows finite {z. (∑ i≤n. c i * zi) = 0} ↔ (∃ k. k ≤ n ∧ c k ≠ 0)
proof (cases ∃ k≤n. c k ≠ 0)
  case True then show ?thesis
  by (blast intro: polyfun_rootbound_finite)
next
  case False then show ?thesis
  by (auto simp: infinite_UNIV_char_0)

```

qed

lemma *polyfun_eq_0*:

fixes $c :: \text{nat} \Rightarrow 'a::\{\text{comm_ring, real_normed_div_algebra}\}$

shows $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = 0) \longleftrightarrow (\forall k. k \leq n \longrightarrow c\ k = 0)$

proof (cases $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = 0)$)

case *True*

then have $\neg \text{finite } \{z. (\sum_{i \leq n}. c\ i * z^i) = 0\}$

by (simp add: *infinite_UNIV_char_0*)

with *True* show ?thesis

by (metis (*poly_guards_query*) *polyfun_rootbound_finite*)

next

case *False*

then show ?thesis

by *auto*

qed

theorem *polyfun_eq_const*:

fixes $c :: \text{nat} \Rightarrow 'a::\{\text{comm_ring, real_normed_div_algebra}\}$

shows $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = k) \longleftrightarrow c\ 0 = k \wedge (\forall k. k \neq 0 \wedge k \leq n \longrightarrow c\ k = 0)$

proof -

{fix z

have $(\sum_{i \leq n}. c\ i * z^i) = (\sum_{i \leq n}. (\text{if } i = 0 \text{ then } c\ 0 - k \text{ else } c\ i) * z^i) + k$

by (*induct n*) *auto*

} then

have $(\forall z. (\sum_{i \leq n}. c\ i * z^i) = k) \longleftrightarrow (\forall z. (\sum_{i \leq n}. (\text{if } i = 0 \text{ then } c\ 0 - k \text{ else } c\ i) * z^i) = 0)$

by *auto*

also have ... $\longleftrightarrow c\ 0 = k \wedge (\forall k. k \neq 0 \wedge k \leq n \longrightarrow c\ k = 0)$

by (*auto simp: polyfun_eq_0*)

finally show ?thesis .

qed

end

9.30 Generalised Binomial Theorem

The proof of the Generalised Binomial Theorem and related results. We prove the generalised binomial theorem for complex numbers, following the proof at: https://proofwiki.org/wiki/Binomial_Theorem/General_Binomial_Theorem

theory *Generalised_Binomial_Theorem*

imports

Complex_Main

Complex_Transcendental

Summation_Tests

begin

```

lemma gbinomial_ratio_limit:
  fixes a :: 'a :: real_normed_field
  assumes a ∉ ℕ
  shows (λn. (a gchoose n) / (a gchoose Suc n)) ⟶ -1
proof (rule Lim_transform_eventually)
  let ?f = λn. inverse (a / of_nat (Suc n) - of_nat n / of_nat (Suc n))
  from eventually_gt_at_top[of 0::nat]
  show eventually (λn. ?f n = (a gchoose n) / (a gchoose Suc n)) sequentially
proof eventually_elim
  fix n :: nat assume n: n > 0
  then obtain q where q: n = Suc q by (cases n) blast
  let ?P = ∏ i=0..

```

```

next
  assume a: a  $\notin$   $\mathbb{N}$ 
  from tendsto_norm[OF gbinomial_ratio_limit[OF this]]
  have conv_radius  $(\lambda n. a \text{ gchoose } n) = 1$ 
  by (intro conv_radius_ratio_limit_nonzero[of _ 1]) (simp_all add: norm_divide)
  with a show ?thesis by simp
qed

```

theorem *gen_binomial_complex*:

```

  fixes z :: complex
  assumes norm z < 1
  shows  $(\lambda n. (a \text{ gchoose } n) * z^n)$  sums  $(1 + z)$  powr a
  proof -
    define K where  $K = 1 - (1 - \text{norm } z) / 2$ 
    from assms have K:  $K > 0 \ K < 1 \ \text{norm } z < K$ 
    unfolding K_def by (auto simp: field_simps intro!: add_pos_nonneg)
    let ?f =  $\lambda n. a \text{ gchoose } n$  and ?f' =  $\text{diffs } (\lambda n. a \text{ gchoose } n)$ 
    have summable_strong: summable  $(\lambda n. ?f n * z^n)$  if  $\text{norm } z < 1$  for z using
    that
    by (intro summable_in_conv_radius) (simp_all add: conv_radius_gchoose)
    with K have summable: summable  $(\lambda n. ?f n * z^n)$  if  $\text{norm } z < K$  for z using
    that by auto
    hence summable': summable  $(\lambda n. ?f' n * z^n)$  if  $\text{norm } z < K$  for z using that
    by (intro termdiff_converges[of _ K]) simp_all

```

```

  define f f' where [abs_def]:  $f z = (\sum n. ?f n * z^n)$   $f' z = (\sum n. ?f' n * z^n)$ 
  for z

```

```

  {
    fix z :: complex assume z:  $\text{norm } z < K$ 
    from summable_mult2[OF summable'[OF z], of z]
    have summable1: summable  $(\lambda n. ?f' n * z^{Suc n})$  by (simp add: mult_ac)
    hence summable2: summable  $(\lambda n. \text{of\_nat } n * ?f n * z^n)$ 
    unfolding diffs_def by (subst (asm) summable_Suc_iff)

```

```

  have  $(1 + z) * f' z = (\sum n. ?f' n * z^n) + (\sum n. ?f' n * z^{Suc n})$ 
  unfolding f'_def using summable' z by (simp add: algebra_simps sum-
  inf_mult)

```

```

  also have  $(\sum n. ?f' n * z^n) = (\sum n. \text{of\_nat } (Suc n) * ?f (Suc n) * z^n)$ 
  by (intro suminf_cong) (simp add: diffs_def)
  also have  $(\sum n. ?f' n * z^{Suc n}) = (\sum n. \text{of\_nat } n * ?f n * z^n)$ 
  using summable1 suminf_split_initial_segment[OF summable1] unfolding
  diffs_def

```

```

  by (subst suminf_split_head, subst (asm) summable_Suc_iff) simp_all
  also have  $(\sum n. \text{of\_nat } (Suc n) * ?f (Suc n) * z^n) + (\sum n. \text{of\_nat } n * ?f n
  * z^n) =$ 

```

```

   $(\sum n. a * ?f n * z^n)$ 
  by (subst gbinomial_mult_1, subst suminf_add)
  (insert summable'[OF z] summable2,
  simp_all add: summable_powser_split_head algebra_simps diffs_def)

```

```

    also have ... = a * f z unfolding f_f'_def
      by (subst suminf_mult[symmetric]) (simp_all add: summable[OF z] mult_ac)
    finally have a * f z = (1 + z) * f' z by simp
  } note deriv = this

  have [derivative_intros]: (f has_field_derivative f' z) (at z) if norm z < of_real
  K for z
    unfolding f_f'_def using K that
    by (intro termdiffs_strong[of ?f K z] summable_strong) simp_all
    have f 0 = (∑ n. if n = 0 then 1 else 0) unfolding f_f'_def by (intro sum-
  inf_cong) simp
    also have ... = 1 using sums_single[of 0 λ_. 1::complex] unfolding sums_iff
  by simp
    finally have [simp]: f 0 = 1 .

  have ∃ c. ∀ z ∈ ball 0 K. f z * (1 + z) powr (-a) = c
  proof (rule has_field_derivative_zero_constant)
    fix z :: complex assume z': z ∈ ball 0 K
    hence z: norm z < K by simp
    with K have nz: 1 + z ≠ 0 by (auto dest!: minus_unique)
    from z K have norm z < 1 by simp
    hence (1 + z) ∉ ℝ≤0 by (cases z) (auto simp: Complex_eq complex_nonpos_Reals_iff)
    hence ((λz. f z * (1 + z) powr (-a)) has_field_derivative
      f' z * (1 + z) powr (-a) - a * f z * (1 + z) powr (-a-1)) (at z)
  using z
    by (auto intro!: derivative_eq_intros)
    also from z have a * f z = (1 + z) * f' z by (rule deriv)
    finally show ((λz. f z * (1 + z) powr (-a)) has_field_derivative 0) (at z
  within ball 0 K)
    using nz by (simp add: field_simps powr_diff at_within_open[OF z])
  qed simp_all
  then obtain c where c: ∧z. z ∈ ball 0 K ⇒ f z * (1 + z) powr (-a) = c by
  blast
  from c[of 0] and K have c = 1 by simp
  with c[of z] have f z = (1 + z) powr a using K
    by (simp add: powr_minus field_simps dist_complex_def)
  with summable K show ?thesis unfolding f_f'_def by (simp add: sums_iff)
qed

lemma gen_binomial_complex':
  fixes x y :: real and a :: complex
  assumes |x| < |y|
  shows (λn. (a gchoose n) * of_real x^n * of_real y powr (a - of_nat n)) sums
    of_real (x + y) powr a (is ?P x y)
proof -
  {
    fix x y :: real assume xy: |x| < |y| y ≥ 0
    hence y > 0 by simp
    note xy = xy this
  }

```

```

from xy have ( $\lambda n. (a \text{ gchoose } n) * \text{of\_real } (x / y) ^ n$ ) sums ( $1 + \text{of\_real } (x / y)$ ) powr a
  by (intro gen_binomial_complex) (simp add: norm_divide)
hence ( $\lambda n. (a \text{ gchoose } n) * \text{of\_real } (x / y) ^ n * y \text{ powr } a$ ) sums
  ( $(1 + \text{of\_real } (x / y)) \text{ powr } a * y \text{ powr } a$ )
  by (rule sums_mult2)
also have ( $1 + \text{complex\_of\_real } (x / y)$ ) = complex_of_real ( $1 + x/y$ ) by
simp
also from xy have ... powr a * of_real y powr a = (... * y) powr a
  by (subst powr_times_real[symmetric]) (simp_all add: field_simps)
also from xy have complex_of_real ( $1 + x / y$ ) * complex_of_real y = of_real
( $x + y$ )
  by (simp add: field_simps)
finally have ?P x y using xy by (simp add: field_simps powr_diff powr_nat)
} note A = this

show ?thesis
proof (cases y < 0)
  assume y:  $y < 0$ 
  with assms have xy:  $x + y < 0$  by simp
  with assms have  $| -x | < | -y |$   $-y \geq 0$  by simp_all
  note A[OF this]
  also have complex_of_real ( $-x + -y$ ) =  $- \text{complex\_of\_real } (x + y)$  by simp
  also from xy assms have ... powr a =  $(-1) \text{ powr } -a * \text{of\_real } (x + y) \text{ powr } a$ 
  by (subst powr_neg_real_complex) (simp add: abs_real_def split: if_split_asm)
  also {
    fix n :: nat
    from y have ( $a \text{ gchoose } n$ ) * of_real ( $-x$ ) ^ n * of_real ( $-y$ ) powr ( $a -$ 
of_nat n) =
      ( $a \text{ gchoose } n$ ) * ( $-\text{of\_real } x / -\text{of\_real } y$ ) ^ n * ( $-\text{of\_real } y$ )
powr a
    by (subst power_divide) (simp add: powr_diff powr_nat)
    also from y have ( $-\text{of\_real } y$ ) powr a =  $(-1) \text{ powr } -a * \text{of\_real } y \text{ powr } a$ 
    by (subst powr_neg_real_complex) simp
    also have  $-\text{complex\_of\_real } x / -\text{complex\_of\_real } y = \text{complex\_of\_real } x /$ 
complex_of_real y
    by simp
    also have ... ^ n =  $\text{of\_real } x ^ n / \text{of\_real } y ^ n$  by (simp add: power_divide)
    also have ( $a \text{ gchoose } n$ ) * ... *  $(-1) \text{ powr } -a * \text{of\_real } y \text{ powr } a$  =
       $(-1) \text{ powr } -a * ((a \text{ gchoose } n) * \text{of\_real } x ^ n * \text{of\_real } y \text{ powr } (a - n))$ 
    by (simp add: algebra_simps powr_diff powr_nat)
    finally have ( $a \text{ gchoose } n$ ) * of_real ( $-x$ ) ^ n * of_real ( $-y$ ) powr ( $a -$ 
of_nat n) =
       $(-1) \text{ powr } -a * ((a \text{ gchoose } n) * \text{of\_real } x ^ n * \text{of\_real } y \text{ powr } (a - \text{of\_nat } n))$  .
  }
  note sums_cong[OF this]
  finally show ?thesis by (simp add: sums_mult_iff)

```

3708

qed (*insert A[of x y] assms, simp_all add: not_less*)
qed

lemma *gen_binomial_complex''*:
 fixes $x\ y :: \text{real}$ **and** $a :: \text{complex}$
 assumes $|y| < |x|$
 shows $(\lambda n. (a \text{ gchoose } n) * \text{of_real } x \text{ powr } (a - \text{of_nat } n) * \text{of_real } y ^ n)$
 sums
 $\text{of_real } (x + y) \text{ powr } a$
 using *gen_binomial_complex'[OF assms]* **by** (*simp add: mult_ac add.commute*)

lemma *gen_binomial_real*:
 fixes $z :: \text{real}$
 assumes $|z| < 1$
 shows $(\lambda n. (a \text{ gchoose } n) * z ^ n)$ *sums* $(1 + z) \text{ powr } a$
proof –
 from *assms* **have** $\text{norm } (\text{of_real } z :: \text{complex}) < 1$ **by** *simp*
 from *gen_binomial_complex[OF this]*
 have $(\lambda n. (\text{of_real } a \text{ gchoose } n :: \text{complex}) * \text{of_real } z ^ n)$ *sums*
 $(\text{of_real } (1 + z)) \text{ powr } (\text{of_real } a)$ **by** *simp*
 also have $(\text{of_real } (1 + z) :: \text{complex}) \text{ powr } (\text{of_real } a) = \text{of_real } ((1 + z)$
 powr a)
 using *assms* **by** (*subst powr_of_real simp_all*)
 also have $(\text{of_real } a \text{ gchoose } n :: \text{complex}) = \text{of_real } (a \text{ gchoose } n)$ **for** n
 by (*simp add: gbinomial_prod_rev*)
 hence $(\lambda n. (\text{of_real } a \text{ gchoose } n :: \text{complex}) * \text{of_real } z ^ n) =$
 $(\lambda n. \text{of_real } ((a \text{ gchoose } n) * z ^ n))$ **by** (*intro ext simp*)
 finally show *?thesis* **by** (*simp only: sums_of_real_iff*)
qed

lemma *gen_binomial_real'*:
 fixes $x\ y\ a :: \text{real}$
 assumes $|x| < y$
 shows $(\lambda n. (a \text{ gchoose } n) * x ^ n * y \text{ powr } (a - \text{of_nat } n))$ *sums* $(x + y) \text{ powr } a$
proof –
 from *assms* **have** $y > 0$ **by** *simp*
 note $xy = \text{this } \text{assms}$
 from *assms* **have** $|x / y| < 1$ **by** *simp*
 hence $(\lambda n. (a \text{ gchoose } n) * (x / y) ^ n)$ *sums* $(1 + x / y) \text{ powr } a$
 by (*rule gen_binomial_real*)
 hence $(\lambda n. (a \text{ gchoose } n) * (x / y) ^ n * y \text{ powr } a)$ *sums* $((1 + x / y) \text{ powr } a * y \text{ powr } a)$
 by (*rule sums_mult2*)
 with xy **show** *?thesis*
 by (*simp add: field_simps powr_divide powr_diff powr_realpow*)
qed

lemma *one_plus_neg_powr_powser*:


```

fixes  $z\ s :: \text{complex}$ 
assumes  $\text{norm } (z :: \text{complex}) < 1$ 
shows  $(\lambda n. (-1)^{\wedge} n * ((s + n - 1) \text{ gchoose } n) * z^{\wedge} n) \text{ sums } (1 + z) \text{ powr } (-s)$ 
using  $\text{gen\_binomial\_complex}[OF\ \text{assms},\ \text{of } -s]$  by  $(\text{simp add: gbinomial\_minus})$ 

```

```

lemma  $\text{gen\_binomial\_real''}$ :
fixes  $x\ y\ a :: \text{real}$ 
assumes  $|y| < x$ 
shows  $(\lambda n. (a \text{ gchoose } n) * x \text{ powr } (a - \text{of\_nat } n) * y^{\wedge} n) \text{ sums } (x + y) \text{ powr } a$ 
using  $\text{gen\_binomial\_real''}[OF\ \text{assms}]$  by  $(\text{simp add: mult\_ac add.commute})$ 

```

```

lemma  $\text{sqrt\_series'}$ :
 $|z| < a \implies (\lambda n. ((1/2) \text{ gchoose } n) * a \text{ powr } (1/2 - \text{real\_of\_nat } n) * z^{\wedge} n) \text{ sums}$ 
 $\text{sqrt } (a + z :: \text{real})$ 
using  $\text{gen\_binomial\_real''}[of\ z\ a\ 1/2]$  by  $(\text{simp add: powr\_half\_sqrt})$ 

```

```

lemma  $\text{sqrt\_series}$ :
 $|z| < 1 \implies (\lambda n. ((1/2) \text{ gchoose } n) * z^{\wedge} n) \text{ sums } \text{sqrt } (1 + z)$ 
using  $\text{gen\_binomial\_real}[of\ z\ 1/2]$  by  $(\text{simp add: powr\_half\_sqrt})$ 

```

end

9.31 Vitali Covering Theorem and an Application to Negligibility

```

theory  $\text{Vitali\_Covering\_Theorem}$ 
imports
   $\text{HOL-Combinatorics.Permutations}$ 
   $\text{Equivalence\_Lebesgue\_Henstock\_Integration}$ 
begin

```

```

lemma  $\text{stretch\_Galois}$ :
fixes  $x :: \text{real}^{\wedge} n$ 
shows  $(\bigwedge k. m\ k \neq 0) \implies ((y = (\chi\ k. m\ k * x\$k)) \longleftrightarrow (\chi\ k. y\$k / m\ k) = x)$ 
by  $\text{auto}$ 

```

```

lemma  $\text{lambda\_swap\_Galois}$ :
 $(x = (\chi\ i. y\ \$\ \text{Transposition.transpose } m\ n\ i) \longleftrightarrow (\chi\ i. x\ \$\ \text{Transposition.transpose } m\ n\ i) = y)$ 
by  $(\text{auto}; \text{simp add: pointfree\_idE vec\_eq\_iff})$ 

```

```

lemma  $\text{lambda\_add\_Galois}$ :
fixes  $x :: \text{real}^{\wedge} n$ 
shows  $m \neq n \implies (x = (\chi\ i. \text{if } i = m \text{ then } y\$m + y\$n \text{ else } y\$i) \longleftrightarrow (\chi\ i. \text{if } i = m \text{ then } x\$m - x\$n \text{ else } x\$i) = y)$ 
by  $(\text{safe}; \text{simp add: vec\_eq\_iff})$ 

```

```

lemma Vitali_covering_lemma_cballs_balls:
  fixes a :: 'a ⇒ 'b::euclidean_space
  assumes  $\bigwedge i. i \in K \implies 0 < r\ i \wedge r\ i \leq B$ 
  obtains C where countable C C  $\subseteq$  K
    pairwise ( $\lambda i\ j. \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j))$ ) C
     $\bigwedge i. i \in K \implies \exists j. j \in C \wedge$ 
       $\neg \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j)) \wedge$ 
       $\text{cball}(a\ i)(r\ i) \subseteq \text{ball}(a\ j)(5 * r\ j)$ 
proof (cases K = {})
  case True
  with that show ?thesis
  by auto
next
  case False
  then have B > 0
    using assms less_le_trans by auto
  have rgt0[simp]:  $\bigwedge i. i \in K \implies 0 < r\ i$ 
    using assms by auto
  let ?djnt = pairwise ( $\lambda i\ j. \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j))$ )
  have  $\exists C. \forall n. (C\ n \subseteq K \wedge$ 
     $(\forall i \in C\ n. B/2 \wedge n \leq r\ i) \wedge ?djnt\ (C\ n) \wedge$ 
     $(\forall i \in K. B/2 \wedge n < r\ i$ 
       $\longrightarrow (\exists j. j \in C\ n \wedge$ 
         $\neg \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j)) \wedge$ 
         $\text{cball}(a\ i)(r\ i) \subseteq \text{ball}(a\ j)(5 * r\ j)))) \wedge (C\ n \subseteq C(\text{Suc}\ n))$ 
  proof (rule dependent_nat_choice, safe)
    fix C n
    define D where  $D \equiv \{i \in K. B/2 \wedge \text{Suc}\ n < r\ i \wedge (\forall j \in C. \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j)))\}$ 
    let ?cover_ar =  $\lambda i\ j. \neg \text{disjnt}(\text{cball}(a\ i)(r\ i))(\text{cball}(a\ j)(r\ j)) \wedge$ 
       $\text{cball}(a\ i)(r\ i) \subseteq \text{ball}(a\ j)(5 * r\ j)$ 
    assume C  $\subseteq$  K
      and Ble:  $\forall i \in C. B/2 \wedge n \leq r\ i$ 
      and djntC: ?djnt C
      and cov_n:  $\forall i \in K. B/2 \wedge n < r\ i \longrightarrow (\exists j. j \in C \wedge ?cover\_ar\ i\ j)$ 
    have *:  $\forall C \in \text{chains}\ \{C. C \subseteq D \wedge ?djnt\ C\}. \bigcup C \in \{C. C \subseteq D \wedge ?djnt\ C\}$ 
    proof (clarsimp simp: chains_def)
      fix C
      assume C:  $C \subseteq \{C. C \subseteq D \wedge ?djnt\ C\}$  and chain $\subseteq$  C
      show  $\bigcup C \subseteq D \wedge ?djnt\ (\bigcup C)$ 
        unfolding pairwise_def
      proof (intro ballI conjI impI)
        show  $\bigcup C \subseteq D$ 
          using C by blast
      next
        fix x y
        assume  $x \in \bigcup C$  and  $y \in \bigcup C$  and  $x \neq y$ 

```

```

then obtain X Y where XY: x ∈ X X ∈ C y ∈ Y Y ∈ C
  by blast
then consider X ⊆ Y | Y ⊆ X
  by (meson ‹chain⊆ C› chain_subset_def)
then show disjnt (cball (a x) (r x)) (cball (a y) (r y))
proof cases
  case 1
  with C XY ‹x ≠ y› show ?thesis
    unfolding pairwise_def by blast
  next
  case 2
  with C XY ‹x ≠ y› show ?thesis
    unfolding pairwise_def by blast
qed
qed
obtain E where E ⊆ D and djntE: ?djnt E and maximalE: ⋀X. ‹X ⊆ D;
?djnt X; E ⊆ X› ⇒ X = E
  using Zorn_Lemma [OF *] by safe blast
show ∃ L. (L ⊆ K ∧
  (∀ i ∈ L. B/2 ^ Suc n ≤ r i) ∧ ?djnt L ∧
  (∀ i ∈ K. B/2 ^ Suc n < r i → (∃ j. j ∈ L ∧ ?cover_ar i j))) ∧ C ⊆ L
proof (intro exI conjI ballI)
  show C ∪ E ⊆ K
    using D_def ‹C ⊆ K› ‹E ⊆ D› by blast
  show B/2 ^ Suc n ≤ r i if i: i ∈ C ∪ E for i
    using i
  proof
    assume i ∈ C
    have B/2 ^ Suc n ≤ B/2 ^ n
      using ‹B > 0› by (simp add: field_split_simps)
    also have ... ≤ r i
      using Ble ‹i ∈ C› by blast
    finally show ?thesis .
  qed (use D_def ‹E ⊆ D› in auto)
  show ?djnt (C ∪ E)
    using D_def ‹C ⊆ K› ‹E ⊆ D› djntC djntE
    unfolding pairwise_def disjnt_def by blast
next
fix i
assume i ∈ K
show B/2 ^ Suc n < r i → (∃ j. j ∈ C ∪ E ∧ ?cover_ar i j)
proof (cases r i ≤ B/2 ^ n)
  case False
  then show ?thesis
    using cov_n ‹i ∈ K› by auto
next
case True
  have cball (a i) (r i) ⊆ ball (a j) (5 * r j)

```

```

    if less:  $B/2 \wedge \text{Suc } n < r \ i$  and  $j: j \in C \cup E$ 
      and nondis:  $\neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))$  for  $j$ 
    proof -
      obtain  $x$  where  $x: \text{dist } (a \ i) \ x \leq r \ i$   $\text{dist } (a \ j) \ x \leq r \ j$ 
        using nondis by (force simp: disjnt_def)
      have  $\text{dist } (a \ i) \ (a \ j) \leq \text{dist } (a \ i) \ x + \text{dist } x \ (a \ j)$ 
        by (simp add: dist_triangle)
      also have  $\dots \leq r \ i + r \ j$ 
        by (metis add_mono_thms_linordered_semiring(1) dist_commute x)
      finally have  $\text{aij}: \text{dist } (a \ i) \ (a \ j) + r \ i < 5 * r \ j$  if  $r \ i < 2 * r \ j$ 
        using that by auto
      show ?thesis
        using j
      proof
        assume  $j \in C$ 
        have  $B/2^n < 2 * r \ j$ 
          using Ble True  $\langle j \in C \rangle$  less by auto
        with  $\text{aij}$  True show  $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
          by (simp add: cball_subset_ball_iff)
        next
          assume  $j \in E$ 
          then have  $B/2^n < 2 * r \ j$ 
            using D_def  $\langle E \subseteq D \rangle$  by auto
          with True have  $r \ i < 2 * r \ j$ 
            by auto
          with  $\text{aij}$  show  $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$ 
            by (simp add: cball_subset_ball_iff)
        qed
      qed
    moreover have  $\exists j. j \in C \cup E \wedge \neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))$ 
      if  $B/2 \wedge \text{Suc } n < r \ i$ 
    proof (rule classical)
      assume NON:  $\neg ?thesis$ 
      show ?thesis
      proof (cases  $i \in D$ )
        case True
          have  $\text{insert } i \ E = E$ 
          proof (rule maximalE)
            show  $\text{insert } i \ E \subseteq D$ 
              by (simp add: True  $\langle E \subseteq D \rangle$ )
            show pairwise  $(\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ (\text{insert } i \ E)$ 
              using False NON by (auto simp: pairwise_insert djntE disjnt_sym)
          qed auto
          then show ?thesis
            using  $\langle i \in K \rangle$  assms by fastforce
        case False
          with that show ?thesis
      next
        case True
          with that show ?thesis
      next
        case False
          with that show ?thesis
    qed
  qed

```

```

      by (auto simp: D_def disjnt_def ‹i ∈ K›)
    qed
  qed
  ultimately
  show B/2 ^ Suc n < r i →
    (∃ j. j ∈ C ∪ E ∧
      ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
      cball (a i) (r i) ⊆ ball (a j) (5 * r j))
  by blast
  qed
  qed auto
  qed (use assms in force)
  then obtain F where FK: ∧ n. F n ⊆ K
    and Fle: ∧ n i. i ∈ F n ⇒ B/2 ^ n ≤ r i
    and Fdjnt: ∧ n. ?djnt (F n)
    and FF: ∧ n i. [i ∈ K; B/2 ^ n < r i]
      ⇒ ∃ j. j ∈ F n ∧ ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
      cball (a i) (r i) ⊆ ball (a j) (5 * r j)
    and inc: ∧ n. F n ⊆ F (Suc n)
  by (force simp: all_conj_distrib)
  show thesis
  proof
    have *: countable I
      if I ⊆ K and pw: pairwise (λ i j. disjnt (cball (a i) (r i)) (cball (a j) (r j)))
    I for I
  proof -
    show ?thesis
    proof (rule countable_image_inj_on [of λ i. cball (a i) (r i)])
      show countable ((λ i. cball (a i) (r i)) ' I)
      proof (rule countable_disjoint_nonempty_interior_subsets)
        show disjoint ((λ i. cball (a i) (r i)) ' I)
          by (auto simp: dest: pairwiseD [OF pw] intro: pairwise_imageI)
        show ∧ S. [S ∈ (λ i. cball (a i) (r i)) ' I; interior S = {}] ⇒ S = {}
          using ‹I ⊆ K›
          by (auto simp: not_less [symmetric])
      qed
    next
      have ∧ x y. [x ∈ I; y ∈ I; a x = a y; r x = r y] ⇒ x = y
        using pw ‹I ⊆ K› assms
        apply (clarsimp simp: pairwise_def disjnt_def)
      by (metis assms centre_in_cball subsetD empty_iff inf.idem less_eq_real_def)
      then show inj_on (λ i. cball (a i) (r i)) I
        using ‹I ⊆ K› by (fastforce simp: inj_on_def cball_eq_cball_iff dest:
      assms)
    qed
  qed
  show (Union (range F)) ⊆ K
  using FK by blast
  moreover show pairwise (λ i j. disjnt (cball (a i) (r i)) (cball (a j) (r j)))

```

```

(Union(range F))
  proof (rule pairwise_chain_Union)
    show chain⊆ (range F)
      unfolding chain_subset_def by clarify (meson inc lift_Suc_mono_le linear
subsetCE)
    qed (use Fdjnt in blast)
    ultimately show countable (Union(range F))
      by (blast intro: *)
  next
  fix i assume i ∈ K
  then obtain n where (1/2) ^ n < r i / B
    using ‹B > 0› assms real_arch_pow_inv by fastforce
  then have B2: B/2 ^ n < r i
    using ‹B > 0› by (simp add: field_split_simps)
  have 0 < r i r i ≤ B
    by (auto simp: ‹i ∈ K› assms)
  show ∃ j. j ∈ (Union(range F)) ∧
    ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
    cball (a i) (r i) ⊆ ball (a j) (5 * r j)
    using FF [OF ‹i ∈ K› B2] by auto
  qed
qed

```

9.31.1 Vitali covering theorem

```

lemma Vitali_covering_lemma_cballs:
  fixes a :: 'a ⇒ 'b::euclidean_space
  assumes S: S ⊆ (⋃ i∈K. cball (a i) (r i))
    and r: ⋀ i. i ∈ K ⇒ 0 < r i ∧ r i ≤ B
  obtains C where countable C C ⊆ K
    pairwise (λ i j. disjnt (cball (a i) (r i)) (cball (a j) (r j))) C
    S ⊆ (⋃ i∈C. cball (a i) (5 * r i))
proof -
  obtain C where C: countable C C ⊆ K
    pairwise (λ i j. disjnt (cball (a i) (r i)) (cball (a j) (r j))) C
    and cov: ⋀ i. i ∈ K ⇒ ∃ j. j ∈ C ∧ ¬ disjnt (cball (a i) (r i)) (cball (a
j) (r j)) ∧
      cball (a i) (r i) ⊆ ball (a j) (5 * r j)
  by (rule Vitali_covering_lemma_cballs_balls [OF r, where a=a]) (blast intro:
that)+
  show ?thesis
proof
  have (⋃ i∈K. cball (a i) (r i)) ⊆ (⋃ i∈C. cball (a i) (5 * r i))
    using cov subset_iff by fastforce
  with S show S ⊆ (⋃ i∈C. cball (a i) (5 * r i))
    by blast
qed (use C in auto)
qed

```

lemma *Vitali_covering_lemma_balls*:
fixes $a :: 'a \Rightarrow 'b::\text{euclidean_space}$
assumes $S: S \subseteq (\bigcup i \in K. \text{ball } (a \ i) \ (r \ i))$
and $r: \bigwedge i. i \in K \implies 0 < r \ i \wedge r \ i \leq B$
obtains C **where** *countable* $C \ C \subseteq K$
pairwise $(\lambda i \ j. \text{disjnt } (\text{ball } (a \ i) \ (r \ i)) \ (\text{ball } (a \ j) \ (r \ j))) \ C$
 $S \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$
proof –
obtain C **where** $C: \text{countable } C \ C \subseteq K$
and $pw: \text{pairwise } (\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C$
and $cov: \bigwedge i. i \in K \implies \exists j. j \in C \wedge \neg \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j)) \wedge$
 $\text{cball } (a \ i) \ (r \ i) \subseteq \text{ball } (a \ j) \ (5 * r \ j)$
by (rule *Vitali_covering_lemma_cballs_balls* [OF r , **where** $a=a$]) (blast intro: *that*)+
show ?thesis
proof
have $(\bigcup i \in K. \text{ball } (a \ i) \ (r \ i)) \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$
using *cov subset_iff*
by *clarsimp* (meson *less_imp_le mem_ball mem_cball subset_eq*)
with S **show** $S \subseteq (\bigcup i \in C. \text{ball } (a \ i) \ (5 * r \ i))$
by *blast*
show *pairwise* $(\lambda i \ j. \text{disjnt } (\text{ball } (a \ i) \ (r \ i)) \ (\text{ball } (a \ j) \ (r \ j))) \ C$
using pw
by (*clarsimp simp: pairwise_def*) (meson *ball_subset_cball disjnt_subset1 disjnt_subset2*)
qed (use C in *auto*)
qed

theorem *Vitali_covering_theorem_cballs*:
fixes $a :: 'a \Rightarrow 'n::\text{euclidean_space}$
assumes $r: \bigwedge i. i \in K \implies 0 < r \ i$
and $S: \bigwedge x \ d. \llbracket x \in S; 0 < d \rrbracket$
 $\implies \exists i. i \in K \wedge x \in \text{cball } (a \ i) \ (r \ i) \wedge r \ i < d$
obtains C **where** *countable* $C \ C \subseteq K$
pairwise $(\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C$
negligible $(S - (\bigcup i \in C. \text{cball } (a \ i) \ (r \ i)))$
proof –
let $?\mu = \text{measure lebesgue}$
have $*$: $\exists C. \text{countable } C \wedge C \subseteq K \wedge$
 $\text{pairwise } (\lambda i \ j. \text{disjnt } (\text{cball } (a \ i) \ (r \ i)) \ (\text{cball } (a \ j) \ (r \ j))) \ C \wedge$
 $\text{negligible}(S - (\bigcup i \in C. \text{cball } (a \ i) \ (r \ i)))$
if $r01: \bigwedge i. i \in K \implies 0 < r \ i \wedge r \ i \leq 1$
and $Sd: \bigwedge x \ d. \llbracket x \in S; 0 < d \rrbracket \implies \exists i. i \in K \wedge x \in \text{cball } (a \ i) \ (r \ i) \wedge r \ i <$
 d
for $K \ r$ **and** $a :: 'a \Rightarrow 'n$
proof –
obtain C **where** $C: \text{countable } C \ C \subseteq K$

```

and pwC: pairwise (λi j. disjnt (cball (a i) (r i)) (cball (a j) (r j))) C
and cov: ∧i. i ∈ K ⇒ ∃j. j ∈ C ∧ ¬ disjnt (cball (a i) (r i)) (cball (a j)
(r j)) ∧
      cball (a i) (r i) ⊆ ball (a j) (5 * r j)
by (rule Vitali_covering_lemma_cballs_balls [of K r 1 a]) (auto simp: r01)
have ar_injective: ∧x y. [x ∈ C; y ∈ C; a x = a y; r x = r y] ⇒ x = y
using ⟨C ⊆ K⟩ pwC cov
by (force simp: pairwise_def disjnt_def)
show ?thesis
proof (intro exI conjI)
show negligible (S - (∪i∈C. cball (a i) (r i)))
proof (clarsimp simp: negligible_on_intervals [of S-T for T])
fix l u
show negligible ((S - (∪i∈C. cball (a i) (r i))) ∩ cbox l u)
unfolding negligible_outer_le
proof (intro allI impI)
fix e::real
assume e > 0
define D where D ≡ {i ∈ C. ¬ disjnt (ball(a i) (5 * r i)) (cbox l u)}
then have D ⊆ C
by auto
have countable D
unfolding D_def using ⟨countable C⟩ by simp
have UD: (∪i∈D. cball (a i) (r i)) ∈ lmeasurable
proof (rule fmeasurableI2)
show cbox (l - 6 *R One) (u + 6 *R One) ∈ lmeasurable
by blast
have y ∈ cbox (l - 6 *R One) (u + 6 *R One)
if i ∈ C and x: x ∈ cbox l u and ai: dist (a i) y ≤ r i dist (a i) x < 5
* r i
for i x y
proof -
have d6: dist y x < 6 * r i
using dist_triangle3 [of y x a i] that by linarith
show ?thesis
proof (clarsimp simp: mem_box algebra_simps)
fix j::'n
assume j: j ∈ Basis
then have xyj: |x · j - y · j| ≤ dist y x
by (metis Basis_le_norm dist_commute dist_norm inner_diff_left)
have l · j ≤ x · j
using ⟨j ∈ Basis⟩ mem_box ⟨x ∈ cbox l u⟩ by blast
also have ... ≤ y · j + 6 * r i
using d6 xyj by (auto simp: algebra_simps)
also have ... ≤ y · j + 6
using r01 [of i] ⟨C ⊆ K⟩ ⟨i ∈ C⟩ by auto
finally have l · j ≤ y · j + 6 .
have y · j ≤ x · j + 6 * r i
using d6 xyj by (auto simp: algebra_simps)

```



```

    also have ... ≤ u · j + 6 * r i
      using j x by (auto simp: mem_box)
    also have ... ≤ u · j + 6
      using r01 [of i] ⟨C ⊆ K⟩ ⟨i ∈ C⟩ by auto
    finally have u: y · j ≤ u · j + 6 .
    show l · j ≤ y · j + 6 ∧ y · j ≤ u · j + 6
      using l u by blast
  qed
qed
then show (⋃ i ∈ D. cball (a i) (r i)) ⊆ cbox (l - 6 *R One) (u + 6 *R
One)
  by (force simp: D_def disjnt_def)
show (⋃ i ∈ D. cball (a i) (r i)) ∈ sets lebesgue
  using ⟨countable D⟩ by auto
qed
obtain D1 where D1 ⊆ D finite D1
  and measD1: ?μ (⋃ i ∈ D. cball (a i) (r i)) - e / 5 ^ DIM('n) < ?μ
(⋃ i ∈ D1. cball (a i) (r i))
proof (rule measure_countable_Union_approachable [where e = e / 5 ^
(DIM('n))])
  show countable ((λi. cball (a i) (r i)) ' D)
    using ⟨countable D⟩ by auto
  show ∧ d. d ∈ (λi. cball (a i) (r i)) ' D ⇒ d ∈ lmeasurable
    by auto
  show ∧ D'. [D' ⊆ (λi. cball (a i) (r i)) ' D; finite D'] ⇒ ?μ (⋃ D') ≤
?μ (⋃ i ∈ D. cball (a i) (r i))
    by (fastforce simp add: intro!: measure_mono_fmeasurable UD)
qed (use ⟨e > 0⟩ in ⟨auto dest: finite_subset_image⟩)
show ∃ T. (S - (⋃ i ∈ C. cball (a i) (r i))) ∩
  cbox l u ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
proof (intro exI conjI)
  show (S - (⋃ i ∈ C. cball (a i) (r i))) ∩ cbox l u ⊆ (⋃ i ∈ D - D1. ball
(a i) (5 * r i))
  proof clarify
    fix x
    assume x: x ∈ cbox l u x ∈ S x ∉ (⋃ i ∈ C. cball (a i) (r i))
    have closed (⋃ i ∈ D1. cball (a i) (r i))
      using ⟨finite D1⟩ by blast
    moreover have x ∉ (⋃ j ∈ D1. cball (a j) (r j))
      using x ⟨D1 ⊆ D⟩ unfolding D_def by blast
    ultimately obtain q where q > 0 and q: ball x q ⊆ - (⋃ i ∈ D1. cball
(a i) (r i))
      by (metis (no_types, lifting) ComplI open_contains_ball closed_def)
    obtain i where i ∈ K and xi: x ∈ cball (a i) (r i) and ri: r i < q/2
      using Sd [OF ⟨x ∈ S⟩] ⟨q > 0⟩ half_gt_zero by blast
    then obtain j where j ∈ C
      and nondisj: ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j))
      and sub5j: cball (a i) (r i) ⊆ ball (a j) (5 * r j)
      using cov [OF ⟨i ∈ K⟩] by metis

```

```

show  $x \in (\bigcup_{i \in D} - D1. \text{ball } (a \ i) \ (5 * r \ i))$ 
proof
  show  $j \in D - D1$ 
  proof
    show  $j \in D$ 
    using  $\langle j \in C \rangle \text{sub5j } \langle x \in \text{cbox } l \ u \rangle \ xi$  by (auto simp: D_def
disjnt_def)
    obtain  $y$  where  $yi: \text{dist } (a \ i) \ y \leq r \ i$  and  $yj: \text{dist } (a \ j) \ y \leq r \ j$ 
      using disjnt_def nondisj by fastforce
    have  $\text{dist } x \ y \leq r \ i + r \ i$ 
      by (metis add_mono dist_commute dist_triangle_le mem_cball
xi yi)
    also have  $\dots < q$ 
      using  $ri$  by linarith
    finally have  $y \in \text{ball } x \ q$ 
      by simp
    with  $yj \ q$  show  $j \notin D1$ 
      by (auto simp: disjoint_UN_iff)
  qed
  show  $x \in \text{ball } (a \ j) \ (5 * r \ j)$ 
    using  $xi \ \text{sub5j}$  by blast
  qed
  have  $\exists: ?\mu (\bigcup_{i \in D2}. \text{ball } (a \ i) \ (5 * r \ i)) \leq e$ 
    if  $D2: D2 \subseteq D - D1$  and finite  $D2$  for  $D2$ 
  proof -
    have  $\text{rgt0}: 0 < r \ i$  if  $i \in D2$  for  $i$ 
      using  $\langle C \subseteq K \rangle \ D\_def \ \langle i \in D2 \rangle \ D2 \ r01$ 
      by (simp add: subset_iff)
    then have  $\text{inj}: \text{inj\_on } (\lambda i. \text{ball } (a \ i) \ (5 * r \ i)) \ D2$ 
      using  $\langle C \subseteq K \rangle \ D2$  by (fastforce simp: inj_on_def D_def
ball_eq_ball_iff intro: ar_injective)
    have  $?\mu (\bigcup_{i \in D2}. \text{ball } (a \ i) \ (5 * r \ i)) \leq \text{sum } (?\mu) ((\lambda i. \text{ball } (a \ i) \ (5 * r \ i)) \ 'D2)$ 
      using that by (force intro: measure_Union_le)
    also have  $\dots = (\sum_{i \in D2}. ?\mu (\text{ball } (a \ i) \ (5 * r \ i)))$ 
      by (simp add: comm_monoid_add_class.sum_reindex [OF inj])
    also have  $\dots = (\sum_{i \in D2}. 5 \wedge DIM('n) * ?\mu (\text{ball } (a \ i) \ (r \ i)))$ 
      proof (rule sum.cong [OF refl])
        fix  $i$  assume  $i \in D2$ 
        thus  $?\mu (\text{ball } (a \ i) \ (5 * r \ i)) = 5 \wedge DIM('n) * ?\mu (\text{ball } (a \ i) \ (r \ i))$ 
          using content_ball_conv_unit_ball[of  $5 * r \ i \ a \ i$ ]
            content_ball_conv_unit_ball[of  $r \ i \ a \ i$ ] rgt0[of  $i$ ] by auto
      qed
    also have  $\dots = (\sum_{i \in D2}. ?\mu (\text{ball } (a \ i) \ (r \ i))) * 5 \wedge DIM('n)$ 
      by (simp add: sum_distrib_left mult.commute)
    finally have  $?\mu (\bigcup_{i \in D2}. \text{ball } (a \ i) \ (5 * r \ i)) \leq (\sum_{i \in D2}. ?\mu (\text{ball } (a \ i) \ (r \ i))) * 5 \wedge DIM('n)$ 
      moreover have  $(\sum_{i \in D2}. ?\mu (\text{ball } (a \ i) \ (r \ i))) \leq e / 5 \wedge DIM('n)$ 

```

```

proof –
  have  $D12\_dis: ((\bigcup x \in D1. cball (a x) (r x)) \cap (\bigcup x \in D2. cball (a x) (r x))) \leq \{\}$ 
  proof clarify
    fix  $w d1 d2$ 
    assume  $d1 \in D1 \ w d1 d2 \in cball (a d1) (r d1) \ d2 \in D2 \ w d1 d2 \in cball (a d2) (r d2)$ 
    then show  $w d1 d2 \in \{\}$ 
      by (metis DiffE disjnt_iff subsetCE D2 <D1 ⊆ D> <D ⊆ C> pairwiseD [OF pwC, of d1 d2])
    qed
  have inj: inj_on  $(\lambda i. cball (a i) (r i)) D2$ 
  using rgt0 D2 <D ⊆ C> by (force simp: inj_on_def cball_eq_cball_iff intro!: ar_injective)
  have ds: disjoint  $((\lambda i. cball (a i) (r i)) ' D2)$ 
  using  $D2 <D ⊆ C>$  by (auto intro: pairwiseI pairwiseD [OF pwC])
  have  $(\sum i \in D2. ?\mu (ball (a i) (r i))) = (\sum i \in D2. ?\mu (cball (a i) (r i)))$ 
  by (simp add: content_cball_conv_ball)
  also have  $\dots = \text{sum } ?\mu ((\lambda i. cball (a i) (r i)) ' D2)$ 
  by (simp add: comm_monoid_add_class.sum_reindex [OF inj])
  also have  $\dots = ?\mu (\bigcup i \in D2. cball (a i) (r i))$ 
  by (auto intro: measure_Union' [symmetric] ds simp add: <finite D2>)
  finally have  $?\mu (\bigcup i \in D1. cball (a i) (r i)) + (\sum i \in D2. ?\mu (ball (a i) (r i))) =$ 
   $?\mu (\bigcup i \in D1. cball (a i) (r i)) + ?\mu (\bigcup i \in D2. cball (a i) (r i))$ 
  by simp
  also have  $\dots = ?\mu (\bigcup i \in D1 \cup D2. cball (a i) (r i))$ 
  using  $D12\_dis$  by (simp add: measure_Un3 <finite D1> <finite D2> fmeasurable.finite_UN)
  also have  $\dots \leq ?\mu (\bigcup i \in D. cball (a i) (r i))$ 
  using  $D2 <D1 ⊆ D>$  by (fastforce intro!: measure_mono_fmeasurable [OF __ UD] <finite D1> <finite D2>)
  finally have  $?\mu (\bigcup i \in D1. cball (a i) (r i)) + (\sum i \in D2. ?\mu (ball (a i) (r i))) \leq ?\mu (\bigcup i \in D. cball (a i) (r i))$  .
  with measD1 show ?thesis
  by simp
qed
ultimately show ?thesis
  by (simp add: field_split_simps)
qed
have co: countable  $(D - D1)$ 
  by (simp add: <countable D>)
show  $(\bigcup i \in D - D1. ball (a i) (5 * r i)) \in \text{lmeasurable}$ 
  using  $\langle e > 0 \rangle$  by (auto simp: fmeasurable_UN_bound [OF co _ 3])
show  $?\mu (\bigcup i \in D - D1. ball (a i) (5 * r i)) \leq e$ 
  using  $\langle e > 0 \rangle$  by (auto simp: measure_UN_bound [OF co _ 3])

```

3720

```
      qed
    qed
  qed
  qed (use C pwC in auto)
  qed
  define K' where K' ≡ {i ∈ K. r i ≤ 1}
  have 1:  $\bigwedge i. i \in K' \implies 0 < r i \wedge r i \leq 1$ 
    using K'_def r by auto
  have 2:  $\exists i. i \in K' \wedge x \in \text{cball } (a i) (r i) \wedge r i < d$ 
    if  $x \in S \wedge 0 < d$  for  $x d$ 
    using that by (auto simp: K'_def dest!: S [where d = min d 1])
  have K' ⊆ K
    using K'_def by auto
  then show thesis
    using * [OF 1 2] that by fastforce
  qed
```

theorem Vitali_covering_theorem_balls:

```
  fixes a :: 'a ⇒ 'b::euclidean_space
  assumes S:  $\bigwedge x d. \llbracket x \in S; 0 < d \rrbracket \implies \exists i. i \in K \wedge x \in \text{ball } (a i) (r i) \wedge r i < d$ 
  obtains C where countable C C ⊆ K
    pairwise (λi j. disjnt (ball (a i) (r i)) (ball (a j) (r j))) C
    negligible(S - (⋃ i ∈ C. ball (a i) (r i)))
  proof -
    have 1:  $\exists i. i \in \{i \in K. 0 < r i\} \wedge x \in \text{cball } (a i) (r i) \wedge r i < d$ 
      if  $xd: x \in S \wedge d > 0$  for  $x d$ 
    by (metis (mono_tags, lifting) assms ball_eq_empty less_eq_real_def mem_Collect_eq
      mem_ball mem_cball not_le xd(1) xd(2))
    obtain C where C: countable C C ⊆ K
      and pw: pairwise (λi j. disjnt (cball (a i) (r i)) (cball (a j) (r j))) C
      and neg: negligible(S - (⋃ i ∈ C. cball (a i) (r i)))
    by (rule Vitali_covering_theorem_cballs [of {i ∈ K. 0 < r i} r S a, OF _ 1])
  auto
  show thesis
  proof
    show pairwise (λi j. disjnt (ball (a i) (r i)) (ball (a j) (r j))) C
      apply (rule pairwise_mono [OF pw])
      apply (auto simp: disjnt_def)
      by (meson disjoint_iff_not_equal less_imp_le mem_cball)
    have negligible (⋃ i ∈ C. sphere (a i) (r i))
      by (auto intro: negligible_sphere ‹countable C›)
    then have negligible (S - (⋃ i ∈ C. cball (a i) (r i)) ∪ (⋃ i ∈ C. sphere (a i)
      (r i)))
      by (rule negligible_Un [OF neg])
    then show negligible (S - (⋃ i ∈ C. ball (a i) (r i)))
      by (rule negligible_subset) force
  qed (use C in auto)
  qed
```

```

lemma negligible_eq_zero_density_alt:
  negligible S  $\longleftrightarrow$ 
  ( $\forall x \in S. \forall e > 0.$ 
     $\exists d U. 0 < d \wedge d \leq e \wedge S \cap \text{ball } x \ d \subseteq U \wedge$ 
       $U \in \text{lmeasurable} \wedge \text{measure lebesgue } U < e * \text{measure lebesgue } (\text{ball } x$ 
d))
  (is _ = ( $\forall x \in S. \forall e > 0. ?Q \ x \ e$ ))
proof (intro iffI ballI allI impI)
  fix x and e :: real
  assume negligible S and x  $\in S$  and e > 0
  then
  show  $\exists d U. 0 < d \wedge d \leq e \wedge S \cap \text{ball } x \ d \subseteq U \wedge U \in \text{lmeasurable} \wedge$ 
     $\text{measure lebesgue } U < e * \text{measure lebesgue } (\text{ball } x \ d)$ 
    apply (rule_tac x=e in exI)
    apply (rule_tac x=S  $\cap$  ball x e in exI)
    apply (auto simp: negligible_imp_measurable negligible_Int negligible_imp_measure0
zero_less_measure_iff
  intro: mult_pos_pos content_ball_pos)
  done
next
assume R [rule_format]:  $\forall x \in S. \forall e > 0. ?Q \ x \ e$ 
let ? $\mu$  = measure lebesgue
have  $\exists U. \text{openin } (\text{top\_of\_set } S) \ U \wedge z \in U \wedge \text{negligible } U$ 
if z  $\in S$  for z
proof (intro exI conjI)
  show openin (top_of_set S) (S  $\cap$  ball z 1)
  by (simp add: openin_open_Int)
  show z  $\in S \cap$  ball z 1
  using <z  $\in S$ > by auto
  show negligible (S  $\cap$  ball z 1)
proof (clarsimp simp: negligible_outer_le)
  fix e :: real
  assume e > 0
  let ?K = {(x,d). x  $\in S \wedge 0 < d \wedge \text{ball } x \ d \subseteq \text{ball } z \ 1 \wedge$ 
    ( $\exists U. S \cap \text{ball } x \ d \subseteq U \wedge U \in \text{lmeasurable} \wedge$ 
       $? \mu \ U < e / ? \mu (\text{ball } z \ 1) * ? \mu (\text{ball } x \ d)$ )}
  obtain C where countable C and Csub: C  $\subseteq$  ?K
  and pwC: pairwise ( $\lambda i \ j. \text{disjnt } (\text{ball } (\text{fst } i) \ (\text{snd } i)) \ (\text{ball } (\text{fst } j) \ (\text{snd } j))$ ) C
  and negC: negligible((S  $\cap$  ball z 1) - ( $\bigcup i \in C. \text{ball } (\text{fst } i) \ (\text{snd } i)$ ))
  proof (rule Vitali_covering_theorem_balls [of S  $\cap$  ball z 1 ?K fst snd])
  fix x and d :: real
  assume x: x  $\in S \cap$  ball z 1 and d > 0
  obtain k where k > 0 and k: ball x k  $\subseteq$  ball z 1
  by (meson Int_iff open_ball openE x)
  let ? $\varepsilon$  = min (e / ? $\mu$  (ball z 1) / 2) (min (d / 2) k)
  obtain r U where r: r > 0 r  $\leq$  ? $\varepsilon$  and U: S  $\cap$  ball x r  $\subseteq$  U U  $\in$  lmeasurable
  and mU: ? $\mu$  U < ? $\varepsilon$  * ? $\mu$  (ball x r)

```

```

using R [of x ?ε] ⟨d > 0⟩ ⟨e > 0⟩ ⟨k > 0⟩ x by (auto simp: content_ball_pos)
show ∃ i. i ∈ ?K ∧ x ∈ ball (fst i) (snd i) ∧ snd i < d
proof (rule exI [of _ (x,r)], simp, intro conjI exI)
  have ball x r ⊆ ball x k
  using r by (simp add: ball_subset_ball_iff)
  also have ... ⊆ ball z 1
  using ball_subset_ball_iff k by auto
  finally show ball x r ⊆ ball z 1 .
  have ?ε * ?μ (ball x r) ≤ e * content (ball x r) / content (ball z 1)
  using r ⟨e > 0⟩ by (simp add: ord_class.min_def field_split_simps
content_ball_pos)
  with mU show ?μ U < e * content (ball x r) / content (ball z 1)
  by auto
qed (use r U x in auto)
qed
have ∃ U. case p of (x,d) ⇒ S ∩ ball x d ⊆ U ∧
  U ∈ lmeasurable ∧ ?μ U < e / ?μ (ball z 1) * ?μ (ball x d)
if p ∈ C for p
using that Csub unfolding case_prod_unfold by blast
then obtain U where U:
  ∧p. p ∈ C ⇒
  case p of (x,d) ⇒ S ∩ ball x d ⊆ U p ∧
  U p ∈ lmeasurable ∧ ?μ (U p) < e / ?μ (ball z 1) * ?μ (ball x d)
by (rule that [OF someI_ex])
let ?T = ((S ∩ ball z 1) - (∪(x,d)∈C. ball x d)) ∪ ∪(U ' C)
show ∃ T. S ∩ ball z 1 ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
proof (intro exI conjI)
  show S ∩ ball z 1 ⊆ ?T
  using U by fastforce
  { have Um: U i ∈ lmeasurable if i ∈ C for i
  using that U by blast
  have lee: ?μ (∪ i∈I. U i) ≤ e if I ⊆ C finite I for I
  proof -
  have ?μ (∪(x,d)∈I. ball x d) ≤ ?μ (ball z 1)
  apply (rule measure_mono_fmeasurable)
  using ⟨I ⊆ C⟩ ⟨finite I⟩ Csub by (force simp: prod.case_eq_if
sets.finite_UN)+
  then have le1: (?μ (∪(x,d)∈I. ball x d) / ?μ (ball z 1)) ≤ 1
  by (simp add: content_ball_pos)
  have ?μ (∪ i∈I. U i) ≤ (∑ i∈I. ?μ (U i))
  using that U by (blast intro: measure_UNION_le)
  also have ... ≤ (∑(x,r)∈I. e / ?μ (ball z 1) * ?μ (ball x r))
  by (rule sum_mono) (use ⟨I ⊆ C⟩ U in force)
  also have ... = (e / ?μ (ball z 1)) * (∑(x,r)∈I. ?μ (ball x r))
  by (simp add: case_prod_app prod.case_distrib sum_distrib_left)
  also have ... = e * (?μ (∪(x,r)∈I. ball x r) / ?μ (ball z 1))
  apply (subst measure_UNION)
  using that pwC by (auto simp: case_prod_unfold elim: pairwise_mono)
  also have ... ≤ e

```

```

      by (metis mult.commute mult.left_neutral mult_le_cancel_right_pos
<e > 0> le1)
    finally show ?thesis .
  qed
  have  $\bigcup(U \text{ ' } C) \in \text{lmeasurable } ?\mu$  ( $\bigcup(U \text{ ' } C) \leq e$ )
    using <e > 0> Um lee
    by(auto intro!: fmeasurable_UN_bound [OF <countable C>] mea-
sure_UN_bound [OF <countable C>])
  }
  moreover have  $?\mu \text{ } ?T = ?\mu$  ( $\bigcup(U \text{ ' } C)$ )
  proof (rule measure_negligible_syndiff [OF < $\bigcup(U \text{ ' } C) \in \text{lmeasurable}$ >])
    show negligible( $(\bigcup(U \text{ ' } C) - ?T) \cup (?T - \bigcup(U \text{ ' } C))$ )
      by (force intro!: negligible_subset [OF negC])
  qed
  ultimately show  $?T \in \text{lmeasurable } ?\mu$   $?T \leq e$ 
    by (simp_all add: fmeasurable.Un negC negligible_imp_measurable
split_def)
  qed
  qed
  qed
  with locally_negligible_alt show negligible S
    by metis
  qed

```

proposition negligible_eq_zero_density:

```

negligible S  $\longleftrightarrow$ 
( $\forall x \in S. \forall r > 0. \forall e > 0. \exists d. 0 < d \wedge d \leq r \wedge$ 
 $(\exists U. S \cap \text{ball } x \text{ } d \subseteq U \wedge U \in \text{lmeasurable} \wedge \text{measure lebesgue } U$ 
 $< e * \text{measure lebesgue } (\text{ball } x \text{ } d))$ )

```

proof –

```

let ?Q =  $\lambda x \text{ } d \text{ } e. \exists U. S \cap \text{ball } x \text{ } d \subseteq U \wedge U \in \text{lmeasurable} \wedge \text{measure lebesgue}$ 
 $U < e * \text{content } (\text{ball } x \text{ } d)$ 
have ( $\forall e > 0. \exists d > 0. d \leq e \wedge ?Q \text{ } x \text{ } d \text{ } e$ ) = ( $\forall r > 0. \forall e > 0. \exists d > 0. d \leq r \wedge ?Q$ 
 $x \text{ } d \text{ } e$ )

```

if $x \in S$ for x

proof (intro iffI allI impI)

fix $r :: \text{real}$ and $e :: \text{real}$

assume L [rule_format]: $\forall e > 0. \exists d > 0. d \leq e \wedge ?Q \text{ } x \text{ } d \text{ } e$ and $r > 0 \text{ } e > 0$

show $\exists d > 0. d \leq r \wedge ?Q \text{ } x \text{ } d \text{ } e$

using L [of min r e] apply (rule ex_forward)

```

using <r > 0> <e > 0> by (auto intro: less_le_trans elim!: ex_forward simp:
content_ball_pos)

```

qed auto

then show ?thesis

by (force simp: negligible_eq_zero_density_alt)

qed

end

9.32 Change of Variables Theorems

```
theory Change_Of_Vars
  imports Vitali_Covering_Theorem Determinants
```

```
begin
```

9.32.1 Measurable Shear and Stretch

```
proposition
```

```
  fixes a :: realn
  assumes m ≠ n and ab_ne: cbox a b ≠ {} and an: 0 ≤ a$m
  shows measurable_shear_interval: (λx. χ i. if i = m then x$m + x$n else x$i)
  ‘ (cbox a b) ∈ lmeasurable
    (is ?f ‘ _ ∈ _)
  and measure_shear_interval: measure lebesgue ((λx. χ i. if i = m then x$m +
  x$n else x$i) ‘ cbox a b)
    = measure lebesgue (cbox a b) (is ?Q)
```

```
proof –
```

```
  have lin: linear ?f
    by (rule linearI) (auto simp: plus_vec_def scaleR_vec_def algebra_simps)
  show fab: ?f ‘ cbox a b ∈ lmeasurable
    by (simp add: lin measurable_linear_image_interval)
  let ?c = χ i. if i = m then b$m + b$n else b$i
  let ?mn = axis m 1 - axis n (1::real)
  have eq1: measure lebesgue (cbox a ?c)
    = measure lebesgue (?f ‘ cbox a b)
    + measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≤ a$m})
    + measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≥ b$m})
  proof (rule measure_Un3_negligible)
    show cbox a ?c ∩ {x. ?mn · x ≤ a$m} ∈ lmeasurable cbox a ?c ∩ {x. ?mn · x
    ≥ b$m} ∈ lmeasurable
    by (auto simp: convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int
    measurable_convex)
    have negligible {x. ?mn · x = a$m}
      by (metis ⟨m ≠ n⟩ axis_index_axis_eq_iff_diff_eq_0 negligible_hyperplane)
    moreover have ?f ‘ cbox a b ∩ (cbox a ?c ∩ {x. ?mn · x ≤ a $ m}) ⊆ {x.
    ?mn · x = a$m}
      using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
    inner_axis’)
    ultimately show negligible ((?f ‘ cbox a b) ∩ (cbox a ?c ∩ {x. ?mn · x ≤ a $
    m}))
      by (rule negligible_subset)
    have negligible {x. ?mn · x = b$m}
      by (metis ⟨m ≠ n⟩ axis_index_axis_eq_iff_diff_eq_0 negligible_hyperplane)
    moreover have (?f ‘ cbox a b) ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m}) ⊆ {x.
    ?mn · x = b$m}
      using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
    inner_axis’)
    ultimately show negligible (?f ‘ cbox a b ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m}))
```



```

  by (rule negligible_subset)
  have negligible {x. ?mn · x = b$m}
  by (metis ‹m ≠ n› axis_index_axis_eq_iff_diff_eq_0 negligible_hyperplane)
  moreover have (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩ (cbox a ?c ∩ {x. ?mn ·
x ≥ b$m})) ⊆ {x. ?mn · x = b$m}
  using ‹m ≠ n› ab_ne
  apply (clarsimp simp: algebra_simps mem_box_cart inner_axis')
  by (smt (verit, ccfv_SIG) interval_ne_empty_cart(1))
  ultimately show negligible (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩ (cbox a ?c
∩ {x. ?mn · x ≥ b$m}))
  by (rule negligible_subset)
  show ?f ' cbox a b ∪ cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∪ cbox a ?c ∩ {x. ?mn
· x ≥ b$m} = cbox a ?c (is ?lhs = _)
  proof
    show ?lhs ⊆ cbox a ?c
      by (auto simp: mem_box_cart add_mono) (meson add_increasing2 an
order_trans)
    show cbox a ?c ⊆ ?lhs
    apply (clarsimp simp: algebra_simps image_iff inner_axis' lambda_add_Galois
[OF ‹m ≠ n›])
    by (smt (verit, del_insts) mem_box_cart(2) vec_lambda_beta)
  qed
  qed (fact fab)
  let ?d = χ i. if i = m then a $ m - b $ m else 0
  have eq2: measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≤ a $ m}) + measure
lebesgue (cbox a ?c ∩ {x. ?mn · x ≥ b$m})
  = measure lebesgue (cbox a (χ i. if i = m then a $ m + b $ n else b $ i))
  proof (rule measure_translate_add[of cbox a ?c ∩ {x. ?mn · x ≤ a$m} cbox a
?c ∩ {x. ?mn · x ≥ b$m}
  (χ i. if i = m then a$m - b$m else 0) cbox a (χ i. if i = m then a$m + b$n
else b$i)])
    show (cbox a ?c ∩ {x. ?mn · x ≤ a$m}) ∈ lmeasurable
      cbox a ?c ∩ {x. ?mn · x ≥ b$m} ∈ lmeasurable
    by (auto simp: convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int
measurable_convex)
    have ∧x. [x $ n + a $ m ≤ x $ m]
      ⇒ x ∈ (+) (χ i. if i = m then a $ m - b $ m else 0) ' {x. x $ n + b $
m ≤ x $ m}
    using ‹m ≠ n›
    by (rule_tac x=x - (χ i. if i = m then a$m - b$m else 0) in image_eqI)
      (simp_all add: mem_box_cart)
    then have imeq: (+) ?d ' {x. b $ m ≤ ?mn · x} = {x. a $ m ≤ ?mn · x}
    using ‹m ≠ n› by (auto simp: mem_box_cart inner_axis' algebra_simps)
    have ∧x. [0 ≤ a $ n; x $ n + a $ m ≤ x $ m;
      ∀ i. i ≠ m → a $ i ≤ x $ i ∧ x $ i ≤ b $ i]
      ⇒ a $ m ≤ x $ m
    using ‹m ≠ n› by force
    then have (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn · x})
      = cbox a (χ i. if i = m then a $ m + b $ n else b $ i) ∩ {x. a $ m ≤

```

3726

```

?mn · x}
  using an ab_ne
  apply (simp add: cbox_translation [symmetric] translation_Int interval_ne_empty_cart
imeq)
  apply (auto simp: mem_box_cart inner_axis' algebra_simps if_distrib
all_if_distrib)
  by (metis (full_types) add_mono mult_2_right)
  then show cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∪
    (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn · x}) =
    cbox a (χ i. if i = m then a $ m + b $ n else b $ i) (is ?lhs = ?rhs)
  using an ⟨m ≠ n⟩
  apply (auto simp: mem_box_cart inner_axis' algebra_simps if_distrib
all_if_distrib, force)
  apply (drule_tac x=n in spec)+
  by (meson ab_ne add_mono thms linordered_semiring(3) dual_order.trans
interval_ne_empty_cart(1))
  have negligible{x. ?mn · x = a$m}
  by (metis ⟨m ≠ n⟩ axis_index_axis_eq_iff_diff_eq_0 negligible_hyperplane)
  moreover have (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩
    (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn · x})) ⊆ {x.
?mn · x = a$m}
  using ⟨m ≠ n⟩ antisym_conv by (fastforce simp: algebra_simps mem_box_cart
inner_axis')
  ultimately show negligible (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩
    (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn · x}))
  by (rule negligible_subset)
qed
have ac_ne: cbox a ?c ≠ {}
by (smt (verit, del_insts) ab_ne an interval_ne_empty_cart(1) vec_lambda_beta)
have ax_ne: cbox a (χ i. if i = m then a $ m + b $ n else b $ i) ≠ {}
using ab_ne an
by (smt (verit, ccfv_threshold) interval_ne_empty_cart(1) vec_lambda_beta)
have eq3: measure lebesgue (cbox a ?c) = measure lebesgue (cbox a (χ i. if i =
m then a$m + b$n else b$i)) + measure lebesgue (cbox a b)
by (simp add: content_cbox_if_cart ab_ne ac_ne ax_ne algebra_simps prod.delta_remove
if_distrib [of λu. u - z for z] prod.remove)
show ?Q
using eq1 eq2 eq3 by (simp add: algebra_simps)
qed

```

proposition

```

fixes S :: (real^~n) set
assumes S ∈ lmeasurable
shows measurable_stretch: ((λx. χ k. m k * x$k) ' S) ∈ lmeasurable (is ?f ' S
∈ _)
and measure_stretch: measure lebesgue ((λx. χ k. m k * x$k) ' S) = |prod m
UNIV| * measure lebesgue S
(is ?MEQ)

```

```

proof –
  have (?f ‘ S) ∈ lmeasurable ∧ ?MEQ
  proof (cases ∑ k. m k ≠ 0)
    case True
      have m0: 0 < |prod m UNIV|
        using True by simp
      have (indicat_real (?f ‘ S) has_integral |prod m UNIV| * measure lebesgue S)
        UNIV
      proof (clarsimp simp add: has_integral_alt [where i=UNIV])
        fix e :: real
        assume e > 0
        have (indicat_real S has_integral (measure lebesgue S)) UNIV
          using assms lmeasurable_iff_has_integral by blast
        then obtain B where B > 0
          and B: ∧ a b. ball 0 B ⊆ cbox a b ⇒
            ∃ z. (indicat_real S has_integral z) (cbox a b) ∧
              |z – measure lebesgue S| < e / |prod m UNIV|
          by (simp add: has_integral_alt [where i=UNIV]) (metis (full_types)
            divide_pos_pos m0 m0 ‹e > 0›)
        show ∃ B > 0. ∀ a b. ball 0 B ⊆ cbox a b →
          (∃ z. (indicat_real (?f ‘ S) has_integral z) (cbox a b) ∧
            |z – |prod m UNIV| * measure lebesgue S| < e)
      proof (intro exI conjI allI)
        let ?C = Max (range (λk. |m k|)) * B
        show ?C > 0
          using True ‹B > 0› by (simp add: Max_gr_iff)
        show ball 0 ?C ⊆ cbox u v →
          (∃ z. (indicat_real (?f ‘ S) has_integral z) (cbox u v) ∧
            |z – |prod m UNIV| * measure lebesgue S| < e) for u v
      proof
        assume uv: ball 0 ?C ⊆ cbox u v
        with ‹?C > 0› have cbox_ne: cbox u v ≠ {}
          using centre_in_ball by blast
        let ?α = λk. u $ k / m k
        let ?β = λk. v $ k / m k
        have invm0: ∧k. inverse (m k) ≠ 0
          using True by auto
        have ball 0 B ⊆ (λx. χ k. x $ k / m k) ‘ ball 0 ?C
        proof clarsimp
          fix x :: real^n
          assume x: norm x < B
          have [simp]: |Max (range (λk. |m k|))| = Max (range (λk. |m k|))
            by (meson Max_ge abs_ge_zero abs_of_nonneg finite finite_imageI
              order_trans rangeI)
          have norm (χ k. m k * x $ k) ≤ norm (Max (range (λk. |m k|)) *R x)
            by (rule norm_le_componentwise_cart) (auto simp: abs_mult intro:
              mult_right_mono)
          also have ... < ?C
            using x ‹0 < (MAX k. |m k|) * B› ‹0 < B› zero_less_mult_pos2 by

```

```

fastforce
  finally have norm ( $\chi$   $k$ .  $m$   $k$  *  $x$  $  $k$ ) < ? $C$  .
  then show  $x \in (\lambda x. \chi$   $k$ .  $x$  $  $k$  /  $m$   $k$ ) ' ball 0 ? $C$ 
    using stretch_Galois [of inverse  $\circ$   $m$ ] True by (auto simp: image_iff
field_simps)
  qed
  then have Bsub: ball 0  $B \subseteq$  cbox ( $\chi$   $k$ . min (? $\alpha$   $k$ ) (? $\beta$   $k$ )) ( $\chi$   $k$ . max (? $\alpha$ 
 $k$ ) (? $\beta$   $k$ ))
    using cbox_ne uv image_stretch_interval_cart [of inverse  $\circ$   $m$   $u$   $v$ ,
symmetric]
    by (force simp: field_simps)
  obtain  $z$  where zint: (indicat_real  $S$  has_integral  $z$ ) (cbox ( $\chi$   $k$ . min (? $\alpha$ 
 $k$ ) (? $\beta$   $k$ )) ( $\chi$   $k$ . max (? $\alpha$   $k$ ) (? $\beta$   $k$ )))
    and zless:  $|z - \text{measure lebesgue } S| < e / |\text{prod } m \text{ UNIV}|$ 
    using B [OF Bsub] by blast
  have ind: indicat_real (? $f$  '  $S$ ) = ( $\lambda x$ . indicator  $S$  ( $\chi$   $k$ .  $x$ $ $k$  /  $m$   $k$ ))
    using True stretch_Galois [of  $m$ ] by (force simp: indicator_def)
  show  $\exists z$ . (indicat_real (? $f$  '  $S$ ) has_integral  $z$ ) (cbox  $u$   $v$ )  $\wedge$ 
     $|z - |\text{prod } m \text{ UNIV}| * \text{measure lebesgue } S| < e$ 
  proof (simp add: ind, intro conjI exI)
    have (( $\lambda x$ . indicat_real  $S$  ( $\chi$   $k$ .  $x$  $  $k$  /  $m$   $k$ )) has_integral  $z$  * $_R$  | $\text{prod } m$ 
UNIV|)
      (( $\lambda x$ .  $\chi$   $k$ .  $x$  $  $k$  *  $m$   $k$ ) ' cbox ( $\chi$   $k$ . min (? $\alpha$   $k$ ) (? $\beta$   $k$ )) ( $\chi$   $k$ . max
(? $\alpha$   $k$ ) (? $\beta$   $k$ )))
      using True has_integral_stretch_cart [OF zint, of inverse  $\circ$   $m$ ]
      by (simp add: field_simps prod_dividef)
    moreover have (( $\lambda x$ .  $\chi$   $k$ .  $x$  $  $k$  *  $m$   $k$ ) ' cbox ( $\chi$   $k$ . min (? $\alpha$   $k$ ) (? $\beta$   $k$ ))
( $\chi$   $k$ . max (? $\alpha$   $k$ ) (? $\beta$   $k$ ))) = cbox  $u$   $v$ 
      using True image_stretch_interval_cart [of inverse  $\circ$   $m$   $u$   $v$ , symmetric]
      image_stretch_interval_cart [of  $\lambda k$ . 1  $u$   $v$ , symmetric] < cbox  $u$   $v$   $\neq$ 
{}
      by (simp add: field_simps image_comp o_def)
    ultimately show (( $\lambda x$ . indicat_real  $S$  ( $\chi$   $k$ .  $x$  $  $k$  /  $m$   $k$ )) has_integral
 $z$  * $_R$  | $\text{prod } m \text{ UNIV}|$ ) (cbox  $u$   $v$ )
      by simp
    have  $|z$  * $_R$  | $\text{prod } m \text{ UNIV}| - |\text{prod } m \text{ UNIV}| * \text{measure lebesgue } S|$ 
      = | $\text{prod } m \text{ UNIV}| * |z - \text{measure lebesgue } S|$ 
      by (metis (no_types, opaque_lifting) abs_abs abs_scaleR mult.commute
real_scaleR_def right_diff_distrib')
    also have ... <  $e$ 
      using zless True by (simp add: field_simps)
    finally show  $|z$  * $_R$  | $\text{prod } m \text{ UNIV}| - |\text{prod } m \text{ UNIV}| * \text{measure lebesgue}
S| < e$  .
  qed
  qed
  qed
  then show ?thesis
  by (auto simp: has_integral_integrable integral_unique lmeasure_integral_UNIV

```

```

measurable_integrable)
next
  case False
  then obtain k where m k = 0 and prm: prod m UNIV = 0
  by auto
  have nfS: negligible (?f ' S)
  by (rule negligible_subset [OF negligible_standard_hyperplane_cart]) (use ⟨m
k = 0⟩ in auto)
  then show ?thesis
  by (simp add: negligible_iff_measure prm)
qed
then show (?f ' S) ∈ lmeasurable ?MEQ
by metis+
qed

```

proposition

```

fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
assumes linear f S ∈ lmeasurable
shows measurable_linear_image: (f ' S) ∈ lmeasurable
  and measure_linear_image: measure lebesgue (f ' S) = |det (matrix f)| *
measure lebesgue S (is ?Q f S)
proof -
  have ∀ S ∈ lmeasurable. (f ' S) ∈ lmeasurable ∧ ?Q f S
  proof (rule induct_linear_elementary [OF ⟨linear f⟩]; intro ballI)
    fix f g and S :: (real,'n) vec set
    assume linear f and linear g
    and f [rule_format]: ∀ S ∈ lmeasurable. f ' S ∈ lmeasurable ∧ ?Q f S
    and g [rule_format]: ∀ S ∈ lmeasurable. g ' S ∈ lmeasurable ∧ ?Q g S
    and S: S ∈ lmeasurable
    then have gS: g ' S ∈ lmeasurable
    by blast
    show (f ∘ g) ' S ∈ lmeasurable ∧ ?Q (f ∘ g) S
    using f [OF gS] g [OF S] matrix_compose [OF ⟨linear g⟩ ⟨linear f⟩]
    by (simp add: o_def image_comp abs_mult det_mul)
  next
    fix f :: real^'n::_ ⇒ real^'n::_ and i and S :: (real^'n::_) set
    assume linear f and 0: ∧ x. f x $ i = 0 and S ∈ lmeasurable
    then have ¬ inj f
    by (metis (full_types) linear_injective_imp_surjective one_neq_zero surjE
vec_component)
    have detf: det (matrix f) = 0
    using ⟨¬ inj f⟩ det_nz_iff_inj[OF ⟨linear f⟩] by blast
    show f ' S ∈ lmeasurable ∧ ?Q f S
  proof
    show f ' S ∈ lmeasurable
    using lmeasurable_iff_indicator_has_integral ⟨linear f⟩ ⟨¬ inj f⟩ negligi-
ble_UNIV negligible_linear_singular_image by blast
    have measure_lebesgue (f ' S) = 0

```

```

    by (meson <¬ inj f> <linear f> negligible_imp_measure0 negligible_linear_singular_image)
    also have ... = |det (matrix f)| * measure lebesgue S
    by (simp add: detf)
    finally show ?Q f S .
  qed
next
fix c and S :: (real^n::_) set
assume S ∈ lmeasurable
show (λa. χ i. c i * a $ i) ' S ∈ lmeasurable ∧ ?Q (λa. χ i. c i * a $ i) S
proof
  show (λa. χ i. c i * a $ i) ' S ∈ lmeasurable
  by (simp add: <S ∈ lmeasurable> measurable_stretch)
  show ?Q (λa. χ i. c i * a $ i) S
  by (simp add: measure_stretch [OF <S ∈ lmeasurable>, of c] axis_def
matrix_def det_diagonal)
  qed
next
fix m :: 'n and n :: 'n and S :: (real, 'n) vec set
assume m ≠ n and S ∈ lmeasurable
let ?h = λv::(real, 'n) vec. χ i. v $ Transposition.transpose m n i
have lin: linear ?h
  by (rule linearI) (simp_all add: plus_vec_def scaleR_vec_def)
have meq: measure lebesgue ((λv::(real, 'n) vec. χ i. v $ Transposition.transpose
m n i) ' cbox a b)
  = measure lebesgue (cbox a b) for a b
proof (cases cbox a b = {})
  case True then show ?thesis
  by simp
next
  case False
  then have him: ?h ' (cbox a b) ≠ {}
  by blast
  have eq: ?h ' (cbox a b) = cbox (?h a) (?h b)
  by (auto simp: image_iff lambda_swap_Galois mem_box_cart) (metis
transpose_involutory)+
  show ?thesis
  using him prod.permute [OF permutes_swap_id, where S=UNIV and
g=λi. (b - a)$i, symmetric]
  by (simp add: eq_content_cbox_cart False)
  qed
  have (χ i j. if Transposition.transpose m n i = j then 1 else 0) = (χ i j. if j =
Transposition.transpose m n i then 1 else (0::real))
  by (auto intro!: Cart_lambda_cong)
  then have matrix ?h = transpose(χ i j. mat 1 $ i $ Transposition.transpose m
n j)
  by (auto simp: matrix_eq transpose_def axis_def mat_def matrix_def)
  then have 1: |det (matrix ?h)| = 1
  by (simp add: det_permute_columns permutes_swap_id sign_swap_id abs_mult)
  show ?h ' S ∈ lmeasurable ∧ ?Q ?h S

```

```

    using measure_linear_sufficient [OF lin ‹S ∈ lmeasurable›] meq 1 by force
  next
    fix m n :: 'n and S :: (real, 'n) vec set
    assume m ≠ n and S ∈ lmeasurable
    let ?h = λv::(real, 'n) vec. χ i. if i = m then v $ m + v $ n else v $ i
    have lin: linear ?h
      by (rule linearI) (auto simp: algebra_simps plus_vec_def scaleR_vec_def
vec_eq_iff)
    consider m < n | n < m
      using ‹m ≠ n› less_linear by blast
    then have 1: det(matrix ?h) = 1
    proof cases
      assume m < n
      have *: matrix ?h $ i $ j = (0::real) if j < i for i j :: 'n
      proof -
        have axis j 1 = (χ n. if n = j then 1 else (0::real))
          using axis_def by blast
        then have (χ p q. if p = m then axis q 1 $ m + axis q 1 $ n else axis q 1
$ p) $ i $ j = (0::real)
          using ‹j < i› axis_def ‹m < n› by auto
        with ‹m < n› show ?thesis
          by (auto simp: matrix_def axis_def cong: if_cong)
      qed
      show ?thesis
        using ‹m ≠ n› by (subst det_upperdiagonal [OF *]) (auto simp: matrix_def
axis_def cong: if_cong)
    next
      assume n < m
      have *: matrix ?h $ i $ j = (0::real) if j > i for i j :: 'n
      proof -
        have axis j 1 = (χ n. if n = j then 1 else (0::real))
          using axis_def by blast
        then have (χ p q. if p = m then axis q 1 $ m + axis q 1 $ n else axis q 1
$ p) $ i $ j = (0::real)
          using ‹j > i› axis_def ‹m > n› by auto
        with ‹m > n› show ?thesis
          by (auto simp: matrix_def axis_def cong: if_cong)
      qed
      show ?thesis
        using ‹m ≠ n›
        by (subst det_lowerdiagonal [OF *]) (auto simp: matrix_def axis_def cong:
if_cong)
    qed
    have meq: measure lebesgue (?h ‘(cbox a b)) = measure lebesgue (cbox a b) for
a b
    proof (cases cbox a b = {})
      case True then show ?thesis by simp
    next
      case False

```

```

then have ne: (+) (χ i. if i = n then - a $ n else 0) ‘ cbox a b ≠ {}
  by auto
let ?v = χ i. if i = n then - a $ n else 0
have ?h ‘ cbox a b
  = (+) (χ i. if i = m ∨ i = n then a $ n else 0) ‘ ?h ‘ (+) ?v ‘ (cbox a b)
  using ⟨m ≠ n⟩ unfolding image_comp o_def by (force simp: vec_eq_iff)
then have measure lebesgue (?h ‘ (cbox a b))
  = measure lebesgue ((λv. χ i. if i = m then v $ m + v $ n else v $ i) ‘
    (+) ?v ‘ cbox a b)
  by (rule ssubst) (rule measure_translation)
also have ... = measure lebesgue ((λv. χ i. if i = m then v $ m + v $ n else
v $ i) ‘ cbox (?v + a) (?v + b))
  by (metis (no_types, lifting) cbox_translation)
also have ... = measure lebesgue ((+) ?v ‘ cbox a b)
  apply (subst measure_shear_interval)
  using ⟨m ≠ n⟩ ne apply auto
  apply (simp add: cbox_translation)
  by (metis cbox_borel cbox_translation measure_completion_sets_lborel)
also have ... = measure lebesgue (cbox a b)
  by (rule measure_translation)
  finally show ?thesis .
qed
show ?h ‘ S ∈ lmeasurable ∧ ?Q ?h S
  using measure_linear_sufficient [OF lin ⟨S ∈ lmeasurable⟩] meq 1 by force
qed
with assms show (f ‘ S) ∈ lmeasurable ?Q f S
  by metis+
qed

```

lemma

```

fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
assumes f: orthogonal_transformation f and S: S ∈ lmeasurable
shows measurable_orthogonal_image: f ‘ S ∈ lmeasurable
  and measure_orthogonal_image: measure lebesgue (f ‘ S) = measure lebesgue
S
proof –
  have linear f
  by (simp add: f orthogonal_transformation_linear)
then show f ‘ S ∈ lmeasurable
  by (metis S measurable_linear_image)
show measure lebesgue (f ‘ S) = measure lebesgue S
  by (simp add: measure_linear_image ⟨linear f⟩ S f)
qed

```

proposition measure_semicontinuous_with_hausdist_explicit:

assumes bounded S **and** neg: negligible(frontier S) **and** e > 0

obtains d **where** d > 0

$\bigwedge T. \llbracket T \in \text{lmeasurable}; \bigwedge y. y \in T \implies \exists x. x \in S \wedge \text{dist } x y < d \rrbracket$


```

       $\implies \text{measure lebesgue } T < \text{measure lebesgue } S + e$ 
proof (cases  $S = \{\}$ )
  case True
    with that  $\langle e > 0 \rangle$  show ?thesis by force
next
  case False
    then have  $\text{fr}S: \text{frontier } S \neq \{\}$ 
      using  $\langle \text{bounded } S \rangle \text{frontier\_eq\_empty\_not\_bounded\_UNIV}$  by blast
    have  $S \in \text{lmeasurable}$ 
      by (simp add:  $\langle \text{bounded } S \rangle \text{measurable\_Jordan\_neg}$ )
    have  $\text{null}: (\text{frontier } S) \in \text{null\_sets lebesgue}$ 
      by (metis neg negligible_iff_null_sets)
    have  $\text{frontier } S \in \text{lmeasurable}$  and  $mS0: \text{measure lebesgue } (\text{frontier } S) = 0$ 
      using neg negligible_imp_measurable negligible_iff_measure by blast+
    with  $\langle e > 0 \rangle \text{sets\_lebesgue\_outer\_open}$ 
    obtain  $U$  where  $\text{open } U$ 
      and  $U: \text{frontier } S \subseteq U$   $U - \text{frontier } S \in \text{lmeasurable}$   $\text{emeasure lebesgue } (U - \text{frontier } S) < e$ 
      by (metis fmeasurableD)
    with null have  $U \in \text{lmeasurable}$ 
      by (metis borel_open measurable_Diff_null_set sets_completionI_sets sets_lborel)
    have  $\text{measure lebesgue } (U - \text{frontier } S) = \text{measure lebesgue } U$ 
      using  $mS0$  by (simp add:  $\langle U \in \text{lmeasurable} \rangle \text{fmeasurableD measure\_Diff\_null\_set null}$ )
    with  $U$  have  $mU: \text{measure lebesgue } U < e$ 
      by (simp add:  $\text{emeasure\_eq\_measure2 ennreal\_less\_iff}$ )
    show ?thesis
  proof
    have  $U \neq \text{UNIV}$ 
      using  $\langle U \in \text{lmeasurable} \rangle$  by auto
    then have  $- U \neq \{\}$ 
      by blast
    with  $\langle \text{open } U \rangle \langle \text{frontier } S \subseteq U \rangle$  show  $\text{setdist } (\text{frontier } S) (- U) > 0$ 
      by (auto simp:  $\langle \text{bounded } S \rangle \text{open\_closed compact\_frontier\_bounded setdist\_gt\_0\_compact\_closed fr}S$ )
    fix  $T$ 
    assume  $T \in \text{lmeasurable}$ 
      and  $T: \bigwedge t. t \in T \implies \exists y. y \in S \wedge \text{dist } y t < \text{setdist } (\text{frontier } S) (- U)$ 
    then have  $\text{measure lebesgue } T - \text{measure lebesgue } S \leq \text{measure lebesgue } (T - S)$ 
      by (simp add:  $\langle S \in \text{lmeasurable} \rangle \text{measure\_diff\_le\_measure\_setdiff}$ )
    also have  $\dots \leq \text{measure lebesgue } U$ 
  proof -
    have  $T - S \subseteq U$ 
  proof clarify
    fix  $x$ 
    assume  $x \in T$  and  $x \notin S$ 
    then obtain  $y$  where  $y \in S$  and  $y: \text{dist } y x < \text{setdist } (\text{frontier } S) (- U)$ 
      using  $T$  by blast

```

```

have closed_segment x y ∩ frontier S ≠ {}
  using connected_Int_frontier ⟨x ∉ S⟩ ⟨y ∈ S⟩ by blast
then obtain z where z: z ∈ closed_segment x y z ∈ frontier S
  by auto
with y have dist z x < setdist(frontier S) (- U)
  by (auto simp: dist_commute dest!: dist_in_closed_segment)
with z have False if x ∈ -U
  using setdist_le_dist [OF ⟨z ∈ frontier S⟩ that] by auto
then show x ∈ U
  by blast
qed
then show ?thesis
  by (simp add: ⟨S ∈ lmeasurable⟩ ⟨T ∈ lmeasurable⟩ ⟨U ∈ lmeasurable⟩
fmeasurableD measure_mono_fmeasurable sets.Diff)
qed
finally have measure_lebesgue T - measure_lebesgue S ≤ measure_lebesgue U .
with mU show measure_lebesgue T < measure_lebesgue S + e
  by linarith
qed
qed

```

proposition

```

fixes f :: real^n::{finite,wellorder} ⇒ real^n::_
assumes S: S ∈ lmeasurable
and deriv: ∧x. x ∈ S ⇒ (f has_derivative f' x) (at x within S)
and int: (λx. |det (matrix (f' x))|) integrable_on S
and bounded: ∧x. x ∈ S ⇒ |det (matrix (f' x))| ≤ B
shows measurable_bounded_differentiable_image:
  f ' S ∈ lmeasurable
and measure_bounded_differentiable_image:
  measure_lebesgue (f ' S) ≤ B * measure_lebesgue S (is ?M)
proof -
have f ' S ∈ lmeasurable ∧ measure_lebesgue (f ' S) ≤ B * measure_lebesgue S
proof (cases B < 0)
case True
then have S = {}
  by (meson abs_ge_zero bounded_empty_iff_equalityI less_le_trans linorder_not_less
subsetI)
then show ?thesis
  by auto
next
case False
then have B ≥ 0
  by arith
let ?μ = measure_lebesgue
have f_diff: f differentiable_on S
  using deriv by (auto simp: differentiable_on_def differentiable_def)
have eps: f ' S ∈ lmeasurable ?μ (f ' S) ≤ (B+e) * ?μ S (is ?ME)
  if e > 0 for e

```

```

proof –
  have  $\text{eps\_}d: f' S \in \text{lmeasurable } ?\mu (f' S) \leq (B+e) * (?\mu S + d)$  (is ?MD)
    if  $d > 0$  for  $d$ 
  proof –
    obtain  $T$  where  $T: \text{open } T \ S \subseteq T$  and  $TS: (T-S) \in \text{lmeasurable}$  and
     $\text{emeasure lebesgue } (T-S) < \text{ennreal } d$ 
    using  $S \langle d > 0 \rangle \text{ sets\_lebesgue\_outer\_open}$  by blast
    then have  $?\mu (T-S) < d$ 
    by (metis emeasure_eq_measure2 ennreal_leI not_less)
    with  $S \ T \ TS$  have  $T \in \text{lmeasurable}$  and  $T\text{less}: ?\mu T < ?\mu S + d$ 
    by (auto simp: measurable_measure_Diff dest!: fmeasurable_Diff_D)
    have  $\exists r. 0 < r \wedge r < d \wedge \text{ball } x \ r \subseteq T \wedge f' (S \cap \text{ball } x \ r) \in \text{lmeasurable} \wedge$ 
       $?\mu (f' (S \cap \text{ball } x \ r)) \leq (B + e) * ?\mu (\text{ball } x \ r)$ 
    if  $x \in S$   $d > 0$  for  $x \ d$ 
    proof –
      have  $\text{lin}: \text{linear } (f' x)$ 
      and  $\text{lim0}: ((\lambda y. (f y - (f x + f' x (y - x))) /_R \text{norm}(y - x)) \longrightarrow 0)$ 
      (at x within S)
      using  $\text{deriv } \langle x \in S \rangle$  by (auto simp: has_derivative_within bounded_linear.linear
      field_simps)
      have  $\text{bo}: \text{bounded } (f' x \text{ ' ball } 0 \ 1)$ 
      by (simp add: bounded_linear_image linear_linear lin)
      have  $\text{neg}: \text{negligible } (\text{frontier } (f' x \text{ ' ball } 0 \ 1))$ 
      using  $\text{deriv has\_derivative\_linear } \langle x \in S \rangle$ 
      by (auto intro!: negligible_convex_frontier [OF convex_linear_image])
      let  $?\text{unit\_vol} = \text{content } (\text{ball } (0 :: \text{real} \ ^n :: \{\text{finite}, \text{wellorder}\}) \ 1)$ 
      have  $0: 0 < e * ?\text{unit\_vol}$ 
      using  $\langle e > 0 \rangle$  by (simp add: content_ball_pos)
      obtain  $k$  where  $k > 0$  and  $k:$ 
       $\bigwedge U. \llbracket U \in \text{lmeasurable}; \bigwedge y. y \in U \implies \exists z. z \in f' x \text{ ' ball } 0 \ 1 \wedge \text{dist}$ 
       $z \ y < k \rrbracket$ 
       $\implies ?\mu U < ?\mu (f' x \text{ ' ball } 0 \ 1) + e * ?\text{unit\_vol}$ 
      using  $\text{measure\_semicontinuous\_with\_hausdist\_explicit}$  [OF bo neg 0]
    by blast
    obtain  $l$  where  $l > 0$  and  $l: \text{ball } x \ l \subseteq T$ 
    using  $\langle x \in S \rangle \langle \text{open } T \rangle \langle S \subseteq T \rangle \text{openE}$  by blast
    obtain  $\zeta$  where  $0 < \zeta$ 
    and  $\zeta: \bigwedge y. \llbracket y \in S; y \neq x; \text{dist } y \ x < \zeta \rrbracket$ 
       $\implies \text{norm } (f y - (f x + f' x (y - x))) / \text{norm } (y - x) < k$ 
    using  $\text{lim0 } \langle k > 0 \rangle$  by (simp add: Lim_within) (auto simp add:
    field_simps)
    define  $r$  where  $r \equiv \min (\min l (\zeta/2)) (\min 1 (d/2))$ 
    show ?thesis
    proof (intro exI conjI)
      show  $r > 0$   $r < d$ 
      using  $\langle l > 0 \rangle \langle \zeta > 0 \rangle \langle d > 0 \rangle$  by (auto simp: r_def)
      have  $r \leq l$ 
      by (auto simp: r_def)
      with  $l$  show  $\text{ball } x \ r \subseteq T$ 

```

```

    by auto
  have ex_lessK:  $\exists x' \in \text{ball } 0 \ 1. \text{ dist } (f' \ x \ x') ((f \ y - f \ x) /_R \ r) < k$ 
    if  $y \in S$  and  $\text{dist } x \ y < r$  for  $y$ 
  proof (cases  $y = x$ )
    case True
      with lin linear_0  $\langle k > 0 \rangle$  that show ?thesis
        by (rule_tac  $x=0$  in bexI) (auto simp: linear_0)
    next
      case False
        then show ?thesis
          proof (rule_tac  $x=(y - x) /_R \ r$  in bexI)
            have  $f' \ x ((y - x) /_R \ r) = f' \ x (y - x) /_R \ r$ 
              by (simp add: lin linear_scale)
            then have  $\text{dist } (f' \ x ((y - x) /_R \ r)) ((f \ y - f \ x) /_R \ r) = \text{norm } (f' \ x (y - x) /_R \ r - (f \ y - f \ x) /_R \ r)$ 
              by (simp add: dist_norm)
            also have  $\dots = \text{norm } (f' \ x (y - x) - (f \ y - f \ x)) / r$ 
              using  $\langle r > 0 \rangle$  by (simp add: divide_simps scale_right_diff_distrib [symmetric])
            also have  $\dots \leq \text{norm } (f \ y - (f \ x + f' \ x (y - x))) / \text{norm } (y - x)$ 
              using that  $\langle r > 0 \rangle$  False by (simp add: field_split_simps dist_norm norm_minus_commute mult_right_mono)
            also have  $\dots < k$ 
              using that  $\langle 0 < \zeta \rangle$  by (simp add: dist_commute r_def  $\zeta$  [OF  $\langle y \in S \rangle$  False])
            finally show  $\text{dist } (f' \ x ((y - x) /_R \ r)) ((f \ y - f \ x) /_R \ r) < k$  .
          show  $(y - x) /_R \ r \in \text{ball } 0 \ 1$ 
            using that  $\langle r > 0 \rangle$  by (simp add: dist_norm divide_simps norm_minus_commute)
          qed
        qed
      let ?rfs =  $(\lambda x. x /_R \ r) \text{ ' } (+) \text{ ' } (- \ f \ x) \text{ ' } f \text{ ' } (S \cap \text{ball } x \ r)$ 
      have rfs_mble: ?rfs  $\in$  lmeasurable
      proof (rule bounded_set_imp_lmeasurable)
        have  $f$  differentiable_on  $S \cap \text{ball } x \ r$ 
          using  $f\_diff$  by (auto simp: fmeasurableD differentiable_on_subset)
        with  $S$  show ?rfs  $\in$  sets lebesgue
          by (auto simp: sets.Int intro!: lebesgue_sets_translation differentiable_image_in_sets_lebesgue)
        let ?B =  $(\lambda(x, y). x + y) \text{ ' } (f' \ x \text{ ' } \text{ball } 0 \ 1 \times \text{ball } 0 \ k)$ 
        have bounded ?B
          by (simp add: bounded_plus [OF bo])
        moreover have ?rfs  $\subseteq$  ?B
          apply (auto simp: dist_norm image_iff dest!: ex_lessK)
          by (metis (no_types, opaque_lifting) add.commute diff_add_cancel dist_0_norm dist_commute dist_norm mem_ball)
        ultimately show bounded (?rfs)
          by (rule bounded_subset)
      qed
    qed
  
```

```

then have  $(\lambda x. r *_{\mathbb{R}} x) \text{ ' } ?rfs \in lmeasurable$ 
  by (simp add: measurable_linear_image)
with  $\langle r > 0 \rangle$  have  $(+) (- f x) \text{ ' } f \text{ ' } (S \cap ball\ x\ r) \in lmeasurable$ 
  by (simp add: image_comp o_def)
then have  $(+) (f x) \text{ ' } (+) (- f x) \text{ ' } f \text{ ' } (S \cap ball\ x\ r) \in lmeasurable$ 
  using measurable_translation by blast
then show  $fsb: f \text{ ' } (S \cap ball\ x\ r) \in lmeasurable$ 
  by (simp add: image_comp o_def)
have  $? \mu (f \text{ ' } (S \cap ball\ x\ r)) = ? \mu (?rfs) * r \wedge CARD('n)$ 
  using  $\langle r > 0 \rangle$  fsb
  by (simp add: measure_linear_image measure_translation_subtract
measurable_translation_subtract field_simps cong: image_cong_simp)
also have  $\dots \leq (|det (matrix (f' x))| * ?unit\_vol + e * ?unit\_vol) * r$ 
 $\wedge CARD('n)$ 
  proof -
    have  $? \mu (?rfs) < ? \mu (f' x \text{ ' } ball\ 0\ 1) + e * ?unit\_vol$ 
      using rfs_mble by (force intro: k dest!: ex_lessK)
    then have  $? \mu (?rfs) < |det (matrix (f' x))| * ?unit\_vol + e * ?unit\_vol$ 
      by (simp add: lin_measure_linear_image [of f' x])
    with  $\langle r > 0 \rangle$  show ?thesis
      by auto
  qed
also have  $\dots \leq (B + e) * ? \mu (ball\ x\ r)$ 
  using bounded [OF  $\langle x \in S \rangle$   $\langle r > 0 \rangle$ ]
  by (simp add: algebra_simps content_ball_conv_unit_ball [of r]
content_ball_pos)
finally show  $? \mu (f \text{ ' } (S \cap ball\ x\ r)) \leq (B + e) * ? \mu (ball\ x\ r) .$ 
qed
qed
then obtain  $r$  where
   $r0d: \bigwedge x d. \llbracket x \in S; d > 0 \rrbracket \implies 0 < r x d \wedge r x d < d$ 
  and  $rT: \bigwedge x d. \llbracket x \in S; d > 0 \rrbracket \implies ball\ x\ (r x d) \subseteq T$ 
  and  $r: \bigwedge x d. \llbracket x \in S; d > 0 \rrbracket \implies$ 
     $(f \text{ ' } (S \cap ball\ x\ (r x d))) \in lmeasurable \wedge$ 
     $? \mu (f \text{ ' } (S \cap ball\ x\ (r x d))) \leq (B + e) * ? \mu (ball\ x\ (r x d))$ 
  by metis
obtain  $C$  where countable  $C$  and  $Csub: C \subseteq \{(x, r x t) \mid x t. x \in S \wedge 0 <$ 
 $t\}$ 
  and  $pwC: pairwise (\lambda i j. disjnt (ball (fst i) (snd i)) (ball (fst j) (snd j)))$ 
 $C$ 
  and  $negC: negligible(S - (\bigcup i \in C. ball (fst i) (snd i)))$ 
  apply (rule Vitali_covering_theorem_balls [of  $S \{(x, r x t) \mid x t. x \in S \wedge$ 
 $0 < t\}$  fst snd])
  apply auto
  by (metis dist_eq_0_iff r0d)
let  $?UB = (\bigcup (x,s) \in C. ball\ x\ s)$ 
have  $eq: f \text{ ' } (S \cap ?UB) = (\bigcup (x,s) \in C. f \text{ ' } (S \cap ball\ x\ s))$ 
  by auto
have  $mle: ? \mu (\bigcup (x,s) \in C. f \text{ ' } (S \cap ball\ x\ s)) \leq (B + e) * (? \mu S + d)$  (is

```

```

?l ≤ ?r)
  if K ⊆ C and finite K for K
  proof -
    have gt0: b > 0 if (a, b) ∈ K for a b
      using Csub that ⟨K ⊆ C⟩ r0d by auto
    have inj: inj_on (λ(x, y). ball x y) K
      by (force simp: inj_on_def ball_eq_ball_iff dest: gt0)
    have disjnt: disjoint ((λ(x, y). ball x y) ‘ K)
      using pwC that pairwise_image pairwise_mono by fastforce
    have ?l ≤ (∑ i∈K. ?μ (case i of (x, s) ⇒ f ‘ (S ∩ ball x s)))
      proof (rule measure_UNION_le [OF ⟨finite K⟩], clarify)
        fix x r
        assume (x, r) ∈ K
        then have x ∈ S
          using Csub ⟨K ⊆ C⟩ by auto
        show f ‘ (S ∩ ball x r) ∈ sets lebesgue
          by (meson Int_lower1 S differentiable_on_subset f_diff fmeasurableD
lmeasurable_ball order_refl sets.Int differentiable_image_in_sets_lebesgue)
        qed
        also have ... ≤ (∑ (x, s) ∈ K. (B + e) * ?μ (ball x s))
          using Csub r ⟨K ⊆ C⟩ by (intro sum_mono) auto
        also have ... = (B + e) * (∑ (x, s) ∈ K. ?μ (ball x s))
          by (simp add: prod.case_distrib sum_distrib_left)
        also have ... = (B + e) * sum ?μ ((λ(x, y). ball x y) ‘ K)
          using ⟨B ≥ 0⟩ ⟨e > 0⟩ by (simp add: inj sum.reindex prod.case_distrib)
        also have ... = (B + e) * ?μ (⋃ (x, s) ∈ K. ball x s)
          using ⟨B ≥ 0⟩ ⟨e > 0⟩ that
          by (subst measure_Union') (auto simp: disjnt measure_Union')
        also have ... ≤ (B + e) * ?μ T
          using ⟨B ≥ 0⟩ ⟨e > 0⟩ that apply simp
        using measure_mono_fmeasurable [OF __ ⟨T ∈ lmeasurable⟩] Csub rT
          by (smt (verit) SUP_least measure_nonneg measure_notin_sets
mem_Collect_eq old.prod.case_subset_iff)
        also have ... ≤ (B + e) * (?μ S + d)
          using ⟨B ≥ 0⟩ ⟨e > 0⟩ Tless by simp
        finally show ?thesis .
      qed
    have fSUB_mble: (f ‘ (S ∩ ?UB)) ∈ lmeasurable
      unfolding eq using Csub r False ⟨e > 0⟩ that
      by (auto simp: intro!: fmeasurable_UN_bound [OF ⟨countable C⟩ _ mle])
    have fSUB_meas: ?μ (f ‘ (S ∩ ?UB)) ≤ (B + e) * (?μ S + d) (is ?MUB)
      unfolding eq using Csub r False ⟨e > 0⟩ that
      by (auto simp: intro!: measure_UN_bound [OF ⟨countable C⟩ _ mle])
    have neg: negligible ((f ‘ (S ∩ ?UB)) - f ‘ S) ∪ (f ‘ S - f ‘ (S ∩ ?UB))
      proof (rule negligible_subset [OF negligible_differentiable_image_negligible
[OF order_refl negC, where f=f]])
        show f differentiable_on S - (⋃ i∈C. ball (fst i) (snd i))
          by (meson DiffE differentiable_on_subset subsetI f_diff)
      qed force

```

```

    show  $f' S \in lmeasurable$ 
      by (rule lmeasurable_negligible_symdiff [OF fSUB_mble neg])
    show ?MD
      using fSUB_meas measure_negligible_symdiff [OF fSUB_mble neg] by
simp
  qed
  show  $f' S \in lmeasurable$ 
    using eps_d [of 1] by simp
  show ?ME
  proof (rule field_le_epsilon)
    fix  $\delta :: real$ 
    assume  $0 < \delta$ 
    then show  $? \mu (f' S) \leq (B + e) * ? \mu S + \delta$ 
      using eps_d [of  $\delta / (B+e)$ ] <e > 0 <B  $\geq$  0> by (auto simp: divide_simps
mult_ac)
  qed
  qed
  show ?thesis
  proof (cases  $? \mu S = 0$ )
    case True
    with eps have  $? \mu (f' S) = 0$ 
      by (metis mult_zero_right not_le zero_less_measure_iff)
    then show ?thesis
      using eps [of 1] by (simp add: True)
  next
    case False
    have  $? \mu (f' S) \leq B * ? \mu S$ 
    proof (rule field_le_epsilon)
      fix  $e :: real$ 
      assume  $e > 0$ 
      then show  $? \mu (f' S) \leq B * ? \mu S + e$ 
        using eps [of e /  $? \mu S$ ] False by (auto simp: algebra_simps zero_less_measure_iff)
    qed
    with eps [of 1] show ?thesis by auto
  qed
  qed
  then show  $f' S \in lmeasurable ?M$  by blast+
qed

lemma m_diff_image_weak:
  fixes  $f :: real^n::\{finite, wellorder\} \Rightarrow real^n::\_$ 
  assumes  $S: S \in lmeasurable$ 
    and deriv:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x)$  (at  $x$  within  $S$ )
    and int:  $(\lambda x. |det (matrix (f' x))|) \text{ integrable\_on } S$ 
  shows  $f' S \in lmeasurable \wedge \text{measure lebesgue } (f' S) \leq \text{integral } S (\lambda x. |det$ 
 $(matrix (f' x))|)$ 
  proof -
    let  $? \mu = \text{measure lebesgue}$ 
    have aint_S:  $(\lambda x. |det (matrix (f' x))|) \text{ absolutely\_integrable\_on } S$ 

```

```

    using int unfolding absolutely_integrable_on_def by auto
    define m where m ≡ integral S (λx. |det (matrix (f' x))|)
    have *: f' S ∈ lmeasurable ?μ (f' S) ≤ m + e * ?μ S
    if e > 0 for e
  proof -
    define T where T ≡ λn. {x ∈ S. n * e ≤ |det (matrix (f' x))| ∧
      |det (matrix (f' x))| < (Suc n) * e}
    have meas_t: T n ∈ lmeasurable for n
    proof -
      have *: (λx. |det (matrix (f' x))|) ∈ borel_measurable (lebesgue_on S)
      using aint_S by (simp add: S borel_measurable_restrict_space_iff fmeasurableD set_integrable_def)
      have [intro]: x ∈ sets (lebesgue_on S) ⇒ x ∈ sets lebesgue for x
      using S sets_restrict_space_subset by blast
      have {x ∈ S. real n * e ≤ |det (matrix (f' x))|} ∈ sets lebesgue
      using * by (auto simp: borel_measurable_iff_halfspace_ge space_restrict_space)
      then have 1: {x ∈ S. real n * e ≤ |det (matrix (f' x))|} ∈ lmeasurable
      using S by (simp add: fmeasurableI2)
      have {x ∈ S. |det (matrix (f' x))| < (1 + real n) * e} ∈ sets lebesgue
      using * by (auto simp: borel_measurable_iff_halfspace_less space_restrict_space)
      then have 2: {x ∈ S. |det (matrix (f' x))| < (1 + real n) * e} ∈ lmeasurable
      using S by (simp add: fmeasurableI2)
      show ?thesis
      using fmeasurable.Int [OF 1 2] by (simp add: T_def Int_def cong:
conj_cong)
    qed
    have aint_T: ∧k. (λx. |det (matrix (f' x))|) absolutely_integrable_on T k
    using set_integrable_subset [OF aint_S] meas_t T_def by blast
    have Seq: S = (⋃ n. T n)
    apply (auto simp: T_def)
    apply (rule_tac x = nat [|det (matrix (f' x))| / e] in exI)
    by (smt (verit, del_insts) divide_nonneg_nonneg floor_eq_iff of_nat_nat
pos_divide_less_eq that zero_le_floor)
    have meas_ft: f' T n ∈ lmeasurable for n
    proof (rule measurable_bounded_differentiable_image)
      show T n ∈ lmeasurable
      by (simp add: meas_t)
    next
    fix x :: (real,'n) vec
    assume x ∈ T n
    show (f has_derivative f' x) (at x within T n)
    by (metis (no_types, lifting) ⟨x ∈ T n⟩ deriv_has_derivative_subset
mem_Collect_eq subsetI T_def)
    show |det (matrix (f' x))| ≤ (Suc n) * e
    using ⟨x ∈ T n⟩ T_def by auto
  next
  show (λx. |det (matrix (f' x))|) integrable_on T n
  using aint_T absolutely_integrable_on_def by blast
  qed

```



```

have disT: disjoint (range T)
  unfolding disjoint_def
proof clarsimp
show  $T\ m \cap T\ n = \{\}$  if  $T\ m \neq T\ n$  for  $m\ n$ 
  using that
proof (induction m n rule: linorder_less_wlog)
case (less m n)
with  $\langle e > 0 \rangle$  show ?case
  unfolding T_def
  proof (clarsimp simp add: Collect_conj_eq [symmetric])
  fix x
  assume  $e > 0\ m < n\ n * e \leq |\det(\text{matrix}(f'\ x))| - |\det(\text{matrix}(f'\ x))| < (1 + \text{real}\ m) * e$ 
  then have  $n < 1 + \text{real}\ m$ 
    by (metis (no_types, opaque_lifting) less_le_trans mult.commute not_le mult_le_cancel_left_pos)
  then show False
    using less.hyps by linarith
  qed
qed auto
qed
have injT: inj_on T ( $\{n.\ T\ n \neq \{\}\}$ )
  unfolding inj_on_def
proof clarsimp
show  $m = n$  if  $T\ m = T\ n\ T\ n \neq \{\}$  for  $m\ n$ 
  using that
proof (induction m n rule: linorder_less_wlog)
case (less m n)
have False if  $T\ n \subseteq T\ m\ x \in T\ n$  for  $x$ 
  using  $\langle e > 0 \rangle\ \langle m < n \rangle$  that
apply (auto simp: T_def mult.commute intro: less_le_trans dest!: subsetD)
  by (smt (verit, best) mult_less_cancel_left_disj nat_less_real_le)
then show ?case
  using less.premis by blast
qed auto
qed
have sum_eq_Tim:  $(\sum k \leq n. f(T\ k)) = \text{sum}\ f\ (T\ \{\dots n\})$  if  $f\ \{\} = 0$  for  $f :: \_ \Rightarrow \text{real}$  and  $n$ 
proof (subst sum.reindex_nontrivial)
fix  $i\ j$  assume  $i \in \{\dots n\}\ j \in \{\dots n\}\ i \neq j\ T\ i = T\ j$ 
with that injT [unfolded inj_on_def] show  $f(T\ i) = 0$ 
  by simp metis
qed (use atMost_atLeast0 in auto)
let ?B =  $m + e * ?\mu\ S$ 
have  $(\sum k \leq n. ?\mu(f\ ' T\ k)) \leq ?B$  for  $n$ 
proof -
have  $(\sum k \leq n. ?\mu(f\ ' T\ k)) \leq (\sum k \leq n. ((k+1) * e) * ?\mu(T\ k))$ 
proof (rule sum_mono [OF measure_bounded_differentiable_image])
show (f has_derivative f' x) (at x within T k) if  $x \in T\ k$  for  $k\ x$ 

```

```

    using that unfolding T_def by (blast intro: deriv has_derivative_subset)
  show (λx. |det (matrix (f' x))|) integrable_on T k for k
    using absolutely_integrable_on_def aint_T by blast
  show |det (matrix (f' x))| ≤ real (k + 1) * e if x ∈ T k for k x
    using T_def that by auto
qed (use meas_t in auto)
also have ... ≤ (∑ k≤n. (k * e) * ?μ(T k)) + (∑ k≤n. e * ?μ(T k))
  by (simp add: algebra_simps sum.distrib)
also have ... ≤ ?B
proof (rule add_mono)
  have (∑ k≤n. real k * e * ?μ (T k)) = (∑ k≤n. integral (T k) (λx. k * e))
    by (simp add: lmeasure_integral [OF meas_t]
        flip: integral_mult_right integral_mult_left)
  also have ... ≤ (∑ k≤n. integral (T k) (λx. (abs (det (matrix (f' x)))))
proof (rule sum_mono)
  fix k
  assume k ∈ {..n}
  show integral (T k) (λx. k * e) ≤ integral (T k) (λx. |det (matrix (f' x))|)
  proof (rule integral_le [OF integrable_on_const [OF meas_t]])
    show (λx. |det (matrix (f' x))|) integrable_on T k
      using absolutely_integrable_on_def aint_T by blast
  next
    fix x assume x ∈ T k
    show k * e ≤ |det (matrix (f' x))|
      using ⟨x ∈ T k⟩ T_def by blast
  qed
qed
also have ... = sum (λT. integral T (λx. |det (matrix (f' x))|)) (T ' {..n})
  by (auto intro: sum_eq_Tim)
also have ... = integral (⋃ k≤n. T k) (λx. |det (matrix (f' x))|)
proof (rule integral_unique [OF has_integral_Union, symmetric])
  fix S assume S ∈ T ' {..n}
  then show ((λx. |det (matrix (f' x))|) has_integral integral S (λx. |det
(matrix (f' x))|)) S
    using absolutely_integrable_on_def aint_T by blast
  next
    show pairwise (λS S'. negligible (S ∩ S')) (T ' {..n})
      using disT unfolding disjnt_iff by (auto simp: pairwise_def intro!:
empty_imp_negligible)
  qed auto
also have ... ≤ m
  unfolding m_def
proof (rule integral_subset_le)
  have (λx. |det (matrix (f' x))|) absolutely_integrable_on (⋃ k≤n. T k)
  proof (rule set_integrable_subset [OF aint_S])
    show ⋃ (T ' {..n}) ∈ sets lebesgue
      by (intro measurable meas_t fmeasurableD)
  qed (force simp: Seq)
  then show (λx. |det (matrix (f' x))|) integrable_on (⋃ k≤n. T k)

```

```

    using absolutely_integrable_on_def by blast
  qed (use Seq_int in auto)
  finally show  $(\sum_{k \leq n}. \text{real } k * e * ?\mu (T k)) \leq m .$ 
next
  have  $(\sum_{k \leq n}. ?\mu (T k)) = \text{sum } ?\mu (T \text{ ' } \{..n\})$ 
    by (auto intro: sum_eq_Tim)
  also have  $\dots = ?\mu (\bigcup_{k \leq n}. T k)$ 
    using S disT by (auto simp: pairwise_def meas_t intro: measure_Union'
[symmetric])
  also have  $\dots \leq ?\mu S$ 
    using S by (auto simp: Seq_intro: meas_t fmeasurableD measure_mono_fmeasurable)
  finally have  $(\sum_{k \leq n}. ?\mu (T k)) \leq ?\mu S .$ 
  then show  $(\sum_{k \leq n}. e * ?\mu (T k)) \leq e * ?\mu S$ 
    by (metis less_eq_real_def ordered_comm_semiring_class.comm_mult_left_mono
sum_distrib_left that)
  qed
  finally show  $(\sum_{k \leq n}. ?\mu (f \text{ ' } T k)) \leq ?B .$ 
qed
moreover have measure_lebesgue  $(\bigcup_{k \leq n}. f \text{ ' } T k) \leq (\sum_{k \leq n}. ?\mu (f \text{ ' } T k))$ 
for n
  by (simp add: fmeasurableD meas_ft measure_UNION_le)
ultimately have B_ge_m:  $?\mu (\bigcup_{k \leq n}. (f \text{ ' } T k)) \leq ?B$  for n
  by (meson order_trans)
have  $(\bigcup_n. f \text{ ' } T n) \in \text{lmeasurable}$ 
  by (rule fmeasurable_countable_Union [OF meas_ft B_ge_m])
moreover have  $?\mu (\bigcup_n. f \text{ ' } T n) \leq m + e * ?\mu S$ 
  by (rule measure_countable_Union_le [OF meas_ft B_ge_m])
ultimately show  $f \text{ ' } S \in \text{lmeasurable } ?\mu (f \text{ ' } S) \leq m + e * ?\mu S$ 
  by (auto simp: Seq_image_Union)
qed
show ?thesis
proof
  show  $f \text{ ' } S \in \text{lmeasurable}$ 
    using * linordered_field_no_ub by blast
  let ?x =  $m - ?\mu (f \text{ ' } S)$ 
  have False if  $?\mu (f \text{ ' } S) > \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
  proof -
    have ml:  $m < ?\mu (f \text{ ' } S)$ 
      using m_def that by blast
    then have  $?\mu S \neq 0$ 
      using *(2) bgauge_existence_lemma by fastforce
    with ml have 0:  $0 < -(m - ?\mu (f \text{ ' } S))/2 / ?\mu S$ 
      using that zero_less_measure_iff by force
    then show ?thesis
      using *[OF 0] that by (auto simp: field_split_simps m_def split: if_split_asm)
  qed
  then show  $?\mu (f \text{ ' } S) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
    by fastforce
qed

```

qed

theorem

```

fixes  $f :: \text{real}^n :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n :: \_$ 
assumes  $S: S \in \text{sets lebesgue}$ 
  and  $\text{deriv}: \bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } S)$ 
  and  $\text{int}: (\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } S$ 
shows  $\text{measurable\_differentiable\_image}: f' S \in \text{lmeasurable}$ 
  and  $\text{measure\_differentiable\_image}: \text{measure lebesgue } (f' S) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|) \text{ (is ?M)}$ 
proof -
  let  $?I = \lambda n :: \text{nat}. \text{cbox } (\text{vec } (-n)) (\text{vec } n) \cap S$ 
  let  $?\mu = \text{measure lebesgue}$ 
  have  $x \in \text{cbox } (\text{vec } (- \text{real } (\text{nat } [\text{norm } x]))) (\text{vec } (\text{real } (\text{nat } [\text{norm } x])))$  for  $x :: \text{real}^n :: \_$ 
    apply ( $\text{simp add: mem\_box\_cart}$ )
    by ( $\text{smt } (\text{verit, best}) \text{ component\_le\_norm\_cart le\_of\_int\_ceiling}$ )
  then have  $\text{Seq}: S = (\bigcup n. ?I n)$ 
    by  $\text{blast}$ 
  have  $fIn: f' ?I n \in \text{lmeasurable}$ 
    and  $mfIn: ?\mu (f' ?I n) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|) \text{ (is ?MN)}$  for
 $n$ 
  proof -
    have  $In: ?I n \in \text{lmeasurable}$ 
      by ( $\text{simp add: } S \text{ bounded\_Int bounded\_set\_imp\_lmeasurable sets.Int}$ )
    moreover have  $\bigwedge x. x \in ?I n \implies (f \text{ has\_derivative } f' x) \text{ (at } x \text{ within } ?I n)$ 
      by ( $\text{meson Int\_iff deriv has\_derivative\_subset subsetI}$ )
    moreover have  $\text{int\_In}: (\lambda x. |\det (\text{matrix } (f' x))|) \text{ integrable\_on } ?I n$ 
      by ( $\text{metis } (\text{mono\_tags}) \text{ Int\_commute int integrable\_altD(1) integrable\_restrict\_Int}$ )
    ultimately have  $f' ?I n \in \text{lmeasurable } ?\mu (f' ?I n) \leq \text{integral } (?I n) (\lambda x. |\det (\text{matrix } (f' x))|)$ 
      using  $m\_diff\_image\_weak$  by  $\text{metis+}$ 
    moreover have  $\text{integral } (?I n) (\lambda x. |\det (\text{matrix } (f' x))|) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
      by ( $\text{simp add: int\_In int integral\_subset\_le}$ )
    ultimately show  $f' ?I n \in \text{lmeasurable } ?MN$ 
      by  $\text{auto}$ 
  qed
  have  $?I k \subseteq ?I n$  if  $k \leq n$  for  $k n$ 
    by ( $\text{rule Int\_mono}$ ) ( $\text{use that in } \langle \text{auto simp: subset\_interval\_imp\_cart} \rangle$ )
  then have  $(\bigcup k \leq n. f' ?I k) = f' ?I n$  for  $n$ 
    by ( $\text{fastforce simp add:}$ )
  with  $mfIn$  have  $?\mu (\bigcup k \leq n. f' ?I k) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|) \text{ for } n$ 
    by  $\text{simp}$ 
  then have  $(\bigcup n. f' ?I n) \in \text{lmeasurable } ?\mu (\bigcup n. f' ?I n) \leq \text{integral } S (\lambda x. |\det (\text{matrix } (f' x))|)$ 
    by ( $\text{rule fmeasurable\_countable\_Union [OF fIn] measure\_countable\_Union\_le}$ )

```

```

[OF fIn])+
  then show  $f' \cdot S \in lmeasurable \ ?M$ 
    by (metis Seq image_UN)+
qed

```

lemma borel_measurable_simple_function_limit_increasing:

```

fixes  $f :: 'a::euclidean\_space \Rightarrow real$ 
shows  $(f \in borel\_measurable\ lebesgue \wedge (\forall x. 0 \leq f\ x)) \longleftrightarrow$ 
 $(\exists g. (\forall n\ x. 0 \leq g\ n\ x \wedge g\ n\ x \leq f\ x) \wedge (\forall n\ x. g\ n\ x \leq (g(Suc\ n)\ x)) \wedge$ 
 $(\forall n. g\ n \in borel\_measurable\ lebesgue) \wedge (\forall n. finite(range\ (g\ n))) \wedge$ 
 $(\forall x. (\lambda n. g\ n\ x) \longrightarrow f\ x))$ 
(is ?lhs = ?rhs)

```

proof

```

assume  $f: ?lhs$ 
have  $leb\_f: \{x. a \leq f\ x \wedge f\ x < b\} \in sets\ lebesgue$  for  $a\ b$ 
proof -
  have  $\{x. a \leq f\ x \wedge f\ x < b\} = \{x. f\ x < b\} - \{x. f\ x < a\}$ 
    by auto
  also have  $\dots \in sets\ lebesgue$ 
    using borel_measurable_vimage_halfspace_component_lt [of  $f\ UNIV$ ]  $f$  by
auto

```

finally show $?thesis$.

qed

```

have  $g\ n\ x \leq f\ x$ 
  if  $inc\_g: \bigwedge n\ x. 0 \leq g\ n\ x \wedge g\ n\ x \leq g(Suc\ n)\ x$ 
    and  $meas\_g: \bigwedge n. g\ n \in borel\_measurable\ lebesgue$ 
    and  $fin: \bigwedge n. finite(range\ (g\ n))$  and  $lim: \bigwedge x. (\lambda n. g\ n\ x) \longrightarrow f\ x$  for

```

$g\ n\ x$

```

proof -
  have  $\exists r > 0. \forall N. \exists n \geq N. dist\ (g\ n\ x)\ (f\ x) \geq r$  if  $g\ n\ x > f\ x$ 
proof -
  have  $g: g\ n\ x \leq g(N + n)\ x$  for  $N$ 
    by (rule transitive_stepwise_le) (use  $inc\_g$  in auto)
  have  $\exists m \geq N. g\ m\ x - f\ x \leq dist\ (g\ m\ x)\ (f\ x)$  for  $N$ 
proof
  show  $N \leq N + n \wedge g\ n\ x - f\ x \leq dist\ (g(N + n)\ x)\ (f\ x)$ 
    using  $g$  [of  $N$ ] by (auto simp: dist_norm)

```

qed

```

with that show  $?thesis$ 
  using diff_gt_0_iff_gt by blast

```

qed

```

with  $lim$  show  $?thesis$ 
  unfolding  $lim\_sequentially$ 
  by (meson less_le_not_le not_le_imp_less)

```

qed

moreover

```

let  $?\Omega = \lambda n\ k. indicator\ \{y. k/2^{\wedge}n \leq f\ y \wedge f\ y < (k+1)/2^{\wedge}n\}$ 
let  $?g = \lambda n\ x. (\sum k::real \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). k/2^{\wedge}n * ?\Omega\ n\ k\ x)$ 

```

have $\exists g. (\forall n x. 0 \leq g n x \wedge g n x \leq (g(\text{Suc } n) x)) \wedge$
 $(\forall n. g n \in \text{borel_measurable_lebesgue}) \wedge (\forall n. \text{finite}(\text{range } (g n))) \wedge (\forall x.$
 $(\lambda n. g n x) \longrightarrow f x)$
proof (*intro exI allI conjI*)
show $0 \leq ?g n x$ **for** $n x$
proof (*clarify intro!: ordered_comm_monoid_add_class.sum_nonneg*)
fix $k::\text{real}$
assume $k \in \mathbf{Z}$ **and** $k: |k| \leq 2 \wedge (2 * n)$
show $0 \leq k/2 \wedge n * ?\Omega n k x$
using $f \langle k \in \mathbf{Z} \rangle$ **apply** (*clarsimp simp: indicator_def field_split_simps*
Ints_def)
by (*smt (verit) int_less_real_le mult_nonneg_nonneg of_int_0 zero_le_power*)
qed
show $?g n x \leq ?g (\text{Suc } n) x$ **for** $n x$
proof –
have $?g n x =$
 $(\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n).$
 $k/2 \wedge n * (\text{indicator } \{y. k/2 \wedge n \leq f y \wedge f y < (k+1)/2 \wedge n\} x +$
 $\text{indicator } \{y. (k+1)/2 \wedge n \leq f y \wedge f y < (k+1)/2 \wedge n\} x)$
by (*rule sum.cong [OF refl]*) (*simp add: indicator_def field_split_simps*)
also have $\dots = (\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n). k/2 \wedge n * \text{indicator } \{y. k/2 \wedge n$
 $\leq f y \wedge f y < (k+1)/2 \wedge n\} x) +$
 $(\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n). k/2 \wedge n * \text{indicator } \{y.$
 $(k+1)/2 \wedge n \leq f y \wedge f y < (k+1)/2 \wedge n\} x)$
by (*simp add: comm_monoid_add_class.sum.distrib algebra_simps*)
also have $\dots = (\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n). (2 * k)/2 \wedge \text{Suc } n * \text{indicator}$
 $\{y. (2 * k)/2 \wedge \text{Suc } n \leq f y \wedge f y < (2 * k+1)/2 \wedge \text{Suc } n\} x) +$
 $(\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n). (2 * k)/2 \wedge \text{Suc } n * \text{indicator}$
 $\{y. (2 * k+1)/2 \wedge \text{Suc } n \leq f y \wedge f y < ((2 * k+1) + 1)/2 \wedge \text{Suc } n\} x)$
by (*force simp: field_simps indicator_def intro: sum.cong*)
also have $\dots \leq (\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * \text{Suc } n). k/2 \wedge \text{Suc } n *$
 $(\text{indicator } \{y. k/2 \wedge \text{Suc } n \leq f y \wedge f y < (k+1)/2 \wedge \text{Suc } n\} x))$
 $(\text{is } ?a + _ \leq ?b)$
proof –
have $*$: $[\text{sum } f I \leq \text{sum } h I; a + \text{sum } h I \leq b] \implies a + \text{sum } f I \leq b$ **for** I
 $a b f$ **and** $h :: \text{real} \implies \text{real}$
by *linarith*
let $?h = \lambda k. (2 * k+1)/2 \wedge \text{Suc } n *$
 $(\text{indicator } \{y. (2 * k+1)/2 \wedge \text{Suc } n \leq f y \wedge f y < ((2 * k+1) +$
 $1)/2 \wedge \text{Suc } n\} x)$
show $?thesis$
proof (*rule **)
show $(\sum k \mid k \in \mathbf{Z} \wedge |k| \leq 2 \wedge (2 * n).$
 $2 * k/2 \wedge \text{Suc } n * \text{indicator } \{y. (2 * k+1)/2 \wedge \text{Suc } n \leq f y \wedge f y$
 $< (2 * k+1 + 1)/2 \wedge \text{Suc } n\} x)$
 $\leq \text{sum } ?h \{k \in \mathbf{Z}. |k| \leq 2 \wedge (2 * n)\}$
by (*rule sum_mono*) (*simp add: indicator_def field_split_simps*)
next
have $\alpha: ?a = (\sum k \in (*) 2 \wedge \{k \in \mathbf{Z}. |k| \leq 2 \wedge (2 * n)\}.$

```

      k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2
^ Suc n} x)
  by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
  have β: sum ?h {k ∈ ℤ. |k| ≤ 2 ^ (2*n)}
    = (∑ k ∈ (λx. 2*x + 1) ' {k ∈ ℤ. |k| ≤ 2 ^ (2*n)}).
      k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2
^ Suc n} x)
  by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
  have 0: (*) 2 ' {k ∈ ℤ. P k} ∩ (λx. 2 * x + 1) ' {k ∈ ℤ. P k} = {} for
P :: real ⇒ bool
  proof -
    have 2 * i ≠ 2 * j + 1 for i j :: int by arith
    thus ?thesis
      unfolding Ints_def by auto (use of_int_eq_iff in fastforce)
  qed
  have ?a + sum ?h {k ∈ ℤ. |k| ≤ 2 ^ (2*n)}
    = (∑ k ∈ (*) 2 ' {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ∪ (λx. 2*x + 1) ' {k ∈ ℤ.
|k| ≤ 2 ^ (2*n)}).
      k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2 ^
Suc n} x)
    unfolding α β
    using finite_abs_int_segment [of 2 ^ (2*n)]
    by (subst sum_Un) (auto simp: 0)
  also have ... ≤ ?b
  proof (rule sum_mono2)
    show finite {k::real. k ∈ ℤ ∧ |k| ≤ 2 ^ (2 * Suc n)}
      by (rule finite_abs_int_segment)
    show (*) 2 ' {k::real. k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n)} ∪ (λx. 2*x + 1) ' {k ∈
ℤ. |k| ≤ 2 ^ (2*n)} ⊆ {k ∈ ℤ. |k| ≤ 2 ^ (2 * Suc n)}
      apply (clarsimp simp: image_subset_iff)
      using one_le_power [of 2::real 2*n] by linarith
    have *: [x ∈ (S ∪ T) - U; ∧x. x ∈ S ⇒ x ∈ U; ∧x. x ∈ T ⇒ x ∈
U] ⇒ P x for S T U P
      by blast
    have 0 ≤ b if b ∈ ℤ f x * (2 * 2^n) < b + 1 for b
      by (smt (verit, ccfv_SIG) Ints_cases f int_le_real_less mult_nonneg_nonneg
of_int_add one_le_power that)
    then show 0 ≤ b/2 ^ Suc n * indicator {y. b/2 ^ Suc n ≤ f y ∧ f y <
(b + 1)/2 ^ Suc n} x
      if b ∈ {k ∈ ℤ. |k| ≤ 2 ^ (2 * Suc n)} -
      ((*) 2 ' {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ∪ (λx. 2*x + 1) ' {k ∈ ℤ.
|k| ≤ 2 ^ (2*n)}) for b
      using that by (simp add: indicator_def divide_simps)
  qed
  finally show ?a + sum ?h {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ≤ ?b .
  qed
  qed
  finally show ?thesis .
  qed

```

```

show ?g n ∈ borel_measurable lebesgue for n
apply (intro borel_measurable_indicator borel_measurable_times borel_measurable_sum)
  using leb_f_sets_restrict_UNIV by auto
show finite (range (?g n)) for n
proof -
  have (∑ k | k ∈ ℤ ∧ |k| ≤ 2^(2*n). k/2^n * ?Ω n k x)
    ∈ (λk. k/2^n) ' {k ∈ ℤ. |k| ≤ 2^(2*n)} for x
  proof (cases ∃ k. k ∈ ℤ ∧ |k| ≤ 2^(2*n) ∧ k/2^n ≤ f x ∧ f x < (k+1)/2^n)
    case True
    then show ?thesis
      by (blast intro: indicator_sum_eq)
    next
    case False
    then have (∑ k | k ∈ ℤ ∧ |k| ≤ 2^(2*n). k/2^n * ?Ω n k x) = 0
      by auto
    then show ?thesis by force
  qed
  then have range (?g n) ⊆ ((λk. (k/2^n)) ' {k. k ∈ ℤ ∧ |k| ≤ 2^(2*n)})
    by auto
  moreover have finite ((λk::real. (k/2^n)) ' {k ∈ ℤ. |k| ≤ 2^(2*n)})
    by (intro finite_imageI finite_abs_int_segment)
  ultimately show ?thesis
    by (rule finite_subset)
qed
show (λn. ?g n x) → f x for x
proof (clarsimp simp add: lim_sequentially)
  fix e::real
  assume e > 0
  obtain N1 where N1: 2^N1 > abs(f x)
    using real_arch_pow by fastforce
  obtain N2 where N2: (1/2)^N2 < e
    using real_arch_pow_inv ⟨e > 0⟩ by fastforce
  have dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2^(2*n). k/2^n * ?Ω n k x) (f x) < e if N1
+ N2 ≤ n for n
  proof -
    let ?m = real_of_int [2^n * f x]
    have |?m| ≤ 2^n * 2^N1
      using N1 apply (simp add: f)
    by (meson floor_mono le_floor_iff less_le_not_le mult_le_cancel_left_pos
zero_less_numeral zero_less_power)
    also have ... ≤ 2^(2*n)
      by (metis that add_leD1 add_le_cancel_left mult.commute mult_2_right
one_less_numeral_iff
power_add power_increasing_iff semiring_norm(76))
    finally have m_le: |?m| ≤ 2^(2*n) .
    have ?m/2^n ≤ f x f x < (?m + 1)/2^n
      by (auto simp: mult.commute pos_divide_le_eq mult_imp_less_div_pos)
    then have eq: dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2^(2*n). k/2^n * ?Ω n k x) (f x)
      = dist (?m/2^n) (f x)

```



```

    by (subst indicator_sum_eq [of ?m]) (auto simp: m_le)
  have  $|2^{\wedge}n| * |?m/2^{\wedge}n - f x| = |2^{\wedge}n * (?m/2^{\wedge}n - f x)|$ 
    by (simp add: abs_mult)
  also have  $\dots < 2^{\wedge}N2 * e$ 
    using N2 by (simp add: divide_simps mult.commute) linarith
  also have  $\dots \leq |2^{\wedge}n| * e$ 
    using that <e > 0 by auto
  finally show ?thesis
    using eq by (simp add: dist_real_def)
qed
then show  $\exists no. \forall n \geq no. dist (\sum k \mid k \in \mathbb{Z} \wedge |k| \leq 2^{\wedge}(2*n). k * ?\Omega n k$ 
 $x/2^{\wedge}n) (f x) < e$ 
  by force
qed
qed
ultimately show ?rhs
  by metis
next
assume RHS: ?rhs
with borel_measurable_simple_function_limit [of f UNIV, unfolded lebesgue_on_UNIV_eq]
show ?lhs
  by (blast intro: order_trans)
qed

```

9.32.2 Borel measurable Jacobian determinant

lemma lemma_partial_derivatives0:

```

  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes linear f and lim0:  $((\lambda x. f x /_R norm x) \longrightarrow 0)$  (at 0 within S)
  and lb:  $\bigwedge v. v \neq 0 \implies (\exists k > 0. \forall e > 0. \exists x. x \in S - \{0\} \wedge norm x < e \wedge k * norm x \leq |v \cdot x|)$ 
  shows  $f x = 0$ 

```

proof -

```

  interpret linear f by fact
  have dim {x. f x = 0}  $\leq DIM('a)$ 
    by (rule dim_subset_UNIV)
  moreover have False if less:  $dim \{x. f x = 0\} < DIM('a)$ 
  proof -
    obtain d where  $d \neq 0$  and  $d: \bigwedge y. f y = 0 \implies d \cdot y = 0$ 
      using orthogonal_to_subspace_exists [OF less] orthogonal_def
      by (metis (mono_tags, lifting) mem_Collect_eq span_base)
    then obtain k where  $k > 0$ 
      and  $k: \bigwedge e. e > 0 \implies \exists y. y \in S - \{0\} \wedge norm y < e \wedge k * norm y \leq |d \cdot y|$ 
    using lb by blast
    have  $\exists h. \forall n. ((h n \in S \wedge h n \neq 0 \wedge k * norm (h n) \leq |d \cdot h n|) \wedge norm (h n) < 1 / real (Suc n)) \wedge norm (h (Suc n)) < norm (h n)$ 
      proof (rule dependent_nat_choice)

```

```

show  $\exists y. (y \in S \wedge y \neq 0 \wedge k * \text{norm } y \leq |d \cdot y|) \wedge \text{norm } y < 1 / \text{real } (\text{Suc } 0)$ 
  by simp (metis DiffE insertCI k not_less not_one_le_zero)
qed (use k [of min (norm x) (1/(Suc n + 1))] for x n] in auto)
then obtain  $\alpha$  where  $\alpha: \bigwedge n. \alpha n \in S - \{0\}$  and  $kd: \bigwedge n. k * \text{norm}(\alpha n) \leq |d \cdot \alpha n|$ 
  and norm_lt:  $\bigwedge n. \text{norm}(\alpha n) < 1/(\text{Suc } n)$ 
  by force
let  $?\beta = \lambda n. \alpha n /_R \text{norm } (\alpha n)$ 
have com:  $\bigwedge g. (\forall n. g n \in \text{sphere } (0::'a) 1) \implies \exists l \in \text{sphere } 0 1. \exists \varrho::\text{nat} \Rightarrow \text{nat. } \text{strict\_mono } \varrho \wedge (g \circ \varrho) \longrightarrow l$ 
  using compact_sphere compact_def by metis
moreover have  $\forall n. ?\beta n \in \text{sphere } 0 1$ 
  using  $\alpha$  by auto
ultimately obtain l::'a and  $\varrho::\text{nat} \Rightarrow \text{nat}$ 
  where  $l: l \in \text{sphere } 0 1$  and strict_mono  $\varrho$  and to_l:  $(?\beta \circ \varrho) \longrightarrow l$ 
  by meson
moreover have continuous (at l)  $(\lambda x. (|d \cdot x| - k))$ 
  by (intro continuous_intros)
ultimately have lim_dl:  $((\lambda x. (|d \cdot x| - k)) \circ (?\beta \circ \varrho)) \longrightarrow (|d \cdot l| - k)$ 
  by (meson continuous_imp_tendsto)
have  $\forall_F i$  in sequentially.  $0 \leq ((\lambda x. |d \cdot x| - k) \circ ((\lambda n. \alpha n /_R \text{norm } (\alpha n)) \circ \varrho)) i$ 
  using  $\alpha$  kd by (auto simp: field_split_simps)
then have  $k \leq |d \cdot l|$ 
  using tendsto_lowerbound [OF lim_dl, of 0] by auto
moreover have  $d \cdot l = 0$ 
proof (rule d)
  show  $f l = 0$ 
  proof (rule LIMSEQ_unique [of  $f \circ ?\beta \circ \varrho$ ])
    have isCont  $f l$ 
      using  $\langle \text{linear } f \rangle$  linear_continuous_at linear_conv_bounded_linear by
blast
    then show  $(f \circ (\lambda n. \alpha n /_R \text{norm } (\alpha n)) \circ \varrho) \longrightarrow f l$ 
      unfolding comp_assoc
      using to_l continuous_imp_tendsto by blast
    have  $\alpha \longrightarrow 0$ 
      using norm_lt LIMSEQ_norm_0 by metis
    with  $\langle \text{strict\_mono } \varrho \rangle$  have  $(\alpha \circ \varrho) \longrightarrow 0$ 
      by (metis LIMSEQ_subseq_LIMSEQ)
    with lim0  $\alpha$  have  $((\lambda x. f x /_R \text{norm } x) \circ (\alpha \circ \varrho)) \longrightarrow 0$ 
      by (force simp: tendsto_at_iff_sequentially)
    then show  $(f \circ (\lambda n. \alpha n /_R \text{norm } (\alpha n)) \circ \varrho) \longrightarrow 0$ 
      by (simp add: o_def scale)
  qed
qed
ultimately show False
  using  $\langle k > 0 \rangle$  by auto
qed

```

ultimately have $\dim: \dim \{x. f x = 0\} = DIM('a)$
 by force
 then show ?thesis
 by (metis (mono_tags, lifting) dim_eq_full UNIV_I eq_0_on_span mem_Collect_eq span_raw_def)
 qed

lemma lemma_partial_derivatives:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$
 assumes linear f and $\lim: ((\lambda x. f (x - a) /_R \text{norm } (x - a)) \longrightarrow 0)$ (at a within S)
 and $lb: \bigwedge v. v \neq 0 \implies (\exists k > 0. \forall e > 0. \exists x \in S - \{a\}. \text{norm}(a - x) < e \wedge k * \text{norm}(a - x) \leq |v \cdot (x - a)|)$
 shows $f x = 0$
 proof -
 have $((\lambda x. f x /_R \text{norm } x) \longrightarrow 0)$ (at 0 within $(\lambda x. x - a) ' S$)
 using \lim by (simp add: Lim_within_dist_norm)
 then show ?thesis
 proof (rule lemma_partial_derivatives0 [OF <linear f>])
 fix $v :: 'a$
 assume $v: v \neq 0$
 show $\exists k > 0. \forall e > 0. \exists x. x \in (\lambda x. x - a) ' S - \{0\} \wedge \text{norm } x < e \wedge k * \text{norm } x \leq |v \cdot x|$
 using lb [OF v] by (force simp: norm_minus_commute)
 qed
 qed

proposition borel_measurable_partial_derivatives:

fixes $f :: \text{real}^m::\{\text{finite,wellorder}\} \Rightarrow \text{real}^n$
 assumes $S: S \in \text{sets lebesgue}$
 and $f: \bigwedge x. x \in S \implies (f \text{ has_derivative } f' x)$ (at x within S)
 shows $(\lambda x. (\text{matrix}(f' x) \$m \$n)) \in \text{borel_measurable } (\text{lebesgue_on } S)$
 proof -
 have $\text{contf}: \text{continuous_on } S f$
 using $\text{continuous_on_eq_continuous_within } f \text{ has_derivative_continuous}$ by blast
 have $\{x \in S. (\text{matrix}(f' x) \$m \$n) \leq b\} \in \text{sets lebesgue}$ for b
 proof (rule sets_negligible_syndiff)
 let $?T = \{x \in S. \forall e > 0. \exists d > 0. \exists A. A \$m \$n < b \wedge (\forall i j. A \$i \$j \in \mathbf{Q}) \wedge (\forall y \in S. \text{norm}(y - x) < d \longrightarrow \text{norm}(f y - f x - A * v (y - x)) \leq e * \text{norm}(y - x))\}$
 let $?U = S \cap$
 $(\bigcap e \in \{e \in \mathbf{Q}. e > 0\}.$
 $\bigcup A \in \{A. A \$m \$n < b \wedge (\forall i j. A \$i \$j \in \mathbf{Q})\}.$
 $\bigcup d \in \{d \in \mathbf{Q}. 0 < d\}.$
 $S \cap (\bigcap y \in S. \{x \in S. \text{norm}(y - x) < d \longrightarrow \text{norm}(f y - f x - A * v (y - x)) \leq e * \text{norm}(y - x)\}))$
 have $?T = ?U$

```

proof (intro set_eqI iffI)
  fix x
  assume xT: x ∈ ?T
  then show x ∈ ?U
  proof (clarsimp simp add:)
    fix q :: real
    assume q ∈ ℚ q > 0
    then obtain d A where d > 0 and A: A $ m $ n < b ∧ i j. A $ i $ j ∈ ℚ
      ∧ y. [y ∈ S; norm (y - x) < d] ⇒ norm (f y - f x - A *v (y - x)) ≤ q
    * norm (y - x)
    using xT by auto
    then obtain δ where d > δ δ > 0 δ ∈ ℚ
    using Rats_dense_in_real by blast
    with A show ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ) ∧
      (∃ s. s ∈ ℚ ∧ 0 < s ∧ (∀ y ∈ S. norm (y - x) < s → norm (f
y - f x - A *v (y - x)) ≤ q * norm (y - x)))
    by force
  qed
next
  fix x
  assume xU: x ∈ ?U
  then show x ∈ ?T
  proof clarsimp
    fix e :: real
    assume e > 0
    then obtain ε where e > ε ε > 0 ε ∈ ℚ
    using Rats_dense_in_real by blast
    with xU obtain A r where x ∈ S and Ar: A $ m $ n < b ∀ i j. A $ i $ j
∈ ℚ r ∈ ℚ r > 0
    and ∀ y ∈ S. norm (y - x) < r → norm (f y - f x - A *v (y - x)) ≤ ε
    * norm (y - x)
    by (auto simp: split: if_split_asm)
    then have ∀ y ∈ S. norm (y - x) < r → norm (f y - f x - A *v (y - x))
≤ e * norm (y - x)
    by (meson ‹e > ε› less_eq_real_def mult_right_mono norm_ge_zero
order_trans)
    then show ∃ d > 0. ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ) ∧ (∀ y ∈ S.
norm (y - x) < d → norm (f y - f x - A *v (y - x)) ≤ e * norm (y - x))
    using ‹x ∈ S› Ar by blast
  qed
qed
moreover have ?U ∈ sets lebesgue
proof -
  have coQ: countable {e ∈ ℚ. 0 < e}
    using countable_Collect countable_rat by blast
  have ne: {e ∈ ℚ. (0::real) < e} ≠ {}
    using zero_less_one Rats_1 by blast
  have coA: countable {A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ)}
  proof (rule countable_subset)

```

```

    show countable {A.  $\forall i j. A \ \$ \ i \ \$ \ j \in \mathbf{Q}$ }
      using countable_vector [OF countable_vector, of  $\lambda i j. \mathbf{Q}$ ] by (simp add:
countable_rat)
    qed blast
    have *:  $\llbracket U \neq \{\} \implies \text{closedin } (\text{top\_of\_set } S) (S \cap \bigcap U) \rrbracket$ 
       $\implies \text{closedin } (\text{top\_of\_set } S) (S \cap \bigcap U)$  for U
    by fastforce
    have eq:  $\{x::(\text{real},'m)\text{vec}. P x \wedge (Q x \longrightarrow R x)\} = \{x. P x \wedge \neg Q x\} \cup \{x.$ 
P x \wedge R x\} for P Q R
    by auto
    have sets:  $S \cap (\bigcap y \in S. \{x \in S. \text{norm } (y - x) < d \longrightarrow \text{norm } (f y - f x - A$ 
*v (y - x)\}  $\leq e * \text{norm } (y - x)\}$ 
       $\in \text{sets lebesgue}$  for e A d
    proof -
      have clo:  $\text{closedin } (\text{top\_of\_set } S)$ 
         $\{x \in S. \text{norm } (y - x) < d \longrightarrow \text{norm } (f y - f x - A * v (y - x))$ 
 $\leq e * \text{norm } (y - x)\}$ 
        for y
      proof -
        have cont1:  $\text{continuous\_on } S (\lambda x. \text{norm } (y - x))$ 
          and cont2:  $\text{continuous\_on } S (\lambda x. e * \text{norm } (y - x) - \text{norm } (f y - f x$ 
- (A * v y - A * v x))
          by (force intro: contf continuous_intros)+
        have clo1:  $\text{closedin } (\text{top\_of\_set } S) \{x \in S. d \leq \text{norm } (y - x)\}$ 
          using continuous_closedin_preimage [OF cont1, of {d..}] by (simp add:
vimage_def Int_def)
        have clo2:  $\text{closedin } (\text{top\_of\_set } S)$ 
         $\{x \in S. \text{norm } (f y - f x - (A * v y - A * v x)) \leq e * \text{norm } (y$ 
- x)\}
          using continuous_closedin_preimage [OF cont2, of {0..}] by (simp add:
vimage_def Int_def)
        show ?thesis
          by (auto simp: eq not_less matrix_vector_mult_diff_distrib intro: clo1
clo2)
      qed
    show ?thesis
      by (rule lebesgue_closedin [of S]) (force intro: * S clo)+
    qed
  show ?thesis
    by (intro sets.Int S sets.countable_UN'' sets.countable_INT'' coQ coA)
  auto
  qed
  ultimately show ?T  $\in \text{sets lebesgue}$ 
    by simp
  let ?M =  $(?T - \{x \in S. \text{matrix } (f' x) \ \$ \ m \ \$ \ n \leq b\}) \cup (\{x \in S. \text{matrix } (f' x)$ 
$ m $ n  $\leq b\} - ?T)$ 
  let ? $\Theta = \lambda x v. \forall \xi > 0. \exists e > 0. \forall y \in S - \{x\}. \text{norm } (x - y) < e \longrightarrow |v \cdot (y -$ 
x)| <  $\xi * \text{norm } (x - y)$ 
  have nN:  $\text{negligible } \{x \in S. \exists v \neq 0. ?\Theta x v\}$ 

```

```

unfolding negligible_eq_zero_density
proof clarsimp
fix  $x v$  and  $r e :: \text{real}$ 
assume  $x \in S \ v \neq 0 \ r > 0 \ e > 0$ 
and  $\Theta$  [rule_format]:  $? \Theta \ x \ v$ 
moreover have  $(\text{norm } v * e / 2) / \text{CARD}'(m) \wedge \text{CARD}'(m) > 0$ 
by (simp add:  $\langle v \neq 0 \rangle \langle e > 0 \rangle$ )
ultimately obtain  $d$  where  $d > 0$ 
and  $d$  less:  $\bigwedge y. \llbracket y \in S - \{x\}; \text{norm } (x - y) < d \rrbracket \implies$ 
 $|v \cdot (y - x)| < ((\text{norm } v * e / 2) / \text{CARD}'(m) \wedge \text{CARD}'(m))$ 
*  $\text{norm } (x - y)$ 
by metis
let  $?W = \text{ball } x \ (\text{min } d \ r) \cap \{y. |v \cdot (y - x)| < (\text{norm } v * e / 2 * \text{min } d \ r) /$ 
 $\text{CARD}'(m) \wedge \text{CARD}'(m)\}$ 
have open  $\{x. |v \cdot (x - a)| < b\}$  for  $a \ b$ 
by (intro open_Collect_less continuous_intros)
show  $\exists d > 0. d \leq r \wedge$ 
 $(\exists U. \{x' \in S. \exists v \neq 0. ? \Theta \ x' \ v\} \cap \text{ball } x \ d \subseteq U \wedge$ 
 $U \in \text{lmeasurable} \wedge \text{measure lebesgue } U < e * \text{content } (\text{ball } x \ d))$ 
proof (intro exI conjI)
show  $0 < \text{min } d \ r \ \text{min } d \ r \leq r$ 
using  $\langle r > 0 \rangle \langle d > 0 \rangle$  by auto
show  $\{x' \in S. \exists v. v \neq 0 \wedge (\forall \xi > 0. \exists e > 0. \forall z \in S - \{x'\}. \text{norm } (x' - z)$ 
 $< e \longrightarrow |v \cdot (z - x')| < \xi * \text{norm } (x' - z))\} \cap \text{ball } x \ (\text{min } d \ r) \subseteq ?W$ 
proof (clarsimp simp: dist_norm norm_minus_commute)
fix  $y \ w$ 
assume  $y \in S \ w \neq 0$ 
and less [rule_format]:
 $\forall \xi > 0. \exists e > 0. \forall z \in S - \{y\}. \text{norm } (y - z) < e \longrightarrow |w \cdot (z - y)|$ 
 $< \xi * \text{norm } (y - z)$ 
and  $d$ :  $\text{norm } (y - x) < d$  and  $r$ :  $\text{norm } (y - x) < r$ 
show  $|v \cdot (y - x)| < \text{norm } v * e * \text{min } d \ r / (2 * \text{real } \text{CARD}'(m) \wedge$ 
 $\text{CARD}'(m))$ 
proof (cases  $y = x$ )
case True
with  $\langle r > 0 \rangle \langle d > 0 \rangle \langle e > 0 \rangle \langle v \neq 0 \rangle$  show ?thesis
by simp
next
case False
have  $|v \cdot (y - x)| < \text{norm } v * e / 2 / \text{real } (\text{CARD}'(m) \wedge \text{CARD}'(m))$ 
*  $\text{norm } (x - y)$ 
by (metis Diff_iff False  $\langle y \in S \rangle \ d$  less empty_iff insert_iff
norm_minus_commute)
also have  $\dots \leq \text{norm } v * e * \text{min } d \ r / (2 * \text{real } \text{CARD}'(m) \wedge$ 
 $\text{CARD}'(m))$ 
using  $d \ r \ \langle e > 0 \rangle$  by (simp add: field_simps norm_minus_commute
mult_left_mono)
finally show ?thesis .
qed

```

```

qed
show ?W ∈ lmeasurable
  by (simp add: fmeasurable_Int_fmeasurable borel_open)
obtain k::'m where True
  by metis
obtain T where T: orthogonal_transformation T and v: v = T(norm v
*_R axis k (1::real))
  using rotation_rightward_line by metis
define b where b ≡ norm v
have b > 0
  using ⟨v ≠ 0⟩ by (auto simp: b_def)
obtain eqb: inv T v = b *_R axis k (1::real) and inj T bij T and invT:
orthogonal_transformation (inv T)
  by (metis UNIV_I b_def T v bij_betw_inv_into_left orthogo-
nal_transformation_inj orthogonal_transformation_bij orthogonal_transformation_inv)
let ?v = χ i. min d r / CARD('m)
let ?v' = χ i. if i = k then (e/2 * min d r) / CARD('m) ^ CARD('m)
else min d r
let ?x' = inv T x
let ?W' = (ball ?x' (min d r) ∩ {y. |(y - ?x')$k| < e * min d r / (2 *
CARD('m) ^ CARD('m))})
have abs: x - e ≤ y ∧ y ≤ x + e ⟷ abs(y - x) ≤ e for x y e::real
  by auto
have ?W = T ' ?W'
proof -
have 1: T ' (ball (inv T x) (min d r)) = ball x (min d r)
  by (simp add: T_image_orthogonal_transformation_ball orthogo-
nal_transformation_surj_surj_f_inv_f)
have 2: {y. |v · (y - x)| < b * e * min d r / (2 * real CARD('m) ^
CARD('m))} =
  T ' {y. |y $ k - ?x' $ k| < e * min d r / (2 * real CARD('m) ^
CARD('m))}
proof -
have *: |T (b *_R axis k 1) · (y - x)| = b * |inv T y $ k - ?x' $ k| for
y
proof -
have |T (b *_R axis k 1) · (y - x)| = |(b *_R axis k 1) · inv T (y - x)|
  by (metis (no_types, opaque_lifting) b_def eqb invT orthogo-
nal_transformation_def v)
also have ... = b * |(axis k 1) · inv T (y - x)|
  using ⟨b > 0⟩ by (simp add: abs_mult)
also have ... = b * |inv T y $ k - ?x' $ k|
  using orthogonal_transformation_linear [OF invT]
  by (simp add: inner_axis' linear_diff)
finally show ?thesis
  by simp
qed
show ?thesis
  using v b_def [symmetric]

```

```

      using ⟨b > 0⟩ by (simp add: * bij_image_Collect_eq [OF ⟨bij T⟩]
mult_less_cancel_left_pos times_divide_eq_right [symmetric] del: times_divide_eq_right)
    qed
    show ?thesis
      using ⟨b > 0⟩ by (simp add: image_Int ⟨inj T⟩ 1 2 b_def [symmetric])
    qed
    moreover have ?W' ∈ lmeasurable
      by (auto intro: fmeasurable_Int_fmeasurable)
    ultimately have measure lebesgue ?W = measure lebesgue ?W'
      by (metis measure_orthogonal_image T)
    also have ... ≤ measure lebesgue (cbox (?x' - ?v') (?x' + ?v'))
    proof (rule measure_mono_fmeasurable)
      show ?W' ⊆ cbox (?x' - ?v') (?x' + ?v')
    apply (clarsimp simp add: mem_box_cart abs_dist_norm norm_minus_commute
simp del: min_less_iff_conj min.bounded_iff)
      by (metis component_le_norm_cart less_eq_real_def le_less_trans
vector_minus_component)
    qed auto
    also have ... ≤ e/2 * measure lebesgue (cbox (?x' - ?v) (?x' + ?v))
    proof -
      have cbox (?x' - ?v) (?x' + ?v) ≠ {}
        using ⟨r > 0⟩ ⟨d > 0⟩ by (auto simp: interval_eq_empty_cart
divide_less_0_iff)
      with ⟨r > 0⟩ ⟨d > 0⟩ ⟨e > 0⟩ show ?thesis
        apply (simp add: content_cbox_if_cart mem_box_cart)
        apply (auto simp: prod_nonneg)
        apply (simp add: abs_if_distrib prod.delta_remove field_simps power_diff
split: if_split_asm)
      done
    qed
    also have ... ≤ e/2 * measure lebesgue (cball ?x' (min d r))
    proof (rule mult_left_mono [OF measure_mono_fmeasurable])
      have *: norm (?x' - y) ≤ min d r
        if y:  $\bigwedge i. |?x' \$ i - y \$ i| \leq \min d r / \text{real CARD}('m)$  for y
      proof -
        have norm (?x' - y) ≤ ( $\sum i \in \text{UNIV}. |(?x' - y) \$ i|$ )
          by (rule norm_le_l1_cart)
        also have ... ≤ real CARD('m) * (min d r / real CARD('m))
          by (rule sum_bounded_above) (use y in auto)
        finally show ?thesis
          by simp
      qed
    qed
    show cbox (?x' - ?v) (?x' + ?v) ⊆ cball ?x' (min d r)
      apply (clarsimp simp only: mem_box_cart dist_norm mem_cball
intro!: *)
      by (simp add: abs_diff_le_iff abs_minus_commute)
    qed (use ⟨e > 0⟩ in auto)
    also have ... < e * content (cball ?x' (min d r))
      using ⟨r > 0⟩ ⟨d > 0⟩ ⟨e > 0⟩ by (auto intro: content_cball_pos)

```



```

also have ... = e * content (ball x (min d r))
using ‹r > 0› ‹d > 0› content_ball_conv_unit_ball[of min d r inv T x]
      content_ball_conv_unit_ball[of min d r x]
by (simp add: content_cball_conv_ball)
finally show measure lebesgue ?W < e * content (ball x (min d r)) .
qed
qed
have *: (∧x. (x ∉ S) ⇒ (x ∈ T ↔ x ∈ U)) ⇒ (T - U) ∪ (U - T) ⊆ S
for S T U :: (real, 'm) vec set
by blast
have MN: ?M ⊆ {x ∈ S. ∃v≠0. ?Θ x v}
proof (rule *)
fix x
assume x: x ∉ {x ∈ S. ∃v≠0. ?Θ x v}
show (x ∈ ?T) ↔ (x ∈ {x ∈ S. matrix (f' x) $ m $ n ≤ b})
proof (cases x ∈ S)
case True
then have x: ¬ ?Θ x v if v ≠ 0 for v
      using x that by force
show ?thesis
proof (rule iffI; clarsimp)
assume b: ∀e>0. ∃d>0. ∃A. A $ m $ n < b ∧ (∀i j. A $ i $ j ∈ ℚ) ∧
      (∀y∈S. norm (y - x) < d → norm (f y - f x - A
*v (y - x)) ≤ e * norm (y - x))
      (is ∀e>0. ∃d>0. ∃A. ?Φ e d A)
then have ∀k. ∃d>0. ∃A. ?Φ (1 / Suc k) d A
by (metis (no_types, opaque_lifting) less_Suc_eq_0_disj_of_nat_0_less_iff
zero_less_divide_1_iff)
then obtain δ A where δ: ∧k. δ k > 0
      and Ab: ∧k. A k $ m $ n < b
      and A: ∧k y. [y ∈ S; norm (y - x) < δ k] ⇒
      norm (f y - f x - A k *v (y - x)) ≤ 1/(Suc k)
* norm (y - x)
by metis
have ∀i j. ∃a. (λn. A n $ i $ j) → a
proof (intro allI)
fix i j
have var: (A n *v axis j 1) $ i = A n $ i $ j for n
      by (metis cart_eq_inner_axis_matrix_vector_mul_component)
let ?CA = {x. Cauchy (λn. (A n) *v x)}
have subspace ?CA
      unfolding subspace_def convergent_eq_Cauchy [symmetric]
      by (force simp: algebra_simps intro: tendsto_intros)
then have CA_eq: ?CA = span ?CA
      by (metis span_eq_iff)
also have ... = UNIV
proof -
have dim ?CA ≤ CARD('m)
      using dim_subset_UNIV[of ?CA] by auto

```

moreover have *False* if less: $\dim ?CA < \text{CARD}('m)$
proof –
obtain d where $d \neq 0$ and $d: \bigwedge y. y \in \text{span } ?CA \implies \text{orthogonal } d \ y$
 using less by (force intro: *orthogonal_to_subspace_exists* [of ?CA])
with x [OF $\langle d \neq 0 \rangle$] **obtain** ξ where $\xi > 0$
and $\xi: \bigwedge e. e > 0 \implies \exists y \in S - \{x\}. \text{norm } (x - y) < e \wedge \xi * \text{norm}$
 $(x - y) \leq |d \cdot (y - x)|$
by (*fastforce simp: not_le Bex_def*)
obtain $\gamma \ z$ where $\gamma Sx: \bigwedge i. \gamma \ i \in S - \{x\}$
and $\gamma le: \bigwedge i. \xi * \text{norm}(\gamma \ i - x) \leq |d \cdot (\gamma \ i - x)|$
and $\gamma x: \gamma \longrightarrow x$
and $z: (\lambda n. (\gamma \ n - x) /_R \text{norm } (\gamma \ n - x)) \longrightarrow z$
proof –
have $\exists \gamma. (\forall i. (\gamma \ i \in S - \{x\} \wedge$
 $\xi * \text{norm}(\gamma \ i - x) \leq |d \cdot (\gamma \ i - x)| \wedge \text{norm}(\gamma \ i - x)$
 $< 1 / \text{Suc } i) \wedge$
 $\text{norm}(\gamma (\text{Suc } i) - x) < \text{norm}(\gamma \ i - x))$
proof (*rule dependent_nat_choice*)
show $\exists y. y \in S - \{x\} \wedge \xi * \text{norm } (y - x) \leq |d \cdot (y - x)| \wedge$
 $\text{norm } (y - x) < 1 / \text{Suc } 0$
using ξ [of 1] **by** (*auto simp: dist_norm norm_minus_commute*)
next
fix $y \ i$
assume $y \in S - \{x\} \wedge \xi * \text{norm } (y - x) \leq |d \cdot (y - x)| \wedge \text{norm}$
 $(y - x) < 1 / \text{Suc } i$
then have $\min (\text{norm}(y - x)) (1 / ((\text{Suc } i) + 1)) > 0$
by *auto*
then obtain y' where $y' \in S - \{x\}$ and $y': \text{norm } (x - y') <$
 $\min (\text{norm } (y - x)) (1 / ((\text{Suc } i) + 1))$
 $\xi * \text{norm } (x - y') \leq |d \cdot (y' - x)|$
using ξ by *metis*
with ξ **show** $\exists y'. (y' \in S - \{x\} \wedge \xi * \text{norm } (y' - x) \leq |d \cdot (y'$
 $- x)| \wedge$
 $\text{norm } (y' - x) < 1 / (\text{Suc } (\text{Suc } i))) \wedge \text{norm } (y' - x) <$
 $\text{norm } (y - x)$
by (*auto simp: dist_norm norm_minus_commute*)
qed
then obtain γ where
 $\gamma Sx: \bigwedge i. \gamma \ i \in S - \{x\}$
and $\gamma le: \bigwedge i. \xi * \text{norm}(\gamma \ i - x) \leq |d \cdot (\gamma \ i - x)|$
and $\gamma conv: \bigwedge i. \text{norm}(\gamma \ i - x) < 1 / (\text{Suc } i)$
by *blast*
let $?f = \lambda i. (\gamma \ i - x) /_R \text{norm } (\gamma \ i - x)$
have $?f \ i \in \text{sphere } 0 \ 1$ **for** i
using γSx **by** *auto*
then obtain $l \ \varrho$ where $l \in \text{sphere } 0 \ 1$ *strict_mono* ϱ and $l: (?f \circ$
 $\varrho) \longrightarrow l$
using *compact_sphere* [of 0::(*real*, $'m$) *vec* 1] **unfolding** *compact_def*
by *meson*

```

show thesis
proof
  show  $(\gamma \circ \varrho) i \in S - \{x\} \ \xi * \text{norm } ((\gamma \circ \varrho) i - x) \leq |d \cdot ((\gamma \circ \varrho) i - x)|$  for  $i$ 
    using  $\gamma Sx \ \gamma le$  by auto
    have  $\gamma \longrightarrow x$ 
    proof (clarsimp simp add: LIMSEQ_def dist_norm)
      fix  $r :: \text{real}$ 
      assume  $r > 0$ 
      with  $\text{real\_arch\_invD}$  obtain  $no$  where  $no \neq 0$   $\text{real } no > 1/r$ 
        by (metis divide_less_0_1_iff_not_less_iff_gr_or_eq of_nat_0_eq_iff reals_Archimedean2)
      with  $\gamma \text{conv}$  show  $\exists no. \forall n \geq no. \text{norm } (\gamma n - x) < r$ 
      by (metis ‹ $r > 0$ › add.commute divide_inverse inverse_inverse_eq inverse_less_imp_less_trans mult.left_neutral nat_le_real_less_of_nat_Suc)
    qed
    with ‹strict_mono  $\varrho$ › show  $(\gamma \circ \varrho) \longrightarrow x$ 
    by (metis LIMSEQ_subseq LIMSEQ)
    show  $(\lambda n. ((\gamma \circ \varrho) n - x) /_R \text{norm } ((\gamma \circ \varrho) n - x)) \longrightarrow l$ 
    using  $l$  by (auto simp: o_def)
  qed
  have  $\text{isCont } (\lambda x. (|d \cdot x| - \xi)) z$ 
    by (intro continuous_intros)
  from  $\text{isCont\_tendsto\_compose}$  [OF this  $z$ ]
  have  $\text{lim: } (\lambda y. |d \cdot ((\gamma y - x) /_R \text{norm } (\gamma y - x))| - \xi) \longrightarrow |d \cdot z| - \xi$ 
    by auto
  moreover have  $\forall_F i$  in sequentially.  $0 \leq |d \cdot ((\gamma i - x) /_R \text{norm } (\gamma i - x))| - \xi$ 
  proof (rule eventuallyI)
    fix  $n$ 
    show  $0 \leq |d \cdot ((\gamma n - x) /_R \text{norm } (\gamma n - x))| - \xi$ 
      using  $\gamma le$  [of  $n$ ]  $\gamma Sx$  by (auto simp: abs_mult divide_simps)
  qed
  ultimately have  $\xi \leq |d \cdot z|$ 
    using  $\text{tendsto\_lowerbound}$  [where  $a=0$ ] by fastforce
  have  $\text{Cauchy } (\lambda n. (A n) *v z)$ 
  proof (clarsimp simp add: Cauchy_def)
    fix  $\varepsilon :: \text{real}$ 
    assume  $0 < \varepsilon$ 
    then obtain  $N :: \text{nat}$  where  $N > 0$  and  $N: \varepsilon/2 > 1/N$ 
  by (metis half_gt_zero inverse_eq_divide neq0_conv real_arch_inverse)
  show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (A m *v z) (A n *v z) < \varepsilon$ 
  proof (intro exI allI impI)
    fix  $i j$ 
    assume  $ij: N \leq i N \leq j$ 
    let  $?V = \lambda i k. A i *v ((\gamma k - x) /_R \text{norm } (\gamma k - x))$ 
    have  $\forall_F k$  in sequentially.  $\text{dist } (\gamma k) x < \min (\delta i) (\delta j)$ 

```

```

using  $\gamma x$  [unfolded tendsto_iff] by (meson min_less_iff_conj  $\delta$ )
then have even:  $\forall_F k$  in sequentially. norm ( $?V i k - ?V j k$ ) -
2 / N  $\leq 0$ 

proof (rule eventually_mono, clarsimp)
fix  $p$ 
assume  $p$ : dist ( $\gamma p$ )  $x < \delta$  i dist ( $\gamma p$ )  $x < \delta$   $j$ 
let  $?C = \lambda k. f (\gamma p) - f x - A k * v (\gamma p - x)$ 
have norm (( $A i - A j$ ) *  $v (\gamma p - x)$ ) = norm ( $?C j - ?C i$ )
by (simp add: algebra_simps)
also have ...  $\leq$  norm ( $?C j$ ) + norm ( $?C i$ )
using norm_triangle_ineq4 by blast
also have ...  $\leq 1 / (\text{Suc } j) * \text{norm } (\gamma p - x) + 1 / (\text{Suc } i) *
\text{norm } (\gamma p - x)$ 
by (metis A Diff_iff  $\gamma Sx$  dist_norm p add_mono)
also have ...  $\leq 1 / N * \text{norm } (\gamma p - x) + 1 / N * \text{norm } (\gamma p -
x)$ 
using  $ij \langle N > 0 \rangle$  by (intro add_mono mult_right_mono)
(auto simp: field_simps)
also have ... = 2 / N * norm ( $\gamma p - x$ )
by simp
finally have no_le: norm (( $A i - A j$ ) *  $v (\gamma p - x)$ )  $\leq 2 / N
* \text{norm } (\gamma p - x)$ .
have norm ( $?V i p - ?V j p$ ) =
norm (( $A i - A j$ ) *  $v ((\gamma p - x) /_R \text{norm } (\gamma p - x))$ )
by (simp add: algebra_simps)
also have ... = norm (( $A i - A j$ ) *  $v (\gamma p - x)$ ) / norm ( $\gamma p
- x$ )
by (simp add: divide_inverse matrix_vector_mult_scaleR)
also have ...  $\leq 2 / N$ 
using no_le by (auto simp: field_split_simps)
finally show norm ( $?V i p - ?V j p$ )  $\leq 2 / N$ .
qed
have isCont ( $\lambda w. (\text{norm}(A i * v w - A j * v w) - 2 / N)$ )  $z$ 
by (intro continuous_intros)
from isCont_tendsto_compose [OF this  $z$ ]
have  $\lim$ : ( $\lambda w. \text{norm}(A i * v ((\gamma w - x) /_R \text{norm } (\gamma w - x)) -
A j * v ((\gamma w - x) /_R \text{norm } (\gamma w - x))) - 2 / N$ )
 $\longrightarrow$  norm ( $A i * v z - A j * v z$ ) - 2 / N
by auto
have dist ( $A i * v z$ ) ( $A j * v z$ )  $\leq 2 / N$ 
using tendsto_upperbound [OF  $\lim$  even] by (auto simp:
dist_norm)
with  $N$  show dist ( $A i * v z$ ) ( $A j * v z$ )  $< \varepsilon$ 
by linarith
qed
qed
then have  $d \cdot z = 0$ 
using CA_eq d orthogonal_def by auto
then show False

```

```

      using ‹ $0 < \xi$ › ‹ $\xi \leq |d \cdot z|$ › by auto
    qed
  ultimately show ?thesis
    using dim_eq_full by fastforce
  qed
  finally have ?CA = UNIV .
  then have Cauchy  $(\lambda n. (A \ n) \ *v \ axis \ j \ 1)$ 
    by auto
  then obtain L where  $(\lambda n. A \ n \ *v \ axis \ j \ 1) \ \longrightarrow \ L$ 
    by (auto simp: Cauchy_convergent_iff convergent_def)
  then have  $(\lambda x. (A \ x \ *v \ axis \ j \ 1) \ \$ \ i) \ \longrightarrow \ L \ \$ \ i$ 
    by (rule tendsto_vec_nth)
  then show  $\exists a. (\lambda n. A \ n \ \$ \ i \ \$ \ j) \ \longrightarrow \ a$ 
    by (force simp: vax)
  qed
  then obtain B where  $B: \bigwedge i \ j. (\lambda n. A \ n \ \$ \ i \ \$ \ j) \ \longrightarrow \ B \ \$ \ i \ \$ \ j$ 
    by (auto simp: lambda_skolem)
  have lin_df: linear  $(f' \ x)$ 
    and lim_df:  $((\lambda y. (1 / norm \ (y - x)) \ *_R \ (f \ y - (f \ x + f' \ x \ (y - x)))) \ \longrightarrow \ 0) \ (at \ x \ within \ S)$ 
    using ‹ $x \in S$ › assms by (auto simp: has_derivative_within linear_linear)
  moreover
  interpret linear  $f' \ x$  by fact
  have  $(matrix \ (f' \ x) - B) \ *v \ w = 0$  for w
  proof (rule lemma_partial_derivatives [of  $(*_v) \ (matrix \ (f' \ x) - B)$ ])
    show linear  $((*_v) \ (matrix \ (f' \ x) - B))$ 
      by (rule matrix_vector_mul_linear)
    have  $((\lambda y. ((f \ x + f' \ x \ (y - x)) - f \ y) /_R \ norm \ (y - x)) \ \longrightarrow \ 0) \ (at \ x \ within \ S)$ 
      using tendsto_minus [OF lim_df] by (simp add: field_split_simps)
    then show  $((\lambda y. (matrix \ (f' \ x) - B) \ *v \ (y - x) /_R \ norm \ (y - x)) \ \longrightarrow \ 0) \ (at \ x \ within \ S)$ 
      proof (rule Lim_transform)
        have  $((\lambda y. ((f \ y + B \ *v \ x - (f \ x + B \ *v \ y)) /_R \ norm \ (y - x))) \ \longrightarrow \ 0) \ (at \ x \ within \ S)$ 
          proof (clarsimp simp add: Lim_within dist_norm)
            fix e :: real
            assume e > 0
            then obtain q::nat where q ≠ 0 and qe2:  $1/q < e/2$ 
              by (metis divide_pos_pos inverse_eq_divide real_arch_inverse zero_less_numeral)
            let ?g =  $\lambda p. \sum (\lambda i. \sum (\lambda j. abs((A \ p - B) \$i \$j)) \ UNIV) \ UNIV$ 
            have  $(\lambda k. onorm \ (\lambda y. (A \ k - B) \ *v \ y)) \ \longrightarrow \ 0$ 
              proof (rule Lim_null_comparison)
                show  $\forall_F \ k \ in \ sequentially. \ norm \ (onorm \ (\lambda y. (A \ k - B) \ *v \ y)) \leq$ 
                  ?g k
                  proof (rule eventually_sequentiallyI)
                    fix k :: nat
                    assume  $0 \leq k$ 

```

```

      have 0 ≤ onorm ((*v) (A k - B))
      using matrix_vector_mul_bounded_linear
      by (rule onorm_pos_le)
      then show norm (onorm ((*v) (A k - B))) ≤ (∑ i ∈ UNIV.
∑ j ∈ UNIV. |(A k - B) $ i $ j|)
      by (simp add: onorm_le_matrix_component_sum del:
vector_minus_component)
    qed
  next
  show ?g ⟶ 0
  using B Lim_null tendsto_rabs_zero_iff by (fastforce intro!:
tendsto_null_sum)
  qed
  with ⟨e > 0⟩ obtain p where ∧ n. n ≥ p ⟹ |onorm ((*v) (A n -
B))| < e/2
  unfolding lim_sequentially by (metis diff_zero dist_real_def
divide_pos_pos zero_less_numeral)
  then have pge2: |onorm ((*v) (A (p + q) - B))| < e/2
  using le_add1 by blast
  show ∃ d > 0. ∀ y ∈ S. y ≠ x ∧ norm (y - x) < d ⟶
inverse (norm (y - x)) * norm (f y + B *v x - (f x + B *v
y)) < e
  proof (intro exI, safe)
    show 0 < δ(p + q)
    by (simp add: δ)
  next
  fix y
  assume y: y ∈ S norm (y - x) < δ(p + q) and y ≠ x
  have *: [|norm(b - c) < e - d; norm(y - x - b) ≤ d| ⟹ norm(y
- x - c) < e
  for b c d e x and y:: real^n
  using norm_triangle_ineq2 [of y - x - c y - x - b] by simp
  have norm (f y - f x - B *v (y - x)) < e * norm (y - x)
  proof (rule *)
    show norm (f y - f x - A (p + q) *v (y - x)) ≤ norm (y - x)
/ (Suc (p + q))
    using A [OF y] by simp
    have norm (A (p + q) *v (y - x) - B *v (y - x)) ≤ onorm(λx.
(A(p + q) - B) *v x) * norm(y - x)
    by (metis linear_linear matrix_vector_mul_linear ma-
trix_vector_mult_diff_rdistrib onorm)
    also have ... < (e/2) * norm (y - x)
    using ⟨y ≠ x⟩ pge2 by auto
    also have ... ≤ (e - 1 / (Suc (p + q))) * norm (y - x)
  proof (rule mult_right_mono)
    have 1 / Suc (p + q) ≤ 1 / q
    using ⟨q ≠ 0⟩ by (auto simp: field_split_simps)
    also have ... < e/2
    using qe2 by auto

```

```

      finally show  $e / 2 \leq e - 1 / \text{real } (\text{Suc } (p + q))$ 
        by linarith
      qed auto
      finally show  $\text{norm } (A (p + q) * v (y - x) - B * v (y - x)) < e * \text{norm } (y - x) - \text{norm } (y - x) / \text{real } (\text{Suc } (p + q))$ 
        by (simp add: algebra_simps)
      qed
      then show  $\text{inverse } (\text{norm } (y - x)) * \text{norm } (f y + B * v x - (f x + B * v y)) < e$ 
        using  $\langle y \neq x \rangle$  by (simp add: field_split_simps algebra_simps)
      qed
    qed
    then show  $((\lambda y. (\text{matrix } (f' x) - B) * v (y - x) /_R \text{norm } (y - x) - (f x + f' x (y - x) - f y) /_R \text{norm } (y - x))) \longrightarrow 0$ 
      (at  $x$  within  $S$ )
      by (simp add: algebra_simps diff lin_df scalar_mult_eq_scaleR)
    qed
  qed (use  $x$  in  $\langle \text{simp}; \text{auto simp: not_less} \rangle$ )
  ultimately have  $f' x = (*v) B$ 
    by (force simp: algebra_simps scalar_mult_eq_scaleR)
  show  $\text{matrix } (f' x) \$ m \$ n \leq b$ 
  proof (rule tendsto_upperbound [of  $\lambda i. (A i \$ m \$ n) \_ \text{sequentially}$ ])
    show  $(\lambda i. A i \$ m \$ n) \longrightarrow \text{matrix } (f' x) \$ m \$ n$ 
      by (simp add: B  $\langle f' x = (*v) B \rangle$ )
    show  $\forall_F i$  in sequentially.  $A i \$ m \$ n \leq b$ 
      by (simp add: Ab less_eq_real_def)
  qed auto
next
fix  $e :: \text{real}$ 
assume  $x \in S$  and  $b$ :  $\text{matrix } (f' x) \$ m \$ n \leq b$  and  $e > 0$ 
then obtain  $d$  where  $d > 0$ 
  and  $d$ :  $\bigwedge y. y \in S \implies 0 < \text{dist } y x \wedge \text{dist } y x < d \implies \text{norm } (f y - f x - f' x (y - x)) / (\text{norm } (y - x)) < e/2$ 
  using  $f$  [OF  $\langle x \in S \rangle$ ]
  by (simp add: Deriv.has_derivative_at_within Lim_within)
  (auto simp add: field_simps dest: spec [of _  $e/2$ ])
let  $?A = \text{matrix}(f' x) - (\chi \ i \ j. \text{if } i = m \wedge j = n \text{ then } e / 4 \text{ else } 0)$ 
obtain  $B$  where  $B \text{Rats}$ :  $\bigwedge i \ j. B \$ i \$ j \in \mathbf{Q}$  and  $B o_e6$ :  $\text{onorm}((*v) (?A - B)) < e/6$ 
  using matrix_rational_approximation  $\langle e > 0 \rangle$ 
  by (metis zero_less_divide_iff zero_less_numeral)
show  $\exists d > 0. \exists A. A \$ m \$ n < b \wedge (\forall i \ j. A \$ i \$ j \in \mathbf{Q}) \wedge (\forall y \in S. \text{norm } (y - x) < d \implies \text{norm } (f y - f x - A * v (y - x)) \leq e * \text{norm } (y - x))$ 
  proof (intro exI conjI ballI allI impI)
    show  $d > 0$ 
      by (rule  $\langle d > 0 \rangle$ )

```

```

show B $ m $ n < b
proof -
  have |matrix ((*v) (?A - B)) $ m $ n| ≤ onorm ((*v) (?A - B))
    using component_le_onorm [OF matrix_vector_mul_linear, of _ m
n] by metis
  then show ?thesis
    using b Bo_e6 by simp
qed
show B $ i $ j ∈ Q for i j
  using BRats by auto
show norm (f y - f x - B *v (y - x)) ≤ e * norm (y - x)
  if y ∈ S and y: norm (y - x) < d for y
proof (cases y = x)
  case True then show ?thesis
    by simp
next
  case False
    have *: norm(d' - d) ≤ e/2 ⇒ norm(y - (x + d')) < e/2 ⇒
norm(y - x - d) ≤ e for d d' e and x y::real^n
      using norm_triangle_le [of d' - d y - (x + d')] by simp
    show ?thesis
      proof (rule *)
        have split246: [norm y ≤ e / 6; norm(x - y) ≤ e / 4] ⇒ norm x
≤ e/2 if e > 0 for e and x y :: real^n
          using norm_triangle_le [of y x-y e/2] ‹e > 0› by simp
        have linear (f' x)
          using True f has_derivative_linear by blast
        then have norm (f' x (y - x) - B *v (y - x)) = norm ((matrix (f'
x) - B) *v (y - x))
          by (simp add: matrix_vector_mult_diff_rdistrib)
        also have ... ≤ (e * norm (y - x)) / 2
          proof (rule split246)
            have norm ((?A - B) *v (y - x)) / norm (y - x) ≤ onorm(λx.
(?A - B) *v x)
              by (rule le_onorm) auto
            also have ... < e/6
              by (rule Bo_e6)
          finally have norm ((?A - B) *v (y - x)) / norm (y - x) < e / 6 .
            then show norm ((?A - B) *v (y - x)) ≤ e * norm (y - x) / 6
              by (simp add: field_split_simps False)
            have norm ((matrix (f' x) - B) *v (y - x) - ((?A - B) *v (y -
x))) = norm ((χ i j. if i = m ∧ j = n then e / 4 else 0) *v (y - x))
              by (simp add: algebra_simps)
            also have ... = norm((e/4) *R (y - x)$n *R axis m (1::real))
          proof -
            have (∑ j∈UNIV. (if i = m ∧ j = n then e / 4 else 0) * (y $ j
- x $ j)) * 4 = e * (y $ n - x $ n) * axis m 1 $ i for i
              proof (cases i=m)
                case True then show ?thesis

```



```

      by (auto simp: if_distrib [of  $\lambda z. z * \_$ ] cong: if_cong)
    next
      case False then show ?thesis
        by (simp add: axis_def)
      qed
    then have ( $\chi$   $i$   $j$ . if  $i = m \wedge j = n$  then  $e / 4$  else  $0$ )  $*v$   $(y - x)$ 
=  $(e/4) *_R (y - x) \$n *_R axis\ m\ (1::real)$ 
      by (auto simp: vec_eq_iff matrix_vector_mult_def)
    then show ?thesis
      by metis
    qed
  also have  $\dots \leq e * norm (y - x) / 4$ 
  proof -
    have  $|y \$n - x \$n| \leq norm (y - x)$ 
      by (metis component_le_norm_cart vector_minus_component)
    with  $\langle e > 0 \rangle$  show ?thesis
      by (simp add: norm_mult abs_mult)
    qed
  finally show  $norm ((matrix (f' x) - B) *v (y - x) - ((?A - B)
*v (y - x))) \leq e * norm (y - x) / 4$  .
    show  $0 < e * norm (y - x)$ 
      by (simp add: False  $\langle e > 0 \rangle$ )
    qed
  finally show  $norm (f' x (y - x) - B *v (y - x)) \leq (e * norm (y -
x)) / 2$  .
    show  $norm (f y - (f x + f' x (y - x))) < (e * norm (y - x)) / 2$ 
      using False  $d [OF \langle y \in S \rangle y]$  by (simp add: dist_norm field_simps)
    qed
  qed
  qed
  qed
  qed auto
  qed
  show negligible ?M
    using negligible_subset [OF  $nN MN$ ] .
  qed
  then show ?thesis
    by (simp add: borel_measurable_vimage_halfspace_component_le sets_restrict_space_iff
assms)
  qed

```

theorem borel_measurable_det_Jacobian:

```

fixes  $f :: real^{n::\{finite,wellorder\}} \Rightarrow real^{n::\_}$ 
assumes  $S \in sets\ lebesgue$  and  $f: \bigwedge x. x \in S \implies (f\ has\_derivative\ f'\ x)$  (at
 $x$  within  $S$ )
shows  $(\lambda x. det(matrix(f' x))) \in borel\_measurable (lebesgue\_on\ S)$ 
unfolding det_def
by (intro measurable) (auto intro: f borel_measurable_partial_derivatives [OF

```

S])

The localisation wrt S uses the same argument for many similar results.

theorem *borel_measurable_lebesgue_on_preimage_borel*:

fixes $f :: 'a::euclidean_space \Rightarrow 'b::euclidean_space$

assumes $S \in \text{sets lebesgue}$

shows $f \in \text{borel_measurable (lebesgue_on } S) \longleftrightarrow$

$(\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue})$

proof –

have $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} \in \text{sets lebesgue} \longleftrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue}$

if $T \in \text{sets borel}$ **for** T

proof (cases $0 \in T$)

case *True*

then have $\{x \in S. f x \in T\} = \{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} \cap S$
 $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} = \{x \in S. f x \in T\} \cup -S$

by *auto*

then show *?thesis*

by (*metis (no_types, lifting) Compl_in_sets_lebesgue assms sets.Int sets.Un*)

next

case *False*

then have $\{x. (\text{if } x \in S \text{ then } f x \text{ else } 0) \in T\} = \{x \in S. f x \in T\}$

by *auto*

then show *?thesis*

by *auto*

qed

then show *?thesis*

unfolding *borel_measurable_lebesgue_preimage_borel borel_measurable_if [OF assms, symmetric]*

by *blast*

qed

lemma *sets_lebesgue_almost_borel*:

assumes $S \in \text{sets lebesgue}$

obtains $B N$ **where** $B \in \text{sets borel negligible } N B \cup N = S$

by (*metis assms negligible_iff_null_sets negligible_subset null_sets_completionI sets_completionE sets_lborel*)

lemma *double_lebesgue_sets*:

assumes $S: S \in \text{sets lebesgue}$ **and** $T: T \in \text{sets lebesgue}$ **and** *fm*: $f ' S \subseteq T$

shows $(\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue})$

\longleftrightarrow

$f \in \text{borel_measurable (lebesgue_on } S) \wedge$

$(\forall U. \text{negligible } U \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue})$

(**is** *?lhs* \longleftrightarrow *__* \wedge *?rhs*)

unfolding *borel_measurable_lebesgue_on_preimage_borel [OF S]*

proof (*intro iffI allI conjI impI, safe*)

fix $V :: 'b \text{ set}$

assume $*$: $\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$

```

    and  $V \in \text{sets borel}$ 
  then have  $V: V \in \text{sets lebesgue}$ 
    by simp
  have  $\{x \in S. f x \in V\} = \{x \in S. f x \in T \cap V\}$ 
    using fim by blast
  also have  $\{x \in S. f x \in T \cap V\} \in \text{sets lebesgue}$ 
    using  $T V * \text{le\_inf\_iff}$  by blast
  finally show  $\{x \in S. f x \in V\} \in \text{sets lebesgue} .$ 
next
  fix  $U :: 'b \text{ set}$ 
  assume  $\forall U. U \in \text{sets lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    negligible  $U U \subseteq T$ 
  then show  $\{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    using negligible_imp_sets by blast
next
  fix  $U :: 'b \text{ set}$ 
  assume 1 [rule_format]:  $(\forall T. T \in \text{sets borel} \longrightarrow \{x \in S. f x \in T\} \in \text{sets lebesgue})$ 
  and 2 [rule_format]:  $\forall U. \text{negligible } U \wedge U \subseteq T \longrightarrow \{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
  and  $U \in \text{sets lebesgue } U \subseteq T$ 
  then obtain  $C N$  where  $C: C \in \text{sets borel} \wedge \text{negligible } N \wedge C \cup N = U$ 
    using sets_lebesgue_almost_borel by metis
  then have  $\{x \in S. f x \in C\} \in \text{sets lebesgue}$ 
    by (blast intro: 1)
  moreover have  $\{x \in S. f x \in N\} \in \text{sets lebesgue}$ 
    using  $C \langle U \subseteq T \rangle$  by (blast intro: 2)
  moreover have  $\{x \in S. f x \in C \cup N\} = \{x \in S. f x \in C\} \cup \{x \in S. f x \in N\}$ 
    by auto
  ultimately show  $\{x \in S. f x \in U\} \in \text{sets lebesgue}$ 
    using  $C$  by auto
qed

```

9.32.3 Simplest case of Sard's theorem (we don't need continuity of derivative)

lemma *Sard_lemma00*:

```

  fixes  $P :: 'b::\text{euclidean\_space} \text{ set}$ 
  assumes  $a \geq 0$  and  $a: a *_{\mathbb{R}} i \neq 0$  and  $i: i \in \text{Basis}$ 
    and  $P: P \subseteq \{x. a *_{\mathbb{R}} i \cdot x = 0\}$ 
    and  $0 \leq m \ 0 \leq e$ 
  obtains  $S$  where  $S \in \text{lmeasurable}$ 
    and  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm}(z - t) \leq e)\} \subseteq S$ 
    and  $\text{measure lebesgue } S \leq (2 * e) * (2 * m) \wedge (\text{DIM}('b) - 1)$ 

```

proof –

```

  have  $a > 0$ 
    using assms by simp
  let  $?v = (\sum_{j \in \text{Basis}. (\text{if } j = i \text{ then } e \text{ else } m) *_{\mathbb{R}} j)$ 
  show thesis

```

```

proof
  have  $- e \leq x \cdot i \ x \cdot i \leq e$ 
    if  $t \in P$  norm  $(x - t) \leq e$  for  $x \ t$ 
    using  $\langle a > 0 \rangle$  that Basis_le_norm [of i x-t] P i
    by (auto simp: inner_commute algebra_simps)
  moreover have  $- m \leq x \cdot j \ x \cdot j \leq m$ 
    if  $\text{norm } x \leq m$   $t \in P$  norm  $(x - t) \leq e$   $j \in \text{Basis}$  and  $j \neq i$ 
    for  $x \ t \ j$ 
    using that Basis_le_norm [of j x] by auto
  ultimately
  show  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm } (z - t) \leq e)\} \subseteq \text{cbox } (-?v) \ ?v$ 
    by (auto simp: mem_box)
  have  $*$ :  $\forall k \in \text{Basis}. - ?v \cdot k \leq ?v \cdot k$ 
    using  $\langle 0 \leq m \rangle \langle 0 \leq e \rangle$  by (auto simp: inner_Basis)
  have  $2$ :  $2 \wedge \text{DIM}('b) = 2 * 2 \wedge (\text{DIM}('b) - \text{Suc } 0)$ 
    by (metis DIM_positive Suc_pred power_Suc)
  show measure lebesgue (cbox (-?v) ?v)  $\leq 2 * e * (2 * m) \wedge (\text{DIM}('b) - 1)$ 
    using  $\langle i \in \text{Basis} \rangle$ 
    by (simp add: content_cbox [OF *] prod.distrib prod.If_cases Diff_eq [symmetric]
  2)
qed blast
qed

```

As above, but reorienting the vector (HOL Light's @textGEOM_BASIS_MULTIPLE_TAC)

lemma *Sard_lemma0*:

```

fixes  $P :: (\text{real}^n :: \{\text{finite}, \text{wellorder}\}) \text{ set}$ 
assumes  $a \neq 0$ 
  and  $P: P \subseteq \{x. a \cdot x = 0\}$  and  $0 \leq m \ 0 \leq e$ 
obtains  $S$  where  $S \in \text{lmeasurable}$ 
  and  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm}(z - t) \leq e)\} \subseteq S$ 
  and measure lebesgue S  $\leq (2 * e) * (2 * m) \wedge (\text{CARD}('n) - 1)$ 
proof -
  obtain  $T$  and  $k :: 'n$  where  $T: \text{orthogonal\_transformation } T$  and  $a: a = T$ 
  (norm a *R axis k (1::real))
  using rotation_rightward_line by metis
  have  $T \text{inv}$  [simp]:  $T (\text{inv } T \ x) = x$  for  $x$ 
  by (simp add: T_orthogonal_transformation_surj_surj_f_inv_f)
  obtain  $S$  where  $S: S \in \text{lmeasurable}$ 
  and  $\text{sub}S: \{z. \text{norm } z \leq m \wedge (\exists t \in T^{-1}P. \text{norm}(z - t) \leq e)\} \subseteq S$ 
  and  $mS: \text{measure lebesgue } S \leq (2 * e) * (2 * m) \wedge (\text{CARD}('n) - 1)$ 
proof (rule Sard_lemma0 [of norm a axis k (1::real) T^{-1}P m e])
  have  $\text{norm a *}_R \text{axis k } 1 \cdot x = 0$  if  $T \ x \in P$  for  $x$ 
  by (smt (verit, del_insts) P T a mem_Collect_eq orthogonal_transformation_def
  subset_eq that)
  then show  $T^{-1}P \subseteq \{x. \text{norm a *}_R \text{axis k } 1 \cdot x = 0\}$ 
  by auto
qed (use assms T in auto)
show thesis

```

```

proof
  show  $T \text{ ' } S \in \text{lmeasurable}$ 
    using  $S \text{ measurable\_orthogonal\_image } T \text{ by blast}$ 
    have  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm } (z - t) \leq e)\} \subseteq T \text{ ' } \{z. \text{norm } z \leq m \wedge$ 
 $(\exists t \in T \text{ ' } P. \text{norm } (z - t) \leq e)\}$ 
    proof clarsimp
      fix  $x t$ 
      assume  $\S: \text{norm } x \leq m \ t \in P \ \text{norm } (x - t) \leq e$ 
      then have  $\text{norm } (\text{inv } T \ x) \leq m$ 
        using orthogonal_transformation_inv [OF T] by (simp add: orthogonal_
 $\text{transformation\_norm})$ 
      moreover have  $\exists t \in T \text{ ' } P. \text{norm } (\text{inv } T \ x - t) \leq e$ 
      by (smt (verit, del_insts) T Tinv \S linear_diff orthogonal_transformation_def
 $\text{orthogonal\_transformation\_norm vimage\_eq})$ 
      ultimately show  $x \in T \text{ ' } \{z. \text{norm } z \leq m \wedge (\exists t \in T \text{ ' } P. \text{norm } (z - t) \leq$ 
 $e)\}$ 
      by force
    qed
  then show  $\{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm } (z - t) \leq e)\} \subseteq T \text{ ' } S$ 
    using image_mono [OF subS] by (rule order_trans)
  show  $\text{measure lebesgue } (T \text{ ' } S) \leq 2 * e * (2 * m) ^ (\text{CARD}('n) - 1)$ 
    using mS T by (simp add: S measure_orthogonal_image)
qed
qed

```

As above, but translating the sets (HOL Light's @textGEN_GEOM_ORIGIN_TAC)

lemma *Sard_lemma1*:

```

fixes  $P :: (\text{real}^n :: \{\text{finite, wellorder}\}) \text{ set}$ 
assumes  $P: \text{dim } P < \text{CARD}('n) \text{ and } 0 \leq m \ 0 \leq e$ 
obtains  $S \text{ where } S \in \text{lmeasurable}$ 
  and  $\{z. \text{norm}(z - w) \leq m \wedge (\exists t \in P. \text{norm}(z - w - t) \leq e)\} \subseteq S$ 
  and  $\text{measure lebesgue } S \leq (2 * e) * (2 * m) ^ (\text{CARD}('n) - 1)$ 
proof -
  obtain  $a \text{ where } a \neq 0 \ P \subseteq \{x. a \cdot x = 0\}$ 
    using lowdim_subset_hyperplane [of P] P span_base by auto
  then obtain  $S \text{ where } S: S \in \text{lmeasurable}$ 
    and  $\text{subS}: \{z. \text{norm } z \leq m \wedge (\exists t \in P. \text{norm}(z - t) \leq e)\} \subseteq S$ 
    and  $mS: \text{measure lebesgue } S \leq (2 * e) * (2 * m) ^ (\text{CARD}('n) - 1)$ 
    by (rule Sard_lemma0 [OF ___ \langle 0 \leq m \rangle \langle 0 \leq e \rangle])
  show thesis
proof
  show  $(+)w \text{ ' } S \in \text{lmeasurable}$ 
    by (metis measurable_translation S)
  show  $\{z. \text{norm } (z - w) \leq m \wedge (\exists t \in P. \text{norm } (z - w - t) \leq e)\} \subseteq (+)w \text{ ' } S$ 
    using subS by force
  show  $\text{measure lebesgue } ((+)w \text{ ' } S) \leq 2 * e * (2 * m) ^ (\text{CARD}('n) - 1)$ 
    by (metis measure_translation mS)
qed

```

3770

qed

lemma *Sard_lemma2*:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \{\text{finite}, \text{wellorder}\}$

assumes $m \leq n$: $\text{CARD}(m) \leq \text{CARD}(n)$ (is $?m \leq ?n$)

and $B > 0$ bounded S

and $derS$: $\bigwedge x. x \in S \implies (f \text{ has_derivative } f' x)$ (at x within S)

and $rank$: $\bigwedge x. x \in S \implies \text{rank}(\text{matrix}(f' x)) < \text{CARD}(n)$

and B : $\bigwedge x. x \in S \implies \text{onorm}(f' x) \leq B$

shows $\text{negligible}(f' S)$

proof -

have lin_f' : $\bigwedge x. x \in S \implies \text{linear}(f' x)$

using $derS$ $\text{has_derivative_linear}$ by $blast$

show $?thesis$

proof (clarsimp simp add: $\text{negligible_outer_le}$)

fix $e :: \text{real}$

assume $e > 0$

obtain c where $csub$: $S \subseteq \text{cbox}(-(\text{vec } c))(\text{vec } c)$ and $c > 0$

proof -

obtain b where b : $\bigwedge x. x \in S \implies \text{norm } x \leq b$

using $\langle \text{bounded } S \rangle$ by (auto simp: bounded_iff)

show $thesis$

proof

have $-|b| - 1 \leq x \ \$ \ i \ \wedge \ x \ \$ \ i \ \leq |b| + 1$ if $x \in S$ for $x \ i$

using $\text{component_le_norm_cart}$ [of $x \ i$] b [OF that] by auto

then show $S \subseteq \text{cbox}(-\text{vec}(|b| + 1))(\text{vec}(|b| + 1))$

by (auto simp: mem_box_cart)

qed auto

qed

then have box_cc : $\text{box}(-(\text{vec } c))(\text{vec } c) \neq \{\}$ and $cbox_cc$: $\text{cbox}(-(\text{vec } c))(\text{vec } c) \neq \{\}$

by (auto simp: $\text{interval_eq_empty_cart}$)

obtain d where $d > 0$ $d \leq B$

and d : $(d * 2) * (4 * B) ^{(?n - 1)} \leq e / (2 * c) ^{?m} / ?m ^{?m}$

apply (rule that [of $\min B (e / (2 * c) ^{?m} / ?m ^{?m} / (4 * B) ^{(?n - 1)} / 2)$])

using $\langle B > 0 \rangle \langle c > 0 \rangle \langle e > 0 \rangle$

by (simp_all add: divide_simps $\text{min_mult_distrib_right}$)

have $\exists r. 0 < r \wedge r \leq 1/2 \wedge$

$(x \in S$

$\implies (\forall y. y \in S \wedge \text{norm}(y - x) < r$

$\implies \text{norm}(f y - f x - f' x (y - x)) \leq d * \text{norm}(y - x))$) for x

proof (cases $x \in S$)

case True

then obtain r where $r > 0$

and $\bigwedge y. [y \in S; \text{norm}(y - x) < r]$

$\implies \text{norm}(f y - f x - f' x (y - x)) \leq d * \text{norm}(y - x)$

using $derS$ $\langle d > 0 \rangle$ by (force simp: $\text{has_derivative_within_alt}$)

then show $?thesis$

```

    by (rule_tac x=min r (1/2) in exI) simp
next
case False
then show ?thesis
  by (rule_tac x=1/2 in exI) simp
qed
then obtain r where r12:  $\bigwedge x. 0 < r x \wedge r x \leq 1/2$ 
  and r:  $\bigwedge x y. \llbracket x \in S; y \in S; \text{norm}(y - x) < r x \rrbracket$ 
     $\implies \text{norm}(f y - f x - f' x (y - x)) \leq d * \text{norm}(y - x)$ 
  by metis
then have ga: gauge ( $\lambda x. \text{ball } x (r x)$ )
  by (auto simp: gauge_def)
obtain  $\mathcal{D}$  where  $\mathcal{D}$ : countable  $\mathcal{D}$  and sub_cc:  $\bigcup \mathcal{D} \subseteq \text{cbox } (- \text{vec } c) (\text{vec } c)$ 
  and cbox:  $\bigwedge K. K \in \mathcal{D} \implies \text{interior } K \neq \{\}$   $\wedge (\exists u v. K = \text{cbox } u v)$ 
  and disjointish: pairwise ( $\lambda A B. \text{interior } A \cap \text{interior } B = \{\}$ )  $\mathcal{D}$ 
  and covered:  $\bigwedge K. K \in \mathcal{D} \implies \exists x \in S \cap K. K \subseteq \text{ball } x (r x)$ 
  and close:  $\bigwedge u v. \text{cbox } u v \in \mathcal{D} \implies \exists n. \forall i::'m. v \$ i - u \$ i = 2*c / 2^{\wedge}n$ 
  and covers:  $S \subseteq \bigcup \mathcal{D}$ 
  apply (rule covering_lemma [OF csub box_cc ga])
  apply (auto simp: Basis_vec_def cart_eq_inner_axis [symmetric])
  done
let ? $\mu$  = measure lebesgue
have  $\exists T. T \in \text{lmeasurable} \wedge f' (K \cap S) \subseteq T \wedge ?\mu T \leq e / (2*c) \wedge ?m * ?\mu$ 
K
  if  $K \in \mathcal{D}$  for  $K$ 
proof -
  obtain  $u v$  where  $uv: K = \text{cbox } u v$ 
    using cbox  $\langle K \in \mathcal{D} \rangle$  by blast
  then have  $uv\_ne: \text{cbox } u v \neq \{\}$ 
    using cbox that by fastforce
  obtain  $x$  where  $x: x \in S \cap \text{cbox } u v \text{ cbox } u v \subseteq \text{ball } x (r x)$ 
    using  $\langle K \in \mathcal{D} \rangle$  covered  $uv$  by blast
  then have  $\text{dim } (\text{range } (f' x)) < ?n$ 
    using rank_dim_range [of matrix (f' x)] x rank[of x]
    by (auto simp: matrix_works scalar_mult_eq_scaleR lin_f')
  then obtain  $T$  where  $T: T \in \text{lmeasurable}$ 
    and subT:  $\{z. \text{norm}(z - f x) \leq (2 * B) * \text{norm}(v - u) \wedge (\exists t \in \text{range } (f' x). \text{norm}(z - f x - t) \leq d * \text{norm}(v - u))\} \subseteq T$ 
    and measT:  $?\mu T \leq (2 * (d * \text{norm}(v - u))) * (2 * ((2 * B) * \text{norm}(v - u))) \wedge (?n - 1)$ 
    (is  $\_ \leq ?DVU$ )
    using Sard_lemma1 [of range (f' x) (2 * B) * norm(v - u) d * norm(v - u)]
    using  $\langle B > 0 \rangle \langle d > 0 \rangle$  by auto
  show ?thesis
proof (intro exI conjI)
  have  $f' (K \cap S) \subseteq \{z. \text{norm}(z - f x) \leq (2 * B) * \text{norm}(v - u) \wedge (\exists t \in \text{range } (f' x). \text{norm}(z - f x - t) \leq d * \text{norm}(v - u))\}$ 
    unfolding uv

```

```

proof (clarsimp simp: mult.assoc, intro conjI)
  fix y
  assume y: y ∈ cbox u v and y ∈ S
  then have norm (y - x) < r x
    by (metis dist_norm mem_ball norm_minus_commute subsetCE x(2))
  then have le_dyx: norm (f y - f x - f' x (y - x)) ≤ d * norm (y - x)
    using r [of x y] x ⟨y ∈ S⟩ by blast
  have yx_le: norm (y - x) ≤ norm (v - u)
  proof (rule norm_le_componentwise_cart)
    show norm ((y - x) $ i) ≤ norm ((v - u) $ i) for i
      using x y by (force simp: mem_box_cart dest!: spec [where x=i])
  qed
  have *: [norm(y - x - z) ≤ d; norm z ≤ B; d ≤ B] ⇒ norm(y - x)
≤ 2 * B
    for x y z :: real^n::_ and d B
    using norm_triangle_ineq2 [of y - x z] by auto
  show norm (f y - f x) ≤ 2 * (B * norm (v - u))
  proof (rule * [OF le_dyx])
    have norm (f' x (y - x)) ≤ onorm (f' x) * norm (y - x)
      using onorm [of f' x y-x] by (meson IntE lin_f' linear_linear x(1))
    also have ... ≤ B * norm (v - u)
      by (meson B IntE lin_f' linear_linear mult_mono' norm_ge_zero
onorm_pos_le x(1) yx_le)
    finally show norm (f' x (y - x)) ≤ B * norm (v - u) .
    show d * norm (y - x) ≤ B * norm (v - u)
      using ⟨B > 0⟩ by (auto intro: mult_mono [OF ⟨d ≤ B⟩ yx_le])
  qed
  show ∃ t. norm (f y - f x - f' x t) ≤ d * norm (v - u)
    by (smt (verit, best) ⟨0 < d⟩ le_dyx mult_le_cancel_left_pos yx_le)
qed
with subT show f '(K ∩ S) ⊆ T by blast
show ?μ T ≤ e / (2*c) ^ ?m * ?μ K
proof (rule order_trans [OF measT])
  have ?DVU = (d * 2 * (4 * B) ^ (?n - 1)) * norm (v - u) ^ ?n
    using ⟨c > 0⟩
    apply (simp add: algebra_simps)
    by (metis Suc_pred power_Suc zero_less_card_finite)
  also have ... ≤ (e / (2*c) ^ ?m / (?m ^ ?m)) * norm(v - u) ^ ?n
    by (rule mult_right_mono [OF d]) auto
  also have ... ≤ e / (2*c) ^ ?m * ?μ K
  proof -
    have u ∈ ball (x) (r x) v ∈ ball x (r x)
    using box_ne_empty(1) contra_subsetD [OF x(2)] mem_box(2) uv_ne
by fastforce+
    moreover have r x ≤ 1/2
      using r12 by auto
    ultimately have norm (v - u) ≤ 1
      using norm_triangle_half_r [of x u 1 v]
      by (metis (no_types, opaque_lifting) dist_commute dist_norm

```



```

less_eq_real_def less_le_trans mem_ball)
  then have norm (v - u) ^ ?n ≤ norm (v - u) ^ ?m
    by (simp add: power_decreasing [OF mlen])
  also have ... ≤ ?μ K * real (?m ^ ?m)
  proof -
    obtain n where n:  $\bigwedge i. v \$ i - u \$ i = 2 * c / 2^n$ 
      using close [of u v]  $\langle K \in \mathcal{D} \rangle uv$  by blast
    have norm (v - u) ^ ?m ≤  $(\sum i \in UNIV. |(v - u) \$ i|) ^ ?m$ 
      by (intro norm_le_l1_cart power_mono) auto
    also have ... ≤  $(\prod i \in UNIV. v \$ i - u \$ i) * real CARD('m) ^$ 
CARD('m)
      by (simp add: n field_simps  $\langle c > 0 \rangle$  less_eq_real_def)
    also have ... = ?μ K * real (?m ^ ?m)
      by (simp add: uv uv_ne content_cbox_cart)
    finally show ?thesis .
  qed
  finally have *:  $1 / real (?m ^ ?m) * norm (v - u) ^ ?n \leq ?\mu K$ 
    by (simp add: field_split_simps)
  show ?thesis
    using mult_left_mono [OF *, of e / (2*c) ^ ?m]  $\langle c > 0 \rangle \langle e > 0 \rangle$  by
auto
  qed
  finally show ?DVU ≤ e / (2*c) ^ ?m * ?μ K .
  qed
qed (use T in auto)
qed
then obtain g where meas_g:  $\bigwedge K. K \in \mathcal{D} \implies g K \in lmeasurable$ 
  and sub_g:  $\bigwedge K. K \in \mathcal{D} \implies f '(K \cap S) \subseteq g K$ 
  and le_g:  $\bigwedge K. K \in \mathcal{D} \implies ?\mu (g K) \leq e / (2*c) ^ ?m * ?\mu K$ 
  by metis
have le_e:  $?\mu (\bigcup i \in \mathcal{F}. g i) \leq e$ 
  if  $\mathcal{F} \subseteq \mathcal{D}$  finite  $\mathcal{F}$  for  $\mathcal{F}$ 
  proof -
    have  $?\mu (\bigcup i \in \mathcal{F}. g i) \leq (\sum i \in \mathcal{F}. ?\mu (g i))$ 
      using meas_g  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  by (auto intro: measure_UNION_le [OF  $\langle finite$ 
 $\mathcal{F} \rangle$ ])
    also have ... ≤  $(\sum K \in \mathcal{F}. e / (2*c) ^ ?m * ?\mu K)$ 
      using  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  sum_mono [OF le_g] by (meson le_g subsetCE sum_mono)
    also have ... = e / (2*c) ^ ?m *  $(\sum K \in \mathcal{F}. ?\mu K)$ 
      by (simp add: sum_distrib_left)
    also have ... ≤ e
  proof -
    have  $\mathcal{F}$  division_of  $\bigcup \mathcal{F}$ 
    proof (rule division_ofI)
      show  $K \subseteq \bigcup \mathcal{F}$   $K \neq \{\}$   $\exists a b. K = cbox a b$  if  $K \in \mathcal{F}$  for  $K$ 
        using  $\langle K \in \mathcal{F} \rangle$  covered_cbox  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  by (auto simp: Union_upper)
      show interior K  $\cap$  interior L =  $\{\}$  if  $K \in \mathcal{F}$  and  $L \in \mathcal{F}$  and  $K \neq L$  for
K L
        by (metis (mono_tags, lifting)  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$  pairwiseD disjointish pairwise_subset

```

that)

```

qed (use that in auto)
then have  $\sum ?\mu \mathcal{F} \leq ?\mu (\bigcup \mathcal{F})$ 
  by (simp add: content_division)
also have  $\dots \leq ?\mu (\text{cbox } (- \text{vec } c) (\text{vec } c) :: (\text{real}, 'm) \text{ vec set})$ 
proof (rule measure_mono_fmeasurable)
  show  $\bigcup \mathcal{F} \subseteq \text{cbox } (- \text{vec } c) (\text{vec } c)$ 
    by (meson Sup_subset_mono sub_cc order_trans  $\langle \mathcal{F} \subseteq \mathcal{D} \rangle$ )
qed (use  $\langle \mathcal{F} \text{ division\_of } \bigcup \mathcal{F} \rangle$  lmeasurable_division in auto)
also have  $\dots = \text{content } (\text{cbox } (- \text{vec } c) (\text{vec } c) :: (\text{real}, 'm) \text{ vec set})$ 
  by simp
also have  $\dots \leq (2 \wedge ?m * c \wedge ?m)$ 
  using  $\langle c > 0 \rangle$  by (simp add: content_cbox_if_cart)
finally have  $\sum ?\mu \mathcal{F} \leq (2 \wedge ?m * c \wedge ?m)$  .
then show ?thesis
  using  $\langle e > 0 \rangle \langle c > 0 \rangle$  by (auto simp: field_split_simps)
qed
finally show ?thesis .
qed
show  $\exists T. f \upharpoonright S \subseteq T \wedge T \in \text{lmeasurable} \wedge ?\mu T \leq e$ 
proof (intro exI conjI)
  show  $f \upharpoonright S \subseteq \bigcup (g \upharpoonright \mathcal{D})$ 
    using covers_sub_g by force
  show  $\bigcup (g \upharpoonright \mathcal{D}) \in \text{lmeasurable}$ 
    by (rule fmeasurable_UN_bound [OF  $\langle \text{countable } \mathcal{D} \rangle$  meas_g le_e])
  show  $?\mu (\bigcup (g \upharpoonright \mathcal{D})) \leq e$ 
    by (rule measure_UN_bound [OF  $\langle \text{countable } \mathcal{D} \rangle$  meas_g le_e])
qed
qed
qed

```

theorem baby_Sard:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: \{\text{finite}, \text{wellorder}\}$
assumes $\text{mlen}: \text{CARD}(m) \leq \text{CARD}(n)$

and $\text{der}: \bigwedge x. x \in S \implies (f \text{ has_derivative } f' x)$ (at x within S)

and $\text{rank}: \bigwedge x. x \in S \implies \text{rank}(\text{matrix}(f' x)) < \text{CARD}(n)$

shows negligible($f \upharpoonright S$)

proof –

let $?U = \lambda n. \{x \in S. \text{norm}(x) \leq n \wedge \text{onorm}(f' x) \leq \text{real } n\}$

have $\bigwedge x. x \in S \implies \exists n. \text{norm } x \leq \text{real } n \wedge \text{onorm } (f' x) \leq \text{real } n$

by (meson linear_order_trans real_arch_simple)

then have $\text{eq}: S = (\bigcup n. ?U n)$

by auto

have negligible ($f \upharpoonright ?U n$) **for** n

proof (rule Sard_lemma2 [OF mlen])

show $0 < \text{real } n + 1$

by auto

show bounded ($?U n$)

```

    using bounded_iff by blast
    show (f has_derivative f' x) (at x within ?U n) if x ∈ ?U n for x
      using der that by (force intro: has_derivative_subset)
  qed (use rank in auto)
  then show ?thesis
    by (subst eq) (simp add: image_Union negligible_Union_nat)
qed

```

9.32.4 A one-way version of change-of-variables not assuming injectivity.

lemma *integral_on_image_ubound_weak*:

```

  fixes f :: real^'n::{finite,wellorder} ⇒ real
  assumes S: S ∈ sets lebesgue
    and f: f ∈ borel_measurable (lebesgue_on (g ` S))
    and nonneg_fg: ∧x. x ∈ S ⇒ 0 ≤ f(g x)
    and der_g: ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
    and det_int_fg: (λx. |det (matrix (g' x))| * f(g x)) integrable_on S
    and meas_gim: ∧T. [T ⊆ g ` S; T ∈ sets lebesgue] ⇒ {x ∈ S. g x ∈ T}
  ∈ sets lebesgue
  shows f integrable_on (g ` S) ∧
    integral (g ` S) f ≤ integral S (λx. |det (matrix (g' x))| * f(g x))
    (is _ ∧ _ ≤ ?b)

```

proof –

```

  let ?D = λx. |det (matrix (g' x))|
  have cont_g: continuous_on S g
    using der_g has_derivative_continuous_on by blast
  have [simp]: space (lebesgue_on S) = S
    by (simp add: S)
  have gS_in_sets_leb: g ` S ∈ sets lebesgue
    apply (rule differentiable_image_in_sets_lebesgue)
    using der_g by (auto simp: S differentiable_def differentiable_on_def)
  obtain h where nonneg_h: ∧n x. 0 ≤ h n x
    and h_le_f: ∧n x. x ∈ S ⇒ h n (g x) ≤ f (g x)
    and h_inc: ∧n x. h n x ≤ h (Suc n) x
    and h_meas: ∧n. h n ∈ borel_measurable lebesgue
    and fin_R: ∧n. finite(range (h n))
    and lim: ∧x. x ∈ g ` S ⇒ (λn. h n x) ⟶ f x
  proof –
    let ?f = λx. if x ∈ g ` S then f x else 0
    have ?f ∈ borel_measurable lebesgue ∧ (∀x. 0 ≤ ?f x)
      by (auto simp: gS_in_sets_leb f nonneg_fg measurable_restrict_space_iff
        [symmetric])
    then show ?thesis
      apply (clarsimp simp add: borel_measurable_simple_function_limit_increasing)
      apply (rename_tac h)
      by (rule_tac h=h in that) (auto split: if_split_asm)
  qed
  have h_lmeas: {t. h n (g t) = y} ∩ S ∈ sets lebesgue for y n

```

```

proof -
  have space (lebesgue_on (UNIV::(real,'n) vec set)) = UNIV
    by simp
  then have  $((h\ n) - \{y\} \cap g\ 'S) \in \text{sets } (\text{lebesgue\_on } (g\ 'S))$ 
    by (metis Int_commute borel_measurable_vimage h_meas image_eqI inf_top.right_neutral
sets_restrict_space space_borel space_completion space_lborel)
  then have  $(\{u. h\ n\ u = y\} \cap g\ 'S) \in \text{sets } \text{lebesgue}$ 
    using gS_in_sets_leb
    by (simp add: integral_indicator_fmeasurableI2 sets_restrict_space_iff vimage_def)
  then have  $\{x \in S. g\ x \in (\{u. h\ n\ u = y\} \cap g\ 'S)\} \in \text{sets } \text{lebesgue}$ 
    using meas_gim[of (\{u. h\ n\ u = y\} \cap g\ 'S)] by force
  moreover have  $\{t. h\ n\ (g\ t) = y\} \cap S = \{x \in S. g\ x \in (\{u. h\ n\ u = y\} \cap g\ 'S)\}$ 
    by blast
  ultimately show ?thesis
    by auto
qed
have hint:  $h\ n\ \text{integrable\_on } g\ 'S \wedge \text{integral } (g\ 'S) (h\ n) \leq \text{integral } S (\lambda x. ?D\ x * h\ n\ (g\ x))$ 
  (is ?INT  $\wedge$  ?lhs  $\leq$  ?rhs) for n
proof -
  let ?R = range (h n)
  have hn_eq:  $h\ n = (\lambda x. \sum_{y \in ?R}. y * \text{indicator\_real } \{x. h\ n\ x = y\} x)$ 
    by (simp add: indicator_def if_distrib fin_R cong: if_cong)
  have yind:  $(\lambda t. y * \text{indicator}\{x. h\ n\ x = y\} t) \text{integrable\_on } (g\ 'S) \wedge$ 
     $(\text{integral } (g\ 'S) (\lambda t. y * \text{indicator } \{x. h\ n\ x = y\} t))$ 
     $\leq \text{integral } S (\lambda t. |\det (\text{matrix } (g\ 't))| * y * \text{indicator } \{x. h\ n\ x = y\}$ 
    (g t))
    if y:  $y \in ?R$  for y::real
  proof (cases y=0)
    case True
      then show ?thesis using gS_in_sets_leb integrable_0 by force
    next
      case False
        with that have  $y > 0$ 
          using less_eq_real_def nonneg_h by fastforce
        have  $(\lambda x. \text{if } x \in \{t. h\ n\ (g\ t) = y\} \text{ then } ?D\ x \text{ else } 0) \text{integrable\_on } S$ 
          proof (rule measurable_bounded_by_integrable_imp_integrable)
            have  $(\lambda x. ?D\ x) \in \text{borel\_measurable } (\text{lebesgue\_on } (\{t. h\ n\ (g\ t) = y\} \cap S))$ 
              proof -
                have  $(\lambda v. \det (\text{matrix } (g\ 'v))) \in \text{borel\_measurable } (\text{lebesgue\_on } (S \cap \{v. h\ n\ (g\ v) = y\}))$ 
                  by (metis Int_lower1 S assms(4) borel_measurable_det_Jacobian measurable_restrict_mono)
                then show ?thesis
                  by (simp add: Int_commute)
              qed
            then have  $(\lambda x. \text{if } x \in \{t. h\ n\ (g\ t) = y\} \cap S \text{ then } ?D\ x \text{ else } 0) \in$ 

```

```

borel_measurable lebesgue
  by (rule borel_measurable_if_I [OF h_lmeas])
  then show  $(\lambda x. \text{if } x \in \{t. h\ n\ (g\ t) = y\} \text{ then } ?D\ x \text{ else } 0) \in \text{borel\_measurable}$ 
    (lebesgue_on S)
    by (simp add: if_if_eq_conj Int_commute borel_measurable_if [OF S,
symmetric])
  show  $(\lambda x. ?D\ x *_{\mathbb{R}} f\ (g\ x) /_{\mathbb{R}} y) \text{ integrable\_on } S$ 
    by (rule integrable_cmul) (use det_int_fg in auto)
  show  $\text{norm } (\text{if } x \in \{t. h\ n\ (g\ t) = y\} \text{ then } ?D\ x \text{ else } 0) \leq ?D\ x *_{\mathbb{R}} f\ (g\ x)$ 
    /_{\mathbb{R}} y
    if  $x \in S$  for  $x$ 
    using nonneg_h [of n x]  $\langle y > 0 \rangle$  nonneg_fg [of x] h_le_f [of x n] that
    by (auto simp: divide_simps mult_left_mono)
  qed (use S in auto)
  then have int_det:  $(\lambda t. |\det (\text{matrix } (g' t))|) \text{ integrable\_on } (\{t. h\ n\ (g\ t) =$ 
 $y\} \cap S)$ 
    using integrable_restrict_Int by force
  have  $g^{-1} (\{t. h\ n\ (g\ t) = y\} \cap S) \in \text{lmeasurable}$ 
  by (blast intro: has_derivative_subset [OF der_g] measurable_differentiable_image
[OF h_lmeas] int_det)
  moreover have  $g^{-1} (\{t. h\ n\ (g\ t) = y\} \cap S) = \{x. h\ n\ x = y\} \cap g^{-1} S$ 
    by blast
  moreover have  $\text{measure lebesgue } (g^{-1} (\{t. h\ n\ (g\ t) = y\} \cap S))$ 
 $\leq \text{integral } (\{t. h\ n\ (g\ t) = y\} \cap S) (\lambda t. |\det (\text{matrix } (g' t))|)$ 
  by (blast intro: has_derivative_subset [OF der_g] measure_differentiable_image
[OF h_lmeas] int_det)
  ultimately show ?thesis
    using  $\langle y > 0 \rangle$  integral_restrict_Int [of S  $\{t. h\ n\ (g\ t) = y\}$   $\lambda t. |\det (\text{matrix}$ 
 $(g' t))| * y]$ 
    apply (simp add: integrable_on_indicator integral_indicator)
    apply (simp add: indicator_def of_bool_def if_distrib cong: if_cong)
    done
  qed
  show ?thesis
  proof
  show  $h\ n \text{ integrable\_on } g^{-1} S$ 
    apply (subst hn_eq)
    using yind by (force intro: integrable_sum [OF fin_R])
  have ?lhs =  $\text{integral } (g^{-1} S) (\lambda x. \sum_{y \in \text{range } (h\ n)}. y * \text{indicat\_real } \{x. h\ n$ 
 $x = y\} x)$ 
    by (metis hn_eq)
  also have  $\dots = (\sum_{y \in \text{range } (h\ n)}. \text{integral } (g^{-1} S) (\lambda x. y * \text{indicat\_real } \{x.$ 
 $h\ n\ x = y\} x))$ 
    by (rule integral_sum [OF fin_R]) (use yind in blast)
  also have  $\dots \leq (\sum_{y \in \text{range } (h\ n)}. \text{integral } S (\lambda u. |\det (\text{matrix } (g' u))| * y$ 
 $* \text{indicat\_real } \{x. h\ n\ x = y\} (g\ u)))$ 
    using yind by (force intro: sum_mono)
  also have  $\dots = \text{integral } S (\lambda u. \sum_{y \in \text{range } (h\ n)}. |\det (\text{matrix } (g' u))| * y *$ 
 $\text{indicat\_real } \{x. h\ n\ x = y\} (g\ u))$ 

```

```

proof (rule integral_sum [OF fin_R, symmetric])
  fix y assume y: y ∈ ?R
  with nonneg_h have y ≥ 0
  by auto
  show (λu. |det (matrix (g' u))| * y * indicat_real {x. h n x = y} (g u))
integrable_on S
  proof (rule measurable_bounded_by_integrable_imp_integrable)
    have (λx. indicat_real {x. h n x = y} (g x)) ∈ borel_measurable
(lebesgue_on S)
    using h_lmeas S
    by (auto simp: indicator_vimage [symmetric] borel_measurable_indicator_iff
sets_restrict_space_iff)
    then show (λu. |det (matrix (g' u))| * y * indicat_real {x. h n x = y} (g
u)) ∈ borel_measurable (lebesgue_on S)
    by (intro borel_measurable_times borel_measurable_abs borel_measurable_const
borel_measurable_det_Jacobian [OF S der_g])
  next
    fix x
    assume x ∈ S
    then have y * indicat_real {x. h n x = y} (g x) ≤ f (g x)
      by (metis (full_types) h_le_f indicator_simps mem_Collect_eq
mult.right_neutral_mult_zero_right nonneg_fg)
    with ⟨y ≥ 0⟩ show norm (?D x * y * indicat_real {x. h n x = y} (g x))
≤ ?D x * f(g x)
    by (simp add: abs_mult mult.assoc mult_left_mono)
    qed (use S det_int_fg in auto)
  qed
  also have ... = integral S (λT. |det (matrix (g' T))| *
(∑ y∈range (h n). y * indicat_real {x. h n x = y}
(g T)))
    by (simp add: sum_distrib_left mult.assoc)
  also have ... = ?rhs
    by (metis hn_eq)
  finally show integral (g ' S) (h n) ≤ ?rhs .
  qed
qed
have le: integral S (λT. |det (matrix (g' T))| * h n (g T)) ≤ ?b for n
proof (rule integral_le)
  show (λT. |det (matrix (g' T))| * h n (g T)) integrable_on S
  proof (rule measurable_bounded_by_integrable_imp_integrable)
  have (λT. |det (matrix (g' T))| *R h n (g T)) ∈ borel_measurable (lebesgue_on
S)
  proof (intro borel_measurable_scaleR borel_measurable_abs borel_measurable_det_Jacobian
⟨S ∈ sets lebesgue⟩)
    have eq: {x ∈ S. f x ≤ a} = (∪ b ∈ (f ' S) ∩ atMost a. {x. f x = b} ∩ S)
for f and a::real
    by auto
    have finite ((λx. h n (g x)) ' S ∩ {...a}) for a
    by (force intro: finite_subset [OF _ fin_R])

```

```

with h_lmeas [of n] show  $(\lambda x. h\ n\ (g\ x)) \in \text{borel\_measurable}\ (\text{lebesgue\_on}\ S)$ 
  apply (simp add: borel_measurable_vimage_halfspace_component_le  $\langle S \in \text{sets\ lebesgue}\ \rangle$  sets_restrict_space_iff eq)
  by (metis (mono_tags) SUP_inf sets.finite_UN)
qed (use der_g in blast)
then show  $(\lambda T. |\det (\text{matrix}\ (g'\ T))| * h\ n\ (g\ T)) \in \text{borel\_measurable}\ (\text{lebesgue\_on}\ S)$ 
  by simp
show norm  $(?D\ x * h\ n\ (g\ x)) \leq ?D\ x *_{\mathbb{R}} f\ (g\ x)$ 
  if  $x \in S$  for  $x$ 
  by (simp add: h_le_f_mult_left_mono nonneg_h that)
qed (use S det_int_fg in auto)
show  $?D\ x * h\ n\ (g\ x) \leq ?D\ x * f\ (g\ x)$  if  $x \in S$  for  $x$ 
  by (simp add:  $\langle x \in S \rangle$  h_le_f_mult_left_mono)
show  $(\lambda x. ?D\ x * f\ (g\ x)) \text{integrable\_on}\ S$ 
  using det_int_fg by blast
qed
have  $f \text{integrable\_on}\ g\ 'S \wedge (\lambda k. \text{integral}\ (g\ 'S)\ (h\ k)) \longrightarrow \text{integral}\ (g\ 'S)\ f$ 
proof (rule monotone_convergence_increasing)
  have  $|\text{integral}\ (g\ 'S)\ (h\ n)| \leq \text{integral}\ S\ (\lambda x. ?D\ x * f\ (g\ x))$  for  $n$ 
  proof -
    have  $|\text{integral}\ (g\ 'S)\ (h\ n)| = \text{integral}\ (g\ 'S)\ (h\ n)$ 
      using hint by (simp add: integral_nonneg nonneg_h)
    also have  $\dots \leq \text{integral}\ S\ (\lambda x. ?D\ x * f\ (g\ x))$ 
      using hint le by (meson order_trans)
    finally show ?thesis .
  qed
  then show bounded (range  $(\lambda k. \text{integral}\ (g\ 'S)\ (h\ k))$ )
    by (force simp: bounded_iff)
  qed (use h_inc_lim hint in auto)
moreover have  $\text{integral}\ (g\ 'S)\ (h\ n) \leq \text{integral}\ S\ (\lambda x. ?D\ x * f\ (g\ x))$  for  $n$ 
  using hint by (blast intro: le order_trans)
ultimately show ?thesis
  by (auto intro: Lim_bounded)
qed

```

lemma *integral_on_image_ubound_nonneg*:

```

fixes  $f :: \text{real}^n :: \{\text{finite, wellorder}\} \Rightarrow \text{real}$ 
assumes nonneg_fg:  $\bigwedge x. x \in S \implies 0 \leq f(g\ x)$ 
  and der_g:  $\bigwedge x. x \in S \implies (g \text{ has\_derivative}\ g'\ x)$  (at  $x$  within  $S$ )
  and intS:  $(\lambda x. |\det (\text{matrix}\ (g'\ x))| * f(g\ x)) \text{integrable\_on}\ S$ 
shows  $f \text{integrable\_on}\ (g\ 'S) \wedge \text{integral}\ (g\ 'S)\ f \leq \text{integral}\ S\ (\lambda x. |\det (\text{matrix}\ (g'\ x))| * f(g\ x))$ 
  (is  $\_ \wedge \_ \leq ?b$ )
proof -
  let  $?D = \lambda x. \det (\text{matrix}\ (g'\ x))$ 
  define  $S'$  where  $S' \equiv \{x \in S. ?D\ x * f(g\ x) \neq 0\}$ 

```

```

then have  $der\_gS'$ :  $\bigwedge x. x \in S' \implies (g \text{ has\_derivative } g' x)$  (at  $x$  within  $S'$ )
  by (metis (mono_tags, lifting)  $der\_g \text{ has\_derivative\_subset}$  mem_Collect_eq
subset_iff)
have  $(\lambda x. \text{if } x \in S \text{ then } |\?D x| * f(g x) \text{ else } 0)$  integrable_on UNIV
  by (simp add: integrable_restrict_UNIV intS)
then have  $Df\_borel$ :  $(\lambda x. \text{if } x \in S \text{ then } |\?D x| * f(g x) \text{ else } 0) \in \text{borel\_measurable}$ 
lebesgue
  using integrable_imp_measurable lebesgue_on_UNIV_eq by force
have  $S'$ :  $S' \in \text{sets lebesgue}$ 
proof -
from  $Df\_borel$  borel_measurable_vimage_open [of UNIV]
have  $\{x. (\text{if } x \in S \text{ then } |\?D x| * f(g x) \text{ else } 0) \in T\} \in \text{sets lebesgue}$ 
if open  $T$  for  $T$ 
  using that unfolding lebesgue_on_UNIV_eq
  by (fastforce simp add: dest!: spec)
then have  $\{x. (\text{if } x \in S \text{ then } |\?D x| * f(g x) \text{ else } 0) \in -\{0\}\} \in \text{sets lebesgue}$ 
using open_Comp by blast
then show ?thesis
  by (simp add:  $S'\_def$  conj_ac split: if_split_asm cong: conj_cong)
qed
then have  $gS'$ :  $g' S' \in \text{sets lebesgue}$ 
proof (rule differentiable_image_in_sets_lebesgue)
  show  $g$  differentiable_on  $S'$ 
    using  $der\_g$  unfolding  $S'\_def$  differentiable_def differentiable_on_def
    by (blast intro: has_derivative_subset)
qed auto
have  $f$ :  $f \in \text{borel\_measurable (lebesgue\_on (} g' S'))$ 
proof (clarsimp simp add: borel_measurable_vimage_open)
  fix  $T :: \text{real set}$ 
  assume open  $T$ 
  have  $\{x \in g' S'. f x \in T\} = g' \{x \in S'. f(g x) \in T\}$ 
    by blast
  moreover have  $g' \{x \in S'. f(g x) \in T\} \in \text{sets lebesgue}$ 
proof (rule differentiable_image_in_sets_lebesgue)
  let  $?h = \lambda x. |\?D x| * f(g x) /_R |\?D x|$ 
  have  $(\lambda x. \text{if } x \in S' \text{ then } |\?D x| * f(g x) \text{ else } 0) = (\lambda x. \text{if } x \in S \text{ then } |\?D x|$ 
 $* f(g x) \text{ else } 0)$ 
    by (auto simp:  $S'\_def$ )
  also have  $\dots \in \text{borel\_measurable lebesgue}$ 
    by (rule  $Df\_borel$ )
  finally have  $*$ :  $(\lambda x. |\?D x| * f(g x)) \in \text{borel\_measurable (lebesgue\_on } S')$ 
    by (simp add: borel_measurable_if_D)
  have  $(\lambda v. \det(\text{matrix}(g' v))) \in \text{borel\_measurable (lebesgue\_on } S')$ 
    using  $S'$  borel_measurable_det_Jacobian  $der\_gS'$  by blast
  then have  $?h \in \text{borel\_measurable (lebesgue\_on } S')$ 
using  $*$  borel_measurable_abs borel_measurable_inverse borel_measurable_scaleR
by blast
  moreover have  $?h x = f(g x)$  if  $x \in S'$  for  $x$ 
    using that by (auto simp:  $S'\_def$ )

```



```

ultimately have  $(\lambda x. f(g x)) \in \text{borel\_measurable } (\text{lebesgue\_on } S')$ 
  by (metis (no_types, lifting) measurable_lebesgue_cong)
then show  $\{x \in S'. f(g x) \in T\} \in \text{sets lebesgue}$ 
  by (simp add:  $\langle S' \in \text{sets lebesgue} \rangle \langle \text{open } T \rangle \text{borel\_measurable\_vimage\_open}$ 
sets_restrict_space_iff)
show  $g \text{ differentiable\_on } \{x \in S'. f(g x) \in T\}$ 
  using der_g unfolding S'_def differentiable_def differentiable_on_def
  by (blast intro: has_derivative_subset)
qed auto
ultimately have  $\{x \in g^{-1} S'. f x \in T\} \in \text{sets lebesgue}$ 
  by metis
then show  $\{x \in g^{-1} S'. f x \in T\} \in \text{sets } (\text{lebesgue\_on } (g^{-1} S'))$ 
  by (simp add:  $\langle g^{-1} S' \in \text{sets lebesgue} \rangle \text{sets\_restrict\_space\_iff}$ )
qed
have intS':  $(\lambda x. |\int D x| * f(g x)) \text{ integrable\_on } S'$ 
  using intS
  by (rule integrable_spike_set) (auto simp: S'_def intro: empty_imp_negligible)
have lebS':  $\{x \in S'. g x \in T\} \in \text{sets lebesgue}$  if  $T \subseteq g^{-1} S' T \in \text{sets lebesgue}$ 
for T
  proof -
  have  $g \in \text{borel\_measurable } (\text{lebesgue\_on } S')$ 
    using der_gS' has_derivative_continuous_on S'
    by (blast intro: continuous_imp_measurable_on_sets_lebesgue)
  moreover have  $\{x \in S'. g x \in U\} \in \text{sets lebesgue}$  if negligible  $U U \subseteq g^{-1} S'$ 
for U
  proof (intro negligible_imp_sets negligible_differentiable_vimage that)
  fix x
  assume  $x: x \in S'$ 
  then have linear  $(g' x)$ 
    using der_gS' has_derivative_linear by blast
  with x show inj  $(g' x)$ 
    by (auto simp: S'_def det_nz_iff_inj)
  qed (use der_gS' in auto)
  ultimately show ?thesis
    using double_lebesgue_sets [OF S' gS' order_refl] that by blast
  qed
  have int_gS':  $f \text{ integrable\_on } g^{-1} S' \wedge \text{integral } (g^{-1} S') f \leq \text{integral } S' (\lambda x. |\int D$ 
 $x| * f(g x))$ 
    using integral_on_image_ubound_weak [OF S' f nonneg_fg der_gS' intS'
lebS'] S'_def by blast
  have negligible  $(g^{-1} \{x \in S. \det(\text{matrix}(g' x)) = 0\})$ 
  proof (rule baby_Sard, simp_all)
  fix x
  assume  $x: x \in S \wedge \det(\text{matrix}(g' x)) = 0$ 
  then show  $(g \text{ has\_derivative } g' x)$  (at  $x$  within  $\{x \in S. \det(\text{matrix}(g' x)) =$ 
 $0\}$ )
    by (metis (no_types, lifting) der_g has_derivative_subset mem_Collect_eq
subsetI)
  then show  $\text{rank } (\text{matrix } (g' x)) < \text{CARD}('n)$ 

```

```

    using det_nz_iff_inj_matrix_vector_mul_linear x
    by (fastforce simp add: less_rank_noninjective)
  qed
  then have negg: negligible (g ' S - g ' {x ∈ S. ?D x ≠ 0})
    by (rule negligible_subset) (auto simp: S'_def)
  have null: g ' {x ∈ S. ?D x ≠ 0} - g ' S = {}
    by (auto simp: S'_def)
  let ?F = {x ∈ S. f (g x) ≠ 0}
  have eq: g ' S' = g ' ?F ∩ g ' {x ∈ S. ?D x ≠ 0}
    by (auto simp: S'_def image_iff)
  show ?thesis
  proof
    have ((λx. if x ∈ g ' ?F then f x else 0) integrable_on g ' {x ∈ S. ?D x ≠ 0})
      using int_gS' eq integrable_restrict_Int [where f=f]
      by simp
    then have f integrable_on g ' {x ∈ S. ?D x ≠ 0}
      by (auto simp: image_iff elim!: integrable_eq)
    then show f integrable_on g ' S
      using negg null
      by (auto intro: integrable_spike_set [OF _ empty_imp_negligible negligible_subset])
    have integral (g ' S) f = integral (g ' {x ∈ S. ?D x ≠ 0}) f
      using negg by (auto intro: negligible_subset integral_spike_set)
    also have ... = integral (g ' {x ∈ S. ?D x ≠ 0}) (λx. if x ∈ g ' ?F then f x
      else 0)
      by (auto simp: image_iff intro!: integral_cong)
    also have ... = integral (g ' S') f
      using eq integral_restrict_Int by simp
    also have ... ≤ integral S' (λx. |?D x| * f(g x))
      by (metis int_gS')
    also have ... ≤ ?b
      by (rule integral_subset_le [OF _ intS' intS]) (use nonneg_fg S'_def in auto)
    finally show integral (g ' S) f ≤ ?b .
  qed
  qed

```

lemma *absolutely_integrable_on_image_real*:

fixes $f :: \text{real}^n :: \{\text{finite, wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n :: _ \Rightarrow \text{real}^n :: _$
assumes $\text{der}_g: \bigwedge x. x \in S \implies (g \text{ has_derivative } g' x)$ (at x within S)
and $\text{intS}: (\lambda x. |\det (\text{matrix } (g' x))| * f(g x))$ *absolutely_integrable_on* S
shows f *absolutely_integrable_on* $(g ' S)$

proof –

```

let ?D = λx. |det (matrix (g' x))| * f (g x)
let ?N = {x ∈ S. f (g x) < 0} and ?P = {x ∈ S. f (g x) > 0}
have eq: {x. (if x ∈ S then ?D x else 0) > 0} = {x ∈ S. ?D x > 0}
      {x. (if x ∈ S then ?D x else 0) < 0} = {x ∈ S. ?D x < 0}
  by auto

```

```

have ?D integrable_on S
  using intS absolutely_integrable_on_def by blast
then have ( $\lambda x. \text{if } x \in S \text{ then } ?D x \text{ else } 0$ ) integrable_on UNIV
  by (simp add: integrable_restrict_UNIV)
then have D_borel: ( $\lambda x. \text{if } x \in S \text{ then } ?D x \text{ else } 0$ )  $\in$  borel_measurable (lebesgue_on UNIV)
  using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
then have Dlt:  $\{x \in S. ?D x < 0\} \in$  sets lebesgue
  unfolding borel_measurable_vimage_halfspace_component_lt
  by (drule_tac x=0 in spec) (auto simp: eq)
from D_borel have Dgt:  $\{x \in S. ?D x > 0\} \in$  sets lebesgue
  unfolding borel_measurable_vimage_halfspace_component_gt
  by (drule_tac x=0 in spec) (auto simp: eq)

have dfgbm: ?D  $\in$  borel_measurable (lebesgue_on S)
  using intS absolutely_integrable_on_def integrable_imp_measurable by blast
have der_gN: ( $g$  has_derivative  $g' x$ ) (at  $x$  within ?N) if  $x \in ?N$  for  $x$ 
  using der_g has_derivative_subset that by force
have ( $\lambda x. - f x$ ) integrable_on  $g^{-1} ?N \wedge$ 
  integral ( $g^{-1} ?N$ ) ( $\lambda x. - f x$ )  $\leq$  integral ?N ( $\lambda x. |\det (\text{matrix } (g' x))| * - f$ 
( $g x$ ))
proof (rule integral_on_image_ubound_nonneg [OF _ der_gN])
  have 1: ?D integrable_on  $\{x \in S. ?D x < 0\}$ 
    using Dlt
  by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
  have uminus  $\circ (\lambda x. |\det (\text{matrix } (g' x))| * - f (g x))$  integrable_on ?N
  by (simp add: o_def mult_less_0_iff empty_imp_negligible integrable_spike_set
[OF 1])
  then show ( $\lambda x. |\det (\text{matrix } (g' x))| * - f (g x)$ ) integrable_on ?N
    by (simp add: integrable_neg_iff o_def)
qed auto
then have  $f$  integrable_on  $g^{-1} ?N$ 
  by (simp add: integrable_neg_iff)
moreover have  $g^{-1} ?N = \{y \in g^{-1} S. f y < 0\}$ 
  by auto
ultimately have  $f$  integrable_on  $\{y \in g^{-1} S. f y < 0\}$ 
  by simp
then have N:  $f$  absolutely_integrable_on  $\{y \in g^{-1} S. f y < 0\}$ 
  by (rule absolutely_integrable_absolutely_integrable_ubound) auto

have der_gP: ( $g$  has_derivative  $g' x$ ) (at  $x$  within ?P) if  $x \in ?P$  for  $x$ 
  using der_g has_derivative_subset that by force
have  $f$  integrable_on  $g^{-1} ?P \wedge$  integral ( $g^{-1} ?P$ )  $f \leq$  integral ?P ?D
proof (rule integral_on_image_ubound_nonneg [OF _ der_gP])
  show ?D integrable_on ?P
  proof (rule integrable_spike_set)
    show ?D integrable_on  $\{x \in S. 0 < ?D x\}$ 
      using Dgt

```

```

    by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
    qed (auto simp: zero_less_mult_iff empty_imp_negligible)
  qed auto
  then have f_integrable_on g ' ?P
    by metis
  moreover have g ' ?P = {y ∈ g ' S. f y > 0}
    by auto
  ultimately have f_integrable_on {y ∈ g ' S. f y > 0}
    by simp
  then have P: f absolutely_integrable_on {y ∈ g ' S. f y > 0}
    by (rule absolutely_integrable_absolutely_integrable_lbound) auto
  have (λx. if x ∈ g ' S ∧ f x < 0 ∨ x ∈ g ' S ∧ 0 < f x then f x else 0) = (λx.
if x ∈ g ' S then f x else 0)
    by auto
  then show ?thesis
    using absolutely_integrable_Un [OF N P] absolutely_integrable_restrict_UNIV
[symmetric, where f=f]
    by simp
  qed

```

proposition *absolutely_integrable_on_image*:

```

  fixes f :: real^m::{finite,wellorder} ⇒ real^n and g :: real^m::_ ⇒ real^n::_
  assumes der_g: ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
    and intS: (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S
  shows f absolutely_integrable_on (g ' S)
  apply (rule absolutely_integrable_componentwise [OF absolutely_integrable_on_image_real
[OF der_g]])
  using absolutely_integrable_component [OF intS] by auto

```

proposition *integral_on_image_ubound*:

```

  fixes f :: real^n::{finite,wellorder} ⇒ real and g :: real^n::_ ⇒ real^n::_
  assumes ∧x. x ∈ S ⇒ 0 ≤ f(g x)
    and ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
    and (λx. |det (matrix (g' x))| * f(g x)) integrable_on S
  shows integral (g ' S) f ≤ integral S (λx. |det (matrix (g' x))| * f(g x))
  using integral_on_image_ubound_nonneg [OF assms] by simp

```

9.32.5 Change-of-variables theorem

The classic change-of-variables theorem. We have two versions with quite general hypotheses, the first that the transforming function has a continuous inverse, the second that the base set is Lebesgue measurable.

lemma *cov_invertible_nonneg_le*:

```

  fixes f :: real^n::{finite,wellorder} ⇒ real and g :: real^n::_ ⇒ real^n::_
  assumes der_g: ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
    and der_h: ∧y. y ∈ T ⇒ (h has_derivative h' y) (at y within T)
    and f0: ∧y. y ∈ T ⇒ 0 ≤ f y

```

```

  and hg:  $\bigwedge x. x \in S \implies g x \in T \wedge h(g x) = x$ 
  and gh:  $\bigwedge y. y \in T \implies h y \in S \wedge g(h y) = y$ 
  and id:  $\bigwedge y. y \in T \implies h' y \circ g'(h y) = id$ 
shows  $f \text{ integrable\_on } T \wedge (\text{integral } T f) \leq b \longleftrightarrow$ 
       $(\lambda x. |\det (\text{matrix } (g' x))| * f(g x)) \text{ integrable\_on } S \wedge$ 
       $\text{integral } S (\lambda x. |\det (\text{matrix } (g' x))| * f(g x)) \leq b$ 
  (is ?lhs = ?rhs)
proof -
  have Teq:  $T = g'S$  and Seq:  $S = h'T$ 
  using hg gh image_iff by fastforce+
  have gS:  $g \text{ differentiable\_on } S$ 
  by (meson der_g differentiable_def differentiable_on_def)
  let ?D =  $\lambda x. |\det (\text{matrix } (g' x))| * f(g x)$ 
  show ?thesis
  proof
    assume ?lhs
    then have fT:  $f \text{ integrable\_on } T$  and intf:  $\text{integral } T f \leq b$ 
    by blast+
    show ?rhs
    proof
      let ?fgh =  $\lambda x. |\det (\text{matrix } (h' x))| * (|\det (\text{matrix } (g' (h x)))| * f(g(h x)))$ 
      have ddf:  $?fgh x = f x$ 
      if  $x \in T$  for  $x$ 
      proof -
        have  $\text{matrix } (h' x) ** \text{matrix } (g' (h x)) = \text{mat } 1$ 
        by (metis der_g der_h gh has_derivative_linear local.id matrix_compose
matrix_id_mat_1 that)
        then have  $|\det (\text{matrix } (h' x))| * |\det (\text{matrix } (g' (h x)))| = 1$ 
        by (metis abs_1 abs_mult det_I det_mul)
        then show ?thesis
        by (simp add: gh that)
      qed
      have ?D integrable_on (h'T)
      proof (intro set_lebesgue_integral_eq_integral absolutely_integrable_on_image_real)
        show  $(\lambda x. ?fgh x) \text{ absolutely\_integrable\_on } T$ 
        by (smt (verit, del_insts) abs_absolutely_integrableI_1 ddf f0 fT integrable_eq)
      qed (use der_h in auto)
      with Seq show  $(\lambda x. ?D x) \text{ integrable\_on } S$ 
      by simp
      have  $\text{integral } S (\lambda x. ?D x) \leq \text{integral } T (\lambda x. ?fgh x)$ 
      unfolding Seq
      proof (rule integral_on_image_ubound)
        show  $(\lambda x. ?fgh x) \text{ integrable\_on } T$ 
        using ddf fT integrable_eq by force
      qed (use f0 gh der_h in auto)
      also have  $\dots = \text{integral } T f$ 
      by (force simp: ddf intro: integral_cong)
      finally show  $\text{integral } S (\lambda x. ?D x) \leq b$ 
    qed
  qed

```

```

      using intf by linarith
    qed
  next
    assume R: ?rhs
    then have f integrable_on g ' S
      using der_g f0 hg integral_on_image_ubound_nonneg by blast
    moreover have integral (g ' S) f ≤ integral S (λx. ?D x)
      by (rule integral_on_image_ubound [OF f0 der_g]) (use R Teq in auto)
    ultimately show ?lhs
      using R by (simp add: Teq)
    qed
  qed

```

lemma cov_invertible_nonneg_eq:
fixes $f :: \text{real}^n::\{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n::_ \Rightarrow \text{real}^n::_$
assumes $\bigwedge x. x \in S \Longrightarrow (g \text{ has_derivative } g' x)$ (at x within S)
and $\bigwedge y. y \in T \Longrightarrow (h \text{ has_derivative } h' y)$ (at y within T)
and $\bigwedge y. y \in T \Longrightarrow 0 \leq f y$
and $\bigwedge x. x \in S \Longrightarrow g x \in T \wedge h(g x) = x$
and $\bigwedge y. y \in T \Longrightarrow h y \in S \wedge g(h y) = y$
and $\bigwedge y. y \in T \Longrightarrow h' y \circ g'(h y) = \text{id}$
shows $((\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) \text{ has_integral } b) S \longleftrightarrow (f \text{ has_integral } b) T$
using cov_invertible_nonneg_le [OF assms]
by (simp add: has_integral_iff) (meson eq_iff)

lemma cov_invertible_real:
fixes $f :: \text{real}^n::\{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}$ **and** $g :: \text{real}^n::_ \Rightarrow \text{real}^n::_$
assumes $\text{der}_g: \bigwedge x. x \in S \Longrightarrow (g \text{ has_derivative } g' x)$ (at x within S)
and $\text{der}_h: \bigwedge y. y \in T \Longrightarrow (h \text{ has_derivative } h' y)$ (at y within T)
and $hg: \bigwedge x. x \in S \Longrightarrow g x \in T \wedge h(g x) = x$
and $gh: \bigwedge y. y \in T \Longrightarrow h y \in S \wedge g(h y) = y$
and $\text{id}: \bigwedge y. y \in T \Longrightarrow h' y \circ g'(h y) = \text{id}$
shows $(\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |\det(\text{matrix}(g' x))| * f(g x)) = b \longleftrightarrow$
 $f \text{ absolutely_integrable_on } T \wedge \text{integral } T f = b$
(is ?lhs = ?rhs)

proof –
have $\text{Teq}: T = g'S$ **and** $\text{Seq}: S = h'T$
using hg gh image_iff **by** fastforce+
let $?DP = \lambda x. |\det(\text{matrix}(g' x))| * f(g x)$ **and** $?DN = \lambda x. |\det(\text{matrix}(g' x))| * -f(g x)$
have $+$: $(?DP \text{ has_integral } b) \{x \in S. f(g x) > 0\} \longleftrightarrow (f \text{ has_integral } b) \{y \in T. f y > 0\}$ **for** b
proof (rule cov_invertible_nonneg_eq)
have $*$: $(\lambda x. f(g x)) -' Y \cap \{x \in S. f(g x) > 0\}$
 $= ((\lambda x. f(g x)) -' Y \cap S) \cap \{x \in S. f(g x) > 0\}$ **for** Y

```

    by auto
    show (g has_derivative g' x) (at x within {x ∈ S. f (g x) > 0}) if x ∈ {x ∈ S.
f (g x) > 0} for x
      using that der_g has_derivative_subset by fastforce
    show (h has_derivative h' y) (at y within {y ∈ T. f y > 0}) if y ∈ {y ∈ T. f
y > 0} for y
      using that der_h has_derivative_subset by fastforce
  qed (use gh hg id in auto)
  have -: (?DN has_integral b) {x ∈ S. f (g x) < 0} ↔ ((λx. - f x) has_integral
b) {y ∈ T. f y < 0} for b
  proof (rule cov_invertible_nonneg_eq)
    have *: (λx. - f (g x)) -' y ∩ {x ∈ S. f (g x) < 0}
      = ((λx. f (g x)) -' uminus ' y ∩ S) ∩ {x ∈ S. f (g x) < 0} for y
    using image_iff by fastforce
    show (g has_derivative g' x) (at x within {x ∈ S. f (g x) < 0}) if x ∈ {x ∈ S.
f (g x) < 0} for x
      using that der_g has_derivative_subset by fastforce
    show (h has_derivative h' y) (at y within {y ∈ T. f y < 0}) if y ∈ {y ∈ T. f
y < 0} for y
      using that der_h has_derivative_subset by fastforce
  qed (use gh hg id in auto)
  show ?thesis
  proof
    assume LHS: ?lhs
    have eq: {x. (if x ∈ S then ?DP x else 0) > 0} = {x ∈ S. ?DP x > 0}
      {x. (if x ∈ S then ?DP x else 0) < 0} = {x ∈ S. ?DP x < 0}
    by auto
    have ?DP integrable_on S
      using LHS absolutely_integrable_on_def by blast
    then have (λx. if x ∈ S then ?DP x else 0) integrable_on UNIV
      by (simp add: integrable_restrict_UNIV)
    then have D_borel: (λx. if x ∈ S then ?DP x else 0) ∈ borel_measurable
(lebesgue_on UNIV)
      using integrable_imp_measurable_lebesgue_on_UNIV_eq by blast
    then have SN: {x ∈ S. ?DP x < 0} ∈ sets lebesgue
      unfolding borel_measurable_vimage_halfspace_component_lt
      by (drule_tac x=0 in spec) (auto simp: eq)
    from D_borel have SP: {x ∈ S. ?DP x > 0} ∈ sets lebesgue
      unfolding borel_measurable_vimage_halfspace_component_gt
      by (drule_tac x=0 in spec) (auto simp: eq)
    have ?DP absolutely_integrable_on {x ∈ S. ?DP x > 0}
      using LHS by (fast intro!: set_integrable_subset [OF _, of _ S] SP)
    then have aP: ?DP absolutely_integrable_on {x ∈ S. f (g x) > 0}
      by (rule absolutely_integrable_spike_set) (auto simp: zero_less_mult_iff
empty_imp_negligible)
    have ?DP absolutely_integrable_on {x ∈ S. ?DP x < 0}
      using LHS by (fast intro!: set_integrable_subset [OF _, of _ S] SN)
    then have aN: ?DP absolutely_integrable_on {x ∈ S. f (g x) < 0}
      by (rule absolutely_integrable_spike_set) (auto simp: mult_less_0_iff empty_imp_negligible)
  end

```

```

have fN: f integrable_on {y ∈ T. f y < 0}
  integral {y ∈ T. f y < 0} f = integral {x ∈ S. f (g x) < 0} ?DP
  using - [of integral {x ∈ S. f(g x) < 0} ?DN] aN
by (auto simp: set_lebesgue_integral_eq_integral has_integral_iff integrable_neg_iff)
have faN: f absolutely_integrable_on {y ∈ T. f y < 0}
proof (rule absolutely_integrable_integrable_bound)
  show (λx. - f x) integrable_on {y ∈ T. f y < 0}
    using fN by (auto simp: integrable_neg_iff)
qed (use fN in auto)
have fP: f integrable_on {y ∈ T. f y > 0}
  integral {y ∈ T. f y > 0} f = integral {x ∈ S. f (g x) > 0} ?DP
  using + [of integral {x ∈ S. f(g x) > 0} ?DP] aP
by (auto simp: set_lebesgue_integral_eq_integral has_integral_iff integrable_neg_iff)
have faP: f absolutely_integrable_on {y ∈ T. f y > 0}
  using fP(1) nonnegative_absolutely_integrable_1 by fastforce
have fa: f absolutely_integrable_on ({y ∈ T. f y < 0} ∪ {y ∈ T. f y > 0})
  by (rule absolutely_integrable_Un [OF faN faP])
show ?rhs
proof
  have eq: ((if x ∈ T ∧ f x < 0 ∨ x ∈ T ∧ 0 < f x then 1 else 0) * f x)
    = (if x ∈ T then 1 else 0) * f x for x
    by auto
  show f absolutely_integrable_on T
    using fa by (simp add: indicator_def of_bool_def set_integrable_def eq)
  have [simp]: {y ∈ T. f y < 0} ∩ {y ∈ T. 0 < f y} = {} for T and f ::
(realn::_) ⇒ real
    by auto
  have integral T f = integral ({y ∈ T. f y < 0} ∪ {y ∈ T. f y > 0}) f
    by (intro empty_imp_negligible integral_spike_set) (auto simp: eq)
  also have ... = integral {y ∈ T. f y < 0} f + integral {y ∈ T. f y > 0} f
    using fN fP by simp
  also have ... = integral {x ∈ S. f (g x) < 0} ?DP + integral {x ∈ S. 0 <
f (g x)} ?DP
    by (simp add: fN fP)
  also have ... = integral ({x ∈ S. f (g x) < 0} ∪ {x ∈ S. 0 < f (g x)}) ?DP
    using aP aN by (simp add: set_lebesgue_integral_eq_integral)
  also have ... = integral S ?DP
    by (intro empty_imp_negligible integral_spike_set) auto
  also have ... = b
    using LHS by simp
  finally show integral T f = b .
qed
next
assume RHS: ?rhs
have eq: {x. (if x ∈ T then f x else 0) > 0} = {x ∈ T. f x > 0}
  {x. (if x ∈ T then f x else 0) < 0} = {x ∈ T. f x < 0}
  by auto
have f integrable_on T
  using RHS absolutely_integrable_on_def by blast

```



```

then have ( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ ) integrable_on UNIV
  by (simp add: integrable_restrict_UNIV)
then have D_borel: ( $\lambda x. \text{if } x \in T \text{ then } f x \text{ else } 0$ )  $\in$  borel_measurable
(lebesgue_on UNIV)
  using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
then have TN:  $\{x \in T. f x < 0\} \in$  sets lebesgue
  unfolding borel_measurable_vimage_halfspace_component_lt
  by (drule_tac x=0 in spec) (auto simp: eq)
from D_borel have TP:  $\{x \in T. f x > 0\} \in$  sets lebesgue
  unfolding borel_measurable_vimage_halfspace_component_gt
  by (drule_tac x=0 in spec) (auto simp: eq)
have aint: f absolutely_integrable_on  $\{y. y \in T \wedge 0 < (f y)\}$ 
  f absolutely_integrable_on  $\{y. y \in T \wedge (f y) < 0\}$ 
  and intT: integral T f = b
  using set_integrable_subset [of _ T] TP TN RHS by blast+
show ?lhs
proof
  have fN: f integrable_on  $\{v \in T. f v < 0\}$ 
    using absolutely_integrable_on_def aint by blast
  then have DN: (?DN has_integral integral  $\{y \in T. f y < 0\}$ ) ( $\lambda x. - f x$ )  $\{x$ 
 $\in S. f (g x) < 0\}$ 
    using  $- [of \text{integral } \{y \in T. f y < 0\} (\lambda x. - f x)]$ 
    by (simp add: has_integral_neg_iff integrable_integral)
  have aDN: ?DP absolutely_integrable_on  $\{x \in S. f (g x) < 0\}$ 
    apply (rule absolutely_integrable_integrable_bound [where g = ?DN])
    using DN hg by (fastforce simp: abs_mult_integrable_neg_iff)+
  have fP: f integrable_on  $\{v \in T. f v > 0\}$ 
    using absolutely_integrable_on_def aint by blast
  then have DP: (?DP has_integral integral  $\{y \in T. f y > 0\}$ ) f  $\{x \in S. f (g$ 
 $x) > 0\}$ 
    using  $+ [of \text{integral } \{y \in T. f y > 0\} f]$ 
    by (simp add: has_integral_neg_iff integrable_integral)
  have aDP: ?DP absolutely_integrable_on  $\{x \in S. f (g x) > 0\}$ 
    apply (rule absolutely_integrable_integrable_bound [where g = ?DP])
    using DP hg by (fastforce simp: integrable_neg_iff)+
  have eq: (if  $x \in S$  then  $1$  else  $0$ )  $*$  ?DP x = (if  $x \in S \wedge f (g x) < 0 \vee x \in S$ 
 $\wedge f (g x) > 0$  then  $1$  else  $0$ )  $*$  ?DP x for  $x$ 
    by force
  have ?DP absolutely_integrable_on  $(\{x \in S. f (g x) < 0\} \cup \{x \in S. f (g x)$ 
 $> 0\})$ 
    by (rule absolutely_integrable_Un [OF aDN aDP])
  then show I: ?DP absolutely_integrable_on S
    by (simp add: indicator_def of_bool_def eq set_integrable_def)
  have [simp]:  $\{y \in S. f y < 0\} \cap \{y \in S. 0 < f y\} = \{\}$  for  $S$  and  $f ::$ 
(realn.: _)  $\Rightarrow$  real
    by auto
  have integral S ?DP = integral  $(\{x \in S. f (g x) < 0\} \cup \{x \in S. f (g x) >$ 
 $0\})$  ?DP
    by (intro empty_imp_negligible integral_spike_set) auto

```

also have $\dots = \text{integral } \{x \in S. f (g x) < 0\} \text{ ?DP} + \text{integral } \{x \in S. 0 <$
 $f (g x)\} \text{ ?DP}$
using *aDN aDP* **by** (*simp add: set_lebesgue_integral_eq_integral*)
also have $\dots = - \text{integral } \{y \in T. f y < 0\} (\lambda x. - f x) + \text{integral } \{y \in T.$
 $f y > 0\} f$
using *DN DP* **by** (*auto simp: has_integral_iff*)
also have $\dots = \text{integral } (\{x \in T. f x < 0\} \cup \{x \in T. 0 < f x\}) f$
by (*simp add: fN fP*)
also have $\dots = \text{integral } T f$
by (*intro empty_imp_negligible_integral_spike_set*) *auto*
also have $\dots = b$
using *intT* **by** *simp*
finally show $\text{integral } S \text{ ?DP} = b .$
qed
qed
qed

lemma *cv_inv_version3*:

fixes $f :: \text{real}^m :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^m :: _$
assumes *der_g*: $\bigwedge x. x \in S \Rightarrow (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S)$
and *der_h*: $\bigwedge y. y \in T \Rightarrow (h \text{ has_derivative } h' y) \text{ (at } y \text{ within } T)$
and *hg*: $\bigwedge x. x \in S \Rightarrow g x \in T \wedge h(g x) = x$
and *gh*: $\bigwedge y. y \in T \Rightarrow h y \in S \wedge g(h y) = y$
and *id*: $\bigwedge y. y \in T \Rightarrow h' y \circ g'(h y) = \text{id}$
shows $(\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } T \wedge \text{integral } T f = b$

proof –

let $?D = \lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)$
have $((\lambda x. |\det (\text{matrix } (g' x))| * f(g x) \$ i) \text{ absolutely_integrable_on } S \wedge \text{integral}$
 $S (\lambda x. |\det (\text{matrix } (g' x))| * (f(g x) \$ i)) = b \$ i) \longleftrightarrow$
 $((\lambda x. f x \$ i) \text{ absolutely_integrable_on } T \wedge \text{integral } T (\lambda x. f x \$ i) = b \$$
 $i) \text{ for } i$
by (*rule cov_invertible_real [OF der_g der_h hg gh id]*)
then have $?D \text{ absolutely_integrable_on } S \wedge (?D \text{ has_integral } b) S \longleftrightarrow$
 $f \text{ absolutely_integrable_on } T \wedge (f \text{ has_integral } b) T$
unfolding *absolutely_integrable_componentwise_iff [where f=f] has_integral_componentwise_iff*
 $[of f]$
 $\text{absolutely_integrable_componentwise_iff [where f=?D] has_integral_componentwise_iff}$
 $[of ?D]$
by (*auto simp: all_conj_distrib Basis_vec_def cart_eq_inner_axis [symmetric]*)
 $\text{has_integral_iff_set_lebesgue_integral_eq_integral}$)
then show *?thesis*
using *absolutely_integrable_on_def* **by** *blast*
qed

lemma *cv_inv_version4*:

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^n :: _$
assumes $\text{der_}g: \bigwedge x. x \in S \Longrightarrow (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S) \wedge \text{invertible}(\text{matrix}(g' x))$
and $hg: \bigwedge x. x \in S \Longrightarrow \text{continuous_on } (g \text{ ' } S) h \wedge h(g x) = x$
shows $(\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$
proof –
have $\forall x. \exists h'. x \in S$
 $\longrightarrow (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S) \wedge \text{linear } h' \wedge g' x \circ h' = \text{id} \wedge$
 $h' \circ g' x = \text{id}$
using $\text{der_}g \text{ matrix_invertible has_derivative_linear by blast}$
then obtain h' **where** h' :
 $\bigwedge x. x \in S$
 $\Longrightarrow (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S) \wedge$
 $\text{linear } (h' x) \wedge g' x \circ (h' x) = \text{id} \wedge (h' x) \circ g' x = \text{id}$
by *metis*
show *?thesis*
proof (*rule cv_inv_version3*)
show $\bigwedge y. y \in g \text{ ' } S \Longrightarrow (h \text{ has_derivative } h' (h y)) \text{ (at } y \text{ within } g \text{ ' } S)$
using $h' hg$
by (*force simp: continuous_on_eq_continuous_within intro!: has_derivative_inverse_within*)
qed (*use h' hg in auto*)
qed

theorem *has_absolute_integral_change_of_variables_invertible:*

fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^n :: _$
assumes $\text{der_}g: \bigwedge x. x \in S \Longrightarrow (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S)$
and $hg: \bigwedge x. x \in S \Longrightarrow h(g x) = x$
and $\text{conth: continuous_on } (g \text{ ' } S) h$
shows $(\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) \text{ absolutely_integrable_on } S \wedge \text{integral}$
 $S (\lambda x. |\det(\text{matrix}(g' x))| *_R f(g x)) = b \longleftrightarrow$
 $f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$
(is ?lhs = ?rhs)
proof –
let $?S = \{x \in S. \text{invertible}(\text{matrix}(g' x))\}$ **and** $?D = \lambda x. |\det(\text{matrix}(g' x))|$
 $*_R f(g x)$
have $?: ?D \text{ absolutely_integrable_on } ?S \wedge \text{integral } ?S ?D = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } ?S) \wedge \text{integral } (g \text{ ' } ?S) f = b$
proof (*rule cv_inv_version4*)
show $(g \text{ has_derivative } g' x) \text{ (at } x \text{ within } ?S) \wedge \text{invertible}(\text{matrix}(g' x))$
if $x \in ?S$ **for** x
using $\text{der_}g \text{ that has_derivative_subset that by fastforce}$
show $\text{continuous_on } (g \text{ ' } ?S) h \wedge h(g x) = x$
if $x \in ?S$ **for** x
using $\text{that continuous_on_subset [OF conth] by (simp add: hg image_mono)}$
qed
have $(g \text{ has_derivative } g' x) \text{ (at } x \text{ within } \{x \in S. \text{rank}(\text{matrix}(g' x)) <$

CARD($'m$)} **if** $x \in S$ **for** x
by (*metis* (*no_types*, *lifting*) *der_g has_derivative_subset mem_Collect_eq subsetI that*)
then have *negligible* ($g \text{ ' } \{x \in S. \neg \text{invertible (matrix (g' x))}\}$)
by (*auto simp: invertible_det_nz det_eq_0_rank intro: baby_Sard*)
then have *neg: negligible* $\{x \in g \text{ ' } S. x \notin g \text{ ' } ?S \wedge f x \neq 0\}$
by (*auto intro: negligible_subset*)
have [*simp*]: $\{x \in g \text{ ' } ?S. x \notin g \text{ ' } S \wedge f x \neq 0\} = \{\}$
by *auto*
have $?D \text{ absolutely_integrable_on } ?S \wedge \text{integral } ?S ?D = b$
 $\longleftrightarrow ?D \text{ absolutely_integrable_on } S \wedge \text{integral } S ?D = b$
apply (*intro conj_cong absolutely_integrable_spike_set_eq*)
apply(*auto simp: integral_spike_set invertible_det_nz empty_imp_negligible*
neg)
done
moreover
have $f \text{ absolutely_integrable_on } (g \text{ ' } ?S) \wedge \text{integral } (g \text{ ' } ?S) f = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$
by (*auto intro!: conj_cong absolutely_integrable_spike_set_eq integral_spike_set*
neg)
ultimately
show *?thesis*
using * **by** *blast*
qed

theorem *has_absolute_integral_change_of_variables_compact*:
fixes $f :: \text{real}^m :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^m :: _$
assumes *compact S*
and *der_g*: $\bigwedge x. x \in S \implies (g \text{ has_derivative } g' x) \text{ (at } x \text{ within } S)$
and *inj*: *inj_on* $g \ S$
shows $((\lambda x. |\det (\text{matrix } (g' x))| *_{\mathbb{R}} f(g x)) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |\det (\text{matrix } (g' x))| *_{\mathbb{R}} f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b)$
proof –
obtain h **where** $hg: \bigwedge x. x \in S \implies h(g x) = x$
using *inj* **by** (*metis the_inv_into_f_f*)
have *conth*: *continuous_on* $(g \text{ ' } S) \ h$
by (*metis* $\langle \text{compact } S \rangle$ *continuous_on_inv der_g has_derivative_continuous_on*
hg)
show *?thesis*
by (*rule* *has_absolute_integral_change_of_variables_invertible* [*OF* *der_g hg*
conth])
qed

lemma *has_absolute_integral_change_of_variables_compact_family*:
fixes $f :: \text{real}^m :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^m :: _$

```

assumes compact:  $\bigwedge n::nat. compact (F n)$ 
and der_g:  $\bigwedge x. x \in (\bigcup n. F n) \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } (\bigcup n. F n))$ 
and inj: inj_on g  $(\bigcup n. F n)$ 
shows  $((\lambda x. |det (matrix (g' x))| *_{\mathbb{R}} f(g x)) \text{ absolutely\_integrable\_on } (\bigcup n. F n))$ 
 $\wedge$ 
 $integral (\bigcup n. F n) (\lambda x. |det (matrix (g' x))| *_{\mathbb{R}} f(g x)) = b$ 
 $\iff f \text{ absolutely\_integrable\_on } (g' (\bigcup n. F n)) \wedge integral (g' (\bigcup n. F n)) f = b$ 
proof -
let ?D =  $\lambda x. |det (matrix (g' x))| *_{\mathbb{R}} f (g x)$ 
let ?U =  $\lambda n. \bigcup m \leq n. F m$ 
let ?lift =  $vec::real \Rightarrow real^1$ 
have F_leb:  $F m \in sets \text{ lebesgue for } m$ 
by (simp add: compact borel_compact)
have iff:  $(\lambda x. |det (matrix (g' x))| *_{\mathbb{R}} f (g x)) \text{ absolutely\_integrable\_on } (?U n)$ 
 $\wedge$ 
 $integral (?U n) (\lambda x. |det (matrix (g' x))| *_{\mathbb{R}} f (g x)) = b$ 
 $\iff f \text{ absolutely\_integrable\_on } (g' (?U n)) \wedge integral (g' (?U n)) f = b$ 
for n b and f ::  $real^1 m::\_ \Rightarrow real^k$ 
proof (rule has_absolute_integral_change_of_variables_compact)
show compact (?U n)
by (simp add: compact_compact_UN)
show (g has_derivative g' x) (at x within (?U n))
if x  $\in ?U n$  for x
using that by (blast intro!: has_derivative_subset [OF der_g])
show inj_on g (?U n)
using inj by (auto simp: inj_on_def)
qed
show ?thesis
unfolding image_UN
proof safe
assume DS: ?D absolutely_integrable_on  $(\bigcup n. F n)$ 
and b:  $b = integral (\bigcup n. F n) ?D$ 
have DU:  $\bigwedge n. ?D \text{ absolutely\_integrable\_on } (?U n)$ 
 $(\lambda n. integral (?U n) ?D) \longrightarrow integral (\bigcup n. F n) ?D$ 
using integral_countable_UN [OF DS F_leb] by auto
with iff have fag: f absolutely_integrable_on  $g' (?U n)$ 
and fg_int:  $integral (\bigcup m \leq n. g' F m) f = integral (?U n) ?D$  for n
by (auto simp: image_UN)
let ?h =  $\lambda x. \text{if } x \in (\bigcup m. g' F m) \text{ then } norm(f x) \text{ else } 0$ 
have  $(\lambda x. \text{if } x \in (\bigcup m. g' F m) \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$ 
proof (rule dominated_convergence_absolutely_integrable)
show  $(\lambda x. \text{if } x \in (\bigcup m \leq k. g' F m) \text{ then } f x \text{ else } 0) \text{ absolutely\_integrable\_on UNIV}$  for k
unfolding absolutely_integrable_restrict_UNIV
using fag by (simp add: image_UN)
let ?nf =  $\lambda n x. \text{if } x \in (\bigcup m \leq n. g' F m) \text{ then } norm(f x) \text{ else } 0$ 
show ?h integrable_on UNIV

```

```

proof (rule monotone_convergence_increasing [THEN conjunct1])
  show  $?nf\ k$  integrable_on UNIV for  $k$ 
    using fag
    unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
(simp add: image_UN)
  { fix  $n$ 
    have  $(norm \circ ?D)$  absolutely_integrable_on  $?U\ n$ 
      by (intro absolutely_integrable_norm DU)
    then have  $integral\ (g\ ' \ ?U\ n)\ (norm \circ f) = integral\ (?U\ n)\ (norm \circ ?D)$ 
      using iff [of  $n\ vec \circ norm \circ f\ integral\ (?U\ n)\ (\lambda x. |det\ (matrix\ (g'\ x))|$ 
*_R (?lift  $\circ norm \circ f$ ) ( $g\ x$ ))]
      unfolding absolutely_integrable_on_1_iff integral_on_1_eq by (auto
simp: o_def)
    }
  moreover have bounded (range ( $\lambda k. integral\ (?U\ k)\ (norm \circ ?D)$ ))
    unfolding bounded_iff
  proof (rule exI, clarify)
    fix  $k$ 
    show  $norm\ (integral\ (?U\ k)\ (norm \circ ?D)) \leq integral\ (\bigcup n. F\ n)\ (norm \circ$ 
?D)
      unfolding integral_restrict_UNIV [of  $\_ norm \circ ?D, symmetric$ ]
    proof (rule integral_norm_bound_integral)
      show ( $\lambda x. if\ x \in \bigcup (F\ ' \ \{..k\})\ then\ (norm \circ ?D)\ x\ else\ 0$ ) integrable_on
UNIV
      ( $\lambda x. if\ x \in (\bigcup n. F\ n)\ then\ (norm \circ ?D)\ x\ else\ 0$ ) integrable_on UNIV
      using DU(1) DS
    unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
by auto
    qed auto
  qed
  ultimately show bounded (range ( $\lambda k. integral\ UNIV\ (?nf\ k)$ ))
    by (simp add: integrable_restrict_UNIV image_UN [symmetric] o_def)
  next
  show ( $\lambda k. if\ x \in (\bigcup m \leq k. g\ ' \ F\ m)\ then\ norm\ (f\ x)\ else\ 0$ )
     $\longrightarrow$  ( $if\ x \in (\bigcup m. g\ ' \ F\ m)\ then\ norm\ (f\ x)\ else\ 0$ ) for  $x$ 
    by (force intro: tendsto_eventually_eventually_sequentiallyI)
  qed auto
  next
  show ( $\lambda k. if\ x \in (\bigcup m \leq k. g\ ' \ F\ m)\ then\ f\ x\ else\ 0$ )
     $\longrightarrow$  ( $if\ x \in (\bigcup m. g\ ' \ F\ m)\ then\ f\ x\ else\ 0$ ) for  $x$ 
  proof clarsimp
    fix  $m\ y$ 
    assume  $y \in F\ m$ 
    show ( $\lambda k. if\ \exists x \in \{..k\}. g\ y \in g\ ' \ F\ x\ then\ f\ (g\ y)\ else\ 0$ )  $\longrightarrow$   $f\ (g\ y)$ 
    using  $\langle y \in F\ m \rangle$  by (force intro: tendsto_eventually_eventually_sequentiallyI
[of  $m$ ])
  qed
  qed auto
  then show fai:  $f$  absolutely_integrable_on  $(\bigcup m. g\ ' \ F\ m)$ 

```

```

    using absolutely_integrable_restrict_UNIV by blast
  show integral (( $\bigcup x. g \text{ ' } F x$ )) f = integral ( $\bigcup n. F n$ ) ?D
  proof (rule LIMSEQ_unique)
    show ( $\lambda n. \text{integral } (?U n) ?D$ )  $\longrightarrow$  integral ( $\bigcup x. g \text{ ' } F x$ ) f
      unfolding fg_int [symmetric]
    proof (rule integral_countable_UN [OF fai])
      show  $g \text{ ' } F m \in \text{sets lebesgue for } m$ 
      proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
        show  $g$  differentiable_on  $F m$ 
        by (meson der_g differentiableI UnionI differentiable_on_def differentiable_on_subset rangeI subsetI)
      qed auto
    qed
  qed (use DU in metis)
next
  assume fs:  $f$  absolutely_integrable_on ( $\bigcup x. g \text{ ' } F x$ )
  and b:  $b = \text{integral } ((\bigcup x. g \text{ ' } F x)) f$ 
  have gF_leb:  $g \text{ ' } F m \in \text{sets lebesgue for } m$ 
  proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
    show  $g$  differentiable_on  $F m$ 
    using der_g unfolding differentiable_def differentiable_on_def
    by (meson Sup_upper UNIV_I UnionI has_derivative_subset image_eqI)
  qed auto
  have fgU:  $\bigwedge n. f$  absolutely_integrable_on ( $\bigcup m \leq n. g \text{ ' } F m$ )
    ( $\lambda n. \text{integral } (\bigcup m \leq n. g \text{ ' } F m) f$ )  $\longrightarrow$  integral ( $\bigcup m. g \text{ ' } F m$ ) f
    using integral_countable_UN [OF fs gF_leb] by auto
  with iff have DUn: ?D absolutely_integrable_on ?U n
    and D_int: integral (?U n) ?D = integral ( $\bigcup m \leq n. g \text{ ' } F m$ ) f for n
    by (auto simp: image_UN)
  let ?h =  $\lambda x. \text{if } x \in (\bigcup n. F n) \text{ then norm } (?D x) \text{ else } 0$ 
  have ( $\lambda x. \text{if } x \in (\bigcup n. F n) \text{ then } ?D x \text{ else } 0$ ) absolutely_integrable_on UNIV
  proof (rule dominated_convergence_absolutely_integrable)
    show ( $\lambda x. \text{if } x \in ?U k \text{ then } ?D x \text{ else } 0$ ) absolutely_integrable_on UNIV for
  k
    unfolding absolutely_integrable_restrict_UNIV using DUn by simp
    let ?nD =  $\lambda n x. \text{if } x \in ?U n \text{ then norm } (?D x) \text{ else } 0$ 
    show ?h integrable_on UNIV
    proof (rule monotone_convergence_increasing [THEN conjunct1])
      show ?nD k integrable_on UNIV for k
        using DUn
        unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
      (simp add: image_UN)
      { fix n::nat
        have (norm  $\circ$  f) absolutely_integrable_on ( $\bigcup m \leq n. g \text{ ' } F m$ )
          using absolutely_integrable_norm fgU by blast
        then have integral (?U n) (norm  $\circ$  ?D) = integral (g ' ?U n) (norm  $\circ$  f)
          using iff [of n ?lift  $\circ$  norm  $\circ$  f integral (g ' ?U n) (?lift  $\circ$  norm  $\circ$  f)]
          unfolding absolutely_integrable_on_1_iff integral_on_1_eq image_UN
        by (auto simp: o_def)
      }
    }
  qed

```

```

}
moreover have bounded (range ( $\lambda k. \text{integral } (g \text{ ' } ?U k) (norm \circ f)$ ))
  unfolding bounded_iff
proof (rule exI, clarify)
  fix k
  show norm (integral ( $g \text{ ' } ?U k$ ) (norm  $\circ f$ ))  $\leq$  integral ( $g \text{ ' } (\bigcup n. F n)$ )
(norm  $\circ f$ )
  unfolding integral_restrict_UNIV [of _ norm  $\circ f$ , symmetric]
proof (rule integral_norm_bound_integral)
  show ( $\lambda x. \text{if } x \in g \text{ ' } ?U k \text{ then } (norm \circ f) x \text{ else } 0$ ) integrable_on UNIV
  ( $\lambda x. \text{if } x \in g \text{ ' } (\bigcup n. F n) \text{ then } (norm \circ f) x \text{ else } 0$ ) integrable_on
UNIV
  using fgU fs
unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
  by (auto simp: image_UN)
qed auto
qed
ultimately show bounded (range ( $\lambda k. \text{integral UNIV } (?nD k)$ ))
  unfolding integral_restrict_UNIV image_UN [symmetric] o_def by simp
next
  show ( $\lambda k. \text{if } x \in ?U k \text{ then } norm (?D x) \text{ else } 0$ )  $\longrightarrow$  (if  $x \in (\bigcup n. F n)$ 
then norm ( $?D x$ ) else 0) for x
  by (force intro: tendsto_eventually eventually_sequentiallyI)
qed auto
next
  show ( $\lambda k. \text{if } x \in ?U k \text{ then } ?D x \text{ else } 0$ )  $\longrightarrow$  (if  $x \in (\bigcup n. F n)$  then  $?D x$ 
else 0) for x
  proof clarsimp
  fix n
  assume  $x \in F n$ 
  show ( $\lambda m. \text{if } \exists j \in \{..m\}. x \in F j \text{ then } ?D x \text{ else } 0$ )  $\longrightarrow$   $?D x$ 
  using  $\langle x \in F n \rangle$  by (auto intro!: tendsto_eventually eventually_sequentiallyI
[of n])
qed
qed auto
then show  $Dai: ?D \text{ absolutely\_integrable\_on } (\bigcup n. F n)$ 
  unfolding absolutely_integrable_restrict_UNIV by simp
show integral ( $\bigcup n. F n$ )  $?D = \text{integral } ((\bigcup x. g \text{ ' } F x)) f$ 
proof (rule LIMSEQ_unique)
  show ( $\lambda n. \text{integral } (\bigcup m \leq n. g \text{ ' } F m) f$ )  $\longrightarrow$  integral ( $\bigcup n. F n$ )  $?D$ 
  unfolding D_int [symmetric] by (rule integral_countable_UN [OF Dai
F_leb])
qed (use fgU in metis)
qed
qed

```

theorem has_absolute_integral_change_of_variables:

fixes $f :: \text{real}^m :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^n :: _$


```

assumes  $S: S \in \text{sets lebesgue}$ 
and  $\text{der\_}g: \bigwedge x. x \in S \implies (g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } S)$ 
and  $\text{inj: inj\_on } g \ S$ 
shows  $(\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) \text{ absolutely\_integrable\_on } S \wedge$ 
 $\text{integral } S (\lambda x. |\det (\text{matrix } (g' x))| *_R f(g x)) = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$ 
proof -
obtain  $C \ N$  where  $\text{fsigma } C$  and  $N: N \in \text{null\_sets lebesgue}$  and  $\text{CNS: } C \cup N$ 
 $= S$  and  $\text{disjnt } C \ N$ 
using  $\text{lebesgue\_set\_almost\_fsigma } [OF \ S]$  .
then obtain  $F :: \text{nat} \Rightarrow (\text{real}^m :: \_) \text{ set}$ 
where  $F: \text{range } F \subseteq \text{Collect compact}$  and  $\text{Ceq: } C = \text{Union}(\text{range } F)$ 
using  $\text{fsigma\_Union\_compact}$  by  $\text{metis}$ 
have  $\text{negligible } N$ 
using  $N$  by  $(\text{simp add: negligible\_iff\_null\_sets})$ 
let  $?D = \lambda x. |\det (\text{matrix } (g' x))| *_R f (g x)$ 
have  $?D \text{ absolutely\_integrable\_on } C \wedge \text{integral } C \ ?D = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } C) \wedge \text{integral } (g \text{ ' } C) f = b$ 
unfolding  $\text{Ceq}$ 
proof  $(\text{rule has\_absolute\_integral\_change\_of\_variables\_compact\_family})$ 
fix  $n \ x$ 
assume  $x \in \bigcup (F \text{ ' } UNIV)$ 
then show  $(g \text{ has\_derivative } g' x) \text{ (at } x \text{ within } \bigcup (F \text{ ' } UNIV))$ 
using  $\text{Ceq } \langle C \cup N = S \rangle$   $\text{der\_}g \text{ has\_derivative\_subset}$  by  $\text{blast}$ 
next
have  $\bigcup (F \text{ ' } UNIV) \subseteq S$ 
using  $\text{Ceq } \langle C \cup N = S \rangle$  by  $\text{blast}$ 
then show  $\text{inj\_on } g (\bigcup (F \text{ ' } UNIV))$ 
using  $\text{inj}$  by  $(\text{meson inj\_on\_subset})$ 
qed  $(\text{use } F \text{ in auto})$ 
moreover
have  $?D \text{ absolutely\_integrable\_on } C \wedge \text{integral } C \ ?D = b$ 
 $\longleftrightarrow ?D \text{ absolutely\_integrable\_on } S \wedge \text{integral } S \ ?D = b$ 
proof  $(\text{rule conj\_cong})$ 
have  $\text{neg: negligible } \{x \in C - S. ?D x \neq 0\}$   $\text{negligible } \{x \in S - C. ?D x \neq 0\}$ 
using  $\text{CNS}$  by  $(\text{blast intro: negligible\_subset } [OF \ \langle \text{negligible } N \rangle])+$ 
then show  $(?D \text{ absolutely\_integrable\_on } C) = (?D \text{ absolutely\_integrable\_on } S)$ 
by  $(\text{rule absolutely\_integrable\_spike\_set\_eq})$ 
show  $(\text{integral } C \ ?D = b) \longleftrightarrow (\text{integral } S \ ?D = b)$ 
using  $\text{integral\_spike\_set } [OF \ \text{neg}]$  by  $\text{simp}$ 
qed
moreover
have  $f \text{ absolutely\_integrable\_on } (g \text{ ' } C) \wedge \text{integral } (g \text{ ' } C) f = b$ 
 $\longleftrightarrow f \text{ absolutely\_integrable\_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$ 
proof  $(\text{rule conj\_cong})$ 
have  $g \text{ differentiable\_on } N$ 
by  $(\text{metis CNS der\_}g \text{ differentiable\_def differentiable\_on\_def differentiable\_on\_subset})$ 

```

```

sup.cobounded2)
  with ⟨negligible N⟩
  have neg_gN: negligible (g ' N)
  by (blast intro: negligible_differentiable_image_negligible)
  have neg: negligible {x ∈ g ' C - g ' S. f x ≠ 0}
    negligible {x ∈ g ' S - g ' C. f x ≠ 0}
  using CNS by (blast intro: negligible_subset [OF neg_gN])+
  then show (f absolutely_integrable_on g ' C) = (f absolutely_integrable_on g
' S)
  by (rule absolutely_integrable_spike_set_eq)
  show (integral (g ' C) f = b) ↔ (integral (g ' S) f = b)
  using integral_spike_set [OF neg] by simp
qed
ultimately show ?thesis
  by simp
qed

```

corollary *absolutely_integrable_change_of_variables:*

```

fixes f :: real^m::{finite,wellorder} ⇒ real^n and g :: real^m::_ ⇒ real^m::_
assumes S ∈ sets lebesgue
  and ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
  and inj_on g S
shows f absolutely_integrable_on (g ' S)
  ↔ (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S
using assms has_absolute_integral_change_of_variables by blast

```

corollary *integral_change_of_variables:*

```

fixes f :: real ⇒ real^n::{finite,wellorder} and g :: real ⇒ real^m::_
assumes S: S ∈ sets lebesgue
  and der_g: ∧x. x ∈ S ⇒ (g has_derivative g' x) (at x within S)
  and inj: inj_on g S
  and disj: (f absolutely_integrable_on (g ' S) ∨
(λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S)
shows integral (g ' S) f = integral S (λx. |det (matrix (g' x))| *R f(g x))
using has_absolute_integral_change_of_variables [OF S der_g inj] disj
by blast

```

lemma *has_absolute_integral_change_of_variables_1:*

```

fixes f :: real ⇒ real^n::{finite,wellorder} and g :: real ⇒ real^m::_
assumes S: S ∈ sets lebesgue
  and der_g: ∧x. x ∈ S ⇒ (g has_vector_derivative g' x) (at x within S)
  and inj: inj_on g S
shows (λx. |g' x| *R f(g x)) absolutely_integrable_on S ∧
  integral S (λx. |g' x| *R f(g x)) = b
  ↔ f absolutely_integrable_on (g ' S) ∧ integral (g ' S) f = b

```

proof –

```

let ?lift = vec :: real ⇒ real^1
let ?drop = (λx::real^1. x $ 1)

```

```

have S': ?lift ' S ∈ sets lebesgue
by (auto intro: differentiable_image_in_sets_lebesgue [OF S] differentiable_vec)
have ((λx. vec (g (x $ 1))) has_derivative (*R) (g' z)) (at (vec z) within ?lift '
S)
  if z ∈ S for z
    using der_g [OF that]
    by (simp add: has_vector_derivative_def has_derivative_vector_1)
then have der': ∧x. x ∈ ?lift ' S ⇒
  ( ?lift ∘ g ∘ ?drop has_derivative (*R) (g' (?drop x)) ) (at x within ?lift ' S)
  by (auto simp: o_def)
have inj': inj_on (vec ∘ g ∘ ?drop) (vec ' S)
  using inj by (simp add: inj_on_def)
let ?fg = λx. |g' x| *R f(g x)
have ((λx. ?fg x $ i) absolutely_integrable_on S ∧ ((λx. ?fg x $ i) has_integral
b $ i) S
  ↔ (λx. f x $ i) absolutely_integrable_on g ' S ∧ ((λx. f x $ i) has_integral
b $ i) (g ' S)) for i
  using has_absolute_integral_change_of_variables [OF S' der' inj', of λx.
?lift(f (?drop x) $ i) ?lift (b$ i)]
  unfolding integrable_on_1_iff_integral_on_1_eq absolutely_integrable_on_1_iff
absolutely_integrable_drop absolutely_integrable_on_def
  by (auto simp: image_comp o_def integral_vec1_eq has_integral_iff)
then have ?fg absolutely_integrable_on S ∧ (?fg has_integral b) S
  ↔ f absolutely_integrable_on (g ' S) ∧ (f has_integral b) (g ' S)
  unfolding has_integral_componentwise_iff [where y=b]
  absolutely_integrable_componentwise_iff [where f=f]
  absolutely_integrable_componentwise_iff [where f = ?fg]
  by (force simp: Basis_vec_def cart_eq_inner_axis)
then show ?thesis
  using absolutely_integrable_on_def by blast
qed

```

corollary absolutely_integrable_change_of_variables_1:

```

fixes f :: real ⇒ realn::{finite,wellorder} and g :: real ⇒ real
assumes S: S ∈ sets lebesgue
  and der_g: ∧x. x ∈ S ⇒ (g has_vector_derivative g' x) (at x within S)
  and inj: inj_on g S
shows (f absolutely_integrable_on g ' S ↔
  (λx. |g' x| *R f(g x)) absolutely_integrable_on S)
using has_absolute_integral_change_of_variables_1 [OF assms] by auto

```

when $n = (1::'a)$

lemma has_absolute_integral_change_of_variables_1':

```

fixes f :: real ⇒ real and g :: real ⇒ real
assumes S: S ∈ sets lebesgue
  and der_g: ∧x. x ∈ S ⇒ (g has_field_derivative g' x) (at x within S)
  and inj: inj_on g S
shows (λx. |g' x| *R f(g x)) absolutely_integrable_on S ∧

```

$\text{integral } S (\lambda x. |g' x| *_R f(g x)) = b$

$\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$

proof –

have $(\lambda x. |g' x| *_R \text{vec } (f(g x)) :: \text{real}^{\wedge 1}) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |g' x| *_R \text{vec } (f(g x))) = (\text{vec } b :: \text{real}^{\wedge 1})$
 $\longleftrightarrow (\lambda x. \text{vec } (f x) :: \text{real}^{\wedge 1}) \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge$
 $\text{integral } (g \text{ ' } S) (\lambda x. \text{vec } (f x)) = (\text{vec } b :: \text{real}^{\wedge 1})$

using *assms unfolding has_real_derivative_iff_has_vector_derivative*

by *(intro has_absolute_integral_change_of_variables_1 assms) auto*

thus *?thesis*

by *(simp add: absolutely_integrable_on_1_iff_integral_on_1_eq)*

qed

9.32.6 Change of variables for integrals: special case of linear function

lemma *has_absolute_integral_change_of_variables_linear*:

fixes $f :: \text{real}^{\wedge m} :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^{\wedge n}$ **and** $g :: \text{real}^{\wedge m} :: _ \Rightarrow \text{real}^{\wedge m} :: _$

assumes *linear g*

shows $(\lambda x. |\det (\text{matrix } g)| *_R f(g x)) \text{ absolutely_integrable_on } S \wedge$
 $\text{integral } S (\lambda x. |\det (\text{matrix } g)| *_R f(g x)) = b$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S) \wedge \text{integral } (g \text{ ' } S) f = b$

proof *(cases det(matrix g) = 0)*

case *True*

then have *negligible(g ' S)*

using *assms det_nz_iff_inj negligible_linear_singular_image* **by** *blast*

with *True* **show** *?thesis*

by *(auto simp: absolutely_integrable_on_def integrable_negligible integral_negligible)*

next

case *False*

then obtain *h* **where** $h: \bigwedge x. x \in S \implies h(g x) = x$ *linear h*

using *assms det_nz_iff_inj linear_injective_isomorphism* **by** *metis*

show *?thesis*

proof *(rule has_absolute_integral_change_of_variables_invertible)*

show *(g has_derivative g) (at x within S) for x*

by *(simp add: assms linear_imp_has_derivative)*

show *continuous_on (g ' S) h*

using *continuous_on_eq_continuous_within has_derivative_continuous linear_imp_has_derivative h* **by** *blast*

qed *(use h in auto)*

qed

lemma *absolutely_integrable_change_of_variables_linear*:

fixes $f :: \text{real}^{\wedge m} :: \{\text{finite, wellorder}\} \Rightarrow \text{real}^{\wedge n}$ **and** $g :: \text{real}^{\wedge m} :: _ \Rightarrow \text{real}^{\wedge m} :: _$

assumes *linear g*

shows $(\lambda x. |\det (\text{matrix } g)| *_R f(g x)) \text{ absolutely_integrable_on } S$
 $\longleftrightarrow f \text{ absolutely_integrable_on } (g \text{ ' } S)$

using *assms has_absolute_integral_change_of_variables_linear* **by** *blast*

lemma *absolutely_integrable_on_linear_image*:
fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^m :: _$
assumes *linear g*
shows f *absolutely_integrable_on* $(g \text{ ' } S)$
 $\longleftrightarrow (f \circ g)$ *absolutely_integrable_on* $S \vee \det(\text{matrix } g) = 0$
unfolding *assms absolutely_integrable_change_of_variables_linear [OF assms, symmetric]* *absolutely_integrable_on_scaleR_iff*
by (*auto simp: set_integrable_def*)

lemma *integral_change_of_variables_linear*:
fixes $f :: \text{real}^m :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n$ **and** $g :: \text{real}^m :: _ \Rightarrow \text{real}^m :: _$
assumes *linear g*
and f *absolutely_integrable_on* $(g \text{ ' } S) \vee (f \circ g)$ *absolutely_integrable_on* S
shows $\text{integral } (g \text{ ' } S) f = |\det(\text{matrix } g)| *_R \text{integral } S (f \circ g)$
proof –
have $((\lambda x. |\det(\text{matrix } g)| *_R f(g x))$ *absolutely_integrable_on* $S) \vee (f$ *absolutely_integrable_on* $g \text{ ' } S)$
using *absolutely_integrable_on_linear_image assms by blast*
moreover
have *?thesis* **if** $((\lambda x. |\det(\text{matrix } g)| *_R f(g x))$ *absolutely_integrable_on* $S) (f$ *absolutely_integrable_on* $g \text{ ' } S)$
using *has_absolute_integral_change_of_variables_linear [OF <linear g>]* *that*
by (*auto simp: o_def*)
ultimately show *?thesis*
using *absolutely_integrable_change_of_variables_linear [OF <linear g>]*
by blast
qed

9.32.7 Change of variable for measure

lemma *has_measure_differentiable_image*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: _$
assumes $S \in \text{sets lebesgue}$
and $\bigwedge x. x \in S \implies (f$ *has_derivative* $f' x)$ (*at x within S*)
and *inj_on f S*
shows $f \text{ ' } S \in \text{lmeasurable} \wedge \text{measure lebesgue } (f \text{ ' } S) = m$
 $\longleftrightarrow ((\lambda x. |\det(\text{matrix } (f' x))|)$ *has_integral* $m) S$
using *has_absolute_integral_change_of_variables [OF assms, of $\lambda x. (1 :: \text{real}^1)$ vec m]*
unfolding *absolutely_integrable_on_1_iff integral_on_1_eq integrable_on_1_iff absolutely_integrable_on_def*
by (*auto simp: has_integral_iff lmeasurable_iff integrable_on lmeasure_integral*)

lemma *measurable_differentiable_image_eq*:
fixes $f :: \text{real}^n :: \{\text{finite}, \text{wellorder}\} \Rightarrow \text{real}^n :: _$
assumes $S \in \text{sets lebesgue}$
and $\bigwedge x. x \in S \implies (f$ *has_derivative* $f' x)$ (*at x within S*)
and *inj_on f S*
shows $f \text{ ' } S \in \text{lmeasurable} \longleftrightarrow (\lambda x. |\det(\text{matrix } (f' x))|)$ *integrable_on* S

3802

```
using has_measure_differentiable_image [OF assms]
by blast
```

```
lemma measurable_differentiable_image_alt:
  fixes f :: real^n::{finite,wellorder} => real^n::_
  assumes S ∈ sets lebesgue
    and  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x)$  (at x within S)
    and inj_on f S
  shows  $f' S \in \text{lmeasurable} \iff (\lambda x. |\det(\text{matrix}(f' x))|)$  absolutely_integrable_on
  S
  using measurable_differentiable_image_eq [OF assms]
  by (simp only: absolutely_integrable_on_iff_nonneg)
```

```
lemma measure_differentiable_image_eq:
  fixes f :: real^n::{finite,wellorder} => real^n::_
  assumes S: S ∈ sets lebesgue
    and der_f:  $\bigwedge x. x \in S \implies (f \text{ has\_derivative } f' x)$  (at x within S)
    and inj: inj_on f S
    and intS:  $(\lambda x. |\det(\text{matrix}(f' x))|)$  integrable_on S
  shows  $\text{measure lebesgue}(f' S) = \text{integral } S (\lambda x. |\det(\text{matrix}(f' x))|)$ 
  using measurable_differentiable_image_eq [OF S der_f inj]
    assms has_measure_differentiable_image by blast
```

end

9.33 Lipschitz Continuity

theory Lipschitz

imports

Derivative Abstract_Metric_Spaces

begin

definition lipschitz_on

where $\text{lipschitz_on } C U f \iff (0 \leq C \wedge (\forall x \in U. \forall y \in U. \text{dist}(f x) (f y) \leq C$
* $\text{dist } x y))$

bundle lipschitz_syntax begin

notation lipschitz_on ($_ - \text{lipschitz}'_ \text{on}$ [1000])

end

bundle no_lipschitz_syntax begin

no_notation lipschitz_on ($_ - \text{lipschitz}'_ \text{on}$ [1000])

end

unbundle lipschitz_syntax

lemma lipschitz_onI: $L - \text{lipschitz_on } X f$

if $\bigwedge x y. x \in X \implies y \in X \implies \text{dist}(f x) (f y) \leq L * \text{dist } x y$ $0 \leq L$

using that by (auto simp: lipschitz_on_def)

lemma *lipschitz_onD*:

$\text{dist } (f\ x) (f\ y) \leq L * \text{dist } x\ y$
if L -lipschitz_on X f $x \in X$ $y \in X$
using *that* **by** (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_nonneg*:

$0 \leq L$ **if** L -lipschitz_on X f
using *that* **by** (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_normD*:

$\text{norm } (f\ x - f\ y) \leq L * \text{norm } (x - y)$
if L -lipschitz_on X f $x \in X$ $y \in X$
using *lipschitz_onD*[*OF that*]
by (*simp add: dist_norm*)

lemma *lipschitz_on_mono*: L -lipschitz_on D f **if** M -lipschitz_on E f $D \subseteq E$ $M \leq L$

using *that*
by (*force simp: lipschitz_on_def intro: order_trans[OF _ mult_right_mono]*)

lemmas *lipschitz_on_subset* = *lipschitz_on_mono*[*OF _ _ order_refl*]

and *lipschitz_on_le* = *lipschitz_on_mono*[*OF _ order_refl*]

lemma *lipschitz_on_leI*:

L -lipschitz_on X f
if $\bigwedge x\ y. x \in X \implies y \in X \implies x \leq y \implies \text{dist } (f\ x) (f\ y) \leq L * \text{dist } x\ y$
 $0 \leq L$
for $f::'a::\{\text{linorder_topology, ordered_real_vector, metric_space}\} \Rightarrow 'b::\text{metric_space}$

proof (*rule lipschitz_onI*)

fix $x\ y$ **assume** xy : $x \in X$ $y \in X$

consider $y \leq x \mid x \leq y$

by (*rule le_cases*)

then show $\text{dist } (f\ x) (f\ y) \leq L * \text{dist } x\ y$

proof *cases*

case 1

then have $\text{dist } (f\ y) (f\ x) \leq L * \text{dist } y\ x$

by (*auto intro!: that xy*)

then show *?thesis*

by (*simp add: dist_commute*)

qed (*auto intro!: that xy*)

qed *fact*

lemma *lipschitz_on_concat*:

fixes $a\ b\ c::\text{real}$

assumes f : L -lipschitz_on $\{a .. b\}$ f

assumes g : L -lipschitz_on $\{b .. c\}$ g

assumes fg : $f\ b = g\ b$

shows L -lipschitz_on L $\{a .. c\}$ $(\lambda x. \text{if } x \leq b \text{ then } f\ x \text{ else } g\ x)$

(*is lipschitz_on _ _ ?f*)

```

proof (rule lipschitz_on_leI)
  fix x y
  assume x: x ∈ {a..c} and y: y ∈ {a..c} and xy: x ≤ y
  consider x ≤ b ∧ b < y | x ≥ b ∨ y ≤ b by arith
  then show dist (?f x) (?f y) ≤ L * dist x y
  proof cases
    case 1
      have dist (f x) (g y) ≤ dist (f x) (f b) + dist (g b) (g y)
        unfolding fg by (rule dist_triangle)
      also have dist (f x) (f b) ≤ L * dist x b
        using 1 x
        by (auto intro!: lipschitz_onD[OF f])
      also have dist (g b) (g y) ≤ L * dist b y
        using 1 x y
        by (auto intro!: lipschitz_onD[OF g] lipschitz_onD[OF f])
      finally have dist (f x) (g y) ≤ L * dist x b + L * dist b y
        by simp
      also have ... = L * (dist x b + dist b y)
        by (simp add: algebra_simps)
      also have dist x b + dist b y = dist x y
        using 1 x y
        by (auto simp: dist_real_def abs_real_def)
      finally show ?thesis
        using 1 by simp
    next
      case 2
      with lipschitz_onD[OF f, of x y] lipschitz_onD[OF g, of x y] x y xy
      show ?thesis
        by (auto simp: fg)
  qed
qed (rule lipschitz_on_nonneg[OF f])

```

```

lemma lipschitz_on_concat_max:
  fixes a b c::real
  assumes f: L-lipschitz_on {a .. b} f
  assumes g: M-lipschitz_on {b .. c} g
  assumes fg: f b = g b
  shows (max L M)-lipschitz_on {a .. c} (λx. if x ≤ b then f x else g x)
proof -
  have lipschitz_on (max L M) {a .. b} f lipschitz_on (max L M) {b .. c} g
    by (auto intro!: lipschitz_on_mono[OF f order_refl] lipschitz_on_mono[OF g
order_refl])
  from lipschitz_on_concat[OF this fg] show ?thesis .
qed

```

Equivalence between "abstract" and "type class" Lipschitz notions, for all types in the metric space class

```

lemma Lipschitz_map_euclidean [simp]:
  Lipschitz_continuous_map euclidean_metric euclidean_metric f

```



```

   $\longleftrightarrow (\exists B. \text{lipschitz\_on } B \text{ UNIV } f) \text{ (is } ?lhs \longleftrightarrow ?rhs)$ 
proof
  show  $?lhs \implies ?rhs$ 
  by (force simp add: Lipschitz_continuous_map_pos lipschitz_on_def less_le_not_le)
  show  $?rhs \implies ?lhs$ 
  by (auto simp: Lipschitz_continuous_map_def lipschitz_on_def)
qed

```

Continuity

proposition *lipschitz_on_uniformly_continuous:*

assumes L -lipschitz_on X f

shows uniformly_continuous_on X f

unfolding uniformly_continuous_on_def

proof safe

fix $e::\text{real}$

assume $0 < e$

from *assms* **have** $l: (L+1)$ -lipschitz_on X f

by (rule lipschitz_on_mono) auto

show $\exists d > 0. \forall x \in X. \forall x' \in X. \text{dist } x' \ x < d \longrightarrow \text{dist } (f \ x') \ (f \ x) < e$

using lipschitz_onD[OF l] lipschitz_on_nonneg[OF *assms*] $\langle 0 < e \rangle$

by (force intro!: exI[**where** $x=e/(L+1)$] simp: field_simps)

qed

proposition *lipschitz_on_continuous_on:*

continuous_on X f **if** L -lipschitz_on X f

by (rule uniformly_continuous_imp_continuous[OF lipschitz_on_uniformly_continuous[OF that]])

lemma *lipschitz_on_continuous_within:*

continuous (at x within X) f **if** L -lipschitz_on X f $x \in X$

using lipschitz_on_continuous_on[OF that(1)] that(2)

by (auto simp: continuous_on_eq_continuous_within)

Differentiable functions

proposition *bounded_derivative_imp_lipschitz:*

assumes $\bigwedge x. x \in X \implies (f \text{ has_derivative } f' \ x) \text{ (at } x \text{ within } X)$

assumes convex: convex X

assumes $\bigwedge x. x \in X \implies \text{onorm } (f' \ x) \leq C$ $0 \leq C$

shows C -lipschitz_on X f

proof (rule lipschitz_onI)

show $\bigwedge x \ y. x \in X \implies y \in X \implies \text{dist } (f \ x) \ (f \ y) \leq C * \text{dist } x \ y$

by (auto intro!: *assms* differentiable_bound[unfolded dist_norm[symmetric], OF convex])

qed fact

Structural introduction rules

named_theorems *lipschitz_intros* structural introduction rules for Lipschitz controls

lemma *lipschitz_on_compose* [*lipschitz_intros*]:
 $(D * C)$ -lipschitz_on U $(g \circ f)$
if f : C -lipschitz_on U f **and** g : D -lipschitz_on $(f'U)$ g
proof (rule *lipschitz_onI*)
show $D * C \geq 0$ **using** *lipschitz_on_nonneg*[*OF f*] *lipschitz_on_nonneg*[*OF g*]
by *auto*
fix x y **assume** H : $x \in U$ $y \in U$
have $\text{dist } (g (f x)) (g (f y)) \leq D * \text{dist } (f x) (f y)$
apply (rule *lipschitz_onD*[*OF g*]) **using** H **by** *auto*
also have $\dots \leq D * C * \text{dist } x y$
using *mult_left_mono*[*OF lipschitz_onD(1)*][*OF f H*] *lipschitz_on_nonneg*[*OF g*]
g] **by** *auto*
finally show $\text{dist } ((g \circ f) x) ((g \circ f) y) \leq D * C * \text{dist } x y$
unfolding *comp_def* **by** (*auto simp add: mult.commute*)
qed

lemma *lipschitz_on_compose2*:
 $(D * C)$ -lipschitz_on U $(\lambda x. g (f x))$
if C -lipschitz_on U f D -lipschitz_on $(f'U)$ g
using *lipschitz_on_compose*[*OF that*] **by** (*simp add: o_def*)

lemma *lipschitz_on_cong*[*cong*]:
 C -lipschitz_on U $g \longleftrightarrow D$ -lipschitz_on V f
if $C = D$ $U = V$ $\bigwedge x. x \in V \implies g x = f x$
using *that* **by** (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_transform*:
 C -lipschitz_on U g
if C -lipschitz_on U f
 $\bigwedge x. x \in U \implies g x = f x$
using *that*
by *simp*

lemma *lipschitz_on_empty_iff*[*simp*]: C -lipschitz_on $\{\}$ $f \longleftrightarrow C \geq 0$
by (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_insert_iff*[*simp*]:
 C -lipschitz_on $(\text{insert } y X)$ $f \longleftrightarrow$
 C -lipschitz_on X $f \wedge (\forall x \in X. \text{dist } (f x) (f y) \leq C * \text{dist } x y)$
by (*auto simp: lipschitz_on_def dist_commute*)

lemma *lipschitz_on_singleton* [*lipschitz_intros*]: $C \geq 0 \implies C$ -lipschitz_on $\{x\}$ f
and *lipschitz_on_empty* [*lipschitz_intros*]: $C \geq 0 \implies C$ -lipschitz_on $\{\}$ f
by *simp_all*

lemma *lipschitz_on_id* [*lipschitz_intros*]: 1 -*lipschitz_on* U $(\lambda x. x)$
by (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_constant* [*lipschitz_intros*]: 0 -*lipschitz_on* U $(\lambda x. c)$
by (*auto simp: lipschitz_on_def*)

lemma *lipschitz_on_add* [*lipschitz_intros*]:
fixes $f::'a::\text{metric_space} \Rightarrow 'b::\text{real_normed_vector}$
assumes C -*lipschitz_on* U f
 D -*lipschitz_on* U g
shows $(C+D)$ -*lipschitz_on* U $(\lambda x. f\ x + g\ x)$
proof (*rule lipschitz_onI*)
show $C + D \geq 0$
using *lipschitz_on_nonneg*[*OF assms(1)*] *lipschitz_on_nonneg*[*OF assms(2)*]
by auto
fix $x\ y$ **assume** $H: x \in U\ y \in U$
have $\text{dist}\ (f\ x + g\ x)\ (f\ y + g\ y) \leq \text{dist}\ (f\ x)\ (f\ y) + \text{dist}\ (g\ x)\ (g\ y)$
by (*simp add: dist_triangle_add*)
also have $\dots \leq C * \text{dist}\ x\ y + D * \text{dist}\ x\ y$
using *lipschitz_onD(1)*[*OF assms(1)*] H *lipschitz_onD(1)*[*OF assms(2)*] H **by auto**
finally show $\text{dist}\ (f\ x + g\ x)\ (f\ y + g\ y) \leq (C+D) * \text{dist}\ x\ y$ **by** (*auto simp add: algebra_simps*)
qed

lemma *lipschitz_on_cmult* [*lipschitz_intros*]:
fixes $f::'a::\text{metric_space} \Rightarrow 'b::\text{real_normed_vector}$
assumes C -*lipschitz_on* U f
shows $(\text{abs}(a) * C)$ -*lipschitz_on* U $(\lambda x. a *_{\mathbb{R}} f\ x)$
proof (*rule lipschitz_onI*)
show $\text{abs}(a) * C \geq 0$ **using** *lipschitz_on_nonneg*[*OF assms(1)*] **by auto**
fix $x\ y$ **assume** $H: x \in U\ y \in U$
have $\text{dist}\ (a *_{\mathbb{R}} f\ x)\ (a *_{\mathbb{R}} f\ y) = \text{abs}(a) * \text{dist}\ (f\ x)\ (f\ y)$
by (*metis dist_norm norm_scaleR real_vector.scale_right_diff_distrib*)
also have $\dots \leq \text{abs}(a) * C * \text{dist}\ x\ y$
using *lipschitz_onD(1)*[*OF assms(1)*] H **by** (*simp add: Groups.mult_ac(1) mult_left_mono*)
finally show $\text{dist}\ (a *_{\mathbb{R}} f\ x)\ (a *_{\mathbb{R}} f\ y) \leq |a| * C * \text{dist}\ x\ y$ **by auto**
qed

lemma *lipschitz_on_cmult_real* [*lipschitz_intros*]:
fixes $f::'a::\text{metric_space} \Rightarrow \text{real}$
assumes C -*lipschitz_on* U f
shows $(\text{abs}(a) * C)$ -*lipschitz_on* U $(\lambda x. a * f\ x)$
using *lipschitz_on_cmult*[*OF assms*] **by auto**

lemma *lipschitz_on_cmult_nonneg* [*lipschitz_intros*]:
fixes $f::'a::\text{metric_space} \Rightarrow 'b::\text{real_normed_vector}$

```

assumes  $C$ -lipschitz_on  $U$   $f$ 
   $a \geq 0$ 
shows  $(a * C)$ -lipschitz_on  $U$   $(\lambda x. a *_{\mathbb{R}} f x)$ 
using lipschitz_on_cmult[OF assms(1), of  $a$ ] assms(2) by auto

```

```

lemma lipschitz_on_cmult_real_nonneg [lipschitz_intros]:
fixes  $f::'a::metric\_space \Rightarrow real$ 
assumes  $C$ -lipschitz_on  $U$   $f$ 
   $a \geq 0$ 
shows  $(a * C)$ -lipschitz_on  $U$   $(\lambda x. a * f x)$ 
using lipschitz_on_cmult_nonneg[OF assms] by auto

```

```

lemma lipschitz_on_cmult_upper [lipschitz_intros]:
fixes  $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
assumes  $C$ -lipschitz_on  $U$   $f$ 
   $abs(a) \leq D$ 
shows  $(D * C)$ -lipschitz_on  $U$   $(\lambda x. a *_{\mathbb{R}} f x)$ 
apply (rule lipschitz_on_mono[OF lipschitz_on_cmult[OF assms(1), of  $a$ ], of
   $D * C$ ])
using assms(2) lipschitz_on_nonneg[OF assms(1)] mult_right_mono by auto

```

```

lemma lipschitz_on_cmult_real_upper [lipschitz_intros]:
fixes  $f::'a::metric\_space \Rightarrow real$ 
assumes  $C$ -lipschitz_on  $U$   $f$ 
   $abs(a) \leq D$ 
shows  $(D * C)$ -lipschitz_on  $U$   $(\lambda x. a * f x)$ 
using lipschitz_on_cmult_upper[OF assms] by auto

```

```

lemma lipschitz_on_minus[lipschitz_intros]:
fixes  $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
assumes  $C$ -lipschitz_on  $U$   $f$ 
shows  $C$ -lipschitz_on  $U$   $(\lambda x. - f x)$ 
by (metis (mono_tags, lifting) assms dist_minus lipschitz_on_def)

```

```

lemma lipschitz_on_minus_iff[simp]:
 $L$ -lipschitz_on  $X$   $(\lambda x. - f x) \longleftrightarrow L$ -lipschitz_on  $X$   $f$ 
 $L$ -lipschitz_on  $X$   $(- f) \longleftrightarrow L$ -lipschitz_on  $X$   $f$ 
for  $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
using lipschitz_on_minus[of  $L$   $X$   $f$ ] lipschitz_on_minus[of  $L$   $X$   $-f$ ]
by auto

```

```

lemma lipschitz_on_diff[lipschitz_intros]:
fixes  $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$ 
assumes  $C$ -lipschitz_on  $U$   $f$   $D$ -lipschitz_on  $U$   $g$ 
shows  $(C + D)$ -lipschitz_on  $U$   $(\lambda x. f x - g x)$ 
  using lipschitz_on_add[OF assms(1) lipschitz_on_minus[OF assms(2)]] by
  auto

```

```

lemma lipschitz_on_closure [lipschitz_intros]:

```

```

  assumes  $C$ -lipschitz_on  $U$   $f$  continuous_on (closure  $U$ )  $f$ 
  shows  $C$ -lipschitz_on (closure  $U$ )  $f$ 
proof (rule lipschitz_onI)
  show  $C \geq 0$  using lipschitz_on_nonneg[OF assms(1)] by simp
  fix  $x$   $y$  assume  $x \in$  closure  $U$   $y \in$  closure  $U$ 
  obtain  $u$   $v$  :: nat  $\Rightarrow$  'a where *:  $\bigwedge n. u\ n \in U$   $u \longrightarrow x$ 
       $\bigwedge n. v\ n \in U$   $v \longrightarrow y$ 
    using  $\langle x \in$  closure  $U \rangle \langle y \in$  closure  $U \rangle$  unfolding closure_sequential by blast
  have  $a$ :  $(\lambda n. f (u\ n)) \longrightarrow f\ x$ 
    using *(1) *(2)  $\langle x \in$  closure  $U \rangle \langle$ continuous_on (closure  $U$ )  $f \rangle$ 
    unfolding comp_def continuous_on_closure_sequentially[of  $U$   $f$ ] by auto
  have  $b$ :  $(\lambda n. f (v\ n)) \longrightarrow f\ y$ 
    using *(3) *(4)  $\langle y \in$  closure  $U \rangle \langle$ continuous_on (closure  $U$ )  $f \rangle$ 
    unfolding comp_def continuous_on_closure_sequentially[of  $U$   $f$ ] by auto
  have  $l$ :  $(\lambda n. C * \text{dist} (u\ n) (v\ n) - \text{dist} (f (u\ n)) (f (v\ n))) \longrightarrow C * \text{dist}\ x\ y$ 
  -  $\text{dist} (f\ x) (f\ y)$ 
    by (intro tendsto_intros *  $a$   $b$ )
  have  $C * \text{dist} (u\ n) (v\ n) - \text{dist} (f (u\ n)) (f (v\ n)) \geq 0$  for  $n$ 
    using lipschitz_onD(1)[OF assms(1)  $\langle u\ n \in U \rangle \langle v\ n \in U \rangle$ ] by simp
  then have  $C * \text{dist}\ x\ y - \text{dist} (f\ x) (f\ y) \geq 0$  using LIMSEQ_le_const[OF  $l$ ,
of 0] by auto
  then show  $\text{dist} (f\ x) (f\ y) \leq C * \text{dist}\ x\ y$  by auto
qed

```

```

lemma lipschitz_on_Pair[lipschitz_intros]:
  assumes  $f$ :  $L$ -lipschitz_on  $A$   $f$ 
  assumes  $g$ :  $M$ -lipschitz_on  $A$   $g$ 
  shows  $(\text{sqrt} (L^2 + M^2))$ -lipschitz_on  $A$   $(\lambda a. (f\ a, g\ a))$ 
proof (rule lipschitz_onI, goal_cases)
  case (1  $x$   $y$ )
  have  $\text{dist} (f\ x, g\ x) (f\ y, g\ y) = \text{sqrt} ((\text{dist} (f\ x) (f\ y))^2 + (\text{dist} (g\ x) (g\ y))^2)$ 
    by (auto simp add: dist_Pair_Pair real_le_sqrt)
  also have  $\dots \leq \text{sqrt} ((L * \text{dist}\ x\ y)^2 + (M * \text{dist}\ x\ y)^2)$ 
    by (auto intro!: real_sqrt_le_mono add_mono power_mono 1 lipschitz_onD  $f$ 
 $g$ )
  also have  $\dots \leq \text{sqrt} (L^2 + M^2) * \text{dist}\ x\ y$ 
    by (auto simp: power_mult_distrib ring_distrib[symmetric] real_sqrt_mult)
  finally show ?case .
qed simp

```

```

lemma lipschitz_extend_closure:
  fixes  $f$ ::('a::metric_space)  $\Rightarrow$  ('b::complete_space)
  assumes  $C$ -lipschitz_on  $U$   $f$ 
  shows  $\exists g. C$ -lipschitz_on (closure  $U$ )  $g \wedge (\forall x \in U. g\ x = f\ x)$ 
proof -
  obtain  $g$  where  $g$ :  $\bigwedge x. x \in U \implies g\ x = f\ x$  uniformly_continuous_on (closure
 $U$ )  $g$ 
  using uniformly_continuous_on_extension_on_closure[OF lipschitz_on_uniformly_continuous[OF
assms]] by metis

```

```

have C-lipschitz_on (closure U) g
  apply (rule lipschitz_on_closure, rule lipschitz_on_transform[OF assms])
  using g uniformly_continuous_imp_continuous[OF g(2)] by auto
  then show ?thesis using g(1) by auto
qed

```

```

lemma (in bounded_linear) lipschitz_boundE:
  obtains B where B-lipschitz_on A f
proof -
  from nonneg_bounded
  obtain B where B: B ≥ 0 ∧ x. norm (f x) ≤ B * norm x
  by (auto simp: ac_simps)
  have B-lipschitz_on A f
  by (auto intro!: lipschitz_onI B simp: dist_norm diff[symmetric])
  thus ?thesis ..
qed

```

9.33.1 Local Lipschitz continuity

Given a function defined on a real interval, it is Lipschitz-continuous if and only if it is locally so, as proved in the following lemmas. It is useful especially for piecewise-defined functions: if each piece is Lipschitz, then so is the whole function. The same goes for functions defined on geodesic spaces, or more generally on geodesic subsets in a metric space (for instance convex subsets in a real vector space), and this follows readily from the real case, but we will not prove it explicitly.

We give several variations around this statement. This is essentially a connectedness argument.

```

lemma locally_lipschitz_imp_lipschitz_aux:
  fixes f::real ⇒ ('a::metric_space)
  assumes a ≤ b
  continuous_on {a..b} f
   $\bigwedge x. x \in \{a..<b\} \implies \exists y \in \{x<..b\}. \text{dist } (f y) (f x) \leq M * (y-x)$ 
  shows dist (f b) (f a) ≤ M * (b-a)
proof -
  define A where A = {x ∈ {a..b}. dist (f x) (f a) ≤ M * (x-a)}
  have *: A = (λx. M * (x-a) - dist (f x) (f a))⁻¹{0..} ∩ {a..b}
  unfolding A_def by auto
  have a ∈ A unfolding A_def using <a ≤ b> by auto
  then have A ≠ {} by auto
  moreover have bdd_above A unfolding A_def by auto
  moreover have closed A unfolding * by (rule closed_vimage_Int, auto intro!: continuous_intros assms)
  ultimately have Sup A ∈ A by (rule closed_contains_Sup)
  have Sup A = b
  proof (rule ccontr)
  assume Sup A ≠ b
  define x where x = Sup A

```

```

have  $I: \text{dist } (f x) (f a) \leq M * (x-a)$  using  $\langle \text{Sup } A \in A \rangle x\_def A\_def$  by auto
have  $x \in \{a..<b\}$  unfolding  $x\_def$  using  $\langle \text{Sup } A \in A \rangle \langle \text{Sup } A \neq b \rangle A\_def$ 
by auto
then obtain  $y$  where  $J: y \in \{x<..b\}$   $\text{dist } (f y) (f x) \leq M * (y-x)$  using
assms(3) by blast
have  $\text{dist } (f y) (f a) \leq \text{dist } (f y) (f x) + \text{dist } (f x) (f a)$  by (rule dist_triangle)
also have  $\dots \leq M * (y-x) + M * (x-a)$  using  $I J(2)$  by auto
finally have  $\text{dist } (f y) (f a) \leq M * (y-a)$  by (auto simp add: algebra_simps)
then have  $y \in A$  unfolding  $A\_def$  using  $\langle y \in \{x<..b\} \rangle \langle x \in \{a..<b\} \rangle$  by
auto
then have  $y \leq \text{Sup } A$  by (rule cSup_upper, auto simp: A_def)
then show False using  $\langle y \in \{x<..b\} \rangle x\_def$  by auto
qed
then show ?thesis using  $\langle \text{Sup } A \in A \rangle A\_def$  by auto
qed

```

lemma *locally_lipschitz_imp_lipschitz*:

```

fixes  $f::\text{real} \Rightarrow ('a::\text{metric\_space})$ 
assumes continuous_on  $\{a..b\}$   $f$ 
 $\bigwedge x y. x \in \{a..<b\} \implies y > x \implies \exists z \in \{x<..y\}. \text{dist } (f z) (f x) \leq M * (z-x)$ 
 $M \geq 0$ 
shows lipschitz_on  $M$   $\{a..b\}$   $f$ 
proof (rule lipschitz_onI[OF _  $\langle M \geq 0 \rangle$ ])
have  $*$ :  $\text{dist } (f t) (f s) \leq M * (t-s)$  if  $s \leq t$   $s \in \{a..b\}$   $t \in \{a..b\}$  for  $s t$ 
proof (rule locally_lipschitz_imp_lipschitz_aux, simp add:  $\langle s \leq t \rangle$ )
show continuous_on  $\{s..t\}$   $f$  using continuous_on_subset[OF assms(1)] that
by auto
fix  $x$  assume  $x \in \{s..<t\}$ 
then have  $x \in \{a..<b\}$  using that by auto
show  $\exists z \in \{x<..t\}. \text{dist } (f z) (f x) \leq M * (z-x)$ 
using assms(2)[OF  $\langle x \in \{a..<b\} \rangle, \text{of } t \rangle \langle x \in \{s..<t\} \rangle$  by auto
qed
fix  $x y$  assume  $x \in \{a..b\}$   $y \in \{a..b\}$ 
consider  $x \leq y \mid y < x$  by linarith
then show  $\text{dist } (f x) (f y) \leq M * \text{dist } x y$ 
apply (cases)
using  $*[OF\_ \langle x \in \{a..b\} \rangle \langle y \in \{a..b\} \rangle] * [OF\_ \langle y \in \{a..b\} \rangle \langle x \in \{a..b\} \rangle]$ 
by (auto simp add: dist_commute dist_real_def)
qed

```

We deduce that if a function is Lipschitz on finitely many closed sets on the real line, then it is Lipschitz on any interval contained in their union. The difficulty in the proof is to show that any point z in this interval (except the maximum) has a point arbitrarily close to it on its right which is contained in a common initial closed set. Otherwise, we show that there is a small interval (z, T) which does not intersect any of the initial closed sets, a contradiction.

proposition *lipschitz_on_closed_Union*:
assumes $\bigwedge i. i \in I \implies \text{lipschitz_on } M (U i) f$
 $\bigwedge i. i \in I \implies \text{closed } (U i)$
finite *I*
 $M \geq 0$
 $\{u..(v::\text{real})\} \subseteq (\bigcup i \in I. U i)$
shows *lipschitz_on* $M \{u..v\} f$
proof (rule *locally_lipschitz_imp_lipschitz*[*OF* __ $\langle M \geq 0 \rangle$])
have *: *continuous_on* $(U i) f$ **if** $i \in I$ **for** i
by (rule *lipschitz_on_continuous_on*[*OF* *assms*(1)][*OF* $\langle i \in I \rangle$])
have *continuous_on* $(\bigcup i \in I. U i) f$
apply (rule *continuous_on_closed_Union*) **using** $\langle \text{finite } I \rangle * \text{assms}(2)$ **by** *auto*
then show *continuous_on* $\{u..v\} f$
using $\langle \{u..(v::\text{real})\} \subseteq (\bigcup i \in I. U i) \rangle$ *continuous_on_subset* **by** *auto*

fix $z \in Z$ **assume** $z: z \in \{u..v\} z < Z$
then have $u \leq v$ **by** *auto*
define T **where** $T = \min Z v$
then have $T: T > z T \leq v T \geq u T \leq Z$ **using** z **by** *auto*
define A **where** $A = (\bigcup i \in I \cap \{i. U i \cap \{z <..T\} \neq \{\}\}. U i \cap \{z..T\})$
have $a: \text{closed } A$
unfolding A_def **apply** (rule *closed_UN*) **using** $\langle \text{finite } I \rangle \langle \bigwedge i. i \in I \implies \text{closed } (U i) \rangle$ **by** *auto*
have $b: \text{bdd_below } A$ **unfolding** A_def **using** $\langle \text{finite } I \rangle$ **by** *auto*
have $\exists i \in I. T \in U i$ **using** $\langle \{u..v\} \subseteq (\bigcup i \in I. U i) \rangle$ T **by** *auto*
then have $c: T \in A$ **unfolding** A_def **using** T **by** (*auto*, *fastforce*)
have $\text{Inf } A \geq z$
apply (rule *cInf_greatest*, *auto*) **using** c **unfolding** A_def **by** *auto*
moreover have $\text{Inf } A \leq z$
proof (rule *ccontr*)
assume $\neg(\text{Inf } A \leq z)$
then obtain w **where** $w: w > z w < \text{Inf } A$ **by** (*meson* *dense* *not_le_imp_less*)
have $\text{Inf } A \leq T$ **using** $a b c$ **by** (*simp* *add*: *cInf_lower*)
then have $w \leq T$ **using** w **by** *auto*
then have $w \in \{u..v\}$ **using** $w \langle z \in \{u..v\} \rangle$ T **by** *auto*
then obtain j **where** $j: j \in I w \in U j$ **using** $\langle \{u..v\} \subseteq (\bigcup i \in I. U i) \rangle$ **by** *fastforce*
then have $w \in U j \cap \{z..T\} U j \cap \{z <..T\} \neq \{\}$ **using** $j T w \langle w \leq T \rangle$ **by** *auto*
then have $w \in A$ **unfolding** A_def **using** $\langle j \in I \rangle$ **by** *auto*
then have $\text{Inf } A \leq w$ **using** $a b$ **by** (*simp* *add*: *cInf_lower*)
then show *False* **using** w **by** *auto*

qed
ultimately have $\text{Inf } A = z$ **by** *simp*
moreover have $\text{Inf } A \in A$
apply (rule *closed_contains_Inf*) **using** $a b c$ **by** *auto*
ultimately have $z \in A$ **by** *simp*
then obtain i **where** $i: i \in I U i \cap \{z <..T\} \neq \{\} z \in U i$ **unfolding** A_def **by** *auto*

then obtain t **where** $t \in U i \cap \{z <..T\}$ **by** *blast*
then have $\text{dist } (f t) (f z) \leq M * (t - z)$
using *lipschitz_onD(1)[OF assms(1)[of i], of t z] i dist_real_def* **by** *auto*
then show $\exists t \in \{z <..Z\}. \text{dist } (f t) (f z) \leq M * (t - z)$ **using** $\langle T \leq Z \rangle \langle t \in U i \cap \{z <..T\} \rangle$ **by** *auto*
qed

9.33.2 Local Lipschitz continuity (uniform for a family of functions)

definition *local_lipschitz::*

'a::metric_space set \Rightarrow 'b::metric_space set \Rightarrow ('a \Rightarrow 'b \Rightarrow 'c::metric_space) \Rightarrow bool

where

local_lipschitz T X f $\equiv \forall x \in X. \forall t \in T.$

$\exists u > 0. \exists L. \forall t \in \text{cball } t u \cap T. L\text{-lipschitz_on } (\text{cball } x u \cap X) (f t)$

lemma *local_lipschitzI:*

assumes $\bigwedge t x. t \in T \implies x \in X \implies \exists u > 0. \exists L. \forall t \in \text{cball } t u \cap T. L\text{-lipschitz_on } (\text{cball } x u \cap X) (f t)$

shows *local_lipschitz T X f*

using *assms*

unfolding *local_lipschitz_def*

by *auto*

lemma *local_lipschitzE:*

assumes *local_lipschitz: local_lipschitz T X f*

assumes $t \in T x \in X$

obtains $u L$ **where** $u > 0 \bigwedge s. s \in \text{cball } t u \cap T \implies L\text{-lipschitz_on } (\text{cball } x u \cap X) (f s)$

using *assms local_lipschitz_def*

by *metis*

lemma *local_lipschitz_continuous_on:*

assumes *local_lipschitz: local_lipschitz T X f*

assumes $t \in T$

shows *continuous_on X (f t)*

unfolding *continuous_on_def*

proof *safe*

fix x **assume** $x \in X$

from *local_lipschitzE[OF local_lipschitz $\langle t \in T \rangle \langle x \in X \rangle]$* **obtain** $u L$

where $0 < u$

and $L: \bigwedge s. s \in \text{cball } t u \cap T \implies L\text{-lipschitz_on } (\text{cball } x u \cap X) (f s)$

by *metis*

have $x \in \text{ball } x u$ **using** $\langle 0 < u \rangle$ **by** *simp*

from *lipschitz_on_continuous_on[OF L]*

have *tendsto: (f t \longrightarrow f t x) (at x within cball x u \cap X)*

using $\langle 0 < u \rangle \langle x \in X \rangle \langle t \in T \rangle$

by (*auto simp: continuous_on_def*)

3814

moreover have $\forall_F xa \text{ in } at \ x. (xa \in cball \ x \ u \cap X) = (xa \in X)$
using *eventually_at_ball*[*OF* $\langle 0 < u \rangle$, *of x UNIV*]
by *eventually_elim auto*
ultimately show $(f \ t \longrightarrow f \ t \ x)$ (*at x within X*)
by (*rule Lim_transform_within_set*)
qed

lemma

local_lipschitz_compose1:
assumes *ll*: *local_lipschitz* $(g \ ' \ T) \ X \ (\lambda t. f \ t)$
assumes *g*: *continuous_on* $T \ g$
shows *local_lipschitz* $T \ X \ (\lambda t. f \ (g \ t))$
proof (*rule local_lipschitzI*)
fix $t \ x$
assume $t \in T \ x \in X$
then have $g \ t \in g \ ' \ T$ **by** *simp*
from *local_lipschitzE*[*OF* *assms(1)* *this* $\langle x \in X \rangle$]
obtain $u \ L$ **where** $0 < u$ **and** l : $(\bigwedge s. s \in cball \ (g \ t) \ u \cap g \ ' \ T \Longrightarrow L\text{-lipschitz_on} \ (cball \ x \ u \cap X) \ (f \ s))$
by *auto*
from g [*unfolded continuous_on_eq_continuous_within*, *rule_format*, *OF* $\langle t \in T \rangle$,
unfolded continuous_within_eps_delta, *rule_format*, *OF* $\langle 0 < u \rangle$]
obtain d **where** $d > 0 \ \bigwedge x'. x' \in T \Longrightarrow dist \ x' \ t < d \Longrightarrow dist \ (g \ x') \ (g \ t) < u$
by (*auto*)
show $\exists u > 0. \exists L. \forall t \in cball \ t \ u \cap T. L\text{-lipschitz_on} \ (cball \ x \ u \cap X) \ (f \ (g \ t))$
using $d \ \langle 0 < u \rangle$
by (*fastforce intro: exI*[**where** $x = (\min \ d \ u) / 2$] *exI*[**where** $x = L$]
intro!: *less_imp_le*[*OF* $d(2)$] *lipschitz_on_subset*[*OF* l] *simp: dist_commute*)
qed

context

fixes $T::'a::metric_space \ set$ **and** $X \ f$
assumes *local_lipschitz*: *local_lipschitz* $T \ X \ f$
begin

lemma *continuous_on_TimesI*:

assumes y : $\bigwedge x. x \in X \Longrightarrow continuous_on \ T \ (\lambda t. f \ t \ x)$
shows *continuous_on* $(T \times X) \ (\lambda(t, x). f \ t \ x)$
unfolding *continuous_on_iff*
proof (*safe, simp*)
fix $a \ b$ **and** $e::real$
assume H : $a \in T \ b \in X \ 0 < e$
hence $0 < e/2$ **by** *simp*
from y [*unfolded continuous_on_iff*, *OF* $\langle b \in X \rangle$, *rule_format*, *OF* $\langle a \in T \rangle \ \langle 0 < e/2 \rangle$]
obtain d **where** $d > 0 \ \bigwedge t. t \in T \Longrightarrow dist \ t \ a < d \Longrightarrow dist \ (f \ t \ b) \ (f \ a \ b) < e/2$
by *auto*

```

from ⟨a : T⟩ ⟨b ∈ X⟩
obtain u L where u: 0 < u
  and L:  $\bigwedge t. t \in \text{cball } a \ u \cap T \implies L\text{-lipschitz\_on } (\text{cball } b \ u \cap X) (f \ t)$ 
  by (erule local_lipschitzE[OF local_lipschitz])

have a ∈ cball a u ∩ T by (auto simp: ⟨0 < u⟩ ⟨a ∈ T⟩ less_imp_le)
from lipschitz_on_nonneg[OF L[OF ⟨a ∈ cball __ ∩ __⟩]] have 0 ≤ L .

let ?d = Min {d, u, (e/2/(L + 1))}
show  $\exists d > 0. \forall x \in T. \forall y \in X. \text{dist } (x, y) (a, b) < d \implies \text{dist } (f \ x \ y) (f \ a \ b) < e$ 
proof (rule exI[where x = ?d], safe)
  show 0 < ?d
    using ⟨0 ≤ L⟩ ⟨0 < u⟩ ⟨0 < e⟩ ⟨0 < d⟩
    by (auto intro!: divide_pos_pos )
  fix x y
  assume x ∈ T y ∈ X
  assume dist_less: dist (x, y) (a, b) < ?d
  have dist y b ≤ dist (x, y) (a, b)
    using dist_snd_le[of (x, y) (a, b)]
    by auto
  also
  note dist_less
  also
  {
    note calculation
    also have ?d ≤ u by simp
    finally have dist y b < u .
  }
  have ?d ≤ e/2/(L + 1) by simp
  also have (L + 1) * ... ≤ e / 2
    using ⟨0 < e⟩ ⟨L ≥ 0⟩
    by (auto simp: field_split_simps)
  finally have le1: (L + 1) * dist y b < e / 2 using ⟨L ≥ 0⟩ by simp

  have dist x a ≤ dist (x, y) (a, b)
    using dist_fst_le[of (x, y) (a, b)]
    by auto
  also note dist_less
  finally have dist x a < ?d .
  also have ?d ≤ d by simp
  finally have dist x a < d .
  note ⟨dist x a < ?d⟩
  also have ?d ≤ u by simp
  finally have dist x a < u .
  then have x ∈ cball a u ∩ T
    using ⟨x ∈ T⟩
    by (auto simp: dist_commute)
  have dist (f x y) (f a b) ≤ dist (f x y) (f x b) + dist (f x b) (f a b)

```

```

    by (rule dist_triangle)
  also have  $(L + 1)$ -lipschitz_on (cball b u  $\cap$  X) (f x)
    using L[OF  $\langle x \in \text{cball } a \ u \ \cap \ T \rangle$ ]
    by (rule lipschitz_on_le) simp
  then have  $\text{dist } (f \ x \ y) \ (f \ x \ b) \leq (L + 1) * \text{dist } y \ b$ 
    apply (rule lipschitz_onD)
  subgoal
    using  $\langle y \in X \rangle \ \langle \text{dist } y \ b < u \rangle$ 
    by (simp add: dist_commute)
  subgoal
    using  $\langle 0 < u \rangle \ \langle b \in X \rangle$ 
    by simp
  done
  also have  $(L + 1) * \text{dist } y \ b \leq e / 2$ 
    using le1  $\langle 0 \leq L \rangle$  by simp
  also have  $\text{dist } (f \ x \ b) \ (f \ a \ b) < e / 2$ 
    by (rule d; fact)
  also have  $e / 2 + e / 2 = e$  by simp
  finally show  $\text{dist } (f \ x \ y) \ (f \ a \ b) < e$  by simp
qed
qed

```

lemma *local_lipschitz_compact_implies_lipschitz*:

```

  assumes compact X compact T
  assumes cont:  $\bigwedge x. x \in X \implies \text{continuous\_on } T \ (\lambda t. f \ t \ x)$ 
  obtains L where  $\bigwedge t. t \in T \implies L$ -lipschitz_on X (f t)
proof -
  {
    assume *:  $\bigwedge n::\text{nat}. \neg(\forall t \in T. n$ -lipschitz_on X (f t))
    {
      fix n::nat
      from *[of n] have  $\exists x \ y \ t. t \in T \wedge x \in X \wedge y \in X \wedge \text{dist } (f \ t \ y) \ (f \ t \ x) > n$ 
      * dist y x
        by (force simp: lipschitz_on_def)
    } then obtain t and x y::nat  $\Rightarrow$  'b where xy:  $\bigwedge n. x \ n \in X \ \bigwedge n. y \ n \in X$ 
      and t:  $\bigwedge n. t \ n \in T$ 
      and d:  $\bigwedge n. \text{dist } (f \ (t \ n) \ (y \ n)) \ (f \ (t \ n) \ (x \ n)) > n * \text{dist } (y \ n) \ (x \ n)$ 
      by metis
    from xy assms obtain lx rx where lx':  $lx \in X \ \text{strict\_mono } (rx :: \text{nat} \Rightarrow \text{nat})$ 
    (x o rx)  $\longrightarrow$  lx
      by (metis compact_def)
    with xy have  $\bigwedge n. (y \ o \ rx) \ n \in X$  by auto
    with assms obtain ly ry where ly':  $ly \in X \ \text{strict\_mono } (ry :: \text{nat} \Rightarrow \text{nat})$  ((y
    o rx) o ry)  $\longrightarrow$  ly
      by (metis compact_def)
    with t have  $\bigwedge n. ((t \ o \ rx) \ o \ ry) \ n \in T$  by simp
    with assms obtain lt rt where lt':  $lt \in T \ \text{strict\_mono } (rt :: \text{nat} \Rightarrow \text{nat})$  (((t
    o rx) o ry) o rt)  $\longrightarrow$  lt
      by (metis compact_def)
  }

```

```

from  $lx' ly'$ 
have  $lx: (x \circ (rx \circ ry \circ rt)) \longrightarrow lx$  (is  $?x \longrightarrow \_$ )
  and  $ly: (y \circ (rx \circ ry \circ rt)) \longrightarrow ly$  (is  $?y \longrightarrow \_$ )
  and  $lt: (t \circ (rx \circ ry \circ rt)) \longrightarrow lt$  (is  $?t \longrightarrow \_$ )
  subgoal by (simp add: LIMSEQ_subseq LIMSEQ_o_assoc lt'(2))
  subgoal by (simp add: LIMSEQ_subseq LIMSEQ_ly'(3) o_assoc lt'(2))
  subgoal by (simp add: o_assoc lt'(3))
  done
hence  $(\lambda n. dist (?y n) (?x n)) \longrightarrow dist ly lx$ 
  by (metis tendsto_dist)
moreover
let  $?S = (\lambda(t, x). f t x) ' (T \times X)$ 
have eventually  $(\lambda n::nat. n > 0)$  sequentially
  by (metis eventually_at_top_dense)
hence eventually  $(\lambda n. norm (dist (?y n) (?x n)) \leq norm (|diameter ?S| / n)$ 
* 1) sequentially
proof eventually_elim
  case (elim n)
    have  $0 < rx (ry (rt n))$  using  $\langle 0 < n \rangle$ 
      by (metis dual_order.strict_trans1 lt'(2) lx'(2) ly'(2) seq_suble)
    have compact: compact ?S
      by (auto intro!: compact_continuous_image continuous_on_subset[OF
continuous_on_TimesI
        compact_Times  $\langle compact X \rangle \langle compact T \rangle cont$ )
    have  $norm (dist (?y n) (?x n)) = dist (?y n) (?x n)$  by simp
    also
from this elim d[of rx (ry (rt n))]
have  $\dots < dist (f (?t n) (?y n)) (f (?t n) (?x n)) / rx (ry (rt (n)))$ 
      using  $lx'(2) ly'(2) lt'(2) \langle 0 < rx \_ \rangle$ 
      by (auto simp add: field_split_simps strict_mono_def)
    also have  $\dots \leq diameter ?S / n$ 
proof (rule frac_le)
  show  $diameter ?S \geq 0$ 
    using compact compact_imp_bounded diameter_ge_0 by blast
  show  $dist (f (?t n) (?y n)) (f (?t n) (?x n)) \leq diameter ((\lambda(t,x). f t x) ' (T \times X))$ 
by (metis (no_types) compact compact_imp_bounded diameter_bounded_bound
image_eqI mem_Sigma_iff o_apply split_conv t xy(1) xy(2))
  show  $real n \leq real (rx (ry (rt n)))$ 
by (meson le_trans lt'(2) lx'(2) ly'(2) of_nat_mono strict_mono_imp_increasing)
qed (use  $\langle n > 0 \rangle$  in auto)
also have  $\dots \leq abs (diameter ?S) / n$ 
  by (auto intro!: divide_right_mono)
finally show ?case by simp
qed
with  $\_$  have  $(\lambda n. dist (?y n) (?x n)) \longrightarrow 0$ 
by (rule tendsto_0_le)
(metis tendsto_divide_0[OF tendsto_const] filterlim_at_top_imp_at_infinity
  filterlim_real_sequentially)

```

```

ultimately have  $lx = ly$ 
  using LIMSEQ_unique by fastforce
with assms  $lx'$  have  $lx \in X$  by auto
from  $\langle lt \in T \rangle$  this obtain  $u L$  where  $L: u > 0 \wedge t. t \in cball\ lt\ u \cap T \implies$ 
 $L$ -lipschitz_on  $(cball\ lx\ u \cap X)$   $(f\ t)$ 
  by (erule local_lipschitzE[OF local_lipschitz])
hence  $L \geq 0$  by (force intro!: lipschitz_on_nonneg  $\langle lt \in T \rangle$ )

from  $L\ lt\ ly\ lx$   $\langle lx = ly \rangle$ 
have
  eventually  $(\lambda n. ?t\ n \in ball\ lt\ u)$  sequentially
  eventually  $(\lambda n. ?y\ n \in ball\ lx\ u)$  sequentially
  eventually  $(\lambda n. ?x\ n \in ball\ lx\ u)$  sequentially
  by (auto simp: dist_commute Lim)
moreover have eventually  $(\lambda n. n > L)$  sequentially
  by (metis filterlim_at_top_dense filterlim_real_sequentially)
ultimately
have eventually  $(\lambda_. False)$  sequentially
proof eventually_elim
  case (elim  $n$ )
  hence  $dist\ (f\ (?t\ n)\ (?y\ n))\ (f\ (?t\ n)\ (?x\ n)) \leq L * dist\ (?y\ n)\ (?x\ n)$ 
    using assms  $xy\ t$ 
    unfolding dist_norm[symmetric]
    by (intro lipschitz_onD[OF  $L(2)$ ]) (auto)
  also have  $\dots \leq n * dist\ (?y\ n)\ (?x\ n)$ 
    using elim by (intro mult_right_mono) auto
  also have  $\dots \leq rx\ (ry\ (rt\ n)) * dist\ (?y\ n)\ (?x\ n)$ 
    by (intro mult_right_mono[OF _ zero_le_dist])
      (meson  $lt'(2)\ lx'(2)\ ly'(2)$  of_nat_le_iff order_trans seq_suble)
  also have  $\dots < dist\ (f\ (?t\ n)\ (?y\ n))\ (f\ (?t\ n)\ (?x\ n))$ 
    by (auto intro!:  $d$ )
  finally show ?case by simp
qed
hence False
  by simp
} then obtain  $L$  where  $\wedge t. t \in T \implies L$ -lipschitz_on  $X$   $(f\ t)$ 
  by metis
thus ?thesis ..
qed

```

lemma local_lipschitz_subset:

assumes $S \subseteq T\ Y \subseteq X$

shows local_lipschitz $S\ Y\ f$

proof (rule local_lipschitzI)

fix $t\ x$ assume $t \in S\ x \in Y$

then have $t \in T\ x \in X$ using *assms* by auto

from local_lipschitzE[OF local_lipschitz, OF this]

obtain $u\ L$ where $0 < u$ and $L: \wedge s. s \in cball\ t\ u \cap T \implies L$ -lipschitz_on $(cball\ x\ u \cap X)$ $(f\ s)$

```

  by blast
  show  $\exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap S. L\text{-lipschitz\_on } (\text{cball } x \ u \cap Y) (f \ t)$ 
  using assms
  by (auto intro: exI[where  $x=u$ ] exI[where  $x=L$ ]
      intro!: u_lipschitz_on_subset[OF Int_mono[OF order_refl  $\langle Y \subseteq X \rangle$ ]] L)
qed
end

```

lemma *local_lipschitz_minus*:

```

  fixes  $f::'a::\text{metric\_space} \Rightarrow 'b::\text{metric\_space} \Rightarrow 'c::\text{real\_normed\_vector}$ 
  shows local_lipschitz  $T \ X \ (\lambda t \ x. - f \ t \ x) = \text{local\_lipschitz } T \ X \ f$ 
  by (auto simp: local_lipschitz_def lipschitz_on_minus)

```

lemma *local_lipschitz_PairI*:

```

  assumes  $f: \text{local\_lipschitz } A \ B \ (\lambda a \ b. f \ a \ b)$ 
  assumes  $g: \text{local\_lipschitz } A \ B \ (\lambda a \ b. g \ a \ b)$ 
  shows local_lipschitz  $A \ B \ (\lambda a \ b. (f \ a \ b, g \ a \ b))$ 
proof (rule local_lipschitzI)
  fix  $t \ x$  assume  $t \in A \ x \in B$ 
  from local_lipschitzE[OF  $f$  this] local_lipschitzE[OF  $g$  this]
  obtain  $u \ L \ v \ M$  where  $0 < u \ (\bigwedge s. s \in \text{cball } t \ u \cap A \Longrightarrow L\text{-lipschitz\_on } (\text{cball } x \ u \cap B) (f \ s))$ 
  and  $0 < v \ (\bigwedge s. s \in \text{cball } t \ v \cap A \Longrightarrow M\text{-lipschitz\_on } (\text{cball } x \ v \cap B) (g \ s))$ 
  by metis
  then show  $\exists u > 0. \exists L. \forall t \in \text{cball } t \ u \cap A. L\text{-lipschitz\_on } (\text{cball } x \ u \cap B) (\lambda b. (f \ t \ b, g \ t \ b))$ 
  by (intro exI[where  $x=\min \ u \ v$ ]
      (force intro: lipschitz_on_subset intro!: lipschitz_on_Pair)
  qed

```

lemma *local_lipschitz_constI*: *local_lipschitz* $S \ T \ (\lambda t \ x. f \ t)$

```

  by (auto simp: intro!: local_lipschitzI lipschitz_on_constant intro: exI[where  $x=1$ ])

```

lemma (*in bounded_linear*) *local_lipschitzI*:

```

  shows local_lipschitz  $A \ B \ (\lambda_. f)$ 
proof (rule local_lipschitzI, goal_cases)
  case (1  $t \ x$ )
  from lipschitz_boundE[of  $(\text{cball } x \ 1 \cap B)$ ] obtain  $C$  where  $C\text{-lipschitz\_on } (\text{cball } x \ 1 \cap B) \ f$  by auto
  then show ?case
  by (auto intro: exI[where  $x=1$ ])
qed

```

proposition *c1_implies_local_lipschitz*:

```

  fixes  $T::\text{real set}$  and  $X::'a::\{\text{banach, heine\_borel}\}$  set
  and  $f::\text{real} \Rightarrow 'a \Rightarrow 'a$ 
  assumes  $f': \bigwedge t \ x. t \in T \Longrightarrow x \in X \Longrightarrow (f \ t \ \text{has\_derivative } \text{blinfun\_apply } (f'$ 
```

```

(t, x)) (at x)
  assumes cont_f': continuous_on (T × X) f'
  assumes open T
  assumes open X
  shows local_lipschitz T X f
proof (rule local_lipschitzI)
  fix t x
  assume t ∈ T x ∈ X
  from open_contains_cball[THEN iffD1, OF ⟨open X⟩, rule_format, OF ⟨x ∈ X⟩]
  obtain u where u: u > 0 cball x u ⊆ X by auto
  moreover
  from open_contains_cball[THEN iffD1, OF ⟨open T⟩, rule_format, OF ⟨t ∈ T⟩]
  obtain v where v: v > 0 cball t v ⊆ T by auto
  ultimately
  have compact (cball t v × cball x u) cball t v × cball x u ⊆ T × X
    by (auto intro!: compact_Times)
  then have compact (f' ` (cball t v × cball x u))
    by (auto intro!: compact_continuous_image continuous_on_subset[OF cont_f'])
  then obtain B where B: B > 0 ∧ s y. s ∈ cball t v ⇒ y ∈ cball x u ⇒ norm
    (f' (s, y)) ≤ B
    by (auto dest!: compact_imp_bounded simp: bounded_pos)

  have lipschitz: B-lipschitz_on (cball x (min u v) ∩ X) (f s) if s: s ∈ cball t v
for s
  proof –
  note s
  also note ⟨cball t v ⊆ T⟩
  finally
  have deriv: ∧y. y ∈ cball x u ⇒ (f s has_derivative blinfun_apply (f' (s, y)))
    (at y within cball x u)
    using ⟨_ ⊆ X⟩
    by (auto intro!: has_derivative_at_withinI[OF f'])
  have norm (f s y - f s z) ≤ B * norm (y - z)
    if y ∈ cball x u z ∈ cball x u
    for y z
    using s that
    by (intro differentiable_bound[OF convex_cball deriv])
    (auto intro!: B simp: norm_blinfun.rep_eq[symmetric])
  then show ?thesis
    using ⟨0 < B⟩
    by (auto intro!: lipschitz_onI simp: dist_norm)
qed
show ∃u>0. ∃L. ∀t∈cball t u ∩ T. L-lipschitz_on (cball x u ∩ X) (f t)
  by (force intro: exI[where x=min u v] exI[where x=B] intro!: lipschitz simp:
u v)
qed

```



```

end
theory
  Multivariate_Analysis
imports
  Ordered_Euclidean_Space
  Determinants
  Cross3
  Lipschitz
  Starlike
begin

Entry point excluding integration and complex analysis.

end

```

9.34 Volume of a Simplex

```

theory Simplex_Content
imports Change_Of_Vars
begin

lemma fact_neq_top_enreal [simp]: fact n ≠ (top :: ennreal)
  by (induction n) (auto simp: ennreal_mult_eq_top_iff)

lemma ennreal_fact: ennreal (fact n) = fact n
  by (induction n) (auto simp: ennreal_mult algebra_simps ennreal_of_nat_eq_real_of_nat)

context
  fixes S :: 'a set ⇒ real ⇒ ('a ⇒ real) set
  defines S ≡ (λA t. {x. (∀ i ∈ A. 0 ≤ x i) ∧ sum x A ≤ t})
begin

lemma emeasure_std_simplex_aux_step:
  assumes b ∉ A finite A
  shows x(b := y) ∈ S (insert b A) t ⟷ y ∈ {0..t} ∧ x ∈ S A (t - y)
  using assms sum_nonneg[of A x] unfolding S_def
  by (force simp: sum_delta_notmem algebra_simps)

lemma emeasure_std_simplex_aux:
  fixes t :: real
  assumes finite (A :: 'a set) t ≥ 0
  shows emeasure (PiM A (λ_. lborel))
    (S A t ∩ space (PiM A (λ_. lborel))) = t ^ card A / fact (card A)
  using assms(1,2)
proof (induction arbitrary: t rule: finite_induct)
  case (empty t)
  thus ?case by (simp add: PiM_empty S_def)
next
  case (insert b A t)
  define n where n = Suc (card A)

```

```

have n_pos: n > 0 by (simp add: n_def)
let ?M = λA. (Pi_M A (λ_. lborel))
{
  fix A :: 'a set and t :: real assume finite A
  have S A t ∩ space (Pi_M A (λ_. lborel)) =
    Pi_E A (λ_. {0..}) ∩ (λx. sum x A) - ' {..t} ∩ space (Pi_M A (λ_. lborel))
  by (auto simp: S_def space_PiM)
  also have ... ∈ sets (Pi_M A (λ_. lborel))
  using ⟨finite A⟩ by measurable
  finally have S A t ∩ space (Pi_M A (λ_. lborel)) ∈ sets (Pi_M A (λ_. lborel)) .
} note meas [measurable] = this

interpret product_sigma_finite λ_. lborel
by standard
have emeasure (?M (insert b A)) (S (insert b A) t ∩ space (?M (insert b A)))
=
  nn_integral (?M (insert b A))
    (λx. indicator (S (insert b A) t ∩ space (?M (insert b A))) x)
  using insert.hyps by (subst nn_integral_indicator) auto
  also have ... = (∫+ y. ∫+ x. indicator (S (insert b A) t ∩ space (?M (insert
    b A)))
      (x(b := y)) ∂?M A ∂lborel)
  using insert.prem1 insert.hyps by (intro product_nn_integral_insert_rev) auto
  also have ... = (∫+ y. ∫+ x. indicator {0..t} y * indicator (S A (t - y) ∩
    space (?M A)) x
      ∂?M A ∂lborel)
  using insert.hyps insert.prem1 emeasure_std_simplex_aux_step[of b A]
  by (intro nn_integral_cong)
  (auto simp: fun_eq_iff indicator_def space_PiM Pi_E_def extensional_def)
  also have ... = (∫+ y. indicator {0..t} y * (∫+ x. indicator (S A (t - y) ∩
    space (?M A)) x
      ∂?M A ∂lborel) using ⟨finite A⟩
  by (subst nn_integral_cmult) auto
  also have ... = (∫+ y. indicator {0..t} y * emeasure (?M A) (S A (t - y) ∩
    space (?M A)) ∂lborel)
  using ⟨finite A⟩ by (subst nn_integral_indicator) auto
  also have ... = (∫+ y. indicator {0..t} y * (t - y) ^ card A / ennreal (fact
    (card A)) ∂lborel)
  using insert.IH by (intro nn_integral_cong) (auto simp: indicator_def di-
    vide_ennreal)
  also have ... = (∫+ y. indicator {0..t} y * (t - y) ^ card A ∂lborel) / ennreal
    (fact (card A))
  using ⟨finite A⟩ by (subst nn_integral_divide) auto
  also have (∫+ y. indicator {0..t} y * (t - y) ^ card A ∂lborel) =
    (∫+ y ∈ {0..t}. ennreal ((t - y) ^ (n - 1)) ∂lborel)
  by (intro nn_integral_cong) (auto simp: indicator_def n_def)
  also have ((λx. - ((t - x) ^ n / n)) has_real_derivative (t - x) ^ (n - 1))
    (at x)
  if x ∈ {0..t} for x by (rule derivative_eq_intros refl | simp add: n_pos)+

```

hence $(\int^+ y \in \{0..t\}. \text{ennreal } ((t - y) ^ (n - 1)) \partial \text{lborel}) =$
 $\text{ennreal } (-(t - t) ^ n / n - (-(t - 0) ^ n / n))$
using *insert.premis insert.hyps* **by** (*intro nn_integral_FTC_Icc*) *auto*
also have $\dots = \text{ennreal } (t ^ n / n)$ **using** *n_pos* **by** (*simp add: zero_power*)
also have $\dots / \text{ennreal } (\text{fact } (\text{card } A)) = \text{ennreal } (t ^ n / n / \text{fact } (\text{card } A))$
using *n_pos* $\langle t \geq 0 \rangle$ **by** (*subst divide_ennreal*) *auto*
also have $t ^ n / n / \text{fact } (\text{card } A) = t ^ n / \text{fact } n$
by (*simp add: n_def*)
also have $n = \text{card } (\text{insert } b A)$
using *insert.hyps* **by** (*subst card.insert_remove*) (*auto simp: n_def*)
finally show *?case* .
qed

end

lemma *emeasure_std_simplex*:

$\text{emeasure lborel } (\text{convex hull } (\text{insert } 0 \text{ Basis } :: 'a :: \text{euclidean_space set})) =$
 $\text{ennreal } (1 / \text{fact } \text{DIM}('a))$

proof –

have $\text{emeasure lborel } \{x :: 'a. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } ((\cdot) x) \text{ Basis} \leq 1\} =$
 $\text{emeasure } (\text{distr } (\text{Pi}_M \text{ Basis } (\lambda b. \text{lborel})) \text{ borel } (\lambda f. \sum_{b \in \text{Basis}. f b} *R$
 $b))$

$\{x :: 'a. (\forall i \in \text{Basis}. 0 \leq x \cdot i) \wedge \text{sum } ((\cdot) x) \text{ Basis} \leq 1\}$

by (*subst lborel_eq*) *simp*

also have $\dots = \text{emeasure } (\text{Pi}_M \text{ Basis } (\lambda b. \text{lborel}))$

$(\{y :: 'a \Rightarrow \text{real}. (\forall i \in \text{Basis}. 0 \leq y i) \wedge \text{sum } y \text{ Basis} \leq 1\} \cap$
 $\text{space } (\text{Pi}_M \text{ Basis } (\lambda b. \text{lborel})))$

by (*subst emeasure_distr*) *auto*

also have $\dots = \text{ennreal } (1 / \text{fact } \text{DIM}('a))$

by (*subst emeasure_std_simplex_aux*) *auto*

finally show *?thesis* **by** (*simp only: std_simplex*)

qed

theorem *content_std_simplex*:

$\text{measure lborel } (\text{convex hull } (\text{insert } 0 \text{ Basis } :: 'a :: \text{euclidean_space set})) =$
 $1 / \text{fact } \text{DIM}('a)$

by (*simp add: measure_def emeasure_std_simplex*)

proposition *measure_lebesgue_linear_transformation*:

fixes $A :: (\text{real} ^ 'n :: \{\text{finite}, \text{wellorder}\}) \text{ set}$

fixes $f :: _ \Rightarrow \text{real} ^ 'n :: \{\text{finite}, \text{wellorder}\}$

assumes *bounded A A ∈ sets lebesgue linear f*

shows $\text{measure lebesgue } (f ' A) = |\det (\text{matrix } f)| * \text{measure lebesgue } A$

proof –

from *assms* **have** [*intro*]: $A \in \text{lmeasurable}$

by (*intro bounded_set_imp_lmeasurable*) *auto*

hence [*intro*]: $f ' A \in \text{lmeasurable}$

by (*intro lmeasure_integral_measurable_linear_image assms*)

```

have measure lebesgue (f ‘ A) = integral (f ‘ A) (λ_. 1)
  by (intro lmeasure_integral measurable_linear_image assms) auto
also have ... = integral (f ‘ A) (λ_. 1 :: real ^ 1) $ 0
  by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
also have ... = |det (matrix f)| * integral A (λx. 1 :: real ^ 1) $ 0
  using assms
  by (subst integral_change_of_variables_linear)
    (auto simp: o_def absolutely_integrable_on_def intro: integrable_on_const)
also have integral A (λx. 1 :: real ^ 1) $ 0 = integral A (λx. 1)
  by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
also have ... = measure lebesgue A
  by (intro lmeasure_integral [symmetric]) auto
finally show ?thesis .

```

qed

theorem *content_simplex*:

```

fixes X :: (real ^ 'n :: {finite, wellorder}) set and f :: 'n :: _ ⇒ real ^ ('n :: _)
assumes finite X card X = Suc CARD('n) and x0: x0 ∈ X and bij: bij_betw f
  UNIV (X - {x0})
defines M ≡ (χ i. χ j. f j $ i - x0 $ i)
shows content (convex hull X) = |det M| / fact (CARD('n))

```

proof –

```

define g where g = (λx. M *v x)
have [simp]: M *v axis i 1 = f i - x0 for i :: 'n
  by (simp add: M_def matrix_vector_mult_basis column_def vec_eq_iff)
define std where std = (convex hull insert 0 Basis :: (real ^ 'n :: _) set)
have compact: compact std unfolding std_def
  by (intro finite_imp_compact_convex_hull) auto

```

```

have measure lebesgue (convex hull X) = measure lebesgue (((+) (-x0)) ‘ (convex hull X))

```

```

  by (rule measure_translation [symmetric])
also have ((+) (-x0)) ‘ (convex hull X) = convex hull (((+) (-x0)) ‘ X)
  by (rule convex_hull_translation [symmetric])
also have ((+) (-x0)) ‘ X = insert 0 ((λx. x - x0) ‘ (X - {x0}))
  using x0 by (auto simp: image_iff)
finally have eq: measure lebesgue (convex hull X) = measure lebesgue (convex hull ...).

```

```

from compact have measure lebesgue (g ‘ std) = |det M| * measure lebesgue std
  by (subst measure_lebesgue_linear_transformation)
  (auto intro: finite_imp_bounded_convex_hull dest: compact_imp_closed)
simp: g _def std_def)

```

```

also have measure lebesgue std = content std using compact
  by (intro measure_completion) (auto dest: compact_imp_closed)
also have content std = 1 / fact CARD('n) unfolding std_def
  by (simp add: content_std_simplex)
also have g ‘ std = convex hull (g ‘ insert 0 Basis) unfolding std_def
  by (rule convex_hull_linear_image) (auto simp: g_def)

```

```

also have  $g \text{ ` insert } 0 \text{ Basis} = \text{insert } 0 \text{ (} g \text{ ` Basis)}$ 
  by (auto simp:  $g\_def$ )
also have  $g \text{ ` Basis} = (\lambda x. x - x0) \text{ ` range } f$ 
  by (auto simp:  $g\_def$   $Basis\_vec\_def$   $image\_iff$ )
also have  $\text{range } f = X - \{x0\}$  using  $bij$ 
  using  $bij\_betw\_imp\_surj\_on$  by blast
also note  $eq$  [ $symmetric$ ]
finally show  $?thesis$ 
  using  $finite\_imp\_compact\_convex\_hull[OF \langle finite \ X \rangle]$  by (auto dest:  $compact\_imp\_closed$ )
qed

```

theorem $content_triangle$:

```

fixes  $A \ B \ C :: real \ ^2$ 
shows  $content \ (convex \ hull \ \{A, B, C\}) =$ 
   $| (C \ \$ \ 1 - A \ \$ \ 1) * (B \ \$ \ 2 - A \ \$ \ 2) - (B \ \$ \ 1 - A \ \$ \ 1) * (C \ \$ \ 2 - A$ 
 $\ \$ \ 2) | / 2$ 
proof -
  define  $M :: real \ ^2 \ ^2$  where  $M \equiv (\chi \ i. \chi \ j. (if \ j = 1 \ then \ B \ else \ C) \ \$ \ i -$ 
 $A \ \$ \ i)$ 
  define  $g$  where  $g = (\lambda x. M * v \ x)$ 
  define  $std$  where  $std = (convex \ hull \ insert \ 0 \ Basis :: (real \ ^2) \ set)$ 
  have [ $simp$ ]:  $M * v \ axis \ i \ 1 = (if \ i = 1 \ then \ B - A \ else \ C - A)$  for  $i$ 
  by (auto simp:  $M\_def$   $matrix\_vector\_mult\_basis$   $column\_def$   $vec\_eq\_iff$ )
  have  $compact$ :  $compact \ std$  unfolding  $std\_def$ 
  by (intro  $finite\_imp\_compact\_convex\_hull$ ) auto

  have  $measure \ lebesgue \ (convex \ hull \ \{A, B, C\}) =$ 
     $measure \ lebesgue \ (((+) \ (-A)) \text{ ` } (convex \ hull \ \{A, B, C\}))$ 
  by (rule  $measure\_translation$  [ $symmetric$ ])
  also have  $((+) \ (-A)) \text{ ` } (convex \ hull \ \{A, B, C\}) = convex \ hull \ (((+) \ (-A)) \text{ ` }$ 
 $\ \{A, B, C\})$ 
  by (rule  $convex\_hull\_translation$  [ $symmetric$ ])
  also have  $((+) \ (-A)) \text{ ` } \{A, B, C\} = \{0, B - A, C - A\}$ 
  by (auto simp:  $image\_iff$ )
  finally have  $eq$ :  $measure \ lebesgue \ (convex \ hull \ \{A, B, C\}) =$ 
     $measure \ lebesgue \ (convex \ hull \ \{0, B - A, C - A\}) .$ 

  from  $compact$  have  $measure \ lebesgue \ (g \text{ ` } std) = |\det \ M| * measure \ lebesgue \ std$ 
  by (subst  $measure\_lebesgue\_linear\_transformation$ )
  (auto intro:  $finite\_imp\_bounded\_convex\_hull$  dest:  $compact\_imp\_closed$ 
 $simp$ :  $g\_def \ std\_def$ )
  also have  $measure \ lebesgue \ std = content \ std$  using  $compact$ 
  by (intro  $measure\_completion$ ) (auto dest:  $compact\_imp\_closed$ )
  also have  $content \ std = 1 / 2$  unfolding  $std\_def$ 
  by ( $simp$  add:  $content\_std\_simplex$ )
  also have  $g \text{ ` } std = convex \ hull \ (g \text{ ` } insert \ 0 \ Basis)$  unfolding  $std\_def$ 
  by (rule  $convex\_hull\_linear\_image$ ) (auto simp:  $g\_def$ )
  also have  $g \text{ ` } insert \ 0 \ Basis = insert \ 0 \ (g \text{ ` } Basis)$ 

```

```

    by (auto simp: g_def)
  also have (2 :: 2) ≠ 1 by auto
  hence ¬(∀ y::2. y = 1) by blast
  hence g `Basis = {B - A, C - A}
    by (auto simp: g_def Basis_vec_def image_iff)
  also note eq [symmetric]
  finally show ?thesis
    using finite_imp_compact_convex_hull[of {A, B, C}]
    by (auto dest!: compact_imp_closed simp: det_2 M_def)
qed

```

theorem heron:

```

  fixes A B C :: real ^ 2
  defines a ≡ dist B C and b ≡ dist A C and c ≡ dist A B
  defines s ≡ (a + b + c) / 2
  shows content (convex hull {A, B, C}) = sqrt (s * (s - a) * (s - b) * (s -
c))
proof -
  have [simp]: (UNIV :: 2 set) = {1, 2}
    using exhaust_2 by auto
  have dist_eq: dist (A :: real ^ 2) B ^ 2 = (A $ 1 - B $ 1) ^ 2 + (A $ 2 - B
$ 2) ^ 2
    for A B by (simp add: dist_vec_def dist_real_def)
  have nonneg: s * (s - a) * (s - b) * (s - c) ≥ 0
    using dist_triangle[of A B C] dist_triangle[of A C B] dist_triangle[of B C A]
    by (intro mult_nonneg_nonneg) (auto simp: s_def a_def b_def c_def dist_commute)

  have 16 * content (convex hull {A, B, C}) ^ 2 =
    4 * ((C $ 1 - A $ 1) * (B $ 2 - A $ 2) - (B $ 1 - A $ 1) * (C $ 2 -
A $ 2)) ^ 2
    by (subst content_triangle) (simp add: power_divide)
  also have ... = (2 * (dist A B ^ 2 * dist A C ^ 2 + dist A B ^ 2 * dist B C
^ 2 +
    dist A C ^ 2 * dist B C ^ 2) - (dist A B ^ 2) ^ 2 - (dist A C ^ 2) ^ 2 -
(dist B C ^ 2) ^ 2)
    unfolding dist_eq unfolding power2_eq_square by algebra
  also have ... = (a + b + c) * ((a + b + c) - 2 * a) * ((a + b + c) - 2 * b) *
    ((a + b + c) - 2 * c)
    unfolding power2_eq_square by (simp add: s_def a_def b_def c_def alge-
bra_simps)
  also have ... = 16 * s * (s - a) * (s - b) * (s - c)
    by (simp add: s_def field_split_simps)
  finally have content (convex hull {A, B, C}) ^ 2 = s * (s - a) * (s - b) * (s
- c)
    by simp
  also have ... = sqrt (s * (s - a) * (s - b) * (s - c)) ^ 2
    by (intro real_sqrt_pow2 [symmetric] nonneg)
  finally show ?thesis using nonneg
    by (subst (asm) power2_eq_iff_nonneg) auto

```

qed

end

9.35 Convergence of Formal Power Series

theory FPS_Convergence

imports

Generalised_Binomial_Theorem

HOL-Computational_Algebra.Formal_Power_Series

begin

In this theory, we will connect formal power series (which are algebraic objects) with analytic functions. This will become more important in complex analysis, and indeed some of the less trivial results will only be proven there.

9.35.1 Balls with extended real radius

The following is a variant of *ball* that also allows an infinite radius.

definition *eball* :: 'a :: metric_space \Rightarrow ereal \Rightarrow 'a set **where**

eball z r = {z'. ereal (dist z z') < r}

lemma *in_eball_iff* [simp]: $z \in \text{eball } z0 \ r \longleftrightarrow \text{ereal } (\text{dist } z0 \ z) < r$

by (simp add: *eball_def*)

lemma *eball_ereal* [simp]: $\text{eball } z \ (\text{ereal } r) = \text{ball } z \ r$

by *auto*

lemma *eball_inf* [simp]: $\text{eball } z \ \infty = \text{UNIV}$

by *auto*

lemma *eball_empty* [simp]: $r \leq 0 \implies \text{eball } z \ r = \{\}$

proof *safe*

fix x **assume** $r \leq 0$ $x \in \text{eball } z \ r$

hence $\text{dist } z \ x < r$ **by** *simp*

also have $\dots \leq \text{ereal } 0$ **using** $\langle r \leq 0 \rangle$ **by** (simp add: *zero_ereal_def*)

finally show $x \in \{\}$ **by** *simp*

qed

lemma *eball_conv_UNION_balls*:

$\text{eball } z \ r = (\bigcup r' \in \{r'. \text{ereal } r' < r\}. \text{ball } z \ r')$

by (cases r) (use *dense_gt_ex* in force)+

lemma *eball_mono*: $r \leq r' \implies \text{eball } z \ r \leq \text{eball } z \ r'$

by *auto*

lemma *ball_eball_mono*: $\text{ereal } r \leq r' \implies \text{ball } z \ r \leq \text{eball } z \ r'$

using *eball_mono*[of *ereal r r'*] **by** *simp*

lemma *open_eball* [*simp*, *intro*]: *open* (*eball* *z* *r*)
 by (*cases* *r*) *auto*

9.35.2 Basic properties of convergent power series

definition *fps_conv_radius* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps* \Rightarrow *ereal* **where**
fps_conv_radius *f* = *conv_radius* (*fps_nth* *f*)

definition *eval_fps* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps* \Rightarrow 'a \Rightarrow 'a **where**
eval_fps *f* *z* = (\sum *n*. *fps_nth* *f* *n* * *z* ^ *n*)

lemma *norm_summable_fps*:
fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
shows *norm* *z* < *fps_conv_radius* *f* \implies *summable* (λ n. *norm* (*fps_nth* *f* *n* * *z* ^ *n*))
 by (*rule* *abs_summable_in_conv_radius*) (*simp_all* *add*: *fps_conv_radius_def*)

lemma *summable_fps*:
fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
shows *norm* *z* < *fps_conv_radius* *f* \implies *summable* (λ n. *fps_nth* *f* *n* * *z* ^ *n*)
 by (*rule* *summable_in_conv_radius*) (*simp_all* *add*: *fps_conv_radius_def*)

theorem *sums_eval_fps*:
fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm* *z* < *fps_conv_radius* *f*
shows (λ n. *fps_nth* *f* *n* * *z* ^ *n*) *sums* *eval_fps* *f* *z*
using *assms* **unfolding** *eval_fps_def* *fps_conv_radius_def*
 by (*intro* *summable_sums* *summable_in_conv_radius*) *simp_all*

lemma *continuous_on_eval_fps*:
fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
shows *continuous_on* (*eball* 0 (*fps_conv_radius* *f*)) (*eval_fps* *f*)
proof (*subst* *continuous_on_eq_continuous_at* [*OF* *open_eball*], *safe*)
fix *x* :: 'a **assume** *x*: *x* \in *eball* 0 (*fps_conv_radius* *f*)
define *r* **where** *r* = (*if* *fps_conv_radius* *f* = ∞ *then* *norm* *x* + 1 *else*
 (*norm* *x* + *real_of_ereal* (*fps_conv_radius* *f*) / 2)
have *r*: *norm* *x* < *r* \wedge *ereal* *r* < *fps_conv_radius* *f*
using *x* **by** (*cases* *fps_conv_radius* *f*)
 (*auto* *simp*: *r_def* *eball_def* *split*: *if_splits*)
have *continuous_on* (*cball* 0 *r*) (λ x. \sum *i*. *fps_nth* *f* *i* * (*x* - 0) ^ *i*)
by (*rule* *powser_continuous_suminf*) (*insert* *r*, *auto* *simp*: *fps_conv_radius_def*)
hence *continuous_on* (*cball* 0 *r*) (*eval_fps* *f*)
by (*simp* *add*: *eval_fps_def*)
thus *isCont* (*eval_fps* *f*) *x*
by (*rule* *continuous_on_interior*) (*use* *r* **in** *auto*)

qed

lemma *continuous_on_eval_fps'* [continuous_intros]:

assumes *continuous_on* A g
assumes $g \text{ ' } A \subseteq \text{eball } 0 \text{ (fps_conv_radius } f)$
shows *continuous_on* A $(\lambda x. \text{eval_fps } f \text{ (} g \text{ } x))$
using *continuous_on_compose2* [OF *continuous_on_eval_fps* *assms*].

lemma *has_field_derivative_powser*:

fixes $z :: 'a :: \{\text{banach, real_normed_field}\}$
assumes *ereal* $(\text{norm } z) < \text{conv_radius } f$
shows $((\lambda z. \sum n. f \ n * z \ ^n) \text{ has_field_derivative } (\sum n. \text{diffs } f \ n * z \ ^n))$
(at z *within* A)

proof –

define K **where** $K = (\text{if } \text{conv_radius } f = \infty \text{ then } \text{norm } z + 1$
 $\text{else } (\text{norm } z + \text{real_of_ereal } (\text{conv_radius } f)) / 2)$

have K : $\text{norm } z < K \wedge \text{ereal } K < \text{conv_radius } f$
using *assms* **by** (*cases* *conv_radius* f) (*auto simp: K_def*)

have $0 \leq \text{norm } z$ **by** *simp*

also from K **have** $\dots < K$ **by** *simp*

finally have K_pos : $K > 0$ **by** *simp*

have *summable* $(\lambda n. f \ n * \text{of_real } K \ ^n)$

using K **and** K_pos **by** (*intro* *summable_in_conv_radius*) *auto*

moreover from K **and** K_pos **have** $\text{norm } z < \text{norm } (\text{of_real } K :: 'a)$ **by** *auto*

ultimately show *?thesis*

by (*rule* *has_field_derivative_at_within* [OF *termdiffs_strong*])

qed

lemma *has_field_derivative_eval_fps*:

fixes $z :: 'a :: \{\text{banach, real_normed_field}\}$
assumes $\text{norm } z < \text{fps_conv_radius } f$
shows $(\text{eval_fps } f \text{ has_field_derivative } \text{eval_fps } (\text{fps_deriv } f) \ z)$ *(at* z *within* A)

proof –

have $(\text{eval_fps } f \text{ has_field_derivative } \text{eval_fps } (\text{Abs_fps } (\text{diffs } (\text{fps_nth } f)))) \ z)$
(at z *within* A)

using *assms* **unfolding** *eval_fps_def* *fps_nth_Abs_fps* *fps_conv_radius_def*

by (*intro* *has_field_derivative_powser*) *auto*

also have $\text{Abs_fps } (\text{diffs } (\text{fps_nth } f)) = \text{fps_deriv } f$

by (*simp add: fps_eq_iff_diffs_def*)

finally show *?thesis* .

qed

lemma *holomorphic_on_eval_fps* [*holomorphic_intros*]:

fixes $z :: 'a :: \{\text{banach, real_normed_field}\}$

assumes $A \subseteq \text{eball } 0 \text{ (fps_conv_radius } f)$

shows *eval_fps* f *holomorphic_on* A

proof (*rule* *holomorphic_on_subset* [OF *_ assms*])

3830

```

show eval_fps f holomorphic_on eball 0 (fps_conv_radius f)
proof (subst holomorphic_on_open [OF open_eball], safe, goal_cases)
  case (1 x)
  thus ?case
  by (intro exI[of _ eval_fps (fps_deriv f) x]) (auto intro: has_field_derivative_eval_fps)
qed

```

```

lemma analytic_on_eval_fps:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes A ⊆ eball 0 (fps_conv_radius f)
  shows eval_fps f analytic_on A
proof (rule analytic_on_subset [OF _ assms])
  show eval_fps f analytic_on eball 0 (fps_conv_radius f)
  using holomorphic_on_eval_fps[of eball 0 (fps_conv_radius f)]
  by (subst analytic_on_open) auto
qed

```

```

lemma continuous_eval_fps [continuous_intros]:
  fixes z :: 'a::{real_normed_field,banach}
  assumes norm z < fps_conv_radius F
  shows continuous (at z within A) (eval_fps F)
proof -
  from ereal_dense2[OF assms] obtain K :: real where K: norm z < K K <
fps_conv_radius F
  by auto
  have 0 ≤ norm z by simp
  also have norm z < K by fact
  finally have K > 0 .
  from K and ⟨K > 0⟩ have summable (λn. fps_nth F n * of_real K ^ n)
  by (intro summable_fps) auto
  from this have isCont (eval_fps F) z unfolding eval_fps_def
  by (rule isCont_powser) (use K in auto)
  thus continuous (at z within A) (eval_fps F)
  by (simp add: continuous_at_imp_continuous_within)
qed

```

9.35.3 Lower bounds on radius of convergence

```

lemma fps_conv_radius_deriv:
  fixes f :: 'a :: {banach, real_normed_field} fps
  shows fps_conv_radius (fps_deriv f) ≥ fps_conv_radius f
  unfolding fps_conv_radius_def
proof (rule conv_radius_geI_ex)
  fix r :: real assume r: r > 0 ereal r < conv_radius (fps_nth f)
  define K where K = (if conv_radius (fps_nth f) = ∞ then r + 1
    else (real_of_ereal (conv_radius (fps_nth f)) + r) / 2)
  have K: r < K ∧ ereal K < conv_radius (fps_nth f)
  using r by (cases conv_radius (fps_nth f)) (auto simp: K_def)

```

```

have summable ( $\lambda n.$  diffs (fps_nth f) n * of_real r ^ n)
proof (rule termdiff_converges)
  fix x :: 'a assume norm x < K
  hence ereal (norm x) < ereal K by simp
  also have ... < conv_radius (fps_nth f) using K by simp
  finally show summable ( $\lambda n.$  fps_nth f n * x ^ n)
    by (intro summable_in_conv_radius) auto
qed (insert K r, auto)
also have ... = ( $\lambda n.$  fps_nth (fps_deriv f) n * of_real r ^ n)
  by (simp add: fps_deriv_def diffs_def)
finally show  $\exists z::'a.$  norm z = r  $\wedge$  summable ( $\lambda n.$  fps_nth (fps_deriv f) n * z
  ^ n)
  using r by (intro exI[of _ of_real r]) auto
qed

```

```

lemma eval_fps_at_0: eval_fps f 0 = fps_nth f 0
  by (simp add: eval_fps_def)

```

```

lemma fps_conv_radius_norm [simp]:
  fps_conv_radius (Abs_fps ( $\lambda n.$  norm (fps_nth f n))) = fps_conv_radius f
  by (simp add: fps_conv_radius_def)

```

```

lemma fps_conv_radius_const [simp]: fps_conv_radius (fps_const c) =  $\infty$ 
proof -
  have fps_conv_radius (fps_const c) = conv_radius ( $\lambda_.$  0 :: 'a)
    unfolding fps_conv_radius_def
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of 0]])
  auto
  thus ?thesis by simp
qed

```

```

lemma fps_conv_radius_0 [simp]: fps_conv_radius 0 =  $\infty$ 
  by (simp only: fps_const_0_eq_0 [symmetric] fps_conv_radius_const)

```

```

lemma fps_conv_radius_1 [simp]: fps_conv_radius 1 =  $\infty$ 
  by (simp only: fps_const_1_eq_1 [symmetric] fps_conv_radius_const)

```

```

lemma fps_conv_radius_numeral [simp]: fps_conv_radius (numeral n) =  $\infty$ 
  by (simp add: numeral_fps_const)

```

```

lemma fps_conv_radius_fps_X_power [simp]: fps_conv_radius (fps_X ^ n) =
 $\infty$ 
proof -
  have fps_conv_radius (fps_X ^ n :: 'a fps) = conv_radius ( $\lambda_.$  0 :: 'a)
    unfolding fps_conv_radius_def
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of n]])

    (auto simp: fps_X_power_iff)
  thus ?thesis by simp

```

3832

qed

lemma *fps_conv_radius_fps_X* [*simp*]: $\text{fps_conv_radius } \text{fps_X} = \infty$
using *fps_conv_radius_fps_X_power*[of 1] **by** (*simp only: power_one_right*)

lemma *fps_conv_radius_shift* [*simp*]:
 $\text{fps_conv_radius } (\text{fps_shift } n \ f) = \text{fps_conv_radius } f$
by (*simp add: fps_conv_radius_def fps_shift_def conv_radius_shift*)

lemma *fps_conv_radius_cmult_left*:
 $c \neq 0 \implies \text{fps_conv_radius } (\text{fps_const } c * f) = \text{fps_conv_radius } f$
unfolding *fps_conv_radius_def* **by** (*simp add: conv_radius_cmult_left*)

lemma *fps_conv_radius_cmult_right*:
 $c \neq 0 \implies \text{fps_conv_radius } (f * \text{fps_const } c) = \text{fps_conv_radius } f$
unfolding *fps_conv_radius_def* **by** (*simp add: conv_radius_cmult_right*)

lemma *fps_conv_radius_uminus* [*simp*]:
 $\text{fps_conv_radius } (-f) = \text{fps_conv_radius } f$
using *fps_conv_radius_cmult_left*[of -1 *f*]
by (*simp flip: fps_const_neg*)

lemma *fps_conv_radius_add*: $\text{fps_conv_radius } (f + g) \geq \min (\text{fps_conv_radius } f) (\text{fps_conv_radius } g)$
unfolding *fps_conv_radius_def* using *conv_radius_add_ge*[of *fps_nth f fps_nth g*]
by *simp*

lemma *fps_conv_radius_diff*: $\text{fps_conv_radius } (f - g) \geq \min (\text{fps_conv_radius } f) (\text{fps_conv_radius } g)$
using *fps_conv_radius_add*[of *f -g*] **by** *simp*

lemma *fps_conv_radius_mult*: $\text{fps_conv_radius } (f * g) \geq \min (\text{fps_conv_radius } f) (\text{fps_conv_radius } g)$
using *conv_radius_mult_ge*[of *fps_nth f fps_nth g*]
by (*simp add: fps_mult_nth fps_conv_radius_def atLeast0AtMost*)

lemma *fps_conv_radius_power*: $\text{fps_conv_radius } (f \wedge n) \geq \text{fps_conv_radius } f$
proof (*induction n*)
case (*Suc n*)
hence $\text{fps_conv_radius } f \leq \min (\text{fps_conv_radius } f) (\text{fps_conv_radius } (f \wedge n))$
by *simp*
also have $\dots \leq \text{fps_conv_radius } (f * f \wedge n)$
by (*rule fps_conv_radius_mult*)
finally show ?case **by** *simp*

qed *simp_all*

context
begin

```

lemma natfun_inverse_bound:
  fixes f :: 'a :: {real_normed_field} fps
  assumes fps_nth f 0 = 1 and  $\delta > 0$ 
    and summable: summable ( $\lambda n. \text{norm} (\text{fps\_nth } f (\text{Suc } n)) * \delta ^ \text{Suc } n$ )
    and le: ( $\sum n. \text{norm} (\text{fps\_nth } f (\text{Suc } n)) * \delta ^ \text{Suc } n \leq 1$ )
  shows  $\text{norm} (\text{natfun\_inverse } f n) \leq \text{inverse} (\delta ^ n)$ 
proof (induction n rule: less_induct)
  case (less m)
  show ?case
  proof (cases m)
    case 0
    thus ?thesis using assms by (simp add: field_split_simps norm_inverse
norm_divide)
  next
  case [simp]: (Suc n)
  have  $\text{norm} (\text{natfun\_inverse } f (\text{Suc } n)) =$ 
 $\text{norm} (\sum i = \text{Suc } 0.. \text{Suc } n. \text{fps\_nth } f i * \text{natfun\_inverse } f (\text{Suc } n - i))$ 
(is  $\_ = \text{norm } ?S$ ) using assms
by (simp add: field_simps norm_mult norm_divide del: sum.cl_ivl_Suc)
also have  $\text{norm } ?S \leq (\sum i = \text{Suc } 0.. \text{Suc } n. \text{norm} (\text{fps\_nth } f i * \text{natfun\_inverse}$ 
 $f (\text{Suc } n - i)))$ 
by (rule norm_sum)
also have  $\dots \leq (\sum i = \text{Suc } 0.. \text{Suc } n. \text{norm} (\text{fps\_nth } f i) / \delta ^ (\text{Suc } n - i))$ 
proof (intro sum_mono, goal_cases)
  case (1 i)
  have  $\text{norm} (\text{fps\_nth } f i * \text{natfun\_inverse } f (\text{Suc } n - i)) =$ 
 $\text{norm} (\text{fps\_nth } f i) * \text{norm} (\text{natfun\_inverse } f (\text{Suc } n - i))$ 
by (simp add: norm_mult)
also have  $\dots \leq \text{norm} (\text{fps\_nth } f i) * \text{inverse} (\delta ^ (\text{Suc } n - i))$ 
using 1 by (intro mult_left_mono less.IH) auto
also have  $\dots = \text{norm} (\text{fps\_nth } f i) / \delta ^ (\text{Suc } n - i)$ 
by (simp add: field_split_simps)
finally show ?case .
qed
also have  $\dots = (\sum i = \text{Suc } 0.. \text{Suc } n. \text{norm} (\text{fps\_nth } f i) * \delta ^ i) / \delta ^ \text{Suc } n$ 
by (subst sum_divide_distrib, rule sum.cong)
(insert  $\langle \delta > 0 \rangle$ , auto simp: field_simps power_diff)
also have  $(\sum i = \text{Suc } 0.. \text{Suc } n. \text{norm} (\text{fps\_nth } f i) * \delta ^ i) =$ 
 $(\sum i=0..n. \text{norm} (\text{fps\_nth } f (\text{Suc } i)) * \delta ^ (\text{Suc } i))$ 
by (subst sum.atLeast_Suc_atMost_Suc_shift) simp_all
also have  $\{0..n\} = \{..<\text{Suc } n\}$  by auto
also have  $(\sum i < \text{Suc } n. \text{norm} (\text{fps\_nth } f (\text{Suc } i)) * \delta ^ (\text{Suc } i)) \leq$ 
 $(\sum n. \text{norm} (\text{fps\_nth } f (\text{Suc } n)) * \delta ^ (\text{Suc } n))$ 
using  $\langle \delta > 0 \rangle$  by (intro sum_le_suminf ballI mult_nonneg_nonneg zero_le_power
summable) auto
also have  $\dots \leq 1$  by fact
finally show ?thesis using  $\langle \delta > 0 \rangle$ 
by (simp add: divide_right_mono field_split_simps)

```

qed
qed

```

private lemma fps_conv_radius_inverse_pos_aux:
  fixes f :: 'a :: {banach, real_normed_field} fps
  assumes fps_nth f 0 = 1 fps_conv_radius f > 0
  shows fps_conv_radius (inverse f) > 0
proof -
  let ?R = fps_conv_radius f
  define h where h = Abs_fps (λn. norm (fps_nth f n))
  have [simp]: fps_conv_radius h = ?R by (simp add: h_def)
  have continuous_on (eball 0 (fps_conv_radius h)) (eval_fps h)
    by (intro continuous_on_eval_fps)
  hence *: open (eval_fps h - ' A ∩ eball 0 ?R) if open A for A
    using that by (subst (asm) continuous_on_open_vimage) auto
  have open (eval_fps h - ' {..<2} ∩ eball 0 ?R)
    by (rule *) auto
  moreover have 0 ∈ eval_fps h - ' {..<2} ∩ eball 0 ?R
    using assms by (auto simp: eball_def zero_ereal_def eval_fps_at_0 h_def)
  ultimately obtain ε where ε: ε > 0 ball 0 ε ⊆ eval_fps h - ' {..<2} ∩ eball 0 ?R
  ?R
    by (subst (asm) open_contains_ball_eq) blast+

  define δ where δ = real_of_ereal (min (ereal ε / 2) (?R / 2))
  have δ: 0 < δ ∧ δ < ε ∧ ereal δ < ?R
    using «ε > 0» and assms by (cases ?R) (auto simp: δ_def min_def)

  have summable: summable (λn. norm (fps_nth f n) * δ ^ n)
  using δ by (intro summable_in_conv_radius) (simp_all add: fps_conv_radius_def)
  hence (λn. norm (fps_nth f n) * δ ^ n) sums eval_fps h δ
    by (simp add: eval_fps_def summable_sums h_def)
  hence (λn. norm (fps_nth f (Suc n)) * δ ^ Suc n) sums (eval_fps h δ - 1)
    by (subst sums_Suc_iff) (auto simp: assms)
  moreover {
    from δ have δ ∈ ball 0 ε by auto
    also have ... ⊆ eval_fps h - ' {..<2} ∩ eball 0 ?R by fact
    finally have eval_fps h δ < 2 by simp
  }
  ultimately have le: (∑ n. norm (fps_nth f (Suc n)) * δ ^ Suc n) ≤ 1
    by (simp add: sums_iff)
  from summable have summable: summable (λn. norm (fps_nth f (Suc n)) * δ
  ^ Suc n)
    by (subst summable_Suc_iff)

  have 0 < δ using δ by blast
  also have δ = inverse (limsup (λn. ereal (inverse δ)))
    using δ by (subst Limsup_const) auto
  also have ... ≤ conv_radius (natfun_inverse f)
    unfolding conv_radius_def

```

```

proof (intro ereal_inverse_antimono Limsup_mono
  eventually_mono[OF eventually_gt_at_top[of 0]])
  fix n :: nat assume n: n > 0
  have root n (norm (natfun_inverse f n)) ≤ root n (inverse (δ ^ n))
    using n assms δ le_summable
    by (intro real_root_le_mono natfun_inverse_bound) auto
  also have ... = inverse δ
    using n δ by (simp add: power_inverse [symmetric] real_root_pos2)
  finally show ereal (inverse δ) ≥ ereal (root n (norm (natfun_inverse f n)))
    by (subst ereal_less_eq)
next
  have 0 = limsup (λn. 0::ereal)
    by (rule Limsup_const [symmetric]) auto
  also have ... ≤ limsup (λn. ereal (root n (norm (natfun_inverse f n))))
    by (intro Limsup_mono) (auto simp: real_root_ge_zero)
  finally show 0 ≤ ... by simp
qed
also have ... = fps_conv_radius (inverse f)
  using assms by (simp add: fps_conv_radius_def fps_inverse_def)
finally show ?thesis by (simp add: zero_ereal_def)
qed

lemma fps_conv_radius_inverse_pos:
  fixes f :: 'a :: {banach, real_normed_field} fps
  assumes fps_nth f 0 ≠ 0 and fps_conv_radius f > 0
  shows fps_conv_radius (inverse f) > 0
proof -
  let ?c = fps_nth f 0
  have fps_conv_radius (inverse f) = fps_conv_radius (fps_const ?c * inverse f)
    using assms by (subst fps_conv_radius_cmult_left) auto
  also have fps_const ?c * inverse f = inverse (fps_const (inverse ?c) * f)
    using assms by (simp add: fps_inverse_mult fps_const_inverse)
  also have fps_conv_radius ... > 0 using assms
    by (intro fps_conv_radius_inverse_pos_aux)
    (auto simp: fps_conv_radius_cmult_left)
  finally show ?thesis .
qed

end

lemma fps_conv_radius_exp [simp]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows fps_conv_radius (fps_exp c) = ∞
  unfolding fps_conv_radius_def
proof (rule conv_radius_inftyI')
  fix z :: 'a
  have (λn. norm (c * z) ^ n /R fact n) sums exp (norm (c * z))
    by (rule exp_converges)
  also have (λn. norm (c * z) ^ n /R fact n) = (λn. norm (fps_nth (fps_exp c)

```

3836

```
 $n * z^{\wedge} n$ )  
  by (rule ext) (simp add: norm_divide norm_mult norm_power field_split_simps)  
  finally have summable ... by (simp add: sums_iff)  
  thus summable ( $\lambda n. \text{fps\_nth } (\text{fps\_exp } c) n * z^{\wedge} n$ )  
    by (rule summable_norm_cancel)  
qed
```

9.35.4 Evaluating power series

```
theorem eval_fps_deriv:  
  assumes norm  $z < \text{fps\_conv\_radius } f$   
  shows eval_fps (fps_deriv f)  $z = \text{deriv } (\text{eval\_fps } f) z$   
  by (intro DERIV_imp_deriv [symmetric] has_field_derivative_eval_fps assms)
```

```
theorem fps_nth_conv_deriv:  
  fixes  $f :: \text{complex fps}$   
  assumes fps_conv_radius  $f > 0$   
  shows  $\text{fps\_nth } f n = (\text{deriv } \sim n) (\text{eval\_fps } f) 0 / \text{fact } n$   
  using assms
```

```
proof (induction  $n$  arbitrary:  $f$ )
```

```
  case 0
```

```
  thus ?case by (simp add: eval_fps_def)
```

```
next
```

```
  case (Suc  $n f$ )
```

```
  have  $(\text{deriv } \sim (\text{Suc } n) (\text{eval\_fps } f) 0 = (\text{deriv } \sim n) (\text{deriv } (\text{eval\_fps } f)) 0$ 
```

```
    unfolding funpow_Suc_right o_def ..
```

```
  also have eventually ( $\lambda z :: \text{complex}. z \in \text{eball } 0 (\text{fps\_conv\_radius } f)$ ) (nhds 0)
```

```
    using Suc.premis by (intro eventually_nhds_in_open) (auto simp: zero_ereal_def)
```

```
  hence eventually ( $\lambda z. \text{deriv } (\text{eval\_fps } f) z = \text{eval\_fps } (\text{fps\_deriv } f) z$ ) (nhds 0)
```

```
    by eventually_elim (simp add: eval_fps_deriv)
```

```
  hence  $(\text{deriv } \sim n) (\text{deriv } (\text{eval\_fps } f)) 0 = (\text{deriv } \sim n) (\text{eval\_fps } (\text{fps\_deriv } f)) 0$ 
```

```
    by (intro higher_deriv_cong_ev refl)
```

```
  also have ... / fact  $n = \text{fps\_nth } (\text{fps\_deriv } f) n$ 
```

```
    using Suc.premis fps_conv_radius_deriv[of  $f$ ]
```

```
    by (intro Suc.IH [symmetric]) auto
```

```
  also have ... / of_nat (Suc  $n$ ) =  $\text{fps\_nth } f (\text{Suc } n)$ 
```

```
    by (simp add: fps_deriv_def del: of_nat_Suc)
```

```
  finally show ?case by (simp add: field_split_simps)
```

```
qed
```

```
theorem eval_fps_eqD:
```

```
  fixes  $f g :: \text{complex fps}$ 
```

```
  assumes fps_conv_radius  $f > 0$  fps_conv_radius  $g > 0$ 
```

```
  assumes eventually ( $\lambda z. \text{eval\_fps } f z = \text{eval\_fps } g z$ ) (nhds 0)
```

```
  shows  $f = g$ 
```

```
proof (rule fps_ext)
```

```
  fix  $n :: \text{nat}$ 
```

```
  have  $\text{fps\_nth } f n = (\text{deriv } \sim n) (\text{eval\_fps } f) 0 / \text{fact } n$ 
```



```

    using assms by (intro fps_nth_conv_deriv)
  also have (deriv  $\hat{\sim}$  n) (eval_fps f) 0 = (deriv  $\hat{\sim}$  n) (eval_fps g) 0
    by (intro higher_deriv_cong_ev refl assms)
  also have ... / fact n = fps_nth g n
    using assms by (intro fps_nth_conv_deriv [symmetric])
  finally show fps_nth f n = fps_nth g n .
qed

```

```

lemma eval_fps_const [simp]:
  fixes c :: 'a :: {banach, real_normed_div_algebra}
  shows eval_fps (fps_const c) z = c
proof -
  have ( $\lambda n::nat. \text{if } n \in \{0\} \text{ then } c \text{ else } 0$ ) sums ( $\sum n \in \{0::nat\}. c$ )
    by (rule sums_If_finite_set) auto
  also have ?this  $\longleftrightarrow$  ( $\lambda n::nat. \text{fps\_nth } (fps\_const\ c) n * z^{\wedge} n$ ) sums ( $\sum n \in \{0::nat\}. c$ )
    by (intro sums_cong) auto
  also have ( $\sum n \in \{0::nat\}. c$ ) = c
    by simp
  finally show ?thesis
    by (simp add: eval_fps_def sums_iff)
qed

```

```

lemma eval_fps_0 [simp]:
  eval_fps (0 :: 'a :: {banach, real_normed_div_algebra} fps) z = 0
  by (simp only: fps_const_0_eq_0 [symmetric] eval_fps_const)

```

```

lemma eval_fps_1 [simp]:
  eval_fps (1 :: 'a :: {banach, real_normed_div_algebra} fps) z = 1
  by (simp only: fps_const_1_eq_1 [symmetric] eval_fps_const)

```

```

lemma eval_fps_numeral [simp]:
  eval_fps (numeral n :: 'a :: {banach, real_normed_div_algebra} fps) z = numeral
  n
  by (simp only: numeral_fps_const eval_fps_const)

```

```

lemma eval_fps_X_power [simp]:
  eval_fps (fps_X  $\hat{\sim}$  m :: 'a :: {banach, real_normed_div_algebra} fps) z = z  $\hat{\sim}$  m
proof -
  have ( $\lambda n::nat. \text{if } n \in \{m\} \text{ then } z^{\wedge} n \text{ else } 0 :: 'a$ ) sums ( $\sum n \in \{m::nat\}. z^{\wedge} n$ )
    by (rule sums_If_finite_set) auto
  also have ?this  $\longleftrightarrow$  ( $\lambda n::nat. \text{fps\_nth } (fps\_X\ \hat{\sim}\ m) n * z^{\wedge} n$ ) sums ( $\sum n \in \{m::nat\}. z^{\wedge} n$ )
    by (intro sums_cong) (auto simp: fps_X_power_iff)
  also have ( $\sum n \in \{m::nat\}. z^{\wedge} n$ ) = z  $\hat{\sim}$  m
    by simp
  finally show ?thesis
    by (simp add: eval_fps_def sums_iff)
qed

```

lemma *eval_fps_X* [*simp*]:

eval_fps (*fps_X* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *z*
using *eval_fps_X_power*[*of 1 z*] **by** (*simp only: power_one_right*)

lemma *eval_fps_minus*:

fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z < fps_conv_radius f*
shows *eval_fps* (*-f*) *z* = *-eval_fps f z*
using *assms unfolding eval_fps_def*
by (*subst suminf_minus [symmetric]*) (*auto intro!: summable_fps*)

lemma *eval_fps_add*:

fixes *f g* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z < fps_conv_radius f* *norm z < fps_conv_radius g*
shows *eval_fps* (*f + g*) *z* = *eval_fps f z + eval_fps g z*
using *assms unfolding eval_fps_def*
by (*subst suminf_add*) (*auto simp: ring_distrib intro!: summable_fps*)

lemma *eval_fps_diff*:

fixes *f g* :: 'a :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z < fps_conv_radius f* *norm z < fps_conv_radius g*
shows *eval_fps* (*f - g*) *z* = *eval_fps f z - eval_fps g z*
using *assms unfolding eval_fps_def*
by (*subst suminf_diff*) (*auto simp: ring_distrib intro!: summable_fps*)

lemma *eval_fps_mult*:

fixes *f g* :: 'a :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*} *fps*
assumes *norm z < fps_conv_radius f* *norm z < fps_conv_radius g*
shows *eval_fps* (*f * g*) *z* = *eval_fps f z * eval_fps g z*

proof -

have *eval_fps f z * eval_fps g z* =
 $(\sum k. \sum i \leq k. \text{fps_nth } f \ i * \text{fps_nth } g \ (k - i) * (z^i * z^{k-i}))$
unfolding *eval_fps_def*

proof (*subst Cauchy_product*)

show *summable* ($\lambda k. \text{norm } (\text{fps_nth } f \ k * z^k)$) *summable* ($\lambda k. \text{norm } (\text{fps_nth } g \ k * z^k)$)

by (*rule norm_summable_fps assms*)+

qed (*simp_all add: algebra_simps*)

also have ($\lambda k. \sum i \leq k. \text{fps_nth } f \ i * \text{fps_nth } g \ (k - i) * (z^i * z^{k-i})$) =
 $(\lambda k. \sum i \leq k. \text{fps_nth } f \ i * \text{fps_nth } g \ (k - i) * z^k)$

by (*intro ext sum.cong refl*) (*simp add: power_add [symmetric]*)

also have *suminf ... = eval_fps* (*f * g*) *z*

by (*simp add: eval_fps_def fps_mult_nth atLeast0AtMost sum_distrib_right*)

finally show *?thesis ..*

qed

lemma *eval_fps_shift*:

fixes *f* :: 'a :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*} *fps*

```

assumes  $n \leq \text{subdegree } f \text{ norm } z < \text{fps\_conv\_radius } f$ 
shows  $\text{eval\_fps } (\text{fps\_shift } n \ f) \ z = (\text{if } z = 0 \ \text{then } \text{fps\_nth } f \ n \ \text{else } \text{eval\_fps } f \ z / z \wedge n)$ 
proof ( $\text{cases } z = 0$ )
  case False
  have  $\text{eval\_fps } (\text{fps\_shift } n \ f * \text{fps\_X} \wedge n) \ z = \text{eval\_fps } (\text{fps\_shift } n \ f) \ z * z \wedge n$ 
    using assms by (subst eval_fps_mult) simp_all
  also from assms have  $\text{fps\_shift } n \ f * \text{fps\_X} \wedge n = f$ 
    by (simp add: fps_shift_times_fps_X_power)
  finally show ?thesis using False by (simp add: field_simps)
qed (simp_all add: eval_fps_at_0)

```

```

lemma eval_fps_exp [simp]:
  fixes  $c :: 'a :: \{\text{banach, real\_normed\_field}\}$ 
  shows  $\text{eval\_fps } (\text{fps\_exp } c) \ z = \text{exp } (c * z)$  unfolding eval_fps_def exp_def
  by (simp add: eval_fps_def exp_def scaleR_conv_of_real field_split_simps)

```

The case of division is more complicated and will therefore not be handled here. Handling division becomes much more easy using complex analysis, and we will do so once that is available.

9.35.5 Power series expansions of analytic functions

This predicate contains the notion that the given formal power series converges in some disc of positive radius around the origin and is equal to the given complex function there.

This relationship is unique in the sense that no complex function can have more than one formal power series to which it expands, and if two holomorphic functions that are holomorphic on a connected open set around the origin and have the same power series expansion, they must be equal on that set.

More concrete statements about the radius of convergence can usually be made, but for many purposes, the statement that the series converges to the function in some neighbourhood of the origin is enough, and that can be shown almost fully automatically in most cases, as there are straightforward introduction rules to show this.

In particular, when one wants to relate the coefficients of the power series to the values of the derivatives of the function at the origin, or if one wants to approximate the coefficients of the series with the singularities of the function, this predicate is enough.

definition

```

has_fps_expansion :: ( $'a :: \{\text{banach, real\_normed\_div\_algebra}\} \Rightarrow 'a$ )  $\Rightarrow 'a \ \text{fps} \Rightarrow \text{bool}$ 
(infixl has'_fps'_expansion 60)
where ( $f \ \text{has\_fps\_expansion } F$ )  $\longleftrightarrow$ 
   $\text{fps\_conv\_radius } F > 0 \wedge \text{eventually } (\lambda z. \text{eval\_fps } F \ z = f \ z) \ (\text{nhds } 0)$ 

```

named_theorems *fps_expansion_intros*

lemma *has_fps_expansion_schematicI*:

f has_fps_expansion A \implies *A = B* \implies *f has_fps_expansion B*
by *simp*

lemma *fps_nth_fps_expansion*:

fixes *f* :: *complex* \Rightarrow *complex*

assumes *f has_fps_expansion F*

shows *fps_nth F n = (deriv $\hat{\sim}$ n) f 0 / fact n*

proof –

have *fps_nth F n = (deriv $\hat{\sim}$ n) (eval_fps F) 0 / fact n*

using *assms* **by** (*intro fps_nth_conv_deriv*) (*auto simp: has_fps_expansion_def*)

also have *(deriv $\hat{\sim}$ n) (eval_fps F) 0 = (deriv $\hat{\sim}$ n) f 0*

using *assms* **by** (*intro higher_deriv_cong_ev*) (*auto simp: has_fps_expansion_def*)

finally show *?thesis* .

qed

lemma *has_fps_expansion_imp_continuous*:

fixes *F* :: '*a*::{*real_normed_field*,*banach*} *fps*

assumes *f has_fps_expansion F*

shows *continuous (at 0 within A) f*

proof –

from *assms* **have** *isCont (eval_fps F) 0*

by (*intro continuous_eval_fps*) (*auto simp: has_fps_expansion_def zero_ereal_def*)

also have *?this \longleftrightarrow isCont f 0* **using** *assms*

by (*intro isCont_cong*) (*auto simp: has_fps_expansion_def*)

finally have *isCont f 0* .

thus *continuous (at 0 within A) f*

by (*simp add: continuous_at_imp_continuous_within*)

qed

lemma *has_fps_expansion_const* [*simp, intro, fps_expansion_intros*]:

(λ _. *c*) *has_fps_expansion fps_const c*

by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_0* [*simp, intro, fps_expansion_intros*]:

(λ _. 0) *has_fps_expansion 0*

by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_1* [*simp, intro, fps_expansion_intros*]:

(λ _. 1) *has_fps_expansion 1*

by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_numeral* [*simp, intro, fps_expansion_intros*]:

(λ _. *numeral n*) *has_fps_expansion numeral n*

by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_fps_X_power* [*fps_expansion_intros*]:
 $(\lambda x. x \hat{=} n) \text{ has_fps_expansion } (\text{fps_X} \hat{=} n)$
by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_fps_X* [*fps_expansion_intros*]:
 $(\lambda x. x) \text{ has_fps_expansion } \text{fps_X}$
by (*auto simp: has_fps_expansion_def*)

lemma *has_fps_expansion_cmult_left* [*fps_expansion_intros*]:
fixes $c :: 'a :: \{\text{banach, real_normed_div_algebra, comm_ring_1}\}$
assumes $f \text{ has_fps_expansion } F$
shows $(\lambda x. c * f x) \text{ has_fps_expansion } \text{fps_const } c * F$
proof (*cases c = 0*)
case *False*
from *assms* **have** *eventually* $(\lambda z. z \in \text{eball } 0 (\text{fps_conv_radius } F)) (\text{nhds } 0)$
by (*intro eventually_nhds_in_open*) (*auto simp: has_fps_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* $(\lambda z. \text{eval_fps } F z = f z) (\text{nhds } 0)$
by (*auto simp: has_fps_expansion_def*)
ultimately have *eventually* $(\lambda z. \text{eval_fps } (\text{fps_const } c * F) z = c * f z) (\text{nhds } 0)$
by *eventually_elim* (*simp_all add: eval_fps_mult*)
with *assms* **and** *False* **show** *?thesis*
by (*auto simp: has_fps_expansion_def fps_conv_radius_cmult_left*)
qed *auto*

lemma *has_fps_expansion_cmult_right* [*fps_expansion_intros*]:
fixes $c :: 'a :: \{\text{banach, real_normed_div_algebra, comm_ring_1}\}$
assumes $f \text{ has_fps_expansion } F$
shows $(\lambda x. f x * c) \text{ has_fps_expansion } F * \text{fps_const } c$
proof –
have $F * \text{fps_const } c = \text{fps_const } c * F$
by (*intro fps_ext*) (*auto simp: mult.commute*)
with *has_fps_expansion_cmult_left* [*OF assms*] **show** *?thesis*
by (*simp add: mult.commute*)
qed

lemma *has_fps_expansion_minus* [*fps_expansion_intros*]:
assumes $f \text{ has_fps_expansion } F$
shows $(\lambda x. - f x) \text{ has_fps_expansion } -F$
proof –
from *assms* **have** *eventually* $(\lambda x. x \in \text{eball } 0 (\text{fps_conv_radius } F)) (\text{nhds } 0)$
by (*intro eventually_nhds_in_open*) (*auto simp: has_fps_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* $(\lambda x. \text{eval_fps } F x = f x) (\text{nhds } 0)$
by (*auto simp: has_fps_expansion_def*)
ultimately have *eventually* $(\lambda x. \text{eval_fps } (-F) x = -f x) (\text{nhds } 0)$
by *eventually_elim* (*auto simp: eval_fps_minus*)
thus *?thesis* **using** *assms* **by** (*auto simp: has_fps_expansion_def*)
qed

lemma *has_fps_expansion_add* [*fps_expansion_intros*]:

assumes *f* *has_fps_expansion* *F* *g* *has_fps_expansion* *G*

shows $(\lambda x. f\ x + g\ x)$ *has_fps_expansion* $F + G$

proof –

from *assms* **have** $0 < \min(\text{fps_conv_radius } F) (\text{fps_conv_radius } G)$

by (*auto simp: has_fps_expansion_def*)

also **have** $\dots \leq \text{fps_conv_radius } (F + G)$

by (*rule fps_conv_radius_add*)

finally **have** *radius: ... > 0* .

from *assms* **have** *eventually* $(\lambda x. x \in \text{eball } 0 (\text{fps_conv_radius } F)) (\text{nhds } 0)$

eventually $(\lambda x. x \in \text{eball } 0 (\text{fps_conv_radius } G)) (\text{nhds } 0)$

by (*intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def*)+

moreover **have** *eventually* $(\lambda x. \text{eval_fps } F\ x = f\ x) (\text{nhds } 0)$

and *eventually* $(\lambda x. \text{eval_fps } G\ x = g\ x) (\text{nhds } 0)$

using *assms* **by** (*auto simp: has_fps_expansion_def*)

ultimately **have** *eventually* $(\lambda x. \text{eval_fps } (F + G)\ x = f\ x + g\ x) (\text{nhds } 0)$

by *eventually_elim* (*auto simp: eval_fps_add*)

with *radius* **show** *?thesis* **by** (*auto simp: has_fps_expansion_def*)

qed

lemma *has_fps_expansion_diff* [*fps_expansion_intros*]:

assumes *f* *has_fps_expansion* *F* *g* *has_fps_expansion* *G*

shows $(\lambda x. f\ x - g\ x)$ *has_fps_expansion* $F - G$

using *has_fps_expansion_add*[*of f F λx. - g x -G*] *assms*

by (*simp add: has_fps_expansion_minus*)

lemma *has_fps_expansion_mult* [*fps_expansion_intros*]:

fixes *F G* :: 'a :: {*banach, real_normed_div_algebra, comm_ring_1*} *fps*

assumes *f* *has_fps_expansion* *F* *g* *has_fps_expansion* *G*

shows $(\lambda x. f\ x * g\ x)$ *has_fps_expansion* $F * G$

proof –

from *assms* **have** $0 < \min(\text{fps_conv_radius } F) (\text{fps_conv_radius } G)$

by (*auto simp: has_fps_expansion_def*)

also **have** $\dots \leq \text{fps_conv_radius } (F * G)$

by (*rule fps_conv_radius_mult*)

finally **have** *radius: ... > 0* .

from *assms* **have** *eventually* $(\lambda x. x \in \text{eball } 0 (\text{fps_conv_radius } F)) (\text{nhds } 0)$

eventually $(\lambda x. x \in \text{eball } 0 (\text{fps_conv_radius } G)) (\text{nhds } 0)$

by (*intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def*)+

moreover **have** *eventually* $(\lambda x. \text{eval_fps } F\ x = f\ x) (\text{nhds } 0)$

and *eventually* $(\lambda x. \text{eval_fps } G\ x = g\ x) (\text{nhds } 0)$

using *assms* **by** (*auto simp: has_fps_expansion_def*)

ultimately **have** *eventually* $(\lambda x. \text{eval_fps } (F * G)\ x = f\ x * g\ x) (\text{nhds } 0)$

by *eventually_elim* (*auto simp: eval_fps_mult*)

with *radius* **show** *?thesis* **by** (*auto simp: has_fps_expansion_def*)

qed

```

lemma has_fps_expansion_inverse [fps_expansion_intros]:
  fixes F :: 'a :: {banach, real_normed_field} fps
  assumes f_has_fps_expansion F
  assumes fps_nth F 0  $\neq$  0
  shows  $(\lambda x. \text{inverse } (f x))$  has_fps_expansion inverse F
proof -
  have radius: fps_conv_radius (inverse F) > 0
    using assms unfolding has_fps_expansion_def
    by (intro fps_conv_radius_inverse_pos) auto
  let ?R = min (fps_conv_radius F) (fps_conv_radius (inverse F))
  from assms radius
    have eventually  $(\lambda x. x \in \text{eball } 0 \text{ (fps_conv_radius F)})$  (nhds 0)
      eventually  $(\lambda x. x \in \text{eball } 0 \text{ (fps_conv_radius (inverse F))})$  (nhds 0)
    by (intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def)+
  moreover have eventually  $(\lambda z. \text{eval\_fps } F z = f z)$  (nhds 0)
    using assms by (auto simp: has_fps_expansion_def)
  ultimately have eventually  $(\lambda z. \text{eval\_fps } (\text{inverse } F) z = \text{inverse } (f z))$  (nhds
0)
proof eventually_elim
  case (elim z)
  hence eval_fps (inverse F * F) z = eval_fps (inverse F) z * f z
    by (subst eval_fps_mult) auto
  also have eval_fps (inverse F * F) z = 1
    using assms by (simp add: inverse_mult_eq_1)
  finally show ?case by (auto simp: field_split_simps)
qed
  with radius show ?thesis by (auto simp: has_fps_expansion_def)
qed

```

```

lemma has_fps_expansion_exp [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows  $(\lambda x. \text{exp } (c * x))$  has_fps_expansion fps_exp c
  by (auto simp: has_fps_expansion_def)

```

```

lemma has_fps_expansion_exp1 [fps_expansion_intros]:
   $(\lambda x::'a :: \{\text{banach, real\_normed\_field}\}. \text{exp } x)$  has_fps_expansion fps_exp 1
  using has_fps_expansion_exp[of 1] by simp

```

```

lemma has_fps_expansion_exp_neg1 [fps_expansion_intros]:
   $(\lambda x::'a :: \{\text{banach, real\_normed\_field}\}. \text{exp } (-x))$  has_fps_expansion fps_exp
(-1)
  using has_fps_expansion_exp[of -1] by simp

```

```

lemma has_fps_expansion_deriv [fps_expansion_intros]:
  assumes f_has_fps_expansion F
  shows deriv f has_fps_expansion fps_deriv F
proof -
  have eventually  $(\lambda z. z \in \text{eball } 0 \text{ (fps_conv_radius F)})$  (nhds 0)
    using assms by (intro eventually_nhds_in_open)

```

```

      (auto simp: has_fps_expansion_def zero_ereal_def)
moreover from assms have eventually ( $\lambda z. \text{eval\_fps } F z = f z$ ) (nhds 0)
  by (auto simp: has_fps_expansion_def)
then obtain s where open s  $0 \in s$  and s:  $\bigwedge w. w \in s \implies \text{eval\_fps } F w = f w$ 
  by (auto simp: eventually_nhds)
hence eventually ( $\lambda w. w \in s$ ) (nhds 0)
  by (intro eventually_nhds_in_open) auto
ultimately have eventually ( $\lambda z. \text{eval\_fps } (\text{fps\_deriv } F) z = \text{deriv } f z$ ) (nhds 0)
proof eventually_elim
  case (elim z)
  hence  $\text{eval\_fps } (\text{fps\_deriv } F) z = \text{deriv } (\text{eval\_fps } F) z$ 
    by (simp add: eval_fps_deriv)
  also have eventually ( $\lambda w. w \in s$ ) (nhds z)
    using elim and  $\langle \text{open } s \rangle$  by (intro eventually_nhds_in_open) auto
  hence eventually ( $\lambda w. \text{eval\_fps } F w = f w$ ) (nhds z)
    by eventually_elim (simp add: s)
  hence  $\text{deriv } (\text{eval\_fps } F) z = \text{deriv } f z$ 
    by (intro deriv_cong_ev refl)
  finally show ?case .
qed
with assms and fps_conv_radius_deriv[of F] show ?thesis
  by (auto simp: has_fps_expansion_def)
qed

```

```

lemma fps_conv_radius_binomial:
  fixes c :: 'a :: {real_normed_field,banach}
  shows  $\text{fps\_conv\_radius } (\text{fps\_binomial } c) = (\text{if } c \in \mathbb{N} \text{ then } \infty \text{ else } 1)$ 
  unfolding fps_conv_radius_def by (simp add: conv_radius_gchoose)

```

```

lemma fps_conv_radius_ln:
  fixes c :: 'a :: {banach, real_normed_field, field_char_0}
  shows  $\text{fps\_conv\_radius } (\text{fps\_ln } c) = (\text{if } c = 0 \text{ then } \infty \text{ else } 1)$ 
proof (cases c = 0)
  case False
  have  $\text{conv\_radius } (\lambda n. 1 / \text{of\_nat } n :: 'a) = 1$ 
  proof (rule conv_radius_ratio_limit_nonzero)
    show ( $\lambda n. \text{norm } (1 / \text{of\_nat } n :: 'a) / \text{norm } (1 / \text{of\_nat } (\text{Suc } n) :: 'a)$ )  $\longrightarrow$ 
    1
    using LIMSEQ_Suc_n_over_n by (simp add: norm_divide del: of_nat_Suc)
  qed auto
  also have  $\text{conv\_radius } (\lambda n. 1 / \text{of\_nat } n :: 'a) =$ 
     $\text{conv\_radius } (\lambda n. \text{if } n = 0 \text{ then } 0 \text{ else } (-1) ^ (n - 1) / \text{of\_nat } n ::$ 
    'a)
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of 0]])
      (simp add: norm_mult norm_divide norm_power)
  finally show ?thesis using False unfolding fps_ln_def
    by (subst fps_conv_radius_cmult_left) (simp_all add: fps_conv_radius_def)
qed (auto simp: fps_ln_def)

```



```

lemma fps_conv_radius_ln_nonzero [simp]:
  assumes  $c \neq (0 :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\})$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_ln } c) = 1$ 
  using assms by (simp add: fps_conv_radius_ln)

lemma fps_conv_radius_sin [simp]:
  fixes  $c :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\}$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_sin } c) = \infty$ 
proof (cases  $c = 0$ )
  case False
  have  $\infty = \text{conv\_radius} (\lambda n. \text{of\_real} (\text{sin\_coeff } n) :: 'a)$ 
  proof (rule sym, rule conv_radius_inftyI'', rule summable_norm_cancel, goal_cases)
    case (1 z)
    show ?case using summable_norm_sin[of z] by (simp add: norm_mult)
  qed
  also have  $\dots / \text{norm } c = \text{conv\_radius} (\lambda n. c ^ n * \text{of\_real} (\text{sin\_coeff } n) :: 'a)$ 
  using False by (subst conv_radius_mult_power) auto
  also have  $\dots = \text{fps\_conv\_radius} (\text{fps\_sin } c)$  unfolding fps_conv_radius_def
  by (rule conv_radius_cong_weak) (auto simp add: fps_sin_def sin_coeff_def)
  finally show ?thesis by simp
qed simp_all

lemma fps_conv_radius_cos [simp]:
  fixes  $c :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\}$ 
  shows  $\text{fps\_conv\_radius} (\text{fps\_cos } c) = \infty$ 
proof (cases  $c = 0$ )
  case False
  have  $\infty = \text{conv\_radius} (\lambda n. \text{of\_real} (\text{cos\_coeff } n) :: 'a)$ 
  proof (rule sym, rule conv_radius_inftyI'', rule summable_norm_cancel, goal_cases)
    case (1 z)
    show ?case using summable_norm_cos[of z] by (simp add: norm_mult)
  qed
  also have  $\dots / \text{norm } c = \text{conv\_radius} (\lambda n. c ^ n * \text{of\_real} (\text{cos\_coeff } n) :: 'a)$ 
  using False by (subst conv_radius_mult_power) auto
  also have  $\dots = \text{fps\_conv\_radius} (\text{fps\_cos } c)$  unfolding fps_conv_radius_def
  by (rule conv_radius_cong_weak) (auto simp add: fps_cos_def cos_coeff_def)
  finally show ?thesis by simp
qed simp_all

lemma eval_fps_sin [simp]:
  fixes  $z :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\}$ 
  shows  $\text{eval\_fps} (\text{fps\_sin } c) z = \text{sin} (c * z)$ 
proof -
  have  $(\lambda n. \text{sin\_coeff } n *_{\mathbb{R}} (c * z) ^ n)$  sums  $\text{sin} (c * z)$  by (rule sin_converges)
  also have  $(\lambda n. \text{sin\_coeff } n *_{\mathbb{R}} (c * z) ^ n) = (\lambda n. \text{fps\_nth} (\text{fps\_sin } c) n * z ^ n)$ 
  by (rule ext) (auto simp: sin_coeff_def fps_sin_def power_mult_distrib scaleR_conv_of_real)
  finally show ?thesis by (simp add: sums_iff eval_fps_def)
qed

```

```

lemma eval_fps_cos [simp]:
  fixes z :: 'a :: {banach, real_normed_field, field_char_0}
  shows eval_fps (fps_cos c) z = cos (c * z)
proof -
  have ( $\lambda n. \text{cos\_coeff } n *_{\mathbb{R}} (c * z)^{\wedge} n$ ) sums cos (c * z) by (rule cos_converges)
  also have ( $\lambda n. \text{cos\_coeff } n *_{\mathbb{R}} (c * z)^{\wedge} n$ ) = ( $\lambda n. \text{fps\_nth } (\text{fps\_cos } c) n * z^{\wedge} n$ )
  by (rule ext) (auto simp: cos_coeff_def fps_cos_def power_mult_distrib scaleR_conv_of_real)
  finally show ?thesis by (simp add: sums_iff eval_fps_def)
qed

```

```

lemma cos_eq_zero_imp_norm_ge:
  assumes cos (z :: complex) = 0
  shows norm z  $\geq$  pi / 2
proof -
  from assms obtain n where z = complex_of_real ((of_int n + 1 / 2) * pi)
  by (auto simp: cos_eq_0 algebra_simps)
  also have norm ... = |real_of_int n + 1 / 2| * pi
  by (subst norm_of_real) (simp_all add: abs_mult)
  also have real_of_int n + 1 / 2 = of_int (2 * n + 1) / 2 by simp
  also have |...| = of_int |2 * n + 1| / 2 by (subst abs_divide) simp_all
  also have ... * pi = of_int |2 * n + 1| * (pi / 2) by simp
  also have ...  $\geq$  of_int 1 * (pi / 2)
  by (intro mult_right_mono, subst of_int_le_iff) (auto simp: abs_if)
  finally show ?thesis by simp
qed

```

```

lemma eval_fps_binomial:
  fixes c :: complex
  assumes norm z < 1
  shows eval_fps (fps_binomial c) z = (1 + z) powr c
  using gen_binomial_complex[OF assms] by (simp add: sums_iff eval_fps_def)

```

```

lemma has_fps_expansion_binomial_complex [fps_expansion_intros]:
  fixes c :: complex
  shows ( $\lambda x. (1 + x) \text{ powr } c$ ) has_fps_expansion fps_binomial c
proof -
  have *: eventually ( $\lambda z::\text{complex}. z \in \text{eball } 0 \ 1$ ) (nhds 0)
  by (intro eventually_nhds_in_open) auto
  thus ?thesis
  by (auto simp: has_fps_expansion_def eval_fps_binomial fps_conv_radius_binomial
    intro!: eventually_mono [OF *])
qed

```

```

lemma has_fps_expansion_sin [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_field, field_char_0}

```

shows $(\lambda x. \sin (c * x))$ has_fps_expansion fps_sin c
by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_sin' [fps_expansion_intros]:
 $(\lambda x::'a :: \{\text{banach, real_normed_field}\}. \sin x)$ has_fps_expansion fps_sin 1
using has_fps_expansion_sin[of 1] **by** simp

lemma has_fps_expansion_cos [fps_expansion_intros]:
fixes $c :: 'a :: \{\text{banach, real_normed_field, field_char_0}\}$
shows $(\lambda x. \cos (c * x))$ has_fps_expansion fps_cos c
by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_cos' [fps_expansion_intros]:
 $(\lambda x::'a :: \{\text{banach, real_normed_field}\}. \cos x)$ has_fps_expansion fps_cos 1
using has_fps_expansion_cos[of 1] **by** simp

lemma has_fps_expansion_shift [fps_expansion_intros]:
fixes $F :: 'a :: \{\text{banach, real_normed_field}\}$ fps
assumes f has_fps_expansion F **and** $n \leq \text{subdegree } F$
assumes $c = \text{fps_nth } F \ n$
shows $(\lambda x. \text{if } x = 0 \text{ then } c \text{ else } f \ x / x \ ^n)$ has_fps_expansion (fps_shift n F)
proof –
have eventually $(\lambda x. x \in \text{eball } 0 \ (\text{fps_conv_radius } F))$ (nhds 0)
using assms **by** (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def zero_ereal_def)
moreover **have** eventually $(\lambda x. \text{eval_fps } F \ x = f \ x)$ (nhds 0)
using assms **by** (auto simp: has_fps_expansion_def)
ultimately **have** eventually $(\lambda x. \text{eval_fps } (\text{fps_shift } n \ F) \ x =$
 $(\text{if } x = 0 \text{ then } c \text{ else } f \ x / x \ ^n))$ (nhds 0)
by eventually_elim (auto simp: eval_fps_shift assms)
with assms **show** ?thesis **by** (auto simp: has_fps_expansion_def)
qed

lemma has_fps_expansion_divide [fps_expansion_intros]:
fixes $F \ G :: 'a :: \{\text{banach, real_normed_field}\}$ fps
assumes f has_fps_expansion F **and** g has_fps_expansion G **and**
 $\text{subdegree } G \leq \text{subdegree } F$ $G \neq 0$
 $c = \text{fps_nth } F \ (\text{subdegree } G) / \text{fps_nth } G \ (\text{subdegree } G)$
shows $(\lambda x. \text{if } x = 0 \text{ then } c \text{ else } f \ x / g \ x)$ has_fps_expansion (F / G)
proof –
define n **where** $n = \text{subdegree } G$
define F' **and** G' **where** $F' = \text{fps_shift } n \ F$ **and** $G' = \text{fps_shift } n \ G$
have $F = F' * \text{fps_X} \ ^n$ $G = G' * \text{fps_X} \ ^n$ **unfolding** F'_def G'_def n_def

by (rule fps_shift_times_fps_X_power [symmetric] le_refl | fact)+
moreover **from** assms **have** fps_nth G' 0 $\neq 0$
by (simp add: G'_def n_def)
ultimately **have** FG: $F / G = F' * \text{inverse } G'$
by (simp add: fps_divide_unit)

```

have ( $\lambda x.$  (if  $x = 0$  then  $\text{fps\_nth } F \ n$  else  $f \ x / x \wedge n$ ) *
        inverse (if  $x = 0$  then  $\text{fps\_nth } G \ n$  else  $g \ x / x \wedge n$ )) has_fps_expansion
F / G
  (is ?h has_fps_expansion _) unfolding FG F'_def G'_def n_def using  $\langle G \neq 0 \rangle$ 
  by (intro has_fps_expansion_mult has_fps_expansion_inverse
        has_fps_expansion_shift assms) auto
also have ?h = ( $\lambda x.$  if  $x = 0$  then  $c$  else  $f \ x / g \ x$ )
  using assms(5) unfolding n_def
  by (intro ext) (auto split: if_splits simp: field_simps)
finally show ?thesis .
qed

```

```

lemma has_fps_expansion_divide' [fps_expansion_intros]:
  fixes  $F \ G :: 'a :: \{\text{banach, real\_normed\_field}\}$  fps
  assumes  $f$  has_fps_expansion  $F$  and  $g$  has_fps_expansion  $G$  and  $\text{fps\_nth } G \ 0 \neq 0$ 
  shows ( $\lambda x.$   $f \ x / g \ x$ ) has_fps_expansion ( $F / G$ )
proof -
  have ( $\lambda x.$  if  $x = 0$  then  $\text{fps\_nth } F \ 0 / \text{fps\_nth } G \ 0$  else  $f \ x / g \ x$ ) has_fps_expansion
( $F / G$ )
  (is ?h has_fps_expansion _) using assms(3) by (intro has_fps_expansion_divide
assms) auto
  also from assms have  $\text{fps\_nth } F \ 0 = f \ 0 \ \text{fps\_nth } G \ 0 = g \ 0$ 
  by (auto simp: has_fps_expansion_def eval_fps_at_0 dest: eventually_nhds_x_imp_x)
  hence ?h = ( $\lambda x.$   $f \ x / g \ x$ ) by auto
finally show ?thesis .
qed

```

```

lemma has_fps_expansion_tan [fps_expansion_intros]:
  fixes  $c :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\}$ 
  shows ( $\lambda x.$   $\tan (c * x)$ ) has_fps_expansion  $\text{fps\_tan } c$ 
proof -
  have ( $\lambda x.$   $\sin (c * x) / \cos (c * x)$ ) has_fps_expansion  $\text{fps\_sin } c / \text{fps\_cos } c$ 
  by (intro fps_expansion_intros) auto
  thus ?thesis by (simp add: tan_def fps_tan_def)
qed

```

```

lemma has_fps_expansion_tan' [fps_expansion_intros]:
   $\tan$  has_fps_expansion  $\text{fps\_tan} (1 :: 'a :: \{\text{banach, real\_normed\_field, field\_char\_0}\})$ 
  using has_fps_expansion_tan[of 1] by simp

```

```

lemma has_fps_expansion_imp_holomorphic:
  assumes  $f$  has_fps_expansion  $F$ 
  obtains  $s$  where  $\text{open } s \ 0 \in s \ \wedge \ z. z \in s \implies f \ z = \text{eval\_fps } F \ z$ 
proof -
  from assms obtain  $s$  where  $s: \text{open } s \ 0 \in s \ \wedge \ z. z \in s \implies \text{eval\_fps } F \ z = f \ z$ 

```

```

  unfolding has_fps_expansion_def eventually_nhds by blast
  let ?s' = eball 0 (fps_conv_radius F) ∩ s
  have eval_fps F holomorphic_on ?s'
    by (intro holomorphic_intros) auto
  also have ?thesis  $\longleftrightarrow$  f holomorphic_on ?s'
    using s by (intro holomorphic_cong) auto
  finally show ?thesis using s assms
    by (intro that[of ?s']) (auto simp: has_fps_expansion_def zero_ereal_def)
qed

end

```

9.36 Smooth paths

```

theory Smooth_Paths
  imports Retracts
begin

```

9.36.1 Homeomorphisms of arc images

```

lemma path_connected_arc_complement:
  fixes  $\gamma :: real \Rightarrow 'a::euclidean\_space$ 
  assumes arc  $\gamma$   $2 \leq DIM('a)$ 
  shows path_connected( $\neg$  path_image  $\gamma$ )
proof -
  have path_image  $\gamma$  homeomorphic {0..1::real}
    by (simp add: assms homeomorphic_arc_image_interval)
  then show ?thesis
    by (intro path_connected_complement_homeomorphic_convex_compact) (auto
simp: assms)
qed

```

```

lemma connected_arc_complement:
  fixes  $\gamma :: real \Rightarrow 'a::euclidean\_space$ 
  assumes arc  $\gamma$   $2 \leq DIM('a)$ 
  shows connected( $\neg$  path_image  $\gamma$ )
  by (simp add: assms path_connected_arc_complement path_connected_imp_connected)

```

```

lemma inside_arc_empty:
  fixes  $\gamma :: real \Rightarrow 'a::euclidean\_space$ 
  assumes arc  $\gamma$ 
  shows inside(path_image  $\gamma$ ) = {}
proof (cases  $DIM('a) = 1$ )
  case True
  then show ?thesis
    using assms connected_arc_image connected_convex_1_gen inside_convex by
blast
next
  case False

```

3850

```

then have connected (– path_image  $\gamma$ )
  by (metis DIM_ge_Suc0 One_nat_def Suc_1 antisym assms connected_arc_complement
not_less_eq_eq)
  then
  show ?thesis
  by (simp add: assms bounded_arc_image inside_bounded_complement_connected_empty)
qed

```

```

lemma inside_simple_curve_imp_closed:
  fixes  $\gamma :: \text{real} \Rightarrow 'a::\text{euclidean\_space}$ 
  shows  $\llbracket \text{simple\_path } \gamma; x \in \text{inside}(\text{path\_image } \gamma) \rrbracket \Longrightarrow \text{pathfinish } \gamma = \text{pathstart } \gamma$ 
  using arc_simple_path inside_arc_empty by blast

```

9.36.2 Piecewise differentiability of paths

```

lemma continuous_on_joinpaths_D1:
  assumes continuous_on {0..1} (g1 +++ g2)
  shows continuous_on {0..1} g1
proof (rule continuous_on_eq)
  have continuous_on {0..1/2} (g1 +++ g2)
    using assms continuous_on_subset split_01 by auto
  then show continuous_on {0..1} (g1 +++ g2  $\circ$  (*) (inverse 2))
    by (intro continuous_intros) force
qed (auto simp: joinpaths_def)

```

```

lemma continuous_on_joinpaths_D2:
   $\llbracket \text{continuous\_on } \{0..1\} (g1 +++ g2); \text{pathfinish } g1 = \text{pathstart } g2 \rrbracket \Longrightarrow \text{continuous\_on } \{0..1\} g2$ 
  using path_def path_join by blast

```

```

lemma piecewise_differentiable_D1:
  assumes (g1 +++ g2) piecewise_differentiable_on {0..1}
  shows g1 piecewise_differentiable_on {0..1}
proof –
  obtain S where cont: continuous_on {0..1} g1 and finite S
    and S:  $\bigwedge x. x \in \{0..1\} - S \Longrightarrow g1 +++ g2$  differentiable at  $x$  within {0..1}
    using assms unfolding piecewise_differentiable_on_def
    by (blast dest!: continuous_on_joinpaths_D1)
  show ?thesis
    unfolding piecewise_differentiable_on_def
  proof (intro exI conjI ballI cont)
    show finite (insert 1 (((*)2) ‘ S))
      by (simp add:  $\langle$ finite  $S\rangle$ )
    show g1 differentiable at  $x$  within {0..1} if  $x \in \{0..1\} - \text{insert } 1 ((*) 2 \text{ ‘ } S)$ 
  for x
    proof (rule_tac d=dist (x/2) (1/2) in differentiable_transform_within)
      have g1 +++ g2 differentiable at (x / 2) within {0..1/2}
        by (rule differentiable_subset [OF S [of x/2]] | use that in force) +

```

```

    then show  $g1 \text{ +++ } g2 \circ (*)$  (inverse 2) differentiable at  $x$  within  $\{0..1\}$ 
      using image_affinity_atLeastAtMost_div [of 2 0 0::real 1]
      by (auto intro: differentiable_chain_within)
    qed (use that in <auto simp: joinpaths_def>)
  qed
qed

lemma piecewise_differentiable_D2:
  assumes  $(g1 \text{ +++ } g2)$  piecewise_differentiable_on  $\{0..1\}$  and eq: pathfinish  $g1$ 
  = pathstart  $g2$ 
  shows  $g2$  piecewise_differentiable_on  $\{0..1\}$ 
proof -
  have [simp]:  $g1 \ 1 = g2 \ 0$ 
    using eq by (simp add: pathfinish_def pathstart_def)
  obtain  $S$  where cont: continuous_on  $\{0..1\}$   $g2$  and finite  $S$ 
    and  $S: \bigwedge x. x \in \{0..1\} - S \implies g1 \text{ +++ } g2$  differentiable at  $x$  within  $\{0..1\}$ 
    using assms unfolding piecewise_differentiable_on_def
    by (blast dest!: continuous_on_joinpaths_D2)
  show ?thesis
    unfolding piecewise_differentiable_on_def
  proof (intro exI conjI ballI cont)
    show finite (insert 0 (( $\lambda x. 2*x-1$ )'S))
      by (simp add: <finite S>)
    show  $g2$  differentiable at  $x$  within  $\{0..1\}$  if  $x \in \{0..1\} - \text{insert } 0 \ ((\lambda x. 2*x-1)'S)$  for  $x$ 
      proof (rule_tac  $d = \text{dist } ((x+1)/2) \ (1/2)$  in differentiable_transform_within)
        have  $x2: (x + 1) / 2 \notin S$ 
          using that
          apply (clarsimp simp: image_iff)
          by (metis add.commute add_diff_cancel_left' mult_2 field_sum_of_halves)
        have  $g1 \text{ +++ } g2 \circ (\lambda x. (x+1) / 2)$  differentiable at  $x$  within  $\{0..1\}$ 
          by (rule differentiable_chain_within differentiable_subset [OF S [of  $(x+1)/2$ ]])
        | use  $x2$  that in force)+
        then show  $g1 \text{ +++ } g2 \circ (\lambda x. (x+1) / 2)$  differentiable at  $x$  within  $\{0..1\}$ 
          by (auto intro: differentiable_chain_within)
        show  $(g1 \text{ +++ } g2 \circ (\lambda x. (x + 1) / 2)) \ x' = g2 \ x'$  if  $x' \in \{0..1\}$  dist  $x' \ x < \text{dist } ((x + 1) / 2) \ (1/2)$  for  $x'$ 
          proof -
            have [simp]:  $(2*x'+2)/2 = x'+1$ 
              by (simp add: field_split_simps)
            show ?thesis
              using that by (auto simp: joinpaths_def)
          qed
      qed
    qed (use that in <auto simp: joinpaths_def>)
  qed
qed

lemma piecewise_C1_differentiable_D1:
  fixes  $g1 :: \text{real} \Rightarrow 'a::\text{real\_normed\_field}$ 

```

```

assumes (g1 +++ g2) piecewise_C1_differentiable_on {0..1}
  show g1 piecewise_C1_differentiable_on {0..1}
proof -
  obtain S where finite S
    and co12: continuous_on ({0..1} - S) (λx. vector_derivative (g1 +++
g2) (at x))
    and g12D: ∀ x ∈ {0..1} - S. g1 +++ g2 differentiable at x
  using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
  have g1D: g1 differentiable at x if x ∈ {0..1} - insert 1 ((* 2 ' S) for x
proof (rule differentiable_transform_within)
  show g1 +++ g2 ∘ (*) (inverse 2) differentiable at x
    using that g12D
  unfolding joinpaths_def
  by (intro differentiable_chain_at derivative_intros | force)+
  show ∧ x'. [[dist x' x < dist (x/2) (1/2)]]
    ⇒ (g1 +++ g2 ∘ (*) (inverse 2)) x' = g1 x'
    using that by (auto simp: dist_real_def joinpaths_def)
qed (use that in ⟨auto simp: dist_real_def⟩)
  have [simp]: vector_derivative (g1 ∘ (*) 2) (at (x/2)) = 2 *R vector_derivative
g1 (at x)
    if x ∈ {0..1} - insert 1 ((* 2 ' S) for x
  apply (subst vector_derivative_chain_at)
  using that
  apply (rule derivative_eq_intros g1D | simp)+
  done
  have continuous_on ({0..1/2} - insert (1/2) S) (λx. vector_derivative (g1
+++ g2) (at x))
    using co12 by (rule continuous_on_subset) force
  then have coDhalf: continuous_on ({0..1/2} - insert (1/2) S) (λx. vec-
tor_derivative (g1 ∘ (*) 2) (at x))
  proof (rule continuous_on_eq [OF vector_derivative_at])
    show (g1 +++ g2 has_vector_derivative vector_derivative (g1 ∘ (*) 2) (at
x)) (at x)
      if x ∈ {0..1/2} - insert (1/2) S for x
    proof (rule has_vector_derivative_transform_within)
      show (g1 ∘ (*) 2 has_vector_derivative vector_derivative (g1 ∘ (*) 2) (at
x)) (at x)
        using that
        by (force intro: g1D differentiable_chain_at simp: vector_derivative_works
[symmetric])
      show ∧ x'. [[dist x' x < dist x (1/2)]] ⇒ (g1 ∘ (*) 2) x' = (g1 +++ g2) x'
        using that by (auto simp: dist_norm joinpaths_def)
    qed (use that in ⟨auto simp: dist_norm⟩)
  qed
  have continuous_on ({0..1} - insert 1 ((* 2 ' S))
    ((λx. 1/2 * vector_derivative (g1 ∘ (*) 2) (at x)) ∘ (*) (1/2))
  using coDhalf
  apply (intro continuous_intros)
  by (simp add: scaleR_conv_of_real image_set_diff image_image)

```



```

then have con_g1: continuous_on ( $\{0..1\} - \text{insert } 1 ((*) 2 ' S)$ ) ( $\lambda x. \text{vector\_derivative } g1 \text{ (at } x)$ )
  by (rule continuous_on_eq) (simp add: scaleR_conv_of_real)
have continuous_on  $\{0..1\}$  g1
using continuous_on_joinpaths_D1 assms piecewise_C1_differentiable_on_def
by blast
with  $\langle \text{finite } S \rangle$  show ?thesis
apply (clarsimp simp add: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
apply (rule_tac x=insert 1 (((*)2)'S) in exI)
apply (simp add: g1D con_g1)
done
qed

```

lemma *piecewise_C1_differentiable_D2*:

```

fixes g2 :: real  $\Rightarrow$  'a::real_normed_field
assumes (g1 +++ g2) piecewise_C1_differentiable_on  $\{0..1\}$  pathfinish g1 =
pathstart g2
shows g2 piecewise_C1_differentiable_on  $\{0..1\}$ 
proof -
obtain S where finite S
and co12: continuous_on ( $\{0..1\} - S$ ) ( $\lambda x. \text{vector\_derivative } (g1 \text{ +++ } g2)$  (at x))
and g12D:  $\forall x \in \{0..1\} - S. g1 \text{ +++ } g2$  differentiable at x
using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
have g2D: g2 differentiable at x if  $x \in \{0..1\} - \text{insert } 0 ((\lambda x. 2*x-1) ' S)$  for
x
proof (rule differentiable_transform_within)
show g1 +++ g2  $\circ (\lambda x. (x + 1) / 2)$  differentiable at x
using g12D that
unfolding joinpaths_def
apply (drule_tac x=(x+1) / 2 in bspec, force simp: field_split_simps)
apply (rule differentiable_chain_at derivative_intros | force) +
done
show  $\bigwedge x'. \text{dist } x' x < \text{dist } ((x + 1) / 2) (1/2) \implies (g1 \text{ +++ } g2 \circ (\lambda x. (x + 1) / 2)) x' = g2 x'$ 
using that by (auto simp: dist_real_def joinpaths_def field_simps)
qed (use that in  $\langle \text{auto simp: dist_norm} \rangle$ )
have [simp]: vector_derivative (g2  $\circ (\lambda x. 2*x-1)$ ) (at  $((x+1)/2)$ ) =  $2 *_{\mathbb{R}}$ 
vector_derivative g2 (at x)
if  $x \in \{0..1\} - \text{insert } 0 ((\lambda x. 2*x-1) ' S)$  for x
using that by (auto simp: vector_derivative_chain_at field_split_simps g2D)
have continuous_on ( $\{1/2..1\} - \text{insert } (1/2) S$ ) ( $\lambda x. \text{vector\_derivative } (g1 \text{ +++ } g2)$  (at x))
using co12 by (rule continuous_on_subset) force
then have coDhalf: continuous_on ( $\{1/2..1\} - \text{insert } (1/2) S$ ) ( $\lambda x. \text{vector\_derivative } (g2 \circ (\lambda x. 2*x-1))$  (at x))
proof (rule continuous_on_eq [OF vector_derivative_at])
show (g1 +++ g2) has_vector_derivative vector_derivative (g2  $\circ (\lambda x. 2 * x - 1)$ ) (at x)

```

```

      (at x)
    if x ∈ {1 / 2..1} - insert (1 / 2) S for x
  proof (rule_tac f=g2 ∘ (λx. 2*x-1) and d=dist (3/4) ((x+1)/2) in has_vector_derivative_transfo
    show (g2 ∘ (λx. 2 * x - 1) has_vector_derivative vector_derivative (g2 ∘
(λx. 2 * x - 1)) (at x))
      (at x)
    using that by (force intro: g2D differentiable_chain_at simp: vector_derivative_works
[symmetric])
    show ∧x'. [|dist x' x < dist (3 / 4) ((x + 1) / 2)|] ⇒ (g2 ∘ (λx. 2 * x -
1)) x' = (g1 +++ g2) x'
    using that by (auto simp: dist_norm joinpaths_def add_divide_distrib)
  qed (use that in ⟨auto simp: dist_norm⟩)
  qed
  have [simp]: ((λx. (x+1) / 2) ‘ ({0..1} - insert 0 ((λx. 2 * x - 1) ‘ S))) =
({1/2..1} - insert (1/2) S)
  apply (simp add: image_set_diff inj_on_def image_image)
  apply (auto simp: image_affinity_atLeastAtMost_div add_divide_distrib)
  done
  have continuous_on ({0..1} - insert 0 ((λx. 2*x-1) ‘ S))
    ((λx. 1/2 * vector_derivative (g2 ∘ (λx. 2*x-1)) (at x)) ∘ (λx.
(x+1)/2))
  by (rule continuous_intros | simp add: coDhalf)+
  then have con_g2: continuous_on ({0..1} - insert 0 ((λx. 2*x-1) ‘ S)) (λx.
vector_derivative g2 (at x))
  by (rule continuous_on_eq) (simp add: scaleR_conv_of_real)
  have continuous_on {0..1} g2
  using continuous_on_joinpaths_D2 assms piecewise_C1_differentiable_on_def
  by blast
  with ⟨finite S⟩ show ?thesis
  by (meson C1_differentiable_on_eq con_g2 finite_imageI finite_insert g2D
piecewise_C1_differentiable_on_def)
  qed

```

9.36.3 Valid paths, and their start and finish

definition *valid_path* :: (real ⇒ 'a :: real_normed_vector) ⇒ bool
 where *valid_path* f ≡ f piecewise_C1_differentiable_on {0..1::real}

definition *closed_path* :: (real ⇒ 'a :: real_normed_vector) ⇒ bool
 where *closed_path* g ≡ g 0 = g 1

In particular, all results for paths apply

lemma *valid_path_imp_path*: *valid_path* g ⇒ *path* g
 by (simp add: *path_def* *piecewise_C1_differentiable_on_def* *valid_path_def*)

lemma *connected_valid_path_image*: *valid_path* g ⇒ *connected*(*path_image* g)
 by (*metis* *connected_path_image* *valid_path_imp_path*)

lemma *compact_valid_path_image*: *valid_path* g ⇒ *compact*(*path_image* g)

```

  by (metis compact_path_image valid_path_imp_path)

lemma bounded_valid_path_image: valid_path g  $\implies$  bounded(path_image g)
  by (metis bounded_path_image valid_path_imp_path)

lemma closed_valid_path_image: valid_path g  $\implies$  closed(path_image g)
  by (metis closed_path_image valid_path_imp_path)

lemma valid_path_translation_eq: valid_path ((+) d  $\circ$  p)  $\longleftrightarrow$  valid_path p
  by (simp add: valid_path_def piecewise_C1_differentiable_on_translation_eq)

lemma valid_path_compose:
  assumes valid_path g
    and der:  $\bigwedge x. x \in \text{path\_image } g \implies f \text{ field\_differentiable } (\text{at } x)$ 
    and con: continuous_on (path_image g) (deriv f)
  shows valid_path (f  $\circ$  g)
proof -
  obtain S where finite S and g_diff: g C1_differentiable_on {0..1} - S
  using <valid_path g> unfolding valid_path_def piecewise_C1_differentiable_on_def
  by auto
  have f  $\circ$  g differentiable at t when t  $\in$  {0..1} - S for t
  proof (rule differentiable_chain_at)
    show g differentiable at t using <valid_path g>
    by (meson C1_differentiable_on_eq <g C1_differentiable_on {0..1} - S>
  that)
  next
  have g t  $\in$  path_image g using that DiffD1 image_eqI path_image_def by
  metis
  then show f differentiable at (g t)
    using der[THEN field_differentiable_imp_differentiable] by auto
  qed
  moreover have continuous_on ({0..1} - S) ( $\lambda x. \text{vector\_derivative } (f \circ g) (\text{at } x)$ )
  proof (rule continuous_on_eq [where f =  $\lambda x. \text{vector\_derivative } g (\text{at } x) * \text{deriv } f (g x)$ ],
    rule continuous_intros)
    show continuous_on ({0..1} - S) ( $\lambda x. \text{vector\_derivative } g (\text{at } x)$ )
    using g_diff C1_differentiable_on_eq by auto
  next
  have continuous_on {0..1} ( $\lambda x. \text{deriv } f (g x)$ )
  using continuous_on_compose[OF con[unfolded path_image_def], unfolded
  comp_def]
    <valid_path g> piecewise_C1_differentiable_on_def valid_path_def
  by blast
  then show continuous_on ({0..1} - S) ( $\lambda x. \text{deriv } f (g x)$ )
  using continuous_on_subset by blast
  next
  show vector_derivative g (at t) * deriv f (g t) = vector_derivative (f  $\circ$  g)
  (at t)

```

```

      when  $t \in \{0..1\} - S$  for  $t$ 
    by (metis C1_differentiable_on_eq DiffD1 der g_diff imageI path_image_def
that
      vector_derivative_chain_at_general)
  qed
  ultimately have  $f \circ g$  C1_differentiable_on  $\{0..1\} - S$ 
  using C1_differentiable_on_eq by blast
  moreover have path  $(f \circ g)$ 
  using der
  by (simp add: path_continuous_image[OF valid_path_imp_path[OF  $\langle$ valid_path
 $g$ ]] continuous_at_imp_continuous_on field_differentiable_imp_continuous_at)
  ultimately show ?thesis unfolding valid_path_def piecewise_C1_differentiable_on_def
path_def
  using  $\langle$ finite  $S$  by auto
  qed

```

```

lemma valid_path_uminus_comp[simp]:
  fixes  $g::\text{real} \Rightarrow 'a :: \text{real\_normed\_field}$ 
  shows valid_path  $(\text{uminus} \circ g) \longleftrightarrow \text{valid\_path } g$ 
proof
  show valid_path  $g \implies \text{valid\_path } (\text{uminus} \circ g)$  for  $g::\text{real} \Rightarrow 'a$ 
  by (auto intro: valid_path_compose derivative_intros)
  then show valid_path  $g$  when valid_path  $(\text{uminus} \circ g)$ 
  by (metis fun.map_comp group_add_class.minus_comp_minus id_comp that)
  qed

```

```

lemma valid_path_offset[simp]:
  shows valid_path  $(\lambda t. g t - z) \longleftrightarrow \text{valid\_path } g$ 
proof
  show *: valid_path  $(g::\text{real} \Rightarrow 'a) \implies \text{valid\_path } (\lambda t. g t - z)$  for  $g z$ 
  unfolding valid_path_def
  by (fastforce intro: derivative_intros C1_differentiable_imp_piecewise piecewise_C1_differentiable_diff)
  show valid_path  $(\lambda t. g t - z) \implies \text{valid\_path } g$ 
  using *[of  $\lambda t. g t - z - z$ ,simplified] .
  qed

```

```

lemma valid_path_imp_reverse:
  assumes valid_path  $g$ 
  shows valid_path(reversepath  $g$ )
proof -
  obtain  $S$  where finite  $S$  and  $S: g$  C1_differentiable_on  $(\{0..1\} - S)$ 
  using assms by (auto simp: valid_path_def piecewise_C1_differentiable_on_def)
  then have finite  $((-) 1 ' S)$ 
  by auto
  moreover have (reversepath  $g$  C1_differentiable_on  $(\{0..1\} - (-) 1 ' S))$ 
  unfolding reversepath_def
  apply (rule C1_differentiable_compose [of  $\lambda x::\text{real}. 1-x$   $g$ , unfolded o_def])
  using  $S$ 

```

```

  by (force simp: finite_vimageI inj_on_def C1_differentiable_on_eq elim!: continuous_on_subset)+
  ultimately show ?thesis using assms
  by (auto simp: valid_path_def piecewise_C1_differentiable_on_def path_def [symmetric])
qed

```

```

lemma valid_path_reversepath [simp]: valid_path(reversepath g)  $\longleftrightarrow$  valid_path g
  using valid_path_imp_reverse by force

```

```

lemma valid_path_join:
  assumes valid_path g1 valid_path g2 pathfinish g1 = pathstart g2
  shows valid_path(g1 +++ g2)
proof -
  have g1 1 = g2 0
    using assms by (auto simp: pathfinish_def pathstart_def)
  moreover have (g1  $\circ$  ( $\lambda x. 2*x$ )) piecewise_C1_differentiable_on {0..1/2}
    apply (rule piecewise_C1_differentiable_compose)
    using assms
  apply (auto simp: valid_path_def piecewise_C1_differentiable_on_def continuous_on_joinpaths)
  apply (force intro: finite_vimageI [where h = (*)2] inj_onI)
  done
  moreover have (g2  $\circ$  ( $\lambda x. 2*x-1$ )) piecewise_C1_differentiable_on {1/2..1}
    apply (rule piecewise_C1_differentiable_compose)
    using assms unfolding valid_path_def piecewise_C1_differentiable_on_def
  by (auto intro!: continuous_intros finite_vimageI [where h = ( $\lambda x. 2*x - 1$ )] inj_onI
      simp: image_affinity_atLeastAtMost_diff continuous_on_joinpaths)
  ultimately show ?thesis
  unfolding valid_path_def continuous_on_joinpaths joinpaths_def
  by (intro piecewise_C1_differentiable_cases) (auto simp: o_def)
qed

```

```

lemma valid_path_join_D1:
  fixes g1 :: real  $\Rightarrow$  'a::real_normed_field
  shows valid_path (g1 +++ g2)  $\implies$  valid_path g1
  unfolding valid_path_def
  by (rule piecewise_C1_differentiable_D1)

```

```

lemma valid_path_join_D2:
  fixes g2 :: real  $\Rightarrow$  'a::real_normed_field
  shows  $\llbracket$ valid_path (g1 +++ g2); pathfinish g1 = pathstart g2 $\rrbracket \implies$  valid_path g2
  unfolding valid_path_def
  by (rule piecewise_C1_differentiable_D2)

```

```

lemma valid_path_join_eq [simp]:

```

```

fixes g2 :: real  $\Rightarrow$  'a::real_normed_field
shows pathfinish g1 = pathstart g2  $\implies$  (valid_path(g1 +++ g2)  $\longleftrightarrow$  valid_path
g1  $\wedge$  valid_path g2)
using valid_path_join_D1 valid_path_join_D2 valid_path_join by blast

```

```

lemma valid_path_shiftpath [intro]:
assumes valid_path g pathfinish g = pathstart g a  $\in$  {0..1}
shows valid_path(shiftpath a g)
using assms
unfolding valid_path_def shiftpath_alt_def
apply (intro piecewise_C1_differentiable_cases)
apply (simp_all add: add.commute)
apply (rule piecewise_C1_differentiable_affine [of g 1 a, simplified o_def
scaleR_one])
apply (force simp: pathfinish_def pathstart_def elim: piecewise_C1_differentiable_on_subset)
apply (rule piecewise_C1_differentiable_affine [of g 1 a-1, simplified o_def
scaleR_one algebra_simps])
apply (auto simp: pathfinish_def pathstart_def elim: piecewise_C1_differentiable_on_subset)
done

```

```

lemma vector_derivative_linepath_within:
x  $\in$  {0..1}  $\implies$  vector_derivative (linepath a b) (at x within {0..1}) = b - a
by (simp add: has_vector_derivative_linepath_within vector_derivative_at_within_ivl)

```

```

lemma vector_derivative_linepath_at [simp]: vector_derivative (linepath a b) (at
x) = b - a
by (simp add: has_vector_derivative_linepath_within vector_derivative_at)

```

```

lemma valid_path_linepath [iff]: valid_path (linepath a b)
using C1_differentiable_on_eq piecewise_C1_differentiable_on_def valid_path_def
by fastforce

```

```

lemma valid_path_subpath:
fixes g :: real  $\Rightarrow$  'a :: real_normed_vector
assumes valid_path g u  $\in$  {0..1} v  $\in$  {0..1}
shows valid_path(subpath u v g)
proof (cases v=u)
case True
then show ?thesis
unfolding valid_path_def subpath_def
by (force intro: C1_differentiable_on_const C1_differentiable_imp_piecewise)
next
case False
let ?f =  $\lambda x. ((v-u) * x + u)$ 
have (g  $\circ$  ?f) piecewise_C1_differentiable_on {0..1}
proof (rule piecewise_C1_differentiable_compose)
show ?f piecewise_C1_differentiable_on {0..1}
by (simp add: C1_differentiable_imp_piecewise)
have g piecewise_C1_differentiable_on (if u  $\leq$  v then {u..v} else {v..u})

```

```

    using assms piecewise_C1_differentiable_on_subset valid_path_def by force
  then show  $g$  piecewise_C1_differentiable_on  $?f$  ‘  $\{0..1\}$ 
    by (simp add: image_affinity_atLeastAtMost split: if_split_asm)
  show  $\bigwedge x. \text{finite } (\{0..1\} \cap ?f^{-1} \{x\})$ 
    using False
    by (simp add: Int_commute [of  $\{0..1\}$ ] inj_on_def crossproduct_eq finite_vimage_IntI)
  qed
  then show ?thesis
    by (auto simp: o_def valid_path_def subpath_def)
  qed

```

```

lemma valid_path_rectpath [simp, intro]: valid_path (rectpath a b)
  by (simp add: Let_def rectpath_def)

```

```

lemma linear_image_valid_path:
  fixes  $p :: \text{real} \Rightarrow 'a :: \text{euclidean\_space}$ 
  assumes valid_path  $p$  linear  $f$ 
  shows valid_path  $(f \circ p)$ 
  unfolding valid_path_def piecewise_C1_differentiable_on_def
  proof (intro conjI)
    from assms have path  $p$ 
    by (simp add: valid_path_imp_path)
    thus continuous_on  $\{0..1\}$   $(f \circ p)$ 
      unfolding o_def path_def by (intro linear_continuous_on_compose[OF _
  assms(2)])
    from assms(1) obtain  $S$  where  $S: \text{finite } S \ p \ C1\_differentiable\_on \{0..1\} - S$ 
    by (auto simp: valid_path_def piecewise_C1_differentiable_on_def)
    from  $S(2)$  obtain  $p' :: \text{real} \Rightarrow 'a$ 
      where  $p': \bigwedge x. x \in \{0..1\} - S \implies (p \text{ has\_vector\_derivative } p' \ x) \ (at \ x)$ 
      continuous_on  $(\{0..1\} - S)$   $p'$ 
    by (fastforce simp: C1_differentiable_on_def)

    have  $(f \circ p \text{ has\_vector\_derivative } f \ (p' \ x)) \ (at \ x)$  if  $x \in \{0..1\} - S$  for  $x$ 
      by (rule vector_derivative_diff_chain_within [OF  $p'(1)$  [OF that]]
        linear_imp_has_derivative assms)+
    moreover have continuous_on  $(\{0..1\} - S)$   $(\lambda x. f \ (p' \ x))$ 
      by (rule linear_continuous_on_compose [OF  $p'(2)$  assms(2)])
    ultimately have  $f \circ p \ C1\_differentiable\_on \{0..1\} - S$ 
      unfolding C1_differentiable_on_def by (intro exI [of  $\lambda x. f \ (p' \ x)$ ] fast
    thus  $\exists S. \text{finite } S \wedge f \circ p \ C1\_differentiable\_on \{0..1\} - S$ 
      using  $\langle \text{finite } S \rangle$  by blast
  qed

```

```

lemma valid_path_times:
  fixes  $\gamma :: \text{real} \Rightarrow 'a :: \text{real\_normed\_field}$ 
  assumes  $c \neq 0$ 
  shows valid_path  $((*) \ c \circ \gamma) = \text{valid\_path } \gamma$ 
  proof

```

3860

```
assume valid_path ((* c ∘ γ)
then have valid_path ((* (1/c) ∘ ((* c ∘ γ))
  by (simp add: valid_path_compose)
then show valid_path γ
  unfolding comp_def using ⟨c≠0⟩ by auto
next
assume valid_path γ
then show valid_path ((* c ∘ γ)
  by (simp add: valid_path_compose)
qed
```

```
lemma path_compose_cnj_iff [simp]: path (cnj ∘ p) ⟷ path p
proof -
  have path (cnj ∘ p) if path p for p
    by (intro path_continuous_image continuous_intros that)
  from this[of p] and this[of cnj ∘ p] show ?thesis
    by (auto simp: o_def)
qed
```

```
lemma valid_path_cnj:
  fixes g::real ⇒ complex
  shows valid_path (cnj ∘ g) = valid_path g
proof
  show valid:valid_path (cnj ∘ g) if valid_path g for g
  proof -
    obtain S where finite S and g_diff: g C1_differentiable_on {0..1} - S
    using ⟨valid_path g⟩ unfolding valid_path_def piecewise_C1_differentiable_on_def
  by auto
```

```
  have g_diff':g differentiable at t when t∈{0..1} - S for t
    by (meson C1_differentiable_on_eq ⟨g C1_differentiable_on {0..1} - S⟩
that)
  then have (cnj ∘ g) differentiable at t when t∈{0..1} - S for t
    using bounded_linear_cnj bounded_linear_imp_differentiable_differentiable_chain_at
that by blast
  moreover have continuous_on ({0..1} - S)
    (λx. vector_derivative (cnj ∘ g) (at x))
  proof -
    have continuous_on ({0..1} - S)
      (λx. vector_derivative (cnj ∘ g) (at x))
    = continuous_on ({0..1} - S)
      (λx. cnj (vector_derivative g (at x)))
    apply (rule continuous_on_cong[OF refl])
    unfolding comp_def using g_diff'
  using has_vector_derivative_cnj vector_derivative_at vector_derivative_works
by blast
  also have ...
  apply (intro continuous_intros)
  using C1_differentiable_on_eq g_diff by blast
```



```

    finally show ?thesis .
  qed
  ultimately have  $cnj \circ g \text{ C1\_differentiable\_on } \{0..1\} - S$ 
    using  $\text{C1\_differentiable\_on\_eq}$  by blast
  moreover have  $\text{path } (cnj \circ g)$ 
  apply (rule  $\text{path\_continuous\_image}[OF \text{valid\_path\_imp\_path}[OF \langle \text{valid\_path } g \rangle]]$ )
    by (intro  $\text{continuous\_intros}$ )
  ultimately show ?thesis unfolding  $\text{valid\_path\_def}$   $\text{piecewise\_C1\_differentiable\_on\_def}$ 
 $\text{path\_def}$ 
    using  $\langle \text{finite } S \rangle$  by auto
  qed
  from  $\text{this}[of \text{cnj } \circ g]$ 
  show  $\text{valid\_path } (cnj \circ g) \implies \text{valid\_path } g$ 
    unfolding  $\text{comp\_def}$  by simp
  qed
end

```

9.37 Metrics on product spaces

```

theory Function_Metric
  imports
    Function_Topology
    Elementary_Metric_Spaces
begin

```

In general, the product topology is not metrizable, unless the index set is countable. When the index set is countable, essentially any (convergent) combination of the metrics on the factors will do. We use below the simplest one, based on L^1 , but L^2 would also work, for instance.

What is not completely trivial is that the distance thus defined induces the same topology as the product topology. This is what we have to prove to show that we have an instance of *metric_space*.

The proofs below would work verbatim for general countable products of metric spaces. However, since distances are only implemented in terms of type classes, we only develop the theory for countable products of the same space.

```

instantiation  $\text{fun} :: (\text{countable}, \text{metric\_space}) \text{metric\_space}$ 
begin

```

```

definition  $\text{dist\_fun\_def}$ :

```

$$\text{dist } x \ y = \left(\sum n. (1/2)^{\wedge n} * \min (\text{dist } (x \text{ (from_nat } n)) \ (y \text{ (from_nat } n))) \ 1 \right)$$

```

definition  $\text{uniformity\_fun\_def}$ :

```

$$(\text{uniformity}::((\text{'a} \Rightarrow \text{'b}) \times (\text{'a} \Rightarrow \text{'b})) \text{filter}) = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } (x::(\text{'a} \Rightarrow \text{'b})) \ y < e\})$$

Except for the first one, the auxiliary lemmas below are only useful when proving the instance: once it is proved, they become trivial consequences of the general theory of metric spaces. It would thus be desirable to hide them once the instance is proved, but I do not know how to do this.

lemma *dist_fun_le_dist_first_terms*:

$dist\ x\ y \leq 2 * Max\ \{dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n))\ | n.\ n \leq N\} + (1/2)^N$

proof –

have $(\sum n.\ (1 / 2) ^ (n+Suc\ N) * min\ (dist\ (x\ (from_nat\ (n+Suc\ N)))\ (y\ (from_nat\ (n+Suc\ N))))\ 1)$

$= (\sum n.\ (1 / 2) ^ (Suc\ N) * ((1/2) ^ n * min\ (dist\ (x\ (from_nat\ (n+Suc\ N)))\ (y\ (from_nat\ (n+Suc\ N))))\ 1)$

by (*rule suminf_cong, simp add: power_add*)

also have $... = (1/2)^{(Suc\ N)} * (\sum n.\ (1 / 2) ^ n * min\ (dist\ (x\ (from_nat\ (n+Suc\ N)))\ (y\ (from_nat\ (n+Suc\ N))))\ 1)$

apply (*rule suminf_mult*)

by (*rule summable_comparison_test'[of $\lambda n.\ (1/2)^n$], auto simp add: summable_geometric_iff*)

also have $... \leq (1/2)^{(Suc\ N)} * (\sum n.\ (1 / 2) ^ n)$

apply (*simp, rule suminf_le, simp*)

by (*rule summable_comparison_test'[of $\lambda n.\ (1/2)^n$], auto simp add: summable_geometric_iff*)

also have $... = (1/2)^{(Suc\ N)} * 2$

using *suminf_geometric[of 1/2]* **by** *auto*

also have $... = (1/2)^N$

by *auto*

finally have $*$: $(\sum n.\ (1 / 2) ^ (n+Suc\ N) * min\ (dist\ (x\ (from_nat\ (n+Suc\ N)))\ (y\ (from_nat\ (n+Suc\ N))))\ 1 \leq (1/2)^N$

by *simp*

define *M* **where** $M = Max\ \{dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n))\ | n.\ n \leq N\}$

have $dist\ (x\ (from_nat\ 0))\ (y\ (from_nat\ 0)) \leq M$

unfolding *M_def* **by** (*rule Max_ge, auto*)

then have [*simp*]: $M \geq 0$ **by** (*meson dual_order.trans zero_le_dist*)

have $dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n)) \leq M$ **if** $n \leq N$ **for** n

unfolding *M_def* **apply** (*rule Max_ge*) **using** *that* **by** *auto*

then have i : $min\ (dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n)))\ 1 \leq M$ **if** $n \leq N$ **for** n

using *that* **by** *force*

have $(\sum n < Suc\ N.\ (1 / 2) ^ n * min\ (dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n)))\ 1) \leq$

$(\sum n < Suc\ N.\ M * (1 / 2) ^ n)$

by (*rule sum_mono, simp add: i*)

also have $... = M * (\sum n < Suc\ N.\ (1 / 2) ^ n)$

by (*rule sum_distrib_left[symmetric]*)

also have $... \leq M * (\sum n.\ (1 / 2) ^ n)$

by (*rule mult_left_mono, rule sum_le_suminf, auto simp add: summable_geometric_iff*)

also have $... = M * 2$

using *suminf_geometric[of 1/2]* **by** *auto*

finally have $**$: $(\sum n < Suc\ N.\ (1 / 2) ^ n * min\ (dist\ (x\ (from_nat\ n))\ (y\ (from_nat\ n)))\ 1) \leq 2 * M$

```

by simp

have dist x y = ( $\sum n. (1 / 2) ^ n * \min (\text{dist } (x (\text{from\_nat } n)) (y (\text{from\_nat } n))) 1$ )
  unfolding dist_fun_def by simp
also have ... = ( $\sum n. (1 / 2) ^ (n+\text{Suc } N) * \min (\text{dist } (x (\text{from\_nat } (n+\text{Suc } N))) (y (\text{from\_nat } (n+\text{Suc } N)))) 1$ )
  + ( $\sum n < \text{Suc } N. (1 / 2) ^ n * \min (\text{dist } (x (\text{from\_nat } n)) (y (\text{from\_nat } n))) 1$ )
  apply (rule suminf_split_initial_segment)
  by (rule summable_comparison_test'[of  $\lambda n. (1/2)^n$ ], auto simp add: summable_geometric_iff)
also have ...  $\leq 2 * M + (1/2)^N$ 
  using ** by auto
finally show ?thesis unfolding M_def by simp
qed

```

lemma open_fun_contains_ball_aux:

```

assumes open (U::('a  $\Rightarrow$  'b) set)
  x  $\in$  U
shows  $\exists e > 0. \{y. \text{dist } x y < e\} \subseteq U$ 
proof -
  have *: openin (product_topology ( $\lambda i. \text{euclidean}$ ) UNIV) U
    using open_fun_def assms by auto
  obtain X where H:  $\text{Pi}_E \text{ UNIV } X \subseteq U$ 
     $\wedge i. \text{openin euclidean } (X i)$ 
    finite { $i. X i \neq \text{topspace euclidean}$ }
    x  $\in \text{Pi}_E \text{ UNIV } X$ 
  using product_topology_open_contains_basis[OF *  $\langle x \in U \rangle$ ] by auto
  define I where I = { $i. X i \neq \text{topspace euclidean}$ }
  have finite I unfolding I_def using H(3) by auto
  {
    fix i
    have x i  $\in X i$  using  $\langle x \in U \rangle$  H by auto
    then have  $\exists e. e > 0 \wedge \text{ball } (x i) e \subseteq X i$ 
      using  $\langle \text{openin euclidean } (X i) \rangle$  open_openin_open_contains_ball by blast
    then obtain e where e  $> 0$  ball (x i) e  $\subseteq X i$  by blast
    define f where f = min e (1/2)
    have f  $> 0$  f  $< 1$  unfolding f_def using  $\langle e > 0 \rangle$  by auto
    moreover have ball (x i) f  $\subseteq X i$  unfolding f_def using  $\langle \text{ball } (x i) e \subseteq X i \rangle$  by auto
  }
  ultimately have  $\exists f. f > 0 \wedge f < 1 \wedge \text{ball } (x i) f \subseteq X i$  by auto
}
note * = this
have  $\exists e. \forall i. e i > 0 \wedge e i < 1 \wedge \text{ball } (x i) (e i) \subseteq X i$ 
  by (rule choice, auto simp add: *)
then obtain e where  $\wedge i. e i > 0 \wedge i. e i < 1 \wedge i. \text{ball } (x i) (e i) \subseteq X i$ 
  by blast
define m where m = Min { $(1/2) ^ (to\_nat i) * e i \mid i. i \in I$ }
have m  $> 0$  if I  $\neq \{\}$ 
  unfolding m_def Min_gr_iff using  $\langle \text{finite } I \rangle \langle I \neq \{\} \rangle \langle \wedge i. e i > 0 \rangle$  by auto

```

3864

```

moreover have {y. dist x y < m} ⊆ U
proof (auto)
  fix y assume dist x y < m
  have y i ∈ X i if i ∈ I for i
  proof -
    have *: summable (λn. (1/2)^n * min (dist (x (from_nat n)) (y (from_nat
n)))) 1)
      by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add:
summable_geometric_iff)
    define n where n = to_nat i
    have (1/2)^n * min (dist (x (from_nat n)) (y (from_nat n))) 1 < m
      using ⟨dist x y < m⟩ unfolding dist_fun_def using sum_le_suminf[OF
*, of {n}] by auto
    then have (1/2)^(to_nat i) * min (dist (x i) (y i)) 1 < m
      using ⟨n = to_nat i⟩ by auto
    also have ... ≤ (1/2)^(to_nat i) * e i
      unfolding m_def apply (rule Min_le) using ⟨finite I⟩ ⟨i ∈ I⟩ by auto
    ultimately have min (dist (x i) (y i)) 1 < e i
      by (auto simp add: field_split_simps)
    then have dist (x i) (y i) < e i
      using ⟨e i < 1⟩ by auto
    then show y i ∈ X i using ⟨ball (x i) (e i) ⊆ X i⟩ by auto
  qed
  then have y ∈ Pi_E UNIV X using H(1) unfolding I_def topspace_euclidean
by (auto simp add: Pi_E_iff)
  then show y ∈ U using ⟨Pi_E UNIV X ⊆ U⟩ by auto
  qed
  ultimately have *: ∃ m > 0. {y. dist x y < m} ⊆ U if I ≠ {} using that by
auto

  have U = UNIV if I = {}
    using that H(1) unfolding I_def topspace_euclidean by (auto simp add:
Pi_E_iff)
  then have ∃ m > 0. {y. dist x y < m} ⊆ U if I = {} using that zero_less_one
by blast
  then show ∃ m > 0. {y. dist x y < m} ⊆ U using * by blast
qed

lemma ball_fun_contains_open_aux:
  fixes x::('a ⇒ 'b) and e::real
  assumes e > 0
  shows ∃ U. open U ∧ x ∈ U ∧ U ⊆ {y. dist x y < e}
proof -
  have ∃ N::nat. 2^N > 8/e
    by (simp add: real_arch_pow)
  then obtain N::nat where 2^N > 8/e by auto
  define f where f = e/4
  have [simp]: e > 0 f > 0 unfolding f_def using assms by auto
  define X::('a ⇒ 'b set) where X = (λi. if (∃ n ≤ N. i = from_nat n) then {z.

```

```

dist (x i) z < f} else UNIV)
have {i. X i ≠ UNIV} ⊆ from_nat {0..N}
  unfolding X_def by auto
then have finite {i. X i ≠ topspace euclidean}
  unfolding topspace_euclidean using finite_surj by blast
have ∧i. open (X i)
  unfolding X_def using metric_space_class.open_ball open_UNIV by auto
then have ∧i. openin euclidean (X i)
  using open_openin by auto
define U where U = PiE UNIV X
have open U
  unfolding open_fun_def product_topology_def apply (rule topology_generated_by_Basis)
  unfolding U_def using ⟨∧i. openin euclidean (X i)⟩ ⟨finite {i. X i ≠ topspace
euclidean}⟩
  by auto
moreover have x ∈ U
  unfolding U_def X_def by (auto simp add: PiE_iff)
moreover have dist x y < e if y ∈ U for y
proof -
  have *: dist (x (from_nat n)) (y (from_nat n)) ≤ f if n ≤ N for n
    using ⟨y ∈ U⟩ unfolding U_def apply (auto simp add: PiE_iff)
    unfolding X_def using that by (metis less_imp_le mem_Collect_eq)
  have **: Max {dist (x (from_nat n)) (y (from_nat n)) | n. n ≤ N} ≤ f
    apply (rule Max.boundedI) using * by auto

  have dist x y ≤ 2 * Max {dist (x (from_nat n)) (y (from_nat n)) | n. n ≤ N}
+ (1/2)^N
  by (rule dist_fun_le_dist_first_terms)
  also have ... ≤ 2 * f + e / 8
  apply (rule add_mono) using ** ⟨2^N > 8/e⟩ by (auto simp add: field_split_simps)
  also have ... ≤ e/2 + e/8
  unfolding f_def by auto
  also have ... < e
  by auto
  finally show dist x y < e by simp
qed
ultimately show ?thesis by auto
qed

lemma fun_open_ball_aux:
  fixes U::('a ⇒ 'b) set
  shows open U ⟷ (∀ x ∈ U. ∃ e > 0. ∀ y. dist x y < e ⟶ y ∈ U)
proof (auto)
  assume open U
  fix x assume x ∈ U
  then show ∃ e > 0. ∀ y. dist x y < e ⟶ y ∈ U
    using open_fun_contains_ball_aux[OF ⟨open U⟩ ⟨x ∈ U⟩] by auto
next
  assume H: ∀ x ∈ U. ∃ e > 0. ∀ y. dist x y < e ⟶ y ∈ U

```

```

define K where K = {V. open V ∧ V ⊆ U}
{
  fix x assume x ∈ U
  then obtain e where e: e>0 {y. dist x y < e} ⊆ U using H by blast
  then obtain V where V: open V x ∈ V V ⊆ {y. dist x y < e}
    using ball_fun_contains_open_aux[OF ‹e>0›, of x] by auto
  have V ∈ K
    unfolding K_def using e(2) V(1) V(3) by auto
  then have x ∈ (⋃ K) using ‹x ∈ V› by auto
}
then have (⋃ K) = U
  unfolding K_def by auto
moreover have open (⋃ K)
  unfolding K_def by auto
ultimately show open U by simp
qed

instance proof
fix x y::'a ⇒ 'b show (dist x y = 0) = (x = y)
proof
  assume x = y
  then show dist x y = 0 unfolding dist_fun_def using ‹x = y› by auto
next
  assume dist x y = 0
  have *: summable (λn. (1/2)^n * min (dist (x (from_nat n)) (y (from_nat
n)))) 1)
  by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
  have (1/2)^n * min (dist (x (from_nat n)) (y (from_nat n))) 1 = 0 for n
    using ‹dist x y = 0› unfolding dist_fun_def by (simp add: * sum-
inf_eq_zero_iff)
  then have dist (x (from_nat n)) (y (from_nat n)) = 0 for n
    by (metis div_0 min_def nonzero_mult_div_cancel_left power_eq_0_iff
zero_eq_1_divide_iff zero_neq_numeral)
  then have x (from_nat n) = y (from_nat n) for n
    by auto
  then have x i = y i for i
    by (metis from_nat_to_nat)
  then show x = y
    by auto
qed
next

```

The proof of the triangular inequality is trivial, modulo the fact that we are dealing with infinite series, hence we should justify the convergence at each step. In this respect, the following simplification rule is extremely handy.

```

have [simp]: summable (λn. (1/2)^n * min (dist (u (from_nat n)) (v (from_nat
n)))) 1) for u v::'a ⇒ 'b
  by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
fix x y z::'a ⇒ 'b

```

```

{
  fix n
  have *:  $\text{dist } (x \text{ (from\_nat } n)) \text{ (y (from\_nat } n)) \leq$ 
           $\text{dist } (x \text{ (from\_nat } n)) \text{ (z (from\_nat } n)) + \text{dist } (y \text{ (from\_nat } n)) \text{ (z$ 
 $\text{(from\_nat } n))$ 
    by (simp add: dist_triangle2)
  have  $0 \leq \text{dist } (y \text{ (from\_nat } n)) \text{ (z (from\_nat } n))$ 
    using zero_le_dist by blast
  moreover
  {
    assume  $\text{min } (\text{dist } (y \text{ (from\_nat } n)) \text{ (z (from\_nat } n))) \text{ } 1 \neq \text{dist } (y \text{ (from\_nat$ 
 $n)) \text{ (z (from\_nat } n))$ 
    then have  $1 \leq \text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (z (from\_nat } n))) \text{ } 1 + \text{min } (\text{dist}$ 
 $(y \text{ (from\_nat } n)) \text{ (z (from\_nat } n))) \text{ } 1$ 
      by (metis (no_types) diff_le_eq diff_self min_def zero_le_dist zero_le_one)
    }
    ultimately have  $\text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (y (from\_nat } n))) \text{ } 1 \leq$ 
           $\text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (z (from\_nat } n))) \text{ } 1 + \text{min } (\text{dist } (y \text{ (from\_nat$ 
 $n)) \text{ (z (from\_nat } n))) \text{ } 1$ 
      using * by linarith
    } note ineq = this
  have  $\text{dist } x \text{ } y = (\sum n. (1/2)^{\wedge} n * \text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (y (from\_nat } n)))$ 
 $1)$ 
    unfolding dist_fun_def by simp
  also have  $\dots \leq (\sum n. (1/2)^{\wedge} n * \text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (z (from\_nat } n)))$ 
 $1$ 
           $+ (1/2)^{\wedge} n * \text{min } (\text{dist } (y \text{ (from\_nat } n)) \text{ (z (from\_nat } n))) \text{ } 1)$ 
    apply (rule suminf_le)
    using ineq apply (metis (no_types, opaque_lifting) add.right_neutral distrib_left
le_divide_eq_numeral1(1) mult_2_right mult_left_mono zero_le_one zero_le_power)
    by (auto simp add: summable_add)
  also have  $\dots = (\sum n. (1/2)^{\wedge} n * \text{min } (\text{dist } (x \text{ (from\_nat } n)) \text{ (z (from\_nat } n)))$ 
 $1)$ 
           $+ (\sum n. (1/2)^{\wedge} n * \text{min } (\text{dist } (y \text{ (from\_nat } n)) \text{ (z (from\_nat } n)))$ 
 $1)$ 
    by (rule suminf_add[symmetric], auto)
  also have  $\dots = \text{dist } x \text{ } z + \text{dist } y \text{ } z$ 
    unfolding dist_fun_def by simp
  finally show  $\text{dist } x \text{ } y \leq \text{dist } x \text{ } z + \text{dist } y \text{ } z$ 
    by simp
next

```

Finally, we show that the topology generated by the distance and the product topology coincide. This is essentially contained in Lemma *fun_open_ball_aux*, except that the condition to prove is expressed with filters. To deal with this, we copy some mumbo jumbo from Lemma *eventually_uniformity_metric* in `~/src/HOL/Real_Vector_Spaces.thy`

```
fix U::('a  $\Rightarrow$  'b) set
```

```

have eventually P uniformity  $\longleftrightarrow (\exists e>0. \forall x (y::('a \Rightarrow 'b)). \text{dist } x \ y < e \longrightarrow P (x, y))$  for P
unfolding uniformity_fun_def apply (subst eventually_INF_base)
by (auto simp: eventually_principal subset_eq intro: bexI[of _ min _ _])
then show open U =  $(\forall x \in U. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in U)$ 
unfolding fun_open_ball_aux by simp
qed (simp add: uniformity_fun_def)

```

end

Nice properties of spaces are preserved under countable products. In addition to first countability, second countability and metrizable, as we have seen above, completeness is also preserved, and therefore being Polish.

```

instance fun :: (countable, complete_space) complete_space

```

proof

```

fix u::nat  $\Rightarrow ('a \Rightarrow 'b)$  assume Cauchy u
have Cauchy  $(\lambda n. u \ n \ i)$  for i
unfolding Cauchy_def
proof (intro strip)
fix e::real assume e>0
obtain k where i = from_nat k
using from_nat_to_nat[of i] by metis
have  $(1/2)^k * \min e \ 1 > 0$  using <e>0> by auto
then have  $\exists N. \forall m \ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (u \ m) \ (u \ n) < (1/2)^k * \min e \ 1$ 
using <Cauchy u> by (meson Cauchy_def)
then obtain N::nat where C:  $\forall m \ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (u \ m) \ (u \ n) < (1/2)^k * \min e \ 1$ 
by blast
have  $\forall m \ n. N \leq m \wedge N \leq n \longrightarrow \text{dist } (u \ m \ i) \ (u \ n \ i) < e$ 
proof (auto)
fix m n::nat assume m  $\geq N$  n  $\geq N$ 
have  $(1/2)^k * \min (\text{dist } (u \ m \ i) \ (u \ n \ i)) \ 1$ 
= sum  $(\lambda p. (1/2)^p * \min (\text{dist } (u \ m \ (\text{from\_nat } p)) \ (u \ n \ (\text{from\_nat } p)))) \ 1 \ \{k\}$ 
using <i = from_nat k> by auto
also have  $\dots \leq (\sum p. (1/2)^p * \min (\text{dist } (u \ m \ (\text{from\_nat } p)) \ (u \ n \ (\text{from\_nat } p)))) \ 1$ 
apply (rule sum_le_suminf)
by (rule summable_comparison_test'[of  $\lambda n. (1/2)^n$ ], auto simp add: summable_geometric_iff)
also have  $\dots = \text{dist } (u \ m) \ (u \ n)$ 
unfolding dist_fun_def by auto
also have  $\dots < (1/2)^k * \min e \ 1$ 
using C <m  $\geq N$ > <n  $\geq N$ > by auto
finally have  $\min (\text{dist } (u \ m \ i) \ (u \ n \ i)) \ 1 < \min e \ 1$ 
by (auto simp add: field_split_simps)
then show  $\text{dist } (u \ m \ i) \ (u \ n \ i) < e$  by auto
qed

```



```

    then show  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (u \ m \ i) \ (u \ n \ i) < e$ 
      by blast
  qed
  then have  $\exists x. (\lambda n. u \ n \ i) \longrightarrow x$  for  $i$ 
    using Cauchy_convergent_convergent_def by auto
  then have  $\exists x. \forall i. (\lambda n. u \ n \ i) \longrightarrow x \ i$ 
    using choice by force
  then obtain  $x$  where  $*$ :  $\bigwedge i. (\lambda n. u \ n \ i) \longrightarrow x \ i$  by blast
  have  $u \longrightarrow x$ 
  proof (rule metric_LIMSEQ_I)
    fix  $e$  assume [simp]:  $e > (0::real)$ 
    have  $i: \exists K. \forall n \geq K. \text{dist } (u \ n \ i) \ (x \ i) < e/4$  for  $i$ 
      by (rule metric_LIMSEQ_D, auto simp add: *)
    have  $\exists K. \forall i. \forall n \geq K \ i. \text{dist } (u \ n \ i) \ (x \ i) < e/4$ 
      apply (rule choice) using  $i$  by auto
    then obtain  $K$  where  $K: \bigwedge i \ n. n \geq K \ i \implies \text{dist } (u \ n \ i) \ (x \ i) < e/4$ 
      by blast

  have  $\exists N::nat. 2^N > 4/e$ 
    by (simp add: real_arch_pow)
  then obtain  $N::nat$  where  $2^N > 4/e$  by auto
  define  $L$  where  $L = \text{Max } \{K \ (from\_nat \ n) \mid n. n \leq N\}$ 
  have  $\text{dist } (u \ k) \ x < e$  if  $k \geq L$  for  $k$ 
  proof -
    have  $*$ :  $\text{dist } (u \ k \ (from\_nat \ n)) \ (x \ (from\_nat \ n)) \leq e / 4$  if  $n \leq N$  for  $n$ 
  proof -
    have  $K \ (from\_nat \ n) \leq L$ 
      unfolding  $L\_def$  apply (rule Max_ge) using  $\langle n \leq N \rangle$  by auto
    then have  $k \geq K \ (from\_nat \ n)$  using  $\langle k \geq L \rangle$  by auto
    then show  $?thesis$  using  $K \ \text{less\_imp\_le}$  by auto
  qed
  have  $**$ :  $\text{Max } \{\text{dist } (u \ k \ (from\_nat \ n)) \ (x \ (from\_nat \ n)) \mid n. n \leq N\} \leq e/4$ 
    apply (rule Max.boundedI) using  $*$  by auto
  have  $\text{dist } (u \ k) \ x \leq 2 * \text{Max } \{\text{dist } (u \ k \ (from\_nat \ n)) \ (x \ (from\_nat \ n)) \mid n. n \leq N\} + (1/2)^N$ 
    using  $\text{dist\_fun\_le\_dist\_first\_terms}$  by auto
  also have  $\dots \leq 2 * e/4 + e/4$ 
    apply (rule add_mono)
    using  $** \ \langle 2^N > 4/e \rangle \ \text{less\_imp\_le}$  by (auto simp add: field_split_simps)
  also have  $\dots < e$  by auto
  finally show  $?thesis$  by simp
  qed
  then show  $\exists L. \forall k \geq L. \text{dist } (u \ k) \ x < e$  by blast
  qed
  then show convergent  $u$  using convergent_def by blast
  qed

instance fun :: (countable, polish_space) polish_space
  by standard

```

3870

end

theory *Analysis*

imports

Convex

Determinants

FSigma

Sum_Topology

Abstract_Topological_Spaces

Abstract_Metric_Spaces

Urysohn

Connected

Abstract_Limits

Isolated

Sparse_In

Elementary_Normed_Spaces

Norm_Arith

Convex_Euclidean_Space

Operator_Norm

Line_Segment

Derivative

Cartesian_Euclidean_Space

Kronecker_Approximation_Theorem

Weierstrass_Theorems

Ball_Volume

Integral_Test

Improper_Integral

Equivalence_Measurable_On_Borel

Lebesgue_Integral_Substitution

Embed_Measure

Complete_Measure

Radon_Nikodym

Fashoda_Theorem

Cross3

Homeomorphism

Bounded_Continuous_Function

Abstract_Topology

Product_Topology

Lindelof_Spaces

Infinite_Products

Infinite_Sum

Infinite_Set_Sum

Polytope

Jordan_Curve
Poly_Roots
Generalised_Binomial_Theorem
Gamma_Function
Change_Of_Vars
Multivariate_Analysis
Simplex_Content
FPS_Convergence
Smooth_Paths
Abstract_Euclidean_Space
Function_Metric

begin

end

Bibliography

- [1]
- [2] J. B. Conway. *A course in functional analysis*, volume 96. Springer Science & Business Media, 2013.
- [3] J. Dugundji. An extension of Tietze's theorem. *Pacific J. Math.*, 1(3):353–367, 1951.
- [4] R. Engelking. *General Topology*. Sigma series in pure mathematics. Heldermann, 1989.
- [5] S. Lang. *Real and Functional Analysis*. Springer, 1993.
- [6] M. Maggesi. A formalization of metric spaces in HOL light. *J. Autom. Reasoning*, 60(2):237–254, 2018.